IDEA

The idea for our game initially was to create a tower defense game where turrets could be placed to prevent enemies from making it all the way through a specified path. Although through the process of programing this game we decided that the original idea would not be much fun to play at all, so it got changed slightly into something of very similar idea but much more interactive and much more fast-paced. What we came up with in the end was similar to a tower defense game only it is up to the player to eliminate the enemies using speed and precision while aiming with the mouse.

USER INTERFACE

The way that this works is the enemies start at one end of the path a traverse in waves to the other end with varying speeds. With every wave the average speed of the enemies is increased to increase the difficulty. The idea is that you last as long as you can and accumulate as many points as you can by eliminating enemies. When an enemy reaches the other side you lose health. When your health reaches zero, the game ends. There are crosshairs that follow your cursor around the monitor to indicate where exactly you are shooting. If you shoot while aiming at an enemy's hitbox, the enemy is eliminated and a gravestone appears to indicate that it was a hit. In order to make this much more of a challenge, we created an ammo limit. You have 5 shots before you are required to reload. There is an ammo counter directly beside the crosshairs to indicate how many shots you have left. To reload, a reload button spawns in a random location on the screen and you must locate this box and click it as fast as you can in order to continue. To make the experience more user-friendly there are also menus and title screens to display the wave number and to ask if you would like to retry once the game has ended for an attempt to beat your last score.

SOFTWARE DESIGN

Game class

The game class is the most important class in the game. In this class we hold all the variables that are used in the game. (This includes data that holds the enemy count, score, wave number, lives, ammo, bool triggers, bitmaps, fonts, key and mouse states). We use two different external classes to make our game.cpp run properly. We have a Render object called render and several Item class objects (background, cross, CrossReload, an array of 100 "targets"/enemies, Reload, Pew) These Item objects all contain their own characteristics that will be gone over in the Item class section. Most importantly, the game class has two extremely important functions; Run() and End(). This class are called from Main and essentially represent the primary structure of the game. Any logic that you see in the game is done in Run().

Item class

The item class creates a template for every physical object in the game. This class is called several times as items like crosshairs and enemies, but is even used for images such as the background. This is created so that we can easily implement any bitmap wherever we want it, as well as give these individual objects their own variables (bmp, x, y, size, sx, sy, speed, timer, phase as well as a bool for whether or now they should be visible). Finally, the item class has a simple Move() function to integrate the speed of the objects with a single line of code.

Main class

The header of the Main class is where we store all of our included libraries, including the three other headers. Despite being the main class, the class doesn't serve much of a purpose outside of initializing

allegro, hiding the cursor and creating a Game object called game to run the two main game functions (Run() and End(display)) inside the main() function.

Render class

The render class contains a single Draw function that allows us to effortless draw our items in the game class.

IMPLEMENTATION NOTES

We tried our best to proof our code early so we would run into less problems later. One of the ways we did this was by setting a size for all of our Items. This would allow us to have more flexible code down the road. Towards the end of the programming we weren't satisfied with the size of the enemies so we were effortlessly able to change that and have the code automatically adjust just by setting the [item].size. We also created several different crosshair bitmaps, but none really stuck until we added our sniper rifle scope at the end. We went from working with 40x40 bitmaps to finally adding a 3000x3000 bitmap, with the code automatically adjusting to make it a pain-free process.

One of the bigger challenges was getting the enemies to follow a track. We were able to counter this problem by creating several "phases", each with their own instructions on what direction the Item should travel. Phases were triggered when the items hit certain coordinates and was simplified by only going to a phase that was one higher than the one it was one. ie. items can't skip any phases and must travel from one through six.

Another challenge was having the game start itself again after every wave and recycling the enemy objects. We essentially reinitialized the objects after the wave was over, but keeping the score.

The difficulty was made harder by having variables such as speed and number of enemies use the wave number. Speed was set as the wave number and number of enemies was the wave * 10.

The challenges weren't exclusive to the programming. When implementing the sniper scope, the easiest way of doing so was by having a massive bitmap 3000x3000, which came to around 10MB. Although 10MB isn't huge, it seemed like an unnecessarily large file for such a simple item. We also had issues with Adobe Illustrator automatically anti-aliasing the exports, making our al_convert_mask_to_alpha function have an awkward gray outline that was very noticeable. Both of these problems were fixed by setting the bitmap to a 1-bit bitmap. Since we were only using a black and white crosshair, this was an option for us. The 3000x3000 image came to 75KB and had no antialiasing that would ruin the mask.

TEAM STRUCTURE

Ryan Sylvester	Matt Bell
Pathways for enemies	Artwork
Ammo count & Reload system	Enemy eliminations and score
Mouse state	Shooting sound animation
Item Class and dynamic .bmp size compensation throughout code	Render class and End function
Implementation of the waves	Hiding mouse cursor
Design Doc: Software Design, implementation Notes	Design Doc: Idea, UI