APPENDIX D

TIPS FOR TEACHING FROM THIS BOOK

Since the first edition of this book was published in 2013, it has been the textbook for my undergraduate game design and programming classes, and if you're an educator, I hope that it will be as useful for you as it has for me.

Here you'll find semester calendars, tips, and advice for introductory classes in both design and programming classes.

The Goal of This Appendix

I added this appendix to the third edition of the book to pass on some of the knowledge I've gained through teaching game design and programming classes since 2001. Of course, this is just based on my experiences, and if you find that something else works better for you, I would love to hear about it. Most of my experience is teaching undergraduates with a strong interest in game design and little or no background in programming,[1] so you may need to modify things based on the audience you're teaching. There's a contact form on the book's website (http://book.prototools.net) where you can tell me about how you use this book in your classes.

Teaching Introduction to Game Design

The structure for this class began with the introductory Game Design Workshop class that was first created by Tracy Fullerton and Chris Swain at the University of Southern California, and which I took over and taught the four years that I was at USC. At USC, the course textbook is Tracy's excellent *Game Design Workshop*.[2] I moved to Michigan and published the first edition of this book in 2013, and I've been using this book in introductory design classes ever since.

Sample Class Calendar

Table D.1 presents an example calendar of how I approach teaching this class. I have always taught this class with roughly 4 hours of in-class time per week, split evenly between lecture and lab time. At USC, I had the luxury of lecturing to a class of 80 students and then teaching one of the four 20-student lab sections (my co-instructor and TAs (Teaching Assistants) handled the other labs). Since then, I have not been able to keep the labs that small, though I do recommend it if you can make it work.

1. Though I did teach from this book for three years in the Electrical Engineering and Computer Science department at the University of Michigan, and outside of U-M, I have often had graduate students and computer science students take my intro design and programming courses as well.
2. At the time, the edition I used was 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).

Table D.1 Sample Class Calendar for Introduction to Game Design

Week	In-Class (4 hrs/wk)	Readings for the Following Week
1	Lecture: Introduction to class *Bartok* Exercise in class (see Chapter 1) Lab: Play interesting paper games	1: Thinking Like a Designer 10: Game Testing
2	Lec: Discuss Chapter 1 Lab: Paper Game Project (P1) — Kickoff	2: Game Analysis Frameworks
3	Lec: Chapter 2 Lab: P1 — Playtest 1	3: The Layered Tetrad 4: The Inscribed Layer
4	Lec: Chapters 3 and 4 Lab: P1 — Playtest 2	5: The Dynamic Layer 6: The Cultural Layer
5	Lec: Chapters 5 and 6 Lab: P1 — Playtest 3 End of Week: P1 Project Due	7: Acting Like a Designer
6	Lec: Chapter 7 Lab: Roleplaying Game Project (P2) — Kickoff	8: Design Goals
7	Lec: Chapter 8 Lab: P2 — Play first team member's RPG	9: Paper Prototyping
8	Lec: Chapter 9 Lab: P2 — Play third team member's RPG End of Week: Project 2 Writeup Due	11: Math and Game Balance
9	Lec: Chapter 11 Lab: Paper Game Project (P3) — Kickoff	12: Guiding the Player
10	Lec: Chapter 12 Lab: P3 — Playtest 1	13: Puzzle Design
11	Lec: Chapter 13 Lab: P3 — Playtest 2	14: The Agile Mentality
12	Lec: Chapter 14 Lab: P3 — Playtest 3 End of Week: P3 Project Due	15: The Digital Game Industry
13	Lec: Chapter 15 Lab: Game Pitch Project (P4) — Kickoff	
14	Lec: Example Game Pitches Lab: P4 — Pitches Round 1	
15	Lec: Current events topic of your choosing Lab: P4 — Pitches Round 2	
Final	Final Exam: Pitches for Guest Judges (P4 turn-in is the live pitch to the judges)	

Core Learning Goals of the Class

There are several things that I want to get across in an intro game design class:

1. **The iterative process of design:** In the calendar, you can see that a tremendous amount of the class is devoted to playtesting student games. I have found this to be critical in impressing upon the students the necessity of iterating on their ideas.
2. **Willingness to discard things that aren't working:** New designers are often unwilling to accept sunk cost and throw out elements of their game that they have put a lot of time into. In paper games, I have particularly seen this when students have put significant work into constructing a board or deck of cards before testing enough. Encourage your students to make paper game elements as quickly as possible, then test the game, and only polish the physical board or cards *after* playtesting has verified their design.
3. **Game design is a job:** Some naive students may think that game design is the same kind of fun as playing games. You and I know that's not true, but it might take experience in this class for some students to catch on.
4. **You can't teach design, but you *can* teach process and history:** I feel pretty strongly that students learn design best by experiencing it through strong processes and through learning about examples of design in the history of our industry.

In this class, students spend most of their time designing *paper games*, which encompass everything from board and card games to roleplaying games (i.e., anything that doesn't require a computer). Meanwhile, most of the book material is about digital games. This isn't as much of a conflict as you might assume, and the combination of the two can prepare the students well for a subsequent class in digital game design or level design.

Elements of the Class

Let's look at a few of the core elements of the class.

Lecture

Because students are (or should be) reading the book chapters before coming to class, they will already have the background. Therefore, your lecture can be centered on surfacing current examples of the book topics and tying the concepts to games that the students might be currently playing. Video examples of gameplay can work well here. I have prepared a basic slide deck framework for each chapter that you can download from the book's website, but I strongly recommend that you go through and add your own timely video examples to illustrate the topics.[3] Rather than record these videos

3. I didn't include videos and such in my slides because they would soon be dated, and because getting the legal permission to make those videos part of the book materials would have been impossible.

yourself, you can find tools online to download and crop videos from sites like YouTube. I recommend that you "flip" the lecture as much as possible so that rather than presenting new information to the students, you're discussing the material with them, illustrated by recent examples that you've found. This is also a great time to present examples that counter the material in the book as a way to show the variety of game design out there.

Student Project Teams

All projects in this class are designed for teams of four students. I usually shuffle these teams to make sure that the students in the class are working with different people each time. Table D.2 shows a way that you can shuffle teams in a class of 20 students. As you can see, if you did more projects than there are teams in the class (a sixth project in this case), the pattern repeats, and the original teams would work together again. This system will also have issues if there are more students per team than there are teams in the class.

Table D.2 Shuffle for 20 Students

	Team A				Team B				Team C				Team D				Team E			
Project 1	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T
Project 2	E	J	O	T	I	N	S	D	M	R	C	H	Q	B	G	L	A	F	K	P
Project 3	I	R	G	P	M	B	K	T	Q	F	O	D	A	J	S	H	E	N	C	L
Project 4	M	F	S	L	Q	J	C	P	A	N	G	T	E	R	K	D	I	B	O	H
(Project 5)	Q	N	K	H	A	R	O	L	E	B	S	P	I	F	C	T	M	J	G	D
(Project 6)	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T

As you can see, for Project 2, Team A takes the following people from the Project 1 teams:

- 1st person from Team B
- 2nd person from Team C
- 3rd person from Team D
- 4th person from Team E

All other teams follow this same pattern, and I've created a Google Sheet that will do this for you, which you can find under "Appendix D" on the http://book.prototools.net

website. If you enter your students' names into the Project 1 line, it will shuffle them for all subsequent projects.[4]

Another benefit of this method is that it works very well for multidisciplinary teams. For example, if each team has a Fighter, Ranger, Thief, and Wizard (in that order), then shuffling the teams this way will ensure that each team *always* has a Fighter, Ranger, Thief, and Wizard.[5]

Of course, you will often have a class that doesn't have the right number of students for every team to have exactly four members. I usually find that teams of three are better than teams of five, but you should, of course, just do your best to make teams of relatively equal size.

Lab

As you can see in the calendar, lab time is spent almost entirely on playtesting. In a two-hour lab, I usually run three 30-minute playtest rounds. For each playtest, one member of each team stays behind to run the playtest of their game, while the other three members of the team split up and go play other games. If you or your TAs are in the class, I strongly encourage you to take part in the playtests as well. Make sure that the students running each playtest are keeping notes, and encourage the teams to have a different person run the playtest for each round. If there's time at the end of class, the teams should regroup and discuss the feedback they've received. I have also found that it is extremely helpful to create a digital way for students to submit feedback on the games that they played. Many years ago, I worked with my student Logan Olson[6] to create a php-based website to take feedback from each student and display it to the right teams, but you could much more easily create a simple Google form that took feedback for all teams and then sort the data based on the team it is intended for and send out an email to each team with their feedback (Google App Script could automate this process as well).

Week 1 Lab — "Play Interesting Paper Games"

While some students in your class will have broad experience with board and card games, many others will not. This lab is the chance to show them that board games can be more interesting and fun to play than *Monopoly*. Depending on your situation, you might consider inviting students who have already taken the class to teach and lead games on this lab day. If it is financially viable, you could also consider playing digital

4. You can also follow this link to the Google Sheet: http://prototools.net/r/TeamShuffler .

5. This was done in the excellent Building Virtual Worlds class that I took from Dr. Randy Pausch and Jesse Schell at Carnegie Mellon in 2005. In that class, we were working on virtual reality projects, and the roles were Modeler, Programmer, Sound Designer, and Texturer.

6. Logan has since worked as a Senior Experience Designer at Walt Disney Imagineering, independently created SoundStage VR (https://github.com/googlearchive/soundstagevr), and most recently worked for Google.

tablet versions of board games. Many games like *Carcassonne* and *Castles of Mad King Ludwig* are considerably easier to learn in their digital versions and are relatively cheap on iOS and Android. Look under "Appendix D" on the book website for a list of games that I think are good for this lab.

Paper Game Projects

The calendar includes two paper game projects, where students make board or card games. Each of these is run the same way with the same parameters. Student teams are tasked with designing a game for three to four players that takes 20 minutes or less to play. The kickoff for each paper game project should involve introducing the teams to each other and then guiding them through a brainstorming exercise (for one example, see the "Brainstorming and Ideation" section of Chapter 7, "Acting Like a Designer"). After the initial kickoff, all work on these games is done outside of class, with lab time reserved exclusively for multiple rounds of playtesting. Students must have a *fully playable game* by the time the first playtest lab starts, and they usually learn in the first playtest that many things need to be changed and improved. By the second playtest, they should have written rules ready, which will also be iterated upon. The final turn-in for the paper game projects is a fully playable game with rules and all elements required to play it (e.g., dice, game board, and so on). I usually ask for both digital files (as PDFs) and a physical copy. If the students are willing, I like adding their final physical copies to a library of games that students in future semesters can check out.

Recommendations for Game Rules

Students often have a difficult time getting the rules for their games down on paper, which isn't surprising, considering how difficult it is for even professional game companies to write good rules. Here are a few things that I ask my students to keep in mind:

- **Theme first:** The first paragraph of the rules should be a simple logline for the game. For example:

  > *In **Monkey Mayhem**, players take the role of monkeys at a Costa Rican resort trying to steal as much food from humans as possible before lunch is over.*

- **Objective next:** The second paragraph should be the objective, clearly expressing how players win the game.

  > ***Objective**: At the end of the game, the player with the most food points wins. Points can be earned by stealing food, with higher-value food items and riskier behavior worth more points...as long as you don't get caught!*

- **Use pictures:** Rules are always made better by pictures of the game and gameplay. One of the most important pictures in the rules is the image of the initial layout of the board, cards, scoring track, etc. For the first version of this, students can just set up their game, take a photo before the first turn, and add a few labels in an image editor. Including pictures of the game in progress is also very useful.

- **Use diagrams:** In addition to pictures, diagram images can also be used to explain any mechanics that are more easily shown than described with words. For example, the scoring of farmers in *Carcassonne* is complicated and has several edge cases, so the rules for *Carcassonne* include a diagram of a map with several farmers on it, shaded with colors to show which fields are owned by each player. Of course, there is text to explain the diagram, but without the diagram, farmer scoring would be much more difficult to understand. You can see another diagram example in Figure 33.2 of Chapter 33, "*Prospector Solitaire* — Part 1," where I both show the layout of the game and explain two potential runs of cards that the player could choose. Anywhere that it might help, students should use diagrams.
- **Examples of play:** In addition to describing game mechanics, it also helps to have students give gameplay examples. In these examples, the reader is given a specific gameplay example and shown how it plays out. These are usually something like:

  > ***Example*:** *On his turn, Adam moves to the same table as Heidi and Ozzie. Instead of taking a basic Steal Action, he plays the Tag Team Takeout card from his hand. The text on Tag Team Takeout reads "Action: When you play this card, if you make a successful Steal Roll, ALL monkeys at your table gain the results of your roll, and then you gain duplicates of any items they gained this turn." Adam makes a successful Steal Roll for five Sugar Packets, causing him, Heidi, and Ozzie to each gain five Sugar Packets from the bank. Then Adam also gains two Watermelon Slices and one Pie from the bank because on their turns Heidi had gained two Watermelon Slices and Ozzie had gained one Pie.*

  Gameplay examples like this give designers the opportunity to show how the rules come together in an actual round of play and can help players understand the rules better.

If your students include all of these, their rules will be much better for it.

Roleplaying Game Project

Nothing teaches adaptability in design like running a pen-and-paper roleplaying game. You can look at the "Pen-and-Paper Role-Playing Games" section of Appendix B, "Useful Concepts" for my recommendations on how students should approach them. In terms of the assignment, I try to make groups of four where at least one person in the group has played a paper roleplaying game before.[7] I ask the experienced student to be the first to run their game for their peers.

To kick off the RPG Project, I usually bring in student volunteers who have taken the class before and run through the first scene of an RPG scenario that I developed with

7. If you use the team shuffler from Table D.2, you can ask who has ever played a paper RPG in a poll at the beginning of class and then make that one of your "roles" for randomizing the teams. If the first member of each Project 1 group has prior RPG experience, then when you get to the RPG Project, the first member of each of those groups will have prior RPG experience.

me running the game as the Game Master (GM). After the first scene, I and each of the student volunteers each take one of the RPG project groups and run the rest of the scenario with the members of the project groups each taking on one of the roles. At the end of the lab, I distribute a PDF of the scenario to all the students in the lab. This gives each student at least a little experience playing through a game using the FATE Accelerated[8] system (or the Simple FATE[9] system that I built from it), and it allows them to see an example of what I'm expecting from their project turn-in.

For this project, each student will create and GM their own scenario, with each member of the group turning in their own project. Each member of the team is required to play in all the other members' games, and if they don't, their own grade is docked significantly. Each scenario should be designed to be completed in 1–1.5 hours, and each of the labs during this project are devoted to running a single scenario (leaving two scenarios that must be done by the students outside of lab). The final turn-in for each student includes:

- The scenario for their adventure (i.e., the setup for the adventure that they read to the players at the beginning of the play session)
- Character sheets that they created for each member of the team
- Any materials that they prepared ahead of time to make running the game easier (e.g., rough maps of various locations, a list of potential NPC (Non-Player Character) names, simplified character sheets for potential enemies)
- A brief retelling of the story created through the scenario
- A post-mortem write-up analyzing how well they prepared, how the game went, and what they might do differently if they run a game again

Pitch Project

The final project in this class is traditionally a live pitch in front of a panel of judges from the game industry. In early iterations, I allowed the student groups to pitch completely new games, but this led to a number of ridiculously overscoped pitches. What has worked best for me is to ask the students to create a 4–5 minute pitch of an expansion to an existing digital game (i.e., a DLC (Downloadable Content) pack) that somehow significantly changes the gameplay. A good real-world example of this is the *Undead Nightmare* DLC for *Red Dead Redemption* that dramatically transformed the gameplay into a zombie survival game. One of my favorite student pitches was *Little Big Planet: Hocus Pocus*, by Jay Cramer, Brandon Durgin, Sam Farmer, and Asher Vollmer. This

8. FATE Accelerated is a simplified role-playing game system by Simple Hat Productions. https://www.evilhat.com/home/fae/ — accessed March 11, 2022.
9. Simple FATE is a further-simplified role-paying game system that I actually use in my classes. You can find documents for it under "Appendix D" on the http://book.prototools.net website.

expansion to the original *Little Big Planet* game gave players a magic wand tool that they could use to pause the game and then move and rotate specific objects to solve physics puzzles. This wand used the same cursor as the "Popit Menu" that players can use to make new levels in *Little Big Planet*, so the puzzles of this pitched DLC would teach players how to use the level-creation tools in the original game.

As with any other project, the Pitch Project is focused on playtesting and iteration. In the last two weeks before the final, labs are spent with everyone watching each team give their pitch and then giving the teams feedback. During the final exam time, guest judges are encouraged to give feedback and ask questions, and there is time in between each pitch for them (and me) to fill out a form giving the students a 1–5 score on various aspects of the pitch as well as written feedback. It's a great way to wrap up the semester and is usually a lot of fun for everyone.

Teaching Introduction to Game Programming

I have taught various iterations of this class under several different names for nearly two decades now. In fact, my initial goal in proposing this book was to offload much of the lecture content of this class to the book so that I could focus on live demos and student work during class time. I still teach this class every semester, and I've found a way to do so that works very well for new programmers (and for computer science students looking to get into game programming) while still keeping the individual classes interesting for me to teach.

Sample Class Calendar

In Table D.3, you can see an example calendar of how I approach teaching this class. I have always taught this class with roughly 4 hours of in-class time per week, split into two classes each week. I don't distinguish between lecture and lab in this class, and all classes take place in a computer lab with a nice projection screen. I record all of my classes and post them online so that students can follow any live demos I do as tutorials later.

You'll see that in the first half of class, students are spending most of their time outside of class working through the book and most of the in-class time is me doing live coding demos. I have found this works very well for teaching the students concepts through their readings and reinforcing those concepts by showing them my process as I code simple game prototypes. The second half of class focuses on Coding Challenges and the Final Game Project.

Table D.3 Sample Class Calendar for Introduction to Game Programming

Week	In-Class (4 hrs/wk)	Readings
1	Live Demo: Hello World in Bolt Show them how to create a Unity Account	Syllabus
2	Set up Unity version control Live Demo: *Apple Picker* in Bolt Assignment Due: Hello World in Bolt (from lecture video)	16: Thinking in Digital Systems 17: Introducing Unity Hub and the Unity Editor 18: Introducing Our Language: C# 19: Hello World: Your First Program Appendix A: Standard Project Setup Procedure 29: *Apple Picker*
3	Live Demo: (*Asteroids* clone) Assignments Due: Hello World (Chapter 19) *Apple Picker* (Ch 29)	20: Variables and Components 21: Boolean Operations and Conditionals 22: Loops 23: Collections 30: *Mission Demolition*
4	Live Demo: (First-person controller) Assignment Due: *Mission Demolition* (Ch 30)	24: Functions and Parameters 25: Debugging 31: *Space SHMUP* — Part 1 Appendix C: Online Reference
5	Live Demo: (Interpolation and Bézier Curves) Assignment Due: *Space SHMUP* — Part 1 (Ch 31)	26: Classes 27: Object-Oriented Thinking 32: *Space SHMUP* — Part 2
6	Live Demo: (2D Platformer) Assignment Due: *Space SHMUP* — Part 2 (Ch 32)	33: *Prospector Solitaire* — Part 1 37: Coding Challenges
7	Coding Challenge #1 (in pairs) Assignments Due: *Prospector Solitaire* — Part 1 (Ch 33) Coding Challenge #1	34: *Prospector Solitaire* — Part 2
8	Coding Challenge #2 (in pairs) Assignments Due: *Prospector Solitaire* — Part 2 (Ch 34) Coding Challenge #2	35: *Dungeon Delver* — Part 1
9	Final Game Kickoff/Brainstorming Students pitch Final Game ideas Assignments Due: *Dungeon Delver* — Part 1 (Ch 35)	36: *Dungeon Delver* — Part 2
10	Final Game: Teams of Two Locked Assignments Due: *Dungeon Delver* — Part 2 (Ch 36)	28: Data-Oriented Design

Week	In-Class (4 hrs/wk)	Readings
11	In-class work days and check-ins	
12	Final Game: Playable Prototype presentations	
13	In-class work days and check-ins	
14	Final Game: Alpha Presentations	
15	In-class work days and check-ins	
Final	Final Game: Presented during Final Exam Assignments Due: Final Game Project	

Core Learning Goals of the Class

I want to get across several things in an intro game programming class:

1. **Programming is challenging but possible to learn:** Students need to learn a lot to be able to program games in Unity, but I've structured this book and class to help them take it step-by-step. I often tell them that learning to code is like learning a new language (e.g., Spanish, Mandarin) while also doing logic puzzles at the same time. They shouldn't expect to understand things at the beginning.
2. **They are learning more than they realize:** I always mention this several times early in the semester, but most students in the class feel that they aren't understanding the code and are just going through the motions of the tutorials for the first half of class. Often, week 6 is their low point, where they don't feel like they've learned anything. Then they start tackling the Coding Challenges with a partner, and as they do, they realize that they've started to internalize and understand all the code that they've been copying.
3. **Game programming isn't about knowing everything; it is about knowing how to figure things out and learn new concepts:** Before teaching Unity programming, most of my classes were teaching Flash ActionScript. Strangely enough, even though it was easier to make games in Flash, I found that the first semester I switched to Unity, the students were much better at figuring things out for themselves. I then realized that watching me stumble and struggle with Unity taught the students how they, too, could find solutions to coding problems. Now, any time I do live demos, I not only show (intentional) mistakes and how to correct them, but also try to do new live demos every semester so that students can see me actually approach a coding problem iteratively and with real bugs.
4. **Project management matters:** I require that students use a burndown chart on their Final Game project for two reasons: 1) it allows me to see how much work each

student of the pair did on the project, and 2) in my experience, using a burndown chart greatly improves the students' understanding of their project scope and where they are in finishing it. I spoke about this at the 2014 Game Developers Conference, and you can watch my talk "Implementing Agile and Scrum in the Classroom" on the GDCVault website for much more info.[10]

Elements of the Class

Let's look at a few of the core elements of the class.

Assignments from Chapters in Part III of This Book

In general, successfully completing the tutorial in a chapter is worth 85% credit on the assignment. For the additional 15%, I ask students to create *enhancements* for the project, which are improvements or changes that they design and implement themselves. Early in the semester, an enhancement could be as simple as adding a title scene with a Start button, and I increase my expectations for enhancements over the course of the first half of the semester. If I require specific enhancements to a project (e.g., the gold and poison apples I mention in the "Bolt/Unity Visual Scripting" section that follows), then completing the tutorial is worth 70%, the required enhancements are 15%, and their individual enhancements are worth 15%.

I originally required students to turn in both their Unity Project folder (compressed as a zip) and a WebGL build (also compressed). However, in more recent semesters, I instead require them to turn in a WebGL build and to add me to their version control (e.g., Plastic SCM) as a manager. As a manager on their version control, I have access to all their projects, which is extremely useful when I need to help them fix a bug during office hours or when they have posted a question to the class forum that requires me to look at their project to help them.

They are also required to turn in a ReadMe.txt file. This file must be both in their Unity Project and in their WebGL build folder and must include

- A description of their project enhancements and how to find them
- A list of any assets they used that they didn't create themselves, including links to where they found those assets
- A description of any help they received on the assignment from anyone who is not an instructor (or TA) of the class (including their names and what they helped with)
- Links to any online websites, videos, or tutorials that they used to help them on this assignment

10. https://www.gdcvault.com/play/1020769/Implementing-Agile-and-Scrum-in — accessed September 24, 2021.

- A description of any trouble they had with the project
- Anything else they would like us to know or to consider while grading

tip

GRADING MULTIPLE STUDENTS' WEBGL BUILDS Due to security concerns, it is now no longer possible to view local WebGL builds in your browser by double-clicking the index.html file saved on your computer. Students can circumvent this to test their own WebGL builds by choosing Build and Run in Unity, which creates a temporary web server to host their WebGL build, making it possible for browsers to run it. However, that method would require you to recompile every student's WebGL project on your own machine, taking at minimum 15 minutes per project to download, open, and build the Unity project on your computer.

Happily, there is another way to view all of their WebGL projects on your local machine. Follow the directions to create a temporary Python web server that are in the "Running a Temporary Local Web Server" section of Appendix A. However, rather than running the server on a single WebGL build folder, place all of your students' WebGL build folders within the same parent folder and run the web server on that parent folder. As long as there is no file named index.html inside the parent folder, the http://localhost:8000 link will show a list of all the folders contained within the parent (this is the automatic "index" that index.html is named after). You can then click each folder to view that student's WebGL build.

Live Demos

Live demos help the students see the Unity development process in a different way than the book tutorials allow, and this can work well to reinforce the book information. I record all of my live demos and post the videos so that students can follow them on their own. In the best-case scenario, I record a live demo for the first half of class without the students following along (but with them asking questions), post the video, and then ask the students to start following the demo on their own machines in the second half of class (giving them the equivalent of a quiz grade for turning in progress at the end of the class). In reality, I more often do the live demo for most of the class while answering their questions and then post the video as a tutorial that they can choose to follow later on their own time (including after the class has completed).

In the example calendar, I listed some good topics for live demos that increase in difficulty through the calendar, but I also tend to ask students what they're interested in and cater to that. I've found live demos to also be a great way to demonstrate difficult concepts like coroutines, Bézier curves, and component-oriented design.

Bolt/Unity Visual Scripting

I tend to still use the old name, Bolt, for visual scripting in Unity because in Unity 2020.3 LTS, you must get Bolt from the Asset Store in order to use it (Unity Visual Scripting was first fully integrated into Unity 2021). It would be incredibly difficult to write a Bolt tutorial in a book, but I do use Bolt in my classes with good success. Students are sometimes annoyed at having to "make the same project twice" when I ask them to do the Hello World and *Apple Picker* projects in both Bolt and C#, but I have found that it's easier for new programmers to see what's happening in Bolt code by following the code execution as it flows visually through the Bolt graph as little circular pulses. I often also create the *Asteroids* live demo in both Bolt and C# (though even in the Bolt version, I manage screen-wrapping using a C# script). Another thing I like to do is to implement a couple enhancements to *Apple Picker* (usually gold and poison apples) in the Bolt version of the project as part of my lecture video and then ask them to create gold and poison apples using C#, which gives them some experience translating from Bolt to C#.

Quizzes

In general, I don't like quizzes much, but I have found that they are an excellent way to get students to actually do the reading. I usually give a syllabus quiz in the second week and then give reading quizzes in weeks 3–6. I avoid nitpicky details from the reading and instead try to reinforce general concepts with my questions. This helps me gauge how well the students are understanding the book content and lets me know what I need to review with them.

Coding Challenges

Chapter 37, "Coding Challenges," covers these Coding Challenges in detail, but from a pedagogical standpoint, I have found that having students work in pairs on these challenges consistently improves their confidence in both themselves and their knowledge of game programming. This gives them a great place to start from for the final game project.

Final Game Project

The last several weeks of class are devoted to the Final Game Project. Students can either work in pairs or solo on this project, and I require them to track their progress using the burndown chart that is introduced in Chapter 14, "The Agile Mentality." One of the toughest aspects of this project is helping the students decide on a project that is the right scope for the amount of time they have to spend on it. Most often, students choose a project that is much more than they could accomplish in the little time that they have, so I encourage them to start small with a concept that they can extend if they have time.

Classic Game Project

In past semesters, I have not used some of the book tutorials in my class and have instead devoted four weeks to a Classic Game Project and then only four weeks to the Final Game project. In the Classic Game Project, students choose a game from the NES era and attempt to recreate the mechanics as faithfully as possible in Unity. This allows the students to focus entirely on programming and implementation without also worrying about designing a game at the same time. One major caveat is that students will very often spend far too much time on recreating the graphics of these classic games. Even when I tell them that graphics are worth only 5% of their grade on the project and that I would be perfectly happy with a capsule jumping around as long as the physics of the jump were exactly like Samus in the original *Metroid*, they still often spend too much time on graphics.

In my current iteration of the class, I give students the option of creating their own new game or recreating a classic game for their Final Game Project.

More Information Is Available

I've collected more information about the classes I teach as well as some tools that I use on the website for this book. Look for "Appendix D" on the http://book.prototools.net website. As I mentioned at the top of this chapter, there is also a contact form on the book's website where you can tell me about how you use this book in your classes. I would love to learn more about where and how it is being used. Thank you for teaching from this book and for being someone who shares knowledge with others.