SCORE DISTRIBUTION ANALYSIS, ARTIFICIAL INTELLIGENCE, AND PLAYER MODELING FOR QUANTITATIVE GAME DESIGN

DISSERTATION

Submitted in Partial Fulfillment of

the Requirements for

the Degree of

DOCTOR OF PHILOSOPHY (Computer Science)

at the

NEW YORK UNIVERSITY
TANDON SCHOOL OF ENGINEERING

by

Aaron Isaksen

May 2017
SCORE DISTRIBUTION ANALYSIS, ARTIFICIAL INTELLIGENCE, AND PLAYER MODELING FOR QUANTITATIVE GAME DESIGN

DISSERTATION

Submitted in Partial Fulfillment of
the Requirements for
the Degree of

DOCTOR OF PHILOSOPHY (Computer Science)

at the
NEW YORK UNIVERSITY
TANDON SCHOOL OF ENGINEERING

by
Aaron Isaksen

May 2017

Approved:

_________________________
Department Chair Signature

_________________________
Date

University ID: N18319753
Net ID: ai758
Approved by the Guidance Committee:

Major: Computer Science

__________________________
Andy Nealen
Assistant Professor of Computer Science
New York University, Tandon School of Engineering

__________________________
Julian Togelius
Associate Professor of Computer Science
New York University, Tandon School of Engineering

__________________________
Frank Lantz
Full Arts Professor and Director
New York University, NYU Game Center

__________________________
Michael Mateas
Professor of Computational Media and Director
University of California at Santa Cruz
Center for Games and Playable Media

__________________________
Leonard McMillan
Associate Professor of Computer Science
University of North Carolina at Chapel Hill
Microfilm/Publishing

Microfilm or copies of this dissertation may be obtained from:

UMI Dissertation Publishing
ProQuest CSA
789 E. Eisenhower Parkway
P.O. Box 1346
Ann Arbor, MI 48106-1346
Vita

Aaron Mark Isaksen

Education

Ph.D. in Computer Science Jan. 2014 - May 2017
New York University, Tandon School of Engineering, Game Innovation Lab
- Pearl Brownstein Doctoral Research Award, 2017
- Outstanding Performance on Ph.D. Qualifying Exam, D. Rosenthal, MD Award, 2015
- Best paper in Artificial Intelligence and Game Technology, FDG 2015

Massachusetts Institute of Technology, Laboratory for Computer Science
- NSF Graduate Research Fellowship

University of California at Berkeley
- Eta Kappa Nu (HKN) Honor Society

Research and Funding

The research in this thesis was performed at the NYU Game Innovation Lab from January 2014 through May 2017. Funding for tuition, research assistantships, and travel for the work and to present accepted conference papers was provided by Prof. Andy Nealen, Prof. Julian Togelius, the Game Innovation Lab, and via a CSE department teaching assistantship.

Professional History

Aaron Isaksen has worked in the game development, computer vision, and consumer electronics industry. In 2003, he founded AppAbove, Inc., maker of the first mobile phone Spanish-English phrasebook, and co-founded AppAbove Games, creator of more than 10 mobile games. Aaron’s game Chip Chain has been downloaded over 1 million times and received an Editor’s Choice award from Google. Aaron’s experience in artificial intelligence includes being lead researcher and architect of a patented computer vision based car detection system. He has also contributed his expertise as Founding Partner of indie game funding pioneer Indie Fund; Chairman of IndieBox; Advisor for the game crowd-equity company Fig and medical gaming startup Neuromotion; and NSF-SBIR Principle Investigator for Mental Canvas.
Acknowledgments

First and foremost, I would like to thank my PhD advisor Prof. Andy Nealen for his support, friendship, funding, and for making this process so fruitful and enjoyable. Secondly, I would like to thank Prof. Julian Togelius, who although is not formally my co-advisor, has provided essential funding, research ideas, and mentorship. This thesis would not be possible without my co-authors on the works contained within: Prof. Andy Nealen, Prof. Julian Togelius, Prof. Steven J Brams, Prof. Adam Finkelstein, Dr. Christoffer Holmgård, Mehmet Ismail, Drew Wallace, and Dan Gopstein. Thank you Qualification Committee members Prof. Katherine Isbister, Prof. Andy Nealen, and Prof. Julian Togelius; Dissertation Committee members Prof. Andy Nealen, Julian Togelius, Prof. Frank Lantz, Prof. Leonard McMillan, and Prof. Michael Mateas; the NYU Game Center; and all my fellow students at the NYU Game Innovation Lab.

I would also like to thank: Prof. Frank Lantz, Scott Lee, Ahmed Khalifa, Dr. Alex Jaffe, Fernando de Mesentier Silva, Rishabh Jain, Adam Summerville, Sam Snodgrass, Matthew Guzdial, and Prof. Amy K. Hoover for co-authoring other related AI+games papers; Prof. Amy K. Hoover, Prof. Rafael Bidarra, Prof. Nathan Sturtevant, Prof. Jichen Zhu, and Prof. Julian Togelius for co-organizing the PCG2016 and WNAIG2017 workshops; Prof. Lisa Hellerstein, Prof. Jose Ulerio, Prof. Cameron Browne, Prof. Mark J. Nelson, Prof. Mike Cook, Prof. Gillian Smith, Prof. Antonios Liapis, Prof. Adam M. Smith, Prof. Clara Fernandez, Prof. Eric Zimmerman, Prof. Bennett Foddy, Prof. Tamy Boubekeur, Prof. Daphna Harel, Dr. Alex Zook, Kaho Abe, Chris DiMauro, Chrystanyaa Brown, Dylan McKenzie, Jessica Lam, Gwynna Forgham-Thrift, Susana Garcia, and Eve Henderson for providing external guidance, support, and advice throughout the PhD process; Kyle C. Fox, Trevin Hiebert, Max McDonnell, and Dong Nguyen for help on the Flappy Bird project; Kevin Cancienne, Adam Saltsman, Prof. Frank Lantz, Zach Gage, Ron Carmel, and Steve Swink for providing critical support; and the many Anonymous reviewers for paper edits and suggestions.

From my past adventures in academia, thank you Prof. Carlo Sequin, Prof. Brian Barsky, and Prof. Dan Garcia from UC Berkeley for inspiring me to do graduate work; Prof. Leonard McMillan, Prof. Steven J. Gortler, Prof. Julie Dorsey, the late Prof. Seth Teller; Dr. Bob Sumner, Dr. Rob Jagnow, and Dr. Justin Legakis for their support and inspiration during my Master’s research at MIT; and to Simon Ferrari and Prof. Andy Nealen for reminding me that I wanted to complete a PhD in games.
Dedication

This thesis is dedicated to my wife, Christina, who unconditionally supported my decision to return to graduate school to finish the PhD that I first started almost 20 years ago. Thank you Stina for your love and support – I know that this was a long process but I could not have done it without your unwavering confidence in me. I also dedicate this work to Trixie, who always welcomed me home after a long day of research and reminded me to take a break, at least to feed her or take her for a walk. Finally, I dedicate this to Rose: we had dreamed that you would be here to see me achieve my doctorate, but sadly it was not meant to be. I love all three of you.

Additional Thanks

Thank you to my family: Joanne, Len, Debbie, Dan, Arlen, Rick, Jon, Jan, Kat, Coco, Ghongee, aunts, uncles, cousins, nieces and nephews for their love, encouragement, and support and to my late grandparents Micael, Freda, Bill, and Vera for teaching me the values of hard work and integrity. Thank you to John Bizzarro, my business partner for many years, who helped cultivate my game design skills during the years we made games together. Thanks to Bob Caspe for his critical advice and mentorship over the years, and to Rick Kraemer for giving me my first job and helping me develop my work ethic. Thank you to the wonderful people I’ve worked with in the indie game industry, especially James Morgan III from IndieBox, my partners Kellee Santiago, Ron Carmel, Jonathan Blow, Nathan Vella, Kyle Gabler, and Matthew Wegner at Indie Fund, and all the talented game designers we worked with that taught me how to make a great game. Thank you to Jeff, Liam, David, and Dan for all the video games we played together as kids, and to some of my childhood game designer heroes: Marc Blank and Dave Lebling (Zork I, Enchanter), Steve Meretzky (Planetfall), Bryan Eggers (13 Ghosts), Kotaro Hayashida (Alex Kidd in Miracle World), Brian Fargo (Wasteland), and Richard Garriott (Ultima V). Finally – Thank you Mom and Dad for getting us our first computer in 1982 and thereby starting me off on this path, and to my brother Dan for showing me what amazing things I might be able to do with it.
ABSTRACT

SCORE DISTRIBUTION ANALYSIS, ARTIFICIAL INTELLIGENCE, AND PLAYER MODELING FOR QUANTITATIVE GAME DESIGN

by

Aaron Isaksen

Advisor: Prof. Dr.-Ing. Andy Nealen

Submitted in partial fulfillment of the Requirements for the Degree of Doctor of Philosophy (Computer Science)

May 2017

We use score distribution data analysis, artificial intelligence, and player modeling to better understand games and game design using quantitative techniques, for studying the characteristics of games, and to explore the space of possible games. Score distributions model the probability a score will be achieved by a player; we model and visualize such probabilities using survival analysis, mean score, closeness, high score analysis, and other metrics. We apply artificial intelligence techniques to quantitative game design using tree search, genetic programming, optimization, procedural content generation, and Q-value modeling, focusing on general solutions. We analyze score data collected from human game play, in addition to simulated game play for scalable, repeatable, and controlled data experiments. We employ novel player modeling techniques to more accurately simulate human motor skill, timing accuracy, aiming dexterity, strategic thinking, inequity aversion, and learning effects. Quantitative analysis of simulated and real score data is used to explore various characteristics of games, including human-playable heuristics, game difficulty, score inequity, game balance, length of playtime, randomness, luck, skill, dexterity, and strategy required. We also explore game space to find new game variants using Monte Carlo simulation, computational creativity, sampling, mathematical modeling, and evolutionary search. This thesis contributes to the state of the art in several areas, including the first time that survival analysis has been applied to automated game design, the first quantitative calculation of difficulty curves, proving the probability of setting new high scores, and measuring the interaction of strategy and dexterity in games.
# Table of Contents

## 1 Introduction

1.1 Motivation and Overview ............................................. 1
1.2 Thesis Statement ...................................................... 6
   1.2.1 Analysis of Score Distributions ............................... 7
   1.2.2 Artificial Intelligence ........................................ 8
   1.2.3 Game Play Simulation ......................................... 9
   1.2.4 Player Modeling ................................................ 9
   1.2.5 Human Game Play Data ....................................... 11
   1.2.6 Characteristics of Games ..................................... 11
   1.2.7 Exploring Game Space ......................................... 13
1.3 Contributions to the State of the Art ................................ 14
1.4 Organizational Outline .............................................. 16
1.5 Relevant Publications ................................................ 17
1.6 Notes on Pronouns and Style ...................................... 19
1.7 Source Code .......................................................... 19

## 2 Description of Games

2.1 Flappy Bird .......................................................... 20
2.2 Box-Packing Game .................................................. 22
2.3 Puzzle Bobble ........................................................ 23
2.4 Catch-Up .............................................................. 24
2.5 Dice Battles .......................................................... 24
2.6 Canabalt ............................................................... 25
2.7 Drop7 ................................................................. 26

## 3 Modeling Score Distributions

3.1 Score Probabilities .................................................. 27
3.2 Probability Distribution Functions (PDF) \( f(x) \) ............... 28
3.3 Cumulative Distribution Function (CDF) $F(x)$ ........................................... 29
3.4 Survival Functions $S(x)$ ................................................................. 30
3.5 Hazard Functions $h(x)$ ................................................................. 31
3.6 Working with Discrete Data ............................................................ 33
3.7 Score Distributions Metrics ............................................................ 34
   3.7.1 Mean Score ................................................................. 34
   3.7.2 Puzzle Game Metrics ....................................................... 35
   3.7.3 Win Bias, Tie Percentage and Closeness ......................... 36
3.8 Conclusions .................................................................................. 38

4 Modeling and Simulating Timing Skill in Action Games .................. 39
   4.1 Introduction .................................................................................. 39
   4.2 Player Model for Simulating Timing ........................................... 40
      4.2.1 Player Precision ............................................................... 40
      4.2.2 Reaction Time ............................................................... 41
      4.2.3 Actions Per Second ........................................................ 42
   4.3 Simulating Action Games and Timing ........................................ 42
      4.3.1 Generate ................................................................. 42
      4.3.2 Simulate ................................................................. 43
      4.3.3 Number of Simulations .............................................. 44
      4.3.4 Anomalies in Flappy Bird ........................................... 45
      4.3.5 Determining Model Parameters .................................... 47
   4.4 Conclusions .................................................................................. 47

5 Modeling and Simulation of Strategy and Dexterity in Puzzle Games 49
   5.1 Introduction .................................................................................. 49
      5.1.1 Automated playtesting and Player Modeling ..................... 51
      5.1.2 Existing Methods for Modeling Difficulty ....................... 52
   5.2 Interactive Puzzle Games ........................................................ 54
      5.2.1 Examples of Strategy and Dexterity ............................... 54
   5.3 Mean Score for Comparing Puzzles ............................................ 56
   5.4 Modeling Strategy ...................................................................... 57
      5.4.1 Modeling Strategic Error ............................................... 57
      5.4.2 Player Heuristics .......................................................... 59
   5.5 Modeling Dexterity .................................................................... 60
      5.5.1 Simulating Dexterity ...................................................... 60
      5.5.2 Determining Accuracy Values ........................................ 61
## 5.5.3 Awareness of Accuracy

## 5.6 Algorithm

## 5.7 Applications of our Method

### 5.7.1 Exploring Puzzle Space

### 5.7.2 Determining Strategy and Dexterity Requirements

### 5.7.3 Evaluating the Effect of Scoring Systems

### 5.7.4 Setting Star/Medal Thresholds

### 5.7.5 Ordering Puzzles

## 5.8 Conclusions

## 6 Modeling Learning and High Scores

### 6.1 Introduction

### 6.2 A Short History of High Scores

### 6.3 Methods

### 6.4 How Scores Change Over Time

### 6.5 Probability of Beating the Previous Score

### 6.6 Probability of Reaching a High Score

### 6.7 Number of High Scores Achieved

### 6.8 Leaderboards

### 6.9 Probability of Achieving a Streak

### 6.10 Design Suggestion: Decaying High Scores

### 6.11 Design Suggestion: Beating the Mean

### 6.12 Conclusions

## 7 Survival Analysis of Simulated Games

### 7.1 Constant Hazards and the Exponential Distribution

#### 7.1.1 Analytical Model for Constant Hazard

#### 7.1.2 Simulation Data for Constant Hazard

### 7.2 Linear Hazards and the Rayleigh Distribution

#### 7.2.1 Analytical Model for Constant Hazard

#### 7.2.2 Simulation Data for Linear Hazard

### 7.3 Power-law Hazard and the Weibull Distribution

#### 7.3.1 Analytical Model for Power-law Hazard

#### 7.3.2 Simulation Data for Power-Law Hazard

### 7.4 Varying Skill Levels with Linear Hazards

### 7.5 Modeling Learning

#### 7.5.1 Evidence of Learning in Actual Game Distributions
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>7.5.2</td>
<td>Analytical Model for Modeling Learning</td>
<td>104</td>
</tr>
<tr>
<td>7.5.3</td>
<td>Simulation Data for Modeling Learning</td>
<td>104</td>
</tr>
<tr>
<td>7.6</td>
<td>Conclusions</td>
<td>107</td>
</tr>
<tr>
<td>8</td>
<td>Exploring Game Space and Computational Creativity</td>
<td>108</td>
</tr>
<tr>
<td>8.1</td>
<td>Introduction</td>
<td>108</td>
</tr>
<tr>
<td>8.2</td>
<td>Method</td>
<td>109</td>
</tr>
<tr>
<td>8.3</td>
<td>Exploring Game Space</td>
<td>110</td>
</tr>
<tr>
<td>8.3.1</td>
<td>Sampling and Visualization Methods</td>
<td>110</td>
</tr>
<tr>
<td>8.3.2</td>
<td>Exploration via Optimization</td>
<td>113</td>
</tr>
<tr>
<td>8.4</td>
<td>Validating Generated Content</td>
<td>114</td>
</tr>
<tr>
<td>8.5</td>
<td>Computational Creativity</td>
<td>116</td>
</tr>
<tr>
<td>8.5.1</td>
<td>Discovering Games Procedurally</td>
<td>117</td>
</tr>
<tr>
<td>8.5.2</td>
<td>Bounding the Search Space</td>
<td>118</td>
</tr>
<tr>
<td>8.5.3</td>
<td>Sampling Games in Parallel</td>
<td>119</td>
</tr>
<tr>
<td>8.5.4</td>
<td>Playing Unexpected Games</td>
<td>120</td>
</tr>
<tr>
<td>8.5.5</td>
<td>Finding Unique Games</td>
<td>121</td>
</tr>
<tr>
<td>8.5.6</td>
<td>Most Unique Variants via Genetic Optimization</td>
<td>123</td>
</tr>
<tr>
<td>8.5.7</td>
<td>Results and Discussion</td>
<td>124</td>
</tr>
<tr>
<td>8.6</td>
<td>Conclusions</td>
<td>126</td>
</tr>
<tr>
<td>9</td>
<td>Exploring Characteristics of Combinatorial Games</td>
<td>127</td>
</tr>
<tr>
<td>9.1</td>
<td>Introduction</td>
<td>127</td>
</tr>
<tr>
<td>9.2</td>
<td>Catch-Up</td>
<td>128</td>
</tr>
<tr>
<td>9.3</td>
<td>Examples of Play in Catch-Up</td>
<td>130</td>
</tr>
<tr>
<td>9.3.1</td>
<td>Catch-Up({1, ..., 4})</td>
<td>130</td>
</tr>
<tr>
<td>9.3.2</td>
<td>Physical Implementation</td>
<td>131</td>
</tr>
<tr>
<td>9.3.3</td>
<td>Web-based Implementation</td>
<td>131</td>
</tr>
<tr>
<td>9.4</td>
<td>Characteristics of Catch-Up</td>
<td>132</td>
</tr>
<tr>
<td>9.4.1</td>
<td>Puzzle-Like Quality</td>
<td>132</td>
</tr>
<tr>
<td>9.4.2</td>
<td>Endgame</td>
<td>132</td>
</tr>
<tr>
<td>9.4.3</td>
<td>Drawn Games</td>
<td>134</td>
</tr>
<tr>
<td>9.4.4</td>
<td>Importance of the First Move</td>
<td>135</td>
</tr>
<tr>
<td>9.4.5</td>
<td>Game-Tree Size</td>
<td>135</td>
</tr>
<tr>
<td>9.4.6</td>
<td>State-Space Size</td>
<td>136</td>
</tr>
<tr>
<td>9.4.7</td>
<td>Game-Tree Depth</td>
<td>136</td>
</tr>
<tr>
<td>9.4.8</td>
<td>Branching Factor</td>
<td>137</td>
</tr>
</tbody>
</table>
I 0  Exploring Characteristics of Dice Games

10.1 Introduction ................................. 146

10.2 Rolling Sorted or Unsorted . . . . . . . . . . . . . . . . . . . 149
   10.2.1 Sorting Dice, With Ties .......................... 149
   10.2.2 Dice Unsorted, With Ties .......................... 150
   10.2.3 Sorted Vs. Unsorted .............................. 152

10.3 Resolving Tied Battles ......................... 153

10.4 Favoring One Player ........................... 153
   10.4.1 Rolling Sorted, Player A Wins Ties ............... 153
   10.4.2 Rolling Unsorted, A Wins Ties .................... 154

10.5 Reducing Bias With Fewer Dice ............... 155

10.6 Reducing Bias With Mixed Dice ............... 156
   10.6.1 Mixed Dice Sorted, A Wins Ties ................. 156
   10.6.2 Mixed Dice Unsorted, A Wins Ties .............. 157

10.7 Rerolling Tied Dice .......................... 158
   10.7.1 Rolling Sorted, Rerolling Tied Dice ............ 159
   10.7.2 Rolling Unsorted, Rerolling Ties .............. 161
   10.7.3 Sorted, A Wins Ties, Rerolls Highest ......... 161

10.8 Risk & Risk 2210 A.D. ........................ 162

10.9 Conclusions .................................. 164

II Conclusions .................................... 166

11.1 Confirmation of the Thesis Statement .......... 166

11.2 Future Work .................................. 167

11.3 Final Thoughts ............................... 169
List of Figures

1.1 Example of game space of Space Invader-like variants ........................................... 2
1.2 Difficulty curve measuring perceived difficulty as a game progresses. ...................... 4
1.3 Iterative game design feedback loop. ........................................................................... 4

2.1 Three versions of Flappy Bird with different difficulty ............................................. 21
2.2 Game space parameters for Flappy Bird. .................................................................... 21
2.3 An example of the Box-Packing Game ...................................................................... 23
2.4 An example of a Puzzle Bobble puzzle. ...................................................................... 23
2.5 Canabalt one-button infinite runner ........................................................................... 25
2.6 Drop7 turn-based puzzle game ................................................................................... 26

3.1 Score probability histogram ......................................................................................... 28
3.2 Score probability distribution function \( f(x) \) for two games ........................................ 29
3.3 Cumulative distribution function \( F(x) \) for two games ................................................ 30
3.4 Survival function \( S(x) \) for two games ....................................................................... 31
3.5 Hazard rate function \( h(x) \) for two games .................................................................. 32
3.6 R code for calculating survival function data from discrete scores. ............................ 34

4.1 Modeling skill by adjusting probability of executing moves on time. ......................... 40
4.2 Modeling precision by adjusting timing ...................................................................... 41
4.3 Two Flappy Bird levels with different game parameters ............................................ 43
4.4 Finding an appropriate number of samples ................................................................... 45
4.5 First pipe can be easier than subsequent pipes ............................................................ 46
4.6 Beating pattern anomalies solved by randomizing starting position ........................... 46
4.7 Measuring precision with a user study ....................................................................... 47

5.1 Demonstrating strategy and dexterity errors ............................................................... 50
5.2 Classification of games requiring dexterity and strategy ............................................ 50
5.3 Puzzle Bobble puzzles requiring dexterity but not strategy ..................................... 55
10.17  Reroll probability for Figure 10.16  ........................................ 159
10.18  Rerolling ties with \( nd6 \) sorted  ............................................ 160
10.19  Reroll probability for Figure 10.18  ........................................ 160
10.20  Rerolling ties for \( 5dk \) unsorted  ........................................ 162
10.21  Reroll probability for Figure 10.20  ........................................ 162
10.22  Rerolling ties for \( nd6 \) unsorted  ........................................... 163
10.23  Rerolling ties when rolling unsorted  ..................................... 163
## List of Tables

4.1 Calculating the number of samples for simulation ........................................ 45  
6.1 Probability of achieving a streak length of $k$. ........................................ 86  
6.2 Summary of analytical and empirical findings from this chapter .................. 90  
8.1 Results of the user study for evaluating our method ...................................... 115  
9.1 Percentage change with $P_1$ moving first .................................................. 135  
9.2 Measures for $Catch-Up(\{1, \ldots, N\})$ for values of $N = 3$ to $N = 18$ ........ 136  
9.3 Average ($B_{avg}$) and maximum ($B_{max}$) branching factors .......................... 137  
9.4 Optimal play values for $Catch-Up(\{1, \ldots, N\})$ ..................................... 140  
9.5 Win percentages for $P_1$ against different heuristics ................................... 141  
9.6 $P_2$ win rate using combination heuristic ..................................................... 142
Chapter 1

Introduction

1.1 Motivation and Overview

Scores are an essential characteristic of many games [63], where the player or team with the highest score at the end of the game is declared the winner. Over the course of multiple games, the total number of wins for a team or player is scored to determine the ranking of the participants. For some games, like single player video games or individual sports, the score is also used to compare performance with other events in the past for determining records. Even for games with win conditions that do not rely on score, such as Chess, there are systems like Elo to assign a score to each player to quantify their relative skill at the game [64]. In some games, different numbers of points are given for different achievements (e.g. in American Football different scoring events can award either 6, 3, 2, or 1 points [79]). Changing the points assigned for each achievement has an impact on how the game is played; for example, the introduction and tuning of the 3-point line in Basketball has significantly changed the sport [250, 182]. Clearly, scores are critically important to games.

While scores are a crucial part of many games, the analysis of game scores is also interesting to observers, players, managers, and game designers. Newspapers and websites track box scores for the amusement of sports fans, dating back to at least 1859 when Henry Chadwick published the first baseball box score [169]. The careful collection of score data and game information is useful for examining the performance of each player for managing teams, historical interest, and for playing fantasy league games [104, 128, 239]. Many players use similar score tracking, statistics, and metrics for popular video games like StarCraft II [23], Hearthstone [24], and League of Legends [180] to actively monitor performance of other players and improve their own strategies and play [1, 2, 217]. Finally, game designers use score information and player data for understanding how players react to a game and how to make changes to improve the player experience [56, 62, 101].
Games, regardless if they are played with cards, tokens, dice, or computers, must be carefully designed and tuned to provide a positive experience for players. A game is defined by its rules, and those rules can have parameters. For example, in the classic game Space Invaders [158], rules define that the player dies when hit by a bomb, rockets destroy aliens, and aliens move horizontally and speed up as more of them are killed. The parameters numerically define how fast the bombs fall, the rate that the aliens move and speed up, and the size of the aliens and player (see Figure 1.1). In turn, these rules and parameters define the game, which is then experienced by the players in a manner that is indirectly informed by those rules and parameters through game play. Understanding the characteristics of games – such as how certain types of rules interact [188], how players will perceive the difficulty of the game [4], or how much randomness affects the game play [48] – is critical for good design and good game play [63].

The discovery of better games, new games, and entirely new game genres also requires exploration through the design space of possible games; this search is performed either by human [51, 73, 4, 191], machine [153, 37, 225, 47, 227], or together [203, 130, 112]. We can consider each possible game, with its unique rules and parameters, as a point in a large space of games. We refer to this high-dimensional space of game variants as game space [92]. Game design space is massive, high-dimensional, and noisy: small changes to a game can make a huge difference in its playability. In this thesis, we seek a better understanding of what makes games work using algorithmic and quantitative techniques,
showing they can lead to significant insights. Because scores are quantitative by nature, they are particularly useful data for exploring game space and the quantitative analysis of games.

A significant challenge for designing and analyzing games arises because games are interactive and typically need to be experienced in order to fully understand their characteristics. The availability of strategies to play a game are part of how the game is experienced, but these are not directly defined by the rules and parameters. Instead, strategies that are practical and effective are discovered by engaging with the game itself. Games that afford lots of interesting heuristics that can be applied in various ways can lead to interesting and unexpected game play [122]. Because of bounded rationality, humans do not think entirely logically and we can find deep strategic thinking difficult; heuristics can help provide shortcuts for real-time decision making [77]. Especially for video games and sports (though some tabletop games as well, e.g. Billiards [5], Crokinole [246], or Jenga [192]), there is a critical dexterity component to how the game is played and experienced. Motor skill limitations [134] (e.g. how fast can someone react to a new stimulus) and speed-accuracy trade-offs [243] (i.e. the faster someone needs to react, the less accurate their response will be) lead to different types of in-game challenges. Accessibility [109] of a game – that is, how easy it is for a player to process the game state and respond to its affordances and interface – provides yet another variable that effects how a game is experienced. Finally, engaging games are played repeatedly, and players learn and improve in their abilities and skill. This provides a dynamic environment where players of different skill levels are playing the game in different ways. All of these factors affect how a game is perceived, demonstrating that a game generally must be played in order to understand its characteristics.

The measurement of the characteristic of difficulty is a particularly important focus of this research. By difficulty, we mean perceived difficulty, such that the difficulty of a game is determined with respect to the perception of a particular player or class of player [4]. A game might be easy for an expert, but difficult for a novice. Perceived difficulty is typically not constant for a game; for example, in Figure [1.2] we show a difficulty curve of how difficulty increases as a game progresses for an average player. We expect a training level to be easy, and the final climax of the game to be difficult. The curve increases and decreases, to keep players interested and to not overwhelm them [52, 211]. These curves are typically used in design as guidelines and thought experiments, but here we calculate them quantitatively and precisely from score data. Furthermore, we show how one can use these perceived difficulty curves to tune existing game to varying levels of difficulty for a variety of skill levels.

Nonetheless, designers and players know from experience that some rules or parameters will just not work. So not all characteristics of games must be experienced in order to approximately evaluate them. The careful study of games and game design can give us
shortcuts and deeper understanding of how new rules and parameters may be perceived. A major goal of this thesis is to quantify some principles of game design through techniques that can be transferred and applied more generally to the game design process.

Typically, the game design process is guided by a human game designer's intuition and experience, combined with user feedback to iteratively search for new or better games [73, 159]. We present an interactive design feedback loop in Figure 1.3. First, the designer must estimate player skill and set the game rules and parameters for a game. This generates a new, unique game variant defined by those specific rules and parameters. Users playtest the game, and metrics are recorded that gives data on how each player performed [57, 62]. These metrics are then analyzed to understand and model the perceived player experience. The resulting statistics inform design decisions that the designer then uses to make adjustments in the rules and parameters. This loop is continued, with each iteration generating a new variant, until the designer decides that the design process can stop [94].

However, this time-proven approach to design has several weaknesses. Firstly, playtesting is expensive and time consuming; finding players who will focus on a specific game for extended periods of time is challenging in itself. Because playtesting can be costly and

![Figure 1.3: Iterative game design feedback loop.](image-url)
resource intensive, a designer may be unwilling to try new ideas that have a high risk of failure. Similarly, a designer may be comfortable working in a particular area of design space, and may not think of exploring other areas. When a designer or playtester becomes an expert at the game they are evaluating, they can lose perspective on how the game is experienced by new players. Additionally, a game often needs playtesting very early in the design, but it can be technologically difficult to playtest games that are not fully formed and distributable. Furthermore, after working on the design of a game for extended periods of time, one can be afraid to try new things because of the risk of having to throw away existing work. Yet modifying or eliminating existing ideas and content is typically required to breakout of local optima and explore creative new regions of game space. Finally, when humans are in the design loop, the speed at which search and innovation can occur is limited by the speed at which humans can operate.

Artificial intelligence (AI) [187] has great potential to aid in the iterative design process. Simulation and automated playtesting [152, 256], when combined with player modeling that helps model, predict, and simulate how humans behave, can rapidly and cheaply generate accurate score data by experiencing the game at ever-increasingly high speeds instead of being bound by human time scales. Statistical methods and machine learning [7, 209] can be used to analyze scores and other metrics to detect patterns and characteristics from the data. Computational creativity [25, 244, 131], the process of using machine intelligence to create new artifacts and ideas, can aid the designer’s task of interpreting statistics to decide which changes to make to the game. We use various other techniques and methods from artificial intelligence, such as minimax search [187], Monte Carlo tree search [38], genetic programming [171], procedural content generation [225, 208], and parameter optimization via Differential Evolution [173]. Leveraging these AI techniques, this thesis demonstrates how to automate the entire iterative design loop, leveraging computation to benefit the human goal of generating and understanding interesting games.

Our methodologies do not model many reasons for playing games; instead, we assume that players are trying to maximize their score. While some players may try hard to get on the public leaderboard, we recognize other players may simply be playing to relax or to unlock all of the game content without caring about score. While this aspect of players may be abstracted away as an aspect of play style in the simulation model, certain play styles associate with certain motivations [17, 73], which in turn has implications for evaluating simulation results. For example, players chasing high scores may take more moves with high risk and high potential reward, while someone playing to relax will tend to make low cognitive-load selections [73].

1“Every act of creation is first an act of destruction” – Pablo Picasso
We do consider that a given game will be played by a population of players, where this population will generally encompass a variety of player types and skill levels. When modeling, we must make a decision about the players in the population. To predict how a game will be experienced in the real world, a designer needs to target their game to a particular population and then estimate how that population might experience the game. A puzzle that is trivial for the population of expert players may be highly engaging for players with low game experience – the difference lies in the population being analyzed.

The goal of using artificial intelligence for this research is not to significantly outperform humans, but to provide a framework where score data analysis and game play simulation can be leveraged to learn more about games. We do not aim to play games at ever-improving superhuman levels, for example the powerful AI agents designed to play Chess (Deep Blue [42]), Go (Alpha-Go [199]), Jeopardy (Watson [69]), or Poker (DeepStack [144], Libratus [32]). Nor are we trying to entirely replace humans in the design process [37, 47, 223]. The application we imagine is more in the spirit of co-creativity and mixed initiative design, where human designers work together with computers to build a game in concert, each producing unique ideas [252]. Although our experiments operate without human assistance, they show that one can effectively use AI-assisted quantitative methods to learn more about the design space they are operating in. Not only can this be useful for those engaged in the craft of game design or the experience of playing games, but the study of games extends into fields of sociology [87], anthropology [36], humanities [30], media studies [148, 26], psychology [29], and economics [216]. After all, humans have been creating games for thousands of years, and the need for play goes far back in our evolutionary history [210].

1.2 Thesis Statement

This dissertation addresses the problem of using data analysis, artificial intelligence, and player modeling to better understand games and game design using quantitative techniques. In this section, we present the thesis statement and discuss in detail how each part will be treated in this dissertation. Specifically, we will demonstrate with novel peer-reviewed published research (see Section 1.5) performed in the process of this multi-year project that:

The analysis of score distributions, combined with artificial intelligence, simulation, player modeling, and human game play data, is useful for understanding the characteristics of games and quantitatively exploring the space of possible games.
1.2.1 Analysis of Score Distributions

Score distributions describe how often a particular score will be achieved by a player or a player population. We examine several methods for analyzing score distributions in games:

- **Survival Analysis** is a statistical technique commonly used to model lifetimes, for example for mechanical and electrical parts, drug effectiveness in medical trials, and insurance risk calculations [123]. By equating the final score of a simulated agent with a lifetime – since in many action games the longer the player stays alive the higher their score – we use survival analysis to predict the likelihood of a player achieving a specific final score. We focus on survival analysis in Chapters 3 and 7.

- **Expected Value and Mean Scores** - For games where the score isn’t related to survival time, we use expected values and mean scores to understand the likelihood of players achieving particular scores and records. In Chapter 5, we examine interactive puzzle games that require dexterity and strategy and in Chapter 6, we examine how expected values are related to high scores and streaks.

- **Win Rate, Tie Percentage, and Closeness** - We explore several derivative metrics that are based on score. In Chapters 9 and 10, we examine win rates (also called win bias) to determine how likely one player will beat another in multiplayer games, and use tie percentage metrics to understand how likely a game is to draw. In Chapter 10, we use our novel closeness metric, related to the inverse of the second moment of a distribution, to understand how likely close scores in dice games will occur.

- **Maximal Value and High Score Analysis** - In Chapter 6, we examine how high scores, the maximum score achieved by a player in a time series up to a specific point, become more difficult to set the more times a game is played. This implies unintended negative player experiences may occur by making goals increasingly harder to reach. We also examine how often leaderboard spots (e.g. a top 10 high score list) and streaks (i.e. consecutive wins) are likely to be achieved.

- **Visualization** is the process of projecting data sets into a 2-D graphical illustration to take advantage of the human visual system for information processing [229]. This work leverages many visualization techniques for exploring and understanding score distribution data sets. Of particular interest is the visualization of varying amounts of dexterity and strategy required for games in Chapter 5, quantitative difficulty curves in Chapter 7, game space difficulty plots in Chapter 8, game tree analysis in Chapter 9, and game design metrics and probability charts in Chapter 10.
1.2.2 Artificial Intelligence

In this thesis, we address various applications of artificial intelligence to the problem of better understanding games and game design. The models used in this research are intended to be computationally simple so they can be run quickly, transparent about how they function, and effective at searching game space. In particular, we want to avoid machine learning “black box” models that makes opaque predictions and require long training periods such as for reinforcement learning [209]. We also are more interested in general solutions that can be applied to large classes of games instead of only specific games. By focusing on score distributions, we limit the amount of heuristic knowledge required to analyze games.

We use a variety of techniques, specifically:

- **Tree Search**: Game tree search can determine optimal moves and best available alternatives. In Chapter 5, we use depth-first search with limited search depth to model strategic decision making. In Chapter 9, we use minimax search with transposition tables and move pruning [187] for analyzing small games and Monte Carlo tree search [38] for analyzing larger games too complicated for exhaustive search.

- **Genetic Programming**: This stochastic technique generates functions to maximize a particular fitness metric [171]. In Chapter 9, we use genetic programming to build simple novice-level heuristics, where the heuristic is evolved to maximize win rate.

- **Differential Evolution optimization**: Global optimization algorithms attempt to efficiently find the optima of a function. However, with stochastic simulation, we are optimizing noisy functions and cannot use strategies such as gradient descent that rely on differentiable functions [177]. Differential Evolution [173] is designed for stochastic functions, and we use it in Chapter 8 for searching game space to find new representative variants and for finding games of a specific difficulty.

- **Clustering**: We leverage unsupervised clustering [7] techniques such as k-means [110] to find interesting games in a design space. In Chapter 8, we use clustering and other unsupervised optimization methods for finding representative unique games.

- **Procedural Content Generation**: Procedural content generation (PCG) techniques are used to generate new game content without human input [225, 194, 208]. We use PCG techniques for generating new levels for simulation and novel game variants in Chapter 8.

- **Reinforcement Learning and Q-Learning**: Although we ultimately did not use reinforcement learning [209] techniques due to their slow training speed and di-
vergence when dealing with randomized inputs, we tested their use for modeling strategic decisions in Chapter 5. Nonetheless, action state value functions inspired by Q-learning [241] are used for modeling move selection in our work.

### 1.2.3 Game Play Simulation

As important as it is to have validation and ground truth from human players, gathering data from human playtesters can be time-consuming, expensive, and error prone. However, simulated playtesting can provide many benefits over human playtesting [152]. We use game play simulation throughout this thesis for various purposes, including:

- **Repeatable, Controlled Experiments**: In Chapters 4 through 10 we run repeatable verifiable experiments that control precisely the environment and parameters. AI agents perform exactly as programmed, avoiding issues that affect human playtesters such as fatigue, mood, change in ability, anchoring, etc.

- **Exhaustive Game Coverage**: In Chapters 5, 9, and 10, we evaluate all possible game states for small games to determine the probability of specific events occurring. Exhaustive coverage generally would not be possible with human playtesters.

- **Parallelizable and Scalable**: In addition to parallelizing across multiple cores for low-cost linear speed improvement (Chapters 7 and 10), accuracy can be improved to any desired level by running more simulation samples (Chapters 4 through 10).

- **Variable Skill Evaluation**: While human playtesters have partially unknown previous game experience and skills, simulated skill can be precisely specified. In addition, novice human players will gain skills while playing the game, causing them to no longer play like novices. Simulated game play can control for skill level and learning effects, which we explore in Chapters 4, 5, 6, 7, and 8.

### 1.2.4 Player Modeling

Accurately simulating game play requires an understanding of how players react to game events: this is the process of player modeling [200, 253]. We use several player models in this thesis, to model different types of human limitations. We use open modeling approaches that are interpretable, avoid training time, and limit the amount of game-domain-specific expert knowledge. While our techniques are intended to be generic and can be applied to a wide variety of games without encoding many game specific behaviors into the AI agents, the agents do need to act with human-like error making behavior.
Our goal is to approximate human play – not to emulate it perfectly – so that we can quickly learn something about the underlying game system. With a user study, one can measure how humans estimate difficulty; however, this presents the same problems that cause us to prefer simulation. Fitting difficulty as a function of puzzle game parameters (e.g. size of board, number of unique pieces, etc.), one can accurately predict the difficulty of specific strategic puzzle games [236]. Yet when the models are game specific they can overfit and be ill-suited for search based AI-assisted game design that requires more generic metrics [37]. Given existing human play traces, one can use machine-learning trained player models to predict a wide range of emotions (fun, challenge, frustration, predictability, anxiety, and boredom) [166], to model human-like decision making to play game-theory matrix games [82], to have more human-like behavior in action games [221, 162], to model personas [85], or by modeling slower decision making and limited actions per section [114]. We avoid approaches that prerecord human play traces or use live players for data collection on existing games [256, 58, 88, 139].

In this thesis, we model the following human game play traits:

- **Timing Accuracy** - The timing accuracy player model assumes that much of the difficulty in minimal action games like *Flappy Bird* is due to human motor skill, specifically precision and reaction time [134]. This type of player model is explored in Chapter 4 where we adjust how accurately a player presses buttons. Survival analysis is used to examine the effects of timing skill on perceived difficulty in Chapter 7.

- **Aiming Accuracy** - The targeting and the placement of pieces, a different type of dexterity skill, is modeled in Chapter 5. We model dexterity challenge found in *Puzzle Bobble* when aiming and shooting balls.

- **Strategic Thinking** - Many games also have a challenging strategic component to determine the best move to make. For strategy, as discussed in Chapters 5 and 9, we model what moves a player might select and their error in making that choice.

- **Inequity Aversion** - Humans often desire to keep scores close, to limit feelings of one side dominating the other in an unfair or unbalanced game [67]. In Chapters 9 and 10 we calculate how much score inequity occurs in different types of games.

- **Learning** - As players continually play a game, they improve their skills [97, 94, 98]. This in turn may make a game easier to play. In Chapters 6 and 7 we measure this effect in *Canabalt*, *Drop7*, and *Flappy Bird* score distributions and verify that learning can be modeled with power laws [121].
1.2.5 Human Game Play Data

While a major focus of this dissertation is on scores obtained through simulation, we also demonstrate how our techniques can be used with, or rely on, human game play data [62].

- **Score Data and Learning** - In Chapter 6, we explore how game play data recorded from human players of Canabalt and Drop7 can determine how quickly players learn and improve. We also validate our learning models in Chapter 7 using millions of play sessions from flappybird.io.

- **Quantifying Human Error** - To be sure that our models correctly model human traits, we run small user studies to determine specific model parameters for the AI agents. In Chapter 4, we measure how accurately human players can time button presses as well as the maximum number of button presses per second they can execute. In Chapter 5, we measure human aiming accuracy in Puzzle Bobble.

- **Validating Models** - In Chapter 6, we use human game play data to validate our theoretical models of high scores and player improvement. In Chapter 8, we validate the difficulty of game variants. In a small user study, we compare our estimates and ranking of difficulty with that of human players to measure our algorithm’s accuracy.

1.2.6 Characteristics of Games

In Characteristics of Games, Elias, Garfield, and Gutschera define *characteristics* as “general groups of features that give a high-level description” of the game [63]. For example, the number of players and length of play are two important characteristics for games. They also recognize that these characteristics may be things that “only fairly advanced players are likely” to recognize, such as “is it possible for a losing player to catch up” or “are there a lot of difficult or boring calculations” required to play well. In general, the authors are focused on “player-centric questions” that address how the players will experience the game.

Here are the characteristics of games that are explored in this dissertation using data analysis of score distributions obtained via human play or AI play. All quoted phrases are taken from Characteristics of Games [63], and the “(CoG #.#)” indicator describes which section of the book each characteristic comes from.

- **Length of Playtime (CoG 1.1)**: For games where one’s score is directly related (or at least correlated) to how long the play session lasts, then the final score is an indicator of the length of playtime. This occurs in games like Flappy Bird and Canabalt, where the score is proportional to game time.
- **Heuristics (CoG 1.3):** Heuristics are “rules of thumb” that help players “gain mastery in the game over time”. Games that exhibit easy to learn heuristics are likely to be more enjoyable for beginners, who will be more likely to continue to play. When games have additional heuristics that can be applied by more advanced players, a heuristic ladder can develop where study and repeated play can lead to improved skill and further enjoyment. We examine how simple heuristics affect game play in Chapters 5 and 9. In short, if a simple human-usable heuristic permits an AI agent to reach higher scores, then we can use the score distribution to compare the effectiveness of different heuristics. Similarly, by changing the skill parameters of the agents, we can determine how the heuristics might be effective at different skill levels.

- **Rules (CoG 3.1):** Rules are “instructions telling players what actions they can take and what the outcome of various actions will be”. In this thesis, we separate game rules from game parameters (although this distinction is not made in the Characteristics of Games book) such that “at the end of your turn, draw \(N\) cards” is a game rule but \(N\) is a game parameter. Changing just game parameters can have a major effect on the game; this is explored in Chapters 7, 8, 9, and 10.

- **Outcomes (CoG 3.3):** By creating score distributions with AI agents that model human play, one can estimate what human players might be likely to achieve. By comparing different levels of skill, one can determine the effect skill has on the final ranking of performance of the players (Chapters 5 and 8). Similarly, we determine how often draw games will occur (Chapters 9 and 10).

- **Ending Conditions (CoG 3.4):** Point systems are used to determine who is the winner of a game, or how well someone performed. Using simulation, different scoring systems can be rapidly tested to determine how final scores might be distributed. In Chapter 5, we demonstrate how points are assigned can affect which strategies and heuristics are viable. Furthermore, in Chapters 9 and 10, we examine how rules and parameters can control closeness in the final scores in a multiplayer game.

- **Catch-up (CoG 4.2):** The catch-up characteristic is defined as “features whose purpose is to help losing players catch up.” We examine how building a catch-up rule into a combinatorial game (Chapter 9) ensures that scores do not diverge above a fixed number. We also explore how this determines usable strategies for human players.

- **Game Balance (CoG 4.4):** Lack of balance can occur when “a small number of strategies are much better than the others” making the game less interesting than
intended. In Chapter 5, we explore how much effect an error in evaluating the best strategic move has on the final score that a player achieves. We discuss in Chapter 7 how difficulty curves for modeling how challenge changes throughout a game can be quantified using survival analysis. In Chapter 9, we use AI agents to explore how many strategies are available for novice players, and demonstrate how genetic optimization can generate higher-order heuristics for human play.

- **Randomness** (*CoG 5.1*): Randomness and luck are defined as “uncertainty in outcome”, and there are many sources of randomness in games. We examine how randomness applies to modeling human motor skill in Chapter 4, to dexterity and strategy error in Chapter 5, and for dice games in Chapter 10.

- **Luck and Skill** (*CoG 5.2*): The concept of return-to-skill is based on measuring how much player ability is correlated with the final score achieved. Games with more luck have less return-to-skill. We measure this directly in Chapters 4, 5, 7, and 8 by adjusting the skill level of the players and determining how much the scores distributions change based on the independent skill variable. Luck is also examined in the context of high scores in Chapter 6 and dice games in Chapter 10.

- **Costs and Skills Players Need** (*CoG 6.1*): The costs a player incurs for mastery of a game include “developing (or possessing already) certain skills or abilities”. Simulation with player modeling can help determine what skills are required to master a game variant. This type of skill estimation is explored in Chapters 5, 8, and 9.

### 1.2.7 Exploring Game Space

Game space, the set of all game variants for a specific game, is high-dimensional and often impossible to search exhaustively – imagine adjusting hundreds of independent control knobs to search for the perfect game. In this thesis, we reduce the search space by focusing on game variants that only change parameters, not the larger class of variants that include changes to game rules. We use various techniques for exploring game space, including:

- **Monte Carlo Simulation** - To explore game space, we use unbiased randomized simulations to both cover a wide region of game space but also within each game variant. This stochastic simulation helps model randomness and human error, and is heavily used in Chapters 4 and 5. We also use Monte Carlo techniques to explore large game spaces in Chapter 8, large state-space game variants in Chapter 9, and for simulating dice reroll probabilities in Chapter 10.
• **Computational Creativity** - In Chapter 8, we explore the use of search to generate unique game variants for the purpose of creating novel content via algorithms.

• **Sampling and Visualization** - Chapter 8 includes methods for sampling and visualizing game space along one-, two-, and higher-dimensional slices. Additionally, along with Chapter 5, we discuss methods for sampling two-dimensional difficulty (strategy and dexterity) and for exploring how parameter changes can affect how much of each type of difficulty is expressed in a game.

• **Search through Optimization** - To find unique game variants, we use various search and optimization techniques in Chapter 8. This covers both pure action games like *Flappy Bird* as well as interactive puzzle games like *Tetris* and *Puzzle Bobble*.

• **Applications to Game Design** - Exploring game space to find interesting games and game variants is covered for interactive puzzle games in Chapter 5 and action games in Chapter 8. We also explore game space in Chapter 10 to understand how various dice parameters such as number of dice and number of sides affects game play.

• **Mathematical Modeling** - We apply various techniques that do not use stochastic Monte Carlo simulation to better understand the space of games. In Chapter 6, we use calculus to prove that for all single player games, no matter their difficulty curves, the more a game is played the more difficult it becomes to set a new high score. We use survival analysis in Chapter 7 to analytically model how changes to game parameters can increase the difficulty of a game. In Chapter 9, we analyze a combinatorial game using analytical techniques similar to that used in *Scientific American* articles by Martin Gardner [74] and *Winning Ways For Your Mathematical Plays* [21]. Finally, in Chapter 10 we calculate probability tables for all possible outcomes of various dice games to determine their fitness for use in balanced and interesting game play.

Together, these concepts and chapters work in concert to prove the thesis statement of this dissertation. We now address the novel contributions of this thesis and research.

### 1.3 Contributions to the State of the Art

This thesis contributes to the state of the art of the field of computer science and quantitative game design. To our knowledge, the peer-reviewed published research (and the work still under single-blind review) presented in this dissertation contains several firsts:
• **Survival analysis applied to game design** [92, 97, 94]: Survival analysis – while common in fields such as mechanical engineering, actuarial science, and medicine – has not been used before for the analysis of games and game difficulty.

• **Quantitative Calculation of Difficulty Curves** [97, 94]: While difficulty curves have been widely discussed as a model in qualitative game design, this work presents the first quantitative framework for producing these graphs via hazard rate data analysis.

• **Discovering New Games Through Parameter Search** [92, 93]: Our approach of only changing game parameters while keeping rules fixed is shown in this work to generate unique games such as *Frisbee Bird* [92] that can stand on their own as new games and not just minor variants.

• **Proving High Scores Are Increasingly Difficult To Achieve** [98]: We prove using analytical calculus that the more games played, the harder it is to achieve a high score, no matter the difficulty or skill of the player. Our analysis of *Canabalt* and *Drop7* scores from thousands of players empirically verifies these results.

• **Simulating and Measuring Strategy and Dexterity Requirements** [100]: While researchers have examined games of pure strategy or pure dexterity, we present a new model for exploring interactive puzzle games, which combine dexterity and strategy. Studying strategy and dexterity simultaneously captures nuances and effects that cannot be observed by modeling them independently.

• **Closeness Metric for Measuring Score Inequity** [95]: We present *closeness*, a novel metric related to the inverse of the second moment centered about 0, which measures how close the final scores in a two-player game are expected to be. This helps analyze how much score inequity may be present in a game, for addressing inequity aversion.

• **Modeling Human Motor Skills for Game AI Agents** [92, 94, 100]: We present a novel framework for modeling human *precision*, which is the motor skill accuracy at which someone can precisely perform an action at a predicted time in the future, and *dexterity* for modeling aiming accuracy for moving game pieces.

• **Methodical Analysis of Specific Games** [92, 94, 96, 100, 95]: We perform a deep analysis of *Flappy Bird*, the novel combinatorial game *Catch-Up*, interactive puzzle game *Puzzle Bobble*, and *Dice Battles* commonly found in war and strategy games.
1.4 Organizational Outline

To summarize the chapters and organization of this thesis:

Chapter 1: Introduction begins with an overview of the themes of this dissertation. We declare the thesis statement to be proven, and we break down each of the thesis statement’s constituent parts to explain how will prove each part. This is followed by a list of the main scientific contributions of the dissertation and this outline. We present the research publications that contributed to this thesis and a list of co-authored work that is related to the topic but not part of this dissertation. Finally, we present notes on style for this thesis.

Chapter 2: Description of Games gives a short description of the games that are analyzed in this thesis, including Flappy Bird, a simulated Box-Packing Game, Tetris, Puzzle Bobble, Catch-Up, Dice Battles (for games like Risk), Canabalt, and Drop7. This provides an easy reference as these games are referred to throughout the thesis.

Chapter 3: Modeling Score Distributions provides an introduction to the analysis of score distributions, presenting a simple mathematical framework for discussing and comparing score distributions in continuous and discrete domains. This chapter explains the formulation for score histograms, score probability distributions, cumulative distribution functions, survival functions, hazard rates, and metrics that are used in various chapters.

Chapter 4: Modeling and Simulating Timing Skill in Action Games uses player modeling to simulate how human players perform at action games that require accurate timing. We focus on games that are difficult due to imperfect human motor skill, specifically precision and reaction time. By adjusting the timing accuracy of an AI agent, we can simulate different player types, since novices react slower and are less precise than experts.

Chapter 5: Modeling and Simulation of Strategy and Dexterity in Puzzle Games examines the impact of strategy and dexterity on video games that require a player to use strategy to decide between multiple moves while requiring dexterity to correctly execute such moves. We run simulation experiments on variants of Puzzle Bobble. By modeling dexterity and strategy as separate components, we quantify the effect of each type of difficulty using AI player modeling.

Chapter 6: Modeling Learning and High Scores investigates using score distributions to model how players learn over time, and how that affects a player’s ability to set a new high score. Using analytical probabilities, simulated game data, and actual game analytics data from Canabalt and Drop7 popular mobile games, we show the probability of reaching a high score decrease rapidly the more one plays, even when players are learning and improving.

Chapter 7: Survival Analysis of Simulated Games analyzes the difficulty of two different simulated games by using survival analysis to generate quantitative difficulty
We examine how changes to the simple Box-Packing Game and Flappy Bird impacts the distributions of scores that result from the simulation. We test versions where parameters are constant or increasing, with simulated novice and expert AI players.

**Chapter 8: Exploring Game Space and Computational Creativity** provides methods for exploring the high-dimensional space of game variants, using score distributions to measure each variant’s quality. This helps us find specific settings for a desired difficulty, as well as to better understand the relationship between game parameters and player experience. We use search, optimization, and visualization to analyze game space and to discover new game variants using computational creativity.

**Chapter 9: Exploring Characteristics of Combinatorial Games** examines how score distributions within a game can affect the potential quality of a game. In particular, we examine how the score difference between two players can be used as a mechanic in the novel combinatorial game Catch-Up. We analyze the game for interesting game properties, and we perform an analysis of game strategies using simple AI agents that model novice play. In addition, we use genetic programming to evolve new strategies for the game.

**Chapter 10: Exploring Characteristics of Dice Games** explores how score distributions can be used to measure the characteristics of closeness, win rate, and tie percentage in simple dice games (such as what might appear as an individual battle in a strategy war board game). By adjusting parameters such as the numbers of dice rolled or the number of sides on each dice, we can measure the resulting effects on the game’s characteristics.

**Chapter 11: Conclusions** reviews the entire thesis, summing up the contributions and confirms the thesis statement. We present ideas for future work on advancing the state of the art beyond this dissertation research, and give some final thoughts on AI and games.

### 1.5 Relevant Publications

Much of the work in this thesis is the result of published research or papers currently under review, in collaboration with other researchers in the field of AI-assisted game design. In particular, the following papers make up a large part of this dissertation:


Aaron Isaksen, Dan Gopstein, Julian Togelius, and Andy Nealen. Exploring Game Space of Minimal Action Games Via Parameter Tuning and Survival Analysis. *Submitted to IEEE Transactions on Computational Intelligence and AI in Games*, 2017. [94] (Chapters 3, 4, 7, and 8)
Aaron Isaksen, Drew Wallace, Adam Finkelstein, and Andy Nealen. Simulating Strategy and Dexterity for Game Design. Submitted to IEEE Transactions on Computational Intelligence and AI in Games, 2017. [100] (Chapter 5)


Aaron Isaksen, Dan Gopstein, Julian Togelius, and Andy Nealen. Discovering Unique Game Variants. In Computational Creativity and Games Workshop at Intl. Conf. on Computational Creativity, 2015. [93] (Chapter 8)


In addition, these published papers about the theory of games, player modeling, and the application of artificial intelligence and machine learning to games, do not comprise major sections of the thesis text but are related to the topic. Although I have contributed significant portions to these papers as first author or co-author, they are not about the analysis of score distributions and are thus not included as chapters in this thesis.

Aaron Isaksen, Julian Togelius, Frank Lantz, and Andy Nealen. Playing Games Across the Superintelligence Divide. In AI, Ethics, and Society Workshop at the Thirtieth AAAI Conference on Artificial Intelligence, 2016. [99]

Rishabh Jain, Aaron Isaksen, Christoffer Holmgård, and Julian Togelius. Autoencoders for Level Generation, Repair, and Recognition. In Computational Creativity and Games Workshop (CCGW) at International Conference on Computational Creativity (ICCC), 2016. [103]

Ahmed Khalifa, Aaron Isaksen, Julian Togelius, and Andy Nealen. Modifying MCTS for Human-like General Video Game Playing. In International Joint Conference on Artificial Intelligence (IJCAI), 2016. [114]


Adam Summerville, Sam Snodgrass, Matthew Guzdial, Christoffer Holmgård, Amy K. Hoover, Aaron Isaksen, Andy Nealen, and Julian Togelius. Procedural Content Generation Via Machine Learning (PCGML). *Submitted to IEEE Transactions on Computational Intelligence and AI in Games*, 2017. [208]

1.6 Notes on Pronouns and Style

Much of the work in this thesis is the result of collaboration, even though I am the first author of all the work discussed in herein. Instead of switching between “I” and “we”, the thesis will use the common scientific practice of using the pronoun “we” to refer to all the researchers involved in the work, including when it is just myself. As much as possible, “we” can also refer to the author and reader.2

This thesis is written in a gender-inclusive style. We use “they” and “their” as gender-neutral singular pronouns, to avoid attributing gender where it is not necessary. The singular *they* has been used in the English language since the 14th century.3

The term “player” can interchangeably refer in this thesis to a human or a machine, unless explicitly denoted as “human player” or “AI player”. We therefore often refer to a player as “they” even though a machine player is factually an “it”. This does not imply that we think machines are human, should be treated as humans, or will some day replace humans.4 It is simply better to err on the side of referring to a machine player as “they” instead of a human player as “it”.

1.7 Source Code

Source code for the work in this thesis is available at [https://github.com/aisaksen/phd](https://github.com/aisaksen/phd).

---

2This follows the style guidelines from Steven Pinker’s 2015 book *The Sense of Style: The Thinking Person’s Guide to Writing in the 21st Century*.

3An excellent overview of the singular *they* can be found here: [https://en.wikipedia.org/wiki/Singular_they](https://en.wikipedia.org/wiki/Singular_they)
Chapter 2

Description of Games

In this chapter, we give a short description of the games that are discussed in this thesis. Instead of the definition of these games being spread throughout the dissertation, they are collected in this chapter for ease of reference. Many of these games we study are minimal action games, defined as games with “small rule sets, narrow decision spaces, and abstract audiovisual representations” but are still deep and interesting games [151].

2.1 Flappy Bird

*Flappy Bird* [156] is a minimal one-button action game originally released on iOS in 2013. It is a commercial and critical success [120], spawning hundreds of similar games, and the rules are simple to implement and understand. In *Flappy Bird*, a player must fly a constantly moving bird without crashing into a series of pipes placed along the top and bottom of the screen (Figure 2.1a). Each time the player taps the screen, the bird flaps its wings, moving upward in an arc while gravity constantly pulls downward. Each time the bird passes through a pipe gap, the player scores a point.

Part of the appeal for *Flappy Bird* is the comically high difficulty, especially when simple mobile games are typical easy and forgiving. *Flappy Bird* could have been much easier with a few adjustments, such as increasing the pipe gap or decreasing the width of the pipes (Fig. 2.1b), or made harder by decreasing the gap or increasing the pipe width (Fig. 2.1c). However, these changes would have led to different, potentially less rewarding experiences.

We define the following parameters for our implementation of *Flappy Bird* (see Figure 2.2). The original *Flappy Bird* has a constant value for each parameter since the game does not change as the player progresses. In general, game parameters can change as the player progresses in the level, so we also examine variants where parameters can change as the player scores points.
Figure 2.1: In *Flappy Bird*, the player must navigate the bird through a series of pipes without crashing. Three versions of the game with different difficulty.

Figure 2.2: Game space parameters for *Flappy Bird*. 
• **Pipe Separation** $p_s$ – Horizontal space between consecutive pipes. More distance between pipes is easier to play, giving more time to react to changing gap locations.

• **Pipe Gap** $p_g$ – The distance between the upper and lower pipes. Narrower gaps are more difficult as the bird has less room to maneuver, requiring better motor skills.

• **Pipe Width** $p_w$ – Wider pipes increase difficulty as the bird spends more time in the narrow pipe gap.

• **Pipe Gap Location Range** $l_r$ – Gaps are uniformly distributed in a range between ceiling and floor. Larger range between high and low gaps is harder.

• **Gravitational Constant** $g$ – Acceleration of the bird in the $y$ direction, subtracted from the bird’s $y$ velocity each frame. Higher gravity causes the bird to drop faster, lowering the margin of error.

• **Jump Velocity** $j$ – When the bird flaps, its vertical velocity is set to $j$, making it jump upward. Higher velocity makes higher jumps.

• **Bird Velocity** $v$ – Speed at which the bird travels to the right (alternately, the speed at which pipes travel to the left).

• **World Height** $H$ – Distance between ceiling and floor, defined by the display.

• **Bird Width and Height** $b_w, b_h$ – Size of the bird’s hit box. The wider and taller the bird, the harder it will be to jump through gaps.

By varying these parameters within sensible ranges, we can generate all variants of *Flappy Bird* that use the same set of rules. Many of these parameters have constraints; for example, they all must be positive, and Bird Height $b_h$ can not be larger than Pipe Gap $p_g$ or the bird can not fit through the gap.

We use *Flappy Bird* for experiments in modeling human precision error (Chapter 4), generating empirical data for survival analysis (Chapter 7), and for exploring game space and computational creativity (Chapter 8). We will also use actual human game data collected from *flappybird.io* [136], an online variant of the game.

### 2.2 Box-Packing Game

Our *Box-Packing Game* [97], shown in Figure 2.3, was inspired by the packing peanut mini-game in *Peter Panic* [135]. Our game is a minimal one-button game, designed to be
easy and fast to simulate. The player is standing above a conveyor belt with an endless series of empty boxes that slide from right to left. The player must tap a button to instantly pack each box while it is under the player. If the player misses a box or taps when there is no box under them, they lose the game. The final score of the game is equal to the number of boxes successfully packed. We can expect that wider boxes would make the game easier since the player has a wider window for error, and faster belt speed would make the game harder since the player would have less time to react to each box. In Chapter 7, we generate empirical data for this game to explore the effect of belt speed on game difficulty.

Figure 2.3: In this simple simulated Box-Packing Game, the player earns a point by tapping a button once when the box is underneath. The game ends if a box is missed or the player taps when there is no box. In some variants, the belt speeds up after each point.

2.3 Puzzle Bobble

In Puzzle Bobble [149] (Figure 2.4), the player shoots colored balls towards other colored balls hanging from the top of the play field. The fired ball bounces off the walls and sticks to any balls it hits or the top of the field, aligning to a hexagonal grid. Once the ball lands, if it creates a connected component of at least 3 balls with its own color, the balls will disappear

Figure 2.4: An example of a Puzzle Bobble puzzle.
and any balls no longer connected to the top of the field are dropped and removed. When firing at other balls, points are scored by the size of the connected component and optionally the number of unconnected balls that are dropped (depending on the experiment). Each puzzle sets the balls already in the play field, the number of colors, and the order and length of the queue of balls to fire. Our variant also contains a swap mechanic, which lets the player switch the current 1st and 2nd balls in the queue.

Game parameters for Puzzle Bobble include the width and height of the play field, the placement of the initial colored balls, the composition of the queue, the optional amount of points received for dropped balls, and an optional chain-bonus factor for rewarding consecutive successful shots. In Chapter 5, we run various experiments on our Puzzle Bobble variant to quantify strategy and dexterity requirements in interactive puzzle games.

### 2.4 Catch-Up

Catch-Up [96], is a minimal game with simple rules that can be learned quickly, designed to incorporate inequity aversion into its core game play. The rules are as follows:

1. Catch-Up starts with a collection of numbers $S$, which we call the set. Two players, $P_1$ and $P_2$, each begin with a score of zero. Player $P_1$ starts the game.

2. Starting with $P_1$, the players alternate taking turns. On a turn, the acting player removes a number from $S$ and adds that number to their score. The player keeps removing numbers, one by one, until their score equals or exceeds the score of the other player.

3. When there are no more numbers in $S$, the game ends. The player with the higher score wins; the game is drawn if scores are tied.

We analyze Catch-Up and its rules in Chapter 9 to examine how score distributions within a game can affect the quality of a game, focusing on reducing score inequity and other metrics.

### 2.5 Dice Battles

Dice Battles are not a complete game, but a subgame within a larger strategy game such as Risk. In a dice battle, players roll several dice and compare the individual dice values. The

---

1The demo game and code are available at: [http://game.engineering.nyu.edu/catch-up](http://game.engineering.nyu.edu/catch-up)
dice are sorted in decreasing order and then paired up. Whichever player rolled a higher value on the pair wins a point. The player has more points wins the battle.

In Chapter [10] we examine different variants of dice battles and show how different factors affect the distribution of scores and other metrics that are helpful for evaluating a game. The variants we examine include different numbers of dice, various sided dice, different ways to sort the dice, and various ways to break ties. By adjusting the dice mechanics, a designer can influence the expected closeness of the battles, the win bias in favor of one of the players, and the tie percentage of battles that end in a draw.

### 2.6 Canabalt

*Canabalt* [189] was created by Adam Saltsman in 2009 and is one of the canonical examples of the one-button infinite runner genre, available on many platforms including iOS, Android, Mac, PC, Linux, and more. The player is running over a series of building roofs, and taps a button to jump over gaps, falling objects, and gaps between buildings. The longer the button is held, the higher the player jumps. The faster the player is running, the less time they have to react to the randomized events. Thus, one strategy is to purposely run into boxes to slow down the player. In Chapter [6] we examine thousands of game play scores from *Canabalt* to determine how frequently players set high scores in simple action games and to evaluate player learning rates.

![Canabalt, a canonical example of the one-button infinite runner genre.](image)
2.7 Drop7

*Drop7* [12] is a mobile turn-based puzzle game originally developed and released by Area/Code Entertainment in 2009. The game is played on a 7x7 grid where the player stacks numbered circles making rows and columns that contain the indicated same number of circles; clearing circles requires careful strategic planning as typical in many deep puzzle games (see Figure 2.6).

In Chapter 6 we examine thousands of game play scores from *Drop7* hardcore mode to determine how frequent players set high scores in turn-based puzzle games and to evaluate learning rates for player improvement.

![Drop7 screenshot](image)

Figure 2.6: *Drop7*, a canonical example of the turn-based puzzle game genre.
Chapter 3

Modeling Score Distributions

Since this thesis focuses on the analysis of score distributions, this entire chapter is devoted to presenting a simple mathematical framework for modeling and comparing score distributions. These models are agnostic to how the game play data is collected: it might be from AI agents, from human players, or from a mixture of the two. Although player data, both recorded and simulated, will typically arrive in the form of discrete data, it is useful to provide an analysis of score distributions in both the continuous and discrete domain. We also examine various metrics that can be used to describe score distributions. Armed with the framework introduced in this chapter, we can then later progress to the application and exploration of the framework throughout the rest of the dissertation. This chapter uses models and metrics also used in several of our existing publications: [92, 97, 95, 94, 100].

3.1 Score Probabilities

We begin by defining a score probability as the probability that a player will achieve a particular score in a game. We can use the notation $Pr[score = x]$ for the probability that a player achieves a score of $x$ on a particular playthrough. This probability depends on the game’s rules and parameters, as well as the skill level of the player.

A score probability can be calculated for a game by analyzing a histogram of all the scores achieved by all of the players. For example, lets say that a game is played 20 times, and the final scores are [0, 0, 0, 10, 10, 10, 20, 20, 20, 30, 30, 30, 30, 30, 40, 40, 40, 50, 50]. We can count how often each value appears and plot it in a histogram, as indicated in Figure 3.1. On the left y-axis we list the frequency of each score (e.g. a score of 10 occurs 3 times). On the right y-axis we list the probability of each score, which is calculated by dividing the frequency of each score by the total number of scores (e.g. a score of 10 has a 3/20=.15 probability).
3.2 Probability Distribution Functions (PDF) $f(x)$

Given the definition of score probability from the previous section, we can define a score probability distribution function $f(x)$ for a player’s experience over all the scores they might achieve. This score probability distribution function (PDF)$^1$ tells us the probability that the player will achieve a score of $x$ on the next play, for all valid values of $x$. Because players perform differently on every play of a game, $x$ can take on many values and we treat the player’s score as a random variable. For example, if $f(30) = .25$ then the probability a player will achieve a score of 30 on the next play is 25%.

In practice, we can create a discrete probability distribution by recording all of the scores on each play of the game, then summing up the frequency of each score, and dividing by the total number of plays. This is often recorded for each play when using analytics such as Google Analytics, and the frequency of each score is easily output by these systems. For games with a wide range of scores or with gaps between scores, specific individual scores might have zero or very low frequencies and thus one can first quantize and bucket the scores in a histogram.

In Figure 3.2, we see an example of two different games. In the easier game, indicated in black, there is more likelihood of achieving a higher score than in the harder game, indicated in red. Thus, if we play both games the same number of times, then we would expect higher scores more often in the easier game.

In this thesis, we use score probability distributions to compare games or compare

---

$^1$For simplicity and clarity, we use probability distribution function (PDF) to refer to a probability density function when using continuous probabilities and a probability mass function when using discrete probabilities.
players. Instead of treating the examples in Figure 3.2 as an easy game and hard game, we use an alternate model and say each line represents different players with different abilities playing the same game. As we will see in Chapter 7, this may also resolve into two different probability distributions, depending on the player’s ability and the game’s characteristics.

![Score Probability Distribution](image)

Figure 3.2: Score probability distribution function $f(x)$ for two games. The game indicated in red is harder, because it has more probability of having a lower score than the easier game indicated in black.

### 3.3 Cumulative Distribution Function (CDF) $F(x)$

The cumulative distribution function $F(x)$ tells us the probability that a player’s final score on a play of the game will be less than $x$. For example, if $F(10) = .75$ there is a 75% probability that the player will achieve a score less than 10. A lower value in the CDF means a higher likelihood that player will get a higher score than $x$.

The cumulative distribution function (CDF) is non-decreasing and can be defined for continuous and discrete domains as follows:

CDF (continuous): $F(x) = \int_{s=0}^{x} f(s)ds$  \hspace{1cm} (3.1)

CDF (discrete): $F(x) = \sum_{s=0}^{x-1} f(s)$  \hspace{1cm} (3.2)

For continuous mathematics, it does not matter if we define the CDF as ‘$< x$’ or ‘$\leq x$’ because the integral is the same either way. However for the discrete case, the math will
be simpler if we use the \(< x\) formulation, so that \(F(0) = 0\) when \(x \leq 0\) because every player will at least achieve a score of 0. \(F(\infty) = 1\) because all players will eventually reach a termination state and receive a final score. Typically a game will have a maximum achievable score \(M\), in which case \(F(M) = 1\) as well.

In Figure 3.3, we see the cumulative distribution function \(F(x)\) for the two sample games previously used in Figure 3.2. The shapes of the CDF for the two games is once again considerably different. The curve for the easier game is below the harder game, indicating that the probability of ending with specific score is higher in the harder game. The CDF for the easier game also has a slight S shape, indicating that the game is easier at the start.

![Cumulative Score Probabilities F(x) for Two Games](image)

Figure 3.3: Cumulative distribution function \(F(x)\) for the two example games of Fig. 3.2. The game indicated in red is harder, because its curve is higher – implying that this game has a higher probability of achieving a score \(< x\) than the easier game indicated in black.

### 3.4 Survival Functions \(S(x)\)

The survival function \(S(x)\) tells us the probability that a player will reach a score \(\geq x\) on the current play. In other words, it describes the likelihood that a player’s character will still be alive at a score of \(x\). For example, if \(S(10) = .25\) there is a 25% probability that the player will achieve a score higher than 10. The survival function is defined from the CDF:

\[
\text{survival function (continuous): } S(x) = 1 - F(x) = 1 - \int_0^x f(s)ds \tag{3.3}
\]
survival function (discrete): \( S(x) = 1 - F(x) = 1 - \sum_{s=0}^{x} f(s) \) \hspace{1cm} (3.4)

\( S(x) = 1 \) when \( x \leq 0 \) because every player will at least achieve a score of 0, and \( S(\infty) = 0 \) because all players will eventually reach an end state and receive a final score.

In Fig. 3.4 we see the survival function \( S(x) \) for the two sample games previously used in Fig. 3.2. These follow the same shapes as the CDF \( F(x) \), but flipped vertically. Thus, the two games have identifiably different characteristics because of the underlying difficulty.

![Survival Function S(x) for Two Games](image)

**Figure 3.4**: Survival function \( S(x) \) for the two example games of Fig. 3.2. The game indicated in red is harder, because it appears lower on the graph – implying that there is a lower probability of surviving past a score of \( x \).

### 3.5 Hazard Functions \( h(x) \)

The hazard function \( h(x) \) is useful for comparing probability distributions and understanding the difficulty of a game. Also called the hazard rate, the hazard rate function, or just the hazard, it is defined as:

\[ h(x) = \frac{f(x)}{S(x)} \quad f(x) = h(x)S(x) \] \hspace{1cm} (3.5)

The hazard function tells us the rate at which we should expect to fail, given we’ve already reached a specific score \( x \). This is not the probability that we will fail at this exact score \( x \), given by \( f(x) \), but a conditional probability that the player has already survived to
Figure 3.5: Hazard rate function $h(x)$ for the two example games of Fig. 3.2. The red game has a constant difficulty. The black game linearly increases in difficulty.

For a score of $x$. For example, if $h(10) = .15$, this means that once the player gets to a score of 10, they now have a 15% chance of failing at this point.

In Figure 3.5 we plot the hazard function for the two example games. We can see that the game in red has a constant hazard rate while the game in black has a linearly increasing hazard rate. From the hazard, we can now see that the game we’ve been calling the “easier” game is actually only easier for scores < 20, and is actually more difficult for scores > 20. However, this is only clearly apparent by examining the hazard rate.

The hazard function is especially useful when analyzing games because its directly related to how we as designers think about adjusting difficulty curves in a game. We are not as concerned about the entire probability distribution as much as how difficult a game is at a specific section, assuming the player has already reached that point in the game.

Therefore, game designers are effectively working on modifying the hazard function $h(x)$ so the difficulty curve is well balanced and feels good to players [211 52]. If we want to understand how these changes affect the resulting probability distribution $f(x)$ and survival function $S(x)$ for the game, we can derive $f(x)$ and $S(x)$ from a given hazard $h(x)$.

First, we write the cumulative hazard function $H(x)$, which represents the total amount of risk [43] that a player has faced up to their current score $x$, as:

$$H(x) = \int_0^x h(u)du$$

(3.6)
We then take the derivative of Eq. 3.3:

\[ f(x) = -\frac{dS(x)}{dx} = -S'(x) \]  

(3.7)

Using Eq. 3.7, we rewrite Eq. 3.5 as \( h(x) = -S'(x)/S(x) \). This is a first order differential equation with the following solution [28]:

\[ S(x) = e^{-\int h(x)dx} = e^{-H(x)} \]  

(3.8)

By using Eqs. 3.7 and 3.8 we obtain the final relation:

\[ f(x) = -\frac{dS(x)}{dx} = -\frac{d}{dx}e^{-H(x)} \]  

(3.9)

We can now derive a theoretical \( f(x) \) and \( S(x) \) from any theoretical \( h(x) \), which helps us understand how changes in difficulty affect the resulting score distributions. We will use this technique extensively in Chapter 7 when discussing game specific difficulty curves. For more information on hazard rates, an excellent analysis can be found in [179].

### 3.6 Working with Discrete Data

We use continuous distributions when modeling, but since scores from our games are discrete values (that is we can receive a score of 1 or 2, but not 1.245), we work in the discrete domain when analyzing game data. Kaplan-Meier Curves [115] can be used to generate discrete survival functions from data and are somewhat related to the method we use here.

In Figure 3.6, we present a simple R function \texttt{survival} for calculating a data frame \( z \) of survival analysis data from a vector of unsorted scores. The \texttt{table} function counts how often each score appears in \( \texttt{scores} \), creating a histogram. We set \( n \) to be the total number of scores. \( z \) will be a data frame with columns representing each unique score \( x \), PDF \( f \), survival function \( S \), CDF \( F \), and hazard function \( h \). The function \texttt{cumsum} calculates a cumulative sum, and \texttt{rev} reverses a vector of elements.

Because \( S(x) \) becomes very small as higher scores become less likely, \( h(x) \) can be susceptible to noise. One option is to smooth the hazard function using techniques based on kernel smoothing [147] or splines [183]. For this dissertation, we simply do not plot \( h(x) \) for values of \( x \) where \( S(x) < \epsilon \), where \( \epsilon \) ranges between 0.001 and 0.03 depending on how many samples were used to generate the plots. This avoids the noisiest parts of the hazard function, which are undersampled due to low probability.
survival <- function (scores) {
    histogram = table(scores)
    n = sum(histogram)
    z = data.frame(x = as.numeric(names(histogram)))
    z$f = histogram/n
    z$S = rev(cumsum(rev(z$f)))
    z$F = 1-z$S
    z$h = z$f/z$S
    z
}

Figure 3.6: R code for calculating survival function data from discrete scores.

3.7 Score Distributions Metrics

It can be useful to use statistics of a score distribution to describe the shape and to provide numeric comparisons between distributions. However, there is no obvious single metric that works for all games. Instead, a metric can be chosen to measure a particular trait that the designer is trying to optimize. This section describes several descriptive metrics and how they can be used to measure and compare score distributions. Quantitative metrics have been used to computationally analyze outcome uncertainty in games, typically for the purpose of generating novel games [33, 8].

3.7.1 Mean Score

We can use the mean score as a descriptive statistic that indicates which game is easier or harder. Given two game variants, the variant with a higher mean score is easier than the one with a lower mean score. To calculate the mean score of a score PDF, we use the continuous or discrete equations:

\[
\text{Mean score (continuous)} = \int_{0}^{\infty} x f(x) dx
\]

\[
\text{Mean score (discrete)} = \sum_{0}^{\text{max}} x f(x)
\]

The mean score of the easier game in Figure 3.2 is 17.28, while the mean score of the harder game is 9.41. Note that it is impossible to achieve a score of 17.28, so the median may be a more useful statistic in practice as it is more robust to outliers, and can be calculated analytically by solving \( F(x) = .5 \) for \( x \). If we assume that these example games were
played 10,000 each, the median for the easier game would be 17 and median for the harder game would be 7. This comes to the same relative result.

### 3.7.2 Puzzle Game Metrics

For many games, especially puzzle games, there is a maximum score per puzzle. In order to compare histograms of scores achieved by a variety of players on a different puzzles, we can normalize the scores between 0 (lowest) and 1 (highest). To normalize, we simply divide each score by the maximum achieved score. Depending on the scoring system used in the game, it may also make sense to normalize based on the logarithm of the score; this would be useful for games with bonus multipliers that imply exponentially distributed scores.

To compare different score distributions, several common metrics for probability distributions can be useful. Because we are dealing with discrete scores for puzzles, we use the discrete formulations for these metrics. We assume large sample sizes to avoid statistical corrections for small sample sizes. Some common and useful statistical metrics are:

- **normalized mean:** The mean score of a puzzle is the expected value of the normalized score probability distribution. A low mean implies we expect a player to get a lower score; when comparing two puzzles with normalized scores, the one with the lower mean is expected to be more difficult. The mean is not guaranteed to be an actual achievable score: if one only has two equally likely normalized scores of 0.0 and 1.0, then the mean will be an unachievable score of 0.5. We will use the normalized mean score in Chapter 5.

- **median** - The median score of a puzzle is the score at which half of the players achieve a lower score and half achieve a higher score. The median is much less sensitive to outliers than the mean, and when there are a lot of example scores in the histogram, the median will typically represent an actual normalized score achievable in the puzzle. Lower medians imply harder puzzles; higher medians imply easier puzzles.

- **mode** - The mode score of a puzzle is the most common score, telling the designer which score is most likely to occur. For a tutorial or easy level, the designer will likely want the mode score to be the maximum score.

- **standard deviation:** The standard deviation of a puzzle score distribution tells us how spread out the scores will be. Lower variances imply the scores are tightly distributed about the mean. Higher variances imply the scores are more spread out. One may desire higher spread for more distance between players on the leaderboard.
other hand, lower standard deviation is desirable for training levels to ensure the players are following.

- **number of unique scores**: The number of unique scores describes how many different solutions will be achieved by a player population. This helps determine the resolution of the scoring system; the more scores that are likely, the more possible grades and ranks a player can receive.

- **maximum achievable score**: The maximum achievable unnormalized score on a game is useful when comparing different games. Perceptually, players may prefer a game that has higher points, as people tend to prefer larger numbers over smaller ones [232, 168, 238].

- **skewness** - The skewness metric describes the amount of asymmetry in the score distribution. When the skewness is zero, the scores are symmetric about the mean; when skewness is positive, the distribution is shifted to the right and the tails on the right side have more probability; when skewness is negative, the distribution is shifted to the left and tails on the left side have more probability. Larger positive values of skewness imply a harder game with most players getting a low score and just a few getting a higher score; more negative values of skewness imply an easier game with most players getting a high score and just a few getting a low score.

### 3.7.3 Win Bias, Tie Percentage and Closeness

We now focus on metrics that examine the final scores of two-player games, developing the metrics of **win bias**, **tie percentage** and **closeness** to capture properties of the relationship between the two players’ scores. **Win bias** and **tie probability** are similar to those used in previous work, but one of our metrics, **closeness**, is something we have not seen used before in game analysis and we introduced in [95].

We now define these three metrics precisely. Let $s_A$ be the final score for Player A, and $s_B$ be the final score for Player B. The **score difference** $d$ is defined as $d = s_A - s_B$. If we iterate over all the possible ways that the final scores can occur, and count the number of times each score difference occurs, we can make a **score difference probability distribution**, $f(d)$. This describes the probability of achieving a score difference of $d$ in the game. We calculate $f(d)$ by first counting every resulting score difference in a histogram data structure, and then dividing each bin by the total sum of all the bins.

We now define the **win percentage** as the percent probability of Player A winning a game. This can be calculated by summing the probabilities where the score difference is
positive and is therefore a win for Player A. This is calculated as $100 \sum_{d>0} f(d)$ and will be between 0% and 100%. Loss percentage is the percent probability of Player A losing a battle, and is calculated as $100 \sum_{d<0} f(d)$, also between 0% and 100%.

We take the difference of the win and loss percentages to calculate win bias:

$$\text{win bias} = 100 \left( \sum_{d>0} f(d) - \sum_{d<0} f(d) \right)$$ (3.12)

This will vary between -100% and 100%. Games with a win bias of 0% are balanced, with no preference of Player A over Player B. If the win bias is > 0% then Player A is favored; if < 0% then Player B is favored. This metric is similar to the Balance metric in [33] but here we include the effect of ties and are concerned with the direction of the bias. A non-zero win bias is often desired, for example when simulating that one player is in a stronger situation than the other.

Next, we have the tie percentage, which tells us the percent probability of the battle ending in a tie, defined as:

$$\text{tie percentage} = 100 (f(0))$$ (3.13)

Some designers may want a possibility of ties, while others may not. This metric is analogous to drawishness in [33].

Finally, we present closeness, our metric that measures how much the final score values center around a tied game. Game that often ends within 1 point should have higher closeness than games that often end with a score difference of 5 or -5. The related statistical term precision is defined as the inverse of variance about the mean. For closeness, we define this as the square root of the inverse of variance in the score difference distribution about the tie value $d = 0$:

$$\text{closeness} = \frac{1}{\sqrt{\sum_{d} d^2 f(d)}}$$ (3.14)

To explain this, we look at the denominator, which is similar to the standard deviation as the square root of variance. However, we do not want this to be centered about the mean as in the typical formulation. A game that always ends tied 0-0 would have a variance of 0, but so would a game that always ends in 5-0 because the outcome is always the same. Yet 5-0 is certainly not a close score. Thus, we center the second moment around 0 since close games are those where the final score differences are almost 0. Finally, we take the inverse because we want the metric to increase as the scores become closer and decrease as the scores become further apart.
This formulation mirrors the well-known term “close game”, and the values of closeness have some intuitive meaning. Closeness approaching 0 means that the final score differences are very spread out. Closeness approaching $\infty$ means the scores are effectively always tied. A closeness of $C$ means that a majority of the score differences will fall between $-1/C$ and $1/C$. If a game can only have a score difference of -1 or 1, its closeness will be exactly 1, no matter if it is biased or unbiased. If we also allow tie scores (score differences of -1, 0, or 1), we would expect the game to have more closeness – in fact for this case closeness will always be $> 1$.

### 3.8 Conclusions

Now that the mathematical framework and metrics have been described for analyzing score distributions, we can proceed to applying them to game data in subsequent chapters. Chapters 4 and 5 focus on generating data for score distributions. For analyzing game difficulty, Chapters 5 and 8 use score metrics and Chapter 7 uses survival and hazard functions. We use metrics and probability distributions for analyzing high scores in Chapter 6, action games in Chapter 8, combinatorial games in Chapter 9, and dice games in Chapter 10.
Chapter 4

Modeling and Simulating Timing Skill in Action Games

We present a method for modeling player motor skills and simulating how players will experience simple action games that rely on timing, based on our work published in [92, 94]. Accurately simulating game play requires an understanding of how players react to games: this is the process of player modeling [200]. Our model assumes that difficulty in simple action games is due to human motor skill, specifically precision and reaction time [134].

4.1 Introduction

Our goal is to make the simulator play like a human (novice, average, or skilled), not play with superhuman ability. As long as the model properly predicts human perception of difficulty, it fits our purposes. In minimal, well-balanced, and compelling action games like Flappy Bird (Sec. 2.1) or Canabalt (Sec. 2.6), the player takes a relatively obvious path, but executing that simple path is challenging [151].

In this chapter we examine how to model dexterity skill in Flappy Bird (rules of the game are described in Section 2.1). Our method is similar to modeling “biological constraints” [72] but focuses on human motor skill ability instead of perception and sensory error. Because Flappy Bird requires very simple path planning and does not have enemies, we can focus on the player’s ability to control the game – without analytical evaluation of possible player movement [45], multi-factor analysis of player or enemy strategies [71], dynamic scripting of opponents [205], building machine learning estimators [221, 195, 251], or evaluating design via heuristics [55]. Since we use a static, objective, simulation-based player experience model [254], we do not need to analyze prerecorded human play sessions of the specific
game to train our player model, and do not rely on live players to estimate the level of challenge or fun [58, 88, 139, 166, 221, 256].

4.2 Player Model for Simulating Timing

We begin with a model of a player with perfect motor skills – a perfect player with instantaneous reaction who would never lose at the original Flappy Bird. Given a version of Flappy Bird defined by its game parameters as described in Section 2.1 and Figure 2.2, we create an AI that finds a path through the pipes without crashing. Instead of using an A* planner that finds the shortest path, we use a simpler AI that performs well but is easier to implement and faster to run. Each time the bird drops below the target path, the AI immediately executes a flap (which sets vertical bird velocity $v_y$ instantly to jump velocity $j$). Whatever AI is used, it should play with very good performance on solvable levels, and should mainly only fail on impossible levels, such as a level with a tiny pipe gap where the bird cannot fit through. There are many paths which a bird could take to survive in a game; our chosen path satisfies survival with computation time.

We then extend the AI to perform more human-like by modeling the main components of human motor skill that impact difficulty in these types of action games: precision (Sec. 4.2.1), reaction time (Sec. 4.2.2), and actions per second (Sec. 4.2.3). Adjusting these values lets us model different player types, since novices react slower and are less precise than experts.

4.2.1 Player Precision

When a player plans to press a button at an exact time, they execute this action with some imprecision. We model this error as a normal distribution with standard deviation proportional to a player’s imprecision (see Figure 4.1). Imprecision is an inherent trait, but is also related to the time a subject has to react to an event, called the speed-accuracy tradeoff.
trade-off: the less time they have to react, the less accurately they will respond [243]. For simplification, our player model assumes precision is an independent variable and not dependent on bird speed.

To measure the value of $\sigma_p$, we performed a user study (Section 4.3.5) to measure precision as an error with standard deviation ranging between $\sigma_p = 35.9 \text{ ms}$ and $\sigma_p = 61.1 \text{ ms}$ ($N = 20$), and use this range for our simulations. Because this data collection was performed in a setting abstracted from Flappy Bird and focuses only on the measurement of precision and not the game’s visual presentation, this study provides an base from which we can apply its player model parameters to other action-based games.

We model imperfect precision in our AI by calculating an ideal time $t$ to flap, then adding to $t$ a small perturbation $\epsilon$, drawn randomly from a normal distribution $\mathcal{N}(0, \sigma_p)$ with 0 mean and standard deviation $\sigma_p$, as shown in Figure 4.2. By increasing the standard deviation $\sigma_p$, the AI plays less well and makes more errors, leading to a higher difficulty estimate (see Section 8.3.1.1 and Figure 8.2 for the impact of varying precision). Reducing $\sigma_p$ to 0 ms lets us test if a level is solvable by the AI without human error.

This technique can be used when the decision to press a button at a particular time has been made by the player, but we need to simulate how accurately that press is made. We also use this to model precision for the Box-Packing Game in Chapter 7 and in [97].

### 4.2.2 Reaction Time

When a player sees a new pipe show up on the screen, it takes some time to react. The speed of the player’s reaction is influenced by factors inherent to the system [89], as well as factors affecting the player themselves [215]. We measured an average reaction time of $\tau = 288 \text{ ms}$ ($N = 20$) in our user study (Section 4.3.5).
We constrain the AI to react only after it has observed a new pipe for \( \tau \) ms of simulated time: the AI will keep the previous target height until enough simulated time has passed for the AI to react to the next pipe location. We found in our Flappy Bird experiments the delay has minor impact on estimating difficulty, and mostly matters for bird speed settings that are exceedingly fast and unlikely to be received positively by players.

4.2.3 Actions Per Second

Humans can only perform a limited number of accurate button presses per second. In our user study, we measured an average maximum rate of 7.7 actions per second \((N = 20)\). We also limit our AI to this same number of actions per second. We simplify our model by keeping this constant, although a more complex model would account for fatigue, since players can not keep actions per second constant for long periods.

4.3 Simulating Action Games and Timing

We now present our methodology for collecting score distributions data with AI agents using randomized simulations and our player model. The main loop works as follows. First, we determine how many samples to collect. Then, for each sample, we生成 a new game variant based on the given parameters and then simulate using an AI to play the game with human-like behavior. Generate and Simulate steps are repeated until we have a stable score distribution for analyzing.

Under the player modeling taxonomy [200], this simulation step is a Universal Synthetic Generative Action player model, as it makes choices about when to take actions by simulating the types of errors that humans make.

4.3.1 Generate

Each simulation begins by taking a set of parameters and generating a new game variant. This involves placing the bird and pipes in their starting positions and randomly distributing the pipe gaps. In Figure 4.3, we show two different generated game variants.

Because the levels are generated using a random process, it is important to generate a new level each time the AI runs, even though the parameters do not change. Otherwise, if the same random layout of pipe gaps is used repeatedly, artifacts can arise in the score distribution caused by a particular placement of the gaps. For example, the gap locations can randomly come out approximately equal for a section of the level, making that section
Figure 4.3: Two different levels created by the Generate step given different game parameters. More simulations complete the second version, so it has an easier estimated difficulty. The red lines indicate the target location for the AI to flap.

easier. These artifacts are averaged out by generating a new level each time. We explain these anomalies in more detail in Section 4.3.4

4.3.2 Simulate

Given the level created in the Generate step, we use a simple heuristic for Flappy Bird to find a path through the pipes by flapping when the bird reaches a line in the lower half of the gap (these lines are drawn in red in Figure 4.3). At each frame of the simulation, we predict the next time \( t \) in the future when the bird will drop below this ideal flapping location – the ideal player would flap at exactly this time \( t \). By reducing or increasing the standard deviation \( \sigma_p \) of our precision model (Section 4.2.1), the AI plays more accurately or less accurately. We can model less skilled players with a higher standard deviation. We quickly check if a variant is impossible by using \( \sigma_p = 0 \) ms on a limited number of simulations, and only continue testing if a majority of the these simulations score highly. It is important to note that our AI does not need to be perfect to detect if a game is possible, as the AI with \( \sigma_p = 0 \) ms performs far better than humans.

To keep the AI from flapping faster than a human could tap, \( t \) is limited by the number of actions per second. We also limit the AI lookahead to only use information that has been visible on the screen for at least \( \tau \) (the time it takes for a player to react to a new event). In our experiments, \( \tau \) did not have much effect except in extreme situations where humans would perform poorly anyway.

For each simulation, we get a score equal to number of pipes that the AI passes before crashing, and we record each score in a histogram. If the AI reaches a goal score \( s_{\text{max}} \), we terminate the run so we do not get stuck simulating easy games where the AI will never
crash. Although Flappy Bird can theoretically go on forever, human players tire or get distracted and will eventually make a terminal mistake, but the AI can truly play forever unless we enforce a maximum score. We set $s_{\text{max}}$ to range between 20 and 200, depending on the experiment we are running (higher $s_{\text{max}}$ is slightly more accurate, but takes longer to simulate). The Generate and Simulate steps are run repeatedly until we have enough samples to adequately analyze the histogram. We calculate the proper number of simulations in the next subsection.

### 4.3.3 Number of Simulations

When running randomized algorithms, it is essential to have a high number of samples to avoid artifacts: too few samples and the estimate of difficulty will be highly variable, while too many samples requires longer simulation time. We can find a good number of simulations $n_s$ by running an experiment $k$ times for a fixed set of parameters, and examining the resulting score distribution. Lets say we have some metric for difficulty based on the score distribution, which we can generally call $D$ for difficulty. For Flappy Bird and the Box-Packing Game, the general metric $D$ will be the average hazard rate of the game (see Section 3.5), which is a good estimate of the true hazard rate if the game has constant difficulty. We describe how to calculate the hazard rate for Flappy Bird and other games in Chapter 7. For this section, the important thing about $D$ is that higher values mean the player finds the game more difficult.

Each experiment will give a slightly different value of $D$ due to randomness in the stochastic simulation. After $k$ trials, we measure the mean $\mu_D$ and standard deviation $\sigma_D$ of the distribution of values for $D$. As $k$ increases, $\mu_D$ will trend towards the true difficulty value $D'$, giving us a confidence interval that our simulation correctly estimates the difficulty with some given probability. By increasing $n_s$ the standard deviation $\sigma_D$ will decrease, tightening the confidence interval. Similar to Probably Approximately Correct (PAC) learning bounds [7], we choose a bound $\epsilon$ such that with chosen probability $\delta$ all estimates of $D$ will fall within $\epsilon$ of the true $D'$.

Using the definition of a CDF of a normal distribution [125], we can solve for $\sigma_D$:

\[
\text{CDF}(x, \mu_D, \sigma_D) = \frac{1}{2} \left[ 1 + \text{erf} \left( \frac{x - \mu_D}{\sigma_D \sqrt{2}} \right) \right]
\]

\[
\delta \leq \text{CDF}(\mu_D + \epsilon, \mu_D, \sigma_D) - \text{CDF}(\mu_D - \epsilon, \mu_D, \sigma_D)
\]

\[
\delta \leq \text{erf} \left( \frac{\epsilon}{\sigma_D \sqrt{2}} \right)
\]

\[
\sigma_D \leq \frac{\epsilon}{\sqrt{2} \text{erf}^{-1}(\delta)}
\]
Figure 4.4: Finding an appropriate value of $n_s$. As we increase the number of simulations, we increase the accuracy of our estimates. We set bounds on $n_s$ based on desired accuracy.

Table 4.1: Calculating the number of samples $n_s$ required in the Monte Carlo simulation. $\delta$ is probability of being within $\epsilon$ of expected mean of $D$, implying a standard deviation of $\sigma_D$.

<table>
<thead>
<tr>
<th>$\epsilon$</th>
<th>$\delta$</th>
<th>$\sigma_D$</th>
<th>$n_s$</th>
</tr>
</thead>
<tbody>
<tr>
<td>.01</td>
<td>.95</td>
<td>0.005102</td>
<td>$\sim$ 5000</td>
</tr>
<tr>
<td>.01</td>
<td>.99</td>
<td>0.003882</td>
<td>$\sim$ 9000</td>
</tr>
<tr>
<td>.01</td>
<td>.999</td>
<td>0.003039</td>
<td>$\sim$ 12000</td>
</tr>
<tr>
<td>.005</td>
<td>.95</td>
<td>0.002551</td>
<td>$\sim$ 20000</td>
</tr>
<tr>
<td>.005</td>
<td>.99</td>
<td>0.001941</td>
<td>$\sim$ 30000</td>
</tr>
<tr>
<td>.005</td>
<td>.999</td>
<td>0.001520</td>
<td>$\sim$ 50000</td>
</tr>
</tbody>
</table>

We increase $n_s$ until $\sigma_D$ is under the bounds defined by our accuracy thresholds $\epsilon$ and $\delta$. Table 4.1 shows how varying $\epsilon$ and $\delta$ affects standard deviation and number of simulations.

4.3.4 Anomalies in Flappy Bird

When investigating the score distributions, we noticed two types of anomalies in the distributions: (1) lower difficulty for the first gap and (2) oscillating beat patterns visible in the resulting score distribution.

In some variants, the first gap is easier for the AI than other gaps: the first gap has a lower failure rate (see Figure 4.5). Before any pipes arrive on the screen, the most sensible action is for the AI to hover around the middle of the screen, so at worst the bird only needs to travel half of the gap location range $l_r/2$. But for the rest of the game it can be as much as the full gap location range $l_r$, if a low gap is followed by a high gap. We can ignore the unique first gap and fit the decay to the remaining gaps for a more accurate estimate of $D$. 
Figure 4.5: The first pipe can be significantly easier than the subsequent pipes; this is due to shorter distance to travel and more time for the player to prepare.

Figure 4.6: (a) Player always starting at the same location causes beating pattern in the scores, (b) caused by some pipes having more jump events. We solve by randomizing the starting position on each simulation.

Scoring in *Flappy Bird* is usually measured at the middle of the pipe – the bird only needs to pass through .5 pipes to get a score of 1, but must pass through 1.5 pipes to get a score of 2. We mitigate this by shifting our scoring location to the end of the gap.

Finally, if the bird starts at the same distance from the first pipe during each run, a beating effect can occur in the score distribution. As we see in Figure 4.6a, there is a clear pattern where every other pipe is more difficult. This happens because some times the player must jump twice within a pipe while other times they only need to jump once, as shown in Figure 4.6b. We eliminate this effect by starting the bird at a random $x$ offset on each run of the simulation. This smooths out the beating effect, removing it from the final distribution.
4.3.5 Determining Model Parameters

We estimated the timing parameters using a small user study. The data collection was performed in a web browser; \( N = 20 \) participants were monitored to ensure they complete the entire study correctly. In the study, we measure precision, reaction time, and actions per second. We measure the standard deviation of precision \( \sigma_p \) by asking participants to tap a button when a horizontally moving line aligns with a target (Fig. 4.7). For each participant, we repeat the test 20 times at 3 speeds. The standard deviation of the measured time error for each speed is used in our player model (see Sec. 4.2). We measured precision to range between \( \sigma_p = 35.9 \text{ms} \) for the slowest line speeds and \( \sigma_p = 61.1 \text{ms} \) for highest line speeds, verifying the speed-accuracy trade-off [243].

To measure reaction time, the user is asked to press a button as soon as they see a horizontally moving line appear on the right side of the window. The average delay in time between when the line first shows up and the user presses the button is \( \tau \). We measured this average value as \( \tau = 288 \text{ms} \), which is in line with classical reaction time measurements [89]. We do not currently use reaction time standard deviation as we found the reaction time does not have a large impact on our AI for playable games.

To measure actions per second, the user rapidly presses a button for 10 seconds. The mean was 7.7 actions per second. This is an upper bound, as players can not be expected to perform this fast with any accuracy.

Figure 4.7: Tool for measuring precision, reaction time, and actions per second.

4.4 Conclusions

The framework presented here can be used for modeling many types of action games. However, the player model presented primarily models player accuracy and timing errors.
Therefore, it is mainly suited for action games where difficulty is determined by motor skill, not by strategic decisions.

There is future work to be done on improving the accuracy of our player model. For example, increased time pressure decreases a player’s precision, so precision is not entirely independent from the other game parameters. Future extensions could use a dynamic player model that adjusts accuracy based on the speed at which the player must react to challenges. Our model also currently ignores the difficulty due to path planning, since *Flappy Bird* does not require it.

In the next chapter, we extend our simulation techniques to focus on games that incorporate (1) challenge due to strategic decisions and (2) different kinds of dexterity skills. This requires new modeling and simulation methods.
Chapter 5

Modeling and Simulation of Strategy and Dexterity in Puzzle Games

In this chapter, we use simulation and score distributions to study the impact of strategy and dexterity on video games that require a player to use strategy to decide between multiple moves while requiring dexterity to correctly execute such moves. We run simulation experiments on the popular interactive puzzle games **Puzzle Bobble**, which exhibits dexterity in the form of precise aiming. By modeling dexterity and strategy as separate components, we quantify the effect of each type of difficulty using artificial intelligence agents that make human-like error. Studying strategy and dexterity simultaneously captures nuances and effects that cannot be observed by modeling them independently. We show how these techniques can be used for several applications in game design, including visualizing the expressive range of puzzle difficulty, understanding the effect of scoring systems on move selection, setting score thresholds for rewards, and ordering puzzles by overall difficulty.

The work in this chapter is part of a submitted journal paper for a special issue on AI-assisted game design \[100\]; more details especially on applying our technique to **Tetris** are in the original paper.

5.1 Introduction

Video games have various sources of difficulty that affect the experience of players with different skills in different ways. For example, in **Tetris** \[163\] (probably the most successful video game of all time \[214\]), players use strategy to plan where to place each piece. Strategy errors occur when a player tries to figure out the best move but chooses an inferior move that leads to a lower final score (Figure 5.1a). A player’s dexterity affects their ability to
accurately use game controls to execute a strategic plan. *Dexterity errors* occur when a player does not execute a move correctly (Figure 5.1b); they might want to place a piece in a particular location but fail, perhaps due to time pressure, such as rapidly falling pieces in the *Tetris* example. Both are critical, but often orthogonal, components of the *perceived difficulty* of an interactive puzzle game. By perceived difficulty \[4\], we mean the experience of difficulty of a game for a particular player population, which might be composed of human or artificial intelligence (AI) players of different skill levels.

The combination of these two different types of difficulty sets interactive puzzle games apart from other kinds of games, as shown in Figure 5.2. For example, perfect information

---

Figure 5.1: Examples from *Tetris* how players use strategy to decide between multiple moves and use dexterity to correctly execute such moves.

Figure 5.2: Games require varying amounts of dexterity and strategy. The bubbles indicate rough ranges, and can overlap in practice. Games in *italics* are examples of games that would exist within the categories.
board games like *Chess* require only strategy, whereas simple action games like *Flappy Bird* require only dexterity. Moreover, games like *Mazes* require neither type of skill. Games that incorporate both dexterity and strategy are engaging because they incorporate a finely tuned balance of both kinds of challenge. Eliminating the dexterity component in an interactive puzzle game (e.g. giving *Tetris* players unlimited time to place pieces, or in *Puzzle Bobble* by giving the player a guideline that shows where every fired ball will land) would change the game radically, such that many players would find the game less interesting.

The design and modeling of games that combine strategy and dexterity requires methods different from techniques for games that exhibit only one source of difficulty. Existing work analyzes strategy games and dexterity games separately (see Section 5.1.2). In contrast, we present an approach for simulating and quantifying the interactions between these two types of difficulty – to our knowledge, the first to do so. We integrate strategy and dexterity into a single framework that can quickly measure score distributions for interactive puzzle games, while limiting the amount of expert knowledge and game-specific code for the AI agents to play effectively. Instead of focusing on expert level play, we simulate human-like strategy and dexterity using player modeling and AI agents.

We demonstrate our method in the seminal game *Puzzle Bobble* where dexterity comes from aiming accuracy\(^1\). The resulting analysis of score distributions demonstrates that difficulty is not a single-dimensional characteristic of games. We show that studying strategy and dexterity simultaneously captures nuances and effects that cannot be observed by modeling them independently. We then apply our analysis to various game design problems, such as understanding how different scoring systems affects the choice of moves, setting score thresholds for rewards, and ordering puzzles by overall difficulty.

### 5.1.1 Automated playtesting and Player Modeling

Our methodology uses automated playtesting \([152]\) and an AI model of game players so that we can rapidly explore multiple dimensions of puzzle game design space. As with the previous chapter, we utilize *player modeling* to represent how a human might perform in a game \([200\ 253]\). There are many approaches to player modeling, but we use an open modeling approach that is interpretable, avoids training time, and limits the amount of game-domain-specific expert knowledge. Our goal is to approximate human play – not to emulate it perfectly – so we can quickly learn something about the underlying game system.

Some existing approaches to player modeling may be more accurate, but they come at a cost. With a user study, one can measure how humans estimate difficulty – however these

\(^1\)We additionally demonstrate our method for *Tetris* in \([100]\).
techniques typically require time-consuming playtesting to build player models. Fitting these outputs as a function of puzzle game parameters (e.g. size of board, number of unique pieces, etc.), one can accurately predict the difficulty of specific strategic puzzle games [236]; however, because the models are game specific they are not suited for search based AI-assisted game design [37]. Given human play traces, one can use machine-learning trained player models to predict a wide range of emotions (fun, challenge, frustration, predictability, anxiety, and boredom) [166], to model human-like decision making to play game-theory matrix games [82], to have more human-like behavior in action games [221, 162], or to simulate slower decision making and limited actions per second [114]. However, these approaches require data collection on existing games to create the play traces for analysis.

In this work, we examine how score distributions change when modifying the player populations to contain players with different amounts of strategy and dexterity skill. By comparing the score distributions, we can measure the strategy and dexterity required to successfully play the game. Because we are examining how changes to game design parameters affect game play, we avoid machine learning “black box” models that make opaque predictions. By focusing on game trees and score distributions, we limit the amount of heuristic knowledge required to analyze the game. While our technique is intended to be generic and can be applied to a wide variety of games without encoding many game specific behaviors into the AI agents, the agents do need to act with human-like error making behavior. Thus, our approach requires a decision on how human error will map to move selection. We use forward models to avoid significantly long training periods typically required with reinforcement learning [209].

5.1.2 Existing Methods for Modeling Difficulty

Strategy difficulty in puzzles and games can come from many sources [78], including the size of the search space for the puzzle [155], applications of logic to deduce correct moves [240], previous knowledge about how to solve similar puzzles [54], and the physical and/or visual representation of a puzzle and its accessibility [119, 109]. Difficulty in Sudoku [248] has been modeled as a constraint satisfaction problem, where difficulty is related to a combination of the number of logic inferences required, types of techniques, and the number of dependency steps to determine move accuracy [167]. Sudoku difficulty has also been modeled by simulation as a function of the number of calculations required, guesses by the solver, back trace steps, obvious branches, and uncertain branches [249]. For Sokoban [91], measuring probability-of-failure for genetically evolved AI agents is an effective model for difficulty [14]. The depth of a game, or how much impact learning and
cognitive resources has on the viability of different strategies, can be viewed as a method for measuring puzzle difficulty [122]. In addition to strategic search depth, hint symmetry can be an important factor in measuring puzzle quality [34]. Play traces for puzzle solutions have also been used to rank puzzles by difficulty [9]. However, all this research focuses on games and puzzles that have no dexterity component, only strategy.

Dexterity difficulty in puzzles and games also arise in various forms. Puzzle games will commonly include a time-pressure component that forces players to act quickly and therefore with some error, known as the speed-accuracy trade-off [243]. For action games without strategic difficulty, we model dexterity error by time-shifting button presses by a normal distribution, as explained in Chapter [4] Dexterity difficulty can also be modeled by calculating geometric trajectories and margin of errors for platformer jumps [145]. Note that time pressure can induce strategy errors as well as dexterity errors.

Some existing methods for measuring game difficulty could be used to measure games that incorporate strategy and dexterity. One such approach is to define a puzzle as a series of subgoals, and then to measure the likelihood that a player possesses the abilities required to beat all of the subgoals [11]; however, this approach requires expert knowledge of what types of abilities are needed. To determine which game parameters might lead to more difficult games, a full factorial analysis of multiple game parameters can be used to determine how parameters affect game difficulty; however, this approach is focused on dynamic difficulty adjustment and uses a variety of game-specific (i.e. Pacman-specific) performance metrics instead of focusing on score distributions [71].

One challenge with estimating the difficulty of a puzzle game using AI simulation algorithms is the tendency to predict the future by unfairly gaining foresight into the repeated outcome of pseudo-random generators. For algorithms that rely on a forward model such as Monte Carlo Tree Search (MCTS) [38], the algorithm will learn what will come in the future, even if that information is not truly available to the player at the time they need to make their decision. To avoid relying on future information, a more complicated forward model can simulate multiple future possibilities. The tree can contain multiple possible futures [172, 50] or hidden information can be modeled in the algorithm as information sets [49], but this adds significant time and complexity. This issue can also happen in reinforcement learning: in particular, Atari 2600 games did not have a way to randomly seed a pseudo random number generator, so Atari games are completely predictable given the same inputs. AI algorithms can therefore over-learn a predictable, ungeneralizable response [142]. In our work, we address this problem by designing our game variants to have no randomness or hidden information, and thus the future state is already available to the player.
5.2 Interactive Puzzle Games

Interactive puzzle games are a popular genre of video game, especially on mobile platforms, popularized by Tetris and tile-swapping games [108] such as Bejeweled [111]. This chapter examines a canonical example of the interactive puzzle genre Puzzle Bobble (also known in the US as Bust-a-Move). We define a puzzle as specific list of game rules and game parameters, in which the game parameters includes the ordering of pieces and board starting state. For example, the game rules will define what the buttons do, how pieces move, and what events cause scores to be awarded. The game parameters define exactly how fast each piece moves, which pieces are available, and how many points are scored for an event. In practice, a full game may have many modes and/or unique puzzles included, each with different ordering of pieces, board layouts, speeds, board size, and/or scoring bonuses.

To avoid problems of predicting future hidden information, we restrict our games to have fixed length and no hidden information, so that the AI and human have exactly the same information available. With no randomness in the game definition, the entire order of game pieces is presented to the player. While this restricts our model to focus on game variants that are finite, there exists a large number of puzzle games that have a fixed number of moves, especially in mobile puzzle games where short play sessions are encouraged. In practice, it is not necessary for the pieces to be actually written out for the player, as long as the order is repeatable, because the player can write down or memorize the moves from previous play sessions.

We vary both game and player parameters to estimate the difficulty of a puzzle. We explore game parameters such as the starting board, availability and variety of pieces, size of the board, time available for decisions, and the assignment of scores to in-game events and tasks. Player parameters in our model (described in Sections 5.4 and 5.5) include dexterity skill, strategic skill, move selection heuristics (which models play style), and awareness of their own dexterity (which comes into play when evaluating the riskiness of a move).

5.2.1 Examples of Strategy and Dexterity

This subsection further explains how the same game can require different amounts of strategy and dexterity based on the layout of the puzzle. Figure 5.3 shows two Puzzle Bobble puzzles that require no strategy and varying amounts of dexterity. It is obvious where to shoot the balls, yet aiming is clearly easier in Figure 5.3a than in Figure 5.3b.

Figure 5.4 shows two Tetris puzzles that require different amounts of strategy but the same amount of dexterity. In the easier puzzle (Fig. 5.4a), the player can chose any order to place the pieces; therefore the puzzle can be solved without look-ahead. The harder puzzle
Figure 5.3: Two *Puzzle Bobble* puzzles requiring varying amounts of dexterity but no strategy. It does not matter which order the player fires the balls, but the targets are easier to hit on the left example than the right one.

Figure 5.4: Two *Tetris* puzzles that require varying levels of strategy. (a) The player can place the pieces in any order, so there is no lookahead required. (b) The player must carefully place the pieces in a particular order, requiring more strategic thinking.

Figure 5.5: A *Puzzle Bobble* puzzle requiring both strategy and dexterity. The best strategy is to swap, fire the yellow ball at D (as its easier to hit than B), and then carefully fire the red ball at F to connect the remaining red groups.
(Fig. 5.4b) requires a specific order to score the maximum number of points by clearing 4 rows at one time (in Tetris, point bonuses are given out for clearing more lines at once). The harder puzzle requires planning moves ahead while the easier puzzle does not.

In Figure 5.5, both strategy and dexterity are required to score the maximum number of points. The optimal play to clear the board is to swap the red and yellow balls, then fire yellow at D (since its a larger target than B), then fire the red ball at the location indicated by F to match both the red group on the left and right at the same time. Additionally, note that multiple moves can give the same result; there are many angles that fire the red ball at C.

5.3 Mean Score for Comparing Puzzles

Because our puzzles have perfect information and fixed length, each unique puzzle exhibits a determinable maximum achievable score for a perfect player making no strategy or dexterity errors. In reality, non-perfect players will only occasionally reach this maximum score; the easier it is to achieve this score, the easier the puzzle. Examining the distribution of scores achieved within that puzzle can help determine the perceived difficulty of a puzzle.

For analyzing and comparing puzzles, we will use the mean normalized score metric as described in Section 3.7. To normalize, we divide each score by the maximum achieved score so that scores are normalized between 0 and 1. When comparing two puzzles with normalized scores, the one with the lower mean is expected to be more difficult.

We found the mean score – the expected value of the normalized score probability distribution – is one of the most useful statistics for comparing puzzles, yet there are other metrics that can give useful information. In Fig. 5.6, we demonstrate how small changes to a puzzle can lead to a wide variety in the distribution of these metrics. Beginning with the 3-row puzzle shown in Fig. 2.4, we generated every puzzle that differed from the source by the color change of a single ball paired with every unique queue of 3 balls, for a total of 1566 unique puzzle variants. The puzzle was played by AIs that make strategy and dexterity errors using the algorithm presented in Sec. 5.6. The normalized mean score (Fig. 5.6a) shows the distribution of difficulties from these puzzles. The puzzle space exhibits a range of standard deviations (Fig. 5.6b), indicating that a designer can find games where the scores are tightly grouped and others that exhibit more return-to-skill [63]. There is also a wide diversity in scoring possibilities: the maximum achievable unnormalized score (Fig. 5.6c) varies widely from 4 to 22 while the number of unique scores that might be achieved by the player (Fig. 5.6d) ranges from 2 to 13 different scores, depending on the puzzle.

Every one-ball change in the board is $29 \times 3 = 87$ boards, and every unique queue is 18 queues (ignoring duplicates when a single swap action will make them equivalent) for a total of $18 \times 87 = 1566$ puzzle variants.
Figure 5.6: Evaluating metrics for closely related puzzles shows a wide expressive range.

5.4 Modeling Strategy

Because our model requires strategy to select a move and then dexterity to execute it, we examine the strategy portion first. As in game theory [68], we model strategic move selection with a game tree, with nodes representing states and edges representing possible moves.

5.4.1 Modeling Strategic Error

When modeling strategic thinking, we need a variety of heuristics that simulate how players might think [78, 63]. These heuristics can be deterministic or stochastic, and imperfect heuristics represent some of the assumptions that one makes about how to value a move. For example, when navigating a maze, a player might choose a path that minimizes the Euclidean distance to the exit even though this is an incorrect heuristic for a cleverly designed maze. We also model uncertainty in the value estimation; this represents how players might not always select the optimal move returned by the heuristic due to error in estimation.

We model strategic move selection using an action evaluation function $Q(s, a)$ that estimates the expected reward of taking an action $a$ from state $s$, as in Q-learning [209]. We select the action $a$ that maximizes the expected reward. One reason we chose Puzzle Bobble is because the puzzles are small enough to expand the full game tree, enabling perfect play.
We then model error in the player’s ability to select the $Q$-maximizing move. This represents the difficulty in estimating effects of actions, uncertainty about which move is best, and/or time pressure. Instead of modeling biological constraints to change the $Q$-learning reward function [72], we assume this strategy error is normally distributed, namely $\mathcal{N}(0, \sigma_s)$ added to the $Q$-values returned by the move valuation function. The ability of the player to correctly select the actual best move is reflected in the strategic error standard deviation $\sigma_s$. Higher $\sigma_s$ models a more challenging situation or a player with less ability to select the best move. Because we modify the $Q$-values from the heuristic evaluation, a move with high $Q$ will be selected more often than a move with low $Q$; moves of approximately equal $Q$ will be selected by the randomness rather than the heuristic. For games where some moves are more error prone than others, one can replace $\sigma_s$ with $\sigma_s|m$ to represent that the strategy standard deviation is dependent on the type of move. Our framework supports this, although our experiments use the same $\sigma_s$ for all strategy moves.

Note that this strategy of randomizing the move evaluation and selection does not necessarily make realistic AI opponents. In practice, AI-opponents may intentionally make non-optimal moves that put the player in interesting board states that the human player must reason through to gain an advantage, at which point the AI will resume higher performance [133, 242]. Because we are modeling the player not the opponent, we can avoid this complication.

Although an interactive puzzle may have a high branching factor, in practice many of these moves will lead to the same outcome or equivalent outcomes. For example in our Puzzle Bobble variant, the branching factor is 100 different angles, but many of these angles will clear the same group (see Figure 5.7). To address this, we assign the same strategic value and error estimate to the moves that give the same result. This is done by first identifying which set of actions map to the same future state $s'$ and then only estimating the $Q$ value once for all actions within that set. We then duplicate the same strategy value with randomized error for all actions within the set.

$Q$-value functions are often discussed in reinforcement learning (RL), where an agent learns the expected rewards for each move, for $t$ iterations or episodes [209]. Such algorithms learn their own heuristics to play the game, and can in theory learn how to play most video games (though combining RL with tree-search appears to do better than RL alone [199]). Presumably more training and iterations $t$ will increase the skill of the player, although complicated games often do not converge in practice causing the estimated value-function to diverge without careful attention to the value-networks [235]. The training time can also be quite long, making it often more practical to use the other search-based algorithms that avoid a training phase. This training period is especially a problem when modifying the
scoring system for the game, as the reward values are changing and long training periods are too time consuming for a genetic population search to optimize scoring systems. Therefore we abandoned RL for this research due to the long training times that made it difficult to quickly iterate on puzzle and game parameters, though it has been used by others for these games [213, 190]. Instead, we use forward models and focus on non-game-specific heuristics that can play different games generally.

5.4.2 Player Heuristics

To simulate players with different types of strategy skill, we implement $Q(s, a)$ using a variety of heuristics and algorithms that represent different types of players present in the player population. In practice, a designer will need to select a population of heuristics they feel represents the players that will engage with the game. For example, a casual game will have players that are more likely to play with simple heuristics such as picking the best immediate move; a hardcore puzzle game will have players that tend to look deeper in the tree and may even expand the game tree with pruning to find the optimal solution.

**Greedy $G_n$** - Each move from the current game position is evaluated $n$ plies deep. A move is awarded the highest value discovered from a descendant of the move within $n$ plies. When $n = 1$, this approximates a player who is making quick judgments; this type of player will be easily caught by traps that locally give the player the most points, but do not lead to the global optimum. As $n$ increases, we can simulate players with a better ability to do lookahead. In practice, $n$ will be limited due to human search ability [78], and $G_2$ represents a 2-ply lookahead which we believe models many players.

**Full Tree Search $T$** - This player builds out the entire game tree to find optimal moves. Each move is awarded a value equal to the best value possible in any terminal node that is a descendant of the move. This models a thoughtful player searching for optimal scores.

Figure 5.7: Branching factor is high when examining angles, but lower when looking at positions the ball can be placed.
**Random Moves** \( R \) - This player makes random moves; each node is given an identical value of 1.0 so there is no preference from one move over another. This models a player that does not understand the rules, as even casual players can eliminate obviously bad moves.

In addition to the heuristics above, we require a board evaluation function to determine the value of making particular moves from particular states. In *Puzzle Bobble* points are typically earned on every move, so the value of a board state is simply the number of points earned along the path back to the root of the search.

Note that more complicated functions to better model human-like strategy, such as those discussed in Section 5.1.2 could also be plugged into our framework by using a different \( Q \)-value estimator and strategy error mapping \( \sigma_s | m \). If one cannot expand the entire game tree due to computational limits, then Monte Carlo sampling of the game tree can still give an approximate estimate of the \( Q \)-value.

### 5.5 Modeling Dexterity

In this section, we discuss how to simulate human-like dexterity for AI agents. We discuss how to algorithmically evaluate the amount of dexterity required for a puzzle. We also extract the empirical standard deviation of dexterity error with a small user study.

#### 5.5.1 Simulating Dexterity

We simulate players with differing levels of dexterity by adding random error to the AI’s ability to execute moves. Given a particular move selected by the agent, we modify the accuracy of the move by an error drawn from a normal distribution \( \mathcal{N}(0, \sigma_d) \), with mean 0 and dexterity standard deviation \( \sigma_d \). Larger \( \sigma_d \) reflects lower accuracy; smaller \( \sigma_d \) reflects higher accuracy.

For *Puzzle Bobble* we modify the angle the ball is fired with probability drawn from \( \mathcal{N}(0, \sigma_d) \). Note that \( \sigma_d \) is defined with respect to the game; the values will be different depending on the speed-accuracy trade-off and the mechanics of the skill required.

For a particular game, the dexterity standard deviation \( \sigma_d \) will not always be a constant, because some moves may be more difficult to execute than others. For example, in *Puzzle Bobble*, we found that players are significantly less accurate at executing moves with a bounce off the walls than straight shots with no bounce. Thus, we improve the simulation by using a different value for the standard deviation when a bouncing move will be made. With this extension, for a move \( m \), we can replace \( \sigma_d \) with \( \sigma_d | m \), reflecting that the dexterity standard deviation can change given the type of move.
5.5.2 Determining Accuracy Values

To determine values for $\sigma_d|m$ we performed a small user study for each game. We asked participants to fire a ball at a target at the top of the board and varied parameters such as the starting position of the ball, position of the target, and number of bounces required to hit the target (ranging between 0 and 2). For each condition, we calculated the ideal target angle to perform the task with minimum error. We gave each player 64 different target scenarios. After each shot was fired, we measured the number of bounces, the difference between the ideal angle and the selected angle, and the final collision point. We removed shots when a player didn’t execute the requested number of bounces (e.g. the player shoots straight at the target when they were asked to bounce it). From this data we determined that $\sigma_d|\text{no\_bounce} = 0.0274$ radians ($n = 94, SE = 0.00283$) and $\sigma_d|bou\_\text{c} = 0.0854$ radians ($n = 246, SE = 0.00544$). Factors such as travel distance and $x$-position of the target did not have a significant impact on the error. Using a normality test and qq-plot [75], the data shows that the aiming errors are sufficiently normally distributed. Therefore, using a normal distribution to simulate aiming accuracy error is justified, as we showed in the previous chapter that normal distributions are justified for modeling timing accuracy [92].

5.5.3 Awareness of Accuracy

The final component to modeling dexterity is to incorporate the player’s own awareness of their accuracy, $a_d$, which the player uses for maximizing their chances of successfully performing a move. Given a choice between multiple moves, a player who knows they are not perfectly accurate may choose to take a less risky move to increase their expected value, rather than trying for the maximum possible score. This also occurs when multiple moves can give the same outcome: for example, in Puzzle Bobble to reduce the likelihood of missing, the player can aim for the center of the cluster and avoid error-prone bounces. Awareness $a_d$ has the following meaning: 0.0 means the player is oblivious to the possibility that they could make dexterity errors (maximally aggressive); between 0.0 and 1.0 means the player acts as if they make less errors than they actually do (aggressive); 1.0 is optimizing expected value with full knowledge of one’s dexterity (realistic); $>1.0$ assumes more error than is likely to happen (conservative).

We incorporate dexterity awareness into our simulation by smoothing the move value functions with a truncated fixed-width Gaussian filter [113], with filter standard deviation equal to the player’s dexterity standard deviation for the move $\sigma_d|m$ multiplied by awareness $a_d$. This filter is equivalent to a variable-filter convolution over the space of possible move

---

3For example, they may be trying only to clear a level with a lower “one star” or “bronze” threshold.
executions. When awareness is 0.0, we skip the filtering step. Figure 5.8 uses \( a_d \) to change the standard deviation of the Gaussian filter. Moves requiring more accuracy lose value as \( a_d \) increases. With an awareness of 0.0, many moves have the maximum value and it is not clear which is best to pick; as awareness increases only one action is clearly the best as it is the least risky.

By playing the same puzzle at multiple awareness settings, one can estimate the effect of awareness on the player’s ability to execute their strategy. In Figure 5.9, we show score histograms for perfect strategy and imperfect dexterity on the same puzzle, at 5 awareness settings. For this puzzle, the player maximizes their ability to reach the top score when they aggressively try moves beyond their skill. Maximizing expected score with full knowledge

Figure 5.9: Impact of the awareness parameter. (a) Changing the awareness setting can affect the distribution of scores for players with imperfect dexterity. (b) For this puzzle, the mean score is highest when the awareness is 0.5 (aggressive), but not 0.0 (assuming perfect accuracy), 1.0 (perfectly aware and realistic), or > 1.0 (conservative).
of their skill causes the player to avoid achieving the top score for the puzzle, as that requires one or more low-probability shots. The mean is maximized in this example with $a_d = 0.5$, but this is not a general claim for all puzzles.

This model gives us an empirical quantitative model for demonstrating that players can reach higher average scores by adjusting how aggressive or conservative they are playing. However, this is puzzle dependent: different values of awareness appear to be best depending on the structure of the puzzle. For example, when comparing two puzzles – one puzzle where a small deviation from the best scoring move leads to much lower scoring moves later in the puzzle and one puzzle where a small deviation leads to just slightly worse scoring moves – awareness will have a much higher impact on the former puzzle than the latter. Future work would be valuable to understand this phenomenon in more detail.

5.6 Algorithm

The method for calculating a score distribution for an interactive puzzle is presented in Algorithm 1. The caller defines 1) a puzzle, with game rules and parameters, 2) a population of AI agents that represent the set of players that will experience the puzzle, and 3) the number of times $n$ to repeat the puzzle, which must be large enough to reach a desired level of accuracy [59]. The population greatly determines the outcome of the score distribution.

The bulk of evaluating how a player performs in the game occurs in lines 7-22. Lines 7-8 determine the heuristic value of each unique move from a given state, using one of the heuristics described in Sec. 5.4.2. We identify unique moves so that if there are multiple actions that lead to the same future state, they all receive the same value. These heuristics may approximate the value, or they may determine the true value of a move based on fully expanding the game tree. In Lines 9-12, we add randomness to each of the values – this simulates the player’s uncertainty at picking the strategically correct move. However, moves that are superior are still more likely to be chosen than inferior moves. In line 13, we duplicate the estimated strategy value plus error to the identical moves within the same group. This ensures that large groups do not get picked more often merely because they have more chance to be randomly assigned a larger error value. In lines 14-15, we use the player’s awareness of their dexterity to filter the possible score values they might achieve, as described in Sec. 5.5.3. This has the effect of making more difficult shots less likely to be chosen by risk-averse and low-skilled players. In lines 16-17, we choose the move that the player estimates is the best after adding post-strategy error. If more than one move has the same value, we chose the move in the middle of the longest span of maximum values, which is least likely to result in a dexterity error. Finally, in lines 18-22, we try to execute
Function $\text{BuildPuzzleHistogram}(\text{population}, \text{puzzle}, n)$:

input : $\text{population}$ of AI players, each with strategy and dexterity parameters, move-evaluation heuristic, and dexterity awareness. $\text{puzzle}$ to evaluate. $n$ samples for the histogram.

output : Histogram of simulated scores

$H \leftarrow$ empty histogram

for $n$ times do

/* initialize new player and new puzzle */

$p \leftarrow$ choose player from $\text{population}$ at random

$s \leftarrow$ initial puzzle state

while game state $s$ is not terminal do

/* estimate reward values for unique moves */

$M \leftarrow \text{GetAllUniqueMoves}(s)$

$Q \leftarrow \text{HeuristicRewardValues}(p, M)$

/* simulate strategic uncertainty; worse moves may get higher estimate */

foreach move $m$ in $M$ do

$\sigma_s|m \leftarrow p$’s ability to estimate strategy

$\epsilon_s \sim \mathcal{N}(\mu = 0, \sigma = \sigma_s|m)$

$Q[m] \leftarrow Q[m] + \epsilon_s$

/* copy moves mapping to same board state */

$Q_{\text{dup}} \leftarrow \text{DuplicateIdenticalMoves}(Q, M, s)$

/* player estimates cost of dexterity error */

$a_d \leftarrow p$’s awareness of their dexterity

$Q_X \leftarrow \text{ConvolveMoves}(p, a_d, M, Q_{\text{dup}})$

/* select move with largest expected reward */

$i \leftarrow \text{argmax } Q_X$ (tiebreak in the center of longest span)

$m \leftarrow M[i]$

/* execute move with dexterity error, possibly executing nearby moves */

$\sigma_d|m \leftarrow p$’s dexterity at move $m$

$\epsilon_d \sim \mathcal{N}(\mu = 0, \sigma = \sigma_d|m)$

$i' \leftarrow \text{Round}(i + \epsilon_d)$

$m' \leftarrow M[i']$

$s \leftarrow \text{ExecuteMove}(m', s)$

add final score from $s$ to histogram $H$

return $H$

Algorithm 1. $\text{BuildPuzzleHistogram}$ evaluates a puzzle given a population of players with parameterized strategy and dexterity, returning a histogram of scores experienced by the population
that move, but possibly perform a nearby move by adding Gaussian randomness to the move index \(i\). This assumes that the moves are sorted in an order proportional to position in the game, for example from minimum to maximum fire angle in *Puzzle Bobble*. The dexterity and strategy standard deviations can be dependent on the type of move (e.g. in *Puzzle Bobble* we have more dexterity error when a move requires a bounce off the walls).

### 5.7 Applications of our Method

In this section, we discuss various applications of our method. We discuss techniques for exploring the rich puzzle space spanned by the game design parameters. Small changes in parameters give rise to a wide variety of score distributions. We also extract strategy and dexterity requirements for each game, evaluating the difficulty as a 2D point for visualization.

#### 5.7.1 Exploring Puzzle Space

The rules of a puzzle game create a high-dimensional space called *game space* [92]. A specific instance of spatial puzzle arrangement and the associated parameters defines a point in game space. Having a computer iterate through thousands of game puzzles can lead to interesting designs that may not be found when human designers design a puzzle (though human designed puzzles may still be preferred [35]).

This wide variety in the descriptive metrics shown in Figure 5.6 indicates that puzzle game space is rich, discontinuous, and difficult to search. Therefore using AI-assisted techniques to discover puzzles via computational creativity [244, 131], such as those discussed in Chapter 8, may be helpful. The histograms do not show the entire expressive space, only immediate neighbors from the starting puzzle. For example, a hill climbing approach could generate difficult puzzles. First, one searches neighbors and picks the most difficult puzzle. That puzzle is then used as a seed for a new neighborhood search. The search terminates at a given difficulty threshold, or when the puzzles no longer increase in difficulty.

#### 5.7.2 Determining Strategy and Dexterity Requirements

As shown in earlier sections, a puzzle can be challenging due to strategy, dexterity, or a combination of the two. Here we present a method for finding amount of strategy and dexterity required for a particular puzzle. We first create two populations of AI agents: one with perfect dexterity but imperfect strategy and the other with perfect strategy but imperfect dexterity. Running these two separate populations through our simulation, we obtain two
Figure 5.10: Our algorithm quantifies how much strategy and dexterity is required to solve puzzles. The larger the values on an axis, the more the effect an error has on the final score.

unique histograms. Since normalized mean score decreases with difficulty, we subtract the normalized mean score for each histogram from 1.0. This gives us the amount of skill required, ranging between normalized values of 0 and 1. In turn, the strategy required and dexterity required gives us a 2D coordinate.

Plotting these 2D coordinates gives a quantitative method for creating charts similar to Figure 5.2. For example, Figure 5.10 shows several Puzzle Bobble puzzles analyzed with this two-population method. The first population contains five players with perfect dexterity, but a mixture of heuristics (as defined in Sec. 5.4.2: greedy one-ply $G_1$ search, greedy two-ply $G_2$ search, and full tree search $T$) and strategy standard deviation $\sigma_s$: \{(G_1, \sigma_s = 0.0), (G_1, \sigma_s = 1.0), (G_2, \sigma_s = 0.0), (G_2, \sigma_s = 1.0), (T, \sigma_s = 1.0)\}. The second population contains nine players, all with perfect strategy ($T, \sigma_s = 0.0$), but with each pairwise combination of dexterity standard deviation $\sigma_d \in \{0.5, 1.0, 1.5\}$ and awareness $a_d \in \{0.5, 1.0, 1.5\}$. Each puzzle exhibits a different combination of skill required, as demonstrated by its 2D location on the graph.

### 5.7.3 Evaluating the Effect of Scoring Systems

Designers can modify characteristics of a puzzle by changing how points are allocated. In this subsection, we examine how changes to Puzzle Bobble scoring systems changes the difficulty of games as well as the range of puzzles that can be created. For example, giving bonus points for dropping hanging bubbles and including an exponential chain bonus can potentially reward strategy. This measures the expressive range [202] of a game system.
Figure 5.11: Changing scoring can affect the difficulty of a puzzle game. These histograms show the impact of dexterity and strategy on 1000 randomly generated *Puzzle Bobble* boards.

We generate 1000 random *Puzzle Bobble* puzzles (randomizing the board and the queue) to determine the expressive range of two scoring systems. In Figure 5.11, we calculate the 2D strategy and dexterity requirements for random puzzles and plot them in a hex bin histogram. This tells how likely we are to find puzzles requiring different amounts of each kind of difficulty based on the scoring system. When adding points for dropped balls, there is not much change to the histogram; however, when adding a chain bonus, the range expands significantly, and the mode shifts to the right towards more strategic challenge. The combination of drop points and chain bonus reduces the likelihood of finding a higher dexterity challenge while keeping the wider expressive range. Note that any particular *Puzzle Bobble* puzzle does not necessarily get more difficult just by changing the scoring system. These plots merely show that the puzzle space, which we randomly sample, has different characteristics based on the scoring system.

### 5.7.4 Setting Star/Medal Thresholds

Many puzzle games use a “three-star” system, as popularized by *Angry Birds* [90]. In such a system, three target scores are set (low/one-star/bronze, medium/two-stars/silver, high/three-stars/gold) that the players try to achieve. Often a puzzle can be passed by achieving one
star, but achieving two or three stars can unlock additional levels, give experience points, or reward the player with other types of in-game unlockables. Setting these target scores at appropriate values for the level of difficulty can be time-consuming for game designers.

First, a designer must set a quantile $q$ for each star goal to determine what percentage of trials should fail to reach each goal. For example, a designer may wish for 60% of tries to fail the one-star goal, 80% to fail the two-star goal, and 90% to fail the three-star goal. Harder-to-reach goals will have a higher quantile. We then calculate an empirical cumulative distribution function (ECDF) that tells us the probability that a player will achieve a score of $s$ or less. To calculate the ECDF, we first define $P_i$ to be the cumulative probability for $s_i$, calculated as $P_i = \sum_{j=1}^{i} p_j$. Given the quantile $q$ for the star goal, we find the largest index $g$ where $P_g \leq q$. We then use $s_g$ as the target goal score to expect that $q$ percentage of tries reach the goal score.

Figure 5.12 shows an example of this system on a hypothetical puzzle. Each bar represents a possible normalized score in the puzzle; the height of the bar represents the cumulative probability of a player achieving a score equal or less to this score. We find the scores to be used for one-star ($q = .6$), two-stars ($q = .8$), and three-stars ($q = .9$).

Because the percentages are calculated across the entire player population, some low-skilled players may find it much harder than intended to reach the three-star goals while high-skilled players may find it easier than intended – a drawback of this approach. In addition, if a particular skill or trick is needed to solve a level, the actual score distributions might be different than the estimated ones unless this type of skill learning is modeled.

---

If $q$ is the likelihood that player will fail to reach the goal on an individual try, $1 - q$ is the likelihood that a player will pass the goal on a try.

---

Figure 5.12: Automatically setting star levels using an empirical cumulative distribution function. One star is rewarded for norm. score .50; two stars for .70, and three stars for .94.
5.7.5 Ordering Puzzles

Often when building a puzzle sequence, a designer likely will wish to order the puzzles by increasing difficulty, perhaps assisted by an algorithm [41, 9, 40]. However, with both strategy and dexterity required to solve a puzzle, its not entirely clear which ordering is best. In this section, we propose a simple ordering system and demonstrate ways to order them.

In Figure 5.13 we generated 10 random Puzzle Bobble puzzles (each puzzle is a different 10x3 board with 3 colors, and random ball queue of length 3). In Figure 5.13a, we sort them by strategic order only; because dexterity is ignored, puzzle 1 – the most difficult dexterity puzzle – ends up coming first. In Figure 5.13b, we sort by dexterity only; again this ordering is strange with puzzle 1 having more strategic difficulty than puzzle 2, and puzzles 8 and 9 being more difficult than puzzle 10. In Figure 5.13c, we sort by the Euclidean distance to the origin; this does a good job at ordering the early puzzles as well as the later ones. Alternately, Figure 5.13d uses Manhattan distance (adding dexterity and strategy together), to give an different but believable ordering.

We do not claim that ordering by Euclidean or Manhattan distance will necessarily give an appropriate puzzle order for players, as it relies on our simplified player model. In particular, if certain skills are required to beat certain levels, it may do a poor job at measuring

![Figure 5.13: Ordering puzzles automatically. We generate random puzzles and determine an ordering. Using only (a) strategic difficulty or (b) dexterity difficulty is not suggested. Instead using (c) Euclidean or (d) Manhattan distance to the origin is more believable.](image-url)
these types of thresholds where learning the skill is part of the difficulty of a puzzle. While others have focused on this type of skill-focused difficulty estimation [11, 9, 41, 40], we believe our approach is more easily applied to games in general as it only requires an analysis of the output scores instead of the underlying game mechanics. Thus, this simpler method can help a game designer get a quick sense of what order the puzzles might be ordered, based on the expected skills of the player population. The designer can then explore this ordering with self-play and further playtesting and make modifications as necessary. Nonetheless, skill ordering is an important line of research for improving the accuracy of modeling perceived difficulty, but it requires a much deeper connection to the underlying game system and is therefore less general than score-based approaches.

5.8 Conclusions

We have shown how modeling and measuring the effect of strategy and dexterity can be used for quantitative AI-assisted game design. This two-dimensional analysis of game difficulty allows a designer or system to create games that are tuned for certain types of skill. Using simulation, one can explore the effect of parameter and rules changes on game variants to tune and analyze games, and to better understand games in general. This demonstrates the necessity of modeling different types of difficulty in many kinds of video games.

While we examine the effects of strategy and dexterity as major sources of difficulty for players, it is important to identify that accessibility is an additional dimension that adds to the difficulty of a game. The concept of accessibility comes from bounded rationality theory, which identifies that information can be harder to process based on the way that it is presented and perceived [109]. For example, in tile-swap games such as Bejeweled, much of the challenge arises because it is difficult to quickly see connections on a large board with many different symbols [108]. This becomes even more difficult with time-pressure, such as in the Bejeweled Blitz variant where the player has a limited time per game. Accessibility can come in the form of rules complexity, how easy it is to perceive the game state, and affordances for controlling or communicating in the game. Some games, such as Keep Talking and Nobody Explodes [170], are enjoyable and challenging specifically because they are intentionally inaccessible. Modeling this type difficulty is a promising future step for this branch of research, bringing a third-dimension to the analysis of game difficulty.

Now that we have examined how strategy and dexterity difficulty can be estimated for a game, we proceed in the next chapter to exploring how human players learn and improve while playing a game multiple times. Instead of using simulated game play, we will examine real player data from two popular mobile games.
Chapter 6

Modeling Learning and High Scores

In this chapter, we examine how players learn over repeated plays, and how that affects their ability to set a new high score. We present the analytical and empirical probabilities of a player achieving a single-player high score after playing a series of games. Using analytical probabilities, simulated game data, and actual game score distributions from two popular mobile games Canabalt (Sec. 2.6) and Drop7 (Sec. 2.7), we show the probability of reaching a high score decrease rapidly the more one plays, even when players are learning and improving. We analyze the probability of beating the previous $k$ scores, placing on a leaderboard, completing a streak of consecutively increasing scores, beating the mean score, and introduce a metric called decaying high score. We show players exhibit different types of learning behavior that can be modeled with linear or power-law functions; yet, in many conditions skill improvement happens too slowly to affect the probability of beating one’s high score. The results of this chapter were first published in [98].

6.1 Introduction

High scores – the maximum score that a player has achieved – appear commonly in many video games. For single player games, they are often used as a motivator to encourage players to keep playing a game, and as a metric to demonstrate improvement and mastery [63, 191, 228]. High scores can also add a competitive component to single player games, allowing players to compete by comparing their scores. In this chapter, we focus on high scores when used as a metric for single-player games. Figure 6.1 shows examples of single-player scores achieved by playing a game 1000 times – we show data for both a simulated game [152] and actual player data obtained via game analytics [62]. We show in Chapter 7 that one can draw random scores from a probability distribution to simulate the outcome of a game, so our simulated game here has constant difficulty and thus is represented by an
Figure 6.1: High scores (red line) are rarely set as more games (black dots) are played. (a) Simulated data assuming an exponential score distribution. (b,c) Scores captured from game analytics for Canabalt and Drop7 (Hardcore mode).

An exponential distribution (such as Flappy Bird). Each point in the graph represents the final score achieved, and the red line shows the current high score for each player. Except for the beginning when a player sets high scores often, a new high score is rarely set.

This implies that it becomes increasingly difficult to beat one’s best score. If high scores are used as encouragement to keep playing, game designers should at least be aware that achieving the goal becomes increasingly rare. Therefore, we present and analyze several other metrics that could be used for motivating players with score-based goals: beating the previous score, beating the previous $k$ scores, number of high scores achieved, placing on a Top $m$ Leaderboard, completing a streak of $k$ consecutively increasing scores, and beating the mean score. We also introduce a new metric called “decaying high score” that is a parameterized, designer-tunable metric that becomes easier to reach the more the player fails to reach it. We also present simulated and actual game data recorded from two popular mobile games to verify our results and show where empirical data diverges from our models.

We do not make claims about which metric would be best to include [55], or how to effectively motivate or reward players [81, 27]. Instead, we present several options that a designer might incorporate into their games, armed with a hopefully deeper understanding of how often a player will achieve those goals. We use data-driven player modeling [200, 254] for the goal of helping designers analyze and craft better games [203, 252].

High scores are related to extreme value theory [178], a branch of statistics for predicting the outcome of a maximal event occurring over a specific period of time. Often used to model the likelihood of floods, severe events in financial markets, or the fastest time for runners [61], one could use a generalized extreme value distribution [44] to model the probability of a player beating a chosen score over a given time period. Instead, we focus on a player’s single-player experience and the probability they will achieve a new high score on the next play, based on their current score series – without being concerned what exact value that high score might be.
Scores can also be used to measure player improvement and learning. In the game data we analyze, much of the player improvement happens either so slowly it does not significantly impact the player’s ability to achieve a high score, or happens mainly in the first few plays of the game. After many plays, player improvement is relatively minor and effectively negligible. If there is no learning or slow learning, we can model the scores as independently and identically distributed (i.i.d.), which is considerably simpler to model. Under conditions of rapid learning, we have independently and non-identically distributed data, which is more complicated to model because it involves time-varying probabilities. When possible, we analyze the more general case when a player exhibits learning and we can quantify improvement with a skill acquisition model [121].

6.2 A Short History of High Scores

High scores were originally used as target scores to reward pinball players with prizes for reaching a particular score [184]. Although unofficial high scores were kept by pinball owners to award weekly prizes, Bally started including printed official target score cards to be mounted on its pinball machines in 1932, on games such as 3-Ring Circus (1932) [1]. A patent for an “automatic score totalizer” was filed in 1933, and was used mechanically in Lone Eagle (1933) and electronically in Round ’n’ Round (1936). Kewpie Doll (1960) displays the adjustable “score to beat”, rewarding players automatically with a replay for reaching the target score [2]. Deluxe Batting Champ (1961) records, displays, and gives rewards for the “high score” [3].

The history of high scores for video games begins with Sea Wolf (1976) [160, 212], allowing people to return to the same machine to compete against each other for the single top score; however, the scores were lost when the machine was powered off (or when a player-accessable button was pushed on the front of the machine to reset the score). Star Fire (1979) [65] was the first game to allow players to enter three initials next to their score, also indicating how many coins were used to achieve that score and only storing one score per total number of coins. Later in the same year, Asteroids (1979) [175] introduced the standard Top 10 Leaderboard, also with initials. Defender (1981) [105] included a AAA-battery powered memory to semi-permanently record the eight “All-Time Greatest” for the machine even if the machine was unplugged, as well as non-permanent “Today’s Greatest” score table [60]. Because arcades were public, the focus on high scores was more targeted for

1 http://www.ipdb.org/machine.cgi?id=2543
2 http://www.ipdb.org/machine.cgi?id=1358
3 http://www.ipdb.org/machine.cgi?id=657
multiplayer competition. Twin Galaxies, started in 1981, officially records the worldwide top scores for each game to allow high score competition to expand outside of the local arcade [53].

6.3 Methods

For each of the metrics discussed in this chapter, our general strategy is to first analyze the theoretical probability of a specific event occurring, such as achieving a new high score or beating one’s last score. We can represent the chance that a player achieves a specific final score in a game as a probability distribution, as demonstrated in Chapter 3. We then compare our analytical models to simulated player data and actual player data. Simulated player data provides a base to explore various score distributions as well as the contribution of different theoretical rates of learning. Unless otherwise indicated, the simulations assume exponentially distributed scores, which shows up in games of constant difficulty as we will show in Chapter 7. When modeling learning and improvement, we adjust the probability distribution after each game to increase the simulated player’s expected final score – this is the mean of the distribution.

Actual player data is essential for verifying the accuracy of our models. For player data obtained via game analytics, we use a collection of scores obtained from two independently developed video games. The first dataset uses scores obtained between 2009 and 2012 for Canabalt (described in Section 2.6), a mobile one-button infinite runner action game that focuses on dexterity skill and accurate timing of jumps in a procedurally generated scrolling map. The second data set uses scores obtained between 2009 and 2010 from the Hardcore mode of the original mobile version of Drop7 (described in Section 2.7), a mobile turn-based puzzle game where the player stacks numbered circles making rows and columns that contain the indicated same number of circles; clearing circles requires careful strategic planning as typical in many deep puzzle games. These two games have many differences in their scoring systems, real-time vs turn-based, strategic planning vs dexterity, etc. and therefore provide two unique and independent case studies that show the applicability of our analysis to a wide variety of games. In particular, our analysis does not require knowledge of the underlying game rules or player skill distributions and so is less likely to be affected by the particular games chosen to validate the models.

For each reported score, we assume all scores with the same device ID come from the

---

4Ideally we would desire to examine more games, but there were only two datasets available to us that contained every score achieved, organized by device ID and timestamp. In contrast, datasets obtained from popular analytics sites such as Google Analytics do not provide access to every score achieved – they provide aggregate data over time which would not allow us to perform the type of analysis we use in this chapter.
same player – this is a commonly accepted practice from game analytics of this era in mobile gaming where people typically only had one device. We are not comparing scores between Drop7 and Canabalt, or making any attempt to match up players who may have played both games. We then sort each player’s scores by the timestamp, giving us a series of scores $X_p = x_{p,1}, x_{p,2}, ... x_{p,n_p}$ for each unique player identifier $p$ in the order they achieved them.

Each player $p$ has played a different number of games $n_p$, equal to the length of $X_p$. Because players all have a different number of games and we want to maximize the number of scores we can study, we perform the analysis on the top $N$ players, sorted by decreasing $n_p$. $N$ varies between 200 and 1000, depending on the metric we are examining, and is chosen to (1) make sure the figures can clearly demonstrate the empirical probabilities and (2) so that $n_p > 100$ for enough plays to see trends in the data. For our simulated games, $N = 1000$ and $n_p = 1000$ for all simulated players. Using $k$-fold cross validation, we can show that the models are robust to selecting different subsets of the $N$ games.

We cleaned the Canabalt data before analyzing: the Canabalt game analytic system always sorted any batched offline scores in decreasing order no matter what order they were actually played in, so we threw out any of the players with clearly modified data.

### 6.4 How Scores Change Over Time

As players repeatedly play a game, they become better at it. We can examine the results of players playing the same game multiple times to see how their scores improve on average over time. We examine two models of learning commonly discussed in the skill acquisition literature [121]: linear learning and power-law learning. Linear learning can be modeled by an average score $\mu_{\text{linear}}$ increasing linearly; Power-law learning can be modeled by an average score $\mu_{\text{power}}$ increasing as a negative-exponent power law:

$$
\mu_{\text{linear}}(t) = at + b \\
\mu_{\text{power}}(t) = A - Bt^{-r}
$$

For $\mu_{\text{linear}}$, the average minimum score $b$ increases by $a$ on each game. For $\mu_{\text{power}}$, $A$ is the asymptotic average score, with learning rate $r > 0$ and average minimum score $A - B$.

Figure 6.2 gives examples of learning rates for linear and power law learning. The dotted black lines represent no learning ($b = 0$ for linear learning and $r = 0$ for power law learning); these appear as horizontal lines since the expected score will not change over time when there is no learning. Power law learning slows down as more games are played.

To see how actual game scores improve over repeated plays, we examine the top $N = 500$ players ranked by number of plays. We take the average score of all players achieving a
Figure 6.2: Examples of (a) linear learning and (b) power-law learning on average score. The dotted black lines represent no learning. For (b), $A = 100$ and $B = 40$.

Figure 6.3: Average scores improve as players play more games, due to learning. Best fit least-squares regressions drawn with dashed lines. (a) Simulated exponentially distributed scores with linear learning ($R^2 = 0.967$). (b) Canabalt exhibits approximately linear learning ($R^2 = 0.675$). (c) Simulated scores with a power law learning ($R^2 = 0.914$). (d) Drop7 exhibits approximately power law learning ($R^2 = 0.929$).
score on game \( t \). In Fig. 6.3 we plot the average over all \( N \) players for all game numbers \( t \):

\[
\text{average score on game } t = \frac{1}{N} \sum_{p=1}^{N} x_{p,t}
\]  

(6.2)

Figure 6.3 demonstrates how average scores increase with the number of games played. Comparing the simulated game data on the left with the actual game data on the right, we can see that *Canabalt* exhibits an approximately linear improvement, while *Drop7 Hardcore* exhibits an approximate power-law curve. We fit the models with ordinary least squares regressions, indicated with the dashed lines; because we are fitting mean scores, the central limit theorem justifies assuming normally distributed residuals. We do not have a predictive model for why each game exhibits a specific type of learning, but it is known that different tasks will exhibit different learning rates [121]. We also note the two games are in different game genres and require different types of skill. Based on the learning rates visible in the game data, we aim to model learning into our equations for a better fit.

### 6.5 Probability of Beating the Previous Score

We now calculate the analytical probability of a player beating their previous score on average, without specifying their exact previous score. Let \( X = x_1, x_2, ..., x_n \) be the series of scores that the player has achieved. During design, we do not know what these values will be, so we calculate the probabilities for all players of the game, not any specific player. Let \( f(x, t) \) be the probability of achieving score \( x \) on game number \( t \). Let \( F(x, t) \) be the cumulative distribution function (CDF) of \( f(x, t) \) (the probability of achieving a score < \( x \) on game number \( t \)). Without specifying the previous score, the probability of beating one’s immediately previous score on game \( t \) is the probability of a score \( x \) on game \( t \) times the probability of score < \( x \) on the previous game \( t - 1 \), integrated over all possible scores:

\[
Pr[\text{beat previous score at game } t] = \int_0^\infty F(x, t - 1) f(x, t) dx
\]  

(6.3)

This is not a joint probability, because \( t \) is not a random variable, and we are free to pick any particular pair of games \( t \) and \( t - 1 \). Note that we assume we do not know the particular score the player might achieve on game \( t - 1 \). However, if one does know the previous score \( x_{t-1} \), then the probability of beating the previous score is equal to \( 1 - F(x_{t-1}, t) \).

To evaluate Eq. (6.3) we first model the player improving very slowly over consecutive games and therefore has effectively constant skill, so their score distribution is independently
and identically distributed (i.i.d.) and not dependent on $t$. Thus, we eliminate $t$ so $f(x, t)$ becomes $f(x)$ (the probability of scoring exactly $x$ on the next game) and $F(x, t - 1)$ becomes $F(x)$ (the probability of scoring $< x$ on the previous game). Therefore, $F(x)f(x)$ is the probability of beating the previous score (which must be $< x$) by scoring exactly $x$ on the current game. Definitions and properties of $f(x)$ and $F(x)$ are provided in Chapter 3.

Under this i.i.d. condition, and knowing by definition that $F(\infty) = 1$, $F(0) = 0$, and the first derivative of $F(x)$ is $f(x)$, we simplify Eq. 6.3 to:

$$Pr[\text{beat previous score} \mid \text{no learning}] = \int_0^\infty f(x)F(x)dx = \left. \frac{F(x)^2}{2} \right|_0^\infty = \frac{1}{2} \quad (6.4)$$

This shows the probability of beating one’s previous score is always 1/2, if the scores are i.i.d. This is independent of the underlying score distribution $f(x)$ and the hazard rate that describes how the game increases in difficulty.

One distinct advantage to this formulation is that as we integrate over $x$, each of the distinct probabilities is independent. Thus, we can simply sum them together using the theorem that $Pr(\sum x) = \sum Pr(x)$ if and only if all values of $x$ are independent. The integral calculates over independent probabilities because the integrand is the probability of achieving exactly $x$; two different values of $x$ will not overlap since one can not achieve two different values of $x$ on the current game.

Another possibility instead of using $F(x)f(x)$ is to use $f(x)S(x)$. This would be the probability of achieving a score of exactly $x$ on the previous game, and any score $> x$ on the current game to beat the previous score. Since $S(x) = 1 - F(x)$ we have $f(x)(1 - F(x)) = f(x) - f(x)F(x)$. Doing the integral $\int_0^x (f(x) - f(x)F(x))dx$ is still

![Figure 6.4](image)

Figure 6.4: Varying mean score to model learning for simulated games. (a) Increasing the mean $\mu$ changes the exponential score probability distribution defined by $\lambda_t = 1/\mu$. (b) Linearly increasing $\mu_{\text{exp}}(t)$ reduces $\lambda_t$. (c) Power-law learning model where learning slows over time.
tractable because by definition of a probability distribution \( f(x) \) we know the integral will sum to 1. The \(-f(x)F(x)\) part will sum to \(-1/2\), and we end with \(1 - 1/2 = 1/2\), the same result as if we had examined \(f(x)F(x)\).

If the player is improving over time, we can not eliminate the \(t\) variable, so the simplification of Eq. 6.4 can not apply. To evaluate Eq. 6.3, we need the analytical equations for the underlying distributions \(f(x, t)\) and \(F(x, t)\), which models how the player performs and improves over time. To begin, we first discuss the case where the score distribution is exponential, as happens in games with constant difficulty. The time-varying probability distribution \(f_{\text{exp}}\), cumulative distribution \(F_{\text{exp}}\), and mean score \(\mu_{\text{exp}}\) can be written as:

\[
f_{\text{exp}}(x, t) = \lambda_t e^{-\lambda_t x} \quad F_{\text{exp}}(x, t) = 1 - e^{-\lambda_t x} \quad \mu_{\text{exp}}(t) = 1/\lambda_t \tag{6.5}
\]

Figure 6.4 demonstrates how the decay rate \(\lambda_t\), normally a constant \(\lambda\), is now time varying and thus changes as \(t\) increases. Fig. 6.4a shows how increasing the mean score \(\mu_{\text{exp}}(t)\) affects the exponential probability distributions. As the mean score increases, the distribution flattens out—low scores become less likely and high scores become more likely. The figure also shows how using a (b) linear model and (c) power-law model (as defined in Eq. 6.1) change the mean score and \(\lambda_t\). As a player improves, their mean score \(\mu_{\text{exp}}(t)\) increases; this implies that \(\lambda_t\) decreases as a player improves.

Now, returning to Eq. 6.3, we substitute the equations for time-varying exponential \(f(x, t)\) and \(F(x, t)\) from Eq. 6.5. Using a symbolic integrator such as Mathematica, we get:

\[
Pr[\text{beat prev. score} \mid \text{exponential dist.}] = \int_0^\infty \lambda_t e^{-\lambda_t x} \left(1 - e^{-\lambda_{t-1} x}\right) dx = \frac{\lambda_{t-1}}{\lambda_t + \lambda_{t-1}} \tag{6.6}
\]

We can examine how Eq. 6.6 might change given different learning rates—i.e. different functions of \(\lambda_t\). If we have a minimal amount of learning, then \(\lambda_t \approx \lambda_{t-1}\) and Eq. 6.6 reduces to 1/2, the result of Eq. 6.4. With more significant linear learning, as described in Eq. 6.1 then \(\lambda_t = 1/(at + b)\). In this case, Eq. 6.6 reduces to:

\[
Pr[\text{beat prev. score} \mid \text{exp. dist., linear learning}] = \frac{\frac{1}{a(t-1)+b}}{\frac{1}{at+b} + \frac{1}{a(t-1)+b}} = \frac{1}{2 - \frac{a}{at+b}} \tag{6.7}
\]

We can see that for all \(a, b > 0\), the above equation is always \(> 1/2\), and as \(t \to \infty\), it approaches 1/2. For small values of \(t\), learning increases the probability of a player beating their previous score. However, as the player plays more games, linearly improving learning has diminishing impact on the probability of a player beating their previous score, assuming exponentially distributed scores. We can also see that in the case of no learning, when \(a = 0\),
or limited learning, when \( a \ll b \), the above equation also reduces to \( 1/2 \). Thus, in many conditions, assuming i.i.d. scores is a reasonable approximation.

If the mean score were to increase as a power law, then Eq. 6.6 becomes:

\[
Pr[\text{beat prev. score | exp. dist., power law learning}] = \frac{1 - (B/A)t^{-r}}{2 - (B/A)(t^{-r} + (t - 1)^{-r})}
\]  

(6.8)

As the number of games \( t \to \infty \) or the learning rate \( r \to \infty \), then \( t^{-r} \to 0 \), and the above equation approaches \( 1/2 \). Also, if the learning rate is very small, such that \( r \to 0 \), then we also approach \( 1/2 \). Therefore, a small learning rate or a very large learning rate has little impact on the probability of beating one’s previous score after playing several games.

For both linear learning and power-law learning, we see that player improvement mostly matters in early games, and has relatively little impact as more games are played. The learning effect, in the limit of many games, is approximately zero. Thus, i.i.d. scores can be a reasonable assumption under certain conditions, which allows us to analyze much more complex probabilities in the following sections.

### 6.6 Probability of Reaching a High Score

We now proceed to the case of beating all of one’s previous scores – the condition for achieving a single-player high score. We calculate the analytical probability of achieving a high score on game \( n \). If we know that \( x \) is the high score set on game \( n \), then we must have all the other \( n - 1 \) scores be less than \( x \). The probability of setting a high score on game \( n \) is therefore the probability of setting a score of \( x \) on game \( n \) times all the cumulative probabilities of setting a score \( < x \) on the other \( n - 1 \) games, integrated over all values of \( x \):

\[
Pr[\text{high score on game } n] = \int_0^\infty f(x, n) \prod_{t=1}^{n-1} F(x, t) dx
\]  

(6.9)

In the cases where there is approximately no learning, then the probabilities are not dependent on \( t \) and we replace \( f(x, t) \) with \( f(x) \) and \( F(x, t) \) with \( F(x) \). Given \( F(\infty) = 1, F(0) = 0 \), and the first derivative of \( F(x) \) is \( f(x) \) for all probability distributions, we get:

\[
Pr[\text{high score on game } n \mid \text{no learning}] = \int_0^\infty f(x)F(x)^{n-1} dx = \frac{F(x)^n}{n} \bigg|_0^\infty = \frac{1}{n}
\]  

(6.10)

This result says that the more one plays a game, the less likely they will beat their best score. Given the importance that game designers often give to the high score (e.g. displaying it on
the top of the screen or at the end of each game), this reward becomes increasingly more
difficult for all players to achieve.

Eq. 6.9 can be solved for cases where there is a significant amount of learning, but the
solution is complicated due to non-identically distributed scores. Due to the complexity of
the result, the analytical solution does not provide much insight and we do not present it
here. However, since learning would be expected to make it easier to perform better and
therefore increase the probability of getting a high score, we can assume a simple linear
correction factor $\kappa \geq 1$, which allows us to replace $1/n$ with $\kappa/n$ when fitting data. This
linear correction factor is the simplest modification to make to the model predict high scores
at a better rate than the non-learning model. We can then use a linear regression to solve for
$\kappa$ to see the overall effect that learning causes us to diverge from the standard model. As
before, because we are dealing with mean scores, the central limit theorem justifies the use
of least squares to fit normally distributed residuals.

We now analyze the empirical data containing player improvement. We verify the results
of this section by examining the probability of achieving a high score on each game using
simulated games as well as actual score data, as shown in Figure 6.5. The plots on the left
show the actual probability of achieving a high score after $n$ games, while on the right we
display the same probabilities in log-log plots; a function that follows a power law, such as
$1/x = x^{-1}$, will show up as a straight line in a log-log plot.

For the simulated games in Fig. 6.5a and Fig. 6.5b, with no learning and i.i.d. scores, we
generate a series of $N$ scores for each of $M$ simulated players using a chosen probability
distribution. For each of the series of scores, we calculate when a new single player high
score is achieved. We store the game number when each high score is achieved into a single
histogram, with one bin for each game number. By dividing the histogram bin values by
$M$, each bin $k$ tells us the probability of achieving a high score on game $k$. We then display
these probabilities vs the current game number. We can see that the observed values from
the simulated games closely match the expected distribution, no matter what underlying
chosen score distribution is used. We used $M = 1000$ and $N = 100$ to generate these
figures. Using a least squares linear regression to fit $\kappa/n$ to the exponential data, we find
$\kappa = 0.9983$ with 95% confidence interval (CI) [0.990, 1.006], indicating that our model $1/n$
is a very good fit. Similarly for the uniform data, we fit $\kappa = 0.9927$ (CI [0.984, 1.001]), also
a good fit for the $1/n$ model.

For the plots in Fig. 6.5c and Fig. 6.5d, we use real game data from Canabalt and Drop7
Hardcore mode to generate the same plots. Again, we use the most active 1000 players,
showing the first $N = 100$ scores for each player. For Canabalt, we fit $\kappa = 1.0293$ (CI
[1.014, 1.044]) and for Drop7 Hardcore we fit $\kappa = 1.0682$ (CI [1.018, 1.119]) also indicating
our $1/n$ model only requires a small correction, which can be explained by non-i.i.d scores.

For the plots in Fig. 6.5e and Fig. 6.5f, we generate sets of simulated exponentially distributed scores, but increase the mean value for the exponential distribution by a linear learning model and a power law learning model. We see that this gives rise to a similar departure from the model $1/n$, indicating that learning effects are likely responsible for this type of departure. Linear learning fits a correction of $\kappa = 1.1094$ (CI [1.076, 1.143]) and power law learning fits a smaller correction of $\kappa = 1.0567$ (CI [1.015, 1.099]).

It is a surprising result that learning and improvement has such a small impact on the probability of achieving a high score. This phenomenon largely occurs because in many action games the random factors between achieving a good score and bad score, which we model with $f(x)$, dominate the learning which slowly modifies the increase the mean

Figure 6.5: Probability of achieving a high score after $n$ games decreases as $1/n$. This shows up as a hyperbola on the left and linearly in the log-log plots on the right. (a,b) Simulated games with no learning, no matter the underlying distribution, tightly follow the model. (c,d) For Canabalt and Drop7 Hardcore mode the probabilities slightly diverge. (e,f) Adding learning effects in the simulation have a similar divergence.
score of \( f(x) \). In the beginning, when learning is occurring rapidly, high scores are also being set with high frequency regardless of learning effects. As \( n \) increases, learning slows down making any two games effectively played at the same skill. Certainly, if the score one achieved was perfectly correlated to one’s skill, such that there were no random effects in achieving a score, then high scores would be set much more often (if not every game if learning occurs continuously). For the games we studied in this thesis – and we believe this hypothesis applies to many action games – the game systems are tuned to exhibit a considerable amount of unexpectedness and randomness in the final scores which justify an analysis that relies on probability distributions.

From a design perspective, a designer can increase the probability that a player will achieve the goal of setting a high score by resetting the high score so that \( n \) can not get too large. For example, the designer could use a rolling window and only compare the high score for the last \( k \) games: then the probability of setting a new maximum would be \( 1/k \) (so if the game gave the high score for the last 10 games, the probability of setting it would be 10%). Another approach to limiting \( n \), which is often used in practice, is to reset the high score after an elapsed amount of time (e.g. daily, weekly, monthly, or by season). This also makes it more likely that a player will achieve the reward of setting a high score. For example, Tsum Tsum [132], a popular mobile game, uses a weekly high score, Spelunky [255] has a “daily” mode where a unique map is generated each day and the scores are only presented for that day, and Defender’s “Today’s Greatest” leaderboard is also reset daily.

### 6.7 Number of High Scores Achieved

Given i.i.d. scores, the expected number of high scores set after \( n \) plays is \( H(n) \), the harmonic series sum up to \( 1/n \). It can be shown that \( H(n) \approx \ln(n) + \gamma + O(1/n) \) where \( \gamma \approx 0.5772... \) is the Euler-Mascheroni constant [204]. Thus, the number of high scores achieved after \( n \) plays is \( O(\ln n) \). To prove this, we need to show that \( H(n) \) is the expected number of high scores, assuming i.i.d. scores. The expected value is the sum of probabilities of achieving a high score on each game, so we calculate the probability of achieving a high score after 1 game, 2 games, and so on up to \( n \) games. Given that the probability of achieving a high score after \( N \) games is \( 1/N \), we have \( 1/1 + 1/2 + 1/3 + ... 1/n \), which is the definition of the harmonic series. Thus the expected number of high scores by game \( n \) is:

\[
E[\text{number high scores achieved by game } n] = H(n) = \sum_{k=1}^{n} 1/k \quad (6.11)
\]
Figure 6.6: The expected number of high scores achieved after \( n \) games is approximately \( H(n) \), the harmonic series summing from 1/1 to 1/n. Although actual player data diverges from this, the average actual number of high scores achieved is still within about 2 even after 200 games.

Figure 6.6 shows how simulated data and actual data conforms to the expected number of high scores. For simulated scores with no learning, the number of high scores tracks the expected \( H(n) \) closely. For actual game data, with learning, the expected number of high scores diverges from \( H(n) \). However, the rate at which it increases is very slow – we show in the figure that even after 200 games the divergence is about 2 high scores. Thus, unless the player is improving at a very rapid rate, learning has a negligible effect on the number of high scores achieved.

### 6.8 Leaderboards

Many games report and store more than one high score, often called a top-ten list or a leaderboard. For approximately no learning and i.i.d. scores, we can show that the probability of placing on the leaderboard is related to the size of the leaderboard \( m \) and the number of games \( n \) played.

\[
Pr[\text{placing on leaderboard of size } m \mid \text{no learning}] = \begin{cases} 
1 & \text{if } n \leq m \\
\frac{m}{n} & \text{if } n > m 
\end{cases} \quad (6.12)
\]

We prove this combinatorially. The leaderboard starts empty, so the first \( m \) games are guaranteed placement on the leaderboard. After playing \( n \) times, we have a series of scores \( x_1, x_2, \ldots, x_n \). For \( x_n \) to place on the leaderboard, we need \( x_n \) to be in the top \( m \) scores. However, we do not care about the actual scores for each of the games in the series, only the
relative rankings between them. So we can give each score $x_i$ a rank $r_i$ between 1 and $n$, with 1 signifying the highest score in the series. There are $n!$ possible ways the scores could be ranked. However, we require $r_n \leq m$ because there are $m$ possible ranks that would get the player on the leaderboard. Once we have this condition, there are $(n-1)!$ ways to rank the other games in the series. This gives us $m(n-1)!/n! = m/n$ ways to place on the leaderboard once the leaderboard is full.

Calculating a closed-form analytical solution for the probability under non-i.i.d learning conditions is difficult, because of the many ways of ordering the scores that filled the leaderboard. Figure 6.7 empirically demonstrates the accuracy of the model under various learning conditions and for different sized leaderboards. Lines represent the idealized probability for reaching a leaderboard assuming approximately no learning and i.i.d. scores, while dots indicate the average probability calculated from simulated and actual game play data. Simulated exponentially distributed scores match the model, but with learning, the probability of reaching the leaderboard significantly improves for small values of $n$. This

![Figure 6.7: Leaderboard probabilities for varying leaderboard sizes. Lines represent ideal model assuming i.i.d. scores. Dots indicate averages calculated from simulated and actual game play data. (a) Simulated exponentially distributed scores match the model closely. (b) With learning, the probability of placing improve for small $n$. (c) Canabalt matches the ideal model while (d) Drop7 Hardcore exhibits a significant departure for small $n$.](image-url)
effect decreases as \( n \) gets larger. *Canabalt* matches the ideal model relatively closely while *Drop7* Hardcore exhibits a significant departure early. As \( n \) grows, all games more closely approximate the ideal model of Eq. 6.12.

### 6.9 Probability of Achieving a Streak

A scoring streak or streak occurs when a player beats their previous score on consecutive plays. If we have a series of \( k \) scores \( x_0, x_1, \ldots x_k \) such that \( x_0 < x_1 < \ldots < x_k \), then we have a streak of length \( k \). By this definition, the shortest streak is \( k = 1 \), which occurs when beating one’s previous score. We analyze the probability of achieving a streak of length \( k \), assuming that the scores are i.i.d. This means that the player can not be intentionally trying to score low in order to make it easier to get a streak. Also, the player can not be receiving score bonuses for playing longer. We will show the probability of achieving a streak of length \( k \) in the next \( k \) games is:

\[
Pr[k \text{ streak } | \text{ no learning}] = \frac{1}{(k + 1)!}
\]  

(6.13)

We prove by counting the possible ways that the unique scores \( x_0 \ldots x_k \) could be sorted; there are \((k + 1)!\) possible permutations. Only one of these permutations has a streak of length \( k \), namely the one which is \( x_0 < x_1 < \ldots < x_k \). Thus, the probability is \( 1/(k + 1)! \) of achieving a run of length \( k \), no matter what the underlying score distribution is for the game.

The probability of extending a \( k \)-streak to a \( k + 1 \)-streak, given that the player has already achieved a \( k \)-streak, is \( 1/(k + 1) \). This is the same result of determining the chances of setting a high score for \( k + 1 \) games. In order to extend the streak by 1 game, the most recent score must be larger than all previous \( k \) scores.

Note that we have assumed all the scores \( x_0 \ldots x_k \) are unique. If we have scores that are tied, this breaks the streak. If we wanted to allow tied scores to continue a streak, such that \( x_0 \leq x_1 \leq \ldots \leq x_k \), then the odds would be higher. However, this calculation requires us to estimate the odds of receiving a score more than once. In a score distribution, this becomes more unlikely as more scores become possible to achieve. In the limit of a continuous distribution of scores, a tie score will practically never occur twice.

<table>
<thead>
<tr>
<th>( k ): streak length</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>probability</td>
<td>1/2!</td>
<td>1/3!</td>
<td>1/4!</td>
<td>1/5!</td>
<td>1/6!</td>
<td>1/7!</td>
<td>1/8!</td>
<td>1/9!</td>
<td>1/10!</td>
</tr>
<tr>
<td>0.5</td>
<td>0.1667</td>
<td>0.0417</td>
<td>0.0083</td>
<td>0.0014</td>
<td>0.0002</td>
<td>2.48e-05</td>
<td>2.76e-06</td>
<td>2.76e-07</td>
<td></td>
</tr>
</tbody>
</table>

Table 6.1: Probability of achieving a streak length of \( k \).
Figure 6.8: (a) Expected and observed probability of achieving a streak, using both simulated and real data. The observed probability closely match the expected value of \(1/(k + 1)!\). (b) Log-probability shows a small unexplained departure for Drop7 Hardcore, which is not likely to be due to learning since it does not show up in the simulated learning data.

In Table 6.1, we can see how longer streaks become increasingly more difficult to achieve. 1-streaks happen 50% of the time, but 2-streaks (16.7%), 3-streaks (4.1%), and 4-streaks (.83%) seem most practical for in-game rewards. However, the probability of achieving a 5-streak is an unnoticeable .14% and the probability of achieving a 9-streak is less than 1 in a million.

In Figure 6.8, we verify the probability of achieving a streak with simulated and actual game data. We generate a histogram of score streaks and compare to a model histogram. The model matches the analytically expected probability from Eq. 6.13. Since the bins become exceedingly small as \(k\) grows, we only will use bins up to \(k = 5\). For each game we process 500 players, counting all score streaks into a histogram with one bin for each \(k\)-streak. The model and observed probabilities of achieving a streak are close, using both simulated and real data. Generally, the observed probability closely match the expected value of \(1/(k + 1)!\). However we can see from Fig. 6.8b that the log-probability show a small unexplained departure for Drop7 Hardcore, which is not likely to be due to learning since it does not show up in the simulated learning case. This warrants more investigation, although the effect is very slight as it barely shows up in Fig 6.8a and is also unlikely to be noticed by a player.

6.10 Design Suggestion: Decaying High Scores

We propose a decaying “score to beat” to make a high score easier to set. If \(B\) is the score to beat, and the player surpasses this with a score \(s > B\), then \(s\) becomes the new score to beat. Otherwise, \(B\) is not reached, and we multiply \(B\) by a decay constant \(\delta \leq 1\). After \(k\) games
Figure 6.9: Probability of Achieving a Decaying High Score. (a,b) Decreasing the decay constant increases the probability of achieving a decayed high score; this constant is tunable by the designer. (c,d) Different score distributions react differently to decayed high scores.

where $B$ is not reached, the score to beat will be $B\delta^k$. The equation for the probability is difficult to write because it depends on the underlying score distribution, but we examine it empirically in Figure 6.9. $\delta = 1$ means no decay, so this is the same as beating a high score. However, as $\delta$ decreases, then the probability of beating a decaying high score increases, allowing the game designer to choose the rate at which they want a player to likely reach the decayed high score, as shown in Fig. 6.9a,b. In Fig. 6.9c,d, we fix the decay constant $\delta = .90$ and try different underlying score distributions – this shows that the probabilities are influenced by the distribution $f(x)$ for decaying high scores.

Although we do not analyze it here, a similar method would be to give players a bonus or score multiplier the more games they play. The amount of multiplier is also designer-tunable, and allows the designer to control how often they would expect a player to receive a reward.
6.11 Design Suggestion: Beating the Mean

We now consider how often a player would beat their mean score. The probability of the player beating their mean score, that is surviving past the mean score $\mu$, is $Pr[x \geq \mu] = S(\mu) = 1 - F(\mu)$. Because the mean of a distribution depends on $f(x)$, the value of $S(\mu)$ is distribution-dependent. We analyze the probability of this in various classes of games.

If a game has scores that are exponentially distributed, then we can show that this probability is a constant independent of the difficulty tuning parameter $\lambda$. Starting with the definition of an exponential distribution $f_{\text{exp}}(x) = \lambda e^{-\lambda x}$:

$$S_{\text{exp}}(x) = e^{-\lambda x} \quad \mu_{\text{exp}} = 1/\lambda \quad S_{\text{exp}}(\mu_{\text{exp}}) = e^{-1} \approx 0.3679... \quad (6.14)$$

Therefore, the probability of beating the mean score in a exponential distribution have nothing to do with how difficult the game is. Exponentially distributed scores occur in games that have a repeated challenge with a constant level of difficulty, unchanging as the player progresses. This type of game is discussed in detail in Chapter 7, specifically Section 7.1.

If a game has a linearly increasing difficulty, then it can be modeled using a Rayleigh probability distribution $f_{\text{ray}}(x) = bxe^{-bx^2/2}$ (see Section 7.2):

$$S_{\text{ray}}(x) = e^{-bx^2/2} \quad \mu_{\text{ray}}(x) = \sqrt{\frac{\pi}{2b}} \quad S_{\text{ray}}(\mu_{\text{ray}}) = e^{-\pi/4} \approx 0.4559... \quad (6.15)$$

We see that the probability of beating the mean score for a game with linearly increasing difficulty is independent of the rate at which the game becomes more difficult. The probability of beating the mean for other common probability distributions, such as Weibull or Pareto, depend on the parameters that define the shape of the distributions and do not have constant values independent of the distribution parameters.

6.12 Conclusions

To summarize, we present our analytical and empirical findings in Table 6.2. For designers who wish to include score-based achievements in their single-player games, an understanding of the probability of players achieving those goals can influence the design of those achievements. We also aim to give designers a better understanding of how learning slows down after a relatively small number of games – this means that players could potentially become frustrated at being unable to beat their highest score. To address this, the decaying high scores metric becomes easier to achieve the more a player fails to meet it. Ideally, this
Metric | Learning? | Result
---|---|---
Prob. Beating Previous Score | No | 1/2
Prob. Beating Prev. Score (after \( n \) games) | Yes | \( > 1/2; \) as \( n \to \infty, \to 1/2 \)
Prob. Beating High Score | No | \( 1/n \)
Prob. Beating High Score | Yes | \( \text{as } n \to \infty, \to 1/n \)
Prob. Beating Last \( k \) Scores | No | \( 1/k \)
Number of High Scores Achieved | No | \( H(n) = \sum_{k=1}^{n} 1/k \)
Number of High Scores Achieved | Yes | \( \geq H(n) \)
Prob. Reaching a Top \( m \) Leaderboard | No | \( L(m, n) = \{1 \text{ if } n \leq m\text{ else } m/n\} \)
Prob. Reaching a Top \( m \) Leaderboard | Yes | \( \geq L(m, n) \)
Prob. Making a \( k \)-Streak | No | \( 1/(k + 1)! \)
Prob. Making a \( k \)-Streak | Yes | \( \approx 1/(k + 1)! \)
Prob. of Beating Decaying High Score | Yes/No | Depends on \( f(x) \) and Decay \( \delta \)
Prob. of Beating Mean for Exponential \( f(x) \) | No | \( e^{-1} \approx 0.3679... \)
Prob. of Beating Mean for Rayleigh \( f(x) \) | No | \( e^{-\pi/4} \approx 0.4559... \)

Table 6.2: Summary of analytical and empirical findings from this chapter

would keep a player more interested in playing a game and less likely to quit – they may be improving but because a high score is not reached they may quit in frustration.

In the next chapter, we will explore how the learning models presented in this chapter can be used to model how a game gets less difficult the more it is played. Thus, even though high scores may not be set very often, players can still notice that a game is getting easier for them over time. We will model these effects using survival analysis.
Chapter 7

Survival Analysis of Simulated Games

In this chapter, we will analyze the characteristic of difficulty of two different simulated games by making quantitative difficulty curves. We examine our simple Box-Packing Game for easy, clean evaluation (see Section 2.2) and the more complicated Flappy Bird simulation (see Section 2.1). Both games use the variable timing player model introduced in Chapter 4. We try different versions of the games, varying a single design parameter, and examine the impact on the distributions of scores that result from the simulation. We test versions where the parameters are constant, as well as some that increase as the player progresses. We also vary the modeled skill level of the AI, allowing us to simulate novice player, expert players, and players that learn and improve each time they play. This data is used to create a empirical discrete probability distribution $f(x)$ for each game, and we calculate the empirical discrete hazard $h(x)$ from $f(x)$, as explained in Chapter 3. We then show how each system can also be modeled with a matching theoretical hazard $h(x)$ that leads us to a theoretical probability distribution $f(x)$ that predicts the probabilities generated by our simulation. This chapter combines previous work from [97] and [94], with some new theoretical examples and equations for this dissertation.

7.1 Constant Hazards and the Exponential Distribution

We start by examining games that do not modify their parameters as the game progresses, and therefore can have a constant difficulty if one ignores learning effects. Constant difficulty equates to a constant hazard, since a constant hazard means that the player is equally likely to die at every moment in the game. The original Flappy Bird is an example of this type of game. Craps [245] also exhibits a constant hazard once the point is set: the player is equally likely to roll a 7 on every roll and end their turn.
7.1.1 Analytical Model for Constant Hazard

We begin by theoretically modeling the constant hazard function. Using Eqs. [3.5-3.9]:

\[ h(x) = \lambda \] (7.1)
\[ H(x) = \int_0^x h(u)du = \lambda x \] (7.2)
\[ S(x) = e^{-H(x)} = e^{-\lambda x} \] (7.3)
\[ f(x) = h(x)S(x) = \lambda e^{-\lambda x} \] (7.4)
\[ \log f(x) = \log(\lambda) - \lambda x \] (7.5)

Thus, a constant hazard rate \( \lambda \) leads to an exponential probability distribution \( f(x) \). More difficult games have higher \( \lambda \): a player is more likely to die after each point and has a lower likelihood of reaching a higher score. The cumulative hazard \( H(x) \), also called the risk function, indicates that a player linearly accumulates hazard the longer they play. \( \log f(x) \) is also linear, as exponentials are linear in a logarithmic plot. This makes it easy to determine from data if the underlying generative model is exponential. That is, if we see a line when plotting \( x \) vs \( \log f(x) \), we can be confident that \( f(x) \) is exponential. This can also be used to determine the \( \lambda \) coefficient, using linear regression to determine the coefficients of the line.

In Figure [7.1] we plot the theoretical probability distributions, log probabilities, and hazards for several exponential distributions. The only variable is the hazard rate constant \( \lambda \), where \( \lambda \in \{0.1, 0.2, 0.3, 0.4\} \). Increasing/decreasing the hazard rate increases/decreases the probability of getting a lower score. This makes sense as making a game more difficult should decrease the expected scores. Log probabilities are linear as expected from Eq. [7.5].

Figure 7.1: Theoretical probability distributions for constant hazards. (a) Larger values of \( \lambda \) have a higher probability of getting a lower score. (b) The log probabilities are linear, indicating an exponential distribution. (c) Constant hazard rates indicate constant difficulty.
### 7.1.2 Simulation Data for Constant Hazard

We now examine game simulations with constant difficulty conditions that empirically lead to a constant hazard such that $h(x) = \lambda$. First, we examine simulation data using the *Box-Packing Game* from Section 2.2. Using the score results from the simulated game, the data shown in Figure 7.2 shows empirical evidence of an exponential distribution and constant hazard rate when using a constant belt speed, constant skill level, and ignoring learning effects. In Fig. 7.2a we show probabilities for 4 versions of the game, each with a different belt speed (increasing in speed from black to red). Exponential distributions become linear in log plots, so we can tell from Fig. 7.2b that the data comes from the exponential distribution. The derived hazard rates, shown in Fig. 7.2c, also increase, indicating as expected that faster belts lead to a more difficult game. There is some noise in the hazard function as we are simulating human error using a stochastic process. This noise can be reduced with more iterations of the simulation. The harder games have shorter lines, because its unlikely a player will achieve higher scores in them.

Next, we examine simulation data from *Flappy Bird*. Using the scores from our simulations (Ch. 4), the data shown in Fig. 7.3 shows empirical evidence of an exponential distribution and constant hazard rate when using a constant pipe gap, constant skill level, and ignoring learning effects. In Fig. 7.3a we see probabilities for 4 versions of the game, each with a different pipe gap (decreasing in size from black to red). Exponential distributions become linear in log plots, so we can tell from Fig. 7.3b that the data indeed comes from the exponential distribution. The derived hazard rates, shown in Fig. 7.3c, also increase.

![Figure 7.2: Simulated Box-Packing Game with constant belt speed, skill, and no learning.](image)

(a) Harder games have higher probability of low scores. (b) Log probabilities are linear, indicating an exponential distribution. (c) Constant hazard rates indicate constant difficulty.
Figure 7.3: Simulated Flappy Bird games with constant bird speed, skill, and no learning. (a) Harder games have higher probability of low scores. (b) Log probabilities are linear, indicating exponential distributions. (c) Constant hazard rates indicate constant difficulty.

indicating as expected narrower gaps lead to more difficulty. There is noise in the hazard as we simulate using a stochastic process. This can be reduced with more samples. The harder games have shorter lines, because its unlikely a player will achieve higher scores in them.

7.2 Linear Hazards and the Rayleigh Distribution

Many games do not exhibit constant difficulty, but instead increase in difficulty as the player gets further in to the game. In this section, we examine such games.

7.2.1 Analytical Model for Constant Hazard

We can theoretically model this with a linear hazard function \( h(x) = a + bx \), where \( a \) defines the game’s base difficulty and \( b > 0 \) defines the rate at which difficulty increases. Using Eqs. 3.5, 3.9, we find the theoretical probability distribution:

\[
\begin{align*}
    h(x) &= a + bx \\
    H(x) &= \int_0^x h(u) \, du = ax + \frac{b}{2} x^2 \\
    S(x) &= e^{-H(x)} = e^{-ax - \frac{b}{2} x^2} \\
    f(x) &= h(x)S(x) = (a + bx)e^{-ax - \frac{b}{2} x^2}
\end{align*}
\]

When \( a = 0 \) and \( b = 1/\sigma^2 \), this reduces to the well known one-parameter Rayleigh distribution \( f(x) = \frac{x}{\sigma^2} e^{-x^2/2\sigma^2} \).
Figure 7.4: Theoretical model of linearly increasing hazards \( h(x) = a + bx \). (a,b) Linear increases in \( b \) with same starting value \( a \) cause difficulty to increase, shifting the mode of the distribution to lower scores. (c,d) Changing the starting hazard value \( a \), with the same rate increase \( b \), modifies the shape of the distribution from non-zero mode to zero mode.

In Figure 7.4 we see the effect of the \( a \) and \( b \) linear parameters on the score distribution. On the left column (Fig. 7.4a,c), we see different score probability distributions \( f_1(x) \) and \( f_2(x) \) for two hypothetical games \( G_1 \) and \( G_2 \). On the right column (Fig. 7.4b,d), we see the matching hazard functions \( h_1(x) \) and \( h_2(x) \). Parameters \( a \) and \( b \) have the effect of both shifting the mode of \( f_1 \) and \( f_2 \) as well as the shape of the curve. In particular we can see that for \( a = .30, b = .02 \) in \( f_2(x) \) in the lower row, the curve is always decreasing, indicating that players will be most likely to receive the lowest score, and each increasing score becomes less likely (as in the exponential case). However, for other games, such as when \( a = .15, b = .02 \) in \( f_2(x) \), the mostly likely score is non-zero. This means that the player will be more likely to achieve higher scores at the start, until the game gets more difficult, and then the likelihood starts to decrease.
7.2.2 Simulation Data for Linear Hazard

Figure 7.5 shows four Box-Packing Game variants starting with a shared belt speed and increasing by a fixed amount after each successfully packed box. The black line indicates the smallest increase in speed and the red line is the largest increase. Fig. 7.5a shows the resulting empirical probability distribution for each variant. As expected, games with the slower belt speed increase show a higher likelihood of a higher scores. The initial game difficulty chosen for the experiment shows the nice design property that the player is more likely to achieve a score of around 3 to 5, rather than 0, which means the player will likely experience some small success at the start (unlike the constant hazards in Sec. 7.1).

In Fig. 7.5b we show the derived empirical hazard rate for each variant, which are approximately linear, although there is a slight curve downwards showing that the hazards are not perfectly linear. Each line comes to the same point because each variant starts out with the same belt speed. Increasing the belt speed at a faster rate means the game gets more difficult more quickly, indicated by a steeper slope in the hazard plot.

Figure 7.6 shows four simulated Flappy Bird variants where we start with a common pipe gap and multiply it by $1 - \Delta p_g$ after each consecutive pipe. We include a lower bound on the pipe gap so that it does not get so small that the bird can not pass through it. The data here is noisier than for Figure 7.5 because less samples were used, as Flappy Bird takes longer to simulate than the Box-Packing Game.

Figure 7.5: Simulation Box-Packing Game variants with linearly increasing belt speed, constant skill, and no learning. After each point, the belt speed goes up by speed increase. (a) Slower speed increases are more likely to exhibit higher scores, following a shifted Rayleigh probability distribution. (b) Steeper hazard lines indicate faster difficulty increases.
Figure 7.6: Simulated *Flappy Bird* variants with pipe gap decreasing by the given percentage, constant skill, and no learning. (a) Smaller decreases are easier games and more likely to exhibit higher scores. (b) Steeper hazard lines indicate more rapid difficulty increases.

Fig. 7.6a shows the resulting empirical probability distribution for each *Flappy Bird* variant. Games with the slower pipe gap decreases show a higher probability of a higher scores. The bump in the probability plots indicates that the player is more likely to achieve a score of around 3-5 than a score of 0, which means the player will likely experience some success at the start (unlike the constant hazard example that is equally difficult everywhere).

Fig. 7.6b shows the hazard rate for each variant. The hazards are approximately linear, although there is a slight curve visible in the harder variants. Each line comes to the same point because each variant starts out with the same pipe gap value. Decreasing the pipe gap at a larger rate means the game gets more difficult more quickly, indicated by a steeper hazard slope.

### 7.3 Power-law Hazard and the Weibull Distribution

The Weibull distribution commonly occurs when a manufactured part, such as a light bulb, becomes more likely to fail the longer it is used [179][115]. The Weibull distribution can be used to model games where the rate of difficulty accelerates as the player progresses.

#### 7.3.1 Analytical Model for Power-law Hazard

The three-parameter Weibull distribution is defined in Eqs. [7.10][7.13][179]. The complicated form for $h(x)$ and $f(x)$ permits the cumulative hazard $H(x)$ to be defined simply.
\[ h(x) = \frac{c}{b} \left( \frac{x - a}{b} \right)^{c-1} \]  
\( (7.10) \)

\[ H(x) = \int_0^x h(u)du = \left( \frac{x - a}{b} \right)^c \]  
\( (7.11) \)

\[ S(x) = e^{-H(x)} = e^{-(\frac{x-a}{b})^c} \]  
\( (7.12) \)

\[ f(x) = h(x)S(x) = \frac{c}{b} \left( \frac{x - a}{b} \right)^{c-1} e^{-(\frac{x-a}{b})^c} \]  
\( (7.13) \)

Where \( a \) is a location parameter, \( b \) is a scale parameter, and \( c \) is a shape parameter. In Fig. 7.7 we have several examples of different values of \( c \). When \( c < 1 \) (solid blue line) we have an asymptotically decreasing hazard rate; this can occur when players are developing skills that effectively makes the game easier the more they play. This has a thicker tail than an exponential (dotted black line), which occurs when \( c = 1 \) for the distribution.

When \( c > 1 \), we have an increasing hazard rate. For \( c > 1 \) and \( c < 2 \) we have a decelerating but increasing hazard rate (green dash-dot line). The game gets harder at a power-law increasing rate, but high scores have a less noticeable effect on change in difficulty. When \( c = 2 \) (yellow line) the hazard is linear, and the Weibull distribution is equivalent to the Rayleigh Distribution in Equation 7.9 (but with different meanings for \( a \) and \( b \)). When \( c > 2 \) (red dashed line), the hazard rate is accelerating – the game becomes harder and harder at an increasingly rapid rate.

When \( c = 1 \), Equation 7.13 reduces to \( \frac{1}{b} e^{-\left(\frac{x-a}{b}\right)^c} \). Replacing \( \lambda = \frac{1}{b} \) and \( a = 0 \) we get \( \lambda e^{-\lambda x} \) (Eq. 7.4).

![Figure 7.7](image)

Figure 7.7: Examples of the theoretical Weibull model, with power-law hazards. (a) Varying the shape parameter \( c \). (b) \( c \) controls if the hazard is decreasing, constant, decelerating, linear, or accelerating. (c) Lines in a Weibull plot indicates data matches the Weibull distribution.
A Weibull plot in Figure 7.7c shows the relationship between \( \ln x \) and \( \ln(H(x)) \) (equivalently, \( \ln x \) and \( -\ln S(x) \)). In such a plot, when \( a = 0 \), curves will be linear if the data follows a Weibull distribution \[115\]. The slope of the line is related to the shape parameter \( c \). For small values of \( a \), the Weibull plot will still be approximately linear.

### 7.3.2 Simulation Data for Power-Law Hazard

Instead of multiplying *Flappy Bird’s* pipe gap by a scale factor as in Section 7.2.2, here we increase bird speed by adding a fixed constant. We are using a different variable to show that our methods work for other parameters besides pipe gap. We can see a curving hazard function in Fig. 7.8a. In Figure 7.8b, we use a Weibull plot to show that this data follows the Weibull distribution. We can see that these hazard plots are increasing but decelerating, indicating that \( c > 1 \) and \( c < 2 \). We can expect this decelerating effect because adding a fixed constant changes the pipe gap by a smaller fraction of the total. Of course, if we continue to decrease the pipe gap, at some point the game will be impossible to play. This type of analysis will often be more complicated than following a simple analytical model.

![Graphs](image)

**Figure 7.8**: Simulated *Flappy Bird* games with bird speed increasing by the given *additional amount* after each point, constant skill, and no learning. (a) Hazard is more curved than in Fig. 7.6b. (b) Lines in the Weibull plot indicates data matches Weibull distribution.

### 7.4 Varying Skill Levels with Linear Hazards

We now explore how a game with linear hazard can be experienced by players of different skill levels. We explore this for the *Box-Packing Game* (Fig. 7.9) as well as for *Flappy Bird*
Figure 7.9: Data from simulated Box-Packing Games with linear increasing difficulty. Smaller standard deviation models higher skill. (a) Skill greatly influences the shape of the empirical probability distribution. (b) Skill affects the y-intercept for the hazard rate, which causes the probability distribution in a. to shift and change shape.

(Fig. 7.10). We simulate the same game with linearly increasing difficulty, but use a different player skill for each line. Recall that we increase simulated player skill by decreasing the standard deviation of the time adjustment (Chapter 4).

In Fig. 7.9a we can see that players of low skill experience a very different game from high skilled players. The low skilled player finds that a score of 0 is most likely, meaning they do not experience any positive feedback early on to encourage them. The high skilled player however has some early notion they are doing well as the most likely score for them is around 25, and it is very unlikely they will achieve a score $\leq 5$.

In Fig. 7.9b, we see that the hazard rates derived from the data are still approximately linear as in Figure 7.5, but here the intercept $a$ is changing as well as the slope $b$. Because hazard rates can not be negative, the black hazard line flattens near the origin, while the trend of the line is towards a negative y-intercept $a$. Easier parts of the game are trivial and unlikely to lead to the AI failing, which causes a flat hazard rate. It is not until the line starts turning upward that this AI begins to experience a challenge.

The resulting empirical probability distributions from the Flappy Bird experiment are shown in Fig. 7.10a. We can see that players of low skill may experience a very different game from high skilled players. The low skilled player finds that a score of 0 is most likely; they do not experience any positive feedback early on to encourage them. The high skilled player however has some early notion they are doing well as their probabilities increase at the beginning. In Fig. 7.10b, the hazard rates for this variant are no longer linear for the
higher skilled players as in Fig. 7.6. This occurs because the hazard rate cannot be negative, so the curve trends towards horizontal at lower scores as visible in the data from higher skilled players [97].

When comparing Figures 7.9 and 7.10 one can see that the Box-Packing Game looks more like the theoretical curves in Figure 7.4. This is because the Box-Packing Game is far simpler than Flappy Bird, and exhibits far less anomalies due to the geometry of the game. Because of the shapes of the pipes vs bird, various beating/aliasing effects between the bird jump arcs and the distance between pipes, and the simplicity of the path planner for our Flappy Bird simulator, the hazard curves have unique shapes that are not perfect fits to the theoretical models (see Section 4.3.4 for more information on these anomalies). However, the hazard analysis is still effective at visualizing how the game becomes more difficult for players as their skill changes.

It is important to reiterate that the shape of the score probability distribution and hazard curves are dependent on the player’s skill – the low skilled player and high skilled player do not experience the game in the same way. This quantitatively impacts the designer’s ability to make a game that can please all players without making some sacrifices on game balance [102].
7.5 Modeling Learning

Now that we have seen how players of different skill changes the perceived difficulty of the game, we now apply survival analysis to examine the distribution of scores from real-life (i.e. not simulated) game data. As we will see, this demonstrates that players are learning while they play, which affects the perceived difficulty for more advanced players.

7.5.1 Evidence of Learning in Actual Game Distributions

We examine data obtained from flappybird.io [136], a popular web-based version of Flappy Bird. As explained in Sec. 7.1, Flappy Bird has a constant difficulty, so without learning effects would exhibit a constant hazard rate and an exponential probability distribution.

Figure 7.11 shows the actual score distributions for 4 months from March 2014, when flappybird.io first launched, to June 2014. This time period covers over 175 million individual plays. The spikes at the left edge occur since the first pipe is easier to score due to the original game scoring at the pipe center and additional setup time for the first pipe. In our system, we eliminated this spike by randomizing the starting location and shifting the scoring position to the end of each pipe. It is not apparent from Fig. 7.11a which distribution is occurring, but non-linear log plots in Fig. 7.11b shows it is not exponential.

By deriving the hazard from the data, we see in Fig. 7.11c that the hazard rate decreases rapidly, indicating that learning and past experience may be a factor at making the game less difficult for higher scoring players. Plotting score vs reciprocal hazard rate $1/h(x)$ in Fig. 7.11d shows a linear relationship, indicative of the Generalized Pareto Distribution [126] (which will be described in more detail in the next section).

The hazard rate curves appear to be proportional hazards [115], but show a trend in Fig. 7.11c where later months have lower hazard rates at higher score values. We hypothesize this is due to (1) players having more time to practice and improve and (2) poorly performing players becoming frustrated and exiting the sampling pool, which shifts the probabilities towards more skilled players. Although the difference in the graphs appear slight, because of the high number of samples these effects are significant.

To find an explanation for the hyperbolic inverse-linear hazard rates exhibited in the Flappy Bird real game play data, we will combine survival analysis with learning models from Chapter 6.
Figure 7.11: Actual player data from over 175 million plays of flappybird.io matches a Generalized Pareto distribution. (a) The spike occurs because the first pipe is easier to pass than the rest. (b) Non-linear log probability means the distribution is not exponential. (c) Hazard rates show divergence at higher scores. (d) Divergence more apparent with reciprocal hazards.
7.5.2 Analytical Model for Modeling Learning

We theoretically model the hyperbolic hazard observed in the *Flappy Bird* data under conditions of learning and diverse skill populations using hyperbolic hazards:

\[
\begin{align*}
    h(x) &= a + \frac{b}{x + c} \\
    H(x) &= \int_0^x h(u)du = ax + b \log(1 + x/c) \\
    S(x) &= e^{-H(x)} = e^{-ax}(1 + x/c)^{-b} \\
    f(x) &= h(x)S(x) = \left(a + \frac{b}{x + c}\right)e^{-ax}(1 + x/c)^{-b}
\end{align*}
\]

where \(a\) is related to initial difficulty at the start of the game, \(b\) determines the learning rate, and \(c\) allows us to have scores \(x = 0\) and helps model previous experience. These equations model the Generalized Pareto Distribution [126, 125], commonly used to understand extreme events such as floods and earthquakes [178].

7.5.3 Simulation Data for Modeling Learning

As in Chapter 6, we model players improving over time as they repeatedly play the same game using a power law function:

\[
\epsilon = A + B(n + E)^{-R}
\]

(7.14)

where the standard deviation of error \(\epsilon\) in performing a task decreases as the number of repetitions \(n\) increases, \(A\) defines the best possible time to achieve the task, \(B\) defines the performance on the first trial, \(R > 0\) is the learning rate, and \(E\) represents prior experience [121]. Power law functions model improvement that goes quickly at the beginning, but then slows down as the player learns the easiest ways to improve, but then takes more time to develop the ability to improve at higher level skills.

In our system, we model this improvement by decreasing the standard deviation of the time adjustment after each play. For the *Box-Packing Game*, we simulate 50,000 AI players, each repeating the game 10 times. After each time, the standard deviation is reduced to follow the power law learning model. We vary the learning rate \(R\) to explore learning effects.

We can see the empirical results of modeling learning for the *Box-Packing Game* in Figure 7.12. This is the same game as in Sec. 7.1 with constant belt speed, but now each player has a learning rate \(R\). Black has no learning, and light green has fastest learning. From the empirical distributions in Fig. 7.12, we can not tell exactly what the distribution
Figure 7.12: Simulated Box-Packing Games with constant belt speed when modeling learning. (a) Empirical PDF follows the Generalized Pareto Distribution (GPD). (b) Log plots show faster learning diverges from exponential. (c) Hazard rates are decreasing as the learning rate increases. (d) Reciprocal hazards $1/h(x)$ derived from the data are linear, which fit the GPD.

may be, but the log plot in Fig. 7.12b shows that faster learning rates cause a larger departure from exponential (as exponential curves are lines in a log plot). By deriving the hazard from our data, as shown in Fig. 7.12c, we see for the first time a decreasing hazard rate with increasing score $x$. The decreasing behavior arises because with repeated plays, the player is learning and improving, which makes higher scores easier to obtain. The hazard decreases faster with a higher learning rate, and reduces to a constant hazard for a zero learning rate. We can see in Fig. 7.12d further evidence these curves are hyperbolic (i.e. reciprocal linear) hazard rates, when free parameters for inverting the hazard are set appropriately.

We can see the empirical results of modeling learning in Figure 7.13. To generate the data for Flappy Bird, we simulate 5,000 AI players, each repeating the game 10 times. After each time, the standard deviation is reduced to follow the power law learning. We vary the learning rate $R$ for each test: the black curve has no learning, and lightest green curve has
fastest learning. As for the previous example, from the empirical distributions in Fig. 7.13a, we can not determine the distribution, but the log plot in Fig. 7.13b demonstrates that faster learning rates cause a larger departure from exponential (we can see an upward bend in the lines with most learning).

In Fig. 7.13c, we see the same decreasing hazard rate as the score $x$ increases. Additionally, with higher learning rates we will observe better performance in the data; this helps model that good players may play more games than poor players due to satisfaction and positive feedback. We can interpret the decreasing hazards as the game getting easier when you have players increasing their skill, leading to a higher probability of higher scores. We can see in Fig. 7.13d further evidence these curves are hyperbolic (i.e. reciprocal linear) hazard rates, when free parameters for inverting the hazard are set appropriately.

While the reciprocal hazard is linear in both Fig. 7.11d and Fig. 7.13d, the scales are not. This occurs because the population skill of our model is not the same as the population skill.

![Graphs showing empirical PDF, log probability, hazard, and reciprocal hazard](image)

Figure 7.13: Data from simulated *Flappy Bird* with constant game parameters when modeling learning effects. (a) Empirical PDF follows the Generalized Pareto Distribution (GPD). (b) Log plots show learning increasingly diverges from exponential. (c) Hazard rates shows a decreasing trend. (d) Reciprocal hazards $1/h(x)$ derived from the data are linear, which fit the GPD.
of those players in the data set. Figure 7.11 is the population of players when *Flappy Bird* was most popular, and likely contains many expert players who are continually playing the game due to its popularity and players seeking leaderboard spots. Our skill parameters used in Figure 7.13 were estimated from our user study, and have a homogenous skill profile where every player has the same skill and is learning at the same rate.

### 7.6 Conclusions

This chapter has demonstrated how to generate difficulty curves in a quantitative manner, by examining score distributions using survival analysis. To our knowledge, our work is the first time that quantitative difficulty curves have been modeled with survival analysis for research into game design. We showed how to use survival analysis on both human game play metrics and simulation-based data obtained from AI agents that model human motor skill, cognitive limitations, and learning.

In the next chapter, we examine how to use survival analysis for exploring the space of possible games. This will allow us to discover new games, as well as better understand the characteristics of a game.
Chapter 8

Exploring Game Space and Computational Creativity

In this chapter, we use quantitative techniques to explore the space of possible games. By modifying game parameters, without changing the game rules, we can visualize the effects of game tuning. This technique also allows us to better understand the characteristics of the games being studied. Finally, we can use computational creativity to create novel game variants that have a quantifiably different player experience. This chapter contains parts of several published papers [92, 97, 94], collected here for this chapter.

8.1 Introduction

In Chapter [1], we described how every unique set of game parameter settings creates a new game variant. Most generally, the term game space refers to all the possible games that can exist given a language for defining rules and parameters. We also use the same term game space, when used in a particular context, to refer to games for a particular domain (e.g. card games, dice games, minimal action games, etc.) or for a specific game and its variants (e.g. Chess variants, Poker variants, etc.). Variants can include small adjustments, such as the minor differences between baseball rules in the American League and National League [141], or menu settings of difficulty in a video game. In this thesis, we mainly focus on games where we fix the rules, and then only modify the game parameters. Each game parameter thus defines a dimension in game space.

Exploring game space to find specific settings for an optimal experience is a considerable challenge, and we aim to better understand the relationship between game parameters and player experience. Automatically creating new rules [37, 47, 86, 201, 223] is a related
problem, but parameters alone have a significant impact on game feel [211]: getting Mario’s jump to feel right is more about adjusting parameters than coding accurate physics.

The set of all game variants for a specific game is high-dimensional and often impossible to search exhaustively – imagine adjusting hundreds of independent control knobs to search for the perfect game. We reduce the search space by focusing on game variants that only change parameters, not the larger class of variants that include changes to game rules. We aim to provide quantitative techniques to iteratively search game space that can be use together with qualitative methods that designers rely on such as intuition, experience, and user feedback. Quantitative techniques can help us gain perspective on how games are experienced by new players and explore creative new regions of game space by breaking out of local design optima. Automated playtesting [152, 256] and visualization [229, 237] help with this process, guiding designers to create games best suited to individual skill levels and play styles. We also use this approach as a computational creativity [25, 244, 131] method for generating interesting new game variants within an existing game space.

8.2 Method

Our general approach is shown in Figure 8.1. To explore a new game variant, we select parameters from a valid range of values (Section 8.5.2), generate a level (Section 4.3.1), and simulate playing it using an AI that models human imprecision (Section 4.3.2). We repeat the Generate and Simulate steps (Section 4.3.3) until we have a reliable histogram of scores for the game variant (Section 3.6). We then analyze the histogram using survival analysis to find the hazard rate of the distribution, as described in Chapters 3 and 7. For games of constant difficulty, we will get a constant hazard rate. Larger hazard rates predict a harder game as higher scores are increasingly less likely than lower scores. For games with non-constant hazards, we can examine the average hazard rate, or compare hazard rates at a particular point in the game, for example at the 50% quantile.

For this chapter, we use this method to explore the game space of Flappy Bird (Sec. 2.1). As shown in Chapter 7, this game exhibits constant difficulty, so we can simply examine the hazard rate constant $\lambda$ or the mean score $1/\lambda$ when comparing the difficulty of Flappy Bird variants. Similar techniques could also be used to explore games with two-dimensional difficulty, such as in Chapter 5 by using Euclidean or Manhattan distance to the origin (Sec. 5.7.5). This extracts a single difficulty value from the 2D difficulty space with games that have strategy and dexterity components. However in this chapter, we explore games that only have a dexterity component.
8.3 Exploring Game Space

Efficiently exploring high-dimensional game space is not trivial: an exhaustive search over all parameters will not work. Through intelligent sampling and visualization, we can gain insight about the game (Sec. 8.3.1) or adjust parameters to target a specific difficulty (Sec. 8.3.2).

8.3.1 Sampling and Visualization Methods

We use the following techniques to sample and visualize difficulty changes as we explore different points in game space. We start the search from the game parameters that define the original Flappy Bird. We estimated the original parameters by examining video capture of the game, carefully measuring frame to frame differences [92].

8.3.1.1 Single Dimensional Sampling

Beginning with the original Flappy Bird, we keep each parameter fixed and vary one at a time, exploring along each dimension, and sampling at fixed intervals. Figure 8.2 shows a
plot of pipe gap $p_g$ vs hazard $\lambda$, where the hazard is assumed to be constant (Section 7.1). Each line uses a different value for player standard deviation $\sigma$. Lighter lines in the figure have a higher standard deviation, so the AI makes more errors, and the game is more difficult for the modeled player. As one expects, the model predicts that players with less precision will find the same game more difficult to play, and narrower gaps are harder for everyone.

8.3.1.2 Two-Dimensional Sampling

Varying in two dimensions shows dependent parameters, and can help designers find interesting relationships between dimensions of game space. We visualize these results using dot plots, displaying varying difficulty by the radius and color saturation of each point.

For example, we see in Figure 8.3 that jump velocity $j$ and gravity $g$ are dependent. When gravity is too high or low relative to jump velocity, the bird crashes into the floor or ceiling. In the middle, gravity and jump velocity are balanced, and we see as they increase together, the game gets more difficult – faster reaction times are required as the bird is moving rapidly up and down. Lower values of gravity and jump velocity give the player more time to react and are easier to play. Holes and islands are due to stochastic simulation, and can be reduced with a larger number of simulations $n_s$ (Section 4.3.3).

In Figure 8.4 we see the hyperbolic-like relationship between bird horizontal velocity $v_x$ versus pipe gap location range $l_r$. As the bird moves faster, there is less time to aim for the next pipe gap. As we increase the gap location range, the bird must on average travel further to clear the pipes, so the player requires more time to adjust.

Our best visualization results come from using hexagonal sampling. Evenly spaced rectangular grids, which are easy to implement and work well with standard contour plotting.
Figure 8.3: Sampling game space in two dimensions, jump velocity $j$ vs gravity $g$, shows a narrow band of playable games.

Figure 8.4: Sampling in two dimensions, $l_r$ pipe gap location range vs $v_x$ horizontal bird velocity. High speeds require a lower pipe range, so the player has enough time to react to the varying gap locations.
algorithms, introduce visual artifacts because the sample points are not equidistant from each other. Stratified sampling, which avoids clumping from uniform sampling, ensures that the entire space is well covered and eliminates aliasing artifacts from grid sampling, but was not necessary because we sample at a high enough resolution.

Importance Sampling or Monte Carlo Sampling [83] would allow us to investigate areas of space where difficulty is changing rapidly, to make sure we do more sampling in those areas. After doing a first pass of grid or stratified sampling, we can measure the locations where the difficulty gradient changes most rapidly, and then focus additional stratified sampling in these locations. In practice, because each sampling point only took between 0 and 4 seconds (harder games are faster to simulate because they terminate early), we did not require importance sampling and simply increased the density of our initial sampling.

8.3.1.3 High-Dimensional Sampling

Proceeding to higher dimensions, we sample the entire space by varying all the parameters without keeping any fixed. Latin Hypercube Sampling [206] provides us with well distributed stratified sampling that covers the entire space. This is useful for sampling, but is not useful to visualize beyond two-dimensions.

With high-dimensional sampling, we can then approximate and reconstruct the difficulty function at any point in game space by using Moving Least Squares (MLS) [150], without running the simulation. MLS provides a way to interpolate a function from scattered data points, such that the interpolation globally minimizes the weighted least squares error. Essentially, this is the best fit surface when given a cloud of points, and is used for creating smooth surface models from point clouds when using 3D range scanners. MLS interpolation is useful if the simulation is slow, either offline when searching for good parameters or online during dynamic difficulty adjustment [88] where there is no time to run simulations. We experimented with MLS reconstruction, showing that it performed well, but our simulation ran fast enough we did not need to use it during search or visualizations.

8.3.2 Exploration via Optimization

Global optimization algorithms efficiently search parameter space to find the optima of a function [107]. We use optimization to find the parameters that will give a specific hazard \( h(x) = \lambda_{\text{target}} \) by searching the parameter space to minimize \( (\lambda - \lambda_{\text{target}})^2 \). Because we evaluate \( \lambda \) stochastically, we are optimizing over a noisy function and can not use strategies that rely on differentiable functions and gradients [177]. Differential Evolution [173] is designed for stochastic functions, and the DEoptim optimizer [146] quickly explores game
Differential Evolution Optimization helps us search for a target difficulty. Each point indicates a variant tested. X indicates impossible games. Dot size and color indicates closeness to the target. 280 points searched to find a value within .1% of target $\lambda = .155$.

Finding all games that match a particular difficulty $\lambda$ is more difficult, as most optimization algorithms aim to find a single global optimum. One solution is to use multiobjective evolutionary algorithms [222], which handle multiple optima. Another solution is to partition space using an $n$-dimensional octree, and search with $DEoptim$ for a matching game inside each cell. If a game is found, the cell is subdivided and searched recursively. This approach increases in speed as the cell sizes get smaller, but requires repeated evaluations of the same space as a cell is subdivided. These techniques can be sped up using parallelization [234].

8.4 Validating Generated Content

To validate our ability to discover games of different levels of difficulty, we compare our difficulty predictions with difficulty rankings obtained through human playtesters. Our user study is performed in a web browser; participants are monitored to ensure they complete
the entire study correctly. Our questionnaire asked 20 participants for gender, age, game playing experience, and exposure to Flappy Bird games and its clones.

We asked each participant to play 7 pairs of game variants and to rate the difficulty of each game relative to the other in the pair (see Table 8.1). For example, variant 1A is compared with 1B, 2A is compared with 2B, etc. We also measure the mean score each player achieves on each variant. The user can switch between variants in each pair and can play each variant as many times as they want. For each pair, they are asked to compare the games on a 7-point scale: “A is {much easier, easier, a little easier, the same, a little harder, harder, much harder} than B.”

To create the pairs, we used the techniques of Section 8.3.2 to generate 6 unique variants with varying hazard constant $\lambda \in \{0, 0.105, 0.223, 0.357, 0.511, 0.693\}$. If $\lambda(A) < \lambda(B)$, then we predict that $A$ is easier than $B$. To limit the number of changing variables, we only changed gravity $g$ and jump velocity $j$ while fixing all other parameters. Gravity and jump velocity are interesting variables since they need to be experienced in-game to understand how the variant feels, and they would normally require design iteration and playtesting to set correctly.

For each pair, we compare our prediction with what each participant rated more difficult, and their actual performance on each variant, as shown in Table 8.1. We note if the participant (a) agrees or disagrees with our algorithm’s prediction and (b) achieves a higher mean score on the variant predicted to be easier. Since every pair has a different predicted difficulty, if a user indicates “same” we say they disagree with our prediction. We tested on 20 users.

![Table 8.1](image.png)

Table 8.1: Our predictions generally agree with players’ evaluation of perceived difficulty and with their actual scores. Only in one case (Pair 3) do we disagree, and in that situation, the players perceive the pair oppositely to their own performance.

---

1These numbers come from an alternate formulation for difficulty that we used in [92]. In that paper, we measured difficulty as $d = 1 - e^{-\lambda}$, and used values of $d \in \{0, .1, .2, .3, .4, .5\}$. Thus, for this thesis we convert from $d$ to $\lambda$ using $\lambda = -\log(1 - d)$. 
and found our population agreed with our prediction 77.9% of the time when asked, and agreed with our prediction 79.3% when comparing mean scores. Both are highly significant under a binomial test. However, some pairs have more agreement than others: in particular, Pairs 1 & 6 have strong agreement, and we agreed with the majority in all cases except one. Pair 3 stands out as participants performed worse on the variant they perceived to be easier; more investigation is needed as our model provides no explanation for this behavior.

8.5 Computational Creativity

Quantitative methods have a great potential to help designers try new ideas and explore new game variants. In this section, we examine a method for computationally discovering diverse playable game variants, using a fine-tuned exploration of game space to explore a game’s design [93]. Using a parameterized implementation of the popular mobile game Flappy Bird, we vary its parameters to create unique and interesting game variants. An evolutionary algorithm is used to find game variants in the playable space that are as far apart from each other as possible, helping find the most unique versions; we also use a clustering algorithm to find representative variants. Manual playtesting confirms that the discovered game variants are playable and significantly different to the original game in challenge, game feel, and theme.

The process of exploratory computational creativity [25,244,131] includes searching the conceptual space of possible creations to find viable and interesting games. A game is composed of many parts, including visuals, audio, narrative, systems of rules, level architecture, game play, and interactions between those facets [131]. In this section, we focus on the parameters that tune gameplay. As with the rest of this chapter, we explore game space defined by only changing game parameters, but not changing or evolving game rules or mechanics [154,37,47].

By changing parameters alone, we can find surprising and interesting game variants, such as our computationally discovered Frisbee Bird [92], shown in Figure 8.6. This variant was unexpectedly created when searching Flappy Bird game space for variants of a specific difficulty. In this variant, the player is wide and flat, and smoothly sails through the air like a flying disc. We had not expected that a game of this type was in the same game space as Flappy Bird, which encouraged us to develop an algorithm to discover more unique and surprising game variants.

In Fig. 8.7, we show variants of Flappy Bird created by exploring only the 2D game space defined by player width and height. Each game has a different difficulty, and we would expect Tiny Bird to be the easiest because the bird has more room to maneuver between...
Figure 8.6: *Frisbee Bird* was computationally discovered while searching for *Flappy Bird* variants. Its unique location in game space leads to a different game feel and appearance.

Figure 8.7: 2D game space of *Flappy Bird* variants with different player width and height. The only difference is the shape of the player, yet we can distinguish these as unique games.

pipes. There are upper bounds to the parameters in this example game space: no player can score points if the bird is so tall it can not fit through the pipe gap. It is notable that by only changing the player size parameters, we can ascribe a different theme and name to each.

### 8.5.1 Discovering Games Procedurally

We now present our method for finding the most unique variants in game space. The first step is to separate the games that are playable from those that are unplayable: in an unplayable game, every player will die at or before a specific event in the game, no matter how good the players are. These unplayable games can have artistic merit, for example those intentionally unbeatable games that are trying to convey a message about futility of a political situation via procedural rhetoric [26]. However in this chapter, we are generally interested in scored action games in which an excellent player, with enough practice and patience, will rise to
the top of the leaderboard. This is clearly not the entire space of interesting games, but it is a large and popular one, and provides us with a simpler domain to explore.

Note that target player could be a human or an artificial intelligence. We do not make a judgment on what kind of player is playing the game, just that the game is interesting and fun to that player because through learning it can improve [118]. The difficulty will be significantly different depending on the skill of the player, and we do not expect that games interesting for AIs [224, 193] will be the same as those interesting for humans, just as many games interesting for children (i.e. Candy Land [3]) are not interesting for adults [99].

As shown in Chapters [5] and [7] it is important to consider the target player when evaluating game space to find playable and unplayable games. The skill, experience, and learning ability of the player has a major impact on their performance. Our algorithm uses a tunable parameter for skill, which we keep fixed throughout the experiments, but for consistency and simplicity, ignores past experience and learning effects. We also need to constrain the time that it takes to play an average game. Although some parameter settings create playable games, they would take too long to play and are therefore not interesting to the player (e.g. a game that takes a year to score a single point).

For every point in game space we wish to test, we generate a unique game and simulate playing it, having an AI repeatedly play thousands of times to estimate the difficulty and the average time it takes to play (see Chapter [4]).

8.5.2 Bounding the Search Space

The search domain can be arbitrarily large, but as we are focused on games that are playable for humans, we begin by creating reasonable, but artificial, bounds on the parameter space to explore. Although unbounded creativity could lead to new games, the extra effort required to search the larger space could mean that many computational resources are wasted on searching fruitlessly. Thus, we aim to bound our search to maximize the chance that the computer will find playable games, while minimizing the chance that the computer will miss playable games within the domain.

Many of our parameters have natural upper and lower bounds: for example the gap between the top and bottom pipe must be positive because a negative pipe gap means the pipes would overlap, and it must be less than the world height because a pipe gap extending beyond the game bounds is also nonsensical. Some only have a natural lower bound: horizontal pipe separation must be \( \geq 0 \) so that the pipes do not overlap, but there is no upper bound how far apart the pipes can be. This technically does not have an upper bound, as one could make them millions of pixels apart so that it takes hours to travel between
pipes. However, by ensuring an average or maximum play time for a playable game, we can eliminate these extreme variants. Horizontal speed has an effective upper limit, determined by both the geometry of the level and the skill of the target player, but this can only be discovered through simulation or playtesting. Finally, some parameters have no required bounds: for example, gravity could be positive or negative and a sensible game would arise, but jump velocity must be in the opposite direction from gravity for the game to be playable.

We begin by seeding reasonable bounds and then generating samples using a stratified sampling scheme that covers the search space without clumping that may occur with uniform, random sampling. With a sufficiently large number of samples, a uniform sampling can be used instead to avoid the extra computation of stratified sampling.

Using the methods described in the previous section, we can simulate each point and calculate its difficulty and average play time. We mark as playable all the games that have a difficulty where between 1% and 60% of players are expected to crash on the first pipe, and the average human play time is between 1 sec and 30 sec. We eliminate games with a difficulty < 1% as they are too easy and would likely be boring for a player. The playtime is calculated as if the game were played by a human, not as the time it takes for the computer to simulate a game (< 1ms).

We now calculate the hypervolume bounding box that tightly contains the playable games. For each dimension, we compare the effective upper and lower bounds that lead to playable games to the original search range. If the effective bounds are near the guessed bounds, we could, if desired, expand the search area and try again because it is possible we have missed sampling some playable games.

Instead of setting bounds, we could create a probability density to estimate the likelihood of discovering a playable game within a parameter space. This is similar to how genetic algorithms work to improve their search domain over successive generations. Novelty Search can also be used to guess the next best location to search [129].

### 8.5.3 Sampling Games in Parallel

The process of simulating the games is the most expensive part of the process, and it can be run in parallel, speeding up the sampling process almost linearly. Because simulating a game tens of thousands of times takes on the order of 250-1000ms, the dominant amount of time is spent using the CPU to simulate the game, not in spinning up jobs or assigning tasks.

We first generate a table of sample points in the host process, then use the mclapply function in R [174] to run the simulation across multiple cores. This function assigns the list of points to each core at the start, forks the host process, and then runs the simulations in
parallel, letting each core execute its list of games to test. When each core process finishes its list, it terminates and joins with the host process. This efficiently amortizes the process forking and setup cost across each of the games. One downside to this process is that some cores may finish processing before others, due to the random distribution of game difficulty. Because simulation time is linear in the number of frames that the AI survives, easier games take longer to simulate than harder games. So a core that happens to be assigned harder games will finish before the other cores.

An alternate method would be to let each process work on one game point at a time (or a small batch of games at a time), then ask the host for a new sample (or batch) each time it completes one. This keeps each core active, and all cores will finish at about the same time. However, we found this was much slower in practice due to the expense of process startup and repeated communication with the host process.

### 8.5.4 Playing Unexpected Games

As the difficulty and playability of the game is determined by the AI, it will only find games that it is equipped to play. However, there may be games interesting to players of human intelligence that are ignored by the AI’s deficiencies. The method of search can influence which points in conceptual space are discovered [244].

When designing the AI to play the original game, we made assumptions that ended up being violated when exploring new parts of the game space. Specifically, we assumed that gravity would always point down, and jump would always point up, so the AI heuristic focused on jumping when going below a certain target height. Thus, the original AI found inverted gravity games impossible to play, eliminating entire classes of playable games that

![Figure 8.8: Divey Duck](image)

Figure 8.8: *Divey Duck*, a variant with negative gravity and jump, simulates an underwater duck that naturally floats. However, the original AI couldn’t handle negative gravity. Without supporting both types of gravity, this class of games would be labeled unplayable.
we might find interesting. By changing the AI heuristics to be smart enough to handle both
types of gravity, we were then able to find games like *Divey Duck*, shown in Figure 8.8.

There are two insights to be gained here. Firstly, the AI used to simulate play determines
the types of games that are discovered [244]. Secondly, using more sophisticated AIs can be
important even when a simpler AI works fine for the original game. Regarding better AIs,
an A* or Monte Carlo Tree Search algorithm would likely be sufficient for solving *Flappy
Bird* like games. Alternatively, an reinforcement learning AI that can learn over repeated
plays of a video game could learn how to effectively play inverted gravity [209].

### 8.5.5 Finding Unique Games

Now that we have a point cloud of thousands of playable games variants, we seek a small
number of unique games for inspiration and further tuning. There are three approaches we
tried to narrow down this number: Clustering, Convex Hulls, and Genetic Optimization.

#### 8.5.5.1 Finding Representative Games with Clustering

Our first approach is to perform a $k$-clustering on the playable game point cloud, which will
find groups of similar games, where each medoid is the representative of one cluster. We
call these *representative games* because they represent an entire cluster of games.

First, one must determine what value of $k$ to use. This could be determined a priori, and
then defines how many representative games will be returned. Alternately, one can use a
method that determines the ideal $k$ by trying repeated clusterings and stopping when finding
the $k$ that gives the optimum average silhouette width [186]. In practice, we use the `pamk`
function in R [84][174] to find an optimal value of $k$. One also typically sets a maximum
value of $k$, which is especially important for clustering high dimensional spaces.

Second, since our parameters have different units, it is important to normalize the
parameter space so that different variables can be compared. For example, if gravity
ranges between 1 and 4000, while pipe gap varies between 10 and 30, the clustering on
unnormalized space would incorrectly suggest that changes in gravity are far more important
than changes in pipe gap. By normalizing each parameter between 0 and 1, we can compare
different parameters using Euclidean distance. We normalize after finding the playable
games, such that for each parameter 0 is the minimum and 1 is the maximum playable value.

As we can see in Figure 8.9, the clustering method avoids games on the frontiers
of playable games, because the medoids are at the center of each cluster. Although the
clustering works in $n$-dimensional space, for visualization we have taken 204 playable
games and projected them onto two different 2D planes, one for gravity vs jump velocity,
8.5.5.2 Finding Extreme Games With Convex Hulls

Since clustering finds interior points, we are also interested in finding games that are further apart, showing us what is possible on the frontiers of game space. Games on the frontier are not guaranteed to be the most interesting, but they push the parameters to their extremes.

The easiest way to look at extreme games is to find the games with minimal and maximal values for each dimension. If we have $n$ parameters, this will give us $2^n$ games to explore. These lie on each of the boundaries of the hypervolume that contains all playable games.

Next, we can try to find representative games that lie on the $n$-dimensional convex hull, using the \texttt{convhulln} function in R \cite{80}. A convex hull can often have too many points for our purposes, or too many points close together, so the hull should be decimated or clustered to find a smaller number of representative points.

We show in Figure 8.10 two examples of using a 2D convex hull to find the extreme parameter values for playable games. As we are using a hull, this method only finds points on the exterior, as opposed to the clustering method that tends to only find points on the interior. The right image shows a concern with using the convex hull when there are outliers: we can see that the outlier moves the convex hull far from the rest of the points, so a large number of games that would normally be on the boundary are skipped. An outlier reduction could be used, or another method of finding the boundary points that uses windowing to allow some concavity.

Figure 8.9: Searching 2D game space to find $k$ representative games using clustering. Blue dots are medoids calculated for $k = 5$. Red triangles are medoids calculated when using a dynamic clustering that finds an optimal value of $k$.

and one for pipe location randomness vs player speed. Blue dots indicate cluster medoids for $k = 5$, while red triangles indicate medoids selected when searching for an optimal $k$. 
Most Unique Variants via Genetic Optimization

The convex hull method finds frontier games, but not interior games. Clustering finds interior games, but not frontier games. We now find games in both the interior and exterior of the cloud, using genetic optimization to find the most unique variants.

To find games as far apart from each other as possible, we find a subset of $k$ games that maximizes the minimum Euclidean distance (in normalized parameter space) between any pair of points in the set. We use a genetic search to evolve the $k$ elements to be included in the set. For this search, we use the DEoptim Differential Evolution library in R [146].

In Figure 8.11 we can see that representative points are now found on the frontier and interior. These points give a good quality coverage. This method does a much better job at finding games on the exterior, since we expect to select points at the fringe of the point cloud (which is furthest from the center of the cloud). By running for more generations, the examples will get closer to optimally unique.

Figure 8.10: Using the convex hull to find extreme games. Blue dots are on the convex hull, while red triangles are $k = 5$ clustering of blue points, to find a smaller set on the hull.

Figure 8.11: Using genetic optimization, we find games that are far apart from each other in 2D game space. Blue dots indicate the set for $k = 8$ and red triangles are for $k = 5$. 
8.5.7 Results and Discussion

We now present several novel Flappy Bird variants discovered by our methods. Varying all nine game parameters, we asked both algorithms to create 4 unique games. The “Most Unique” method generated four highly different games. We have named them Needle Gnat, Lazy Blimp, Droppy Brick, and Pogo Pigeon [93] and show examples of them in Figure 8.12. These appear to us to approximate the creativity a human designer might have used if tasked with designing four unique variants. The Clustering method generated four versions that do appear closer to each other as expected, as shown in Figure 8.13. Due to their similarity we have difficulty giving them unique names: this is a metric that could use further investigation.

Game parameters do not necessarily have linear responses, so measuring in Euclidean space can be misleading. That is, games with pipe gap of 10 and 15 (50% difference) are more different for a player’s experience than for a pair of games with pipe gap 30 and 45, with a larger absolute change and the same relative change.

A perceptual metric is desirable, so we can compare games from the player’s perspective, and find the most unique games in perceptual space, instead of parameter space. This metric could also tell us when games are perceptually similar, reducing the search resolution required. Given the relationship between some of these parameters – for example, we could scale all of the distance metrics by the same amount – it also seems possible that there could be games that are ostensibly far apart in parameter space, but actually appear very similar in perceptual space. This would probably become more likely the larger the parameter space. One could also develop metrics for game similarity based on simulation, where games eliciting similar playing behavior (or learned behavior) from the AI are judged to be similar.

Our methods presented here are focused on uniqueness, which is one aspect that makes games interesting to players and designers, but “interesting” has a much broader meaning and significantly more complex to measure. Future work on developing metrics that can quantitatively measure expected human interest are highly desirable for creative exploration and will require a deeper understanding of player behavior.

Our system is related to novelty search [127] in that we are looking for novel examples. However, our approach is somewhat different in that we do not have any goal in mind to determine when the search is complete. We merely wish to downsample game space to a small number of samples that are unique from each other. Clearly they are related, in that novelty search tries to search spaces that are distant in parameter space, but our goal is finding a set of globally novel points, not to find a particular goal.
Figure 8.12: The four game variants discovered using the Most Unique evolution method with $k = 4$. This method searches for the $k$ games that maximize the minimum distance between any two points in the set. The games are generated by the algorithm; the names are provided by the authors. (a) Needle Gnat: tiny player trying to thread a tight horizontal space. (b) Lazy Blimp: slow moving blimp-like player with minimal gravity and jump. (c) Droppy Brick: frequent rise and fall with high gravity. (d) Pogo Pigeon: very tall, thin bird that frequently hops to avoid crashing into the ground.
8.6 Conclusions

In theory, we expect high-dimensional game space to have dependencies that can be squeezed into a lower dimensional space using model reduction techniques, finding the intrinsic dimensionality of a game space. This would reduce the number of knobs a designer needs to adjust, assuming that there is some lower dimensional iso-manifold of estimated difficulty. In addition, lower dimensional spaces are faster and easier to search.

Setting game parameters can be assisted by algorithms, and take us towards more practical “computer-aided game design” or “AI-assisted game design”: designers and computers working together to craft a better player experience. We look forward to more creative output both from algorithms generating new game variants, but even more so from teams of humans and computers working together to create better games and inspiring future game designers.

We now progress to analyzing games that are not single-player, but are two-player games. This requires a different set of techniques to analyze two-player games and their characteristics.
Chapter 9

Exploring Characteristics of Combinatorial Games

Up to now, we have been examining single player games. For the next two chapters, we examine score distributions in two-player games. In this chapter, we examine how score distributions within a game – as opposed to examining the final score distributions, as discussed in other chapters – can affect the potential quality of a game. In particular, we examine how the score difference between two players can be used as a mechanic in the novel combinatorial game Catch-Up [96]. We analyze the game for interesting game properties, and we perform an analysis of game strategies using simple AI agents that model novice play. In addition, we use genetic programming, an AI technique that can generate novel programs, to evolve new strategies for the game. The presence of interesting strategies for novice-level AIs that simulate novice-level human play can be a useful measurement to determine if a game may be interesting to play. The work in this chapter has been previously published [96], with the notable exception that the work on genetically evolving heuristics for Catch-Up is has not been published other than in this thesis.

9.1 Introduction

It is a challenge to design interesting two-player games with simple rules that keep the score close, even between players of different skill. When the game score is close, players experience tension and drama by not knowing too far in advance who will win. This drama has been discussed qualitatively [63, 218] and quantitatively [37].

Economists describe the desire to minimize inequality as inequity aversion, wherein people prefer rewards to be allocated evenly [67]. Designing games with inequity aversion
can create a more balanced competitive experience, allowing experts and novices to enjoy playing together as the score will remain close throughout the game. To enhance tension, games often have catch-up mechanisms, sometimes called rubber banding [63]. A game is also often more enjoyable if one is not losing by a large amount. Players who are behind can receive a boost to help them recover, and players who are ahead are prevented from maintaining or accelerating their lead. However, too much catching up can lead to games in which the winner is not determined until the very end, making early moves meaningless.

Catch-up mechanisms exist in many games, from board games using variable scoring (e.g. Hare & Tortoise [165]) or time tracks (e.g. Tokaido [18]) to video games with variable powerups (e.g. Super Mario Kart [140]). The game Catchup [20], by Nick Bentley, uses a catch-up rule that permits the player who is behind to add an extra piece each turn. Zhang-Qi [220] is similar (though we were not aware of it when designing ours) but uses a specific 32-element set, places markers on a uniquely shaped board, and describes the catch-up rule as an optional rule.

In this chapter, we examine inequity aversion and closeness in Catch-Up, a two-player game in which the players’ scores remain close throughout the game, making the eventual winner – if there is one – hard to predict [96]. Because neither player can build up an insurmountable lead, its play creates tension and drama, even between players of different skill. We show how the game is played, demonstrate that its simple rules lead to complex game dynamics, analyze some of its most important properties, and discuss possible extensions.

### 9.2 Catch-Up

Catch-Up [96] is described in Section 2.4 but the rules are repeated here:

1. **Catch-Up** starts with a collection of numbers $S$, which we call the set. Two players, $P_1$ and $P_2$, each begin with a score of zero. Player $P_1$ starts the game.

2. Starting with $P_1$, the players alternate taking turns. On a turn, the acting player removes a number from $S$ and adds that number to their score. The player keeps removing numbers, one by one, until their score equals or exceeds the score of the other player.

3. When there are no more numbers in $S$, the game ends. The player with the higher score wins; the game is drawn if scores are tied.

Figure 9.1 shows an example short game won by player $P_2$. Catch-Up provides meaningful choices, with score balancing built into its rules. Players alternate holding the lead, with
the score difference bounded by a relatively small number. Players are therefore uncertain who will win a game of *Catch-Up* until the end. The game is surprisingly complex, given the simplicity of its rules, with no trivial heuristics that enable players to win every time. Note that the set of numbers is actually a *multiset* or *bag* of numbers, i.e. some numbers may be repeated, but we use the term ‘set’ for simplicity.

*Catch-Up* is a combinatorial perfect information game, so even though the players have close scores throughout the game, there exist optimal strategies to win or tie. Thus, the scoring mechanism does not necessarily reflect who is more likely to win the game: a player may be in a game-theoretic winning position even though their score is lower than the other player’s. *Catch-Up* shows that creating scoring systems in which the current score is a reliable and meaningful indicator, in games with significant catch-up mechanisms, is indeed a challenge. Additionally, we show how $S$, the numbers the players start with, affects the game’s complexity and play dynamics.

The rules of *Catch-Up*, although minimal, define a game with interesting non-trivial properties. We study these properties using an approach similar to that used by Martin Gardner [74] and *Winning Ways For Your Mathematical Plays* [21].
9.3 Examples of Play in Catch-Up

We use the notation $\text{Catch-Up}(S)$ to describe the game played with the collection of numbers $S$. For example, $\text{Catch-Up}([1, \ldots, N])$ is played with $S = \{1, \ldots, N\}$, the consecutive positive integers from 1 through $N$.

9.3.1 Catch-Up($\{1, \ldots, 4\}$)

Figure 9.2 shows the full game tree for $\text{Catch-Up}([1, \ldots, 4])$. Assuming optimal play by $P_1$ (triangle) and $P_2$ (square), winning, drawing, and losing positions, and moves for the acting player are indicated. One possible game might play out as follows, which is shown in steps in Figure 9.1. The set starts with $S = \{1, 2, 3, 4\}$. $P_1$ initially removes $\langle 3 \rangle$, and is ahead $3 - 0$. Play then switches to $P_2$, who can choose from $\{1, 2, 4\}$ and removes $\langle 2 \rangle$. Since the score is $3 - 2$ and $P_2$ is still behind, $P_2$ needs to remove another number. $P_2$, choosing from $\{1, 4\}$, removes $\langle 4 \rangle$. Thus, on $P_2$’s turn, the entire move was to remove $\langle 2, 4 \rangle$, and the score is now $3 - 6$. Since $P_2$ is ahead, play switches back to $P_1$. The set contains only $\{1\}$, which $P_1$ removes. The game ends with a final score of $4 - 6$, so $P_2$ is the winner by 2. This game could also have ended in a draw as follows: $P_1$ selects $\langle 2 \rangle$, $P_2$ selects $\langle 1, 4 \rangle$, and $P_1$ selects $\langle 3 \rangle$, tying the game at $5 - 5$ and illustrating how $P_2$ can force a draw.

Because of Rule 2, players always start their turns either tied or behind the other player. This means the player’s task is at least to catch up to the other player, but neither player can ‘snowball’ or jump far ahead. Conversely, this also means players will always end their turns either tied or ahead of the other player. This continual catching up or pulling ahead can add to the drama of the game, making each player feel the game is competitive to the end.
In order to keep players from memorizing strong opening moves, we propose that players could play with a randomized set – with repeated or missing numbers – such that there are too many possible game trees for players to memorize.

### 9.3.2 Physical Implementation

If Catch-Up is played as an abstract mathematical game, it requires detailed bookkeeping, which some players may find difficult. We propose a version played with physical pieces on a table, as shown in Figure 9.1 illustrating the moves described in Section 9.3.1. The pieces are designed to fit next to each other, such that the lengths can be quickly determined to see whose turn it is.

We believe the physical version is more pleasurable to play because the physical pieces simplify the arithmetic calculations, making the game more accessible [109]. If the shortest piece is 1 centimeter long, a tie game of Catch-Up({1, ..., 12}) would end up being \(\frac{1 + \ldots + 12}{2} = 39\) cm long, with the largest win margin at most 12 cm long.

### 9.3.3 Web-based Implementation

To help demonstrate how Catch-Up is played, we created a Javascript web-based implementation hosted at [http://game.engineering.nyu.edu/projects/catch-up/](http://game.engineering.nyu.edu/projects/catch-up/) as shown in Figure 9.3. There is no AI implemented, but two players can share a device to play the game. This version allows \(N\) to range between 2 and 12, inclusive.

![Figure 9.3: The Javascript web-based implementation of Catch-Up.](image)
9.4 Characteristics of Catch-Up

In this section, we analyze some of the characteristics of Catch-Up that result from forcing the score to stay close. Some of these metrics, such as branching factor, game-state size, and game-tree size, are classical metrics that are used for analyzing the complexity of common games \[247, 6\]. Other metrics, such as puzzle-like quality or the ability to come back in the endgame, have also been shown to be effective at determining if a computer-generated game is well-designed \[37\]. The availability of interesting heuristics and non-trivial strategies also is known to be a useful characteristic for interesting games \[63\]. Additional metrics are presented in the original research paper (but are less relevant for proving the thesis).

9.4.1 Puzzle-Like Quality

Catch-Up has a puzzle-like quality \[37\]; it is challenging to find solutions that lead to a win or draw. For example, Figure 9.4 shows two subtrees of Catch-Up($\{1, ..., 7\}$). In Fig. 9.4a, $P_2$ (squares) is in a winning position, but they must proceed carefully. This position was reached by $P_1$ (triangles) initially taking $\langle 3 \rangle$, $P_2$ taking $\langle 5 \rangle$, and $P_1$ taking $\langle 6 \rangle$, leaving a set of $\{1, 2, 4, 7\}$ and a score difference of 4. One move leads to a win, one leads to a draw, and all other moves lead to losses (random play would lead to a 7/8 chance of choosing a sub-optimal move). $P_1$’s optimal move is to choose the largest sum possible, removing either $\langle 1, 6 \rangle$ or $\langle 2, 1, 7 \rangle$. Each uses the same numbers and reaches the same score.

The strategy of maximizing one’s lead does not always work. The subtree of Fig. 9.4b is reached by $P_1$ taking $\langle 2 \rangle$, $P_2$ taking $\langle 5 \rangle$, and $P_1$ taking $\langle 7 \rangle$, giving a score difference of 4. If $P_2$ then maximizes his score by choosing $\langle 3, 6 \rangle$, worth 9 points, this leads to a forced draw. But if $P_2$ chooses the lower valued $\langle 1, 6 \rangle$, worth only 7 points, they force a win. Making this even more tricky, choosing $\langle 3, 4 \rangle$, also worth 7 points, leads to a forced loss for $P_2$.

In Sec. 9.5 we discuss various simple strategies and heuristics that novice players might use to help navigate the game tree. This shows the relative effectiveness of each heuristic.

9.4.2 Endgame

On every turn, a player of Catch-Up comes from behind or from a tied score. However, there are many cases in which a player, who will lose if the opponent plays optimally, can still come back to win very late in the game if the opponent makes a mistake on their last move. This implies that both players must focus on winning up until their very last moves.

In Figure 9.5 we show an example of a subtree of Catch-Up($\{1, ..., 7\}$) wherein optimal play produces a loss for $P_1$, but there is still a chance for a win with the last moves in the
Figure 9.4: Puzzle-like quality of two subtrees of $\text{Catch-Up}(\{1, \ldots, 7\})$.

Figure 9.5: Endgame example in a subtree of $\text{Catch-Up}(\{1, \ldots, 7\})$: $P_1$ (triangle) has lost the game, assuming optimal play, but choosing $\langle 3 \rangle$ or $\langle 4 \rangle$ requires $P_2$ to make a final winning move in the endgame, whereas choosing $\langle 7 \rangle$ guarantees a loss.
game if $P_2$ plays non-optimally. To reach this position, assume $P_1$ chooses $\langle 1 \rangle$, $P_2$ chooses $\langle 2 \rangle$, $P_1$ chooses $\langle 5 \rangle$, and $P_2$ chooses $\langle 6 \rangle$, so the score difference is 2 and $\{3, 4, 7\}$ remain in the set. Now, if $P_1$ chooses $\langle 7 \rangle$, they will lose when $P_2$ is forced to choose $\langle 3, 4 \rangle$. However, if $P_1$ chooses $\langle 3 \rangle$ or $\langle 4 \rangle$ – putting them 1 or 2 ahead – then $P_2$ must choose $\langle 7 \rangle$ to win.

9.4.3 Drawn Games

Drawn games are sometimes possible in Catch-Up if the sum of the numbers in $S$ is even. Whether optimal play leads to a draw, or a win for $P_1$, depends on $S$. Games that end in a draw may be dissatisfying for some players because there is no winner (although draws do not seem to bother many Chess players, for example).

Whether Catch-Up permits draws is solely determined by the set $S$. In the case of $\text{Catch-Up}(\{1, \ldots, N\})$, it depends on the value $N$. For all $n \geq 0$, games of the form $N = 4n + 1$ and $N = 4n + 2$ always have a winner by at least one point, because the sum of all the points $1, 2, \ldots, N$ is odd; there is no way to split them evenly. Conversely, games of the form $N = 4n + 3$ or $N = 4n + 4$ can have games that end in a draw, because the sum of all the numbers is even. We provide a proof of this in the original Catch-Up paper [96]. We calculate in Section 9.4.4 how often draws will occur as a function of $N$.

For $\text{Catch-Up}(\{1, \ldots, N\})$ with $N = 4n + 3$ or $N = 4n + 4$, we have calculated up to $N = 20$ that optimal play by both players leads to a draw (see Section 9.5.2). However, optimal play in any even-sum game of $\text{Catch-Up}(S)$ for any arbitrary $S$ does not necessarily produce a draw. Consider an even-sum game with repeated numbers $S = \{2, 2, 2, 3, 3\}$, shown in Figure 9.6, which sum to 12. Here $P_1$ can force a win by initially choosing $\langle 2 \rangle$. Drawn games are still possible for this set, but they are not the result of optimal play.

Furthermore, it is easy to see that some even-sum games do not even permit a draw. Consider $\text{Catch-Up}(\{2, 4, 6, 8, 10\})$, which is even-sum, but obviously no subsets of these numbers can produce a $15 - 15$ tie, since even numbers cannot sum to an odd number.

Figure 9.6: Even-sum game in which $P_1$ can force a win, but which can also end in a draw.
9.4.4 Importance of the First Move

One criticism of catch-up type mechanisms is that the early moves in the game have no importance. We show here that the first move $P_1$ makes in $\text{Catch-Up}(\{1, \ldots, 7\})$ has an impact on the percentage of ways that $P_1$ can win, lose, or draw. In Table 9.1 each row shows the change from $50\%$–$50\%$ in the percentage of ways that the game can end in a win, lose, or draw, given that $P_1$ makes the indicated first move.

By choosing $\langle 3 \rangle$, $P_1$ increases the ways of winning by $6.43\%$ and reduces the ways of losing by $6.90\%$. Conversely, choosing $\langle 6 \rangle$ decreases the ways of winning by $4.13\%$ and increases the ways of losing by $3.65\%$. Clearly, the first move has an impact on the ability of non-optimal players to achieve a win, loss, or draw; but this has no bearing on optimal play.

<table>
<thead>
<tr>
<th>Move</th>
<th>$\Delta$ Win$%$</th>
<th>$\Delta$ Lose$%$</th>
<th>$\Delta$ Draw$%$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\langle 1 \rangle$</td>
<td>0.60$%$</td>
<td>0.60$%$</td>
<td>-1.19$%$</td>
</tr>
<tr>
<td>$\langle 2 \rangle$</td>
<td>-2.46$%$</td>
<td>3.65$%$</td>
<td>-1.19$%$</td>
</tr>
<tr>
<td>$\langle 3 \rangle$</td>
<td>6.43$%$</td>
<td>-6.90$%$</td>
<td>0.48$%$</td>
</tr>
<tr>
<td>$\langle 4 \rangle$</td>
<td>3.10$%$</td>
<td>-1.90$%$</td>
<td>-1.19$%$</td>
</tr>
<tr>
<td>$\langle 5 \rangle$</td>
<td>0.32$%$</td>
<td>-0.79$%$</td>
<td>0.48$%$</td>
</tr>
<tr>
<td>$\langle 6 \rangle$</td>
<td>-4.13$%$</td>
<td>3.65$%$</td>
<td>0.48$%$</td>
</tr>
<tr>
<td>$\langle 7 \rangle$</td>
<td>-3.85$%$</td>
<td>1.71$%$</td>
<td>2.14$%$</td>
</tr>
</tbody>
</table>

Table 9.1: Percentage change with $P_1$ moving first.

9.4.5 Game-Tree Size

The game-tree size gives the total number of unique play-throughs, iterating through all valid moves of the game. This is equivalent to counting the number of terminal nodes in the game tree. For simplicity, we consider each permutation of a player’s removal choices in a single turn to be a distinct branch, although the order of removals within a turn does not matter during play.

Large game trees are more difficult for players to utilize in play, as they do not permit memorization of the best moves; however, they also make it computationally harder for analysis by adversarial search. By increasing the size of the set $S$, the game tree rapidly increases in size.

For $\text{Catch-Up}(\{1, \ldots, N\})$, the game-tree size is exactly $N!$, which is the number of ways the numbers in the set can be picked, and then assigning turns after determining the order the numbers are picked to make it a valid game. Table 9.2 enumerates all possible games of $\text{Catch-Up}(\{1, \ldots, N\})$ for up to $N = 18$ and counts the number of terminal nodes, verifying the game-tree size is indeed $N!$. 
9.4.6 State-Space Size

State-space size is the number of possible states of the game, reflecting the fact that many states can be reached from multiple moves [6]. This converts the game tree into a directed acyclic graph, because a game state represented in the graph can have multiple parents.

In Catch-Up, the necessary states to track are: current player, current score and numbers remaining in the set. We do not have an analytical bound for the state-space size, but empirical data generated for small $N$, shown in Table 9.2, demonstrates that it grows much more slowly than the game-tree size. For large $N$, the state-space size is much smaller because there are many ways to reach the same game state using different moves.

For example, for any $Catch-Up\{1, ..., N\}$ for $N \geq 3$, the following game traces all reach an identical game state with tied score $3 - 3$: $(\langle 3 \rangle, \langle 1, 2 \rangle); (\langle 3 \rangle, \langle 2, 1 \rangle); ((2), \langle 3 \rangle, \langle 1 \rangle); ((1), \langle 3 \rangle, \langle 2 \rangle)$. Thus, huge benefits occur from caching results in a transposition table [187] when exploring the game graph for optimal moves.

9.4.7 Game-Tree Depth

The depth of a game tree for $Catch-Up(S)$ can be no deeper than $|S|$ turns. Thereby, the designer or players can control the length of the game by choosing the size of $S$.

This maximal depth occurs when each player selects the smallest number in the set, with each ending a turn with only one number removed. This gives a total of $N$ turns. Games

---

Table 9.2: Measures for $Catch-Up(\{1, ..., N\})$ for values of $N = 3$ to $N = 18$.

<table>
<thead>
<tr>
<th>N</th>
<th>game-tree size ($N!$)</th>
<th>state-space size</th>
<th>$P_1$ optimal play value</th>
<th>max branch factor</th>
<th>min depth</th>
<th>win %</th>
<th>tie %</th>
<th>loss %</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>6</td>
<td>11</td>
<td>0</td>
<td>3</td>
<td>2</td>
<td>16.67%</td>
<td>66.67%</td>
<td>16.67%</td>
</tr>
<tr>
<td>4</td>
<td>24</td>
<td>33</td>
<td>0</td>
<td>6</td>
<td>2</td>
<td>33.33%</td>
<td>33.33%</td>
<td>33.33%</td>
</tr>
<tr>
<td>5</td>
<td>120</td>
<td>90</td>
<td>1</td>
<td>16</td>
<td>3</td>
<td>50.00%</td>
<td>0.00%</td>
<td>50.00%</td>
</tr>
<tr>
<td>6</td>
<td>720</td>
<td>236</td>
<td>1</td>
<td>36</td>
<td>3</td>
<td>50.00%</td>
<td>0.00%</td>
<td>50.00%</td>
</tr>
<tr>
<td>7</td>
<td>5040</td>
<td>591</td>
<td>0</td>
<td>78</td>
<td>3</td>
<td>38.57%</td>
<td>22.86%</td>
<td>38.57%</td>
</tr>
<tr>
<td>8</td>
<td>40320</td>
<td>1453</td>
<td>0</td>
<td>150</td>
<td>3</td>
<td>38.93%</td>
<td>22.14%</td>
<td>38.93%</td>
</tr>
<tr>
<td>9</td>
<td>3628800</td>
<td>3484</td>
<td>-1</td>
<td>272</td>
<td>4</td>
<td>50.00%</td>
<td>0.00%</td>
<td>50.00%</td>
</tr>
<tr>
<td>10</td>
<td>39916800</td>
<td>1874</td>
<td>-1</td>
<td>474</td>
<td>4</td>
<td>50.00%</td>
<td>0.00%</td>
<td>50.00%</td>
</tr>
<tr>
<td>11</td>
<td>479001600</td>
<td>42587</td>
<td>0</td>
<td>1470</td>
<td>5</td>
<td>42.47%</td>
<td>15.06%</td>
<td>42.47%</td>
</tr>
<tr>
<td>12</td>
<td>6227020800</td>
<td>95126</td>
<td>1</td>
<td>2448</td>
<td>5</td>
<td>50.00%</td>
<td>0.00%</td>
<td>50.00%</td>
</tr>
<tr>
<td>13</td>
<td>87178291200</td>
<td>210064</td>
<td>-1</td>
<td>3894</td>
<td>6</td>
<td>50.00%</td>
<td>0.00%</td>
<td>50.00%</td>
</tr>
<tr>
<td>14</td>
<td>1.30767E+12</td>
<td>459225</td>
<td>0</td>
<td>6158</td>
<td>6</td>
<td>43.86%</td>
<td>12.28%</td>
<td>43.86%</td>
</tr>
<tr>
<td>15</td>
<td>2.09228E+13</td>
<td>995349</td>
<td>0</td>
<td>10284</td>
<td>6</td>
<td>44.23%</td>
<td>11.53%</td>
<td>44.23%</td>
</tr>
<tr>
<td>16</td>
<td>6.40237E+15</td>
<td>2141652</td>
<td>1</td>
<td>16048</td>
<td>6</td>
<td>50.00%</td>
<td>0.00%</td>
<td>50.00%</td>
</tr>
<tr>
<td>17</td>
<td>3.55687E+14</td>
<td>4579850</td>
<td>-1</td>
<td>24762</td>
<td>7</td>
<td>50.00%</td>
<td>0.00%</td>
<td>50.00%</td>
</tr>
<tr>
<td>18</td>
<td>1.30767E+12</td>
<td>459225</td>
<td>0</td>
<td>1470</td>
<td>5</td>
<td>42.47%</td>
<td>15.06%</td>
<td>42.47%</td>
</tr>
</tbody>
</table>

---

1 If this information is known, it does not matter how the removed numbers were chosen to get to this state.
can certainly end sooner, because on some turns a player may select more than one number, decreasing the number of turns for that path in the tree.

The minimum length of the game is also determined by the size of the initial set. We do not have an analytical lower bound for $\text{Catch-Up}(\{1, \ldots, N\})$, but we present calculated minimum-depth values in Table 9.2.

### 9.4.8 Branching Factor

The maximum branching factor, which we call $B_{\text{max}}$, tells us how many possible moves there are on a turn in the worst case. The higher the branching factor, the more complicated a game can be for a player to explore. The maximum branching factor for $\text{Catch-Up}(\{1, \ldots, N\})$ is:

$$B_{\text{max}} = O\left(N^{\sqrt{2N}+1}\right)$$

A derivation of this upper bound is provided in the original $\text{Catch-Up}$ paper [96].

In Table 9.2, we show the empirical maximum branching factor, which is the maximum of the number of first moves by $P_1$ and the number of replies (to first moves) by $P_2$. This table clearly shows that the maximum branching factor is exceedingly high for a game, making it difficult to explore the entire early game tree for large $N$.

As $\text{Catch-Up}$ proceeds, $S$ has fewer numbers to choose from, so the branching factor $B_t$ for each turn $t$ will decrease until the final move, which forces the last player to select all remaining numbers. The average branching factor $B_{\text{avg}}$ will be less at each layer of the game tree. We do not have an analytical bound for $B_{\text{avg}}$, but can calculate it empirically for small $N$, allowing us to generate the entire tree.

In Table 9.3, we give the average and maximum branching factors per turn for $\text{Catch-Up}(\{1, \ldots, 12\})$. Level $l$ of the tree represents the possible game states and moves available on turn $l$. To calculate the average branching factor, we expand the entire game tree and then calculate how many moves there are available on each level of the tree divided by the number of unique states on that level. The average and maximum branching factors peak on Turn 2 ($P_2$’s first turn) and then rapidly decrease as the game progresses to the end.

<table>
<thead>
<tr>
<th>Turn</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
</tr>
</thead>
<tbody>
<tr>
<td>$B_{\text{avg}}$</td>
<td>12</td>
<td>366.67</td>
<td>35.42</td>
<td>15.52</td>
<td>7.80</td>
<td>4.32</td>
<td>2.72</td>
<td>1.94</td>
<td>1.52</td>
<td>1.27</td>
<td>1.11</td>
<td>1.00</td>
</tr>
<tr>
<td>$B_{\text{max}}$</td>
<td>12</td>
<td>1470</td>
<td>738</td>
<td>738</td>
<td>560</td>
<td>560</td>
<td>258</td>
<td>108</td>
<td>24</td>
<td>6</td>
<td>2</td>
<td>1</td>
</tr>
</tbody>
</table>

Table 9.3: Average ($B_{\text{avg}}$) and maximum ($B_{\text{max}}$) branching factors for $\text{Catch-Up}(\{1, \ldots, 12\})$. 
9.4.9 Win/Loss/Draw Ratios

It is useful to understand if a game is balanced by looking at a game’s win/loss/draw ratios. For small \(N\), we can analyze the entire game tree to calculate the percentage of wins, losses, and draws. The Win %, Tie %, and Loss % columns in Table 9.2 show the results of exploring all possible games of \(\text{Catch-Up}([1, \ldots, N])\) from \(N = 3\) to \(18\). As explained in Section 9.4.3, tie games are impossible in games where \(T(N)\) is odd, and these tie percentages are indicated as 0.00%. As \(N\) increases, the chance of a random game ending in a draw decreases, which suggests that the games are not too ‘drawish’. We also note the games are balanced between \(P_1\) and \(P_2\), suggesting that there is no advantage in going first or second if not playing optimally.

In Figure 9.7, we show a circular tree plot for the entire game of \(\text{Catch-Up}([1\ldots7])\), which shows that wins and loses are distributed throughout the game tree.

9.5 Analyzing Strategies for Catch-Up

\(\text{Catch-Up}\), for any finite set \(S\), is a finite two-person zero-sum game of perfect information, so there exists a pair of optimal strategies such that (i) \(P_1\) can guarantee a win, (ii) \(P_2\) can guarantee a win, or (iii) the game is a draw [68]. In order for a perfect-information game to be non-trivial, the optimal strategy should not be obvious to play.

In addition, different strategies should present a heuristic tree [63], such that there are some simple heuristics that new players can learn, and better performing but more complicated heuristics for more sophisticated players.

9.5.1 Greedy Maximizing Is Not An Optimal Strategy

We showed in Section 9.4.1 that a strategy of selecting the largest sum of numbers possible is not always an optimal strategy, though it is an obvious heuristic that a player might try. As another example, in a game of \(N = 5\), with a set \(\{1, 2, 3, 4, 5\}\), if \(P_1\) always selects the numbers that gives them the largest lead, they will lose: \(P_1\) initially removes \(\langle 5 \rangle\), \(P_2\) can then remove \(\langle 1, 3, 4 \rangle\), forcing \(P_1\) to choose \(\langle 2 \rangle\) and lose the game 7 – 8.

This happens specifically because of the inequity aversion of Rule 2. If instead players were required to select a fixed number of numbers on each turn, then a maximizing-score strategy would be dominant, making the game trivial. By contrast, the rules of \(\text{Catch-Up}\) lead to a game tree that makes optimal choices non-trivial: there are no obvious strategies to win every game.
Figure 9.7: The distribution of winning and losing moves in Catch-Up({1...7}) is mixed, making the strategy for winning non-trivial. Even with 7 numbers the game is difficult to explore.

9.5.2 Optimal Play

We do not know whether Catch-Up({1, ..., N}) is a win, loss, or draw for P₁ for any N; however, for a given set, we can efficiently run a minimax algorithm with alpha-beta pruning and transposition tables [187] to solve the game value, assuming optimal play by both players. We have calculated the game values for Catch-Up({1, ..., N}) up to N=20. Results for optimal play are shown in Table 9.4 in the optimal play row, with -1 being a loss for P₁, 1 being a win for P₁, and 0 being a tie game.

As described in Section 9.4.3, Catch-Up({1, ..., N}) games of the form $N = 4n + 3$ or $N = 4n + 4$ permit draws. We have calculated that these games, up to at least $N = 20$,
are draws with optimal play. We believe that this pattern holds for all \( n \), though we have not been able to prove this and can only offer it as a conjecture. Using Monte Carlo Tree Search \([38]\), we have explored values of \( N = 23, 24, 27, \) and 28 and did not find any contradictions.

<table>
<thead>
<tr>
<th>( N )</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
<th>16</th>
<th>17</th>
<th>18</th>
<th>19</th>
<th>20</th>
</tr>
</thead>
<tbody>
<tr>
<td>optimal play</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>-1</td>
<td>-1</td>
<td>0</td>
<td>0</td>
<td>-1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>-1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 9.4: Optimal play values for \( \text{Catch-Up}(\{1, ..., N\}) \) relative to \( P_1 \): 1 is a win, -1 is a loss, and 0 is a draw.

### 9.5.3 Human-Playable Heuristics

Although machines can efficiently search a game tree for optimal moves, humans do not think in the same way and, generally, do not find it enjoyable – or possible – to exhaustively explore every move when playing a game. In order for a strategy to work for human players, we need effective heuristics that are accessible and can be easily used. And for a game to have lasting depth, simple heuristics must be generally less effective than more complex ones, so there is a benefit for continued study and improvement.

We analyzed several simple human-usable heuristics for playing \( \text{Catch-Up} \). For these heuristics, if multiple moves could be chosen, one of them is picked at random. We do not claim that these are the only heuristics for players, or that players should follow any of them. Instead, they provide a starting point for strategies that new players might try, which help us understand if the game can be enjoyed by beginners.

1. **Random**: Players choose any move at random.
2. **MaxScore**: Players maximize scores every turn, extending leads by as much as possible.
3. **MinScore**: Players minimize scores every turn, keeping scores as close as possible.
4. **UseMostNums**: Players use as many numbers as possible, reducing the numbers available for the opponent.

For \( \text{Catch-Up}(\{1, ..., 10\}) \), which is an odd-sum game, optimal play by both players leads to a loss for \( P_1 \), but it is difficult for humans to play optimally. Instead, we can test the various simple heuristics and compare how they perform against each other.

For example, Table 9.5 shows the probabilities of \( P_1 \) winning when playing each of their heuristics against each of \( P_2 \)’s 100,000 times. The value in each cell indicates the percentage
of games in which $P_1$ wins; a value of 1 means that $P_1$ always wins, whereas a value of 0 means that $P_1$ always loses. Values $> 50\%$ in Table 9.5 are good for $P_1$, whereas values $< 50\%$ are good for $P_2$. Players are assumed to use the same heuristic throughout the entire game, without switching or adapting within a game to what the other player is doing.

The $P_1$ Random vs $P_2$ Random cell shows that random play gives close to a 50\% chance of winning, which indicates completely unskilled play will not favor one player over the other. Looking at the first column, we see the effect of $P_1$ using each heuristic against $P_2$ Random, and that $P_1$ MaxScore is the best of the four heuristics, improving $P_1$’s win rate to approximately 63\%, whereas $P_1$ MinScore is a bad heuristic, reducing the win rate to around 37.6\%. Similarly, if we look at the first row, which shows the effect of $P_2$ using each heuristic against $P_1$ Random, we see that $P_2$ MaxScore is the best heuristic for $P_2$.

However, if both players adopt the MaxScore heuristic, this is bad for $P_1$, reducing $P_1$’s win rate to around 14.3\%. $P_1$, playing against a $P_2$ MaxScore heuristic, would do better to use the $P_1$ MinScore heuristic, which was previously a bad choice. But this can lead to $P_2$ in turn switching to the $P_2$ MinScore heuristic, in which $P_2$ now wins every game. Likewise, $P_1$ now does better by switching back to the $P_1$ Random heuristic.

Given these simple heuristics, we already see an interesting pattern, in which there is not one dominating heuristic. This is an indication that Catch-Up does not have a trivial or obvious solution for human players. We believe this rock-paper-scissors balance, in which different heuristics perform better in some cases but not others, but no one heuristic dominates, is an important characteristic of deep and interesting games.

These heuristics do not necessarily generalize to other sets $S$. Just because a heuristic does well in Catch-Up($\{1, \ldots, 10\}$) does not mean it does well in Catch-Up($\{1, \ldots, 9\}$), another odd-sum game. For example, $P_1$ MinScore vs $P_2$ MaxScore wins 79.7\% for $P_1$ in the former game, but flips to only a 34.1\% win rate for $P_1$ in the latter game. Clearly, these heuristics offer only a glimpse into optimal play of Catch-Up.

Table 9.5: Win percentages for $P_1$ when playing different heuristics against $P_2$ in Catch-Up($\{1, \ldots, 10\}$).
9.5.4 Climbing the Heuristics Tree

We can simulate a player adopting a new, better heuristic by combining the previous four heuristics. Instead of deciding between multiple moves randomly, we can apply a second-level heuristic to choose between multiple moves. For example, $P_1$ using $\text{UseMostNums} + \text{MinScore}$ would first pick moves to use the most numbers, and if there is more than one remaining move to choose from, they choose the move that sums to the smallest number. As before, any final remaining options are eliminated by selecting one at random.

If $P_2$ adopts this $\text{UseMostNums} + \text{MinScore}$ combination heuristic, but $P_1$ stays with the original heuristics, $P_2$ now wins every game against two $P_1$ heuristics, and wins a slight majority of games otherwise, as shown in Table 9.6. Note that the purpose of this section is not to present the reader with the best heuristics, but to show that Catch-Up provides a compelling platform for developing effective heuristics for human play.

<table>
<thead>
<tr>
<th>$P_2$</th>
<th>$P_1$</th>
<th>$P_1$ Random</th>
<th>$P_1$ MaxScore</th>
<th>$P_1$ MinScore</th>
<th>$P_1$ UseMostNums</th>
</tr>
</thead>
<tbody>
<tr>
<td>UseMostNums + MinScore</td>
<td>Random</td>
<td>45.88%</td>
<td>0.00%</td>
<td>0.00%</td>
<td>45.55%</td>
</tr>
</tbody>
</table>

Table 9.6: $P_2$ win rate using combination heuristic.

9.5.5 Generating Heuristics with Genetic Programming

We can expect that higher order heuristics or deeper search would be necessary for players to improve, as they build better performing strategies and to leverage their experience [63, 122]. The simple heuristics presented in the previous subsections are a starting point for a new player, but we now address heuristics that are slightly more complicated and can use branching. This subsection describes a previously unpublished pilot experiment with Catch-Up heuristics that inspired future work for generating heuristics for novice players in Blackjack [219, 198].

One way to model human-usable heuristics is to use a simple decision tree. By limiting the size of the tree and the complexity of the decision clauses, we can model limited human processing power. Other possibilities are to use fast and frugal trees (FFT), which have been shown to effectively model human reasoning in resource limited situations [77, 76]. Other similar approaches are to use a decision list [181, 13]. For this pilot experiment we chose to use decision trees [117] because they are simpler to implement and can demonstrate what types of heuristics can be generated before imposing additional constraints.
Instead of iterating over all possible decision trees, we employed genetic programming \cite{171} to build the trees algorithmically against a fixed opponent strategy. In genetic programming, a primitive set that contains possible statements and clauses is used to build up novel trees. In each generation, a population of heuristic trees is generated and tested in a tournament to see which heuristic performs the best against the given opponent strategy. The top performing trees are allowed to reproduce via crossover (mixing of the two trees) as well as mutation (making random changes to a tree) for the next generation. We implemented the search algorithm using the \texttt{deap} genetic programming framework in Python \cite{176}, using the $\mu + \lambda - ES$ method with crossover probability of .5, mutation probability of .1, parent population size $\mu = 100$, offspring population size of $\lambda = 300$, and $n = 30$ generations. The initial population is generated with the “half and half” \cite{171} technique with a maximum tree depth of 4.

In addition to the Random, MaxScore, MinScore, and UseMostNums, we also added UseMostNums where players use as few numbers as possible and SelectMove – Containing($i$) where players choose moves that contain the $i$ piece (the value for $i$ is also generated in the tree). The primitive set also includes numbers from 1 to $N$, $<, >, \geq, \neq$, and, or, not, and IfThenElse operators, plus True and False. Each decision node is provided the current board, how many moves already played, and the score difference.

When playing against a Random opponent, the genetic algorithm’s best generated strategy is given in Figure 9.8. This strategy performs with a win rate of 67.9%, outperforming all the other heuristics described in this chapter against Random. Although this heuristic is more complex, it still can be learned and executed relatively simply by players.

While we were able to generate the superior heuristic given in Figure 9.8 that is still easy for a human to learn, there are many unaddressed challenges uncovered by this approach. Firstly, we hand selected the primitive set for the genetic algorithm, which implicitly

![Diagram](image)

Figure 9.8: Generating a heuristic with genetic programming. The best performing heuristic with depth no greater than 4, winning 67.9% of the time against the Random heuristic.
encodes human intelligence into the process. Presumably a genetic algorithm with enough time would generate subroutines [10] or base-level heuristics but this was not tested or demonstrated by this pilot experiment. Secondly, we did not experiment with the two players co-evolving their strategies, but instead kept one player’s strategy fixed. Co-evolving strategies would model game play more realistically, as we expect a player community to evolve their strategies together (e.g. the meta-game of trying to choose a strategy based on what is currently popular among opponent strategies [22, 63]). Third, we did not examine how the quality and availability of the heuristics might be used to iteratively tune the game to optimize for more interesting game play. For example, we could potentially search for different values of $N$ to define the board, to find the one that gives rise to more interesting heuristics. This in turn requires a metric for understanding if a heuristic is interesting or not, which we did not explore in this experiment.

9.6 Conclusions

One of the most interesting characteristics of Catch-Up is the complexity of the game tree, given its minimal game rules. This makes it challenging for players to calculate optimal moves by backward induction and adversarial search, necessitating the use of heuristics to play the game. Catch-up rules not only can encourage drama and tension in games, but they also have interesting mathematically emergent properties.

In two-player or multiplayer games, the players’ current scores often provide a clue as to how well they are doing in comparison to the other players. One interesting aspect of Catch-Up is that until the last few moves, the scores do not provide this information because the lead switches on every turn (except for ties). Thus, players need to generate other methods of evaluating the state of the game so that they can tell if they are ahead or behind, but these are not obvious in a game like Catch-Up. Players accustomed to treating current scores as an indication of who is winning may find this to be an interesting feature, or an unpleasant surprise.

We believe that our work on Catch-Up offers some important lessons:

- The structure and form of the pieces can greatly change how the game is perceived. The accessibility of the physical version facilitates play because it does not require the players to keep track of their scores.

- Catch-up mechanisms are intended to keep players feeling that the game is close. In combinatorial games, however, it can be disguising the actual state of the game and the likelihood of each player to win, lose, or draw.
• The starting conditions for a game – the set $S$ in *Catch-Up* – has a huge impact on the solution space and play dynamics.

• Simple heuristics are easy to implement in software and can help determine if new players can successfully compete.

• Proving if a game has good characteristics is often significantly more difficult than simulating them; yet much can still be learned from simulating game play.

This chapter has focused on perfect information two-player games. In the next chapter, we focus on the quantitative analysis of two-player games with randomness and imperfect information.
Chapter 10
Exploring Characteristics of Dice Games

In this chapter we examine how score distributions can be used to measure the characteristics of closeness, win rate, and tie percentage in simple two-player dice games, such as what might appear as an individual battle in a strategy war board game. By adjusting parameters of the dice game, for example the number of dice rolled or the number of sides on the dice, we can measure the resulting effects on the game’s characteristics. These score distributions are built through analyzing the probability of each dice roll and calculating the outcomes for every possible roll. Throughout this chapter, we use the common notation $n_{dk}$ to mean a player rolls $n$ dice that have $k$ sides (e.g. 5d6 means rolling 5 dice, each with 6 sides). This work was previously published as [95].

10.1 Introduction

Dice are a popular source of randomness in games, and can be used to simulate combat and other contests. While some games have deterministic rules for exactly how a battle will resolve, many games add some randomness, so that it is uncertain exactly who will win a battle. In games like Risk [164], two players roll dice at the same time, and then compare their values, with the higher value eliminating the opponent’s unit. Others use a hit-based system, like in Axis and Allies [138], where a die roll of a target value or less is a successful hit, with stronger units simulated by larger target values and larger armies rolling more dice. In both games, stronger forces are more likely to win the battle, but lucky or unlucky rolls can result in one player performing far better, leading to a wide difference in scores.

Given a large number of games played between players, unlucky and lucky rolls will balance out so players with better strategy will probably end up winning; however, people might not play the same game enough times for the probabilities to even out. Instead, they play a much smaller number of rolls spread across one or a couple of play sessions.
The gambler’s fallacy is the common belief that dice act with local representativeness: a small number of dice rolls should be very close to the expected probabilities. Therefore, it can be frustrating when rolling poorly against an opponent: players often blame the dice, or themselves, for bad rolls, even though logic and reason indicates that everyone has the same skill at rolling dice.

Although there are thousands of games based on dice (BoardGameGeek lists over 7,000 entries for dice and hundreds of games are described in detail in [19, 116]), we examine games where players roll and compare the individual dice values, as shown in Figure 10.1. In a dice battle, the dice are sorted in decreasing order and then paired up. Whichever player rolled a higher value on the pair wins a point. The points are summed, and whomever has more points wins the battle. We use the term battle to imply an event resolved within a larger game. The word is normally used to refer to combat, but our analysis can be used any time players compare dice outcomes in a contest.

We examine different variants and show how different factors affect the distribution of scores and other metrics that are helpful for evaluating a game. By adjusting the dice mechanics, a designer can influence several metrics we discuss in Section 3.7.3: the expected closeness of the outcomes of a battle, the win bias in favor of one of the players, and the tie percentage, the fraction of battles that end in a tie. The variants we examine include different numbers of dice, various sided dice, different ways to sort the dice, and various ways to break ties. We focus only on metrics that examine the final scores of the dice battle; we do not evaluate anything about how scores evolve during the battle itself (which we

https://boardgamegeek.com/boardgamedumbnails/1017/dice

Figure 10.1: An example of a dice battle.
believe would be essential for more complicated games). But for simple dice battles, which are a component of a longer game, we can just focus on the end results. Closeness is a new metric we introduced in [95] and is useful for measuring score inequity in games.

Dice have come in various numbers of sides for millennia [106]: some of the oldest dice, dating back to at least 3500 B.C., were made from knucklebones with 4 flat sides and 2 rounded ones. Later, 6-sided dice were created by polishing down the rounded sides. The dot layout we see on today’s d6 also come from antiquity. Ancient dice also come in the form of sticks with 4 long sides for Pachisi or 2 long sides for Senet [66]. The common dice in use for modern games are 4, 6, 8, 10, 12, and 20-sided, but other variants exist.

By understanding how rules and randomness affect closeness, a designer can then choose the appropriate combination to try to achieve their desired game experience. One may prefer for their game to be highly unpredictable with large swings, intentionally increasing the risk for players to commit their limited resources. In addition, randomness can make a game appear to be more balanced because the weaker player can occasionally win against the stronger player [63]. Large swings may be more emotional and chaotic, and the “struggle to master uncertainty” can be considered “central to the appeal of games” [48]. Or, a designer may prefer for each battle to be close, to limit the feelings of one side dominating the other in what might be experienced as unfair or unbalanced, in a trait known as inequity aversion [67, 96], as also discussed in Chapter 9. Similarly, one may prefer to allow ties (simulating evenly matched battles), or wish to eliminate the opportunity for ties (forcing one side to win). Finally, one may vary the rules between each battle within a game, to represent changing strengths and weaknesses of the players and to provide aid to the losing player. A designer can adjust randomness to encourage situations appropriate for their game.

For most sections in this chapter, we calculate the exact probabilities for each outcome by iterating over all possible rolls, tabulating the final score difference. Because each outcome is independent, we can parallelize the experiments across multiple processors to speed up the calculations (details about how many calculations are given in the Appendix of [95]). There are other methods one could use to computationally evaluate the odds, such as with a dice probability language like AnyDice [70] or Troll [143], or by using Monte Carlo simulation (we use simulation when examining rerolls in Section 10.7). Writing the analytical probabilities becomes difficult for more complex games and we feel that presenting equations of this type has limited utility for most game designers, so we do not do so in this chapter.
10.2 Rolling Sorted or Unsorted

Many games ask the players to roll a handful of dice. A method to assign the dice into pairs is required. *Risk* sorts the dice in numerical order, from largest value rolled to smallest, which is the approach we will take here. We also consider games where the dice are rolled one at a time (or perhaps one die is rolled several times) and left unsorted. We now show how these two methods of rolling dice significantly change the distribution of score differences.

10.2.1 Sorting Dice, With Ties

We first look at the case where each player rolls all $n$ of their $k$-sided dice and then sorts them in decreasing order. The two sets of dice are then matched and compared. If a player rolls one or more identical numbers, the relative order of those two dice does not matter.

Figure 10.2 shows the distribution of score differences when each player rolls $n = 5$ dice and sorts them. We vary $k$, the number of sides. Ties are allowed, with neither player earning a point. We see the games all have a win bias of 0, as expected from the player rule symmetry. Additionally, tie percentage decreases as we increase sides: the more possible numbers to roll, the less likely the players will roll the same values. Increasing sides also decreases closeness, making higher score differences more likely to occur. For the case of 5d8 and 5d10, the score distribution is approximately flat: wide differences in scores are equally common to close scores.

5d2 stands out with a bell-shaped curve with significantly higher closeness: close games are more likely, but ties are more likely as well. Nonetheless, two-sided dice, i.e. coins, are not typically used in games partly because coins are difficult to toss and keep from rolling off the table ([*Coin Age*][137] and [*Shift*][207] are notable counter-examples, and some

<table>
<thead>
<tr>
<th>Game</th>
<th>win bias</th>
<th>tie %</th>
<th>closeness</th>
</tr>
</thead>
<tbody>
<tr>
<td>5d2</td>
<td>0.00</td>
<td>24.61</td>
<td>0.632</td>
</tr>
<tr>
<td>5d4</td>
<td>0.00</td>
<td>11.97</td>
<td>0.384</td>
</tr>
<tr>
<td>5d6</td>
<td>0.00</td>
<td>9.91</td>
<td>0.340</td>
</tr>
<tr>
<td>5d8</td>
<td>0.00</td>
<td>9.15</td>
<td>0.323</td>
</tr>
<tr>
<td>5d10</td>
<td>0.00</td>
<td>8.64</td>
<td>0.315</td>
</tr>
</tbody>
</table>

Figure 10.2: Rolling 5dk sorted, with ties.
10.2.1 Dice Sorted, With Ties

Figure 10.3: Rolling \( n \)d6 sorted, with ties.

countries use square coins). However, stick dice – elongated dice that only land on the two long sides – do not roll away. Alternately, one could use even-sided dice and use the odd numbers to represent 1 and even numbers to represent 2.

In Figure [10.3] we see how changing the number of six-sided dice affects the distribution of score differences. They remain symmetric with a win bias of 0, and after 2d6, adding more dice decreases the tie percentage. Closeness decreases as we add dice, which makes sense as with more dice there is a higher probability of score differences tending away from 0. 1d6 has a closeness > 1, because it allows 0-0 ties; without ties, closeness would be exactly 1.0.

10.2.2 Dice Unsorted, With Ties

We now examine the case where the dice are rolled and left unsorted. The dice could be rolled one at a time, possibly bringing out more drama as the battle is played out in single die rounds. Both players still roll \( n \) dice, but the order they were rolled in is used when
comparing, as shown in Figure 10.4a. As before, the player with the higher value earns a point and if tied then neither player earns a point. Although we will think of the dice being rolled one at a time (and actually generate them in our simulations this way), it’s also possible for players to roll a handful of dice to quickly create a sequence, as shown in Figure 10.4b. A player first rolls a handful of dice on the table. The dice are then put in order from left to right as they settled on the table. If two dice have the same horizontal position on the table (as the ♠ and ♣ do in the example), the die further away from the player will come before the die that is near.

In Figure 10.5, we examine how changing the number of sides of dice changes the distribution of ties and close games. We compare 2-sided dice (coins), 4-sided, 6-sided, 8-sided, and 10-sided dice. In all cases, the game is balanced, because the win bias is 0. We can see that more sides decreases the odds of the battle ending in a tie score. We can also see that more sides decreases closeness and therefore increases the odds of a lopsided victory with more extreme score differences between the players.

In Figure 10.6, we examine how changing the number of dice rolled affects the score difference. All games are balanced, since the win bias remains 0 for these games no matter how many dice are rolled. Ties are much more common when rolling an even number of dice. When comparing with Figure 10.3, we see that rolling unsorted increases the percentage of ties. As for closeness, more sides decrease the closeness, as we’ve also seen when rolling sorted dice.

---

2An anonymous reviewer of the original paper mentioned their preferred method for rolling unsorted $nd6$ is to throw dice against a sloped box lid: the dice line up in a random order as they slide against the lid wall. Occasionally one die might stop against another die instead of the wall; in that case, simply jiggle the lid.

<table>
<thead>
<tr>
<th>Game</th>
<th>win bias</th>
<th>tie %</th>
<th>closeness</th>
</tr>
</thead>
<tbody>
<tr>
<td>5d2</td>
<td>0.00</td>
<td>24.61</td>
<td>0.632</td>
</tr>
<tr>
<td>5d4</td>
<td>0.00</td>
<td>19.32</td>
<td>0.516</td>
</tr>
<tr>
<td>5d6</td>
<td>0.00</td>
<td>16.69</td>
<td>0.490</td>
</tr>
<tr>
<td>5d8</td>
<td>0.00</td>
<td>14.49</td>
<td>0.478</td>
</tr>
<tr>
<td>5d10</td>
<td>0.00</td>
<td>12.71</td>
<td>0.471</td>
</tr>
</tbody>
</table>

---

Figure 10.5: Rolling $5dk$ unsorted, with ties.
10.2.3 Sorted Vs. Unsorted

In Figure 10.7 we review the effect of changing the way that dice are rolled, while keeping the same number of dice and number of sides. Rolling sorted has a flat distribution that leads to a higher likelihood of larger score differences, while rolling unsorted has a more normal-like distribution where closer games are more likely and closeness is higher. However, higher closeness increase tie percentage.

The game designer can choose the method they find more desirable for the particular game they are creating. In addition to choosing between rolling sorted or unsorted, the designer can change the number of dice and number of sides on the dice. Using fewer sides on the dice increases closeness, but also increases the tie percentage. Using fewer dice increases closeness, but again generally increases the tie percentage. We address ties in the next sections.

<table>
<thead>
<tr>
<th>Game</th>
<th>win bias</th>
<th>tie %</th>
<th>closeness</th>
</tr>
</thead>
<tbody>
<tr>
<td>1d6</td>
<td>0.00</td>
<td>16.67</td>
<td>1.095</td>
</tr>
<tr>
<td>2d6</td>
<td>0.00</td>
<td>37.50</td>
<td>0.775</td>
</tr>
<tr>
<td>3d6</td>
<td>0.00</td>
<td>17.82</td>
<td>0.632</td>
</tr>
<tr>
<td>4d6</td>
<td>0.00</td>
<td>23.95</td>
<td>0.548</td>
</tr>
<tr>
<td>5d6</td>
<td>0.00</td>
<td>16.69</td>
<td>0.490</td>
</tr>
</tbody>
</table>

Figure 10.7: 5d6 rolled sorted vs unsorted.
10.3 Resolving Tied Battles

In the previous section, when dice were rolled with the same values, neither player received a point for that pair of dice. This leads to some situations where the players get a 0 score difference and tie the game (with as much as 24.6% for the 5d2 case). For games where \( n \) is even, a score difference of 0 can occur (becoming less likely as \( k \) increases).

A game designer might wish that tie games are not allowed. One simple way would be to have Player A automatically win whenever the battle ends with a score difference of 0; however, this would have a massive bias in favor of Player A. In this example, this would add an additional 24.6% bias that is likely unacceptable when trying to make the games close. To eliminate the bias over repeated battles, Player A and B could take turns receiving the win (perhaps by using a marker or coin indicate who next receives the tiebreak).

Another simple way that would not have bias would be for the players to flip a coin (or some other random 50% chance event) to decide who is the winner of the battle. Using dice, the players could roll 1\(dk\) and let the player with the higher value win the battle. If they tie again, they repeat the 1\(dk\) roll until there is not a tie – we analyze this type of rerolling in Section 10.7.

In the next few sections, we will examine other ways to change the rules of the game so that score differences of 0 will not occur for games when \( n \) is odd. When \( n \) is even, score differences of 0 can still occur, and one of the above final tiebreaker methods can be used.

10.4 Favoring One Player

We now investigate breaking tied dice by always having one player winning a point when two dice are equal. We examine the case where Player A will always win the point (as in Risk where defenders always win ties against attackers), but in general the same results apply if A and B are swapped. Favoring one player causes a bias, helping that player win more battles, so we also examine several ways to address this bias.

10.4.1 Rolling Sorted, Player A Wins Ties

In Figure 10.8 we see the score distributions that occur when tied dice give a point to Player A. First, we see these distributions are not symmetric, and are heavily skewed towards Player A, as reflected in the positive win bias. As one would expect, giving the ties to Player A causes that player to have an advantage over B. Increasing the number of sides on the die decreases the win bias – this is expected as with more sides on a die, it’s less likely for the
players to both roll the same number. When $n$ is odd, we also see that even score differences are no longer possible, and most importantly a tied score difference of 0 is no longer possible so tie percentage is always 0%. For the first time, we see closeness increasing as the number of sides increases, because the distributions are less skewed towards lopsided 5-0 wins.

**Figure 10.8:** Rolling $k$-sided dice sorted, A wins ties.

### 10.4.2 Rolling Unsorted, A Wins Ties

By rolling dice unsorted, closeness is increased for all numbers of dice, and the distribution is more centered, but there is also a significant bias towards Player A, as we can see from Figure 10.9. This is an improvement, but one might desire another way to eliminate the bias.

In conclusion, breaking ties in favor of one player eliminates ties, but creates a large win bias. However, this can be reduced with more sides on the dice. This bias occurs for both rolling sorted and unsorted, although rolling unsorted results in higher closeness and slightly lower win bias. We now examine ways to reduce this bias in various ways.

**Figure 10.9:** $k$-sided dice unsorted, A wins ties.
The bias introduced by having one player win ties can be undesirable for some designers and players, so we now look at a method of reducing this bias by having Player A roll fewer dice than Player B, to make up for the advantage they earn by winning ties. This is the strategy used in *Risk*: the winning-ties bias towards the Player A (defender) is reduced by allowing Player B (attacker) to roll an extra die when both sides are fighting with large armies. When rolling sorted, the dice are sorted in decreasing order, and the lowest-valued dice that are not matched are ignored. When rolled unsorted, if one player rolls fewer dice there is no way to decide which dice should be ignored. We thus only examine the case of rolling sorted.

We examine the effect of requiring Player A to roll fewer dice in Figure 10.10. Rolling two or three fewer dice significantly favors Player B, and rolling the same number of dice favors Player A. However, Player A rolling 4d6 against Player B rolling 5d6 has a relatively balanced distribution, no longer significantly favoring one player over the other. Unfortunately, ties once again occur for 4d6 vs 5d6 – they occur for any battle where A rolls an even number of dice since half of the games can be one by A and the other half by player B, ending up with a score difference of 0 – with a significant likelihood of a final tie score as shown in the figure.

Since having one fewer die made A and B relatively balanced when B rolls 5 dice, we can look at more cases when B rolls $n$ dice. In Figure 10.11 we have more cases where A has one fewer die than B. Most of these cases are relatively balanced, although 1d6 vs 2d6 still gives a significant advantage to B. Note that 1d6 v 2d6 and 2d6 v 3d6 occur in *Risk*. While allowing 3d6 vs 4d6 in *Risk* would have no ties and lower bias, some bias can still be valuable to encourage attacking (Player B) and a tie still means that both sides lose a unit.

To reduce the win bias introduced by having Player A win all ties, we reduced this bias.

<table>
<thead>
<tr>
<th>Game</th>
<th>win bias</th>
<th>tie %</th>
<th>closeness</th>
</tr>
</thead>
<tbody>
<tr>
<td>2d6 v 5d6</td>
<td>-35.61</td>
<td>32.37</td>
<td>0.608</td>
</tr>
<tr>
<td>3d6 v 5d6</td>
<td>-23.63</td>
<td>0.00</td>
<td>0.451</td>
</tr>
<tr>
<td>4d6 v 5d6</td>
<td>3.23</td>
<td>20.40</td>
<td>0.357</td>
</tr>
<tr>
<td>5d6 v 5d6</td>
<td>38.21</td>
<td>0.00</td>
<td>0.282</td>
</tr>
</tbody>
</table>

Figure 10.10: Player A rolls fewer dice to control bias.
by having Player A roll fewer dice. As we have seen, rolling one fewer dice is the best choice that leads to the smallest win bias, and having both players roll more dice also reduces the win bias, but decreases the closeness. Instead of having the players rolling different numbers of dice, we now will examine having the players roll different number of sides for the dice.

### 10.6 Reducing Bias With Mixed Dice

Another way we can reduce the bias towards Player A when they always win ties is to give Player B some dice with more sides. For example, we could have Player A roll 5d6 and have Player B roll 3d6 and 2d8, to give them a small advantage to help eliminate the advantage A receives for winning ties. Because bias does not occur when we allow ties, we will only examine using mixed dice for games where Player A wins ties.

#### 10.6.1 Mixed Dice Sorted, A Wins Ties

In Figure 10.12 we show the distribution of score differences for different mixes of d6 and d8 for B, while A always rolls 5d6. We can see that adding more d8 adjusts the bias in favor of B, but adding too many then biases B’s win rate too far. The most balanced position is to have B roll 2d6 and 3d8 against A’s 5d6 (solid line in the figure), with win bias of -2.24%.

Trying all possible mixes of 5 dice made of 6-sided, 8-sided, and 10-sided dice, and find that only 3 cases have a win minus loss bias under 10%; these cases are shown in Figure 10.13. The bias is still most balanced when Player B rolls 2d6 and 3d8 against Player A’s 5d6. However, by rolling 3d6/1d8/1d10, we can get a slight bias towards Player A, if that is desired.
Mixed Dice Rolled Sorted, A Wins Ties

<table>
<thead>
<tr>
<th>Game</th>
<th>win bias</th>
<th>tie %</th>
<th>closeness</th>
</tr>
</thead>
<tbody>
<tr>
<td>5d6/0d8</td>
<td>38.21</td>
<td>0.00</td>
<td>0.282</td>
</tr>
<tr>
<td>4d6/1d8</td>
<td>24.36</td>
<td>0.00</td>
<td>0.294</td>
</tr>
<tr>
<td>3d6/2d8</td>
<td>10.80</td>
<td>0.00</td>
<td>0.302</td>
</tr>
<tr>
<td>2d6/3d8</td>
<td>-2.24</td>
<td>0.00</td>
<td>0.305</td>
</tr>
<tr>
<td>1d6/4d8</td>
<td>-14.56</td>
<td>0.00</td>
<td>0.305</td>
</tr>
<tr>
<td>0d6/5d8</td>
<td>-25.98</td>
<td>0.00</td>
<td>0.301</td>
</tr>
</tbody>
</table>

Figure 10.12: Mixed d6 and d8 to control bias.

Mixed Dice Rolled Sorted, A Wins Ties

<table>
<thead>
<tr>
<th>Game</th>
<th>win bias</th>
<th>tie %</th>
<th>closeness</th>
</tr>
</thead>
<tbody>
<tr>
<td>3d6/1d8/1d10</td>
<td>2.67</td>
<td>0.00</td>
<td>0.307</td>
</tr>
<tr>
<td>2d6/3d8</td>
<td>-2.24</td>
<td>0.00</td>
<td>0.305</td>
</tr>
<tr>
<td>3d6/2d10</td>
<td>-5.36</td>
<td>0.00</td>
<td>0.310</td>
</tr>
</tbody>
</table>

Figure 10.13: Least biased mixes of d6, d8, and d10.

### 10.6.2 Mixed Dice Unsorted, A Wins Ties

We can do the same type of experiment for all variations of Player B rolling unsorted a mix of 5 d6s and d8s against Player A’s 5d6, getting the results as shown in Figure 10.14. By using 2d6 and 3d8, we can reduce the bias down to a small 1.61% in favor of Player B.

By trying all variations of 5 d6s, d8s, and d10s, we find that there are 5 cases where the bias is kept under 10%, which are shown in Figure 10.15. Rolling 2d6 and 3d8 is still the lowest overall bias; rolling 3d6/1d8/1d10 is the lowest bias that favors Player A.

In conclusion, we can reduce the win bias by having the unfavored player roll different sided dice. Looking at all mixes of five dice composed of d6, d8, and d10, we find that rolling 5d6 against 2d6/3d8 produced the smallest win bias, for both rolling sorted and unsorted. In fact, there is no way to completely eliminate the win bias. Nonetheless, we will examine one final way to break ties that will lead to a zero win bias.
10.7 Rerolling Tied Dice

We now examine rerolling tied dice as a final way to break ties. For example, it is quite common to reroll 1d6 at the start of a game to decide who goes first. This can be generalized to $ndk$, but it is quite cumbersome and this section exists mainly as an explanation on why we believe this is inadvisable in practice. Because rerolling can go on for many iterations, we use Monte Carlo simulation to evaluate the odds empirically instead of exactly, since these games can continue indefinitely with increasingly unlikely probability. We used $N=6^{10}=60,466,176$ simulations per game, as this is the same number of cases that evaluated for the other sections (see [95] for this calculation). When these are simulated and not exact values, we use the $\approx$ symbol in the figures.
10.7.1 Rolling Sorted, Rerolling Tied Dice

We first examine the case where we roll a handful of dice and then sort them from highest to lowest. Any dice that are not tied are scored first. Then, any remaining dice that are tied are rerolled by both players at once in a sub-game. This process is repeated for any remaining tied dice in the sub-game, until there are no more ties. All the scores from the first game and all sub-games are summed together for the final score.

The resulting score difference distributions are shown in Figure 10.16. The battles are all unbiased and without ties. For 5d4, 5d6, 5d8, and 5d10 the distributions are effectively flat with low closeness and have approximately the same shape as when rolling sorted with ties (as in Figure 10.2) but now do not permit tie games. Compared to 5d4 and higher, 5d2 has a higher closeness. However, this closeness comes at a significant cost of requiring
many rerolls, as demonstrated in Figure [10.17]. This shows that more sides decreases the probability of a reroll, and with 5d2 or 5d4 there are significant chances at rolling 2 or more rerolls for a single battle, which could be cumbersome for the players in practice. Higher sided dice are less likely to tie, so the probability of rerolling decreases quickly when using six or more sides.

We also examine the effect of changing the number of dice while holding the number of sides fixed in Figure [10.18]. The distributions are all flat, but closeness can be increased by using fewer dice, as we’ve seen in previous sections. The probability of rerolls is also affected by the number of dice, as shown in Figure [10.19]. For 1d6 and 2d6, the most common outcome is no rerolls. Increasing the number of dice makes rerolls more likely, but the probabilities of having additional rerolls decreases rapidly.
10.7.2 Rolling Unsorted, Rerolling Ties

Finally, we examine the case of rolling \( n \) \( k \)-sided dice unsorted when rerolling ties. The dice are rolled one at a time, and any time there is a tie, the two dice must be rerolled until they are no longer tied. This occurs for each of the \( n \) dice. In practice, this is unlikely to be much fun for the players, but we present the analysis here for completeness.

Because neither player is favored, the metrics can be analytically calculated from the binomial distribution \( \binom{n}{w} p^w (1-p)^{n-w} \) with \( w \) being the number of wins for Player A in the battle, \( n \) dice rolled, and probability \( p = .5 \) (no matter the value of \( k \)) of Player A winning each point. Given a score difference \( d \), we can calculate \( w = (n + d)/2 \).

In Figure [10.20] we see that the score difference distribution is identical for all dice, no matter how many sides. The battle is unbiased, with no ties, and has a closeness of .447. However, they do not have the same number of rerolls, as shown in Figure [10.21] generated with Monte Carlo simulation. To reduce rerolls, the designer can use higher sided dice.

The closeness can be increased by reducing the number of dice rolled, as shown in Figure [10.22]. Rolling fewer dice also reduces the probability of rerolls, as shown in Figure [10.23].

10.7.3 Sorted, A Wins Ties, Rerolls Highest

We can make a hybrid case, where A wins all ties but must reroll when A rolls the die’s highest value (e.g. a 6 on a 6-sided die). This effectively means that Player A is rolling a \( k - 1 \) sided die while Player B is rolling a \( k \) sided die. This gives an advantage to Player B to make up for the advantage that Player A has when breaking ties.

Interestingly, this has the same effect as in the previous reroll sections, for both rolling sorted or unsorted. Therefore, the plots are the same as in Figures [10.16], [10.18], [10.20], and [10.22]. However, only Player A has to reroll dice, and Player B can keep the dice untouched no matter what they roll. Therefore, there are many less rerolls in total.

We can show analytically why this is unbiased for the simple case of one \( k \)-sided die. Player A will reroll when rolling a \( k \), which is the same as rolling a \( k - 1 \) sided die. If Player A rolls a value of \( i \) with probability \( 1/(k - 1) \), then they win when Player B rolls a value \( \leq i \) with probability \( i/k \), since A wins ties. Calculating the expected number of wins for Player A, over all values of \( i \) from 1 to \( k - 1 \) we have:

\[
\sum_{i=1}^{k-1} \frac{1}{k-1} \frac{i}{k} = \frac{1}{k(k-1)} \frac{(k-1)(k)}{2} = \frac{1}{2}
\]  
(10.1)
which is independent of $k$, and always $1/2$.

When breaking ties by rerolling, we get unbiased results at the cost of requiring players to reroll, which can take longer. By using higher sided dice or fewer dice, the designer can mitigate the expected number of rerolls. Because other tie-breaks presented in this chapter do not require extra rolls, we suggest following another approach to breaking ties.

### 10.8 Risk & Risk 2210 A.D.

We can use the results of this chapter to examine how the original Risk compares with the popular variant Risk 2210 A.D. In both games, the players roll sorted dice and the defender wins tied dice, which we showed in Section 10.4.1 gives a strong advantage to the defender when rolling the same number of dice. A game with the defender having an
advantage can lead to a static game where neither player wants to attack.

To counteract this, both games allow the attacker to roll an extra die (3d6 v 2d6). We show in Section 10.5 this flips the advantage towards the attacker. This advantage encourages players to play more aggressively, as it's better to be the attacker than the defender.

In Risk 2210 A.D. special units called commanders and space stations will swap in one or more d8 instead of d6 when engaging in battles. As we showed in Section 10.6, using mixed dice biases wins towards the player rolling higher valued dice, which can be either be used by attackers to have a stronger advantage (less closeness but more predictability) or by defenders to even out the bias inherent in letting the attacker roll more dice.

As we’ve shown in this chapter, the rules in dice games require careful balancing as the exact number of dice and number of sides can often have a large impact on the statistical
outcome of the battles. Risk and Risk 2210 A.D. are no exception and they appear to have carefully tuned dice mechanics to have reasonable win bias and closeness values.

10.9 Conclusions

We have demonstrated the use of *win bias, tie percentage, and closeness* to analyze a collection of dice battle variants for use as a component in a larger game. We introduce closeness, which is related to the precision statistic about 0, and matches the intuitive concept of a game being close. We have not seen this statistic used before to analyze games.

By examining the results of the previous sections, we can make some general statements about this category of dice battles where the number values are compared.

In Section 10.2 we showed that when allowing ties, rolling dice unsorted results in higher closeness, and therefore a lower chance of games with large point differences; however, this comes at the cost of increasing the tie percentage. Using fewer sides on the dice increases closeness, but also increases the tie percentage. Using fewer dice increases closeness, but again generally increases the tie percentage.

Battles that end tied with a score difference of 0 can be broken with a coin flip or other 50/50 random event, as discussed in Section 10.3. However, we also wanted to explore rule changes that would cause odd-numbers of dice to never end in a tie. Breaking ties in favor of one player, as shown in Section 10.4, eliminates ties but creates a large win bias, although this can be reduced with more sides on the dice. This bias occurs for both rolling sorted and unsorted, although rolling unsorted results in higher closeness and slightly lower win bias.

To reduce this win bias, in Section 10.5 the favored player rolls fewer dice. Rolling one fewer die is the best choice that leads to the smallest win bias, and having both players roll more dice also reduces the win bias (but decreases closeness). In Section 10.6 we reduce the win bias by having the unfavored player roll different sided dice. Looking at all mixes of five dice composed of d6, d8, and d10, rolling 5d6 against 2d6/3d8 produces the smallest win bias, for both rolling sorted and unsorted. However, there is no way to completely eliminate the win bias. Finally, in Section 10.7 we break ties by rerolling. This gives unbiased results, but at the cost of requiring the players to reroll, which can take longer. By using higher sided dice or fewer dice, one can reduce the expected number of rerolls, although we recommend other tie breaking methods that are less cumbersome for the players.

One surprising outcome of this study is that *nd2* sorted with ties may be an under-used dice mechanic for games. This has high closeness, and can be easily done by throwing 2-sided coins or stick dice, subtracting the number of heads from the tails. Stick dice with flat edges do not roll off the table easily.
Additionally, for finer grained control over the experience, a game can instead use a bag of dice tokens (e.g. small cardboard chits with a dice face printed on them) or a deck of dice cards to enforce that certain distributions are obeyed with local representation – this is choosing without replacement instead of the typical choosing with replacement that occurs with dice.

In summary, there is no perfect solution to the dice battle mechanic, and a designer must make a series of trade-offs. This chapter can provide some quantitative guidance to a designer looking for a specific type of game feel when using dice. For a designer that wishes to use rules that we did not discuss in this chapter, we expect it would not be difficult to use the same technique to evaluate how the players might experience the distribution of score differences by measuring win biases, ties, and closeness.
Chapter 11

Conclusions

11.1 Confirmation of the Thesis Statement

This dissertation has demonstrated the validity of its thesis statement:

*The analysis of score distributions, combined with artificial intelligence, simulation, player modeling, and human game play data, is useful for understanding the characteristics of games and quantitatively exploring the space of possible games.*

- **Analysis of score distributions**: We have demonstrated several techniques for understanding how scores are distributed in games, including survival analysis, probability theory, calculus, exhaustive simulation of dice rolls, and analytical proofs. We have shown the utility of metrics such as score expected value, win rate, tie percentages, closeness, and maximal values for high score analysis.

- **Artificial intelligence**: We have explored various applications of simulation and artificial intelligence, including procedural content generation, tree search, genetic programming, Differential Evolution optimization, and clustering.

- **Game play simulation**: We have demonstrated techniques for rapidly generating score data using AI agents that are intended to play orders of magnitude faster than human players, in repeated, controlled experiments that would not be feasible with human players.

- **Player modeling**: We have explained how to incorporate models of human motor skill, cognitive limitations, and learning into AI agents for simulation of human
behavior. Specifically, we have focused on timing accuracy, aiming accuracy, strategic thinking, inequity aversion, and power-law learning models.

- **Human game play metrics**: We have shown how human game play data, specifically from *Flappy Bird*, *Canabalt*, and *Drop7*, can be used to understand learning rates, to quantify human error, and to validate perceived difficulty of games.

- **Characteristics of games**: We provided quantitative techniques for measuring various game characteristics, including length of playtime, heuristics, rules, outcomes, ending conditions, catch-up, game balance, randomness, luck, skill, and costs for players.

- **Exploring game space**: We have demonstrated various ways to explore what games are possible within a particular region of game space, both to better understand specific games, games in general, and to discover new games. We use Monte Carlo simulation, computational creativity, sampling, visualization, search, and mathematical modeling.

### 11.2 Future Work

While we have demonstrated that these score analysis and AI-assisted techniques are suitable for exploring game space for minimal action games (Chapters 4 and 8), two-player combinatorial games (Chapter 9), two-player dice games (Chapter 10), and interactive puzzle games (Chapter 5), there is significant follow on work to extend these techniques to more complicated games and new genres. Popular genres that combine strategy and dexterity include real-time strategy games, dungeon crawlers, 2D and 3D platformers, and first person shooters. These would each pose unique challenges, including a much larger game state space and likely a need for more complicated AI agents. While much progress is being made in machine learning for playing real-time strategy games [161] the general focus of that type of research is still trying to outperform human players, not to try and model human-like fallibility and error. With first-person shooters and real-time strategy games, many AI bots try to play like humans to avoid being banned from multiplayer servers, so these in particular seem like promising areas to explore similar AI-assisted techniques.

While the algorithms presented in this thesis are intended to be a framework that can be integrated into a game design process, we did not attempt to make these into accessible general purpose tools. The frameworks we developed currently require that they have direct access to the code, have an AI that is integrated for playing the game as the player, and are thus best implemented by the game programmers and game designers. We imagine that future work could examine tools that play the game without linking into the code, perhaps
by reading the output of the screen and using the same controls as human players, such as in the OpenAI Gym framework [31]. General purpose playtesting tools would be a significant step forward for AI-assisted game space exploration, and we look forward to continued research in this area.

AI-generated novice-level heuristics for human play, as discussed in Sec. 9.5.5, is further explored for the game of Blackjack [198] but this remains a promising area of research. In particular, generating heuristics from a base primitive set using subroutines, co-evolving strategies to model meta-games among player communities, and using levels of heuristics to evaluate the quality of a game all seem within reach, as described in that section.

A major area related to game difficulty that was unexplored in this dissertation is how easy it is for players to understand the game rules and game state. Game designers want their games to be easily comprehensible to users, and AI-assisted techniques can potentially help point out areas where games are confusing to players. At the same time, game designers intentionally obscure game details to provide challenge to players. For example, intentionally making a game less accessible can be done with visual obfuscation, in the case of hiding clues or secret doors in RPGs, making matches hard to find in tile-swappers, or encoding weaknesses about enemies in their visual appearance or motion characteristics. Understanding and modeling of the human perceptual system at a high level will be critical for advancing player modeling in this direction.

A major challenge in computational creativity is creating very high quality content that rivals the content of the best human authors and designers. Many people have successfully created endless amounts of content for games like Spelunky or Rogue [226] where procedural content generation is used to randomize the game so the player does not know what is coming. These types of generators need to have a predictable quality, but the sheer volume of content they generate is not necessarily desireable [46]. That is, the point of these generators is not to find one or two great, timeless examples as we expect from the very best human artists. The ability to critique the mass of generated art to find the best work is far more difficult. In this thesis, we too have the ability to create endless variants of Flappy Bird, but its only the unique, interesting ones that are worth it for a human to play. High-quality procedurally generated content is still rare, yet it certainly is possible as at least one skilled expert author has used PCG to generate expert quality novel content [197]. A focus on producing and identifying great work requires models to understand and quantify what makes great work. This likely requires further collaboration with experts in the AI field as well as experts at creating content, to truly understand the design process from those best at design.

A final area that could use significant further research is combining these AI-assisted playtesting techniques with human playtesters in an integrated system. Further validation
that humans are experiencing the game the way that the AI predicts is critical, and more research needs to be done on merging more expensive user testing with cheaper simulation.

11.3 Final Thoughts

At this moment in the field of artificial intelligence, machines can play most games better than humans can, and the general consensus among researchers is that in the near future AI agents will be able to play all games better than humans. Combinatorial and perfect information games (like Chess, Checkers, and Go) and imperfect information games (like Poker) are dominated by machines. With the advent of natural language processing, machines are already champions at word games like Scrabble [196], trivia games such as Jeopardy [69], and are approaching master level at Crossword Puzzles [233].

Humans are still better at games that require human creativity and misdirection, such as Dixit [185], and it will be interesting to research champion-level AI for these more human-emotion based games. Like the AI agents in this thesis, games that focus on human storytelling and human emotion must have a model of how humans think, act, and play. These games won’t be beaten by just exploring game trees or querying massive databases faster than a human can think, but will require more sophisticated player modeling that understands (or at least seems to understand) what the human player will do next. The fact that machines have already learned to bluff better than any human is a possible indication that AI agents will be able to make even more progress into modeling human thought, at least in order to simulate it well enough to beat human players at games. Whether machines will be able to beat humans at the grand prize of games, Turing’s Imitation Game (a.k.a the Turing Test) [230], remains to be seen, but it would not surprise me if the answer is yes.

Advances in the analysis of scores, player modeling, and artificial intelligence can quantifiably improve games and our understanding of games. In the short term, I believe this work can help shift the focus of developing expert AIs that can unequivocally beat human players into AIs that are designed to assist humans operating in the game domain, either as designers or players. One aspect is for creating more ideal opponents, providing just the right amount of challenge for the enjoyment, growth, and advancement of human players. Another aspect is to enhance the design process by automating the more laborious parts of design space exploration. But thinking bigger, the most exciting aspect is to find new games that humans have not yet thought about. Certainly there is no reason why the game genres we have today are the only game genres that humans will discover. The challenge is finding new ways to find those games – whether they are discovered by humans, by machines, or by humans and machines working together.
Bibliography


