Final Report

Introduction

For our game design project, we made a simple shooting game where the player's objective is to clear as many levels as possible on a limited amount of health and ammunition. In each level, a random number of monsters with varying strengths spawn on the screen. The monsters attempt to catch the player, while the player must avoid the monsters and shoot them to death. If the monsters come into contact with the player, the monster dies and the player's health decreases. To clear the level, the player have to eliminate all the monsters on the screen, while keeping his or her own health above zero. In the first level - the tutorial - the player is introduced to the various commands necessary to play the game.

Fig. 1 - Instructions - what the player first sees when the game is opened.

For more of a challenge, the player should select the space bar mode, as the player can only shoot in the direction the sprite the facing. Using the mouse mode is more advantageous, as the player is allowed to shoot at any point on the screen, the only handicap being that the player cannot move and shoot at the same time, unlike in the space bar mode.

Fig. 2 - The tutorial level.

After the first level, there is a possibility that the player might run into a hidden mine. The player can avoid the mines by listening for a ringing sound. If the sound gets higher in pitch, the player should back off to avoid getting blown up (health will decrease). Luckily for the player, bombs can be triggered by monsters as well. (That means less monsters to kill for the player.) Also, there is a possibility that the monster will drop a health or ammunition pack after being defeated by the player.

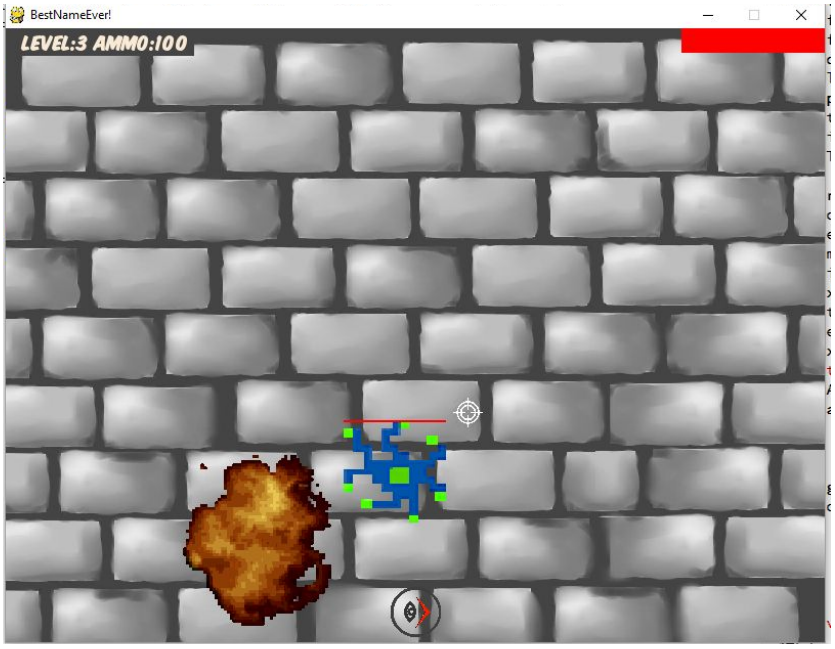

Fig. 3 - The player accidently sets off an explosion, resulting in a decrease in health.

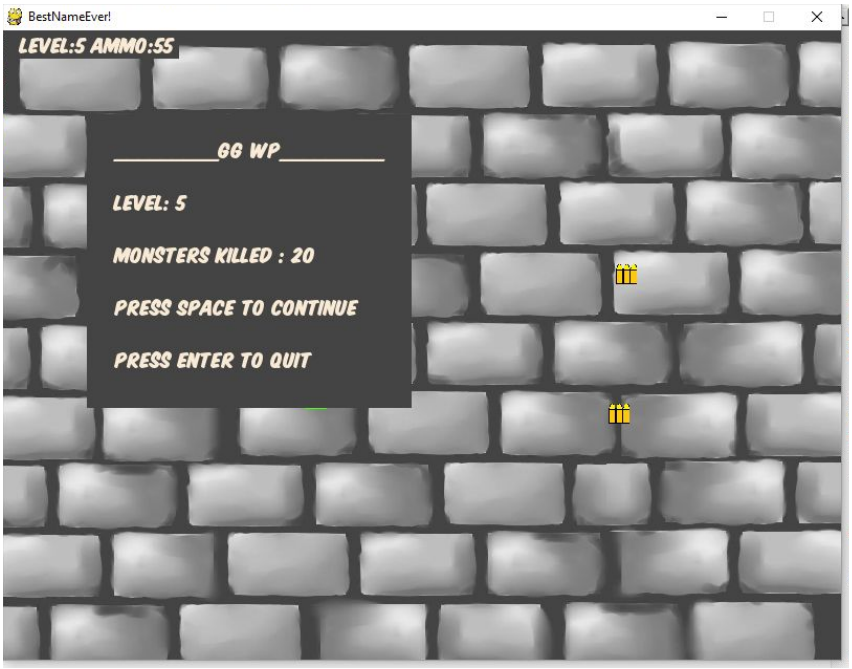

Fig. 4 - The death of the player results in a notification box listing the results of the game, including the option to continue or to quit the game. Notice the (yellow) dropped ammunition on the screen as well.

Background

We designed our shooting game based off of the idea of the stereotypical 2D role playing game, and a simple space shooter game. We combined the elements of both games during the developmental stage. In a basic space shooting game the player shoots to kill enemies or objects that obstructs the path of the player's sprite. Basically, the shooter game constantly spawns enemies on the screen for the player to shoot. However, a space shooting game is much more predictable, as the enemy only attacks the player from the opposite direction, and the player is forced to move forwards. In a stereotypical 2D role playing game, the player is given a wider range of movement, although the background remains static. The player can also be attacked by enemies from any direction, but more often than not, the enemies appear at designated points on the screen.

Fig. 3 - A space shooter game (left) and a typical 2D RPG (right).

The end result of the combination of these two types of games was a flexible 2D shooter RPG, where the enemies are constantly, randomly spawned throughout the screen, and the player interacts with the enemies in every direction. The hidden bomb was an additional element that spiced the gameplay up. Initially, we also meant to have another RPG-like element, in which the player could choose to open one of three

doors, which would lead to the subsequent level. However, the group decided to discard the element, since each level randomly spawned enemies anyways.

After deciding what type of game we would make, we began looking at the functionalities of Pygame, as well as the method through which each component of the game could be programmed. We weren't too worried about creating sprites for the player or the enemy, as we knew, from looking at other games made with Pygame, that you could use PNG images for the monsters and player. Furthermore, we also knew that it was possible to use the coordinates, or positions of the sprites (player and monster) to set up conditionals that would allow us to determine what event of the game would occur. (A death, a decrease in health, or simply being confined by the border of the screen.) This was apparent to us, through simpler games also made with Pygame, such as Pong.

Implementation

On a developmental level, certain basic elements needed to function properly for the basis of the game to work. A brief overview of the Game.py function reveals the basic process through which the game is built. Naturally, all the necessary modules, including those in Pygame, must be imported and initialized. Then the game is given a canvas, or display. Necessary variables (gameExit, tutorial, startText, level, clock) are all initialized to the appropriate values. For every game that is played, all variables are reset to the initial values. After that, containers for the objects/monsters in the game, are also initialized. Empty lists are initialized for the Monster, Health Pack, Ammo Pack, and Hidden Bombs classes. These containers will be appended or deleted to as the player interacts with the game.

The file "Game.py," in essence, is a gigantic while loop that exits when the bool gameExit is set to True. For every iteration of the while loop, a new frame of the game is displayed, reflecting the changes made to the elements inside the lists. In each iteration of the loop, each visual element's position and private members are updated through their member functions, and the necessary objects are reinitialized. A whole

new set of monsters must be reinitialized or “respawned” following all of their deaths in a level. The new set of monsters should be stronger and/or more numerous than the set of monsters from the previous level. A new hidden mine, in a random position, must be reinitialized as well. The player object only reinitializes if a game has ended, and the person playing the game chooses to restart the game. The game will automatically redirect the user to the tutorial level. In the tutorial level, a text display is shown, and only one monster, without the ability to move, is spawned. Once the tutorial has been completed, the game automatically begins in earnest.

For all elements in the game, box coordinates were used to dictate movement and collisions. For instance, when the player presses the up arrow key in the game, the player’s sprite will face up and continue to move in that direction, as long as the up key is pressed, in the following iterations of the while loop. For the player object, this means that its private y-position variable is increased as long as the player is pressing the up arrow, or until the y-position variable reaches its limit (the edge of the screen). The player class uses boolean states to determine the direction the player is moving in using the keys. The boolean states are also used when the player chooses to shoot using the spacebar, as it shoots in the direction that it is facing. In mouse mode, the player object uses an algorithm to calculate the position of the mouse and shoot in the direction that the mouse is pointing in.

```
##The code that player uses to update its image
def reloadImage(self):
      if self.currentFace != self.newFace:
                self.image =
                     pygame.transform.scale(pygame.image.load(Gam
                     eImages.PLAYERIMAGE[self.newFace]).convert_a
                     lpha(),(PlayerConstants.PLAYERWIDTH,PlayerCo
                     nstants.PLAYERHEIGHT))
          self.currentFace = self.newFace
     return self.image
```

The objects of the monster class have their positions updated in a similar method as the player except for three things - (1) monsters, upon aligning coordinates with the sprite from the player class, or enough bullets, “die,” meaning that the monster object is deleted from the list in the “Game.py” file, (2) monsters have the possibility of dropping a health or ammunition pack in the position that they “died” in, and (3) they are spawned randomly, but gradually chase the player in a linear fashion. Both the player and monster class contain private member variables correlating to the position of the bomb, such that when the coordinates of the bomb object aligns with the coordinates of the moving sprites, the bomb explodes, harming the sprites.

Results

(2-4 paragraphs) - Did your game work? Did you manage to implement everything that you wanted to? How did the playtests go? Did people like playing the game? Was it too easy, too hard? Have 3 people playtest the game, and include their feedback in your report.

Our game did indeed work, due to the hard diligence of the entire group. As mentioned in the background, we did not make the game such that the player could choose one of three doors for each level. Furthermore, we also planned to have two tutorials - one tutorial in the beginning, and one tutorial later in the gameplay, for the hidden mines. However, we decided just to have one tutorial and include the hidden mines in the first level of the game. Although we didn’t implement everything that we planned to in the beginning, we also succeeded in adding an additional element-sound - something that we hadn’t even considered when we were initially developing the game. When the player shoots, picks up a health/ammo pack, or comes near a hidden mine, the different sounds are played.

The general consensus of our beta testers was generally positive. Tester 1 liked the game, thought that it was of a mediocre difficulty, but deemed the game “confusing and scary at first,” because the monsters kept on chasing the sprite. Tester

1 was also confused by the ringing noise made the hidden mine. However, Tester 1 asked me to play the game again, demonstrating the game's addictive properties. Tester 2 disliked the game, and also thought it was of mediocre difficulty. Additionally, Tester 2 told us, "The game isn't the problem, the problem is me. I don't like games where I get chased." Tester 2 also suggested that the mines be displayed, because the ringing noise was ineffective, and "the element of surprise gives me high blood pressure." Tester 2 also wanted to play the game again, despite "disliking" the game. Tester 3 expressed interest and enjoyment at playing the game. Tester 3 thought that the game was hard, and noted that it could "become very addicting." Tester 3 suggested that the game allow the user to customize the sprite of the player.

In regards to the future of the game, we may decide to include the option to allow the user to customize the sprite of the player, as Tester 3 suggested. We may also add more backgrounds to differentiate between the levels. Instead of having the stone floor for every level, for instance, we may have a desert theme background at level 5 that changes to a forest theme background at level 10. Furthermore, we may add an option to display the mines, for those with "high blood pressure." However, I doubt that we will ever program the monsters to not chase the player, as this increases the level of tension and stress for the player. These psychological reactions are necessary for the game to be frustrating, and subsequently, addicting. For instance, Flappy Bird is quite simple in concept, but its difficulty entraps the player, driving the player to reach higher and higher scores. Our game's addictiveness relies on that same concept.

Link to youtube video:
https://www.youtube.com/watch?v=jjcqaw1Anjw&feature=youtu.be