Aloha! Tower Defense

Aaron Huff & Ryan Briner

Aloha! Tower Defense is a strategy game focused on the placement of towers to stop enemies from reaching an end point. Our tower defense game has one type of tower that you can build and there is one type of enemy that you will encounter. As the players tower destroys an enemy, they will gain points, these points can be redeemed to purchase more towers to assist in destroying more enemies. After a set amount of time the game will end, and the players score will be presented.

Popular tower defense games like “Kingdom Rush” or “Bloons Tower Defense” offer a plethora of tower options with different abilities, attack speeds, and cost. These games also have different types of enemies that are resistant to one tower attack while being affected greatly by another. These games have an endless format where the levels become increasingly harder over time until the player eventually loses.

We utilized C++ and OpenGL to create the GUI and the backend of our game. We utilized OpenGL to draw create a coordinate plane and map our different polygons to represent different things. The GLFW library allows us to determine if a mouse button is pressed and where the mouse coordinates are. This was kind of tricky as windows utilizes a different window scaling than OpenGL is, so we had to do some conversions to figure out the right locations. There is also a counter for when turrets are placed that decrements “coins” and an if statement to determine if you have enough coins to be able to activate a turret. The damage class just references the towerProx class. The towerProx class determines if an “enemy” is within the radius of an active tower and if so, it marks that enemy and decrements its health. The enemy class draws our enemy in a specific coordinate on the map, if the enemy has been affected by “isdamaged” then it will flicker white to indicate damage being taken. Flickering and making the objects change caused us to have to delve into the functionality of glfwSwapBuffers and how OpenGL renders different images and the layering.

Besides the main function another major class used is the turret class. The turret class utilizes switch cases to draw the turrets whenever they are clicked. The grey positions on the turret are placeholders for where a turret can be placed, once that location is clicked (assuming you have ample coins) the turret is placed, changing the color to purple and “activating the turret”.

== INSERT USER MANUAL ==

Grey Squares: Available Turret Location

Purple Square: Active Turret

Red Triangle: Mob

Flickering Triangle (Red/White): Mob has taken damage

Blue Square: Start Button

Green Area: Unplayable Land

Brown Area: Mob path

You start with 1 coin.

You start with 4 health.

If a minion makes it to the end without “dying” you lose 1 health.

A turret cost 1 coin to place.

Press the blue button to begin the round.

Press the blue button to begin the next round once the minion has died or reached the end.

Problems that we ran into:

* Calling a C++ function through an assembly program
  + Solution:
* Implementing graphics into our program
  + Researched OpenGL and watched instructional videos on OpenGL use with glew and glfw. Tried using glut but removed it because of other issues.
* Multitude of issues surrounding the while loop that checks if the game is still open.
  + Utilized different sets of counters and the Sleep function to insert pauses so holding the mouse does not cause major repeats in the console.
* Implementing text on the screen
  + This involved importing another library and we continuously had issues, so we decided to run all text through the console and chalk this up as a future implementation with more time
* More than one minion and different difficulties of levels
  + With how we utilized OpenGL we would need to draw different shapes continuously which would add an abundance of code. This is possible the way we have it written but further research into OpenGL would allow us to find more elegant solutions.

In the future this game could be improved by updating the graphics and beginning to add different towers and enemies. Alongside increasing the overall amount of playable objects, we could develop other maps for the player to pick from and implement a round system. Creating a leaderboard to store and present scores at the end like an arcade system would be an additional feature in the future as well. We have written our code with scalability in mind. We utilized different methods rather than writing sub-methods to piece things together so if we implemented more towers or minions later down the road then it would be easier to implement.

Overall, we learned a lot about C++ and OpenGL through this project. We discovered the difficulties and pleasantries of working together on a programming assignment. GitHub was an invaluable resource for implementing changes and seeing where something may have gone wrong. I think with the right amount of time and effort we could continue developing this game into something less rudimentary, but under the circumstances we feel that our program represents our collective brain power sufficiently.