FirstRand

What I learned making a video game

Author: RACHEL EDELSTEIN

In 1981, the arcade game Centipede was released by Atari, a world leader of the video game market in the late 20th century. Following the same formula as several other famous arcade games of the time, such as Pac-Man and Galaxian, the game involves a moving, player-controlled object (also known as the Bug Blaster) that tries to eliminate the enemy (Centipede) that appears on the screen. The game, as per usual, has several possible outcomes. One such outcome is that the player is destroyed upon being hit by the enemy object. In this case, the player loses instantly. Alternatively, the player succeeds in destroying the enemy object, instantly winning the game. Another outcome exists where the player can advance to the next level by making it to the end of the round unscathed.

Sounds pretty straightforward, right? Well, to a large extent it is. But it didn't seem so simple when, as a third-year engineering student, I was tasked to design, code, and essentially replicate a version of it in the programming language C++.

With only a little experience in programming and basic understanding under my belt, this task seemed overwhelming. Slowly though, by breaking the project down into manageable, bite-sized pieces, the end goal drew closer and felt more attainable.

In our version of Centipede, four major components were required: the Centipede itself (the enemy object), which crawled down the screen approaching the player; the Bug Blaster (player object), which moves left and right with the arrows on the keyboard; a field of stationary mushrooms obstructing the Centipede's path; and the player's ammunition, specifically lasers, shot using the space key to eliminate the Centipede. The game needed to work in practice and be playable by anyone.

Although the coding itself wasn't scary, grasping the tools and new concepts in a relatively short time-frame was. Retrospectively, the learnings that took place are valuable in other contexts and can be applied to all kinds of overwhelming projects.

While most of my learning only happened after the final deadline, I am confident that prior knowledge of these tools would have made the whole process less stressful.

My number one learning objective is to code 'clean'.

One of the main goals in coding 'clean' is to make the code easily readable, both to others and to oneself, so that when reviewing it, it is easy to understand and debug. More generally, good comprehension is important in a project of any kind. My learning here has been to avoid working sloppily with the intention of coming back to it later, as it will only make the project more tedious and long-winded at the end.

Elon Musk once famously shared his two rules for learning, one of which is particularly relevant here. This rule is to build a tree of knowledge. Musk has said that when it comes to learning, one must view information as a kind of semantic tree. The foundation and major theories form the trunk and branches of the tree and, likewise, the subtleties form the leaves. Naturally, without the trunk and branches there can be no leaves or flowers. So, to extend the metaphor, ensure a solid foundation before expending energy on the nuances.

Keeping the end goal in mind is a considerable help when it comes to staying on track and making consistent progress. So, rather than putting a half-baked plan in place now, only to find that it inhibits the final project's success later, it is preferrable to have planned the foundation well.

The term object-oriented programming (OOP) refers to the concept where code is written with the intention of layering pieces of reusable code throughout the programme. For example, the Centipede would be a suitable object in the game, as it has responsibilities specific to it that are used repeatedly. One such responsibility would be its need to interact with the lasers shot at it, as well as the mushrooms in its path. In this context, applying Musk's rule would have meant making use of OOP from the start, as instructed, rather than trying to piece it together for the final deadline, since the OOP concept forms the project's foundation.

Another valuable lesson I wish I'd known at the time is one that might seem counter-intuitive at first. While one may be inclined to gather all the necessary research at the start of the project, going through this process myself has highlighted that it is a better use of time and headspace to research as you go.

For the Centipede project specifically, the first hand-in deadline required all the students to deliver a game consisting of only a movable object, controlled by the keyboard arrows. Doing light research was enough in this case, and soon I had a purple square that moved across the game screen. Later in the project, though, I discovered that bonus marks would be awarded for 'good' graphics. In other words, having a creative-looking Bug-Blaster appearing on the screen would earn more marks than a plain-old purple square.

The positive correlation between good graphics and a higher project mark is more obvious to me now, but this probably worked in my favour after all. Had I been caught up in sophisticated graphics at the beginning of the project, I might not have had the focus necessary to achieve a moving object at all. With a mass of information present on the internet, it is easy to diverge by trying to grasp too many concepts at once. Rather, a better technique is simply to start somewhere and then fill in the gaps as needed.

Believe it or not, I had a working game in the end: a moving Centipede, Bug Blaster, lasers, and all. There was even some valuable knowledge accumulated along the way. In short, the lessons I found most valuable were to have a general direction in mind for where you want to go with the project, to code 'clean' (in a figurative sense when the project doesn't involve coding and in a literal one when it does), to focus on the fundamentals, and simply to make a start, rather than trying to gather all the knowledge you think you need at the beginning.

Although these tools might seem too obvious to need spelling out, I think I would have benefitted from bearing them in mind upfront.

The fact is, when it comes to big projects, everyone has their unique approach. That said, there are some definite game-changing tools when it comes to 'getting it done' efficiently and seamlessly. These are my current favourites.