Jayden Ma and Mohamed Camara (E2 Team #10)

Professor Konrad

EK131: Kinect Gaming

12 December 2018

**Space Invaders**

**Background:**

Our intention was to make a simple, yet engaging, game. With the time constraints and the limitations of GridSandbox, along with our programming inexperience, we had to think small. However, a simple game with few features would not keep people engaged if it took too long to complete. Therefore, the game would have to be fast-paced, so players will keep replaying in an effort to get further into the game. This is the mindset of many mobile games, such as Flappy Bird. It had a very simple concept, yet its fast pace made it a worldwide phenomenon.

We had a few ideas for games that would fit into the constraints of GridSandbox. Our primary choice was inspired from old arcade games: *Space Invaders* from 1987. The game’s fun and addicting nature made it perfect. We modified the concept by removing the features of the walls and the enemies shooting back. Otherwise, we remained faithful to our recreation.

**Objective:**

The objective of the game is to kill the aliens before they reach the bottom of the screen and invade your home. The aliens are killed by action of closing the fist, and only one can be killed in a single action: that is, the foremost alien that is in the same column as the ship, which is guided by the player's hand.

Because of this game mechanic, the player must be swift and reactive, knowing which alien to kill first and being fast enough to kill all of them. The aliens start to come faster, which means even faster reaction times will be needed later on in the game. Therefore, the game trains the dexterity of hand motions and the mental responsiveness to stress.

**Steps:**

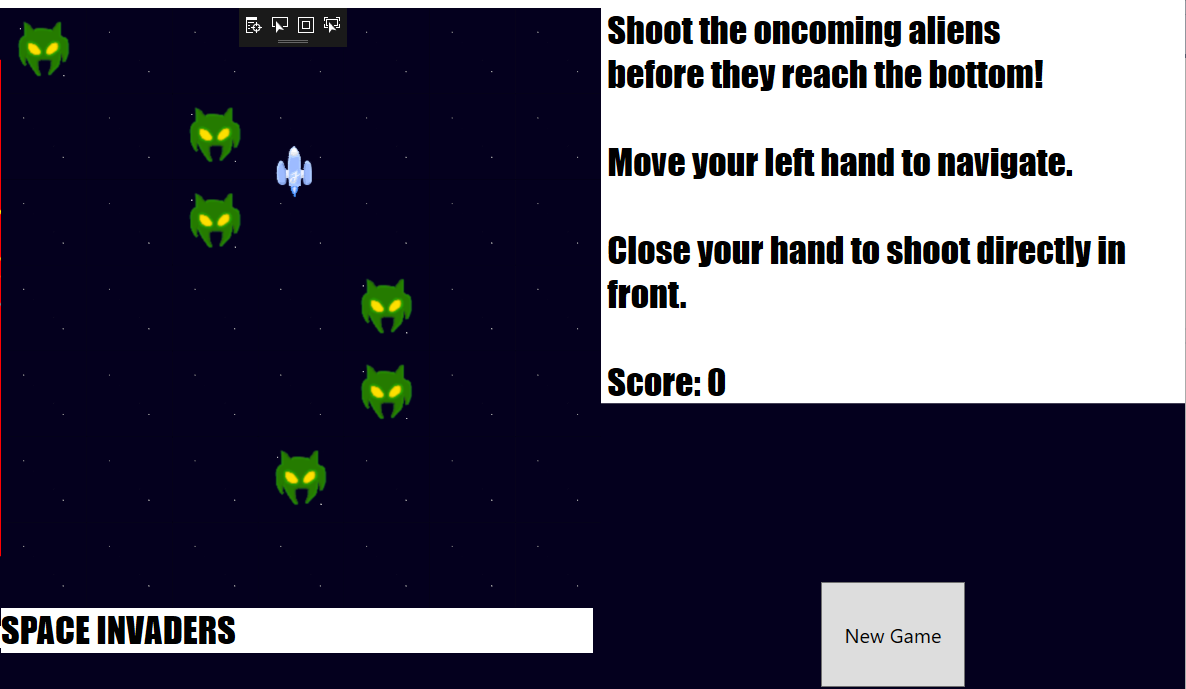
The game initializes, displaying detailed information for the instructions of the game. It reads the highest score from a text file and displays that as well. It waits for the player’s hand to close, which would start the game and allow the functions that generate the aliens and move them down to begin.

After the player’s hand closes, two functions based on delay timers will start. The aliens will start to move down, generated by a randomizer that generates anywhere from one to three aliens in the first row after every delay. After this delay of eighty frames, another function that checks the rows and columns will then move any aliens that it detects down a row.

Another function checks for the closure of the hand constantly. If the player’s hand closes while the game is playing, the function checks the column that the player is in, and every row along that column. The alien closest to the bottom of the screen will be destroyed by this function.

All the while, a function will constantly check if an alien is in the bottommost row. If so, the game is declared over. The functions that generate the aliens will stop, and your score will be displayed in the textbox. If it ends up being higher than the displayed highest score, it will write to the file your score and be declared as the new high score.

A function also increases the difficulty of the game gradually. The way in which the difficulty increases is manually written and therefore not linear, and would be time-consuming to explain in full detail. Essentially however, after a certain score is reached, either the delay of row generation will decrease or the maximum number of aliens in each row will increase. Both will make the game harder to play, until it is eventually impossible.

**Figure:**

**Difficulties:**

Our primary difficulty was our inexperience with programming. This became a huge block, as most of the project was done in programming. However, with Professor Konrad, Jeff, and Natasha’s help, we were able to pull through. In the end, we definitely ended up with far more knowledge of C# logic because of their assistance, and became more confident and successful with our attempts to program new features.

This inexperience led to other difficulties. One of the first major blocks in programming was trying to figure out how to generate rows of aliens and move them down. The way we initially wrote it, the function would duplicate the row down instead of moving it, as the generation function kept repeating itself. This was due to a lack of knowledge for the logic of the same. With help from everyone, we were able to write a function that would generate rows only after a certain amount of time, and then each row and column would be scanned for aliens that would be moved down. This was important, because it became the general idea for the kill function as well: instead of moving them down, the found alien would be destroyed.

We ran into other problems as well. For instance, the randomizer for the row generation led to a generation of more aliens than we intended. With the Professor Konrad’s help, we found out that the function for randomizing was repeating itself, which we wouldn’t have figured out by ourselves. We simply had to put in a return function and that solved. We also had to figure out how to display the high score. After some time of reading up on C#, we figured out how to add a text file into the folder and make it read from and write to the file, logging the score there. These were just a few of the problems we went through, and it showed us how important debugging is in game design.

**Conclusion:**

Given more time, we would likely try to make our Kinect game more faithful to the original game. We could add the walls that act as health points and the ability for the aliens to shoot back in, to make the game last longer and give the player more room for error before they lose. Instead of the game ending as soon as an alien crosses the screen, the aliens must try to shoot the player’s walls down gradually.

Besides this, we could also add in an animation for the destruction of the aliens, as currently they simply disappear. We could add in a more extensive scoreboard instead of displaying only the highest score. Finally, with great improvement in our programming, we could also innovate on the original game by adding special kinds of aliens that give more points and power-ups that give the player an advantage.

We learned a lot through our development of Space Invaders, not only about programming in C#, but also about game logic. Game design clearly isn’t as easy as simply imagining the idea, and a lot goes into actually making it. It’s one thing to think of a good idea, but executing it well is another thing altogether.

Through this project, we also learned a lot about the engineering design process, from thinking of ideas, to taking our first steps, to creating preliminary versions of the game, to finalizing it. While this project, based in programming, may seem ECE-oriented, it teaches a lot about being an engineer in general.

Finally, we hope we made an engaging game that will be played by people in Photonics for years to come. While we did cut down on some ideas, we put a lot of work into it anyways and are proud of the work we’ve done. Hopefully those who play it will find it enjoyable, and when Professor Konrad teaches his module again in the future, we can inspire future engineers to make even better Kinect games.

**References:**

*Professor Janusz Konrad* - Professor, game design and programming assistance

*Natalia Frumkin, Jeffrey Leong* - Teaching Assistants, programming assistance

[*https://docs.microsoft.com/en-us/dotnet/csharp/index*](https://docs.microsoft.com/en-us/dotnet/csharp/index) - General C# reference guide

[*https://stackoverflow.com/*](https://stackoverflow.com/) - Specific help for certain game mechanics

**Appendix**

