Re-Imagining Early GameMaker Assignments in Game Design Studio 1

Jeremy Miles Johnson, Ph.D.

Mini-Portfolio for Self-Evaluation

May 2023

Assignment Reviewed from Game Design Studio 1 VGAM 3330

Abstract:

To teach game design and introductory GameMaker Studio skills early in Game Design Studio 1, I created a series of assignments where students would first analyze two simple games and then follow tutorials to make them. For the final phase of the project, students would combine those two games to create something original and solve the resulting game design problems. One advantage of this is that the design problems that the students are tasked to solve present themselves naturally over the course of determining how to make these two games might blend together. Another advantage is that the students are using with two projects that they've created as the base, so that should help take this big step. The results from the fall of 2022 were not what I wanted, but there is still hope.

The Problems

GameMaker is the first professional-quality game engine the students are introduced to. This happens at the beginning of their junior year. Up to this point, the students have built board games and have had some exposure to creating very simple video games. For example, Robert Denton Bryant has given a game design assignment in History of Video games where the students create a game in Scratch, a simplified game engine. Students also tend to make at least one text game in Beginning Coding for Non-Coders in Python, though not through the use of any game engine. It is in Game Design Studio 1 where students are first introduced to working in a full-featured game engine, and because this occurs in the second half of their degree, there is a real urgency around getting them to a place where they can create original games as soon as possible.

Like many of the courses I teach, there are multiple, sometimes conflicting goals for Game Design Studio 1. One fundamental goal is that students spend time thinking about the *design* of two-dimensional (2D) games. The other fundamental goal is that students learn to create original games by doing just that, creating games. The emphasis of this course is not on how to code well or how to overcome all the fiddly technical hurdles that any game engine presents. However, I must spend a non-trivial amount of time teaching many of those technical details so that the students develop skills and knowledge sufficient to create games. Only when they have a working game can they get feedback, analyze their work, reevaluate their decisions, and improve their design skills.

To help students become comfortable with GameMaker, I have used a series of tutorials of some kind. When I have assigned tutorials made by others, I've found that student enthusiasm tends to be lower than if I create the tutorials. Some students won't do the assignments at all, and then do not develop the fundamental skills needed to progress in the class. Another problem is that unless I vet those tutorials *extremely* carefully, they may teach

bad practices or choose a structure that cannot be iterated upon. Even when I take the care to work through those tutorials, they may no longer be viable the next year. Regardless of their technical quality, tutorials available on the internet generally provide step-by-step instruction into the 'how' of making a game, but not the 'why' that goes into each step.

When I provide students with tutorials I have made, student engagement tends to be much higher, but creating those tutorials is arduous. I have estimated that between planning, doing the work, debugging the process, redoing the work on camera, and then editing video, it takes me about 5 hours of work to put together each hour of a game development tutorial. I was crafting between 6 and 10 hours of video tutorials each semester for this class, which amounts to an additional full week of work. In the same semester, I tend to have to do about this much tutorial creation in Advanced Animation each semester. To make matters worse, game engines change significantly from year-to-year, which frequently requires that I replace all those tutorials within a year or two. So while students do tend to appreciate those efforts, they don't feel sustainable for me.

I noticed another issue as well. When a student encounters a problem while following an online tutorial, they seem much more inclined to try to solve the problem than when they encounter a problem with a tutorial that I have made. There is an expectation that if I provide a tutorial, it is self-contained and will work perfectly.

A concern I have with learning by following a tutorial is that video game development tutorials tend to resemble recipes. When you follow one exactly, you should get the desired product, but unless you really spend time delving into the 'why?' of each step, at best you learn how to make that one thing in that exact way. More likely, however, little-to-none of the information gets converted into usable knowledge. Forget about being able to create something original; you likely cannot even create that exact thing again without once again following the recipe.

While the students are following tutorials in order to learn the tools, they are doing virtually no design work of their own. Design requires a problem to solve, and it's often a messy process. I have been working on a game lately, and I recently made and remade a maze navigation system four separate times. Each time it worked, but presented a problem that I had to go back and address, which required iteration. When you don't know the problems you are solving, you aren't able to design. In fact, most people who make tutorials will secretly do most of the iteration behind the scenes and present an efficient solution where you take a series of careful steps directly to a desirable outcome without ever knowing the true process that determined why that was the desired outcome. And learner's reinforce this. Most will eschew a messy four-hour-long tutorial that includes all the necessary false starts and revisions in favor of a snappy thirty-minute tutorial that gets right to the end product flawlessly. Even with these severe drawbacks, virtually all education in game development involves following tutorials in the beginning. Despite repeatedly tweaking the early tutorial assignments to include more of the 'why' and show more of the chaos, I wanted to find a better solution.

The Original Assignments

It is important to build a student's familiarity with the game engine and to help them feel that making a game is something that they can do. I try to give them an early win. Some semesters I have done this by having them make a game in an extremely limited game engine called Bitsy. I have also put them right into GameMaker Studio but with a very simple tutorial. This year (fall of 2022), I started with a Bitsy assignment again, as I did in 2020. This eases them into the class and gives us a little breathing room to learn some principles before we have to start learning technical details of a game engine. I mention this now, because by the end of this mini-portfolio I will decide to eliminate the Bitsy assignment once again in favor of the simple GameMaker Studio tutorial.

The Crux(es) of the Problem

Students need to follow an example of some kind. The majority of students would like a thorough tutorial that will walk them through every step of creation of a game. A few students who love programming or are already familiar with GameMaker may find following a meticulous tutorial to be a slog. It would be nice if there were the possibility for meaningful creativity to be found in some kind of tutorial assignment.

Students also don't tend to do extra work while following a tutorial. Few take notes or write their own comments in the code. This means they are less likely to grasp what is going on while they work, or if they do, they're unlikely to retain that information. Furthermore, if they revisit their old code and it hasn't been commented thoroughly, it will probably be of little use. If the assignment explicitly requires the student to reuse this exact code, they may spend more time putting comments in their code and trying to understand what they're doing as they do it.

Possible Solution: A Mash-up of Two Classic Games

The solution I came up with was to have students choose one simple, classic game from a list that I provided. They would play the game and analyze it carefully. Then they would follow a tutorial to make that game as faithfully as they could.

They would then repeat this process for a second game on the list.

Finally, the students would "mash them up" to make a third game with properties of both. My intention is that the students would have completed two games knowing that they'll need to revisit them at the end. Since they would be explicitly told that they would be revisiting and reusing this exact code, they would have a real incentive to comment the code and learn what they are doing at a deeper level while they follow the tutorials.

The mash-up would provide some space for original game design work. That work would involve determining what gets included from each of the games, what gets left out, and how

exactly to solve all those interstitial design problems that will arise naturally from trying to combine two games.

As an example, if you were to mash Asteroids and Snake together, you have to answer a number of important design questions:

- Are you still on a grid like in Snake, or if not will it be too hard not to crash into yourself?
- Do we even still need to keep the idea of crashing into yourself if there are asteroids and ships shooting at you? Maybe just growing longer is enough.
- Does the screen wrap like in Asteroids, or are there deadly walls like in Snake?
- What makes the player avatar grow longer? Do you collect or "eat" something to grow?
- Can you shoot? If so, can you only shoot enemies and asteroids, or can you accidentally shoot your "food"? If you do, what does that do?

Two students who choose the same game tutorials might answer these questions differently and end up with radically different final mash-ups. That, to me, is a sign that the outcome of this initial get-to-know-GameMaker project is also a true design project.

Once those questions are answered, students must revisit their tutorial games, decide whether to start from a base of one of the games or to start from scratch. Then they have to get these two games to work together.

The Games

For each assigned classic game, provided links to gameplay footage of the original game, a link to play an emulated version of the game, and to one or more GameMaker tutorials that show students how to create some or all of the game. In some cases not all of these were available for every game, and there were a few that had no associated tutorials. I advised students to avoid those unless they already had experience with GameMaker Studio.

The list of games and the provided resources:

1. Space Invaders - gameplay video and online play
2. Asteroids - gameplay video, tutorial, and online play
3. Snake - gameplay video, tutorial, and online play
4. Flappy Bird - gameplay video, partial tutorial, and online play
5. Breakout - gameplay video and two tutorials
6. Pac-Man - gameplay video, partial tutorial, and online play
7. Frogger - gameplay video, tutorial, and online play
8. Centipede - gameplay video, development process video (I created this tutorial)
9. Maze Craze - gameplay video, partial tutorial, online play

Design Considerations Even In the Tutorials

In this scenario, students know from the beginning that they are going to have to make design decisions about their mash-up. That gives them incentive to think seriously about the design of each individual game. In this way, even when just following a tutorial, students have additional motivation to think like a game designer. They evaluate core game loops, look for patterns, examine the 'why' behind each game design choice.

This would prepare students to think about how those core loops interact with one another. Do they conflict or enhance each other? Are they so close that the result actually doesn't feel original at all? This can give the student a series of design questions to explore as they create something original from two thoroughly unoriginal games. This also helps students to see that creativity is often nothing more than putting two things together in a new or unusual way.

Limitation to Match Original Game

By using well-known classic games and requiring students to reproduce them as accurately as possible, they were required to study those games closely. The burden of "the blank page" is lifted from them. If they don't know what to do, they have only to go study what the original game did. However, they gain an additional burden in that if they are struggling to get something working, they can't just fudge any old solution. They need to solve the problem. This gives them concrete goals by which to measure their success and to help drive them forward.

Approachable Creativity

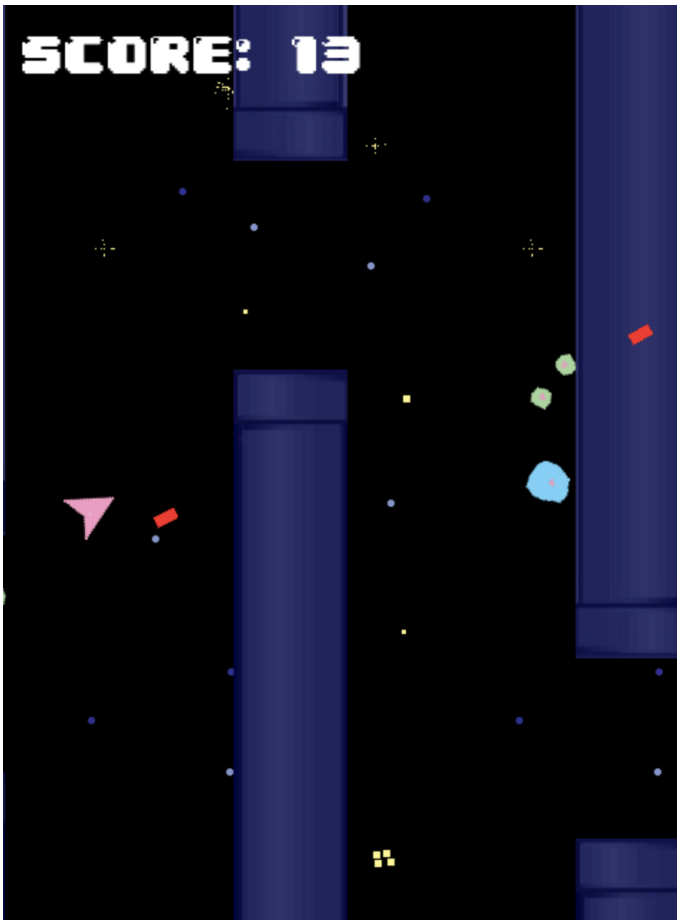
I wanted students to see that, generally speaking, originality and creativity arise from: putting two things together in a novel way. Even when putting two extremely simple games together dozens of interesting design problems can present themselves. In this way, they never struggle with "the blank page" problem. They are given a series of tasks that are extremely constrained and yet lead readily to the creation of a unique game. It can also lead to some silliness, which is always appreciated by the class.

The Outcome

As much as I want to tell you that the experiment went smoothly and everything worked out well, it did not. Of the 16 students in the class, 4 failed to submit anything for the final combined game. Another 8 students submitted a game that was only a slight variation on one of the original two games. There were 4 who mashed up their games properly, and of those, 2 created brilliantly clever final games.

Isabel Rasmussen created a game that was a mashup of Asteroids and Snake. In it, you flew around dodging and shooting as one does in Asteroids. There were also food pellets to

collect, and as you collected those your ship grew longer. That made it much harder to dodge asteroids.



Aaliyah Hampton made a mashup of Flappy Bird and Asteroids in which your ship flies along horizontally dodging pipes and "flapping" upward when you click the mouse. There's actually no flapping involved, but you do get a burst of upward momentum and rotation, like in flappy bird. When you click, your ship also shoots, which is good because there are asteroids coming at you amidst the pipes. You can play *Flappy Space* here:

https://stedsvgam.com/visu/games/aaliyah_hampton/

So What Went Wrong?

Several factors contributed to a lackluster outcome for this series of assignments. The first is that not all tutorials are created equal, especially for the simpler games. Some tutorial creators, particularly those who made tutorials for Snake, realized that one could completely ignore GameMaker best practices and make a version of Snake that used none of the built-in systems. While this works if your end goal is simply to reproduce Snake as efficiently as possible, it made for a version of the game that could not actually be mashed-up with another. In fact, Isabel Rasmussen was only able to include Snake in her mashup because she spent hours in my office sorting out the details. In effect, I became her revised Snake tutorial. The Snake

tutorial was the most egregious example of this, but wasn't the only game tutorial to do it. Unfortunately, the Snake tutorial was the shortest one, so it was taken up by more than half the class. Though I tried to vet the tutorials, I didn't catch this problem. Also several students found their own tutorials, something that I had initially encouraged, and many discovered similar problems.

Secondly, not all simple games are comparably simple. Flappy Bird, Snake, Breakout, and Asteroids have very few objects and systems. Frogger, Pac-Man, and Centipede are probably an order of magnitude more difficult and time consuming. For example, frogger has several different zones of the screen that behave differently. At the bottom there are cars, those cars have patterns. When you get to the river, jumping in the water is deadly. Some logs disappear, some are alligators, etc. I discovered this while making a shortened Centipede tutorial for the assignment. Virtually nobody chose any of those options, because the number and length of the tutorials made the complexity differential clear. The very few who picked the more complicated games anyway struggled to complete those tutorials in time, and faced a particularly daunting challenge in mashing up something even so very modestly complex as Space Invaders with anything else.

Moving Forward

The experiment wasn't a complete disaster. Many students were not properly prepared to execute the mash-up part of the project when they got to it, but they did still follow two complete tutorials and did some design thinking in anticipation of the mash-up. Most students attempted the final stage of the project and had something to present in class. Because of these things, I surmise that this project was at least as effective as two unrelated tutorials at teaching skills with the game engine. Furthermore, it presented the students with design challenges to contemplate, which is rare when following tutorials.

Based on the work of those students who were able to successfully complete the final mash-up, I still think this could be a fun and effective assignment with the following changes:

1. Eliminate the Bitsy game and provide a simple initial GameMaker assignment to build their familiarity with the game engine a bit more before expecting this of them.
2. Reduce the selection of games to only the simplest ones so that all of the options are closer in difficulty.
3. With fewer choices, I can take time to vet the available tutorials thoroughly to make sure I'm not setting students up for an impossible task.
4. If there is no suitable choice, I can consider making a tutorial myself. I know that this puts me right back on that slippery slope, but surely this time the tutorial will work forever…

Conclusion

I didn't get the results I was hoping for. I think this was due to a number of errors on my part. I will attempt a revised version of the assignment with fewer games to choose from. I will select only the simplest games, and I will vet the tutorials more thoroughly to ensure that they will result in a code base that is compatible with the other games on the list. Furthermore, I will limit the students to specific tutorials that I include. If necessary, I may make tutorial content to cover games for which no suitable tutorial exists. I also plan to remove the Bitsy assignment and

replace it with an assignment in GameMaker that is designed to introduce students to the basics of the game engine. The results demonstrated to me that the students required more scaffolding to help them develop their knowledge of GameMaker before beginning this series of assignments. Fortunately, I already have a space for that assignment in the schedule.