Re-Re-Imagining Early GameMaker Assignments in Game Design Studio 1

Jeremy Miles Johnson, Ph.D.

Mini-Portfolio for Self-Evaluation

May 2024

Assignment Revisited from Previous Mini-Portfolio

Abstract:

For the fall semester of 2022, I changed the introductory assignments in Game Design Studio 1 to address a specific problem: students were essentially spending the first half of the class following tutorials, and as a result, they were doing very little design work.

In order to make games, students in Game Design Studio 1 must first acquire sufficient technical skill in the game engine, GameMaker Studio. Unfortunately, the previous methods by which the students were acquiring those skills prevented them from engaging with the design process for far too long. In the fall of 2022, I created a new series of assignments that I had hoped would address this: The students would spend the first and second weeks of class following two tutorials and creating two simple games. For the third and fourth weeks of the class, they would create an original game by “mashing up” the two games they had already made into a third, original game. The first two weeks of tutorials would give students experience in the game engine and provide two games worth of code from which to build. Mashing the games up would provide a modest, but unique collection of design problems for each student to solve. That was my intention, but the outcomes weren’t great. Students were not able to create a unique mashup in the spirit of the assignment. I documented this in my previous Mini-Portfolio, “Re-Imagining Early GameMaker Assignments in Game Design Studio 1.”

This year, in the fall of 2023, I created one additional series of tutorials designed to boost the students' core understanding of how GameMaker works. The students first completed these tutorials before engaging with the aforementioned tutorial-tutorial-mashup assignment sequence. The resulting mashup games were a showcase of well-executed, original designs. I presented the results at The Southwest Popular/American Culture Association (SWPACA) conference.

The Problems (Old and New)

Game Design Studio 1 is taught at the beginning of a typical Video Game Development major's junior year. Up to this point, the students have built board games and have had some exposure to creating video games in simple engines. But, they haven't yet used a professional-quality game engine nor tackled video game design challenges in a systematic way. I will refrain from too much discussion of the details of those problems, because they are thoroughly outlined in my 2023 Mini-Portfolio, but I will give a quick summary.

Professional-quality game engines are complicated pieces of software that require a high level of technical knowledge to operate. Additionally, the specific techniques of game development—for example, understanding what a state machine is, or how and when to create a system manager—are another required set of skills. Finally, solving the design problems of an original game requires a third, independent set of skills. It's possible (and not uncommon) to make a class that only teaches the first two skills. I would argue that this is the default for YouTube tutorials. But, Game Design Studio 1 is foremost a design class, and so I struggle to keep the students focused on design as their attention is pulled to the more immediate roadblock of gaining the requisite technical expertise that they'll need to do anything at all.

In 2022, I changed the first assignments in GameMaker so that the students followed a tutorial to create a simple game in a week. They repeated that for a different game the next week. For the following two weeks students were asked to iteratively create a mashup game using the two games they had already created. It did not go well. Few students had enough

technical skill by the end of those first two weeks to even take a modest initial step into creating their own unique combined game. Those who had done sufficient design thinking when planning out their mashup game mostly abandoned their clever ideas in favor of simply re-skinning one game with assets from the other. This, I think, was a response to how unprepared they felt to tackle the challenge of making something original in this way by the third week of the semester.

The crux of the problem, I think, is that there were three major things for these students to learn before they could execute on their own game design. The first is how the game engine works. You can learn and apply various skills in any game engine, but every game engine works differently, so you have to learn the particulars of how that engine operates. Secondly, GameMaker uses a proprietary programming language, GML, which is relatively simple to learn, but students must adapt what they know from their coding course(s). Finally, there is the problem of learning game development-specific best practices, tricks, hacks, and the like. As an example, there is the problem of how to make something fly forever in a space of finite dimension like in *Flappy Bird*. In that game, the bird is forever flying to the right and dodging pipes—so it appears, anyway. In reality the bird only moves up and down. It is the pipes that are moving from right-to-left, which give the illusion of the bird's motion. New pipes are created off-screen on the right, and destroyed after leaving the screen on the left. Game development is full of these kinds of specific solutions to common problems. Before they can design, students need to have a decent grasp of the game engine, the coding language, and a few of the relevant tricks. It would have been a good idea for me to provide at least three major assignments where students could focus on each one of these skills in turn before they were presented with the mashup. I had only provided two.

A second problem I discovered is that I left the scope of the tutorial content somewhat open. I wanted to provide a breadth of choice, and so I included games like *Asteroids*, *Snake*, *Flappy Bird*, *Frogger*, *Pac-Man*, *Centipede* and others. While all of those games are simple from a modern perspective, games like *Pac-Man* or *Frogger* are possibly an order-of-magnitude more complex than games like *Asteroids* or *Flappy Bird.*

Solution Part 1: An Additional Tutorial Series

It was clear that the students needed to spend more time gaining technical skills in GameMaker before they'd be able to tackle designing their own games. My challenges were: 1) to make a tutorial that focused on a technical skill in game development that most tutorials left out, 2) to avoid boring any exceptionally experienced programmers or developers in the class, and 3) to tie the tutorial into the rest of the curriculum. I had one additional concern, which was to keep from exhausting all my time making game tutorials only to have to make them again whenever GameMaker is updated significantly.

My solution was to make one simple, two-part tutorial assignment where the students would re-create a "game" (specifically game 2) from *Combat* for the Atari VCS/2600 as close to the original as possible. I did it in the following way: I created a written tutorial rather than a video one. In this way, I could revisit and edit it piecemeal as the game engine is updated. Previously, I would have to re-create a video tutorial that might have taken 40 hours of work because the software had changed enough to introduce significant complications. This happens

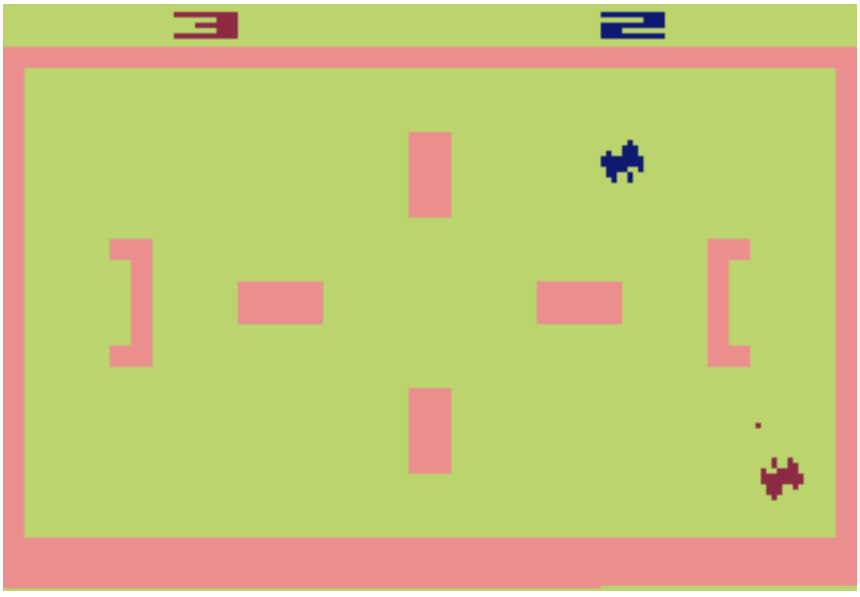


Figure 1 - Two tanks fight in the original *Combat* (game 2).

surprisingly often. I made this *Combat* tutorial to serve my own pedagogical needs as well. I wanted to help students avoid growing accustomed to just mindlessly following instructions by following a recipe. For example, I might I provide an example of code relating to getting input and moving in the x direction and then say, “Repeat that process with appropriate modifications for the y direction.” In this way, students get specific instruction with examples, but they must also digest and understand what I have presented so that they can modify and reuse it immediately.

We have an Atari VCS system in Game Lab with the original *Combat* cartridge. Having them re-create this game ties the assignment back to their first class in the major, History of Video Games, in which they played this *Combat* on this console. Having a physical example of the game in class also allows me to leave some specifics completely out of the tutorial and let the students experiment with the original game to figure things out like what a reasonable number for bullet speed would be.

After re-creating *Combat* game 2, each student would demonstrate what they had done to me. After a brief conversation, I would assign a second project, a different ‘game’ from *Combat.* There are 27 game modes, on the cartridge, so this assignment was chosen to uniquely suit each student. All games in *Combat* are similar in terms of the underlying code, but they can appear and play very differently. For students who breezed through the first tutorial, I would assign a second game that was significantly different from the first. Instead of a two player tank fight with guided missiles, they might find themselves creating a game with biplanes that never stop moving and who, whenever they leave the field of battle, reenter on the opposite side of the screen. For students who succeeded but who struggled, I might choose a different tank game that changed some significant aspects from the original. Some of those games include having bouncy bullets instead of guided ones, or tanks that are invisible except when shooting. Finally, for those students who tried, but were unable to complete the initial tutorial, the second game would be little changed from game 2. For example, game 3 is exactly like

game 2, but with a different configuration of obstacles on the field. With an assignment like this, students were effectively able to just keep working on the initial assignment.

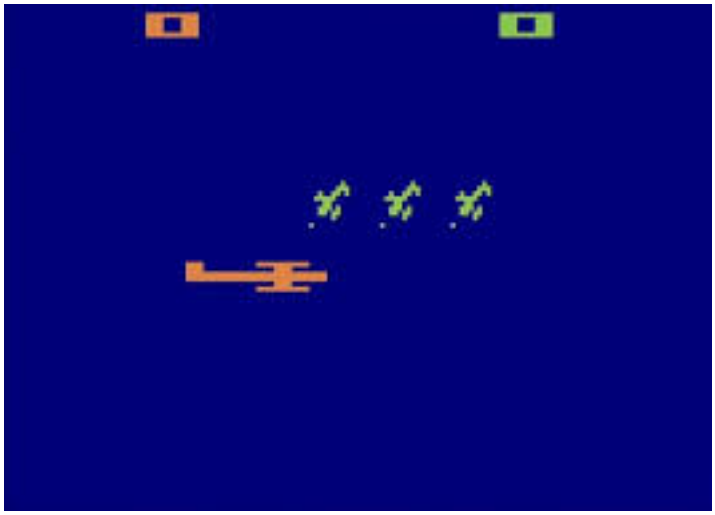


Figure 2 - Biplanes fight in a 3v1 pattern. This was only assigned to those who found the first tutorial very easy.

This was incredibly successful. I measured this in a few ways: 1) All students had focused for a week on thoroughly learning GameMaker, which showed in their ability to talk about its basic systems, 2) students were getting practice experimenting to find information on their own or with the help of their peers, and 3) the second assignment provided an appropriate challenge for each student despite their varying levels of aptitude with coding. There was only one student for whom this did not work well. The student in question struggled with the written tutorial due to severe dyslexia. This student did not submit an accommodation letter, so I had no way of knowing. While I won't return to making the long video tutorials, realizing that there could be students who struggle with anything that must be read, will help me watch out for this sort of thing happening in the future.

Ensuring that students understood the game engine was a solution to bridging that gap, but I still hadn't addressed the varying levels of tutorial complexity in the mash-up assignment.

Solution Part 2: Further Restricting the Allowed Mash-Up Games

In my second iteration of the mash-up assignment, I eliminated *Pac-Man*, *Frogger*, *Centipede*, and *Maze Craze* from the list of allowed games, leaving only *Space Invaders, Asteroids, Snake, Flappy Bird, and Breakout*. *Space Invaders* is slightly more complex than the others, but I warned students, and consequently only the most skilled students attempted to use it as one of their mash-up games. Because the students had already had some additional development practice in GameMaker, they had a more reasonable understanding of their own skill level, and could be trusted to make that choice.

The Outcome

In 2022, when I assigned the mashup without the *Combat* tutorial, only two students submitted projects that met the design goals of the mashup assignment. One did it by coming to office hours for help every day—a solution that is not scalable to the entire class—and the other did it by taking an incomplete in the course and then completing the assignment after all the rest of the classwork.

In 2023 however, I got the outcome I had been hoping for. Every student presented an original game that demonstrated a thoughtful design which attempted to solve the problems from mashing up two other games.

For the remainder of that semester of Game Design Studio 1, students seemed to tackle harder projects and would challenge themselves more than I had seen in previous classes. In the next course in the sequence, Game Design Studio 2, the students also seemed to struggle less with learning the Unity game engine when compared with previous years. It's hard to know if this collection of changes is solely responsible, but I do think that having a series of positive initial interactions with a professional game engine may have helped. This class seemed to to more easily embrace the myriad challenges that come with making a game and seemed less

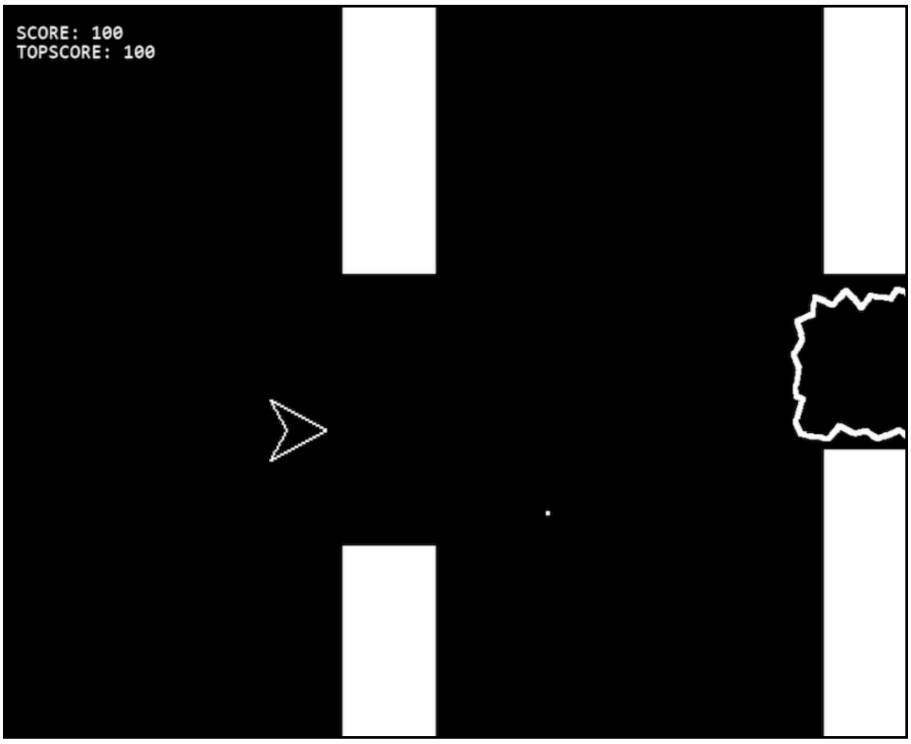
inclined than previous years to opt to sacrifice quality or originality when encountering difficulty. I have attached screenshots of most of the games in the Appendix at the end of this document.

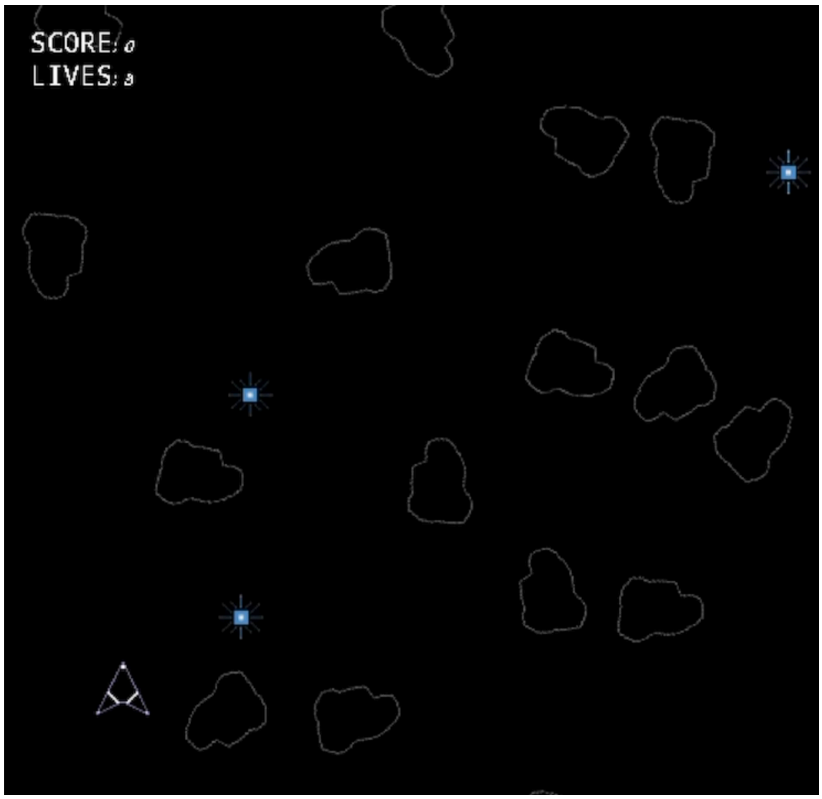
Conclusion

By adding two weeks of *Combat* tutorials at the start of Game Design Studio 1, I started seeing the results I have been wanting from this course. The additional series of tutorials on *Combat* did a number of services for this class. First of all, it gave them time to build confidence and expertise in GameMaker before having to tackle unique design problems, it tied the first game back to previous coursework explicitly, and though I haven't mentioned it previously, *Combat* is a two-player game, which demonstrated that it is something they can do. Additionally, I learned enough of value that I was able to present on my findings at the 2024 Southwest Popular/American Culture Association (SWPACA). I will continue to refine the tutorials as needed, but after this semester, I think the major work on this course is done for a while. Future students will follow this curriculum and I will endeavor to confirm that this structure and that this particular class wasn't just a statistical anomaly.

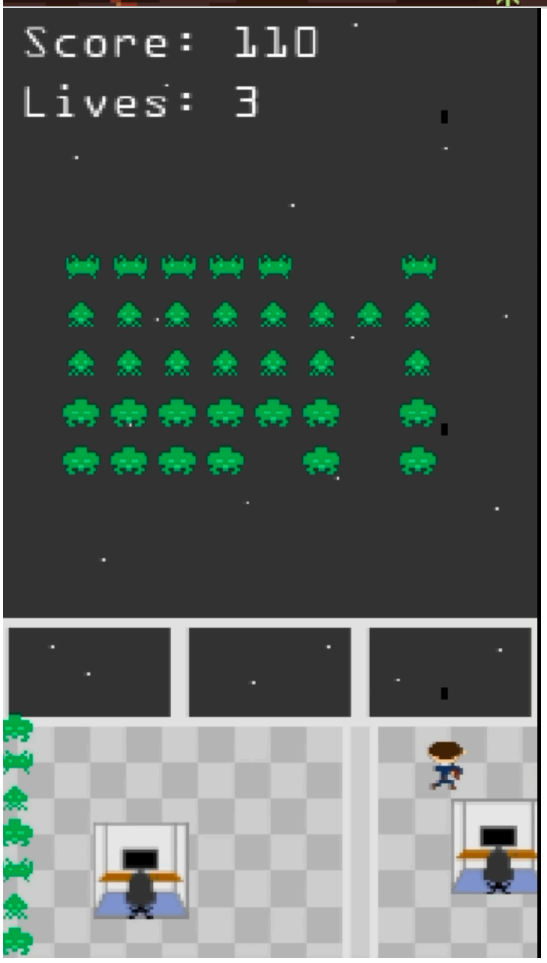
Appendix - Screenshots of the Student Games

Mash-Ups: Asteroids and Flappy Bird

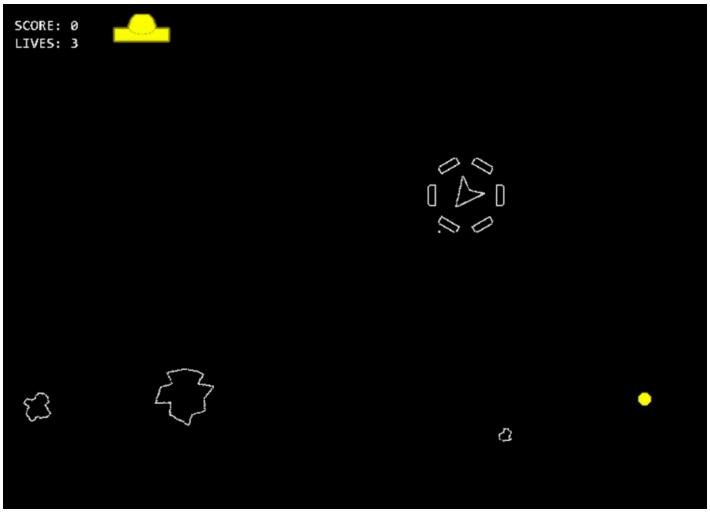


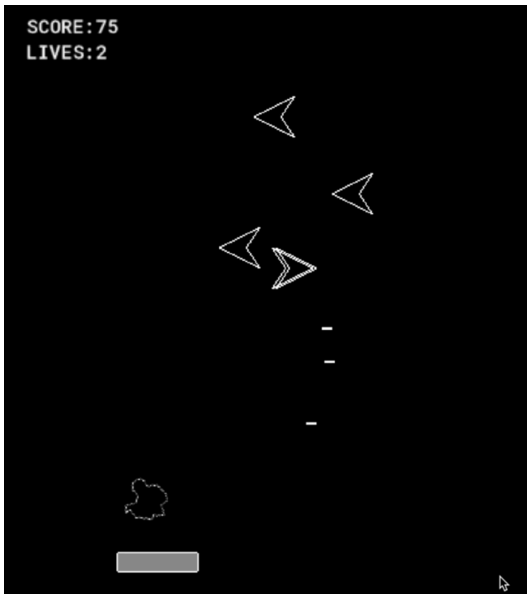


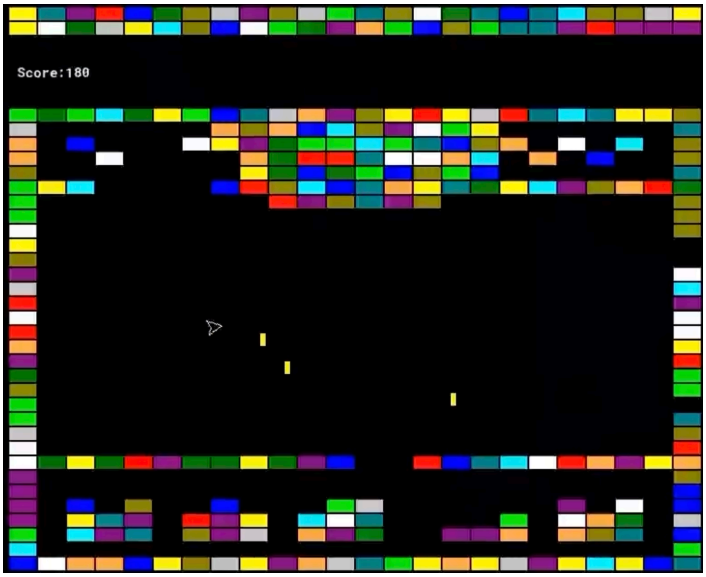

811


Score: 110
Lives: 3

Mash-Up: Asteroids and Snake


SCORE: 0
LIVES: 3


SCORE:75
LIVES:2


Score:180