Introduction to GAME DESIGN, PROTOTYPING, and DEVELOPMENT
Third Edition

Jeremy Gibson BOND
Foreword by Richard Lemarchand

FREE SAMPLE CHAPTER
Praise for the Second Edition

"When teaching about game design and development, you often get asked the dreaded question: 'Where can I learn all this?' *Introduction to Game Design, Prototyping, and Development* has been my deliverance, as it provides a one-stop solution and answer. This book is quite unique in covering in-depth both game design and development: it embraces and exemplifies the idea that design, prototyping, development, and balancing combine in an iterative process. By sending the message that creating games is both complex and feasible, I believe this to be a great learning tool; and the new edition with even more detailed examples seems even better."

— Pietro Polsinelli, Applied Game Designer at Open Lab

"Introduction to Game Design, Prototyping, and Development has truly helped me in my game development journey and has opened my mind to many helpful techniques and practices. This book not only contains a full introduction to the C# language, but also includes information about playtesting, game frameworks, and the game industry itself. Jeremy is able to explain complex concepts in a way that is very informative and straightforward. I have also found the prototype tutorials to be useful and effective for developing good programming practices. I would highly recommend this book to anyone looking to learn game development from scratch, or simply brush up on their skills. I look forward to using it as a guide and reference for future projects."

— Logan Sandberg, Pinwheel Games & Animation

"Jeremy's approach to game design shows the importance of prototyping game rules and prepares the readers to be able to test their own ideas. Being able to create your own prototypes allows for rapid iteration and experimentation, and makes better Game Designers."

— Juan Gril, Executive Producer, Flowplay

"Introduction to Game Design, Prototyping, and Development combines the necessary philosophical and practical concepts for anyone looking to become a Game Designer. This book will take you on a journey from high-level design theories, through game development concepts and programming foundations. I regularly recommend this book to any aspiring game designers who are looking to learn new skills or strengthen their design chops. Jeremy uses his years of experience as a professor to teach you how to think with vital game design mindsets so that you can create a game with all the right
tools at hand. Regardless of how long you've been in the games industry, you're bound to find inspirational ideas that will help you improve your design process. I'm personally excited to dive into the updates in this latest edition and get a refresher course on some of the best practices for creating amazing games!

—Michelle Pun, Game Producer at Osmo, former Lead Game Designer at Disney and Zynga

"I used Professor Bond’s book to teach myself how to code in C# and familiarize myself with Unity. Since then I have used the book as the backbone for my high school Digital Game Design class. The programming lessons are top-notch, the prototypes clearly demonstrate the myriad facets of programming and how those are used to create recognizable game mechanics, and the prototypes are easily adapted for student personalization. I can't wait to get hold of the second edition and begin using it in my classroom."

—Wesley Jeffries, Game Design Teacher, Riverside Unified School District

"With the latest edition of Introduction to Game Design, Prototyping, and Development, Bond builds on the solid foundation of the first. The new edition adds new content throughout the book, with updated examples and topics across all the chapters. This is a thorough and thoughtful exploration of the process of making games."

—Drew Davidson, Director, Entertainment Technology Center at Carnegie Mellon University

"If you want to take your game development to the next level, this book is a must! Not only does it give you a lot of game examples from beginning to end, it also—and this is the most important part—makes you think like a game designer. What makes a game fun and engaging? What makes a player come back to your game over and over again? The answers are all here. This book gives you a lot more than a couple of online tutorials can give you. It gives you the whole picture!"

—David Lindskog, Founder, Monster Grog Games
Introduction to Game Design, Prototyping, and Development
Introduction to Game Design, Prototyping, and Development

From Concept to Playable Game with Unity and C#

Jeremy Gibson Bond

Addison-Wesley
PEARSON'S COMMITMENT TO DIVERSITY, EQUITY, AND INCLUSION

Pearson is dedicated to creating bias-free content that reflects the diversity of all learners. We embrace the many dimensions of diversity, including but not limited to race, ethnicity, gender, socioeconomic status, ability, age, sexual orientation, and religious or political beliefs.

Education is a powerful force for equity and change in our world. It has the potential to deliver opportunities that improve lives and enable economic mobility. As we work with authors to create content for every product and service, we acknowledge our responsibility to demonstrate inclusivity and incorporate diverse scholarship so that everyone can achieve their potential through learning. As the world's leading learning company, we have a duty to help drive change and live up to our purpose to help more people create a better life for themselves and to create a better world.

Our ambition is to purposefully contribute to a world where

- Everyone has an equitable and lifelong opportunity to succeed through learning
- Our educational products and services are inclusive and represent the rich diversity of learners
- Our educational content accurately reflects the histories and experiences of the learners we serve
- Our educational content prompts deeper discussions with learners and motivates them to expand their own learning (and worldview)

While we work hard to present unbiased content, we want to hear from you about any concerns or needs with this Pearson product so that we can investigate and address them.

Please contact us with concerns about any potential bias at https://www.pearson.com/report-bias.html.
This book is dedicated to:

My wife Melanie, the love of my life, 
for her love, intellect, and support

My son Jordan and godson Phoenix, 
whom I hope to make games with one day

My parents and sisters

My friend and mentor Mike Sellers, 
a brilliant designer and professor

And my many professors, colleagues, and students 
who inspired me to write this book
## Contents at a Glance

### Foreword
- xxv

### Preface
- xxviii

### Acknowledgments
- xli

### About the Author
- xliii

### PART I: Game Design and Paper Prototyping

<table>
<thead>
<tr>
<th>Number</th>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Thinking Like a Designer</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>Game Analysis Frameworks</td>
<td>21</td>
</tr>
<tr>
<td>3</td>
<td>The Layered Tetrad</td>
<td>33</td>
</tr>
<tr>
<td>4</td>
<td>The Inscribed Layer</td>
<td>41</td>
</tr>
<tr>
<td>5</td>
<td>The Dynamic Layer</td>
<td>67</td>
</tr>
<tr>
<td>6</td>
<td>The Cultural Layer</td>
<td>89</td>
</tr>
<tr>
<td>7</td>
<td>Acting Like a Designer</td>
<td>103</td>
</tr>
<tr>
<td>8</td>
<td>Design Goals</td>
<td>129</td>
</tr>
<tr>
<td>9</td>
<td>Paper Prototyping</td>
<td>151</td>
</tr>
<tr>
<td>10</td>
<td>Game Testing</td>
<td>167</td>
</tr>
<tr>
<td>11</td>
<td>Math and Game Balance</td>
<td>187</td>
</tr>
<tr>
<td>12</td>
<td>Guiding the Player</td>
<td>231</td>
</tr>
<tr>
<td>13</td>
<td>Puzzle Design</td>
<td>247</td>
</tr>
<tr>
<td>14</td>
<td>The Agile Mentality</td>
<td>265</td>
</tr>
<tr>
<td>15</td>
<td>The Digital Game Industry</td>
<td>287</td>
</tr>
</tbody>
</table>

### PART II: Programming C# in Unity

<table>
<thead>
<tr>
<th>Number</th>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>16</td>
<td>Thinking in Digital Systems</td>
<td>311</td>
</tr>
<tr>
<td>17</td>
<td>Introducing Unity Hub and the Unity Editor</td>
<td>323</td>
</tr>
<tr>
<td>18</td>
<td>Introducing Our Language: C#</td>
<td>347</td>
</tr>
<tr>
<td>19</td>
<td>Hello World: Your First Program</td>
<td>359</td>
</tr>
<tr>
<td>20</td>
<td>Variables and Components</td>
<td>383</td>
</tr>
<tr>
<td>21</td>
<td>Boolean Operations and Conditionals</td>
<td>405</td>
</tr>
<tr>
<td>-----</td>
<td>--------------------------------------------------------</td>
<td>-----</td>
</tr>
<tr>
<td>22</td>
<td>Loops</td>
<td>423</td>
</tr>
<tr>
<td>23</td>
<td>Collections in C#</td>
<td>437</td>
</tr>
<tr>
<td>24</td>
<td>Functions and Parameters</td>
<td>473</td>
</tr>
<tr>
<td>25</td>
<td>Debugging</td>
<td>493</td>
</tr>
<tr>
<td>26</td>
<td>Classes</td>
<td>521</td>
</tr>
<tr>
<td>27</td>
<td>Object-Oriented Thinking</td>
<td>539</td>
</tr>
<tr>
<td>28</td>
<td>Data-Oriented Design</td>
<td>575</td>
</tr>
</tbody>
</table>

**PART III** Game Prototype Tutorials ........................... 619

<table>
<thead>
<tr>
<th>29</th>
<th>Apple Picker</th>
<th>621</th>
</tr>
</thead>
<tbody>
<tr>
<td>30</td>
<td>Mission Demolition</td>
<td>681</td>
</tr>
<tr>
<td>31</td>
<td>Space SHMUP — Part 1</td>
<td>753</td>
</tr>
<tr>
<td>32</td>
<td>Space SHMUP — Part 2</td>
<td>807</td>
</tr>
<tr>
<td>33</td>
<td>Prospector Solitaire — Part 1</td>
<td>897</td>
</tr>
<tr>
<td>34</td>
<td>Prospector Solitaire — Part 2</td>
<td>969</td>
</tr>
<tr>
<td>35</td>
<td>Dungeon Delver — Part 1</td>
<td>1019</td>
</tr>
<tr>
<td>36</td>
<td>Dungeon Delver — Part 2</td>
<td>1095</td>
</tr>
</tbody>
</table>

**PART IV** Next Steps ........................................... 1187

<table>
<thead>
<tr>
<th>37</th>
<th>Coding Challenges</th>
<th>1197</th>
</tr>
</thead>
<tbody>
<tr>
<td>38</td>
<td>Beyond This Book</td>
<td>1197</td>
</tr>
</tbody>
</table>

**PART V** Online Appendices — http://book.prototools.net or informit.com/title/9780136619949

A  Standard Project Setup Procedure

B  Useful Concepts

C  Online Reference

D  Tips for Teaching from This Book
# Contents

## Foreword
xxv

## Preface
xxviii

## Acknowledgments
xli

## About the Author
xliii

## PART I  Game Design and Paper Prototyping

### 1  Thinking Like a Designer
3

- You Are a Game Designer
- Bartok: A Game Design Exercise
- The Definition of *Game*
- Summary

### 2  Game Analysis Frameworks
21

- Common Frameworks for Ludology
- MDA: Mechanics, Dynamics, and Aesthetics
- Formal, Dramatic, and Dynamic Elements
- The Elemental Tetrad
- Summary

### 3  The Layered Tetrad
33

- The Inscribed Layer
- The Dynamic Layer
- The Cultural Layer
- The Responsibility of the Designer
- Summary

### 4  The Inscribed Layer
41

- Inscribed Mechanics
- Inscribed Aesthetics
CONTENTS

Inscribed Narrative ............................................ 54
Inscribed Technology .......................................... 65
Summary .......................................................... 66

5 The Dynamic Layer ........................................... 67
The Role of the Player ......................................... 68
Emergence ......................................................... 69
Dynamic Mechanics .......................................... 70
Dynamic Aesthetics ............................................ 77
Dynamic Narrative ........................................... 85
Dynamic Technology .......................................... 88
Summary .......................................................... 88

6 The Cultural Layer ............................................ 89
Beyond Play ....................................................... 90
Cultural Mechanics .......................................... 91
Cultural Aesthetics ............................................ 93
Cultural Narrative ............................................ 93
Cultural Technology ......................................... 95
Authorized Transmedia Are Not Part of the Cultural Layer. ....... 96
The Cultural Impact of a Game ............................. 97
Summary .......................................................... 101

7 Acting Like a Designer ....................................... 103
Iterative Design ................................................ 104
Innovation ......................................................... 112
Brainstorming and Ideation .................................. 113
Changing Your Mind ......................................... 117
Scoping! .......................................................... 120
Summary .......................................................... 127
# Table of Contents

8 Design Goals .............................................. 129  
  Design Goals: An Incomplete List ......................... 130  
  Designer-Centric Goals .................................. 131  
  Player-Centric Goals .................................... 134  
  Summary ............................................... 150  

9 Paper Prototyping ....................................... 151  
  The Benefits of Paper Prototyping ......................... 152  
  Paper Prototyping Tools .................................. 153  
  Paper Prototyping for Interfaces ......................... 156  
  A Paper Prototype Example ............................... 157  
  Best Uses for Paper Prototyping ......................... 162  
  Poor Uses for Paper Prototyping ......................... 163  
  Summary ............................................... 164  

10 Game Testing ........................................... 167  
  Why Playtest? .......................................... 168  
  Being a Great Playtester Yourself ......................... 168  
  The Circles of Playtesters ................................ 169  
  Methods of Playtesting .................................. 172  
  Other Important Types of Testing ......................... 183  
  Summary ............................................... 185  

11 Math and Game Balance ............................... 187  
  The Meaning of Game Balance ............................ 188  
  The Importance of Spreadsheets ........................... 188  
  Examining Dice Probability with Sheets .................. 190  
  The Math of Probability ................................ 207  
  Randomizer Technologies in Paper Games .................. 212  
  Weighted Distributions ................................ 215  
  Weighted Probability in Google Sheets .................... 216  
  Permutations .......................................... 217
<table>
<thead>
<tr>
<th>Chapter</th>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>12</td>
<td>Guiding the Player</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Direct Guidance</td>
<td>232</td>
</tr>
<tr>
<td></td>
<td>Indirect Guidance</td>
<td>234</td>
</tr>
<tr>
<td></td>
<td>Teaching New Skills and Concepts</td>
<td>242</td>
</tr>
<tr>
<td></td>
<td>Summary</td>
<td>245</td>
</tr>
<tr>
<td>13</td>
<td>Puzzle Design</td>
<td>247</td>
</tr>
<tr>
<td></td>
<td>Scott Kim on Puzzle Design</td>
<td>248</td>
</tr>
<tr>
<td></td>
<td>The Steps of Solving a Puzzle</td>
<td>257</td>
</tr>
<tr>
<td></td>
<td>Puzzle Examples in Action Games</td>
<td>260</td>
</tr>
<tr>
<td></td>
<td>Designing and Developing Puzzle Games</td>
<td>262</td>
</tr>
<tr>
<td></td>
<td>Summary</td>
<td>263</td>
</tr>
<tr>
<td>14</td>
<td>The Agile Mentality</td>
<td>265</td>
</tr>
<tr>
<td></td>
<td>The Manifesto for Agile Software Development</td>
<td>266</td>
</tr>
<tr>
<td></td>
<td>Scrum Methodology</td>
<td>267</td>
</tr>
<tr>
<td></td>
<td>Burndown Chart Example</td>
<td>271</td>
</tr>
<tr>
<td></td>
<td>Creating Your Own Burndown Charts</td>
<td>285</td>
</tr>
<tr>
<td></td>
<td>Summary</td>
<td>286</td>
</tr>
<tr>
<td>15</td>
<td>The Digital Game Industry</td>
<td>287</td>
</tr>
<tr>
<td></td>
<td>About the Game Industry</td>
<td>288</td>
</tr>
<tr>
<td></td>
<td>Game Education</td>
<td>296</td>
</tr>
<tr>
<td></td>
<td>Getting Into the Industry</td>
<td>299</td>
</tr>
<tr>
<td></td>
<td>Don’t Wait to Start Making Games!</td>
<td>305</td>
</tr>
<tr>
<td></td>
<td>Summary</td>
<td>308</td>
</tr>
</tbody>
</table>
# Contents

## PART II Programming C# in Unity

<table>
<thead>
<tr>
<th>Chapter</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>16</td>
<td>Thinking in Digital Systems</td>
<td>311</td>
</tr>
<tr>
<td></td>
<td>Systems Thinking in Board Games</td>
<td>312</td>
</tr>
<tr>
<td></td>
<td>An Exercise in Simple Instructions</td>
<td>313</td>
</tr>
<tr>
<td></td>
<td>Game Analysis: <em>Apple Picker</em></td>
<td>316</td>
</tr>
<tr>
<td></td>
<td>Summary</td>
<td>322</td>
</tr>
<tr>
<td>17</td>
<td>Introducing Unity Hub and the Unity Editor</td>
<td>323</td>
</tr>
<tr>
<td></td>
<td>Downloading Unity</td>
<td>324</td>
</tr>
<tr>
<td></td>
<td>Introducing Our Development Environment</td>
<td>327</td>
</tr>
<tr>
<td></td>
<td>Creating a Unity Account</td>
<td>334</td>
</tr>
<tr>
<td></td>
<td>Checking Out a Sample Project</td>
<td>335</td>
</tr>
<tr>
<td></td>
<td>Creating Your First Unity Project</td>
<td>335</td>
</tr>
<tr>
<td></td>
<td>Learning Your Way Around Unity</td>
<td>338</td>
</tr>
<tr>
<td></td>
<td>Setting Up the Unity Window Layout</td>
<td>339</td>
</tr>
<tr>
<td></td>
<td>Summary</td>
<td>345</td>
</tr>
<tr>
<td>18</td>
<td>Introducing Our Language: C#</td>
<td>347</td>
</tr>
<tr>
<td></td>
<td>Understanding the Features of C#</td>
<td>348</td>
</tr>
<tr>
<td></td>
<td>Reading and Understanding C# Syntax</td>
<td>355</td>
</tr>
<tr>
<td></td>
<td>Summary</td>
<td>358</td>
</tr>
<tr>
<td>19</td>
<td>Hello World: Your First Program</td>
<td>359</td>
</tr>
<tr>
<td></td>
<td>Creating a New Project</td>
<td>360</td>
</tr>
<tr>
<td></td>
<td>Making a New C# Script</td>
<td>363</td>
</tr>
<tr>
<td></td>
<td>Making Things More Interesting</td>
<td>370</td>
</tr>
<tr>
<td></td>
<td>Summary</td>
<td>382</td>
</tr>
<tr>
<td>20</td>
<td>Variables and Components</td>
<td>383</td>
</tr>
<tr>
<td></td>
<td>Introducing Variables</td>
<td>384</td>
</tr>
<tr>
<td></td>
<td>Statically Typed Variables in C#</td>
<td>384</td>
</tr>
<tr>
<td></td>
<td>Important C# Variable Types</td>
<td>386</td>
</tr>
</tbody>
</table>
The Scope of Variables .................. 389
Naming Conventions .................. 389
Important Unity Variable Types .......... 390
Unity GameObjects and Components .......... 399
Summary .................................. 403

21 Boolean Operations and Conditionals .......... 405
Booleans ................................ 406
Comparison Operators ................... 410
Conditional Statements .................. 416
Summary .................................. 422

22 Loops .................................. 423
Types of Loops .......................... 424
while Loops ............................. 425
do...while Loops ......................... 429
for Loops ................................ 429
foreach Loops ........................... 433
Jump Statements within Loops ............... 433
Summary .................................. 436

23 Collections in C# ....................... 437
C# Collections ........................... 438
Using Generic Collections ................. 442
List<T> .................................. 443
Dictionary<TKey, TValue> ................. 447
Array .................................... 451
Multidimensional Arrays .................. 457
Jagged Arrays ........................... 461
Jagged List<T>s ........................... 465
Choosing Whether to Use an Array or List ........ 466
Summary .................................. 467
## Contents

The Future of Unity DOTS ........................................ 617
Summary ....................................................... 618

### PART III

**Game Prototype Tutorials** ........................................ 619

#### 29 Apple Picker .................................................. 621
What You Will Learn ............................................ 622
The Apple Picker Prototype ...................................... 622
The Purpose of a Digital Prototype ............................. 623
Preparing ................................................................ 624
Coding the Apple Picker Prototype ............................ 637
GUI and Game Management ...................................... 661
Summary ................................................................ 678

#### 30 Mission Demolition ............................................. 681
What You Will Learn ............................................ 682
The Mission Demolition Prototype ............................ 682
Getting Started: Mission Demolition ......................... 683
Game Prototype Concept ........................................ 683
Art Assets ................................................................ 684
Coding the Prototype ............................................. 691
From Prototype to First Playable ............................... 736
Summary ................................................................ 751

#### 31 Space SHMUP — Part 1 ....................................... 753
What You Will Learn ............................................ 754
Getting Started: Space SHMUP ............................... 755
Setting the Scene ................................................. 757
Making the Hero Ship ............................................. 758
Adding Some Enemies ............................................ 771
Spawning Enemies at Random ................................. 787
Setting Tags, Layers, and Physics .............................. 790
Making the Enemies Damage the Player .................... 792
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Restarting the Game</td>
<td>797</td>
</tr>
<tr>
<td>Shooting (Finally)</td>
<td>800</td>
</tr>
<tr>
<td>Summary</td>
<td>805</td>
</tr>
<tr>
<td><strong>32</strong> Space SHMUP – Part 2</td>
<td>807</td>
</tr>
<tr>
<td>What You Will Learn</td>
<td>808</td>
</tr>
<tr>
<td>Getting Started: Space SHMUP – Part 2</td>
<td>809</td>
</tr>
<tr>
<td>Enemy to Enemy_0</td>
<td>810</td>
</tr>
<tr>
<td>Programming Other Enemies</td>
<td>811</td>
</tr>
<tr>
<td>Shooting Revisited</td>
<td>833</td>
</tr>
<tr>
<td>Showing Enemy Damage</td>
<td>853</td>
</tr>
<tr>
<td>Adding PowerUps and Boosting Weapons</td>
<td>857</td>
</tr>
<tr>
<td>Race Conditions &amp; Script Execution Order</td>
<td>869</td>
</tr>
<tr>
<td>Making Enemies Drop PowerUps</td>
<td>872</td>
</tr>
<tr>
<td>Enemy_4 — A More Complex Enemy</td>
<td>876</td>
</tr>
<tr>
<td>Tuning Settings for the Game Entities</td>
<td>888</td>
</tr>
<tr>
<td>Adding a Scrolling Starfield Background</td>
<td>890</td>
</tr>
<tr>
<td>Summary</td>
<td>893</td>
</tr>
<tr>
<td><strong>33</strong> Prospector Solitaire – Part 1</td>
<td>897</td>
</tr>
<tr>
<td>What You Will Learn</td>
<td>898</td>
</tr>
<tr>
<td>The Prospector Game</td>
<td>899</td>
</tr>
<tr>
<td>Getting Started: Prospector Solitaire</td>
<td>901</td>
</tr>
<tr>
<td>Build Settings</td>
<td>902</td>
</tr>
<tr>
<td>Setting Up the Unity Window Layout</td>
<td>906</td>
</tr>
<tr>
<td>Setting Up the Camera and Game Pane</td>
<td>906</td>
</tr>
<tr>
<td>Importing Images as Sprites</td>
<td>907</td>
</tr>
<tr>
<td>Constructing Cards from Sprites</td>
<td>911</td>
</tr>
<tr>
<td>Implementing Prospector in Code</td>
<td>940</td>
</tr>
<tr>
<td>Implementing Game Logic</td>
<td>961</td>
</tr>
<tr>
<td>Summary</td>
<td>968</td>
</tr>
</tbody>
</table>
## Contents

### 34 Prospector Solitaire – Part 2
- What You Will Learn .................................................. 970
- Getting Started: Prospector – Part 2 .......................... 971
- Additional Prospector Game Elements ......................... 972
- Adding GUI Elements to Display the Score .................. 985
- Building and Running Your WebGL Build .................... 1013
- Summary .................................................................. 1016

### 35 Dungeon Delver – Part 1
- What You Will Learn .................................................. 1020
- The Dungeon Delver Game ......................................... 1021
- Getting Started: Dungeon Delver ............................... 1022
- Setting Up the Cameras ............................................. 1023
- Understanding the Dungeon Data ............................... 1026
- Showing the Map with a Unity Tilemap ......................... 1031
- Adding the Hero ...................................................... 1042
- Giving Dray an Attack Animation ............................... 1055
- Dray’s Sword .......................................................... 1059
- Programmatic Collision in Unity Tilemap ...................... 1061
- The InRoom Script ................................................... 1070
- Enemy: Skeletos ...................................................... 1072
- Keeping GameObjects in the Room .............................. 1075
- Aligning to the Grid .................................................. 1078
- Moving from Room to Room ...................................... 1087
- Making the Camera Follow Dray ................................. 1091
- Summary .................................................................. 1094

### 36 Dungeon Delver – Part 2
- What You Will Learn .................................................. 1096
- Getting Started: Dungeon Delver – Part 2 .................... 1097
- Dungeon Delver – Part 2 Overview .............................. 1098
- Implementing TileSwaps ............................................. 1098
<table>
<thead>
<tr>
<th><strong>CONTENTS</strong></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Swapping in LockedDoor GameObjects</td>
<td>1105</td>
</tr>
<tr>
<td>Implementing Keys and Unlocking Doors</td>
<td>1111</td>
</tr>
<tr>
<td>Adding GUI to Track Key Count and Health</td>
<td>1119</td>
</tr>
<tr>
<td>Enabling Enemies to Damage Dray</td>
<td>1125</td>
</tr>
<tr>
<td>Making Dray's Attack Damage Enemies</td>
<td>1130</td>
</tr>
<tr>
<td>Modifying Enemy to Take Damage</td>
<td>1131</td>
</tr>
<tr>
<td>Picking Up Items</td>
<td>1135</td>
</tr>
<tr>
<td>Enemies Dropping Items on Death</td>
<td>1138</td>
</tr>
<tr>
<td>Implementing a New Dungeon — The Hat</td>
<td>1143</td>
</tr>
<tr>
<td>Implementing a Grappler</td>
<td>1147</td>
</tr>
<tr>
<td>Summary</td>
<td>1184</td>
</tr>
</tbody>
</table>

**Part IV**  
Next Steps  

<table>
<thead>
<tr>
<th><strong>37</strong> Coding Challenges</th>
<th>1189</th>
</tr>
</thead>
<tbody>
<tr>
<td>What Is a Coding Challenge?</td>
<td>1190</td>
</tr>
<tr>
<td>Getting Started on a Coding Challenge</td>
<td>1191</td>
</tr>
<tr>
<td>Filling in the Blanks</td>
<td>1192</td>
</tr>
<tr>
<td>How to Approach Each Challenge</td>
<td>1194</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>38</strong> Beyond This Book</th>
<th>1197</th>
</tr>
</thead>
<tbody>
<tr>
<td>Continue to Learn Unity Development</td>
<td>1198</td>
</tr>
<tr>
<td>Build a Classic Game</td>
<td>1199</td>
</tr>
<tr>
<td>Start a Small Game Project or Prototype</td>
<td>1199</td>
</tr>
<tr>
<td>Make Games for Lifelong Enrichment</td>
<td>1200</td>
</tr>
<tr>
<td>Consider Going to School for GameDev</td>
<td>1200</td>
</tr>
<tr>
<td>Explore Advanced Game Design</td>
<td>1201</td>
</tr>
<tr>
<td>Finally, Drop Me a Line</td>
<td>1201</td>
</tr>
</tbody>
</table>

**Index**  

|       | 1203  |
PART V  Online Appendices — http://book.prototools.net or informit.com/title/9780136619949

A  Standard Project Setup Procedure
The Set Up Sidebar for Tutorial Projects
Setting Up a New Project
Importing a Starter UnityPackage
Setting the Scene Name
Setting the Game Pane to Full HD (1080p)
Setting Up a WebGL Build
Understanding Unity Version Control
Summary

B  Useful Concepts
Topics Covered
C# and Unity Coding Concepts
Attributes
Automatic Properties
Bitwise Boolean Operators and Layer Masks
Coroutines
Unity Example—Coroutines
Delegates, Events, and UnityEvents
UnityEvents
Enums
Extension Methods
Interfaces
Unity Example—Interfaces
Unity Makes Frequent Use of Interfaces for Observer Pattern
JSON (JavaScript Object Notation) in Unity
Lambda Expressions =>
Naming Conventions
Object-Oriented Software Design Patterns
Component Pattern
Observer Pattern
Singleton Pattern
Strategy Pattern
More Information on Design Patterns in Game Programming

Operator Precedence and Order of Operations

Recursive Functions

String Interpolation — "$"'

StringBuilder

Structs

Unity Messages Beyond Start() and Update()

Life-Cycle Messages
Frame-Based Messages
Physics-Based Messages

Variable Scope

XML

XML Documentation in C#

Math Concepts

Cosine and Sine (Cos and Sin)

Unity Example—Sine and Cosine

Dice Probability Enumeration

Unity Example—Dice Probability

Using Data-Oriented Design to Improve the DiceProbability Code

Dot Product

Interpolation

Linear Interpolation
Time-Based Linear Interpolations
Linear Interpolations Using Zeno’s Paradox
Interpolating More Than Just Position
Linear Extrapolation
Easing for Linear Interpolations

Bézier Curves

Three-Point and Four-Point Bézier Curves
A Recursive Bézier Curve Function
A Data-Oriented Bézier Function
Pen-and-Paper Roleplaying Games
   Tips for Running a Good Roleplaying Campaign
User Interface Concepts
   Complex Game Controller Input
   Input Manager Mapping for Various Controllers
Right-Click on macOS
   Control-Click = Right-Click
   Use Any PC Mouse
   Set Your macOS Mouse to Right-Click
   Set Your macOS Trackpad to Right-Click

C Online Reference
   Tutorials
   Unite Conference
   Unity's YouTube Channel
   Programming
   Searching Tips
   Finding and Creating Assets
   Other Tools and Educational Discounts

D Tips for Teaching from This Book
   The Goal of This Appendix
   Teaching Introduction to Game Design
   Teaching Introduction to Game Programming
   More Information Is Available
FOREWORD

Jeremy Gibson Bond taught me how to code. When I joined the University of Southern California Games program in 2012, one of the first things I did was to sign up for Jeremy's class in Unity and C# programming. I had just left Naughty Dog, where I'd worked as a lead game designer on the Uncharted series. I'd done a lot of scripting—simplified programming—during my career, but I'd always had a chip on my shoulder about not being a "real" coder. Jeremy's class fixed that, in just fifteen weeks.

In the class, I made a version of the classic game Asteroids in Unity, which my teammate and I then modded into an original game, and even though it was probably the simplest game I'd made since I was a kid, it was one of the most satisfying development experiences of my life. Every single one of Jeremy's classes was not only jam-packed with information about Unity and C# but was also peppered with inspirational wisdom about game design and practical pieces of advice related to game development—everything from his thoughts about good "lerping," to great tips for time management and task prioritization, to the ways that game designers can use spreadsheets to make their games better. I was blown away by Jeremy's skill as a teacher, and by his ability to make the process of creating gameplay into its own kind of exciting fun. Of course, I was delighted when I learned that he was packing all of that inspiring knowledge into the book you're now starting to read.

I'd first met Jeremy at the Game Developers Conference in 2002, and we hit it off immediately. Jeremy already had a successful career as a game developer, and his enthusiasm for game design struck a chord with me. I was drawn to his sharp understanding of game development and design, his easy, friendly manner, and the engaging way he loves to talk about game design as a craft, a design practice, and an emerging art form. We stayed friends down the years; I was excited when Jeremy got his master's degree from the world-famous Entertainment Technology Center at Carnegie Mellon University and was happy to see him go from strength to strength in his career. And of course, I was delighted to briefly be colleagues with him at USC, before he moved to teach at Michigan State University.

I graduated from Jeremy's class wishing that I could take it again, knowing that there was a huge amount more that I could learn from him. So now you're very fortunate, because you're holding in your hands what is essentially the textbook—and much, much more—of the class that I took with Jeremy. With an incredible wealth of knowledge
about game design, Unity, and C#, and highly detailed, step-by-step instructions, this book is a sure-fire method of realizing your game development dreams. Not only that, but it's the third edition, and over the years, Jeremy has been continuously refining and updating this superb volume, seeing it in action in his own classes and in those of others.

As you'll see, the book opens with a section on Jeremy's wide-reaching, wise, and grounded philosophy of game design, a section that is worth the price of admission on its own. Jeremy is extraordinarily well-read about game design, and this book is going to give you an overview of all the best game design theory to know about. After laying out the most useful definitions of "game," Jeremy will present you with his idea of the "Layered Tetrad," a valuable synthesis of the finest game analysis frameworks. He'll go on to give you a clear breakdown of how to design a game, including paper prototyping, playtesting, game balancing, guiding the player, and designing puzzles. He'll talk you through the best practices of Agile development, including the "burndown chart" scheduling tool, which Jeremy taught me. This tool is now a core part of my own classes and has helped countless game developers to both keep their projects on track and avoid running out of time. Part II of the book will ease you painlessly into the world of programming in C# and working in Unity. The careful way that Jeremy introduces and explains often abstract and difficult-to-grasp concepts is brilliant and works like magic to turn non-programmers into wised-up coders. Once you've worked through this section, you'll be ready to dive into the excellent tutorials in Part III.

This third edition of the book is the best yet, packed with up-to-date and essential information. It includes a new chapter on Data-Oriented Design—which thinks about code from the point of view of how data is managed by the computer—and Unity's new Data-Oriented Tech Stack, which can help you speed up the performance of your games enormously. The C# terms and samples in the book are now highlighted and color-coded in a very similar way to that of Microsoft Visual Studio, the C# development environment installed with Unity. The book's tutorials, an important part of the special magic of Jeremy's teaching, are more refined and detailed than ever before. These tutorials will supercharge your game coding practice, just like they did mine, as Jeremy guides you through the creation of small games that build your knowledge in a systematic way. In addition, Jeremy is now providing you with a set of "Coding Challenges" that can be found on the book's website: partially complete games that guide you in the creation of the code to make them work. These will help you transition from the tutorials in the book to writing your own games from scratch. As if all that wasn't enough, the book now has excellent new 3D art by Peter Burroughs and an appendix on how to teach using the book, which will be invaluable to game professors around the world. And don't miss the other appendices in the last part of the book, a grab-bag of knowledge and wisdom that are the diamonds and rubies at the very bottom of this mine.
Jeremy is an immensely talented and knowledgeable game developer and game educator. He's put in many multiples of the ten thousand hours said to be needed to become an expert, and done it several times over, in the disciplines of game design, C#, Unity, and game education. Not only that, but his integrity, his kindness, and his sense of fun shine through in these pages. This is the book that I recommend to my students when I want to help them transform themselves from a game engine dabbler into a Unity adept, and I am delighted to recommend it to you.

Good luck, and have fun!

Richard Lemarchand
Associate Professor, USC Games
Welcome to the third edition of *Introduction to Game Design, Prototyping, and Development*. This book is based on my work over many years as both a professional game designer and a professor of game design at several universities, including the Media and Information Department at Michigan State University and the Interactive Media and Games Division at the University of Southern California.

This preface introduces you to the purpose, scope, and approach of this book.

**The Purpose of This Book**

My goal in this book is simple: I want to give you all the tools and knowledge you need to get started down the path to being a successful game designer and prototyper. This book is the distillation of as much knowledge as I can cram into it to help you toward that goal. Unlike most books out there, this book combines both the disciplines of game design and digital game programming and development and wraps them both in the essential practice of iterative prototyping. The growth of advanced, yet approachable, game development engines such as Unity has made it easier than ever before to create playable prototypes that express your game design concepts to others. Whether you wish to be a game programmer, game designer, or a bit of both, this book has much to offer you.

**What's New in the Third Edition?**

Since 2017, when the second edition of this book was published, Unity has grown and changed considerably. To give you the best possible book for learning Unity, I had to change this book as well. Some of the major changes include:

- **About 400 additional pages:** With the inclusion of the online appendices, the additional content in this book is longer than the entirety of some books on learning to program. Unity has grown significantly over the years, and you need to know more to be able to use it well. I have added tons of content to the game prototypes so that you can experience more of the expanded features of Unity. Among the new chapters I have added is a new chapter explaining Data-Oriented Design, a new approach to Unity programming that can drastically increase performance and efficiency but
requires a completely different mindset from the Object-Oriented Programming that has been taught at universities for the past 30 years.

- **Improved, more polished tutorials:** The first edition contained eight rather small tutorials that provided a good introduction to Unity at the time. Now that Unity has more capabilities, I have worked these into the tutorials as well. Of the five tutorials in Part III of this book, three are now spread across two chapters (a space shooter, a card game, and an action/adventure game). For each of these, the first chapter sets up the underlying technology and gets you to a rough prototype, while the second chapter expands the prototype into a **first playable**, a more polished version of the game that is ready to be shown to others for feedback.

- **Better code throughout:** As I have improved as a programmer, so has the code in this book. Each of the Part III prototypes are designed to be a framework upon which you could build your own games, so code throughout the book has been revised to be more understandable and extensible. Additionally, immense care has been taken to implement consistent syntax coloring throughout the book, making the code clearer and easier to read. In the many places where you are modifying existing scripts, the areas that you must modify have also been made clearer.

- **Coding Challenges:** One wholly new aspect of the book is the online Coding Challenges, which are designed to aid your transition from following the book tutorials to creating your own games from scratch. Each challenge is a complete Unity game project with much of the key code missing. In place of this code are comments that explain what the code should do and how it should work. Replacing the missing code draws upon your experience from this book and helps you better internalize what you learn here. I have used these successfully in my classes for a few years to great effect!

- **Unity 2020.3 LTS:** Unity’s new commitment to Long Term Support (LTS) releases means that they will make only bug-fix and security fixes to LTS releases and will avoid any changes that could break code or tutorials like those in this book. By committing the book to version 2020.3 LTS, I avoid many of the issues that could come up if you tried to use a more recent version of Unity with these tutorials. 2020.3 LTS was released in mid-2021 and will be updated monthly until mid-2023, but it will be a viable, solid release for years after that.

- **Better online tools:** Many of the online tools that I offer you through this book are the same tools that I developed for my own game development projects and the classes I teach. Hundreds of students have used these tools for dozens of projects, and I have improved them every semester. This now even includes an online code-checker that can help you find issues in your code at any point in one of the tutorials.

These are just a few of the many improvements I have worked into the book since the previous edition. While the game design chapters in Part I and the C# programming
chapters in Part II have several revisions throughout, the game prototypes in Part III and beyond contain the most drastic changes. I have put well over 1,000 hours into improving this book to make it the best possible way for you to learn Unity. It contains as much content as I could possibly fit into it (more pages than they would allow me to print!), and I know it will be a great resource for you.

Who This Book Is For

There are many books about game design, and there are many books about programming. This book seeks to fill the gap between the two. As game development technologies like Unity become more ubiquitous, it is increasingly important that game designers have the ability to sketch their design ideas not only on paper but also through working digital prototypes. This book exists to help you learn to do just that:

- **If you’re interested in game design but have never programmed**, this book is perfect for you.
- **Part I: Game Design and Paper Prototyping** introduces you to several practical theories for game design and presents you with the practices that can help you develop and refine your design ideas.
- **Part II: Programming C# in Unity** teaches you how to program from nothing to understanding object-oriented class hierarchies in C# (pronounced See-Sharp). Since I became a college professor, the majority of my classes have focused on teaching nonprogrammers how to program games. I have distilled all of my experience doing so into Part II of this book.
- **Part III: Game Prototype Tutorials** takes you through the process of developing several different game prototypes across several different game genres. Each demonstrates fast methods to get from concept to working digital prototype.
- **Part IV: Next Steps** covers what you can do once you’ve finished this book. It introduces the Coding Challenges that have been extremely successful in helping my students transition from following tutorials to creating their own games and gives you many ideas for what you can do next in your journey.
- Lastly, the online **Appendices** in **Part V** explain specific game development and programming concepts in-depth and guide you to other online resources that may be useful.

- **If you’re a programmer who is interested in game design**, Parts I and III of this book will be of most interest to you.
- **Part I: Game Design and Paper Prototyping** introduces you to several practical theories for game design and presents you with the practices that can help you develop and refine your design ideas.
You can skim **Part II: Programming C# in Unity**, which introduces C# (pronounced *See-Sharp*) and how it is used in Unity. If you are familiar with other programming languages, C# looks like C++ but has the advanced features of Java.

**Part III: Game Prototype Tutorials** takes you through the process of developing several different game prototypes across several different game genres. Game development in Unity is very different from what you may be used to from other game engines, as many elements of development are managed outside of the code. Each prototype will demonstrate the style of development that works best in Unity to get from concept to working digital prototype quickly.

**Part IV: Next Steps** covers what you can do once you’ve finished this book. It introduces the Coding Challenges that have been extremely successful in helping my students transition from following tutorials to creating their own games and gives you many ideas for what you can do next in your journey.

You will also want to look carefully at **Part V: Appendices**, which is full of detailed information about various Unity development concepts and is arranged as a reference that you can return to later.

**If you’re teaching game design or programming**, you’re not alone. Many universities worldwide use this book as their game design and programming textbook. I have added a new **Appendix D** that outlines how I recommend teaching from this book.

---

**The Structure of This Book**

The book is divided into five parts:

**Part I: Game Design and Paper Prototyping**

The first part of the book starts by exploring various theories of game design and the analytical frameworks for game design that have been proposed by several earlier books. This section then describes the Layered Tetrad as a way of combining and expanding on many of the best features of these earlier theories. The Layered Tetrad is explored in depth as it relates to various decisions that you must make as a designer of interactive experiences. This part also covers information about the interesting challenges of different game design disciplines; describes the process of paper prototyping, testing, and iteration; gives you concrete information to help you become a better designer; and presents you with effective project and time management strategies to help keep your projects on track. The final chapter examines the game industry and gives you several tips for how to approach finding a job.
Part II: Programming C# in Unity

The second part teaches you C#—our programming language—from the basics through class inheritance and object-oriented programming. This part draws upon my many years of experience as a professor teaching nontechnical students how to express their game design ideas through digital code. If you have no prior knowledge or experience with programming or development, this part is designed for you. However, even if you do have some programming experience, you might want to take a look at this part to learn a few new tricks or get a refresher on some approaches.

The final chapters of this part explore Object-Oriented Programming and Data-Oriented Design, two very different approaches to designing advanced code. Data-Oriented Design is the core of Unity’s new Data-Oriented Tech Stack (DOTS), which can drastically improve the performance and efficiency of your code.

Part III: Game Prototype Tutorials

The third part of the book encompasses several different tutorials, each of which guides you through the development of a prototype for a specific style of game. The purpose of this part is twofold: It reveals some best practices for rapid game prototyping by showing you how I personally approach prototypes for various kinds of games, and it provides you with a basic foundation on which to build your own games in the future. Many other books on the market that attempt to teach Unity (our game development environment) do so by taking the reader through a single, monolithic tutorial that is hundreds of pages long. In contrast, this book takes you through several much smaller tutorials. The final products of these tutorials are necessarily less robust than those found in some other books, but it is my belief that the variety of projects in this book will better prepare you for creating your own projects in the future.

The three final projects of this part each span two chapters. The first chapter gets you to the playable prototype stage of the project, where the basic technology is in place and the core mechanics of the game work. The second chapter of each takes the game to what is known in the industry as a first playable, the state of the game where you would actually show it to other people and get their feedback. Each of these projects has grown as Unity has grown. The versions of these projects in the first edition of the book were basic and rough, while the versions in this third edition have grown more refined and take advantage of more interesting and useful aspects of both Unity and C# programming.

Part IV: Next Steps

This entirely new section comprises two chapters that will help you take the next steps in your game programming journey after you have finished the book. After completing
prior editions of the book, readers and students often had difficulty transitioning from following the detailed book tutorials to creating their own projects from scratch. To rectify this in my classes, I introduced Coding Challenges, game prototypes that are nearly complete except for the code. In place of the code, there are detailed comments outlining what the code needs to do there, and you can follow those comments to create the needed code and make the prototypes work. Chapter 37, "Coding Challenges," introduces you to these challenges and guides you to finding them online. I plan to add a new challenge at least once per semester following publication of the book. The final chapter of Part IV, "Beyond This Book," gives you several ideas for the next projects you can tackle and where to find resources to do so.

Part V: Online Appendices
This book has several important appendices that merit mention here. Rather than repeat information throughout the book or require you to go hunting through various chapters for it, any piece of information that is referenced several times in the book or that I think you would want to look back on (after you've finished reading the book once) is placed in the appendices, which are online-only, both to reduce the immense size of this book and make searching them easier. To find them, head to this book's website: http://book.prototools.net or informit.com/title/9780136619949.

- **Appendix A: Standard Project Setup Procedure** is a step-by-step introduction to the initial creation process for a game project in Unity. There is a lot to know, and this appendix will make sure your projects start on the right foot.

- **Appendix B: Useful Concepts.** Though it has a rather lackluster name, this is the portion of the book that I believe you will return to most often in the years following your initial read through the book. "Useful Concepts" is a collection of several go-to technologies and strategies that I use constantly in my personal game prototyping process, and I think you'll find a great deal of it to be very useful. To be honest, I brush up on topics in this appendix pretty often myself!

- **Appendix C: Online Reference** is a list of very useful online references where you can find answers to questions not covered in this book. It is often difficult to know the right places to look for help online; this appendix lists those that I personally turn to most often.

- **Appendix D: Tips for Teaching from This Book** covers my best practices for instructors using this book in a classroom. I have taught from this book every semester since the first edition was published, and I have iterated many times to find the right way to present the information. This includes sample schedules for both Game Design and Game Programming classes.
Book Website
The website for this book includes all of the files referenced in the chapters, lecturer notes, starter packages, and errata for anything that we somehow failed to correct in the many passes through editing and the tutorial projects. Find it at

http://book.prototools.net

or

informit.com/title/9780136619949

Why You Should Learn Unity and C#
All the digital game examples in this book are based on the C# programming language and the Unity Game Engine. I have taught students to develop digital games and interactive experiences for two decades now, and in my experience, Unity is—by far—the best environment that I have found for learning to develop games. I have also found that C# is the best initial language for game prototypers to learn.

The Unity 2020.3 LTS Development Environment
Some other tools out there are easier to learn and require no real programming (Game Maker is a great example), but Unity allows you much more flexibility and performance in a package that is basically free (the free version of Unity includes nearly all the capabilities of the paid version, and it is the version used throughout this book). Unreal is another game engine used by some studios, but in Unreal, there is very little middle ground between the simplified graphical programming of the Blueprint system and the very complex C++ code on which the engine is built. If you want to actually learn to program games and have success doing it, Unity is the engine you want to use.

Unity has both Tech Stream releases that include all the newest features (in a sometimes buggy state) and Long Term Support (LTS) releases that are stable and supported for many years. Unity 2020.3 LTS, which we use in this book, was initially released in 2021 and represents an extremely stable and feature-rich release of Unity. It will be updated monthly until 2023 and will be stable and usable for several years beyond that. I do not recommend attempting to follow the book tutorials with a future version of Unity, but transitioning from 2020.3 LTS to later versions of Unity after completing the book will be easy for you.

The C# Programming Language
In the past, I have taught my students many languages, including C++, JavaScript, and ActionScript. However, C# is the one language that I have used that continually
impresses me with its flexibility and feature set. Learning C# means learning not only programming but also good programming practices. Languages such as JavaScript allow a lot of sloppy behaviors that I have found actually lead to slower development. C# keeps you honest (via things like strongly typed variables), and that honesty will not only make you a better programmer but will also result in your being able to code more quickly (e.g., strong variable typing enables very robust code hinting and auto-completion, which makes coding faster and more accurate).

Conventions in This Book
This book maintains several writing conventions to help make the text more easily understandable.

Any place that specific button names, menu commands, or other multi-word nouns appear in the text, they will be listed in italics. This includes terms like the Main Camera GameObject. An example menu command is Edit > Project Settings > Physics, which would instruct you to select the Edit menu from the menu bar, choose the Project Settings submenu, and then select Physics. I also tend to italicize important terms when first introducing them and use bold and italics for emphasis throughout the book. When specific terms from C# code are used in text, they are in bold code font for emphasis and clarity. Examples include float, List<>, and text like "Hello World", and MonoBehaviour (which uses the British spelling because Unity originated in Europe).

Book Elements
The book includes several different types of asides that feature useful or important information that does not fit in the flow of the regular body text.

**note**
Callouts in this format are for information that is useful but not critical. Information in notes will often be an interesting aside to the main text that provides a little bit more info about the topic.

**tip**
This element provides additional information that is related to the book content and can help you as you explore the concepts in the book.
**warning**

BE CAREFUL  Warning cover information about things that you need to be aware of to avoid mistakes or other pitfalls.

**SIDEBAR**

The sidebar is for discussions of longer topics that are important to the text but should be considered separately from it.

---

**Code**

Several conventions apply to the code samples in this book. When specific elements from the code listing are placed in regular paragraph text, they appear in a monospaced font. The variable `variableOnNewLine` from the code listing below is an example of this.

Code Listings also utilize a monospaced font and appear as follows. Code Listings are all numbered (here 0.1), and the name of the code file you’re editing is also included (e.g., SampleClass.cs).

**Code Listing 0.1  SampleClass.cs**

```csharp
1 public class SampleClass {
2     public GameObject variableOnExistingLine; // a
3     public GameObject variableOnNewLine; // b
4 }
```

**a.** Code Listings are often annotated; in this case, additional information about the line marked with `// a` would appear in this first annotation. Annotations are always bold to call attention to them.

**b.** Some code listings will be expansions on code that you’ve already written or that already exists in the C# script file for another reason. In this case, the old lines will be at normal weight, and the new lines will be at **bold weight**.

Most of the code listings in the first two parts of the book will include line numbers (as seen in the preceding listing). **You do not need to type the line numbers** when entering the code into Visual Studio (it will automatically number all lines). In Part III of the book, there are no line numbers due to the size and complexity of the code listings increasing the chance that your line numbers would differ from mine. However, later
code listings precede each line with a pipe character "|" to clarify the indentation level of each line of code, and new lines are preceded by a bold right angle bracket ">" as shown in Code Listing 0.2. You also should not type these | or > characters.

Code Listing 0.2  SampleClassFromLaterInTheBook.cs

```csharp
| public class SampleClassFromLaterInTheBook {
|     public GameObject variableOnExistingLine;
>     public GameObject variableOnNewLine;
| }
```

**tip**

THE CODE YOU WRITE IN YOUR PROJECTS WON'T LOOK LIKE MINE
This is something that a reader asked me to add to the beginning of this book. I spend many hours and many passes working to make my code as clear and understandable as possible. When you start writing your own C# code for your own games, it is not going to be as clean, and that is absolutely okay. Game prototyping is not about beautiful, clean code; it is about getting a game working as quickly as possible. Once the game is working, if you want to continue and expand the project, you can always go back and refactor the code into something cleaner. And, if you want to write tutorials to teach other people, you can refactor it a third or even fourth time, like I have.

There Are Other Books Out There

As a designer or creator of any kind, I think that it’s absolutely essential to acknowledge those on whose shoulders you stand. Many books have been written on games and game design, and the few that I list here are those that have had the most profound effect on either my process or my thinking about game design. You will see several of these books referenced many times throughout this text, and I encourage you to read as many of them as possible.

**Game Design Workshop by Tracy Fullerton**

Initially penned by Tracy Fullerton, Chris Swain, and Steven S. Hoffman, *Game Design Workshop* is now in its third edition. This book was initially based on the Game Design Workshop class that Tracy and Chris taught at the University of Southern California, a class that formed the foundation for the entire games program at USC (and a class
that I taught at USC from 2009–2013). The USC Interactive Media and Games graduate program has been named the number one private university for game design in North America by Princeton Review nearly every year that it has been ranking game programs, and the Game Design Workshop book and class were the foundation for that success.

Unlike many other books that speak volumes of theory about games, Tracy’s book maintains a laser focus on information that helps budding designers improve their craft. I taught from this book for many years (even before I started working at USC), and I believe that if you actually attempt all the exercises listed in the book, you can’t help but have a pretty good paper game at the end.

Tracy Fullerton, Christopher Swain, and Steven Hoffman, *Game Design Workshop: A Playcentric Approach to Creating Innovative Games*, 2nd ed. (Boca Raton, FL: Elsevier Morgan Kaufmann, 2008)

**The Art of Game Design by Jesse Schell**

Jesse Schell was one of my professors at Carnegie Mellon University and is a fantastic game designer with a background in theme park design gained from years working for Walt Disney Imagineering. Jesse's book is a favorite of many working designers because it approaches game design as a discipline to be examined through 100 different lenses that are revealed throughout the book. Jesse's book is a very entertaining read and broaches several topics not covered in this book.


**Advanced Game Design: A Systems Approach by Michael Sellers**

Mike Sellers once told me that "Systems thinking is the literacy of the 21st century," and I think that he is absolutely right. A large portion of the world population began the 20th century unable to read, but it is now a nearly ubiquitous skill. Similarly, he believes that to survive in the future, people must learn to understand the interrelated systems that impact their lives, and one of the best ways to understand systems is to design games. Mike is one of the most brilliant game and artificial intelligence developers that I have ever met, and he distilled a tremendous amount of his knowledge, understanding, and processes of design into this excellent book.

Richard Lemarchand, the author of the forewords to all three editions of this book, has thought deeply for nearly 30 years about how the production process of a game (or any project) can be designed to make the experience of working on the game playful and joyful for everyone involved. Those years of experience as both a co-lead designer on the Uncharted series and a professor at the University of Southern California have all led to this book. Discussions with Richard and my reading of this book changed how I approach some of my group-based game development classes and led to many of the third edition changes that I made to Chapter 7, "Acting Like a Designer."


**Games, Design and Play by Colleen Macklin and John Sharp**

Unlike some other game design texts that talk about the process of game design in theory, *Games, Design and Play* digs deeply into the details of design; into the nuts and bolts of what actually goes into making good design decisions as well as the impact of those decisions. Colleen and John do not cover game development at all—in fact, they recommend that you read this book to learn game programming—and instead focus exclusively on game design. This book illustrates its points with real examples from independent game developers, meaning that their examples are much more similar in scope to those you might encounter as you’re getting into game development.

Colleen Macklin and John Sharp, *Games, Design and Play: A Detailed Approach to Iterative Game Design* (Boston, MA: Addison-Wesley, 2016)

**Level Up! by Scott Rogers**

Rogers distills his knowledge from many years in the trenches of game development into a book that is fun, approachable, and very practical. When he and I co-taught a level design class, this was the textbook that we used. Scott is also a comic book artist, and his book is full of humorous and helpful illustrations that drive home the design concepts.

Imaginary Games by Chris Bateman
Bateman uses this book to argue that games are a legitimate medium for scholarly study. He pulls from several scholarly, practical, and philosophical sources; and his discussions of books like *Homo Ludens* by Johan Huizinga, *Man, Play, and Games* by Roger Caillois, and the paper "The Game Game" by Mary Midgley are both smart and accessible.


The Grasshopper by Bernard Suits
While not actually a book on game design at all, *The Grasshopper* is an excellent exploration of the definition of the word *game*. Presented in a style reminiscent of the Socratic method, the book presents its definition of game very early in the text as the Grasshopper (from Aesop’s fable *The Ant and the Grasshopper*) gives his definition on his deathbed, and his disciples spend the remainder of the book attempting to critique and understand this definition. This book also explores the question of the place of games and play in society.


Game Design Theory by Keith Burgun
In this book, Burgun explores what he believes are faults in the current state of game design and development and proposes a much narrower definition of *game* than does Bernard Suits. Burgun’s goal in writing this text was to be provocative and to push the discussion of game design theory forward. While largely negative in tone, Burgun’s text raises a number of interesting points and helped me refine my personal understanding of game design.

ACKNOWLEDGMENTS

A tremendous number of people deserve to be thanked here. First and foremost, I want to thank my wife, Melanie, whose help and feedback on my chapters throughout the entire process of all three editions of this book improved them tremendously. She is not only my inspiration but has also been an excellent copy editor. I also want to thank my family for their many years of support, with special thanks to my father for teaching me how to program as a child.

As with every edition, there were several people at Pearson who provided support to me and shepherded me through this process. Chief among them was Chris Zahn, who has been with me since the first edition. Laura Lewin initially approached me about writing a book and served as the acquisitions editor for the first two editions. Also at Pearson, Malobika Chakraborty, Lori Lyons, and Tonya Simpson each demonstrated incredible patience in working with me as I worked to complete this book with a new child and throughout the COVID pandemic. Margaret Moser continued her fantastic work as a technical reviewer on this edition of the book and not only caught my mistakes but also added her brilliant insight throughout the book. Thanks also to the excellent proofreader, Donna E. Mulder, as well as Aswini Kumar and the team at Codemantra for their work in the production phase.

I would also like to thank all the educators who have taught me and worked as my colleagues. Special thanks go to Dr. Randy Pausch and Jesse Schell. Though I had worked as a professor and game designer before meeting them, they each had a profound effect on my understanding of design and education. I also owe tremendous thanks to Tracy Fullerton, Mark Bolas, and Scott Fisher, who were friends and mentors to me in the years I taught at the University of Southern California’s Games and Interactive Media Division. There were also many other brilliant faculty and friends at USC who helped me flesh out the ideas in this book, including Adam Sulzdorf-Liszkwicz, William Huber, Richard Lemarchand, Scott Rogers, Vincent Diamante, Sam Roberts, and Logan Ver Hoef. My current colleagues at Michigan State University have also contributed ideas and feedback on the third edition of the book, especially Andrew Dennis, Elizabeth LaPensée, Adam Sulzdorf-Liszkwicz, and Ryan Thompson.

Many of my friends in the industry have also helped me by giving me suggestions for the book and feedback on the ideas presented therein. These included Michael Sellers, Nicholas Fortugno, Jenova Chen, Zac Pavlov, Joseph Stevens, and many others.
Thanks as well to all the fantastic students that I have taught over the past decade. It is you who inspired me to want to write this book and who convinced me that there was something important and different about the way I was teaching game development. Every day that I teach, I find myself inspired and invigorated by your creativity, intelligence, and passion.

Finally, I would like to thank you. Thank you for purchasing this book and for your interest in developing games. I hope that this book helps you get started, and I would love to see what you make with the knowledge you gain here.
ABOUT THE AUTHOR

Jeremy Gibson Bond is a Professor of Practice teaching game design and development at Michigan State University, which in 2022 was ranked the #1 public university for undergraduate game development by Princeton Review three of the last four years. Since 2013, he has served the IndieCade independent game festival and conference as the Chair of Education and Advancement, where he co-chairs the IndieXchange summit each year and has also chaired the GameU summit. In 2013, Jeremy founded the company ExNinja Interactive, through which he develops his independent game projects. Jeremy has spoken several times at the Game Developers Conference. He also created the official Unity Certified Programmer Exam Review specialization on Coursera, which thousands of developers (including several Unity employees) used to prepare for the UCP exam from 2018–2022.

Prior to joining the Games faculty at Michigan State, Jeremy taught for three years as a lecturer in the Electrical Engineering and Computer Science department at the University of Michigan Ann Arbor where he taught game design and software development. From 2009–2013, Jeremy was an assistant professor teaching game design for the Games and Interactive Media Division of the University of Southern California’s School of Cinematic Arts, which was named the #1 game design school in North America throughout his tenure there.

Jeremy earned a Master of Entertainment Technology degree from Carnegie Mellon University’s Entertainment Technology Center in 2007 and a Bachelor of Science degree in Radio, Television, and Film from the University of Texas at Austin in 1999. Jeremy has worked as a programmer and prototyper for companies such as Human Code and frog design; has taught classes for Great Northern Way Campus (in Vancouver, BC), Texas State University, the Art Institute of Pittsburgh, Austin Community College, and the University of Texas at Austin; and has worked for Walt Disney Imagineering, Maxis, and Electronic Arts/Pogo.com, among others. While in graduate school, his team created the game *Skyrates*, which won the Silver Gleemax Award at the 2008 Independent Games Festival. Jeremy also apparently has the distinction of being the first person to ever teach game design in Costa Rica.
Figure Credits

Cover image by Rost9/Shutterstock

Figure 1-2: Jason Rohrer
Figure 2-1: Robin Hunicke
Figure 2-3, Figure 3-1, Figure 3-2: Taylor & Francis Group
Figure 4-1: Thatgamecompany, Inc
Figure 5-2: Richard A. Bartle
Chapter 5, Zork screengrab: Infocom
Figure 7-1: Elsevier
Figure 8-5: Mattie Brice
Figure 9-3, Figure 9-5: Nintendo
Figure 10-2: You Run Ltd
Figure 19-3, Figure 25-7, Figure 25-10, Figure 25-11, Figure 25-12 (bottom), Figure 26-1,
Figure 26-3, Figure 35-5, Figure 8-7: Microsoft Corporation
Figure 12-1, Figure 12-5: Sony Interactive Entertainment
Figure 12-2, Figure 12-3: Naughty Dog
Figure 12-4, Figure 12-6: Eden Games
Figure 13-1, Figure 13-2: Scott Kim

Figure 16-1, Figure 16-2, Figure 17-1–Figure 17-10, Figure 19-1, Figure 19-4–Figure 19-14,
Figure 20-1, Figure 23-1, Figure 23-2, Figure 25-1–Figure 25-6, Figure 25-8, Figure 25-9, Figure 26-2,
Figure 27-1–Figure 27-8, Figure 28-3, Figure 28-5–Figure 28-7, Figure 28-9–Figure 28-11,
Figure 28-12B, Figure 28-13, Figure 28-14, Figure 29-1–Figure 29-9, Figure 29-11–Figure 29-15,
Figure 30-1–Figure 30-7, Figure 30-10–Figure 30-17, Figure 31-1–Figure 31-10, Figure 32-1,
Figure 32-2, Figure 32-3c, Figure 32-4–Figure 32-12, Figure 33-1, Figure 33-4–Figure 33-11,
Figure 33-13–Figure 33-15, Figure 34-1–Figure 34-9, Figure 35-3, Figure 35-4, Figure 35-8,
Figure 35-10–Figure 35-12, Figure 36-2–Figure 36-7, Figure 37-1, Figure 37-2, Figure A-1–Figure A-7,
Figure B-1, Figure B-2, Figure B-4, Figure B-5, Figure B-9, Figure C-1: Unity Technologies

Figure 19-2, Figure 25-12 (top): Apple Inc
Figure 11-1–Figure 11-8, Figure 11-13, Figure 11-15–Figure 11-18, Figure 14-1–Figure 14-7,
Figure 28-12A, Figure 8-6: Google LLC

Figure 30-9: Skyrates
Figure 33-1A, Figure 33-2, Figure 33-3, Figure 33-7A, Figure 33-9A, Figure 33-11A, Figure 33-13A,
Figure 33-15A: Chris Aguilar

Figure 35-1, Figure 35-2, Figure 35-6, Figure 35-9, Figure 35-13, Figure 35-14,
Figure 36-1: SKIPSTONE PICTURES
This page intentionally left blank
Our journey starts here. This chapter presents the basic theories of design upon which the rest of the book is built. In this chapter, you also encounter your first game design exercise and learn more about the underlying philosophy of this book.
You Are a Game Designer

As of this moment, you are a game designer, and I want you to say it out loud:¹

"I am a game designer."

It's okay. You can say it out loud, even if other people can hear you. In fact, according to psychologist Robert Cialdini’s book, *Influence: The Psychology of Persuasion*,² if other people hear you commit to something, you're more likely to follow through. So, go ahead and tell your friends, tell your family, shout it from the mountain tops, post it to social media:

"I am a game designer!"

But, what does it mean to be a game designer? This book will help you answer that question and will give you the tools to start making your own games. Let's start with a design exercise.

**Bartok: A Game Design Exercise**

I first saw this exercise used by game designer Malcolm Ryan as part of a Game Design Workshop session at the Foundations of Digital Gaming conference. The goal of this exercise is to demonstrate how even a simple change to the rules of a game can have a massive effect on the experience of playing the game.

*Bartok* is a simple game played with a single deck of standard cards that is very similar to the commercial game *Uno*. In the best-case scenario, you would play this game with three friends who are also interested in game design; however, I've also made a digital version of the game that you can play solo. Either the paper or digital version will work fine for our purposes.³

---

1. I thank my former professor Jesse Schell for asking me to make this statement publicly in a class full of people. He also includes this request in his excellent book, *The Art of Game Design: A Book of Lenses* (Boca Raton, FL: CRC Press, 2008).
PLAYING THE DIGITAL VERSION OF BARTOK

To play the digital version of Bartok, simply visit the website for this book:

http://book.prototools.net

You will find the game in the section of the website for Chapter 1.

You can, of course, also just grab a standard deck of playing cards and a few friends and play the game in person, which will allow you to talk with your friends about the feel of the game and the changes you want to make to it.

Objective

Be the first player to get rid of all the cards in your hand.

Getting Started

Here are the basic rules for Bartok:

1. Start with a regular deck of playing cards. Remove the Jokers, leaving you with 52 cards (13 of each suit ranked Ace–King).
2. Shuffle the deck and deal seven cards to each player.
3. Place the rest of the cards face-down in a draw pile.
4. Pick the top card from the draw pile and place it on the table face-up to start the discard pile.
5. Starting with the player to the left of the dealer and proceeding clockwise, each player must play a card onto the discard pile if possible, and if they cannot play a card, the player must draw a single card from the draw pile (see Figure 1.1).
6. A player may play a card onto the discard pile if the card is either:
   a. The same suit as the top card of the discard pile. (For example, if the top card of the discard pile is a 2 of Clubs (2C), any other Club may be played onto the discard pile.)
   b. The same rank as the top card of the discard pile. (For example, if the top card of the discard pile is a 2C, any other 2 may be played onto the discard pile.)
7. The first player to successfully get rid of all their cards wins.
Figure 1.1 The initial layout of Bartok. In the situation shown, the player can choose to play any one of the cards highlighted with blue borders (7C, JC, 2H, 2S).

Playtesting

Try playing the game a couple times to get a feel for it. Be sure to shuffle the cards thoroughly between each playthrough. Games will often result in a somewhat sorted discard pile, and without a good shuffle, subsequent games may have results weighted by the nonrandom post-game card distribution.
Deblocking is the term for strategies used to break up blocks of cards (i.e., groups of similar cards). In Bartok, each successful game ends with all the cards sorted into blocks of the same suit and blocks of the same rank. If you don’t deblock those groups, the subsequent game will end much faster because players are more likely to be dealt cards that match each other.

According to mathematician and magician Persi Diaconis, seven good riffle\textsuperscript{4} shuffles should be sufficient for nearly all games;\textsuperscript{5} if you run into issues, though, some of these deblocking strategies can help.

Here are some standard strategies for deblocking a deck of cards if standard shuffling doesn’t work:

- Deal the cards into several different piles. Then shuffle these piles together.
- Deal the cards out face-down into a large, spread-out pool. Then use both hands to move the cards around almost like mixing water. This is how dominoes are usually shuffled, and it can help break up your card blocks. Then gather all the cards into a single stack.
- Play 52 Pickup: Throw all the cards on the floor and pick them up.

Analysis: Asking the Right Questions

After each playtest, it’s important to ask the right questions. Of course, each game will require slightly different questions, though many of them will be based on these general guidelines:

- **Is the game of the appropriate difficulty for the intended audience?** Is it too difficult, too easy, or just right?

---

\textsuperscript{4}A riffle shuffle is one where half of the deck starts in each hand and you bend the cards up with the thumb and hold the cards down with the index finger of each hand, causing the cards from the left and right to alternate falling into a center pile. See more at https://en.wikipedia.org/wiki/Shuffling#Riffle.

- **Is the outcome of the game based more on strategy or chance?** Does randomness play too strong a role in the game, or, alternatively, is the game too deterministic so that after one player takes the lead, the other players don’t have any chance to catch up?

- **Does the game have meaningful, interesting decisions?** When it’s your turn, do you have several choices, and is the decision between those choices an interesting one?

- **Is the game interesting when it’s not your turn?** Do you have any effect on the other players’ turns, or do their turns have any immediate effect on you?

We could ask many other questions, but these are some of the most common.

Take a moment to think about your answers to these questions relative to the games of *Bartok* you just played and write them down. If you’re playing the paper version of this game with other human players, asking them to write down their own answers to the questions individually and then discussing them after they’re written is worthwhile, because it keeps each player’s responses from being influenced by the other players.

## Modifying the Rules

As you’ll see throughout this book, from a process standpoint, game design is pretty straightforward. The process is almost always:

1. Incrementally modify the rules, changing very few things between each playtest.
2. Playtest the game with the new rules.
3. Analyze how the feel of the game is altered by the new rules.
4. Design new rules that you think might move the feel of the game in the direction you want.
5. Repeat this process until you’re happy with the game.

*Iterative design* is the term for this repetitive process of deciding on a small change to the game design, implementing that change, playtesting the game, analyzing how the change affected the gameplay, and then starting the process over again by deciding on another small change. Chapter 7, "Acting Like a Designer," covers iterative design in detail.

For the *Bartok* example, why don’t you start by picking one of the following three rule changes and playtesting it:

- **Rule 1:** If a player plays a 2, the person to her left must draw two cards instead of playing.

- **Rule 2:** If any player has a card that matches the rank and color (red or black) of the top card, they may announce "Match card!" and play it out of turn. Play then
continues with the player to the left of the one who just played the out-of-turn card. This can lead to players having their turns skipped.

For example: In a four-player game, the first player plays a 3C (three of Clubs). The third player has the 3S (which matches both the rank and color of the 3C), so they call "Match card!" and play the 3S on top of the 3C out-of-turn, skipping the second player’s turn. Play then continues with the fourth player.

- **Rule 3:** A player must announce "Last card" when they have only one card left. If someone else calls it first, the player must draw two cards (bringing their total number of cards to three).

Choose only one of the rule changes from the previous listing and play the game a couple times with the new rule. Then have each player write their answers to the four playtest questions. You should also try playing with another one of the rules (although I would recommend still only using one of them at a time when trying a new rule for the first time).

If you’re playing the digital version of the game, you can use the check boxes on the menu screen to choose various game options.

---

**warning**

**WATCH OUT FOR PLAYTESTING FLUKES** A weird shuffle or other external factor can sometimes cause a single play through the game to feel really different from the others. This is known as a *fluke*, and you want to be careful not to make game design decisions based on flukes. If something you do seems to affect the game feel in a very unexpected way, be sure to play through the game multiple times with those same rules to make sure you’re not experiencing a fluke.

---

**Analysis: Comparing the Rounds**

Now that you’ve played through the game with some different rule options, analyze the results from the different rounds. Look back over your notes and see how each different rule set felt to play. As you experienced, even a simple rule change can greatly change the feel of the game. Here are some common reactions to the previously listed rules:

- **The original rules**
  Many players find the original version of the game to be pretty boring. There are no interesting choices to make, and as the players remove cards from their hands, the number of possible choices dwindles as well, often leaving the player with only one valid choice for most of the later turns of the game. The game is largely based
on chance, and players have no real reason to pay attention to other players' turns because they don't really have any way of affecting each other.

- **Rule 1:** *If a player plays a 2, the person to her left must draw two cards instead of playing.*

  This rule allows players to directly affect each other, which generally increases interest in the game. However, whether a player has 2s is based entirely on luck, and each player only really has the ability to affect the player on their left, which often seems unfair. However, this does make other players' turns a bit more interesting because other players (or at least the player to your right) have the ability to affect you.

- **Rule 2:** *If any player has a card that matches the number and color (red or black) of the top card, they may announce "Match card!" and play it out of turn. Play then continues with the player to the left of the one who just played the out-of-turn card.*

  This rule often has the greatest effect on player attention. Because any player has the opportunity to interrupt another player's turn, all players tend to pay a lot more attention to each other's turns. Games played with this rule often feel more dramatic and exciting than those played with the other rules.

- **Rule 3:** *A player must announce "Last card!" when they have only one card left. If someone else calls it first, the player must draw two cards.*

  This rule only comes into play near the end of the game, so it doesn't have any effect on the majority of gameplay; however, it does change how players behave at the end. This can lead to some interesting tension as players try to jump in and say, "last card" before the player who is down to only one card. This is a common rule in both domino and card games where the players are trying to empty everything from their hands because it gives other players a chance to catch up to the lead player if the leader forgets about the rule.

### Designing for the Game Feel That You Want

Now that you've seen the effects of a few different rules on *Bartok*, it's time to do your job as a designer and make the game better. First, decide on the feel that you want the game to have: Do you want it to be exciting and cutthroat, do you want it to be leisurely and slow, or do you want it to be based more on strategy than chance?

After you have a general idea of how you want the game to feel, think about the rules that you tested and try to come up with additional rules that can push the feel of the game in the direction that you want. Here are some tips to keep in mind as you design new rules for the game:

- Change only one thing in between each playtest. If you change (or even tweak) a number of rules between each play through the game, it can be difficult to determine which rule is affecting the game in what way. Keep your changes incremental, and you'll be better able to understand the effect that each is having.
The bigger change you make, the more playtests will be required to understand how it changes the game feel. If you only make a subtle change to the game, one or two plays can tell you a lot about how that change affects the feel. However, if it's a major rule change, you will need to test it more times to avoid being tricked by a fluke game. Additionally, if the small rule change only happens in rare circumstances, you also may need multiple plays through the game to experience that circumstance.

- Change a number, and you change the experience. Even a seemingly small change can have a huge effect on gameplay. For instance, think about how much faster this game would end if there were two discard piles to choose from or if the players started with five cards instead of seven.

Of course, adding new rules is a lot easier to do when playing the card game in person with friends than when working with a digital prototype. That's one of the reasons that paper prototypes can be so important, even when you're designing digital games. The first part of this book discusses both paper and digital design, but most of the design exercises are done with paper games because they can be so much faster to develop and test than digital games.

### The Definition of Game

Before moving too much further into design and iteration, we should probably clarify what we're talking about when we use terms such as *game* and *game design*. Many very smart people have tried to accurately define the word *game*. Here are a few of them in chronological order:

- In his 1978 book *The Grasshopper*, Bernard Suits (who was a professor of philosophy at the University of Waterloo) declares that "a game is the voluntary attempt to overcome unnecessary obstacles."  

- Game design legend Sid Meier says that "a game is a series of interesting choices."  

- In *Game Design Workshop*, Tracy Fullerton defines a game as "a closed, formal system that engages players in a structured conflict and resolves its uncertainty in an unequal outcome."  

- In *The Art of Game Design*, Jesse Schell playfully examines several definitions for *game* and eventually decides on "a game is a problem-solving activity, approached with a playful attitude."  

---

In the book *Game Design Theory*, Keith Burgun presents a much more limited definition of game: "a system of rules in which agents compete by making ambiguous, endogenously meaningful decisions."  

As you can see, all of these are compelling and correct in their own way. Perhaps even more important than the individual definition is the insight that it gives us into each author’s intent when crafting that definition.

**Bernard Suits’ Definition**

In addition to the short definition "a game is the voluntary attempt to overcome unnecessary obstacles," Suits also offers a longer, more robust version:

> To play a game is to attempt to achieve a specific state of affairs, using only means permitted by rules, where the rules prohibit use of more efficient in favor of less efficient means, and where the rules are accepted just because they make possible such activity.

Throughout his book, Suits proposes and refutes various attacks on this definition; and having read the book, I am certainly willing to say that he has found the definition of "game" that most accurately matches the way that the word is used in day-to-day life.

However, it's also important to realize that this definition was crafted in 1978, and even though digital games and roleplaying games existed at this time, Suits was either unaware of them or intentionally ignored them. In fact, in Chapter 9 of *The Grasshopper*, Suits laments that there is no kind of game with rules for dramatic play through which players could burn off dramatic energy (much like children can burn off excess athletic energy via play of any number of different sports), exactly the kind of play that was enabled by games like *Dungeons & Dragons*.

Although this is a small point, it gets at exactly what is missing from this definition: Whereas Suits’ definition of game is an accurate definition of the word, it offers nothing to designers seeking to craft good games for others.

---


11. *Endogenous* means inherent to or arising from the internal systems of a thing, so "endogenously meaningful decisions" are those decisions that actually affect the game state and change the outcome. Choosing the color of your avatar’s clothing in *Farmville* is not endogenously meaningful, whereas choosing the color of your clothing in *Metal Gear Solid 4* is, because the color of your clothing affects whether your avatar is visible to enemies.


For an example of what I mean, take a moment to play Jason Rohrer’s fantastic game Passage\textsuperscript{14} (see Figure 1.2). The game only takes 5 minutes to play, and it does a fantastic job of demonstrating the power that even short games can have. Try playing through it a couple times. If you can’t find a playable version for your computer, try watching some videos online, though playing it yourself is certainly better.

\textbf{Figure 1.2} Passage by Jason Rohrer (released December 13, 2007)

Suits’ definition will tell you that, yes, this is a game. In fact, it is specifically an “open game,” which he defines as: a game that has as its sole goal the continuance of the game.\textsuperscript{15} In Passage, the goal is to continue to play for as long as possible... or is it? Passage has several potential goals, and it’s up to the player to choose which of these they want to achieve. These goals could include the following:

- Moving as far to the right as possible before dying (exploration)
- Earning as many points as possible by finding treasure chests (achievement)
- Finding a wife (socialization)

The point of Passage as an artistic statement is that each of these can be a goal in life, and to some extent, these goals are all mutually exclusive. If you find a wife early in the game, getting treasure chests becomes more difficult because the two of you are unable to enter areas that could be entered singly. If you choose to seek treasure, you will spend your time exploring the vertical space of the world and won’t be able to see the different scenery to the right. If you choose to move as far to the right as possible, you won’t rack up nearly as much treasure.

In this incredibly simple game, Rohrer exposes a few of the fundamental decisions that every one of us must make in life and demonstrates how even early decisions can have a major effect on the rest of our lives. The important thing here is that he is giving players choice and demonstrating to them that their choices matter.

\begin{itemize}
  \item Passage is downloadable from Rohrer’s website at \url{http://hcsoftware.sourceforge.net/passage/}, or you can find an online version at \url{http://passage.toolness.org/}.
  \item Suits contrasts these with closed games, which have a specific goal (e.g., crossing a finish line in a race or ridding yourself of all your cards in Bartok). Suits’ example of an open game is the game of make-believe that children play.
\end{itemize}
This is an example of the first of a number of designer's goals that I will introduce in this book: *experiential understanding*. Whereas a linear story like a book can encourage empathy with a character by exposing the reader to the character's life and the decisions that they have made, games can allow players to understand not only the outcome of decisions but also to be complicit in that outcome by giving the player the power and the responsibility of decision and then showing them the outcome wrought by their decisions. Chapter 8, "Design Goals," explores these in much greater depth.

**Sid Meier's Definition**

By stating that "a game is a series of interesting choices," Meier is saying very little about the definition of the word *game* (there are many, many things that could be categorized as a series of interesting choices and yet are not games) and quite a bit about what he personally believes makes for a good game. As the designer of games such as *Pirates*, *Civilization*, *Alpha Centauri*, and many more, Sid Meier is one of the most successful game designers alive, and he has consistently produced games that present players with interesting choices. This, of course, raises the question of what makes a choice or decision *interesting*. An interesting decision is generally one where:

- The player has multiple valid options from which to choose.
- Each option has both positive and negative potential consequences.
- The outcome of each option is predictable but not guaranteed.

This brings up the second of our designer's goals: to create *interesting decisions*. If a player is presented with a number of choices, but one choice is obviously superior to the others, the experience of deciding which to choose doesn't actually exist. If a game is designed well, players will often have multiple choices from which to choose, and the decision will often be a tricky one.

**Tracy Fullerton's Definition**

As she states in her book, Tracy Fullerton is much more concerned with giving designers tools to make better games than she is with the philosophical definition of *game*. Accordingly, her definition of a game as "a closed, formal system that engages players in a structured conflict and resolves its uncertainty in an unequal outcome" is not only a good definition of *game* but also a list of elements that designers can modify in their games:

- **Formal elements**: The elements that differentiate a game from other types of media: rules, procedures, players, resources, objectives, boundaries, conflict, and outcome.
- **(Dynamic) systems**: Methods of interaction that evolve as the game is played.
- **Conflict structure**: The ways in which players interact with each other.
- **Uncertainty**: The interaction between randomness, determinism, and player strategy.
- **Unequal outcome**: How does the game end? Do players win, lose, or something else?

Another critical element in Fullerton’s book is her continual insistence on actually making games. The only way to become a better game designer is to make games. Some of the games you’ll design will probably be pretty awful—some of mine certainly have been—but even designing a terrible game is a learning process, and every game you create will improve your design skills and help you better understand how to make great games.

### Jesse Schell's Definition

Schell defines a game as "a problem-solving activity, approached with a playful attitude." This is similar in many ways to Suits' definition, including its consideration of the player’s perspective. According to both, it is the playful attitude of the player that makes something a game.

Suits argues in his book that two people could both be involved in the same activity, and to one, it would be a game, whereas to the other, it would not be. His example is a foot race where one runner is just running because she wants to take part in the race, but the other runner knows that at the finish line there is a bomb they must defuse before it explodes. According to Suits, although the two runners would both be running in the same foot race, the one who is simply racing would follow the rules of the race because of what Suits calls her *lusory attitude*. On the other hand, the bomb-defusing runner would break the rules of the game the first chance they got because they have a serious attitude (as is required to defuse a bomb) and are not engaged in the game. *Ludus* is the Latin word for play, so Suits proposes the term *lusory attitude* to describe the attitude of one who willingly takes part in playing a game.

It is because of their lusory attitude that players will happily follow the rules of a game even though there may be an easier way to achieve the stated goal of the game (what Suits would call the *pre-lusory goal*). For example, the pre-lusory goal of golf is to get the golf ball into the cup, but there are many easier ways to do so than to stand hundreds of yards away and hit the ball with a bent stick. When people have a lusory attitude, they set challenges for themselves just for the joy of overcoming them.

So, another design goal is to *encourage a lusory attitude*. You should design your game to encourage players to enjoy the limitations placed on them by the rules. Think about why each rule is there and how it changes the player experience. If a game is balanced well and has the proper rules, players will enjoy the limitations of the rules rather than feel exasperated by them.
Keith Burgun's Definition

Burgun's definition of a game as "a system of rules in which agents compete by making ambiguous, endogenously meaningful decisions" is his attempt to push the discourse on games forward from a rut that he feels it has fallen into by narrowing the meaning of game down to something that can be better examined and understood. The core of this definition is that the player is making choices and that those choices are both ambiguous (the player doesn't know exactly what the outcome of the choice will be) and endogenously meaningful (the choice is meaningful because it has a noticeable effect upon the game system).

Burgun's definition is intentionally limited and purposefully excludes several of the things that many people think of as games (including foot races and other competitions based on physical skill) as well as reflective games like *The Graveyard*, by Tale of Tales, in which the player experiences wandering through a graveyard as an old woman. Both of these are excluded because the decisions in them lack ambiguity and endogenous meaning.

Burgun chooses such a limited definition because he wants to get down to the essence of games and what makes them unique. In doing so, he makes several good points, including his statement that whether an experience is fun has little to do with the question of whether it is a game. Even a terribly boring game is still a game; it's just a bad game.

In my discussions with other designers, I have found that a lot of contention can exist around this question of what types of things should fall under the term *game*. Games are a medium that has experienced a tremendous amount of growth, expansion, and maturation over the last few decades, and the explosion of independent game development this decade has only hastened the pace. Today, more people than ever before—with disparate voices and varied backgrounds—are contributing work to the field of games, and as a result, the definition of the medium is expanding, which is understandably bothersome to some people because this expanding definition can be seen as blurring the lines of what is considered a game. Burgun's response to this is his concern that it is difficult to rigorously advance a medium if we lack a good definition of what the medium is. I'll come back to this topic in a little while.

Why Care About the Definition of Game?

In his 1953 book *Philosophical Investigations*, Ludwig Wittgenstein proposed that the term *game*, as it is used colloquially, had come at that time to refer to several very different things that shared some traits (which he likened to a family resemblance) but couldn't be encapsulated in a single definition. In 1978, Bernard Suits attacked this idea by using his book, *The Grasshopper*, to argue very stringently for the specific definition of game that you read earlier in this chapter. However, as Chris Bateman points out in his book *Imaginary Games*, though Wittgenstein used the word *game* as his example,
he was really trying to make a larger point: the point that words are created to define
tings rather than things being created to meet the definition of words.

In 1974 (between the publications of Philosophical Investigations and The Grasshopper),
the philosopher Mary Midgley published a paper titled, "The Game Game," in which
she explored and refuted the "family resemblance" claim by Wittgenstein not by argu-
ing for a specific definition of game herself but instead by exploring why the word game
existed. In her paper, she agrees with Wittgenstein that the word game came into being
long after games existed, but she makes the statement that words like game are not
defined by the things that they encompass but instead by the needs that they meet. As
she states:

> Something can be accepted as a chair provided it is properly made for sitting on,
whether it consists of a plastic balloon, a large blob of foam, or a basket slung from
the ceiling. Provided you understand the need you can see whether it has the right
characteristics, and aptness for that need is what chairs have in common.16

In her paper, Midgley seeks to understand some of the needs that games fulfill. She
completely rejects the idea that games are closed systems by both citing many exam-
pies of game outcomes that have effects beyond the game and pointing out that games
cannot be closed because humans have a reason for entering into them. To her, that
reason is paramount. The following are just a few reasons for playing games:

- **Humans desire structured conflict:** As Midgley writes, "The Chess Player's
desire is not for general abstract intellectual activity, curbed and frustrated by a
particular set of rules. It is a desire for a particular kind of intellectual activity, whose
channel is the rules of chess." As Suits pointed out in his definition, the rules that
limit behavior are there precisely because the challenge of those limitations is appeal-
ing to players.

- **Humans desire the experience of being someone else:** We are all acutely
aware that we have but one life to live (or at least one at a time), and play can allow
us to experience another life. Just as a game of Call of Duty allows a player to pretend
to experience the life of a soldier, so too does The Graveyard allow the player to pre-
tend to experience the life of an old woman, and playing the role of Hamlet allows
an actor to pretend to experience the life of a troubled Danish prince.

- **Humans desire excitement:** Much popular media is devoted to this desire for
excitement, be it action films, courtroom dramas, or romance novels. The thing
that makes games different in this regard is that the player is actively taking part in
the excitement rather than vicariously absorbing it, which is the only option for the
majority of linear media. As a player, you aren't watching someone else be chased by
zombies, you're being chased yourself.

---

Midgley found it critical to consider the needs that are fulfilled by games in order to understand both their importance in society and the positive and negative effects that games can have on the people who play them. Both Suits and Midgley spoke about the potentially addictive qualities of games in the 1970s, long before video games became ubiquitous and public concern emerged about players becoming addicted. As game designers, it is useful for us to understand these needs and respect their power.

The Nebulous Nature of Definitions

As Midgley pointed out, it is useful to think of the word *game* as being defined by the need that it fills. However, she also stated that a chess player doesn’t want to play just any kind of game; they specifically want to play chess. Not only is it difficult to come up with an all-encompassing definition for game, it’s also true that the same word will mean different things to different people at different times. When I say that I’m going to play a game, I usually mean a console or video game; when my wife says the same thing, though, she usually means *Scrabble* or another word game. When my parents say they want to play a game, it means something like Alan R. Moon’s *Ticket to Ride* (a board game that is interesting but doesn’t require players to be overly competitive with each other), and my in-laws usually mean a game of cards or dominoes when they use the word. Even within our family, the word has great breadth.

The meaning of the word *game* is also constantly evolving. When the first computer games were created, no one could have possibly imagined the multi-billion-dollar industry that we now have or the rise of the fantastic indie renaissance that we’ve seen over the past decade. All that they knew was that these things people were doing on computers were kind of like tabletop war board games (I’m thinking of *Space War* here), and these new games were called “computer games” to differentiate them from the pre-existing meanings of *game*.

The evolution of digital games was a gradual process with each new genre building in some way on the ones that had come before, and along the way, the term *game* expanded further and further to encompass all of them.

Now, as the art form matures, many designers are entering the field from various other disciplines and bringing with them their own concepts about what can be created with the technologies and design methodologies that have been developed to make digital games. (You may even be one of them.) As these new artists and designers enter the space, some of them are making things that are very different from what we think of as a stereotypical game. That’s okay; in fact, I think it’s fantastic! And, this isn’t just my opinion. IndieCade, the international festival of independent games, seeks every year to find games that push the envelope of what is meant by *game*. According to Festival
Chair Celia Pearce and Festival Director Sam Roberts, if an independent developer wants to call the interactive piece that they have created a game, IndieCade will accept it as one.17

**Summary**

After all these interwoven and sometimes contradictory definitions, you may be wondering why this chapter has spent so much time exploring the definition of the word *game*. I have to admit that in my day-to-day work as an educator and game designer, I don’t spend a lot of time wrestling with the definitions of words. As Shakespeare points out, were a rose to be named something else, it would still smell as sweet, still have thorns, and still be a thing of fragile beauty. However, I believe that an understanding of these definitions can be critical to you as a designer in the following three ways:

- **Definitions help you understand what people expect from your games.** This proves especially true if you’re working in a specific genre or for a specific audience. Understanding how your audience defines the term will help you to craft better games for them.

- **Definitions can lead you to understand not only the core of the defined concept but also the periphery (i.e., games that fit the definition perfectly (the core) and games that just barely fit the definition (the periphery)).** As you read through this chapter, you encountered several different definitions by different people, and each had both a core and a periphery. The places where these peripheries don’t mesh can be hints at some of the interesting areas to explore with a new game. For example, the area of disagreement between Fullerton and Midgley about whether a game is a closed system highlights the previously untracked ground that in the 2000s grew into alternate reality games (ARGs), a genre centered on perforating the closed magic circle of play.18

- **Definitions can help you speak eloquently with others in the field.** This chapter has more references and footnotes than any other in the book because I want you to be able to explore the philosophical understanding of games in ways that are beyond the scope of this one book (especially since this book is really focused on the practicalities of actually making digital games). Following these footnotes and reading the source material can help improve the critical thinking that you do about games.

---

17. This was stated during the Festival Submission Workshop given by Celia Pearce and Sam Roberts at IndieCade East 2014 and is paraphrased on the IndieCade submissions website at https://www.indiecade.com/submissions-help-section/eligibility/#mean-by-game (accessed June 14, 2021).

18. The first large-scale ARG was *Majestic* (Electronic Arts, 2001), a game that would phone players in the middle of the night and send them faxes and emails. Smaller-scale ARGs include the game *Assassin*, which is played on many college campuses, where players can "assassinate" each other (usually with Nerf or water guns, or by snapping a photo) any time that they are outside of classes. One of the fun aspects of these games is that they are always happening and can interfere with normal life.
The Core Lessons of This Book

This book will actually teach you how to design a lot more than just games. In fact, it will teach you how to craft any kind of interactive experience. As I define it:

> An interactive experience is any experience created by a designer; inscribed into rules, media, or technology; and decoded by people through play.

That makes interactive experience a pretty expansive term. In fact, any time that you attempt to craft an experience for people—whether you're designing a game, planning a surprise birthday party, or even planning a wedding—you're using the same tools that you will learn as a game designer. The processes that you will learn in this book are more than just the proper way to approach game design. They are a meaningful way to approach any design problem, and the iterative process of design that is introduced in Chapter 7, "Acting Like a Designer," is the essential method for improving the quality of any design.

No one bursts forth from the womb as a brilliant game designer. My friend Chris Swain\(^{19}\) is fond of saying that "Game design is 1% inspiration and 99% iteration," a play on the famous quote by Thomas Edison. He is absolutely correct, and one of the great things about game design (unlike the previously mentioned examples of the surprise party and the wedding) is that you get the chance to iterate on your designs, to playtest the game, make subtle tweaks, and play it again. With each prototype you make—and with each iteration of your prototypes—your skills as a designer will improve. Similarly, once you reach the parts of this book that teach digital development, be sure to keep experimenting and iterating. The code samples and tutorials are designed to show you how to make a playable game prototype, but every tutorial in this book will end where your work as a designer should begin. Each one of these prototypes could be built into a larger, more robust, better balanced game, and I encourage you to do so.

Moving Forward

Now that you've experienced a bit of game design and explored various definitions of game, it's time to move on to a more in-depth exploration of a few different analytical frameworks that game designers use to understand games and game design. The next chapter explores various frameworks that have been used over the past several years, and the chapter that follows synthesizes those into the framework used throughout the remainder of this book.

---

19. Chris Swain co-wrote the first edition of *Game Design Workshop* with Tracy Fullerton and taught the class of the same name at the University of Southern California for many years, which I took over from him in 2009. He is now an entrepreneur and independent game designer.
INDEX

Symbols
&& (AND operator), 406
< > (angle brackets)
C# programming language, 414–415
generic methods, 398
// (comments), 369, 384
{ } (braces)
C# programming language, 415
opening brace positioning in C#, 499–500
[ ] (brackets), C# programming language, 415
-- (decrement operators), 428, 431
$ (dollar sign), string interpolation, 981–982
= (Assignment operator), 411, 418
==(Is Equal To operator), 411–413
! (NOT operator), 406
!= (Not Equal To operator), 414
> (Greater Than operator), 414
>= (Greater Than or Equal To operator), 415
< (Less Than operator), 414
<= (Less Than or Equal To operator), 415
() (parentheses), C# programming language, 415
% (modulo operators), 436
|| (OR operator), 406
++ (increment operators), 428
; (semicolons)
debugging, 495
for loops, 431

Numbers
2D adventure game level
playtesting, 161–162
prototyping, 157–159
combat, 162
shortcuts, 161
traversal mechanics, 159–161
3D printing, aesthetics, 50
3x5 note cards, prototyping, 155
52 Pickup, 5

A
AAA development, costs, 293
absolute spreadsheet references, 193–194
access speeds, DOD, 577–578
accounts (Unity), creating, 334–335
accumulating points, Apple Picker, 665–668
Achievers (Bartle’s Diamonds), 74
acquaintances, circles of playtesters, 171–172
actions
calls to action, direct player guidance,
233–234
discernable actions, 71ive-act dramatic structures, 54
functions encapsulating actions, 476
integrated actions, 71, 147, 245
player actions, tracking, 61
puzzles, 250–251
rising action, five-act dramatic structures, 54
action lists, Apple Picker, 318
action puzzles, 260–262
Adkison, Peter
innovation, 113
ADL (Automated Data Logging), 182–183
Advanced Game Design: A Systems Approach,
1201
Advantage (Max) NES controllers, 36
aesthetics
cultural aesthetics, 93
cosplay, 93
fan art, 93
gameplay as art, 93
machinima, 93
cultural layer, Layered Tetrad, 36
Defiance, 36
dynamic aesthetics, 77
environmental aesthetics, 82–85
procedural aesthetics, 77–82
dynamic layer, Layered Tetrad, 34
Elemental Tetrad framework, 28
environmental aesthetics, 82
audio play environments, 83–84
player considerations, 84–85
visual play environments, 82–83
five aesthetic senses, 51
goals, 51–53
inscribed aesthetics, 51
inscribed layer, Layered Tetrad, 33
MDA framework, 33
procedural aesthetics, 77
procedural music, 78–79
procedural visual arts, 80–82
Tomb Raider, 36
ages/genders, digital games industry, 291–292
Agile Software Development. See also Scrum, 266–267
agon, 134
Aguilar, Chris
Vectorized Playing Cards, 2
alea, 134
Alexander, Christopher
design patterns, 49
A Pattern Language, 49
purpose of spaces, 49
alpha phase, game development, 119
ambiguous decisions, 147
Among Us, player relationships, 47
ampersands (&), && (AND operator), 406
Amplitude, VRO, 78–79
analysis
Apple Picker, 316
basic gameplay, 317
GameObjects, 317–318
GameObjects, action lists, 318
GameObjects, flowcharts, 319–321
Elemental Tetrad framework, 20, 30–31
FDD elements framework, 20, 25–27
dramatic elements, 25, 27–28
dynamic elements, 25, 28–29
formal elements, 27–28
frames, 319–320
MDA framework, 22
defined, 23
designer views, 23
player views, 23
Snakes and Ladders, 22–24
playtesting, 5–6, 7–8
self-analysis, playtesting, 169
analysis phase, iterative design, 105–107
AND operator (&&), 406
angle brackets (< >)
C# programming language, 414–415
generic methods, 398
animation
blending, 81
Dungeon Delver, 1044–1049
attack animations, 1055–1058
walking animations, 1054–1055
procedural animation, 81
antagonism, three-act dramatic structures, 58
anti-aliasing issues, Dungeon Delver, 1041–1042
Apache OpenOffice Calc, 190
A Pattern Language, 49
Apple Numbers, 190
Apple Picker, 316, 621–623
art assets, 624–633
basic gameplay, 317
baskets
destroying, 670–672
instantiating, 655–656
moving with mouse, 657–658
Cameras, setup, 633–634
catching apples, 658–659
coding, 637–641
directionality, 646–648
dropping apples, 649–651
game management, 661–662
game panel settings, 636–637
GameObjects, 317–318
action lists, 318
flowcharts, 319–321
GUI, 661–662
HighScore texts, 662–664, 672–678
instance overrides, applying to prefabs, 641–642
missed apple notifications, 668–672
movement systems, 643–646
physics layers, 651–652
points accumulation, 665–668
purpose of, 623
random directionality, 647–648
ScoreCounter texts, 662–664
script variables, tuning, 659–660
setup, 624
stopping apples from falling too far, 653–655
application variables, 397
applications, force quitting, 426, 509
approximate float comparisons, 412
archipelagos, turning noise into, 603–612
ARG (Alternate Reality Game)
boundaries, 46–48
Majestic, 17
arguments, functions, 478–479
arrays, 439, 451–453
choosing, 466–467
converting
Lists to arrays, 447
arrays to Lists, 457
empty elements within arrays, 453
jagged arrays, 461–464
linear arrays, storing two-dimensional data in, 1039–1040
multidimensional arrays, 457–461
applications, force quitting, 426, 509
approximate float comparisons, 412
archipelagos, turning noise into, 603–612
ARG (Alternate Reality Game)
boundaries, 46–48
Majestic, 17
arguments, functions, 478–479
arrays, 439, 451–453
choosing, 466–467
converting
Lists to arrays, 447
arrays to Lists, 457
empty elements within arrays, 453
jagged arrays, 461–464
linear arrays, storing two-dimensional data in, 1039–1040
multidimensional arrays, 457–461
null arrays, skipping with foreach loops, 454–455
properties, 455–456
static methods, 456–457
zero-indexed arrays, 440
arrows (visual design), indirect player guidance, 237–238
art
Apple Picker, 624–633
face art, adding to cards, 936–937
fan art, 93
gameplay as art, 93
games as art, 97–98
Mission Demolition, 684–690
procedural visual arts, 80
particle systems, 80
procedural animation, 81
procedural environments, 82
shaders, 81–82
Space SHMUP, enemies, 771–773
Art of Game Design, The, 9, 2
design phase, iterative design, 108–109
Elemental Tetrad framework, 30–31
indirect player guidance, 234–240
inscribed mechanics, 43
interest curves, 145–146
"Ten Rules of Probability Every Game Designer Should Know," 207–211
testing phase, iterative design, 111–112
"Art of Puzzle Design," The, 248–250
Assassin's Creed
audio design, 240
plots versus free will, 57
visual design, indirect player guidance, 237
Assassin's Creed IV: Black Flag
direct player guidance, 234
embedded experiences, 48
Assassin's Creed: Odyssey, HDR lighting, 83
asset packages, importing, 755–757
assets, 49
assigning tasks (Main worksheets), BDC, 274–275, 280
Assignment operator (=), 411, 418
asymmetric games, 188
AT (Automated Testing), 185
Attach to Unity button, repairing in Visual Studio, 513
attaching scripts, 500–502
attack animations, Dungeon Delver, 1055–1058
"Attention, Not Immersion: Making Your Games Better with Psychology and Playtesting, the Uncharted Way," 144–145
attention/involvement, player-centric goals, 145–147
Attractor GameObject, Boids project, 549–551
Attractor script, Boids project, 551, 553–555, 567–570
attributes, 49
audio
background noise, 53
design, indirect player guidance, 240
dialogue, 52
five aesthetic senses, 49–53
music, 52
play environments, 83–84
noisy environments, 84
player-controlled game volume, 84
sound effects, 50–52
authorized transmedia and cultural layer, 96–97
autoformatting for loops, 656
scripts, Visual Studio, 365–366
backlogs (product)/task lists, Scrum, 269
backs, adding to cards, 937–938
balance
game balance
difficulty, levels of, 188
feedback, 228–229
Mario Kart, 228
meaning of, 188
Monopoly, 228–229
spreadsheets, importance of, 188–189
weapon balance
average damage, 222–224
duplicating data, 225–226
example of, 227–228
Google Sheets, 219–228
overall damage, showing, 224–225
percent chance for each shot, 220–221
rebalancing weapons, 226–227
Bartle, Richard
player intent, 74–75
types of players, 74–75

Bartok, 2
analysis, playtesting, 5–6, 7–8
deblocking, 5
digital version, 3
emergence, 69
house rules, 73–74
layout, 4
objective, 3
playtesting, 4–5
procedures, 70
rules, 3, 6–7

baskets, Apple Picker
destroying, 670–672
instantiating, 655–656
moving with mouse, 657–658

BDC (Burndown Charts), 269, 271–272
creating, 286
Daily Scrum worksheets, 283–285
Main worksheets, 273
estimating hours, 274–276, 277–280
sprint progress, 277–279
sprint settings, 273–274
task assignments, 274–275, 280
totalling hours, 277–280

Person Charts, 282–283
Task Rank Charts, 280–282
workheets, 272–273

behaviors
behavioral change, games for, 133
NPC behaviors, modeling
emotional connections, 240–241
negative behaviors, 241
positive behaviors, 241
safety, 241

beta phase, game development, 119
Bethesda SoftWorks
Elder Scrolls, The, 92
Fallout, narrative game mods, 94
Fallout 3, 92
Fallout 4, 58
Skyrim, 58, 92, 94
conflicting objectives, 44–45
optional objectives, 42–44
primary objectives, 42–44

Betrayal at House on the Hill, player
relationships, 44, 47
BézierMover class, Prospector, 987–991
biases, playtesting, 169
BioWare

Mass Effect
multiple dialogue choices, 61–64
novel decisions, 148–149
player interaction patterns, 46

Star Wars: Knights of the Old Republic, plots
versus free will, 57

bitwise Boolean operators, 409

Blade Runner, multiple dialogue choices, 63–64
blending, animation, 81

Blizzard, Defense of the Ancients (DotA), 92

board games, systems thinking, 312

Bogost, Ian
magick circle, 138

Boids project
Attractor GameObject, 549–551
Boids values, 573
Reynolds, Craig W.542
scripts, 551
Attractor script, 551, 553–555
Boid script - part 1, 558
Boid script - part 2, 561–567
Boid script - part 3, 570–573
LookAtAttractor script, 551, 557–558
Neighborhood script, 551, 567–570
Spawner script, 551, 558–561
setup, 542–543
simple Boid model, 543–548
bool variables, 386

Boolean operations
AND operator (&&), 406
bitwise Boolean operators, 409
combining, 409–410
if statements with, 417–418
logical equivalence, 410
NOT operator (!), 406
| (OR operator), 406
boss fights, puzzle design, 261–262
Boston Red Sox, 2013 season, 34

boundaries
FDD elements framework, 26
inscribed mechanics, 40, 46–48
BoundsCheck bndCheck, Space SHMUP, 816–819
Box Collider component, GameObjects, 371
braces ([ ]), C# programming language, 415
opening brace positioning in C#499–500
brackets ([ ]), C# programming language, 415
brainstorming/ideation, 113–114
collection phase, 115–116
collision phase, 116–117
discussion phase, 117
expansion phase, 114–115
idea cards, 115
nodes, 115
rating phase, 117
break statements, exiting loops, 433–435
brevity, direct player guidance, 233
Brice, Mattie
experiential understanding, 149–150
Mainichi, 149–150
brightness
visual design, indirect player guidance, 239
visual play environments, 83
Brigs, Jeff
C.P.I. Bach, 79
building games. See also game prototype
tutorials; projects
2D adventure game level
combat, 162
playtesting, 161–162
prototyping, 157–159
shortcuts, 161
traversal mechanics, 159–161
Apple Picker, 316
basic gameplay, 317
GameObjects, 317–318
GameObjects, action lists, 318
GameObjects, flowcharts, 319–321
classic games, building as a learning
elementary example, 1199
frames, 319–320
for lifelong enrichment, 1200
small game projects, 1199
uroboros
collection phase, brainstorming/ideation, 115
collision phase, brainstorming/ideation, 116–117
discussion phase, brainstorming/ideation, 117
expansion phase, brainstorming/ideation, 114–115
idea cards, 115
idea collisions, 116–117
rating phase, brainstorming/ideation, 117
Bulls & Cows
image/media puzzles, 253
permutations, 217–219
Burgun, Keith
ambiguous decisions, 147
fun, elements of, 134–135
Game Design Theory, 9–10, 93, 134–135, 144
games, defined, 14
performative empowerment, 144
burndown charts (BDC), 269, 271–272
creating, 286
Daily Scrum worksheets, 283–285
Main worksheets, 273
estimating hours, 274–276, 277–280
sprint progress, 277–279
sprint settings, 273–274
task assignments, 274–275, 280
totalling hours, 277–280
Person Charts, 282–283
Task Rank Charts, 280–282
worksheets, 272–273
Burnout
particle systems, 80
puzzle design, 260–261
Burst Compiler, DOTS, 599–600
C
C# programming language
angle brackets (< >), 414–415
arrays, 451–453
arrays, 439
choosing, 466–467
converting Lists to, 447
converting to Lists, 457
empty elements within arrays, 453
jagged arrays, 461–464
multidimensional arrays, 457–461
null arrays, skipping with foreach loops,
454–455
properties, 455–456
static methods, 456–457
zero-indexed arrays, 440
Boolean operations
AND operator (&&), 406
bitwise Boolean operators, 409
combining, 409–410
if statements with, 417–418
logical equivalence, 410
NOT operator (!), 406
OR operator (|), 406
brackets ([ ]), 415
choosing, 330–331
classes, 521
anatomy of, 522–524
Class Declarations, 523
constructors, 524
C# programming language (continued)

Enemy class on GameObjects, 534
fields, 523
fields, methods/properties as fields, 527–530
includes, 523–524
inheritance, 533
instances, 391–392
matching names with scripts, 525–526
methods, 523
methods/properties as fields, 527–530
MonoBehavior subclasses as GameObject components, 530–533
properties, 524
race conditions, 533
subclasses, 535–538, 950
superclasses, 535–538, 950
understanding, 522
viewing private fields in, 734
WeaponDefinition class, SpaceSHMUP, 834–842

collections
commonly used collections, 439–440
defined, 438
generic collections, 438, 442–443
commsas (,), in statements, 432
comments (/), 369, 384
comparison operators, 410
approximate float comparisons, 412
Assignment operator, 411, 418
Greater Than operator (>, 414
Greater Than or Equal To operator (>=), 415
Is Equal To operator (==), 411–413
Less Than operator (<), 414
Less Than or Equal To operator (<=), 415
Not Equal To operator (!=), 414
as a compiled language, 348–350
conditional statements, 416
if statements, 416
if statements, = (Assignment operator), 418
if statements, with Boolean operations, 417–418
if else if else statements, 418–419
if else statements, 418
nesting if statements, 419
switch statements, 419–422
CS0029 compile-time code errors, 388
CS0664 compile-time code errors, 387
CS1012 compile-time code errors, 388
CS1525 compile-time code errors, 388

C# as function-based language, 352–353
calling, 476–477
definition, 474–476
defining order, 480
encapsulating actions, 476
mathf functions, 396
naming, 482
overloading, 485–486
parameters, 478–479
parameters, optional parameters, 486–487
parameters, params keyword, 487–489
as properties, 483–484
reasons for using, 482–483, 484–485
recursive functions, 489–491
returning values (results), 480
returning void, 481–482
scope, 476
static functions, 391–392
generic methods (< >), 398
increment operators (++), 428
is managed code, 351
Lists, 439–440, 443–446
choosing, 466–467
converting arrays to, 457
converting to arrays, 447
jagged Lists, 465–466
methods, 446–447
properties, 446
zero-indexed lists, 440

debugging, 494
attaching scripts, 500–502
capitalization errors, 494
compile-time bugs, 495–500
examining code, 513–519
removing scripts, 500–502
runtime errors, 502–504
semicolons (;), 495
spelling errors, 494
stepping through errors, 506–507
typos, 494–495
Dictionaries, 441, 447–450
methods, 450–451
properties, 450
enums (enumeration), 742
features (overview), 348
force quitting applications, 426

C# PROGR AMMING LANGUAGE
C# programming language (continued)

loops, 423
break statements, 433–435
condition clauses (i<3), 430
continue statements, 435
do.while loops, 424, 429
exiting, 433–435
for loops, 424, 429–431, 432, 656
for loops, jagged arrays, 464–465
foreach loops, 424, 433
foreach loops, skipping null arrays,
454–455
infinite loops, 425–427
initialization clauses (int i=0;), 430
iteration clauses (i++), 430, 431
jump statements, 433
Loop Examples project, 424–426
modulo operators (%), 436
skipping single iterations, 435
types of (overview), 424
while loops, 424, 425, 426–428
modulo operators (%), 436
naming conventions, 389–390
nonshorting operators, 407–409
opening brace positioning (499–500
parentheses ( ), 415
percentage symbols (%), % (modulo
operators), 436
plus signs (+), ++ (increment operators), 428
private fields, viewing in classes, 734
pseudocode, 440
queues, 441
scripts, 402
adding color, 381–382
Boids project, Attractor script, 551,
553–555
Boids project, Boid script, 551
Boids project, Boid script - part 1, 558
Boids project, Boid script - part 2, 561–567
Boids project, Boid script - part 3, 570–573
Boids project, LookAtAttractor script, 551,
557–558
Boids project, Neighborhood script, 551,
567–570
Boids project, Spawner script, 551,
558–561
creating, 363–368
disabling, 370
enemies, Space SHMUP, 773–787
execution order, 1040
GridMove scripts, Dungeon Delver,
1085–1087
headers, 553–555
InRoom scripts, Dungeon Delver, 1070–1072
linear interpolation, 567
manipulating GameObjects, 370–373
matching names with classes, 525–526
prefabs, 373–378
Space SHMUP, projectiles, 803
TileSwapManager scripts, Dungeon Delver,
1099–1101
UITextManager scripts, 1010–1013
Visual Studio, autocompleting scripts,
365–366
Visual Studio, script appearance, 365–366
Visual Studio, spacing, 375
shorting operators, 407–409
stacks, 441–442
start() function versus update() function,
370–398
static typing, 355–357
string interpolation using $981–982
syntax of, 355–357
testing operation equality by value/
reference, 412–413
variables, 384
application variables, 397
bool variables, 386
color variables, 388
defining, 385
delay time in script, 369
deleting, 370
exceptions, 381
for loops, jagged arrays, 464–465
full update(), 380
iteration variables, Loop Example project,
428
naming, 402
null coercion, 381
quaternion variables/functions, 395–396
release time in script, 369
return type in script, 369
returning values, 378
scope, 389
defining static class level, 390–392
static class variables/functions, 390–392
statically typed variables, 384–385
string variables, 388
SystemInfo variables, 397
Vector3 instance variables/functions, 393
adding color, 381–382
Boids project, Attractor script, 551,
553–555
Boids project, Boid script, 551
Boids project, Boid script - part 1, 558
Boids project, Boid script - part 2, 561–567
Boids project, Boid script - part 3, 570–573
Boids project, LookAtAttractor script, 551,
557–558
Boids project, Neighborhood script, 551,
567–570
Boids project, Spawner script, 551,
558–561
creating, 363–368
disabling, 370
enemies, Space SHMUP, 773–787
execution order, 1040
GridMove scripts, Dungeon Delver,
1085–1087
headers, 553–555
InRoom scripts, Dungeon Delver, 1070–1072
linear interpolation, 567
manipulating GameObjects, 370–373
matching names with classes, 525–526
prefabs, 373–378
Space SHMUP, projectiles, 803
TileSwapManager scripts, Dungeon Delver,
1099–1101
UITextManager scripts, 1010–1013
Visual Studio, autocompleting scripts,
365–366
Visual Studio, script appearance, 365–366
Visual Studio, spacing, 375
shorting operators, 407–409
stacks, 441–442
start() function versus update() function,
370–398
static typing, 355–357
string interpolation using $981–982
syntax of, 355–357
testing operation equality by value/
reference, 412–413
variables, 384
application variables, 397
bool variables, 386
color variables, 388
defining, 385
delay time in script, 369
deleting, 370
exceptions, 381
defining static class level, 390–392
static class variables/functions, 390–392
statically typed variables, 384–385
string variables, 388
SystemInfo variables, 397
Vector3 instance variables/functions, 393
CALLS TO ACTION, DIRECT PLAYER GUIDANCE

calls to action, direct player guidance, 233–234
cameras
   Apple Picker, setup, 633–634
   Dungeon Delver, 1023
   following Dray (hero), 1091–1094
   GUI cameras, 1024–1025
   main camera, 1024–1025
   Mission Demolition
      follow cameras, 702–710
      settings, 685–687
   orthographic cameras, 634–636
   perspective cameras, 634–636
   Prospector, 906–907
   view frustum, 634
   visual design, indirect player guidance, 238–239
capitalization errors, debugging, 494
card sleeves, prototyping, 155
cards
   3x5 note cards, 155
   52 Pickup, 5
   deblocking, Bartok, 5
   prototyping, 153
   riffle shuffles, 5
cards, decks of
   customizing, 214
digital decks of cards, 214
   Prospector Solitaire, 898–899, 969–970
      adding backs to cards, 937–938
      adding face art to cards, 936–937
      adding game elements, 972
      background images, 983–985
      BézierMover class, 987–991
      build settings, 903
      building cards, 922–938
      cameras, 906–907
      classes, 948–961
      clickable cards, 962–964
      example of play, 900–901
      feedback on player scores, 1007–1013
      FloatingScore GameObject, 991–999
      game logic, 961–962
      Game pane, 906–907
      gold cards, 1017
      GUI, 985–986
      initial layout, 899–900
      JSON through code, 913–917
      managing rounds, 972–975
      matching cards in mine, 964–968
      Mine Tableau layout, 940–948
      mobile devices, 1018
      moving cards, 1017–1018
      pauses between rounds, 1006–1007
      pips, adding to cards, 934–935
      Prospector_Scene_0, 905
      rules, 900
      ScoreBoard class, 1000–1001
      ScoreBoard GameObject, 999
      scoring, 975–983, 985–986, 999–1006,
      1007–1013
      setup, 901–902, 906–907, 971
      shuffling cards, 939–940
      silver cards, 1017
      sorting cards, 954–958
      sprites, building cards from sprites, 931–934
      sprites, constructing cards from sprites, 911–912
      sprites, gathering references to the deck, 918–920
      sprites, importing images as, 907–909
      sprites, prefab GameObjects as sprites/cards, 921–922
      sprites, slicing rank images as sprites, 909–911
      Unity window layout, 906
      updating ScoreManager script, 1001–1006
      WebGL module, 1013–1016
      WebGL module, installing, 903–904
      WebGL module, switching to, 904–905
      randomizer technologies, 213–215
      shuffling, 215
      Carnegie Mellon University, Entertainment Technology Center (ETC), 297–298
      Cash, Bryan
      sporadic-play games, 135–136
      castles, Mission Demolition, 717–725,
      734–737
      catching apples, Apple Picker, 658–659
      CCG (Collectible Card Games), 113
      cells, editing contents in spreadsheets, 198
      Cerny, Mark
      Cerny Method, 121
      holistic design, 125
      scope management with preproduction deliverables, 121
      chain reaction games, puzzle design, 260–261
      changing
         direction, Apple Picker, 646–648
         Scene pane, 380
         script values in Inspector pane, 660–661
         your mind, iterative design, 117–118
      char variables, 387
characters
- empathetic characters, avatars versus, 61–62
- FDD elements framework, 26
- inscribed narratives, 52
- NPC, minor NPC development, 58–59

charts
- average damage, 223–224
- BDC, 269, 271–272
- creating, 286
- Daily Scrum worksheets, 283–285
- Main worksheets, 273–280
- Person Charts, 282–283
- Task Rank Charts, 280–282
- worksheets (overview), 272–273
- Google Sheets, 204–206
- Macro Charts, 126
- Task Rank Charts, BDC, 280–282

Cheap Ass Games, touch aesthetics, 50

cheaters, 75

Chen, Jenova
- game flow, 138
- Journey, 171

choice paralysis
- 148–149, 234–235

choosing
- Lists or arrays, 466–467
- Unity, 329–330

Chowanec, John "Chow"
- fortune, designer-centric goals, 131

Chrono Trigger, plots versus free will, 58

Chutes and Ladders, 27–29

Cialdini, Robert, 2

circles of playtesters, 169
- acquaintances, 171–172
- Internet, 172
- trusted friends, 170
- you, 170

citizens, player relationships, 45

Civilization
- inscribed mechanics, 43
- tech tree, 43

clarity
- direct player guidance, 233
- in spreadsheets, 199

classes, 521
- anatomy of, 522–524
- BézierMover class, Prospector, 987–991
- Class Declarations, 523
- constructors, 524
- data stored by reference, 579–580
- Enemy class on GameObjects, 534
- fields, 523, 527–530
- Includes, 523–524
- inheritance, 354, 533
- instances, 391–392
- matching names with scripts, 525–526
- methods, 523, 527–530
- MonoBehavior subclasses, as GameObject components, 530–533
- properties, 524, 527–530
- Prospector, 948–961
- race conditions, 533
- ScoreBoard class, Prospector, 1000–1001
- subclasses, 535–538, 950
- superclasses, 535–538, 950
- understanding, 522
- Unity classes, data storage, 580
- variables, 388
- viewing private fields in, 734
- WeaponDefinition class, SpaceSHMUP
dictionaries, 838–842
- serializable, 834–838

classic games, building as a learning example, 1199

clear decisions, 148

clickable cards, Prospector, 962–964

climaxes
- five-act dramatic structures, 54
- three-act dramatic structures, 58

closed games, 11

Clover Studio, Okami, 60

Clubs (Bartle’s Killers), 74

Clue, flow of spaces, 47

coding
- iterative code development, 793–794
- libraries, 314–315

Coding Challenges
- approaches to, 1194–1195
- defined, 1190–1191
- Updraft Coding Challenge
  - filling in blanks, 1192–1194
  - starting, 1191–1192

collaborative prototyping, 152

collaborators, player relationships, 45

Collectible Card Game (CCG), 113

collection phase, brainstorming/ideation, 115–116

collections
- commonly used collections, 439–440
- defined, 438
- generic collections, 438, 442–443
- Collections Examples project, setup, 442–443
- Collider component, GameObjects, 400–401
collision detection, Mission Demolition, 698–699
collision phase, brainstorming/ideation, 116–117
collisions, Grappler (Dungeon Delver), 1169–1173
CollisionTiles sprites, Dungeon Delver, 1061–1064
color
"Hello World" project, 381–382
Unity, adding to projects, 381–382
variables, 393–395
visual design, indirect player guidance, 239
color scale conditional formatting, Google Sheets, 200
colorblindness, 85
columns/rows, Google Sheets
adding columns, 194–195
creating rows, 194
filling rows with data, 196
iterating Die A rows, 197
making Die A rows, 196–197
making Die B rows, 197–198
setting column widths, 195
combat, 2D adventure game level, 162
combining Boolean operations, 409–410
commas (,) in statements, 432
comments (//), C# scripts, 369, 384
communication/personal expression, designer-centric goals, 132–133
community, designer-centric goals, 132, 146–147
comparing rounds, analysis, 7–8
comparison operators, 410
approximate float comparisons, 412
Assignment operator (=), 411, 418
Greater Than operator (>), 414
Greater Than or Equal To operator (>=), 415
Is Equal To operator (==), 411–413
Less Than operator (<), 414
Less Than or Equal To operator (<=), 415
Not Equal To operator (!=), 414
competition
multilateral competition interaction pattern, 44
player relationships, 44
team competition interaction pattern, 44
unilateral competition interaction pattern, 44
compile-time bugs, 495–500
compile-time code errors
CS0029 compile-time code errors, 388
CS0664 compile-time code errors, 387
CS1012 compile-time code errors, 388
CS1525 compile-time code errors, 388
complex problems, breaking down, 315
component-based design, Dungeon Delver, 1021–1022
Component-Oriented Design, 552–553
computer languages, 313–314
"Concept of Flow," The, 139
costs, AAA development, 293
Counter Strike, game mods, 92
counting
all die rolls, 202–203
sums of die rolls, 202–203
counting coup, 141–142
C.P.U. Bach, PCO, 79
Crazy Cakes, ADL, 182–183
Csikszentmihályi, Mihaly
autotelic empowerment, 143
"Concept of Flow," The, 139
Flow: The Psychology of Optimal Experience, 139–140
game flow, 138–140
cubes, "Hello World" project
cube environments, 378–381
deleting cubes, 467–471
manipulating, 370–373
prefabs, 373–378
shrinking cubes, 467–471
cultural aesthetics, 93
cosplay, 93
fan art, 93
gameplay as art, 93
machinema, 93
cultural impact of games, 97–98
cultural layer, Layered Tetrad, 36–37, 90–91
aesthetics, 36
authorized transmedia and, 96–97
cultural aesthetics, 93–0042
mechanics, 91–92
cultural narratives, 36, 93–94–0053
cultural technology, 95–96
GamerGate, 100–101
mechanics, 35–38
narratives, 36
technology, 36
cultural mechanics
custom game levels, 92
game mods, 91–92, 92
cultural narratives, 93–94
fan fiction, 95
machinema, 94–95
narrative game mods, 94
cultural technology, 95
game technology used outside games, 95
player-made external tools, 95–96
cumulative outcomes, 76
customizing
decks of cards, 214
game levels, 92
D
Daily Scrum meetings, 268, 269–270
Daily Scrum worksheets, BDC, 283–285
damage
average damage
charting, 223–224
Dungeon Delver, 1125–1135
overall damage, showing, 224–225
Space SHMUP, 792–797, 853–857
Damage Per Second (DPS) calculators, 95
dashes (--), (decrement operators), 428, 431
Data-Oriented Design (DOD)
access speeds, 577–578
cache lines, 578–579
data locality, 577–578
"Moore's Law," 576
parallel processing, 576
theory of, 576
Unity, 354–355
Data-Oriented Tech Stack (DOTS), 581–582
archipelagos, turning noise into, 603–612
Burst Compiler, 599–600
DOTS Example project, setup, 582–586
example of, 581–582
future of, 617
image creation, 593–602
noise
archipelagos, turning noise into, 603–612
avoiding octaves, 600–602
Perlin noise, 593–602
reference-based data, avoiding, 595–599
tutorial, 581–582
Dean, Dr. Jeff
"Numbers Everyone Should Know," 577
debugging, 494
deblocking, 494
attaching scripts, 500–502
capitalization errors, 494
compile-time bugs, 495–500
examining code, 513–519
macOS, 507–508, 510–511
removing scripts, 500–502
runtime errors, 502–504
semicolons (;), 495
spelling errors, 494
stepping through errors, 506–507
typos, 494–495
Unity, 510
enabling, 509–510
errors, 505
macOS debuggers, 510–511
Windows debugger, 511–513
variables, 517–518
Windows, 507–508, 511–513
decisions
ambiguous decisions, 147
clear decisions, 148
discernable decisions, 147
double-edged decisions, 148
integrated decisions, 147
interesting decisions, 12, 147–149
novel decisions, 148
decks of cards
  customizing, 214
digital decks of cards, 214
randomizer technologies, 213–215
shuffling, 215
declaring
  classes, 523
  variables, 385
decrement operators (–), 428, 431
Defense of the Ancients (DotA), game mods, 92
Defiance, aesthetics, 36
defining variables, 385
delegate events, Space SHMUP, 842–844
deleting
  cubes, "Hello World" project, 467–471
enemies, Space SHMUP, 777–787
DeliverTiles, Dungeon Delver, 1026–1028
DelverLevel_Eagle Text files, Dungeon Delver, 1028–1031, 1033–1035
denouement, five-act dramatic structures, 54
design
  aesthetics, 23
  audio design, indirect player guidance, 240
  component-based design, Dungeon Delver, 1021–1022
Component-Oriented Design, 552–553
game design, 1201
  joining projects, 305
  royalty points, 306–307
  starting projects, 305–308
goals, 130
  designer-centric goals, 130, 131–134
  player-centric goals, 130, 134–150
holistic design, 125
iterative design, 104
  analysis phase, 105–107
  changing your mind, 117–118
  design phase, 104, 107–109
  implementation phase, 105, 111
  interpreting feedback, 112
  testing phase, 105, 112
patterns, 50
puzzle design, 248, 250–251, 262–263
  action puzzles, 250–251, 260–262
  boss fights, 261–262
  construction puzzles, 251
  construction sets, 256
  defining, 248–250
  dexterity/timing, 262
  goals, 256–257
  image/media puzzles, 253
  inspiration, 253
Kim, Scott, 248–250, 255–256
  levels, 255
  logic puzzles, 253
  mixed-mode puzzles, 254
  modes of thought, 252
  physics puzzles, 260
  presentation, 256
  pure puzzles, 251
  reasons for playing, 251–252
  rules, 255
  sequencing, 256
  simplification, 255
  single-mode puzzles, 253
  sliding block/position puzzles, 260
  solving puzzles, 257–259
  stealth puzzles, 261
  story puzzles, 251
  strategy puzzles, 251
  testing, 255
  Tetris, 255
  traversal puzzles, 261
  word puzzles, 253
reality of, 117–118
strategies, 73
systems design, 118–119
visual design, indirect player guidance, 236–239
Design Patterns: Elements of Reusable Object-Oriented Software, 552
design phase, iterative design, 104, 107–109
designer-centric goals, 130
  community, 146–147
  fortune, 130
  community, 132
  fame, 131–132
  greater good, 133
  personal expression/communication, 132–133
  improving as a game designer, 134
  designer views, MDA framework, 23
  designers, responsibilities, Layered Tetrad, 39–40
destroying enemies, Space SHMUP, 804
developing games, 118, 120
  AAA development, costs, 293
  Agile Software Development, 266–267
  alpha phase, 119
  beta phase, 119
  education/programs, 1200–1201
  gold phase, 119
  ideation phase, 118–119
indie gaming scene, 295
post-release phase, 119
preproduction phase, 118–119
production phase, 118–119
Unity, 315
development environment, Unity, 328
development speeds, prototyping, 152–153
development teams, Scrum, 268
dexterity/timing, puzzle design, 262
Diaconis, Persi, 5
dialogue, 52, 61
Diamante, Vincent
Flower, 79
Diamonds (Achievers), 74
dice
probability with Google Sheets, 191
prototyping, 153
randomizer technologies, 212
summing results
  counting all die rolls, 202–203
  counting sums of die rolls, 201–202
two dice, 201
Dictionaries, 441, 447–450
methods, 450–451
properties, 450
WeaponDefinition class, 838–842
difficulty levels, game balance, 188
digital decks of cards, 214
Digital Extremes, Warframe, 93
digital games
industry
AA development costs, 293
ages/genders, 291–292
conditions in, 289–295
Entertainment Software Association (ESA), 288–289
following up with contacts, 301
freemium games, 295
game conferences, 301
game education/programs, 296–299
games as a service, 295
getting into, 299–308
growth, 289–290
indie gaming scene development, 295
interviewing, 302–305
joining game design projects, 305
meeting people in the industry, 300
royalty points, 306–307
starting game design projects, 305–308
working conditions in game companies, 292–293
inscribed technology, 65
digital systems/programming
breaking down complex problems, 315
code libraries, 314–315
computer languages, 313–314
simple instructions, 313
systems thinking, 312
Unity game development environment, 315
DigitalMania, Warshmallows, 179–180
direct player guidance, 232
brevity, 233
calls to action, 233–234
clarity, 233
immediacy, 232
instructions, 233–234
maps/guidance systems, 233–234
pop-ups, 234
scarcity, 232
directional light, Mission Demolition, 685
directionality
Apple Picker, 646–648
random directionality, Apple Picker, 647–648
visual design, indirect player guidance, 239
discernable actions/decisions, 71, 147
discussion phase, brainstorming/ideation, 117
Disneyland, visual design and indirect player guidance, 236–237
distributions, weighted, 215–216
Doctor Who, foreshadowing, 58
documentation
Macro Documents, 124–126
Requirements Documents, 124–126
DOD (Data-Oriented Design)
access speeds, 577–578
cache lines, 578–579
data locality, 577–578
"Moore's Law," 576
parallel processing, 576
theory of, 576
dollar sign ($), string interpolation, 981–982
don't like/liking ideas, playtesting, 169
doors, Dungeon Delver
keys, 1111–1121, 1138–1140
TileSwaps, 1101–1107
DOTS (Data-Oriented Tech Stack), 581–582
archipelagos, turning noise into, 603–612
Burst Compiler, 599–600
DOTS Example project, setup, 582–586
example of, 581–582
future of, 617
image creation, 593–602
noise
  archipelagos, turning noise into, 603–612
  avoiding octaves, 600–602
Perlin noise, 593–602
reference-based data, avoiding, 595–599
tutorial, 581–582
double-edged decisions, 148
doubles, 386–387
do.while loops, 424, 429
downloading
IGDPD layout, Unity, 340
Unity, 324
Unity Hub, 324–326
dramatic elements, 25, 26–28
characters, 26
premises, 28
stories, 26
dramatics
inscribed dramatics, purposes for, 62–64
traditional dramatics, 55
five-act dramatic structures, 54–55
three-act dramatic structures, 56–58
Dray (hero), Dungeon Delver, 1042
animation, 1044–1049
attack animations, 1055–1058
walking animations, 1054–1055
camera movement, 1091–1094
collisions, 1069
giving damage, 1130–1135
Grappler attacks, 1174–1180
GUI connections, 1123–1125
health, 1121–1122
IGadget interface, 1150–1154
movement systems, 1049–1053, 1059–1061, 1087–1091
naming conventions, 1043–1044
picking up items, 1135–1137, 1181–1184
taking damage, 1127–1130
weapons, 1059–1061
dropping
apples, Apple Picker, 649–651
items, Dungeon Delver, 1026–1028
dungeon design, 1019–1021, 1095–1096.
See also The Legend of Zelda
anti-aliasing issues, 1041–1042
cameras, 1023
GUI cameras, 1024–1025
main camera, 1024–1025
component-based design, 1021–1022
damage, 1125–1135
DeliverTiles, 1026–1028
DeliverLevel_Eagle Text files, 1028–1031, 1033–1035
Dray (hero), 1042
animation, 1044–1049
animation, attack animations, 1055–1058
animation, walking animations, 1054–1055
camera movement, 1091–1094
collisions, 1069
giving damage, 1130–1135
Grappler attacks, 1174–1180
GUI connections, 1123–1125
health, 1121–1122
IGadget interface, 1150–1154
movement systems, 1049–1053, 1059–1061, 1087–1091
naming conventions, 1043–1044
picking up items, 1135–1137, 1181–1184
taking damage, 1127–1130
weapons, 1059–1061
dropping items
keys, 1138–1140
randomized items, 1140–1143
dungeon design, 1143–1147
enemies
dropping items, keys, 1138–1140
dropping items, randomized items, 1140–1143
giving damage, 1127–1130
Skeletons, 1072–1075, 1109, 1145–1147
taking damage, 1130–1135
Game pane, 1024
Grappler, 1147–1148
building, 1154–1159
collisions, 1169–1173
firing, 1169
picking up items, 1181–1184
pulling Dray (hero) in, 1174–1180
secondary abilities, 1159–1169
testing, 1180–1181
grid alignment, 1078–1079
GridMove scripts, 1085–1087
IFacingMover interface, 1079–1084
IGadget interface, 1148–1154
ISwappable interface, 1107–1111, 1145–1147
keys, 1111–1121, 1138–1140
maps/guidance systems, 1031–1042
picking up items, 1135–1137, 1181–1184
prefabs, 1109
Project pane, 1026
randomized items, 1140–1143
Resources folder files, 1026
room to room movement, 1087–1091
setup, 1022–1023, 1097–1098
sprites
CollisionTiles sprites, 1061–1064
naming conventions, 1043–1044
storing two-dimensional data in linear
arrays, 1039–1040
Tilemaps, 1031–1042
programmatic collisions, 1061
programmatically filling collisions,
1065–1069
TileSwaps, 1099, 1101–1107, 1144–1147
Dungeons & Dragons, 27, 59
cultural narratives, 93–94
cumulative outcomes, 76
dynamic narratives, 85
emergent narratives, 87
gameplay as art, 93
outcomes, 26
progression tables, 48
duplicating weapon data, 225–226
dynamic aesthetics, 77
environmental aesthetics, 82
audio play environments, 83–84
player considerations, 84–85
visual play environments, 82–83
procedural aesthetics, 77
procedural music, 78–79
procedural visual arts, 80–82
dynamic elements, 25, 28–29
emergence, 27
emergent narratives, 27
playtesting, 28
Dynamic headers, 555
dynamic layer, Layered Tetrad, 35–36, 67
aesthetics, 34
dynamic mechanics, 70
discernable actions, 71
dynamic aesthetics, 77–85
integrated actions, 71
meaningful play, 71
outcomes, 76–77
procedures, 70
strategies, 71–73
dynamic narratives, 85–87
dynamic technology, 88
emergence, 69
emergent narratives, 87
mechanics, 34
narratives, 34
players, roles of, 68–69
technology, 34
dynamic mechanics, 70
discernable actions, 71
dynamic aesthetics, 77
environmental aesthetics, 82–85
procedural aesthetics, 77–82
house rules, 73–74
integrated actions, 71
meaningful play, 71
outcomes, 76–77
player intent, 74–75
procedures, 70
strategies, 71
designing for, 73
optimal strategies, 72
dynamic narratives, 85–86
dynamic systems, 12
dynamic technology, 88

E

ECS (Entity Component Systems), 612–616
editing cell contents in spreadsheets, 198
Editor tool, Unity, 380
Edmund G. Brown, Jr., Governor of California,
et al., Petitioners v. Entertainment Merchants
education/programs, game development,
296–299, 1200–1201
effects, sound, 50–52
efficiency when testing, 974–975
Elder Scrolls game mods, The, 92
Electronic Arts, Majestic, 17, 46–48
Elemental Tetrad, 20, 30
aesthetics, 28
mechanics, 31
story, 29
technology, 28
elements, formal, 12
Elite Beat Agents, VRO, 78–79
embedded experiences, 48
emergence, 69
FDD elements framework, 27–27
narratives, 27–27, 87
unexpected mechanical emergence, 69–70
emotion
inscribed dramatics, 64
modeling NPC behavior, 240–241
empathetic characters versus avatars, 61–62
empowerment, player-centric goals, 142
autotelic empowerment, 143
performative empowerment, 144
empty elements within arrays, 453
endogenous decisions, 10

Dungeon Delver
dropping items, keys, 1138–1140
dropping items, randomized items, 1140–1143
giving damage, 1127–1130
Skeletos, 1072–1075, 1109, 1145–1147
taking damage, 1130–1135

Enemy Class Examples project
Enemy class on GameObjects, 534
setup, 524

Space SHMUP
art assets, 771–773
damage, 792–797
deleting, 777–787
dropping items, randomized items, 1140–1143
dropping items, keys, 1138–1140
giving damage, 1127–1130
Skeletos, 1072–1075, 1109, 1145–1147
taking damage, 1130–1135

Entertainment Software Association (ESA), 288–289
Entrepreneur Technology Center (ETC), Carnegie Mellon University, 297–298
Entity Component Systems (ECS), 612–616
enums (enumeration), 742
PowerUps, 872–876
private BoundsCheck bndCheck, 816–819
programming, 811–832
randomly spawning, 787–790
scripts, 773–787
showing damage, 853–857

Environment Software Association (ESA), 288–289
Entertainment Technology Center (ETC), Carnegie Mellon University, 297–298
Entity Component Systems (ECS), 612–616
enums (enumeration), 742
PowerUps, 872–876
private BoundsCheck bndCheck, 816–819
programming, 811–832
randomly spawning, 787–790
scripts, 773–787
showing damage, 853–857

Epic Games
Fortnite, 294
Unreal, 92

Space SHMUP
art assets, 771–773
damage, 792–797
deleting, 777–787
dropping items, randomized items, 1140–1143
dropping items, keys, 1138–1140
giving damage, 1127–1130
Skeletos, 1072–1075, 1109, 1145–1147
taking damage, 1130–1135

Environment Software Association (ESA), 288–289
Entrepreneur Technology Center (ETC), Carnegie Mellon University, 297–298
Entity Component Systems (ECS), 612–616
enums (enumeration), 742
PowerUps, 872–876
private BoundsCheck bndCheck, 816–819
programming, 811–832
randomly spawning, 787–790
scripts, 773–787
showing damage, 853–857

Environment Software Association (ESA), 288–289
Entrepreneur Technology Center (ETC), Carnegie Mellon University, 297–298
Entity Component Systems (ECS), 612–616
enums (enumeration), 742
PowerUps, 872–876
private BoundsCheck bndCheck, 816–819
programming, 811–832
randomly spawning, 787–790
scripts, 773–787
showing damage, 853–857

Epic Games
Fortnite, 294
Unreal, 92

equal sign (=)
equals sign (=)
= (Assignment operator), 411, 418
== (Is Equal To operator), 411–413

Ernst, James
Cheap Ass Games, 50
touch aesthetics, 50
errors, debugging
runtime errors, 502–504
stepping through errors, 506–507
Unity errors, 505

Estimation hours, Main worksheets (BDC), 274–276, 277–280
ETC (Entrepreneur Technology Center), Carnegie Mellon University, 297–298

Evil Hat Productions, FATE Accelerated system, 59
eWeaponType enum, Space SHMUP, 833–834
Excel (Microsoft), 189–190
evaluation order, scripts, 1040
executive attention, 145
exclamation points (!)
! (NOT operator), 406
!= (Not Equal To operator), 414
exiting loops, 433–435

expansion phase, brainstorming/ideation, 114–115
experience (shared), developing player relationships, 86–87
experiential understanding, 12, 149–150
explicit procedures, 70
explicit written rules, 46
Experimenter's (Bartle's Spades), 74
exponents, 386–387
exposition
five-act dramatic structures, 56
three-act dramatic structures, 55

Fallout
plots versus free will, 57
face art, adding to cards, 936–937
fair play, 312
fairness. See game balance
falling action
five-act dramatic structures, 54
three-act dramatic structures, 56
Fallout, narrative game mods, 94
Fallout 3
custom game levels, 92
  game mods, 92
Fallout 4, plots versus free will, 58
fame, designer-centric goals, 131–132
fan art, 93
fan fiction, 95
Fantastic Contraption, 251
Farmville, 10
assets, 47
  resources, 47
  spoilage mechanics, 47, 135–137
Farscape, foreshadowing, 58
FATE Accelerated system, 59
FDD (Formal, Dramatic, Dynamic) elements, 20, 25–27
  dramatic elements, 25, 26–28
    characters, 26
    premises, 28
    stories, 26
  dynamic elements, 25, 28–29
    emergence, 27
    emergent narratives, 27
    playtesting, 28
  formal elements, 27–28
feedback
  game balance, 228–229
  interpreting, iterative design, 112
  player scores, Prospector, 1007–1013
"Feminist Critics of Video Games Facing Threats in 'GamerGate' Campaign," 99–101
fiction, fan, 95
Field, Syd
  three-act dramatic structures, 56–58
fields
  classes, 523, 527–530
  names in Inspector, 637–641
  overriding values in Inspector, 641
  private fields, viewing in classes, 734
Final Fantasy III, minor NPC development, 58
  Final Fantasy VI, minor NPC development, 58
Final Fantasy VII
  emphatic characters versus avatars, 62
  final outcomes, 76
  novel decisions, 148
  plots versus free will, 58
Final Fantasy X, plots versus free will, 58
  final outcomes, 76–77
  first plot points, three-act dramatic structures, 55
  five-act dramatic structures, 54–55
  five aesthetic senses, 51
  fixed updates, 556–557
  float variables, 387
  FloatingScore GameObject, Prospector, 991–999
"Flocks, Herds, and Schools: A Distributed Behavioral Model," 542
  flow
    player-centric goals, 138–141
    spaces, 47
Flow: The Psychology of Optimal Experience, 139–140
flowcharts, Apple Picker, 319–321
Flow, PCO, 79
flukes, playtesting, 7
focus testing, 152–153, 183
folder names, changing in Unity, 362
follow cameras, Mission Demolition, 702–710
for loops, 424, 429–431, 432
  autoformatting, 656
  jagged arrays, 464–465
  force quitting applications, 426, 509
  foreach loops, 424, 433, 454–455
  foreshadowing, linear narratives, 58
  formal elements, 12, 27
  boundaries, 26
  objectives, 25
  outcomes, 26
  player interaction patterns, 28
  procedures, 25
  resources, 25
  rules, 25
  formal group playtesting, 175–176
  formal individual playtesting, 176–181
Fortnite, 294
fortune, designer-centric goals, 130
  community, 132
  fame, 131–132
  greater good, 133
  personal expression/communication, 132–133
Forza: Horizon, traversal puzzles, 261
frames, defined, 319–320
frameworks, game design
  Elemental Tetrad, 20, 30
    aesthetics, 28
    mechanics, 31
    story, 29
    technology, 28
  FDD elements, 20, 25–27
    dramatic elements, 25, 27–28
    dynamic elements, 25, 28–29
    formal elements, 27–28
MDA, 22
  defined, 23
designer views, 23
player views, 23
Snakes and Ladders, 22–24
free will versus plots, 59
freemium games, 295
Freeq, background noise, 53
Frequency, VRO, 78–79
Freytag, Gustav
  five-act dramatic structures, 54–55
  Technik des Drama (The Technique of Drama), Die, 54–55
friends (trusted), circles of playtesters, 170
Fullerton, Tracy
  dramatic elements, 26–27
dynamic mechanics, 70
formal elements, 28
Game Design Workshop, 9, 18, 70, 118–119
games, defined, 12–13
ideation phase, game development, 118–119
inscribed mechanics, 40
player interaction patterns, 43–46
fun, player-centric goals, 134–135
functions
  arguments, 478–479
calling, 476–477, 1154
defined, 474–476
defining order, 480
evercapulating actions, 476
Function Examples project, setup, 474
Math functions, 396
naming, 482
overloading, 485–486
parameters, 478–479
  optional parameters, 486–487
  params keyword, 487–489
as properties, 483–484
reasons for using, 482–483, 484–485
recursive functions, 489–491
returning
  values (results), 480
  void, 481–482
scope, 476
start() function versus update() function, 370–398
static functions, 391–392
Update() function versus Start() function, 370–398
G
  game builds. See also game prototype tutorials; projects
    2D adventure game level
      combat, 162
      playtesting, 161–162
      prototyping, 157–159
      shortcuts, 161
    traversal mechanics, 159–161
  Apple Picker, 316
    basic gameplay, 317
    GameObjects, 317–318
    GameObjects, action lists, 318
    GameObjects, flowcharts, 319–321
  classic games, building as a learning example, 1199
  frames, 319–320
  for lifelong enrichment, 1200
  small game projects, 1199
  uroboros
    collection phase, brainstorming/ideation, 115
    collision phase, brainstorming/ideation, 116–117
    discussion phase, brainstorming/ideation, 117
    expansion phase, brainstorming/ideation, 114–115
    idea cards, 115
    idea collisions, 116–117
    rating phase, brainstorming/ideation, 117
  game companies, working conditions, 292–293
  game conferences, 301
  Game Design Theory, 9–10, 93
    fun, elements of, 134–135
    performative empowerment, 144
  Game Design Workshop, 9, 18, 40, 70
    dramatic elements, 25, 27–28
    dynamic elements, 25, 28–29
    formal elements, 27–28
    ideation phase, game development, 118–119
    player interaction patterns, 43–46
  game development, 118, 120
    industry See digital games industry
  Game Feel, 623–624
  "Game Game, The," 15–16
  game masters, 45
  Apple Picker, 636–637
  Dungeon Delver, 1024
  Prospector, 906–907
game prototype tutorials. See also game builds; projects

Apple Picker, 621–623
  art assets, 624–633
  boids, 551
  cameras, setup, 633–634
  catching apples, 658–659
  coding, 637–641
  destroying baskets, 670–672
directionality, 646–648
DOTS, 582
  dropping apples, 649–651
game management, 661–662
game panel settings, 636–637
GUI, 661–662
Hello World, 359
HighScore texts, 662–664, 672–678
  instance overrides, applying to prefabs, 641–642
  instantiating baskets, 655–656
  missed apple notifications, 668–672
  movement systems, 643–646
  moving baskets with mouse, 657–658
  physics layers, 651–652
  points accumulation, 665–668
  purpose of, 623
ScoreCounter texts, 662–664
  setup, 624
  stopping apples from falling too far, 653–655
tuning script variables, 659–660
Dungeon Delver, 1019–1021, 1095–1096
  anti-aliasing issues, 1041–1042
  cameras, 1023
cameras, GUI cameras, 1024–1025
cameras, main camera, 1024–1025
  component-based design, 1021–1022
damage, 1125–1135
DeliverTiles, 1026–1028
DelverLevel_Eagle Text files, 1028–1031, 1033–1035
doors, keys, 1111–1121, 1138–1140
Dray (hero), 1042
  Dray (hero), animation, 1044–1049
  Dray (hero), attack animations, 1055–1058
  Dray (hero), camera movement, 1059–1061, 1091–1094
  Dray (hero), collisions, 1069
  Dray (hero), giving damage, 1130–1135
  Dray (hero), Grappler attacks, 1174–1180
  Dray (hero), GUI connections, 1123–1125
  Dray (hero), health, 1121–1122
  Dray (hero), IGadget interface, 1150–1154
  Dray (hero), movement systems, 1049–1053, 1087–1091
  Dray (hero), naming conventions, 1043–1044
Dray (hero), picking up items, 1135–1137, 1181–1184
Dray (hero), taking damage, 1127–1130
Dray (hero), walking animations, 1054–1055
Dray (hero), weapons, 1059–1061
dropping items, keys, 1138–1140
dropping items, randomized items, 1140–1143
dungeon design, 1143–1147
  enemies, dropping keys, 1138–1140
  enemies, dropping randomized items, 1140–1143
  enemies, giving damage, 1127–1130
  enemies, Skeletos, 1072–1075, 1109, 1145–1147
  enemies, taking damage, 1130–1135
Game pane, 1024
Grappler, 1147–1148
Grappler, building, 1154–1159
Grappler, collisions, 1169–1173
Grappler, firing, 1169
Grappler, picking up items, 1181–1184
Grappler, pulling Dray (hero) in, 1174–1180
Grappler, secondary abilities, 1159–1169
Grappler, testing, 1180–1181
grid alignment, 1078–1079
GridMove scripts, 1085–1087
IFacingMover interface, 1079–1084
IGadget interface, 1148–1154
ISwappable interface, 1107–1111, 1145–1147
  keys, 1111–1121, 1138–1140
maps/guidance systems, 1031–1042
  picking up items, 1135–1137, 1181–1184
prefabs, 1109
Project pane, 1026
randomized items, 1140–1143
Resources folder files, 1026
  room to room movement, 1087–1091
setup, 1022–1023, 1097–1098
  sprites, CollisionTiles sprites, 1061–1064
  sprites, naming conventions, 1043–1044
  storing two-dimensional data in linear arrays, 1039–1040
Tilemaps, 1031–1042
tilemaps, programmatically filling collisions, 1065–1069
game prototype tutorials. See also game builds; projects (continued)

Mission Demolition, 681–683
art assets, 684–690
cameras, follow cameras, 702–710
cameras, settings, 685–687
castles, 717–725, 734–737
coding, 691
coding, castles, 717–725, 734–737
coding, collision detection, 698–699
coding, creating slingshot class, 691–702
coding, follow cameras, 702–710
coding, goals, 734–736
coding, instantiating projectiles, 694–698
coding, multiple views, 745–751
coding, organizing Project pane (Unity), 716–717
coding, projectiles, 725–734
coding, showing when slingshot is active, 692–693
coding, UI, 737–738
coding, section/speed, 710–716
directional light, 685
game management, 739–744
goals, 734–736
ground, 684–685
multiple views, 745–751
projectiles, 690
projectiles, instantiating, 694–698
projectiles, ProjectileLine Trails, 728–734
projectiles, Rigidbody insomnia, 725–728
prototype concept, 683–684
setup, 662–683
slingshots, 687–690
slingshots, creating slingshot class, 691–702
slingshots, showing when active, 692–693
UI, 737–738
Prospector, 898–899, 969–970
adding backs to cards, 937–938
adding face art to cards, 936–937
adding game elements, 972
background images, 983–985
BézierMover class, 987–991
build settings, 903
building cards, 922–938
cameras, 906–907
classes, 948–961
clickable cards, 962–964
example of play, 900–901
feedback on player scores, 1007–1013
FloatingScore GameObject, 991–999
game logic, 961–962
Game pane, 906–907
gold cards, 1017
GUI, 985–986
initial layout, 899–900
JSON through code, 913–917
managing rounds, 972–975
matching cards in mine, 964–968
Mine Tableau layout, 940–948
mobile devices, 1018
moving cards, 1017–1018
pauses between rounds, 1006–1007
pips, adding to cards, 934–935
Prospector_Scene_0, 905
rules, 900
ScoreBoard class, 1000–1001
ScoreBoard GameObject, 999
scoring, 975–983, 985–986, 999–1006, 1007–1013
setup, 901–902, 906–907, 971
shuffling cards, 939–940
silver cards, 1017
sorting cards, 954–958
sprites, building cards from sprites, 931–934
sprites, constructing cards from sprites, 911–912
sprites, gathering references to the deck, 918–920
sprites, importing images as, 907–909
sprites, prefab GameObjects as sprites/cards, 921–922
sprites, slicing rank images as sprites, 909–911
Unity window layout, 906
updating ScoreManager script, 1001–1006
WebGL module, 1013–1016
WebGL module, installing, 903–904
WebGL module, switching to, 904–905
Space SHMUP, 753–754, 807–808
adding elements, 894
building game levels, 894–895
delegate events, 842–844
enemies, art assets, 771–773
enemies, damage, 792–797
enemies, deleting, 777–787
enemies, destroying, 804
enemies, Enemy_0, 810–811
enemies, Enemy_1, 812–819
enemies, Enemy_2, 819–826
enemies, Enemy_3, 826–832
enemies, Enemy_4, 876–888
enemies, OnCollisionEnter method, 851–852
enemies, PowerUps, 872–876
enemies, private BoundsCheck bndCheck, 816–819
enemies, programming, 811–832
enemies, randomly spawning, 787–790
enemies, scripts, 773–787
enemies, showing damage, 853–857
expanding weapon options, 865–866
game structure, 895
GUI (Graphical User Interfaces), 895
hero ship, creating, 758–760
hero ship, Hero update() method, 760–764
hero ship, keeping on screen, 767–771
hero ship, shields, 764–766
importing asset packages, 755–757
layers, 790–792
physics, 790–792
PowerUps, 857–869, 872–876
projectiles, adding shooting capability, 800–801
projectiles, destroying enemies, 804
projectiles, hero’s bullet, 800–801
projectiles, scripts, 803
projectiles, shooting, 800
projectiles, weapon GameObjects, 844–851
race conditions, 869–872
restarting games, 797–799
scene setup, 757–758
setting up, 809
setup, 755, 757–758
shooting, 800, 833
shooting, adding shooting capability, 802–803
shooting, delegate events, 842–844
shooting, eWeaponType enum, 833–834
shooting, hero’s bullet, 800–801
shooting, showing damage, 853–857
shooting, WeaponDefinition class, 834–842
starfield backgrounds, 890–893
tags, 790–792
tuning settings, 888–890
tuning variables, 893
GameObjects, 398
Apple Picker, 317–318
action lists, 318
flowcharts, 319–321
Attractor GameObject, Boids project, 549–551
Box Collider component, 371
Collider component, 400–401
Dungeon Delver, keeping GameObjects in the room, 1075–1078
Enemy class on GameObjects, 534
FloatingScore GameObject, Prospector, 991–999
flowcharts, Apple Picker, 319–321
manipulating, 370–373
Mesh Filter component, 371
Mesh Renderer component, 372
MeshFilter component, 400
MonoBehavior subclasses as GameObject components, 530–533
prefabs, 373–378
Prospector, prefabs for sprites/cards, 921–922
Renderer component, 400
RigidBody component, 372, 402
ScoreBoard GameObject, Prospector, 999
Transform component, 372
weapon GameObjects, Space SHMUP, 844–851
gameplay as art, 93
GamerGate, 100–101
games
as art, 97–98
asymmetric games, 188
board games, systems thinking, 312
builds. See game builds
classic games, building as a learning example, 1199
closed games, 11
cultural impact of games, 97–98
custom levels, 92
defined, 9–10
Burgun, Keith, 14
caring about definitions, 14–16
Fullerton, Tracy, 12–13
human desire, 15
IndieCade, 16–17
Meier, Sid, 12
Midgley, Mary, 15–16
nebulous nature of, 16
Pearce, Celia, 16–17
Roberts, Sam, 16–17
Schell, Jesse, 13
Suits, Bernard, 10–12, 14
Wittgenstein, Ludwig, 14
development, 118, 120
AAA development, costs, 293
Agile Software Development, 266–267
alpha phase, 119
beta phase, 119
education/programs, 1200–1201
gold phase, 119
ideation phase, 118–119
indie gaming scene, 295
post-release phase, 119
preproduction phase, 118–119
production phase, 118–119
Unity, 315
education/programs, 296–299
flow, player-centric goals, 138–141
freemium games, 295
logic, Prospector, 961–962
loops, 424
managing
Apple Picker, 661–662
Mission Demolition, 739–744
manipulative game design, 98–99
modifying, 91–92, 94
narrative game mods, 94
open games, 11
as a series of interesting choices, 868
as a service, 295
social media games, 136–137
sporadic-play games, 135–136, 137
symmetric games, 188
technology used outside games, 95
time-base games, 644–645
"Gang of Four," 552
Garfield, Richard
innovation, 113
RoboRally, 113
genders/ages, digital games industry, 291–292
generic collections, 438, 442–443
generic methods (< >), 398
goals
aesthetics, 51–53
design, 130
designer-centric goals, 130, 131–134
player-centric goals, 130, 134–150
indirect player guidance, 235
Mission Demolition, 734–736
Passage, 11
puzzle design, 256–257
God of War
direct player guidance, 234
game flow, 140–141
sequencing, 244
gold cards, Prospector, 1017
gold phase, game development, 119
Google Sheets, 189
balancing weapons, 219–220
calculating average damage, 222
duplicating weapon data, 225–226
example of, 227–228
percent chance for each shot, 220–221
rebalancing weapons, 226–227
showing overall damage, 224–225
charts, 204–206
clarity in, 199
color scale conditional formatting, 200
columns
adding, 194–195
setting widths, 195
conditional formatting, 203
dice probability, 191
going started, 191–192
labels, 199
naming documents, 194
rows
creating, 194
filling with data, 196
iterating Die A rows, 197
making Die A rows, 196–197
making Die B rows, 197–198
saving, 199
summing results
counting all die rolls, 202–203
counting sums of die rolls, 201–202
two dice, 201
weighted probabilities, 216–217
Grand Theft Auto V, direct player guidance, 233–234
Grandia III, novel decisions, 148
Grapple, Dungeon Delver, 1147–1148
building, 1154–1159
collisions, 1169–1173
firing, 1169
picking up items, 1181–1184
pulling Dray (hero) in, 1174–1180
secondary abilities, 1159–1169
testing, 1180–1181
Grasshopper, The, 9, 10, 14, 137, 144–145
greater good, designer-centric goals, 133
greater than symbol (>)
Greater Than operator (>, 414
Greater Than or Equal To operator (>=), 415
GridMove scripts, Dungeon Delver, 1085–1087
 grids
Dungeon Delver, player alignment, 1078–1079
hexagonal grids, 155
movement systems, 154–155
square grids, 154
ground, Mission Demolition, 684–685
Groundhog Day, experiential understanding, 149–150
group playtesting, formal, 175–176
growth, digital games industry, 289–290
GUI (Graphical User Interfaces)
Apple Picker, 661–662
cameras, Dungeon Delver, 1024–1025
Dray (hero), Dungeon Delver, 1123–1125
Prospector
feedback on player scores, 1007–1013
pauses between rounds, 1006–1007
scoring, 985–986, 999–1006, 1007–1013
updating ScoreManager script, 1001–1006
prototyping, 156
guidance systems/maps, 232
direct guidance, 232, 233–234
brevity, 233
calls to action, 233–234
clarity, 233
immediacy, 232
instructions, 233–234
maps/guidance systems, 233–234
pop-ups, 234
scarcity, 232
Dungeon Delver, 1031–1042
indirect player guidance, 234
audio design, 240
constraints, 234–240
goals, 235
NPC, 240
physical interfaces, 235–236
player avatars, 240
visual design, 236–239
integrated actions, 245
sequencing, 243–244
Teaching new skills/concepts, 243
Guitar Hero, indirect player guidance, 235–236
Hamlet on the Holodeck, 85–86
Hand tool (Q), Unity, 380
Hawaii, landmarks, 47–48
headers, 555
health, Dungeon Delver, 1121–1122
hearing
background noise, 53
dialogue, 52
five aesthetic senses, 49–53
music, 52
sound effects, 50–52
Heart of Darkness, 62
Hearts (Bartle’s Socializers), 74
"Hello World" project
adding color, 381–382
cube environments, 378–381
debugging
attaching scripts, 500–502
compile-time bugs, 495–500
removing scripts, 500–502
runtime errors, 502–504
stepping through errors, 506–507
deleting cubes, 467–471
folder configuration, 361–362
scripts
comments in scripts, 369
creating, 363–368
disabling, 370
manipulating GameObjects, 370–373
prefabs, 373–378
start() function versus update() function, 370–398
setup, 360
shrink cubes, 467–471
hero ship, Space SHMUP
 creating, 758–760
Hero update() method, 760–764
keeping on screen, 767–771
shields, 764–766
shooting
adding shooting capability, 802–803
hero’s bullet, 800–801
hexadecimal numbers, DelverLevel_Eagle Text
files, 1030–1031
hexagonal grids, movement systems, 155
Hierarchy pane (Unity), 339
HighScore texts, Apple Picker, 662–664,
672–678
holistic design, 125
Homo Ludens, 26, 138
Honolulu, Hawaii, landmarks, 47–48
hooks, three-act dramatic structures, 57
hours, Main worksheets (BDC)
estimates, 274–276, 277–280
totals, 277–280
house rules, 73–74
Hoye, Mike

Legend of Zelda: The Wind Waker, The, 94
HRS (Horizontal Re-Sequencing), 78–79
Huizinga, John
boundaries, 26
Homo Ludens, 26, 138
magic circle, 138
human desire in gameplay, defined, 15

i++ (iteration clauses), 430, 431
i<3 (conditional clauses), 430
ideation phase, game development, 118–119
ideation/brainstorming, 113–114
   collection phase, 115–116
   collision phase, 116–117
   discussion phase, 117
   expansion phase, 114–115
   idea cards, 115
   nodes, 115
   rating phase, 117
if statements, 416
   Assignment operator (=), 418
   with Boolean operations, 417–418
   nesting if statements, 419
IFacingMover interface, Dungeon Delver, 1079–1084
if.else if.else statements, 418–419
if.else statements, 418
IGadget interface, Dungeon Delver, 1148–1154
IGDPD layout, Unity
   downloading, 340
   manually arranging, 341–344
ilinx, 134
images
   creation, OOP, 586–593
   logic puzzles, 254
   media puzzles, 253
Prospector
   background images, 983–985
   importing as sprites, 907–909
   slicing rank images as sprites, 909–911
   word puzzles, 254
IMGD (Interactive Media & Games Division), University of Southern California, 298
immediacy
   direct player guidance, 232
   of objectives, 42–43
outcomes, 77
impact of games, cultural, 97–98
implementation phase, iterative design, 105, 111
implicit procedures, 70
implicit rules, 46
importing Unity asset packages, 755–757
importance of objectives, 42–44
improving as a game designer, 134
inciting incidents, three-act dramatic structures, 55
Includes, classes, 523–524
Incredible Machine, The, 251
increment operators (++,), 428
incremental innovation, 113
indie gaming scene, development of, 295
IndieCade, 17
   games, defined, 16–17
   scoping, 120
indirect guidance strategies, 42, 234
   audio design, 240
   constraints, 234–240
   goals, 235
   NPC, 240
   physical interfaces, 235–236
   player avatars, 240
   visual design, 236–239
individual playtesting, formal, 176–181
   infinite loops, 425–427
Influence: The Psychology of Persuasion, 2
Infocom, Zork, 86
informal individual playtesting, 172–175
information, aesthetic goals, 51
inheritance (class), 354, 533
initial development speeds, prototyping, 152–153
initialization clauses (int i=0;), 430
in medias res, 145
innovation, 112
   incremental innovation, 113
   intersectional innovation, 113
InRoom scripts, Dungeon Delver, 1070–1072
inscribed aesthetics, 51
   five aesthetic senses, 51
   goals, 51–53
inscribed dramatics
   emotion, 64
   justification, 62
   mechanics reinforcement, 62
   motivation, 62
   progression, 62
   purposes for, 62–64
rewards, 62
Inscribed headers, 555
inscribed layer, Layered Tetrad, 34, 40
aesthetics, 33
inscribed aesthetics, 51
inscribed dramatics, 62–64
inscribed mechanics, 35, 40–46, 48–51
inscribed narratives, 33, 53–54
inscribed technology, 33, 63, 65
traditional dramatics, 55
five-act dramatic structures, 54–55
interactive versus linear narratives, 59–61
three-act dramatic structures, 56–58
inscribed mechanics, 35
boundaries, 40, 46–48
defined, 40
Fullerton, Tracy, 40
objectives, 43
conflicting objectives, 44–45
defining player relationships, 44–47
immediacy of objectives, 42–43
importance of objectives, 42–44
spaces, 48
player relationships, 40
defining with objectives, 44–47
player interaction patterns, 43–46
resources, 40, 47–49
rules, 40, 46–48
Schell, Jesse, 43
spaces, 40, 47–48
tables, 41, 50–51
inscribed narratives, 33, 54
characters, 52
plots, 53
premises, 52–54
settings, 52
inscribed technology, 33, 63
digital games, 65
digital interfaces, indirect player guidance, 235–236
Inspector pane (Unity), 339
fields
naming, 637–641
overriding values, 641
headers, 553–555
naming variables, 402
play mode values, setting, 646
script values, changing, 660–661
inspiration, puzzle design, 255
installing
Unity 2020.3 LTS, 326–327
Unity Hub, 324–326
WebGL module, 903–904
instance overrides, applying to prefabs, 641–642
instance variables/functions, 390
instantiating
baskets, Apple Picker, 655–656
projectiles, Mission Demolition, 694–698
instructions
direct player guidance, 233–234
systems thinking, 313
int i=0; (initialization clauses), 430
int variables, 386
integrated actions, 71, 147, 245
intent of players, 74–75
interaction patterns, players, 43–45
cooperative play interaction pattern, 44
multilateral competition interaction pattern, 44
parallel play interaction pattern, 44
player versus player interaction pattern, 46
single player interaction pattern, 44
team competition interaction pattern, 44
unilateral competition interaction pattern, 44
interactive experiences, defined, 18
interactive fiction, 86
Interactive Media & Games Division (IMGD), University of Southern California, 298
interactive narratives
incunabula, 85–86
linear narratives versus, 59
interest curves, 145–146
interest polling, 184
interesting decisions, 12, 147–149, 868
interfaces
Dungeon Delver
IFacingMover interface, 1079–1084
IGadget interface, 1148–1154
ISwappable interface, 1107–1111, 1145–1147
Mission Demolition, 737–738
physical interfaces, indirect player guidance, 235–236
prototyping, 156
Internet, circles of playtesters, 172
interpolation
linear interpolation, 567
of strings using $981–982
intersectional innovation, 113
interviewing, digital games industry, 302–305
investigators versus playtesters, 168
involvement/attention, player-centric goals, 145–147
Is Equal To operator (==), 411–413
ISerializationCallbackReceiver Interface, 942–943
ISwappable interface, Dungeon Delver, 1107–1111, 1145–1147
iteration clauses (i++), 430, 431
iteration speeds, prototyping, 152
iteration variables, Loop Examples project, 428
iterative code development, 793–794
iterative design, 6, 104
    analysis phase, 105–107
    changing your mind, 117–118
    design phase, 104, 107–109
    feedback, interpreting, 112
    implementation phase, 105, 111
    testing phase, 105, 112

J
jagged arrays, 461–464
jagged Lists, 465–466
JetBrains Rider, 328–329
Johansson, Frans
    innovation, 112–113
joining game design projects, 305
Journey
    indirect player guidance, 236
    NPC emotional connections, 240–242
    tissue playtesters, 171
JSON through code, Prospector, 913–917
jump statements, in loops, 433
justification, inscribed dramatics, 62

K
Kaboom!, 316
keys, Dungeon Delver, 1111–1121, 1138–1140
Killers (Bartle’s Club’s), 74
Kim, Scott
    "Art of Puzzle Design," The, 248–250
    puzzle design, 248–250, 255–256
Kirby, integrated actions, 71
Kya: Dark Lineage
    direct player guidance, 232
    modeling NPC behavior, 240–241
    sequencing, 243–245

L
L.A. Confidential, image/media puzzles, 253
labels, Google Sheets, 199
labs, formal individual playtesting, 178–179
landmarks
    Honolulu, Hawaii, 47–48
    spaces, 47–48
    visual design, indirect player guidance, 236–237
languages, computer, 313–314
Layered Tetrad, 32–34
cultural layer, 36–37, 90–91
    aesthetics, 36
    authorized transmedia and, 96–97
    cultural aesthetics, 93
    cultural impact of games, 97–98
    cultural mechanics, 91–92
    cultural narratives, 93–95
    cultural technology, 95–96
    manipulative game design, 98–99
    mechanics, 35–38
    messages games/fans send, 101
    narratives, 36
    technology, 36
designer responsibilities, 39–40
dynamic layer, 35–36, 67
    aesthetics, 34
    dynamic aesthetics, 77–85
    dynamic mechanics, 70–77
    dynamic narratives, 85–87
    dynamic technology, 88
    emergence, 69
    mechanics, 34
    narratives, 34
    roles of players, 68–69
    technology, 34
inscribed layer, 34
    aesthetics, 33
    boundaries, 46–48
    inscribed aesthetics, 51–54
    inscribed dramatics, 62–64
    inscribed mechanics, objectives, 42–43
    inscribed mechanics, overview, 40–43
    inscribed narratives, 53–54
    inscribed technology, 63–65
    interactive narratives, 59–61
    linear narratives, 59–61
    mechanics, 35
    narratives, 33
    objectives, 42–43
    player relationships, 45–47
    resources, 47–49
    rules, 46–48
    spaces, 47–48
    tables, 50–51
technology, 33
traditional dramatics, 55–58
layers, Space SHMUP, 790–792
layouts
Mine Tableau layout, Prospector, 940–948
Prospector, 899–900
Snakes and Ladders, 22–25
Unity layouts, navigating, 338–339
Unity window, Prospector, 906
learning, programming languages, 331–334
Legend of Zelda: Ocarina of Time, direct player guidance, 233
Legend of Zelda: The Wind Waker
justification, 62
motivation, 62
narrative game mods, 94
Legend of Zelda: Twilight Princess, touch aesthetics, 50
Legend of Zelda. See also Dungeon Delver
attributes, 49
boss fights, 262
resources, 49
silent protagonists, 60
traversal mechanics, 159
LEG0 bricks, prototyping, 155
Lemarchand, Richard
"Attention, Not Immersion: Making Your Games Better with Psychology and Playtesting, the Uncharted Way,
144–145
attention/involvement, player-centric goals, 145
Playful Production Process: For Game Designers (and Everyone), A, 121–122, 126
scope management with preproduction deliverables, 121–122
Les Jeux et Les Hommes, 134–135
less than symbol (<)
Less Than operator (<), 414
Less Than or Equal To operator (<=), 415
levels
custom levels, 92
puzzle design, 255
libraries, code, 314–315
LibreOffice Calc, 190
lifelong enrichment, building games for, 1200
light
directional light, Mission Demolition, 685
visual design, indirect player guidance, 236
Light Editor Theme, Unity, 338
liking/not liking ideas, playtesting, 169
linear arrays, storing two-dimensional data in, 1039–1040
linear interpolation, 567
linear narratives
foreshadowing, 58
interactive narratives versus, 59
minor NPC development, 58–59
side quests, 58
Lionshead Studios, Fable, 57
listening, iterative design, 108–109
Lists, 439–440, 443–446
choosing, 466–467
converting arrays to, 457
converting to arrays, 447
jagged Lists, 465–466
methods, 446–447
properties, 446
zero-indexed lists, 440
Little Big Planet, custom game levels, 92
locality (DOD), data, 577–578
logic, Prospector, 961–962
logic puzzles, 253
logic/image puzzles, 254
logic/word puzzles, 254
logical equivalence, Boolean operations, 410
logs, ADL, 182–183
long-term objectives, 42
LookAtAttractor script, Boids project, 551, 557–558
loops, 423
break statements, 433–435
condition clauses (i<3), 430
do.while loops, 424, 429
exiting, 433–435
for loops, 424, 429–431, 432
autoformatting, 656
jagged arrays, 464–465
foreach loops, 424, 433, 454–455
game loops, 424
infinite loops, 425–427
initialization clauses (int i=0;), 430
iteration clauses (i++), 430, 431
jump statements, 433
Loop Examples project, 424–426
modulo operators (%), 436
skipping single iterations, 435
types of (overview), 424
while loops, 424, 425, 426–428
Lord of the Rings, 59
low technical barriers to entry, prototyping, 152
LucasArts, X-Wing
aesthetic goals, 53–54
procedural music, 78–79
ludology
  defined, 19–21
  frameworks, 22
Ludwig, Manfred
  Up the River, 72
lusory attitude, 13, 136–137

M
machinima, 93, 94–95
macOS
  debugging, 507–508, 510–511
  force quitting applications, 509
  "right-clicking" on mouse, 361
Macro Charts, 126
Macro Documents, 124–126
magic circle, 138
Magic: The Gathering, 113
Main worksheets (BDC), 273
  estimating hours, 274–276, 277–280
  sprint progress, 277–279
  sprint settings, 273–274
  task assignments, 274–275, 280
  totalling hours, 277–280
Mainichi, experiential understanding, 149–150
Majestic, 17, 46–48
managing
  games
    Apple Picker, 661–662
    Mission Demolition, 739–744
  rounds, Prospector, 972–975
  scope with preproduction deliverables, 121–122
    Macro Charts, 126
    Macro Documents, 124–126
    Vertical Slices, 118–119, 122–123
"Mangle of Play," The, 90–91
manipulative game design, 98–99
manually arranging IGDPD layout, Unity, 341–344
maps/guidance systems
  direct player guidance, 233–234
  Dungeon Delver, 1031–1042
Mario Kart, game balance, 228
markers (whiteboard), brainstorming/ideation
  (expansion phase), 115
Mass Effect
  multiple dialogue choices, 61–64
  novel decisions, 148–149
  player interaction patterns, 46
  matching cards in mine, Prospector, 964–968
  math of probability, 207–211
mathf functions, 396
Max (Advantage) controllers, 36
MDA (Mechanics, Dynamics, Aesthetics), 22
  aesthetics, 33
  defined, 23
  designer views, 23
  player views, 23
  Snakes and Ladders, 22–25
  layouts, 22–25
  modifying for strategic game play, 24–26
meaningful play, 71, 147
mechanics
  cultural layer, Layered Tetrad, 35–38
  cultural mechanics
    custom game levels, 92
    game mods, 91–92, 92
  dynamic layer, Layered Tetrad, 34
  dynamic mechanics, 70
    discernable actions, 71
    dynamic aesthetics, 77–85
    house rules, 73–74
    integrated actions, 71
    meaningful play, 71
    outcomes, 76–77
    player intent, 74–75
    procedures, 70
    strategies, 71–73
  Elemental Tetrad framework, 31
  inscribed layer, Layered Tetrad, 35
  reinforcement, inscribed dramatics, 62
  traversal mechanics, prototyping, 159–161
Media and Information Department, Michigan
State University, 298
Media Molecule, Little Big Planet, 92
media/image puzzles, 253
Medici Effect, The, 112–113
Meier, Sid, 9
C.P.U. Bach, 79
games, defined, 12
  games as a series of interesting choices, 868
  interesting decisions, 147–149
Mesh Filter component, GameObjects, 371
Mesh Renderer component, GameObjects, 372
MeshFilter component, GameObjects, 400
messages games/fans send, 101
Metal Gear Solid 4, 10
methods
  awake() method versus start() method, 814–815
  classes, 523, 527–530
  Dictionaries, 450–451
  Lists, 446–447
OnCollisionEnter method, *Space SHMUP*, 851–852
static methods, arrays, 456–457
*Metroid Dread*, direct player guidance, 232–233
Michigan State University, Media and Information Department, 298
Microsoft Excel, 189–190
*Middle-earth: Shadow of Mordor*, minor NPC development, 59
Midgley, Mary
"Game Game, The," 15–16
games, defined, 15–16
mid-term objectives, 42
migraines, 85
mimicry, 134
Mine Tableau layout, *Prospector*, 940–948
*Mcne*raft, 99–101
autotelic empowerment, 143
implementation phase, iterative design, 110
indirect player guidance, 235
player-made external tools, 96
procedural environments, 82
minor NPC development, 58–59
missed apple notifications, *Apple Picker*, 668–672
*Mission Demolition*, 681–683
art assets, 684–690
cameras
follow cameras, 702–710
settings, 685–687
castles, 717–725, 734–737
coding, 691
castles, 717–725, 734–737
collision detection, 698–699
creating slingshot class, 691–702
follow cameras, 702–710
goals, 734–736
instantiating projectiles, 694–698
multiple views, 745–751
organizing Project pane (Unity), 716–717
projectiles, *ProjectileLine Trails*, 728–734
projectiles, RigidBody insomnia, 725–728
showing when slingshot is active, 692–693
UI, 737–738
vection/speed, 710–716
directional light, 685
game management, 739–744
goals, 734–736
ground, 684–685
multiple views, 745–751
projectiles, 690
instantiating, 694–698
*ProjectileLine Trails*, 728–734
RigidBody insomnia, 725–728
prototype concept, 683–684
setup, 662–683
slingshots, 687–690
creating slingshot class, 691–702
showing when active, 692–693
UI, 737–738
misspellings, debugging, 494
mixed-mode puzzles, 254
MMORPG (Massively Multiplayer Online Roleplaying Game)
Damage Per Second (DPS) calculators, 95
parallel play interaction pattern, 44
mobile devices, *Prospector*, 1018
modeling NPC behavior
emotional connections, 240–241
negative behaviors, 241
positive behaviors, 241
safety, 241
modifying
games, 91–92, 94
rules, 6–7
modulo operators (%), 436
Moksha Patamu. See *Snakes and Ladders*
MonoBehavior subclasses, as GameObject components, 530–533
MonoDevelop, 329
Monolith Productions, *Middle-earth: Shadow of Mordor*, 59
monolithic programming (OOP), flock of birds simulation, 540–542
Monopoly
assets, 49
conflicting objectives, 44–45
game balance, 228–229
game mods, 92
house rules, 73–74
immediate outcomes, 77
resources, 49
rules, 45
Monte Carlo method, 211
mood, aesthetic goals, 53
Moore, Gordon
"Moore's Law," DOD, 576
motivation, inscribed dramatics, 62
mouse
moving baskets with mouse, *Apple Picker*, 657–658
"right-clicking," macOS, 361
movement systems
Apple Picker, 643–646, 657–658
Dungeon Delver, 1049–1053, 1087–1094
protopotyping, 154–155
Mr. X (Scotland Yard), unilateral competition interaction pattern, 44
multidimensional arrays, 457–461
multilateral competition interaction pattern, 44
multiple dialogue choices, 61
multiple views, Mission Demolition, 745–751
Murray, Janet
Hamlet on the Holodeck, 85–86
music, 52, 78
HRS, 78–79
PCO, 79
VRO, 78–79

N
Nakamura, Jeanne
"Concept of Flow," The, 139
game flow, 139
naming
documents, Google Sheets, 194
fields in Inspector, 637–641
folders in Unity, 362
functions, 482
sprites, Dungeon Delver, 1043–1044
Unity conventions, 389–390
variables in Unity, 376
narratives
cultural layer, Layered Tetrad, 36
cultural narratives, 93–94
fan fiction, 95
machinema, 94–95
narrative game mods, 94
dynamic layer, Layered Tetrad, 34
dynamic narratives, 85–87
emergent narratives, 27, 87
game mods, 94
inscribed layer, Layered Tetrad, 33
inscribed narratives, 54
characters, 52
plots, 53
premises, 52–54
settings, 52
interactive versus linear narratives, 59
linear narratives
foreshadowing, 58
interactive narratives versus, 59
minor NPC development, 58–59
side quests, 58
Naughty Dog, Uncharted 3: Drake’s Deception
machinema, 94–95
particle systems, 80
visual design, indirect player guidance, 237–239
navigating Unity layouts, 338–339
negative behaviors, NPC, 241
Neighborhood script, Boids project, 551, 567–570
NES (Nintendo Entertainment System), Advantage (Max) controllers, 36
nesting if statements, 419
Neverwinter Nights, narrative game mods, 94
new Unity projects, creating, 360–362
Nintendo Switch, screen size/resolution, 83
"Nintendo thumb," 34
nodes, brainstorming/ideation, 115
noise
archipelagos, turning noise into, 603–612
octaves, avoiding in DOTS, 600–602
Perlin noise
archipelagos, turning noise into, 603–612
DOTS image creation, 593–602
improving with octaves, 590–593
OOP image creation, 586–593
noisy environments, 84
nonshorting operators, 407–409
Not Equal To operator (!=), 414
not liking/liking ideas, playtesting, 169
NOT operator (!), 406
note cards, prototyping, 155
notebooks, prototyping, 155
notes, taking (playtesting), 173–175
notifications, missed apple (Apple Picker), 668–672
novel decisions, 148
NPC (Non-Player Characters)
indirect player guidance, 240
minor NPC development, 58–59
modeling behavior
emotional connections, 240–241
negative behaviors, 241
positive behaviors, 241
safety, 241
null arrays, skipping with foreach loops, 454–455
Numbers (Apple), 190
numerals, hexadecimal, 1030–1031
"Numbers Everyone Should Know," 577

objectives
Bartok, 3
conflicting objectives, 44–45
FDD elements framework, 25
inscribed mechanics, 43
conflicting objectives, 44–45
immediacy of objectives, 42–43
importance of objectives, 42–44
long-term objectives, 42
mid-term objectives, 42
optional objectives, 42–44
player relationships, defining, 44–47
primary objectives, 42–44
short-term objectives, 44
spaces, 48

Object-Oriented Programming (OOP), 540–542

Boids project
Attractor GameObject, 549–551
Attractor script, 551, 553–555
Boid script - part 1, 558
Boid script - part 2, 561–567
Boid script - part 3, 570–573
Boids project, 551
Boids values, 573
LookAtAttractor script, 551, 557–558
Neighborhood script, 551, 567–570
Reynolds, Craig W.542
setup, 542–543
Simple Boid model, 543–548
Spawner script, 551, 558–561
classes
data stored by reference, 579–580
Unity classes, data storage, 580
flock of birds simulation, 540–542
image creation, 586–593
issues with, 579–580
monolithic programming, 540–542
Perlin noise, 586–593
Unity, 354
objects. See GameObjects
octaves
improving Perlin noise, 590–593
noise octaves in DOTS, avoiding, 600–602
Okami
empathetic characters versus avatars, 60
touch aesthetics, 50

OnCollisionEnter method, Space SHMUP,
851–852
online playtesting, 181–183
online resources, Unity development,
1198–1199
open games, 11
OpenOffice Calc (Apache), 190
operations
Bitwise operators, System.Flags enums,
780–782
Boolean operations
AND operator (&), 406
bitwise Boolean operators, 409
combining, 409–410
if statements with, 417–418
logical equivalence, 410
NOT operator (!), 406
| (OR operator), 406
comparisons operators, 410
approximate float comparisons, 412
Assignment operator (=), 411, 418
Greater Than operator (>), 414
Greater Than or Equal To operator (>=), 415
Is Equal To operator (==), 411–413
Less Than operator (<), 414
Less Than or Equal To operator (<=), 415
Not Equal To operator (!=), 414
nonshorting operators, 407–409
shorting operators, 407–409, 1154
testing equality by value/reference, 412–413
OR operator (|), 406
optimal strategies, 72
optional objectives, 42–44
optional parameters, functions, 486–487
organizing Project pane (Unity), 716–717
Origin Systems, Ultima IV, 61
orthographic cameras, 634–636
Osu! Tatakae! Ouendan, VRO, 78–79
out loud (playtesting), thinking, 168–169
outcomes
cumulative outcomes, 76
defined, 76
FDD elements framework, 26
final outcomes, 76–77
immediate outcomes, 77
quest outcomes, 76
unequal outcomes, 13
overall damage, showing, 224–225
overloading functions, 485–486
overriding field values in inspector, 641
overscoping, 120–121
Papa Sangre, background noise, 53
paper games
decks of cards, 213–215
dice, 212
inscribed technology, 65
randomizer technologies, 212
decks of cards, 213–215
dice, 212
spinners, 212–213
weighted distributions, 215–216
spinners, 212–213
weighted probabilities, 216–217
paper prototyping, 151
2D adventure game level, 157–159
combat, 162
playtesting, 161–162
shortcuts, 161
traversal mechanics, 159–161
3x5 note cards, 155
benefits of, 152
best uses, 162–163
card sleeves, 155
cards, 153
collaborative prototyping, 152
dice, 153
focused prototyping/testing, 152–153
grids, 154–155
initial development speeds, 152–153
interfaces, 156
iteration speeds, 152
LEGO bricks, 155
low technical barriers to entry, 152
movement systems, 154–155
notebooks, 155
paper, 153
pipe cleaners, 155
playing pieces, 155
poor uses of, 163–164
post-it-notes, 155
tools, 153–155
traversal mechanics, 159–161
whiteboards, 155
parallel play interaction pattern, 44
parallel processing, DOD, 576
paralysis, choice, 234–235
parameters functions, 478–479
optional parameters, 486–487
params keyword, 487–489
PaRappa the Rapper, VRO, 78–79
parentheses ( ), C# programming language, 415
partial absolute references, 193–194
particle systems, 80
Passage, 11–12
Pauling, Linus
brainstorming/ideation, 113–114
pauses between rounds, Prospector, 1006–1007
PCO (Procedural Composition), 79
Pearce, Celia, 17
games, defined, 16–17
pen-and-paper RPG, 59–61
percent chance for each shot, balancing
weapons, 220–221
percentage symbol (%), % (modulo operators), 436
performative empowerment, 144
Perlin noise
archipelagos, turning noise into, 603–612
DOTS image creation, 593–602
improving with octaves, 590–593
OOP image creation, 586–593
permutations, 217–218
Bulls and Cows, 217–219
with repeating elements, 219
without repeating elements, 219
Person Charts, BDC, 282–283
personal expression/communication,
designer-centric goals, 132–133
perspective cameras, 634–636
Philosophical Investigations, 14
physical interfaces, indirect player guidance,
235–236
physics
engines, fixed updates, 556–557
layers, Apple Picker, 651–652
puzzles, 260
Space SHMUP, 790–792
picking up items, Dungeon Delver, 1135–1137,
1181–1184
pipe cleaners, prototyping, 155
pipes (|), | (OR operator), 406
pips, adding to cards, 934–935
Pixel Junk Shooter, puzzle design, 260–261
Planetfall, developing player relationships,
86–87
planning sprints, Scrum, 270–271
play, meaningful, 71, 147
players
actions, tracking, 61
avatars, 240
colorblindness, 85
epilepsy, 85
external tools, 95–96
fair play, 312

game volume, 84
goals, 130, 134
  attention/involvement, 145–147
  empowerment, 142–144
  experiential understanding, 149–150
  flow, 138–141
  fun, 134–135
  interesting decisions, 147–149
  lusory attitude, 136–137
  magic circle, 138
  structured conflict, player-centric goals, 141–142

grid alignment, Dungeon Delver, 1078–1079

guidance, 232
  direct guidance, 232–234
  indirect guidance, 234–240
  integration, 245
  sequencing, 243–244
  teaching new skills/concepts, 243–245

intent, 74–75

interaction patterns, 43–45
  cooperative play interaction pattern, 44
  FDD elements framework, 28
  multilateral competition interaction pattern, 44
  parallel play interaction pattern, 44
  player versus player interaction pattern, 46
  single player interaction pattern, 44
  team competition interaction pattern, 44
  unilateral competition interaction pattern, 44

interactive fiction, 86

migraines, 85

player versus player interaction pattern, 46

Prospector, feedback on scores, 1007–1013

relationships
  citizens, 45
  collaborators, 45
  competitors, 44
  defining with objectives, 44–47
  developing through shared experiences, 86–87
  game masters, 45
  inscribed mechanics, 40, 43–46
  protagonists, 47
  roles of, 68–69
  types of, 74–75

views, MDA framework, 23

Playful Production Process: For Game Designers (and Everyone), A, 121–122, 126

playing pieces, prototyping, 155

playtesting
  2D adventure game level, 161–162
  ADL, 182–183
  analysis, 5–6, 7–8
  AT, 185
  Bartok, 4–5
  biases, 169
  circles of playtesters, 169
    acquaintances, 171–172
    Internet, 172
    trusted friends, 170
    you, 170
  data tables, 49
  dynamic elements, 28
  FDD elements framework, 28
  flukes, 7
  focus testing, 183
  formal group playtesting, 176
  formal individual playtesting, 176–181
  great playtesters, 168–169
  importance of, 168
  informal individual playtesting, 172–175
  interest polling, 184
  investigators versus playtesters, 168
  liking/don’t liking ideas, 169
  modifying rules, 6–7
  notes, taking, 173–175
  online playtesting, 181–183
  prototyping, 161–162
  Quality Assurance (QA) testing, 184
  self-analysis, 169
  separating elements, 169
  thinking out loud, 168–169
  tissue playtesters, 170–172
  usability testing, 184
  Warshmallows, 179–180

Plenty-Coups, Chief
  counting coup, 141–142

plots
  first plot points, three-act dramatic structures, 55
  free will versus, 59
  inscribed narratives, 53
  second plot points, three-act dramatic structures, 58

plus symbol (+), ++ (increment operators), 428

Pogo.com, Crazy Cakes, 182–183

points accumulation, Apple Picker, 665–668

Pokemon Go
  epilepsy, 85
  parallel play interaction pattern, 44
  player relationships, 45
Poker, 25
pop-ups, direct player guidance, 234
position/sliding block puzzles, 260
positive behaviors, NPC, 241
positive feedback, game balance, 228–229
post-it-notes, prototyping, 155
post-release phase, game development, 119
PowerUps, *Space SHMUP*, 857–869, 872–876
prefabs, 373–378
doors, *Dungeon Delver*, 1109
instance overrides, applying to prefabs, 641–642
Skeletos (enemy), *Dungeon Delver*, 1109
premises
FDD elements framework, 28
inscribed narratives, 52–54
preproduction deliverables, scope management, 121–122
Macro Charts, 126
Macro Documents, 124–126
Vertical Slices, 122–123
vertical slices, 118–119
preproduction phase, game development, 118–119
presentation, puzzle design, 256
pricing, Unity, 330
primary objectives, 42–44
*Prince of Persia: The Sands of Time*, plots versus free will, 57
private BoundsCheck bndCheck, *Space SHMUP*, 816–819
private fields, viewing in classes, 734
probability. See also randomizer technologies
dice probability with Google Sheets, 191
Monte Carlo method, 211
"Ten Rules of Probability Every Game Designer Should Know," 207–211
weighted probabilities, 216–217
probability
math of probability, 207–211
tables, 51
problems (complex), breaking down, 315
procedural aesthetics, 77
procedural music, 78
HRS, 78–79
PCO, 79
VRO, 78–79
procedural visual arts, 80
particle systems, 80
procedural animation, 81
procedural environments, 82
shaders, 81–82
procedural animation, 81
procedural environments, 82
procedural languages, 352–353
procedural music, 78
HRS, 78–79
PCO, 79
VRO, 78–79
procedural visual arts, 80
particle systems, 80
procedural animation, 81
procedural environments, 82
shaders, 81–82
procedures, 25, 70
product backlogs/task lists, Scum, 269
Product Owners, Scrum, 268
production phase, game development, 118–119
programmatic collisions, Tilemaps, 1061
programming languages. See also C#
learning, 331–334
procedural languages, 352–353
programming/digital systems
breaking down complex problems, 315
code libraries, 314–315
computer languages, 313–314
simple instructions, 313
systems thinking, 312
Unity game development environment, 315
programs, game development, 1200–1201
progression
inscribed dramatics, 62
inscribed paper game technology, 63
tables, 48
Project pane (Unity), 339
*Dungeon Delver*, 1026
organizing, 716–717
projectiles
Mission Demolition, 690
instantiating projectiles, 694–698
ProjectileLine Trails, 728–734
RigidBody insomnia, 725–728
*Space SHMUP*
hero’s bullet, 800–801
scripts, 803
shooting, 800
shooting, adding shooting capability, 800–801
shooting, destroying enemies, 804
shooting, weapon GameObjects, 844–851
projects See also game builds; game prototype tutorials
Boids project
Attractor GameObject, 549–551
Attractor script, 551, 553–555
Boid script - part 1, 558
Boid script - part 2, 561–567
Boid script - part 3, 570–573
Boids project, 551
Boids values, 573
LookAtAttractor script, 551, 557–558
Neighborhood script, 551, 567–570
Reynolds, Craig W. 542
setup, 542–543
simple Boid model, 543–548
Spawner script, 551, 558–561
Collections Examples project, setup, 442–443
DOTS Example project, setup, 582–586
Enemy Class Examples project
Enemy class on GameObjects, 534
setup, 524-TEXT NOT FOUND IN PRE XML FILE
Function Examples project, setup, 474
"Hello World" project
adding color, 381–382
comments in scripts, 369
creating scripts, 363–368
cube environments, 378–381
debugging, attaching scripts, 500–502
debugging, compile-time bugs, 495–500
debugging, removing scripts, 500–502
debugging, runtime errors, 502–504
debugging, stepping through errors, 506–507
deleting cubes, 467–471
disabling scripts, 370
folder configuration, 361–362
manipulating GameObjects, 370–373
prefabs, 373–378
setup, 360
shrinking cubes, 467–471
start() function versus update() function, 370–378
Loop Examples, setup, 424–426
new Unity projects, creating, 360–362
small game projects, 1199
Updraft Coding Challenge
filling in blanks, 1192–1194
starting, 1191–1192
properties
arrays, 455–456
classes, 524, 527–530
Dictionaries, 450
as fields, 527–530
functions as, 483–484
Lists, 446
Unity issues, 728
Prospector Solitaire, 898–899, 969–970
adding
backs to cards, 937–938
face art to cards, 936–937
game elements, 972
background images, 983–985
BézierMover class, 987–991
build settings, 903
building cards, 922–938
cameras, 906–907
classes, 948–961
clickable cards, 962–964
element of play, 900–901
feedback on player scores, 1007–1013
FloatingScore GameObject, 991–999
game logic, 961–962
Game pane, 906–907
gold cards, 1017
GUI, 985–986
initial layout, 899–900
JSON through code, 913–917
managing rounds, 972–975
matching cards in mine, 964–968
Mine Tableau layout, 940–948
mobile devices, 1018
moving cards, 1017–1018
pauses between rounds, 1006–1007
pips, adding to cards, 934–935
Prospector_Scene_0, 905
rules, 900
ScoreBoard class, 1000–1001
ScoreBoard GameObject, 999
scoring, 975–983, 985–986, 999–1006, 1007–1013
setup, 901–902, 906–907, 971
shuffling cards, 939–940
silver cards, 1017
sorting cards, 954–958
sprites
building cards from sprites, 931–934
cards, constructing from sprites, 911–912
gathering references to the deck, 918–920
importing images as, 907–909
prefab GameObjects as sprites/cards, 921–922
slicing rank images as sprites, 909–911
Unity window layout, 906
updating ScoreManager script, 1001–1006
WebGL module, 1013–1016
installing, 903–904
switching to, 904–905
protagonists
player relationships, 47
silent protagonists, empathetic characters
versus avatars, 60
prototyping, 151
2D adventure game level, 157–159
combat, 162
playtesting, 161–162
shortcuts, 161
traversal mechanics, 159–161
3x5 note cards, 155
benefits of, 152
best uses, 162–163
card sleeves, 155
cards, 153
collaborative prototyping, 152
dice, 153
focused prototyping/testing, 152–153
girds, 154–155
initial development speeds, 152–153
interfaces, 156
iteration speeds, 152
LEGO bricks, 155
low technical barriers to entry, 152
movement systems, 154–155
notebooks, 155
paper, 153
pipe cleaners, 155
playing pieces, 155
poor uses of, 163–164
post-it-notes, 155
tools, 153–155
traversal mechanics, 159–161
whiteboards, 155
pseudocode, 440
Psychic Bunny, Freeq, 53
pure puzzles, 251
purpose of spaces, 47–50
puzzle design, 248, 262–263
action puzzles, 250–251, 260–262
boss fights, 261–262
construction puzzles, 251, 256
defining, 248–250
dexterity/timing, 262
genres of, 250–251
goals, 256–257
image/media puzzles, 253
inspiration, 255
Kim, Scott, 248–250, 255–256
levels, 255
logic puzzles, 253
mixed-mode puzzles, 254
modes of thought, 252
physics puzzles, 260
presentation, 256
pure puzzles, 251
reasons for playing, 251–252
rules, 255
sequencing, 256
simplification, 255
single-mode puzzles, 253
sliding block/position puzzles, 260
solving puzzles, 257–259
stealth puzzles, 261
story puzzles, 251
strategy puzzles, 251
testing, 255
Tetris, 255
traversal puzzles, 261
word puzzles, 253

Q
Quake, machinema, 94–95
Quake 2, game mods, 92
Quality Assurance (QA) testing, 184
quaternion variables/functions, 395–396
Queasy Games, Sound Shapes, 92
questions, playtesting analysis, 5–6
quests
outcomes, 76
side quests, 58
queues, 441
quitting applications, force, 426, 509

R
race conditions, 533, 869–872
randomization
directionality, Apple Picker, 647–648
inscribed paper game technology, 65
randomized items, Dungeon Delver,
1140–1143
randomizer technologies, 212
decks of cards, 213–215
dice, 212
spinners, 212–213
weighted distributions, 215–216
weighted probabilities, 216–217
spawning enemies, Space SHMUP, 787–790
rating phase, brainstorming/ideation, 117
Ravensburger, Up the River, 62
RectTransform tool (T), Unity, 380
recursive functions, 489–491
Red Dead Redemption, three-act dramatic
structures, 58
Red vs. Blue, machinema, 94
reference-based data, avoiding in DOTS, 595–599
references
absolute references, 193–194
partial absolute references, 193–194
relative references, 193
reflexive attention, 145
reinforcing mechanics, inscribed dramatics, 62
relationships, player
citizens, 45
competitors, 44
defining with objectives, 44–47
developing through shared experiences, 86–87
game masters, 45
inscribed mechanics, 40, 43–46
protagonists, 47
relative references, 193
releases, Scrum, 270–271
removing scripts, 500–502
Renderer component, GameObjects, 400
Requirements Documents, 124–126
resolution
three-act dramatic structures, 56–58
visual play environments, 83
resources
assets, 49
attributes, 49
FDD elements framework, 25
inscribed mechanics, 40, 47–49
online resources, Unity development, 1198–1199
Resources folder, Dungeon Delver, 1026
responsibilities of designers, Layered Tetrad, 39–40
restarting games, Space SHMUP, 797–799
retrospectives, sprint, 271
Return of the Obra Din
image/media puzzles, 253
logic puzzles, 253
rewards, inscribed dramatics, 62
Reynolds, Craig W.
Boids project, 542
"Flocks, Herds, and Schools: A Distributed Behavioral Model," 542
riffle shuffles, 5
"right-clicking" on mouse, macOS, 361
RigidBody component
GameObjects, 372, 402
projectiles, Mission Demolition, 725–728
rising action, five-act dramatic structures, 54
Roberts, Sam, 17
games, defined, 16–17
RoboCup, roles of players, 68–69
RoboRally, 113
Rock Band, 235–236
Rockstar Games, Red Dead Redemption, 58
Rogers, Scott
emergence, 69–70
Rogue, final outcomes, 77
Rohrer, Jason
Passage, 10, 11
role fulfillment, empathetic characters versus avatars, 63
Romeo and Juliet, 54–55
room to room movement, Dungeon Delver, 1087–1091
Rotate tool (E), Unity, 379
rounds
comparing (analysis), 7–8
Prospector
managing, 972–975
pauses between rounds, 1006–1007
rows/columns, Google Sheets
adding columns, 194–195
creating rows, 194
filling rows with data, 196
iterating Die A rows, 197
making Die A rows, 196–197
making Die B rows, 197–198
setting column widths, 195
royalty points, 306–307
RPG (Role-Playing Game), pen-and-paper RPG, 59–61
rules
Bartok 3, 6–7
explicit written rules, 46
FDD elements framework, 25
house rules, 73–74
implicit rules, 46
inscribed mechanics, 40, 46–48
modifying, 6–7
Prospector, 900
puzzle design, 255
written rules, 46
Rules of Play, meaningful play, 71
rumble-style player feedback, touch aesthetics, 51
runtime errors, debugging, 502–504
Ryan, Malcolm, 2
SAFETY, MODELING NPC BEHAVIOR

S

safety, modeling NPC behavior, 241
Salen, Katie
meaningful play, 147
Rules of Play, 71
Sarkeesian, Anna
Feminist Frequency, 99–101
saving Google Sheets, 199
Scalak, puzzle design, 256
Scale tool (R), Unity, 379
scarcity, direct player guidance, 232
Scene pane (Unity), changing, 380
Scene/Prefab pane (Unity), 338–339
Schell, Jesse
design phase, iterative design, 108–109
Elemental Tetrad framework, 30–31
games, defined, 13
indirect player guidance, 234–240
inscribed mechanics, 43
interest curves, 145–146
"Ten Rules of Probability Every Game Designer Should Know," 207–211
testing phase, iterative design, 111–112
scientific notation, 386–387
scope
functions, 476
of variables, 389
scoping
IndieCade Game Festival, 120
managing with preproduction deliverables, 121–122
Macro Charts, 126
Macro Documents, 124–126
Vertical Slices, 118–119, 122–123
overscoping, 120–121
Star Wars, 120–121
ScoreBoard class, Prospector, 1000–1001
ScoreBoard GameObject, Prospector, 999
ScoreCounter texts, Apple Picker, 662–664
ScoreManager script, Prospector, updating, 1001–1006
scoring, Prospector, 975–983, 985–986, 999–1006
Scotland Yard (Mr. X), unilateral competition interaction pattern, 44
screen size/resolution, visual play environments, 83
screen variables, 397
scripts, 402
attaching, 500–502
autocompleting scripts, Visual Studio, 365–366
Boids project
Attractor script, 551, 553–555
Boid script, 551
Boid script - part 1, 558
Boid script - part 2, 561–567
Boid script - part 3, 570–573
LookAtAttractor script, 551, 557–558
Neighborhood script, 551, 567–570
Spawner script, 551, 558–561
enemies, Space SHMUP, 773–787
enums (enumeration), 742
execution order, 1040
formal group playtesting, 176
GridMove scripts, Dungeon Delver, 1085–1087
headers, 553–555
InRoom scripts, Dungeon Delver, 1070–1072
linear interpolation, 567
matching names with classes, 525–526
removing, 500–502
ScoreManager script, Prospector, updating, 1001–1006
Space SHMUP, projectiles, 803
TileSwapManager scripts, Dungeon Delver, 1099–1101
UITextManager scripts, 1010–1013
Unity scripting reference, 642–643
variables, tuning, Apple Picker, 659–660
Visual Studio
autocompleting scripts, 365–366
script appearance, 365–366
scrolling backgrounds, Space SHMUP, 890–893
Scrum, 268. See also Agile Software Development
BDC, 269, 271–272
creating, 286
Daily Scrum worksheets, 283–285
Main worksheets, 273–280
Person Charts, 282–283
Task Rank Charts, 280–282
worksheets (overview), 272–273
Daily Scrum meetings, 268, 269–270
development teams, 268
methodologies, 268
product backlogs/task lists, 269
Product Owners, 268
releases, 270–271
Scrum Masters, 268
sprints, 270–271, 273–274, 277–279
teams, 268
second plot points, three-act dramatic
structures, 58
self-analysis, playtesting, 169
Sellers, Michael
Advanced Game Design: A Systems
Approach, 1201
semicolons (;)
debugging, 495
for loops, 431
separating elements, playtesting, 169
sequencing, 243–244, 256
serializable WeaponDefinition class, Space
SHMUP, 834–838
"serious" games, 133
service, games as a, 295
setting up
Apple Picker,
camera setup, 633–634
project guidelines, 624
Boids project, 542–543
Collections Examples project, 442–443
DOTS Example project, 582–586
Dungeon Delver, 1022–1023, 1097–1098
Enemy Class Examples project, 524-TEXT
NOT FOUND IN PRE XML FILE
Function Examples project, 474
"Hello World" project, 360
Loop Examples project, 424–426
Mission Demolition, 662–683
Prospector, 901–902, 906–907, 971
Space SHMUP, 755, 757–758, 809
settings, inscribed narratives, 52
Settlers of Catan
assets, 49
decks of cards, shuffling, 215
designing for strategy, 73
resources, 49
shaders, 81–82
Shadow of the Colossus, embedded experiences,
48
Shakespeare, William
Romeo and Juliet, 54–55
shared experiences, developing player
relationships, 86–87
Sheets. See Google Sheets
shields (hero ship), Space SHMUP, 764–766
shooting, Space SHMUP, 800, 833
delegate events, 842–844
eWeaponType enum, 833–834
hero’s bullet, 800–801
showing damage, 853–857
WeaponDefinition class, 834–842
shortcuts, dangers of, 161
shorting operators, 407–409, 1154
short-term objectives, 44
shrinking cubes, "Hello World" project,
467–471
shuffling cards
decks of cards, 215
Prospector, 939–940
side quests, 58
significands, 386–387
silent protagonists, empathetic characters
versus avatars, 60
silver cards, Prospector, 1017
similarity, visual design and indirect player
guidance, 236
simple instructions, systems thinking, 313
simplification, puzzle design, 255
single player interaction pattern, 44
single-mode puzzles, 253
Skeletos (enemy), Dungeon Delver, 1072–1075,
1109, 1145–1147
skills/concepts, teaching, 243
 skipping null arrays with foreach loops,
454–455
Skyrates, 135–136
online playtesting, 182
Skyrim
conflicting objectives, 44–45
custom game levels, 92
direct player guidance, 232
final outcomes, 76
game mods, 92
narrative game mods, 94
optional objectives, 42–44
plots versus free will, 58
primary objectives, 42–44
sleeves (card), prototyping, 155
Slices, Vertical, 118–119, 122–123
sliding block/position puzzles, 260
slingshots. Mission Demolition, 687–690,
691–702
small game projects, 1199
smell, five aesthetic senses, 51
Snakes and Ladders, 22–25, 27–29
layouts, 22–25
modifying for strategic game play,
24–26
social change, games for, 133
social media games, 136–137
Socializers (Bartle’s Hearts), 74
software
Agile Software Development, 266–267
ESA, 288–289
solitaire games. See Prospector
solving puzzles, 257–259
Something Else, Papa Sangre, 53
sorting cards in Prospector, 954–958
sound effects, 50–52
Sound Shapes, custom game levels, 92
Space SHMUP, 753–754, 807–808
adding elements, 894
asset packages, importing, 755–757
building game levels, 894–895
delegate events, 842–844
enemies
art assets, 771–773
damage, 792–797
deleting, 777–787
destroying, 804
Enemy_0, 810–811
Enemy_1, 812–819
Enemy_2, 819–826
Enemy_3, 826–832
Enemy_4, 836–888
OnCollisionEnter method, 851–852
PowerUps, 872–876
private BoundsCheck bndCheck, 816–819
programming, 811–813
randomly spawning, 787–790
scripts, 773–787
showing damage, 853–857
expanding weapon options, 865–866
game structure, 895
GUI, 895
hero ship
creating, 758–760
Hero update() method, 760–764
keeping on screen, 767–771
shields, 764–766
layers, 790–792
physics, 790–792
PowerUps, 857–869, 872–876
projectiles
hero’s bullet, 800–801
scripts, 803
shooting, 800
shooting, adding shooting capability, 800–801
shooting, destroying enemies, 804
shooting, weapon GameObjects, 844–851
race conditions, 869–872
restarting games, 797–799
scene setup, 757–758
setting up, 809
setup, 755, 757–758
shooting, 800, 833
adding shooting capability, 802–803
deleate events, 842–844
eWeaponType enum, 833–834
hero’s bullet, 800–801
showing damage, 853–857
WeaponDefinition class, 834–842
starfield backgrounds, 890–893
tags, 790–792
tuning settings, 888–890
tuning variables, 893
spaces
embedded experiences, 48
flow, 47
inscribed mechanics, 40, 47–48
landmarks, 47–48
objectives, 48
purpose of spaces, 47–50
Spades (Bartle’s Explorers), 74
Spawner script, Boids project, 551, 558–561
spawning enemies, Space SHMUP, 787–790
Spec Ops: The Line
justification, 62
motivation, 62
plots versus free will, 57
speeds
Mission Demolition, 710–716
prototyping
initial development speeds, 152–153
iteration speeds, 152
spelling errors, debugging, 494
Spider-Man 2, quest outcomes, 76
spinners, randomizer technologies, 212–213
spoilage mechanics, Farmville, 47, 135–137
spoilsports, 75
sporadic-play games, 135–136, 137
Spore, procedural animation, 81
spreadsheets
absolute references, 193–194
Excel (Microsoft), 189–190
Google Sheets, 189
adding columns, 194–195
balancing weapons, 219–228
charts, 204–206
clarity in, 199
color scale conditional formatting, 200
conditional formatting, 203
creating rows, 194
filling rows with data, 196
getting started, 191–192
iterating Die A rows, 197
labels, 199
making Die A rows, 196–197
making Die B rows, 197–198
naming documents, 194
saving, 199
setting column widths, 195
summing results, counting all die rolls, 202–203
summing results, counting sums of die rolls, 201–202
summing results, two dice, 201
weighted probabilities, 216–217
importance of, 188–189
LibreOffice Calc, 190
Numbers (Apple), 190
OpenOffice Calc (Apache), 190
partial absolute references, 193–194
relative references, 193
sprints, Scrum, 270–271, 273–274, 277–279
sprites
building cards from sprites, 931–934
cards, constructing from sprites, 911–912
Dungeon Delver
CollisionTiles sprites, 1061–1064
naming conventions, 1043–1044
gathering references to the deck, 918–920
importing images as sprites, Prospector, 907–909
slicing rank images as sprites, 909–911
square grids, movement systems, 154
stacks, 441–442
Star Wars, 52, 56–57, 120–121
Star Wars Jedi: Fallen Order, direct player guidance, 234
Star Wars: Knights of the Old Republic, plots versus free will, 57
starfield backgrounds, Space SHMUP, 890–893
Start() function versus update() function, 370–398
Start() method versus awake() method, 814–815
starting
game design projects, 305–308
Updraft, Coding Challenge, 1191–1192
state tracking, inscribed paper game technology, 63
statements
break statements, exiting loops, 433–435
commas (,) in, 432
conditional statements, 416
if statements, 416–418
if.else if.else statements, 418–419
if.else statements, 418
nesting if statements, 419
switch statements, 419–422
continue statements, skipping single iterations, 435
jump statements, in loops, 433
static class variables/functions, 390–392
static functions, 391–392
static methods, arrays, 456–457
static typing
C# programming language, 351–352
variables, 384–385
stealth puzzles, 261
Steinkuehler, Constance, 90–91
stepping through errors, debugging, 506–507
stopping apples from falling too far, Apple Picker, 653–655
summing
counting all die rolls, 202–203
sums of die rolls, 201–202
results of two dice, 201
Super Mario Bros.
integrated actions, 71
procedural music, 78–79
teaching new skills/concepts, 243
Super Mario Galaxy, particle systems, 80
Super Mario Odyssey, GamerGate, 99–100
superclasses, 535–538, 950
support, Unity, 330
Swain, Chris
  game design, 104
  game design, defined, 18
  Game Design Workshop, 18
Swink, Steve
  Game Feel, 623–624
Switch (Nintendo), screen size/resolution, 83
switch statements, 419–422
symmetric games, 188
System.Flags enums, Bitwise operators,
780–782
SystemInfo variables, 397
systems design, 118–119
systems thinking
  board games, 312
  simple instructions, 313

T
  tables
    inscribed mechanics, 41, 50–51
    playtest data tables, 49
    probability tables, 51
    progression tables, 48
tags, Space SHMUP, 790–792
Tales of the Arabian Nights, probability tables,
51
task assignments, Main worksheets (BDC),
274–275, 280
task lists/product backlogs, Scrum, 269
task Rank Charts, BDC, 280–282
taste, five aesthetic senses, 51
teaching, skills/concepts, 243
team competition interaction pattern, 44
Team Fortress 2, 294
tech tree, Civilization, 43
technical barriers to entry, prototyping, 152
Technik des Drama (The Technique of Drama),
Die, 54–55
Technique of Drama, The, 54–55
technology
  cultural layer, Layered Tetrad, 36
  cultural technology, 95
    game technology used outside games, 95
    player-made external tools, 95–96
  dynamic layer, Layered Tetrad, 34, 88
  Elemental Tetrad framework, 28
  game technology used outside games, 95
  inscribed layer, Layered Tetrad, 33
  inscribed technology, 63
  digital games, 65
  paper games, 63–65
testing
  AT, 185
  efficiency when testing, 974–975
  focus testing, 183
  focused prototyping/testing, 152–153
  Grappler, Dungeon Delver, 1180–1181
  interest polling, 184
  operation equality by value/reference,
  412–413
playtesting
  2D adventure game level, 161–162
  analysis, 5–6, 7–8
  ADL, 182–183
  AT, 185
  Bartok, 4–5
  biases, 169
  circles of playtesters, 169–172
data tables, 49
dynamic elements, 28
FDD elements framework, 28
flukes, 7
focus testing, 183
formal group playtesting, 176
formal individual playtesting, 176–181
great playtesters, 168–169
importance of, 168
informal individual playtesting, 172–175
interest polling, 184
investigators versus playtesters, 168
liking/don’t liking ideas, 169
modifying rules, 6–7
online playtesting, 181–183
prototyping, 161–162
QA testing, 184
self-analysis, 169
separating elements, 169
taking notes, 173–175
thinking out loud, 168–169
tissue playtesters, 170–172
usability testing, 184
Warshmallows, 179–180
puzzle design, 255
QA testing, 184
usability testing, 184
testing phase, iterative design, 105, 112
Tetris, 255
Teuber, Klaus
Settlers of Catan, 73
texture, visual design and indirect player
  guidance, 239
thinking out loud, playtesting, 168–169
three-act dramatic structures, 56–58
Tilemaps, Dungeon Delver, 1031–1042
programmatic collisions, 1061
programmatically filling collisions, 1065–1069
TileSwapManager scripts, Dungeon Delver, 1099–1101
TileSwaps, Dungeon Delver, 1099, 1101–1107, 1144–1147
time-base games, 644–645
timing/dexterity, puzzle design, 262
tinting Unity window, 505
tissue playtesters, 170–172
Tomb Raider, aesthetics, 36
Tony Hawk’s Pro Skater
    gameplay as art, 93
    performative empowerment, 144
totalling hours, Main worksheets (BDC), 277–280
touch, five aesthetic senses, 50–51
tracking
    player actions, 61
    state tracking, inscribed paper game technology, 63
traditional dramatics, 55
five-act dramatic structures, 54–55
three-act dramatic structures, 56–58
trails, visual design and indirect player guidance, 236
Transform tools, Unity, 379–380
Transform component, GameObjects, 372, 400
Transform tool (Y), Unity, 380
Translate tool (W), Unity, 379
transmedia (authorized), cultural layer and, 96–97
traversal mechanics, prototyping, 159–161
traversal puzzles, 261
trusted friends, circles of playtesters, 170

tuning variables, 893
tutorials, game prototype. See also game builds; projects
Apple Picker, 621–623
    art assets, 624–633
    boids, 551
    cameras, setup, 633–634
    catching apples, 658–659
    coding, 637–641
    destroying baskets, 670–672
    directionality, 646–648
    DOTS, 582
    dropping apples, 649–651
    game management, 661–662
game panel settings, 636–637
GUI, 661–662
Hello World, 359
HighScore texts, 662–664, 672–678
instance overrides, applying to prefabs, 641–642
instantiating baskets, 655–656
missed apple notifications, 668–672
movement systems, 643–646
moving baskets with mouse, 657–658
physics layers, 651–652
points accumulation, 665–668
purpose of, 623
ScoreCounter texts, 662–664
setup, 624
stopping apples from falling too far, 653–655
tuning script variables, 659–660
Dungeon Delver, 1019–1021, 1095–1096
    anti-aliasing issues, 1041–1042
    cameras, 1023
    cameras, GUI cameras, 1024–1025
    cameras, main camera, 1024–1025
    component-based design, 1021–1022
damage, 1125–1135
    DeliverTiles, 1026–1028
    DeliverLevel_Eagle Text files, 1028–1031, 1033–1035
doors, keys, 1111–1121, 1138–1140
    Droy (hero), 1042
    Droy (hero), animation, 1044–1049
    Droy (hero), attack animations, 1055–1058
    Droy (hero), camera movement, 1059–1061, 1091–1094
    Droy (hero), collisions, 1069
    Droy (hero), giving damage, 1130–1135
    Droy (hero), Grappler attacks, 1174–1180
    Droy (hero), GUI connections, 1123–1125
    Droy (hero), health, 1121–1122
    Droy (hero), IGadget interface, 1150–1154
    Droy (hero), movement systems, 1049–1053, 1087–1091
    Droy (hero), naming conventions, 1043–1044
    Droy (hero), picking up items, 1135–1137, 1181–1184
    Droy (hero), taking damage, 1127–1130
    Droy (hero), walking animations, 1054–1055
    Droy (hero), weapons, 1059–1061
dropping items, keys, 1138–1140
dropping items, randomized items, 1140–1143
TUTORIALS, GAME PROTOTYPE

- tutorials, game prototype. See also game builds; projects (continued)
  - dungeon design, 1143–1147
  - enemies, dropping keys, 1138–1140
  - enemies, dropping randomized items, 1140–1143
  - enemies, giving damage, 1127–1130
  - enemies, Skeletos, 1072–1075, 1109, 1145–1147
  - enemies, taking damage, 1130–1135
  - Game pane, 1024
  - Grappler, 1147–1148
  - Grappler, building, 1154–1159
  - Grappler, collisions, 1169–1173
  - Grappler, firing, 1169
  - Grappler, picking up items, 1181–1184
  - Grappler, pulling Dray (hero) in, 1174–1180
  - Grappler, secondary abilities, 1159–1169
  - Grappler, testing, 1180–1181
  - grid alignment, 1078–1079
  - GridMove scripts, 1085–1087
  - IFacingMover interface, 1079–1084
  - IGadget interface, 1148–1154
  - ISwappable interface, 1107–1111, 1145–1147
  - keys, 1111–1121, 1138–1140
  - maps/guidance systems, 1031–1042
  - picking up items, 1135–1137, 1181–1184
  - prefabs, 1109
  - Project pane, 1026
  - randomized items, 1140–1143
  - Resources folder files, 1026
  - room to room movement, 1087–1091
  - setup, 1022–1023, 1097–1098
  - sprites, CollisionTiles sprites, 1061–1064
  - sprites, naming conventions, 1043–1044
  - storing two-dimensional data in linear arrays, 1039–1040
  - Tilemaps, 1031–1042
  - Tilemaps, programmatic collisions, 1041
  - Tilemaps, programmatically filling collisions, 1065–1069
  - TileSwaps, 1099, 1144–1147
  - TileSwaps, doors, 1101–1107
  - Mission Demolition, 681–683
  - art assets, 684–690
  - cameras, follow cameras, 702–710
  - cameras, settings, 685–688
  - castles, 717–725, 734–737
  - coding, 691
  - coding, castles, 717–725, 734–737
  - coding, collision detection, 689–699
  - coding, creating slingshot class, 691–702
  - coding, follow cameras, 702–710
  - coding, goals, 734–736
  - coding, instantiating projectiles, 694–698
  - coding, multiple views, 745–751
  - coding, organizing Project pane (Unity), 716–717
  - coding, projectiles, 725–734
  - coding, showing when slingshot is active, 692–693
  - coding, UI, 737–738
  - coding, vector/speed, 710–716
  - directional light, 685
  - game management, 739–744
  - goals, 734–736
  - ground, 684–685
  - multiple views, 745–751
  - projectiles, 690
  - projectiles, instantiating, 694–698
  - projectiles, ProjectileLine Trails, 728–734
  - projectiles, Rigidbody insomnia, 725–728
  - prototype concept, 683–684
  - setup, 662–683
  - slingshots, 687–690
  - slingshots, creating slingshot class, 691–702
  - slingshots, showing when active, 692–693
  - UI, 737–738
  - Prospector, 898–899, 969–970
  - adding backs to cards, 937–938
  - adding face art to cards, 936–937
  - adding game elements, 972
  - background images, 983–985
  - BézierMover class, 987–991
  - build settings, 903
  - building cards, 922–938
  - cameras, 906–907
  - classes, 948–961
  - clickable cards, 962–964
  - example of play, 900–901
  - feedback on player scores, 1007–1013
  - FloatingScore GameObject, 991–999
  - game logic, 961–962
  - Game pane, 906–907
  - gold cards, 1017
  - GUI, 985–986
  - initial layout, 899–900
  - JSON through code, 913–917
  - managing rounds, 972–975

Mission Demolition, 681–683
- art assets, 684–690
- cameras, follow cameras, 702–710
- cameras, settings, 685–688
- castles, 717–725, 734–737
- coding, 691

Prospector, 898–899, 969–970
- adding backs to cards, 937–938
- adding face art to cards, 936–937
- adding game elements, 972
- background images, 983–985
- BézierMover class, 987–991
- build settings, 903
- building cards, 922–938
- cameras, 906–907
- classes, 948–961
- clickable cards, 962–964
- example of play, 900–901
- feedback on player scores, 1007–1013
- FloatingScore GameObject, 991–999
- game logic, 961–962
- Game pane, 906–907
- gold cards, 1017
- GUI, 985–986
- initial layout, 899–900
- JSON through code, 913–917
- managing rounds, 972–975
UITEXT MANAGER SCRIPTS

matching cards in mine, 964–968
Mine Tableau layout, 940–948
mobile devices, 1018
moving cards, 1017–1018
pauses between rounds, 1006–1007
pips, adding to cards, 934–935
Prospector_Scene_0, 905
rules, 900
ScoreBoard class, 1000–1001
ScoreBoard GameObject, 999
scoring, 975–983, 985–986, 999–1006, 1007–1013
setup, 901–902, 906–907, 971
shuffling cards, 939–940
silver cards, 1017
sorting cards, 954–958
sprites, building cards from sprites, 931–934
sprites, constructing cards from sprites, 911–912
sprites, gathering references to the deck, 918–920
sprites, importing images as, 907–909
sprites, prefab GameObjects as sprites/cards, 921–922
sprites, slicing rank images as sprites, 909–911
Unity window layout, 906
updating ScoreManager script, 1001–1006
WebGL module, 1013–1016
WebGL module, installing, 903–904
WebGL module, switching to, 904–905
Space SHMUP, 753–754, 807–808
adding elements, 894
building game levels, 894–895
delegate events, 842–844
enemies, art assets, 771–773
enemies, damage, 792–797
enemies, deleting, 777–787
enemies, destroying, 804
enemies, Enemy_0, 810–811
enemies, Enemy_1, 812–819
enemies, Enemy_2, 819–826
enemies, Enemy_3, 826–832
enemies, Enemy_4, 876–888
enemies, OnCollisionEnter method, 851–852
enemies, PowerUps, 872–876
enemies, private BoundsCheck bndCheck, 816–819
enemies, programming, 811–832
enemies, randomly spawning, 787–790
enemies, scripts, 773–787
enemies, showing damage, 853–857
expanding weapon options, 865–866
game structure, 895
GUI (Graphical User Interfaces), 895
hero ship, creating, 758–760
hero ship, Hero update() method, 760–764
hero ship, keeping on screen, 767–771
hero ship, shields, 764–766
importing asset packages, 755–757
layers, 790–792
physics, 790–792
PowerUps, 857–869, 872–876
projectiles, adding shooting capability, 800–801
projectiles, destroying enemies, 804
projectiles, hero’s bullet, 800–801
projectiles, scripts, 803
projectiles, shooting, 800
projectiles, weapon GameObjects, 844–851
race conditions, 869–872
restarting games, 797–799
scene setup, 757–758
setting up, 809
setup, 755, 757–758
shooting, 800, 833
shooting, adding shooting capability, 802–803
shooting, delegate events, 842–844
shooting, eWeaponType enum, 833–834
shooting, hero’s bullet, 800–801
shooting, showing damage, 853–857
shooting, WeaponDefinition class, 834–842
starfield backgrounds, 890–893
tags, 790–792
tuning settings, 888–890
tuning variables, 893
two-dimensional data, storing in linear arrays, 1039–1040
typos, debugging, 494–495

U
UI (User Interface), Mission Demolition, 737–738
UITextManager scripts, 1010–1013
Ultima IV, tracking player actions, 61
uncertainty, 13
Uncharted 3: Drake’s Deception
machinema, 94–95
particle systems, 80
visual design, indirect player guidance,
237–239
understanding, experiential, 12, 149–150
unequal outcomes, 13
unexpected mechanical emergence, 69–70
unilateral competition interaction pattern, 44
Unity, 330
accounts, creating, 334–335
asset packages, importing, 755–757
Attach to Unity button, repairing, 513
class instances, 391–392
classes, data storage, 580
comments (/), 384
Console pane, 339
creating projects, 335–337
CS0029 compile-time code errors, 388
CS0664 compile-time code errors, 387
CS1012 compile-time code errors, 388
DOD, 354–355
DOTS, 581–582
tutorial, 581–582
debugging, 510
enabling, 509–510
errors, 505
macOS debuggers, 510–511
Windows debugger, 511–513
development
evironment, 328
resources, 1198–1199
downloading, 324
ECS, 612–616
Editor tool, 380
ersors, debugging, 505
fixed updates, 556–557
folder names, changing, 362
functions
mathf functions, 396
static functions, 391–392
game development environment, 315
game loops, 424
Game pane, 339 1024
GameObjects, 383, 398
 Attractor GameObject, Boids project,
549–551
Box Collider component, 371
Collider component, 400–401
Dungeon Delver, 1075–1078
Enemy class on GameObjects, 534
Mesh Filter component, 371
Mesh Renderer component, 372
MeshFilter component, 400
MonoBehaviour subclasses as GameObject
components, 530–533
prefabs, 373–378
Renderer component, 400
RigidBody component, 372, 402
Transform component, 372, 400
generic methods (< >), 398
Hand tool (Q), 380
"Hello World" project
adding color, 381–382
comments in scripts, 369
creating scripts, 363–368
cube environments, 378–381
disabling scripts, 370
folder configuration, 361–362
manipulating GameObjects, 370–373
prefabs, 373–378
setup, 360
start() function versus update() function,
370–398
Hierarchy pane, 339
IGDPD layout
downloading, 340
manually arranging, 341–344
Inspector pane, 339
changing script values, 660–661
field names, 637–641
headers, 553–555
naming variables, 402
overriding field values, 641
setting play mode values, 646
 ISerializationCallbackReceiver Interface,
942–943
layouts
 IGDPD layout, downloading, 340
navigating, 338–339
Light Editor Theme, 338
naming conventions, 389–390
new projects, creating, 360–362
OOP, 354
physics engines, fixed updates, 556–557
pricing, 330
Project pane, 339
Dungeon Delver, 1026
organizing, 716–717
property issues, 728
race conditions, 533
reasons for choosing, 329–330
RectTransform tool (T), 380
Rotate tool (E), 379
sample projects, 335
Scale tool (R), 379
Scene pane, changing, 380
Scene/Prefab pane, 338–339
scripting reference, 642–643
support, 330
Transform tools, 379–380
Transform tool (Y), 380
Translate tool (W), 379
Unity 2020.3 LTS, installing, 326–327
Unity Hub
downloading, 324–326
installing, 324–326
variables, 384
application variables, 397
bool variables, 386
color variables, 387
class variables, 388
defining, 385
display variables, 386
false variables, 379
false Boolean variables, 385
float variables, 387
instance variables/functions, 390
int variables, 386
naming, 376 402
quaternion variables/functions, 395–396
scope, 389
static class variables/functions, 390–392
statically typed variables, 384–385
string variables, 388
SystemInfo variables, 397
Vector3 instance variables/functions, 393
Visual Studio, alternatives
JetBrains Rider, 328–329
MonoDevelop, 329
VS Professional/Enterprise, 329
VSCode, 328
VSComm, 328–329
VSMac, 328–329
Visual Studio, connections, 366
WebGL module
installing, 903–904
switching to, 905
website changes, 324
window layout, Prospector, 906
Unity Asset Store, implementation phase (iterative design), 110–111
University of Southern California, Interactive Media & Games Division (IMGD), 298
Uno, 2
Unreal, custom game levels, 92
Up the River
mechanics reinforcement, 62
optimal strategies, 72
Update() function versus Start() function, 370–398
upgrading
fixed updates, 556–557
ScoreManager script, Prospector, 1001–1006
Updraft Coding Challenge
filling in blanks, 1192–1194
starting, 1191–1192
uroboros game build example
brainstorming/ideation

collection phase, 115
collision phase, 116–117
discussion phase, 117
expansion phase, 114–115
rating phase, 117
idea cards, 115
idea collisions, 116–117
usability testing, 184

V
Valkyria Chronicles, direct player guidance, 233
variables, 384
application variables, 397
bool variables, 386
color variables, 387
class variables, 388
declaring, 385
defining, 385
float variables, 387
generic methods (< >), 398
instance variables/functions, 390
int variables, 386
iteration variables, Loop Examples project, 428
naming, 402
naming in Unity, 376
quaternion variables/functions, 395–396
screen variables, 397
static class variables/functions, 390–392
statically typed variables, 384–385
string variables, 388
SystemInfo variables, 397

variables

Vector3 instance variables/functions, 393
vection/speed, Mission Demolition, 710–716
Vector3 instance variables/functions, 393
linear interpolation, 567
Vectorized Playing Cards, 2
Vertical Slices, 118–119, 122–123
VFX Graph, particle systems, 80
view frustum, 634
views (multiple), Mission Demolition, 745–751
vision, five aesthetic senses, 49–51
visual arts (procedural), 80
particle systems, 80
procedural animation, 81
procedural environments, 82
shaders, 81–82
visual design, indirect player guidance, 236–239
visual play environments, 82
brightness, 83
resolution, 83
screen size/resolution, 83
Visual Studio
Attach to Unity button, repairing, 513
C# scripting
autocompleteing scripts, 365–366
script appearance, 365–366
spacing, 375
JetBrains Rider, 328–329
MonoDevelop, 329
VS Professional/Enterprise, 329
Unity, connections, 366
VSCode, 328
VComm, 328–329
VSMac, 328–329
volume, player-controlled game volume, 84
VRO (Vertical Re-Orchestration), 78–79

W
walking animations, Dungeon Delver, 1054–1055
Walt Disney Imagineering, visual design and indirect player guidance, 236–237
Warcraft III, game mods, 92
Warframe, gameplay as art, 93
Warshmallows, playtesting, 179–180
weapon GameObjects, Space SHMUP, 844–851
WeaponDefinition class, SpaceSHMUP
dictionaries, 838–842
serializable, 834–838

weapons
balancing with Google Sheets, 219–220
calculating average damage, 222
charting average damage, 223–224
duplicating weapon data, 225–226
example of, 227–228
percent chance for each shot, 220–221
rebalancing weapons, 226–227
showing overall damage, 224–225
Dray (hero), Dungeon Delver, 1059–1061
WebGL module
installing, 903–904
Prospector, 1013–1016
switching to, 904–905
website (Unity), changes to, 324
weighted distributions, 215–216
weighted probabilities, Google Sheets, 216–217
Werewolf, player relationships, 47
Westwood Studios, Blade Runner, 63–64
while loops, 424, 425, 426–428
whiteboards
markers, brainstorming/ideation (expansion phase), 115
prototyping, 155
Windows
debugging, 507–508, 511–513
force quitting applications, 509
Wittgenstein, Ludwig
games, defined, 14
Philosophical Investigations, 14
Wizards of the Coast
Dungeons & Dragons, 27, 59
cultural narratives, 93–94
cumulative outcomes, 76
dynamic narratives, 85
emergent narratives, 87
gameplay as art, 93
outcomes, 26
progression tables, 48
Magic: The Gathering, 113
Wong, Yin Yin
focused prototyping/testing, 152–153
word puzzles, 253
word/image puzzles, 254
word/logic puzzles, 254
working conditions, game companies, 292–293
worksheets, BDC, 272–273
Daily Scrum worksheets, 283–285
Main worksheets, 273–280
X

*World of Warcraft*
  - Damage Per Second (DPS) calculators, 95
  - parallel play interaction pattern, 44

Wright, Will
  - *Spore*, 81
  - written rules, 46

X-Wing
  - aesthetic goals, 53–54
  - procedural music, 78–79

Y

  - versus free will, 57

Z

zero-indexed arrays/lists, 440
Zimmerman, Eric
  - meaningful play, 147
  - *Rules of Play*, 71
Zork, interactive fiction, 86