Multithreaded Shooting Game

CSCC 31 A4 – Final Project

Sendad, Aloysius Hasheem A.

Xavier University Ateneo de Cagayan

+639952061437

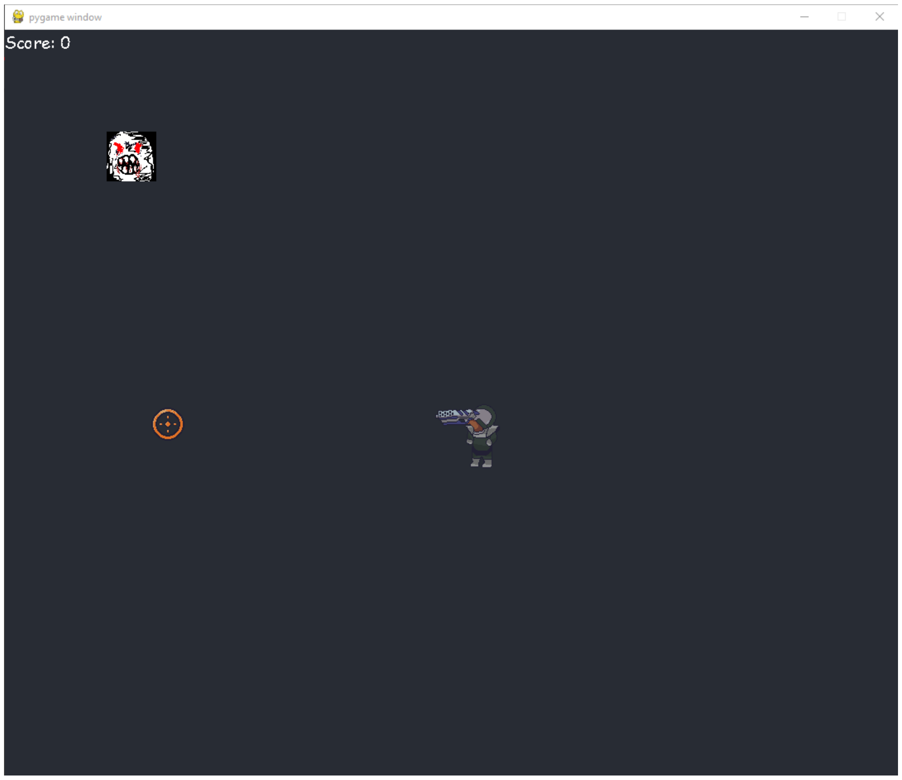
20160010054@my.xu.edu.ph

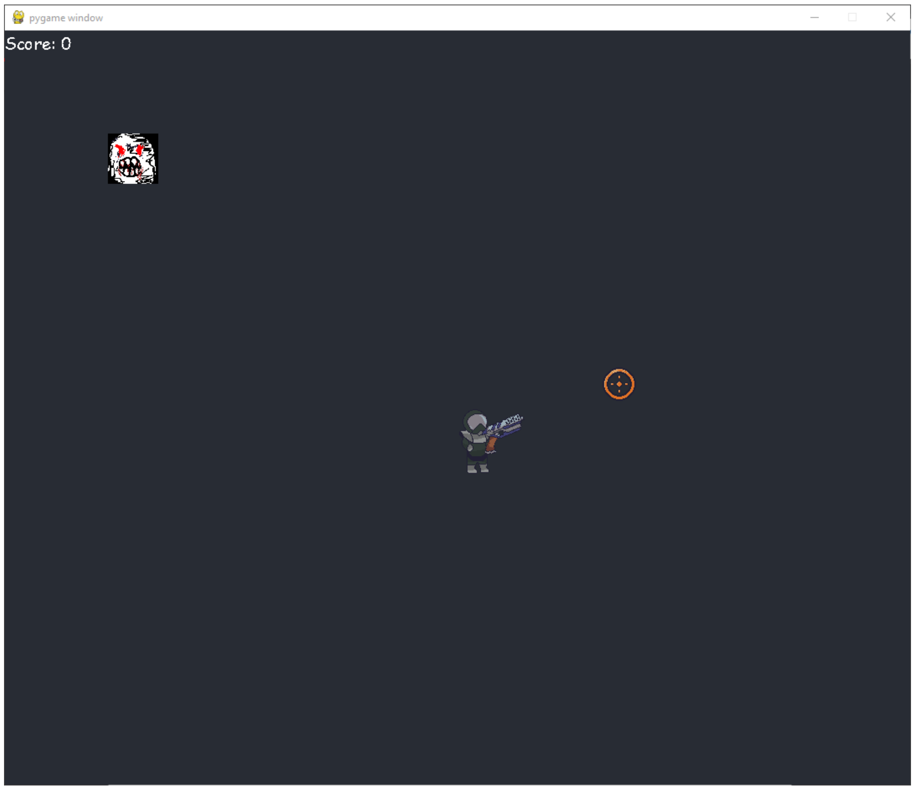
About

As part of the final requirement for the course CSCC 31 – Computer Architecture and Organization, I created a video game project which utilizes current trends in computer architecture, specifically, with multithreading. The purpose of this program is simply for entertainment, and target users of this program are anyone who is familiar with the concept of video games.

Game Design

This game is a simple shooting game in which a player must shoot the enemy before the enemy reaches the player. Upon launching the game, player is displayed in the screen along with the enemy. (See image bellow)





Aside from the player and the enemy, the other things that we see in this screen is: the score, player’s gun, and the cursor.

As displayed with the above images, it is shown that as the cursor moves, the player and the gun move with it. For the player, it detects whether the mouse is to their left or right, and it then faces towards that direction. As for the gun, it always points towards the direction of the cursor.

For the sprites, we have four different sprites: player, gun, enemy, and cursor. Each of these sprites were manually created by me using a pixel art software (namely Aseprite).

Game Mechanics

Mechanics of the game is simple. The player must use the cursor to aim and shoot towards the enemy. If the enemy is hit, the score of the player increases. And, if the enemy reaches the player, it is game over.

This is the gist of the game. For a more detailed explanation, I will be discussing the shooting, the movement of the player and enemy, the scoring, and the edge collision.

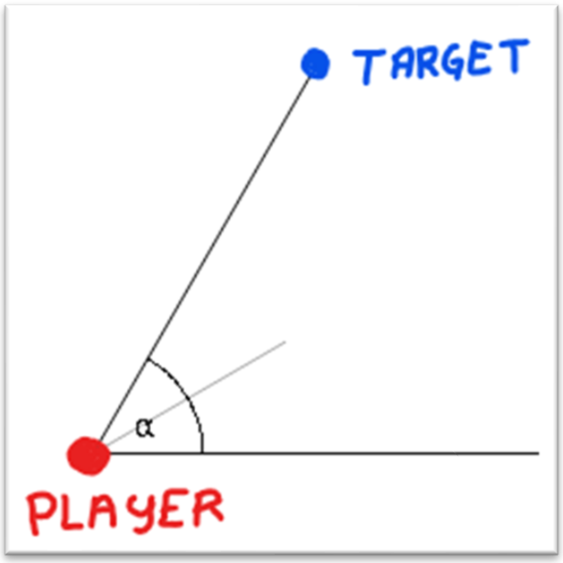
- **Shooting**
  Like most shooting games, player can shoot a target by clicking the left mouse button. Unlike most of the games in the top-down shooting genre, this game does not fire a bullet projectile at a direction, but instead, the position where the cursor

is clicked is the position the bullet is hit. It does not travel from the player to the target, but instead the target is hit immediately. The reasoning for this is simply to keep the project/ code simple.

How this is implemented is having a global ‘hit’ that is assigned an array of [0,0].
>>> hit = [0, 0]

Every time a cursor is clicked, the array is changed to the x, y position of where the mouse was clicked. This position is what determines if it hits the enemy by checking if the position clicked is within the space that the enemy occupies.

- **Movement of Player**
  For the movement of the player, it is directly related to the shooting. In this game, the player cannot move directly, but instead, it moves with the recoil of their shot. As stated earlier, there is a global array called ‘hit’. Along with this, we also have the vector position of the player. With these two positions, we can get the angle along the x- axis to the hit position, where the player vector is the center.



  angle = α

  Given the angle, we can then move the player character to the direction opposite the target by:

  x = -[c * (cos α)]
  y = -[c * (sin α)]

  where:

  (x, y) is the new position
  c is the distance of the recoil

  The negative functions are to move the player opposite to the target, not towards. Upon colliding with the wall, the negative is temporarily removed to produce a bounce like effect

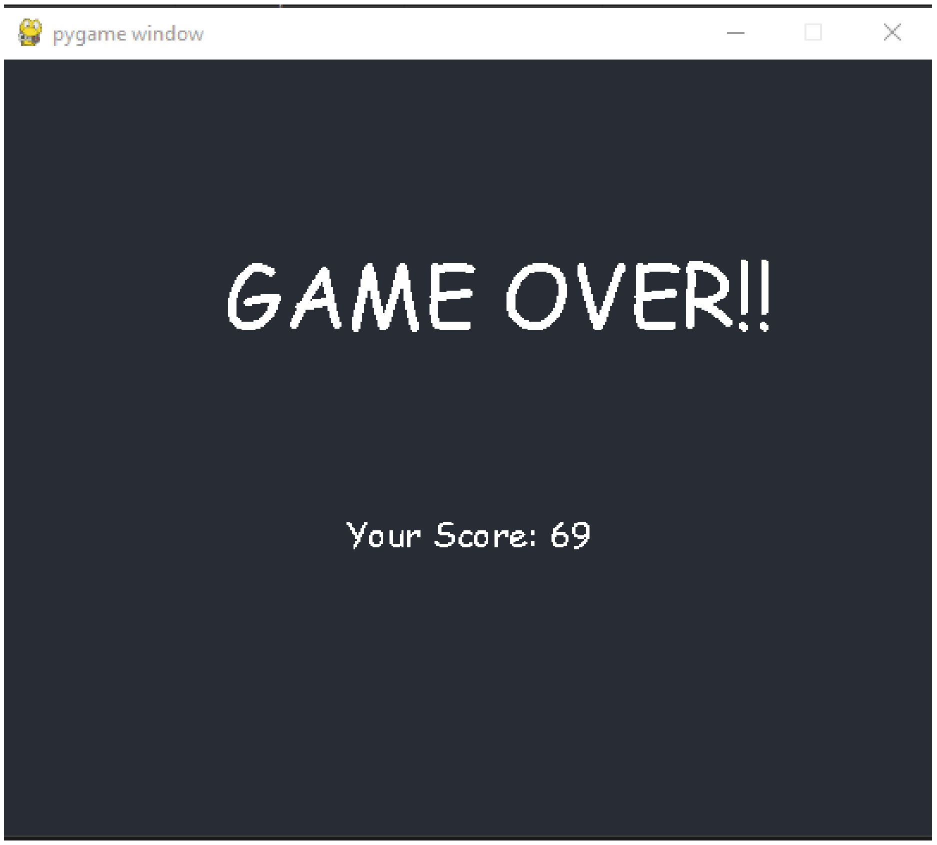
- **Movement of Enemy**
  Enemy movement is much simpler than the player’s movement. From the initial position of the enemy, it moves directly at the slope of 1 or -1 depending on

whether it already collided with the boarders. To explain this simply, the enemy just moves towards a direction, and bounces off the boarders similar to the old bouncing DVD logo screen. Originally, the reason for the enemy to move like this is that this movement is just a placeholder for an AI movement. However, this type of movement turned out to be quite challenging and fun, and it feels like a bullet-hell type of game, so I kept this instead.

- **Scoring and Hitting**
  Scoring is simple in this game. Here, there is a global variable for the score, and every time the player hits the enemy by shooting/ clicking it, the score is incremented by 69. Additionally, if the player hits the exact center, the score is incremented by an extra 420 points.

  Not only does the score increases when the targe is hit, but it also increases its speed. Thus, making the game more challenging the higher the score is.

  Once defeated, the game will end and display your score. This would then conclude the process of the game.



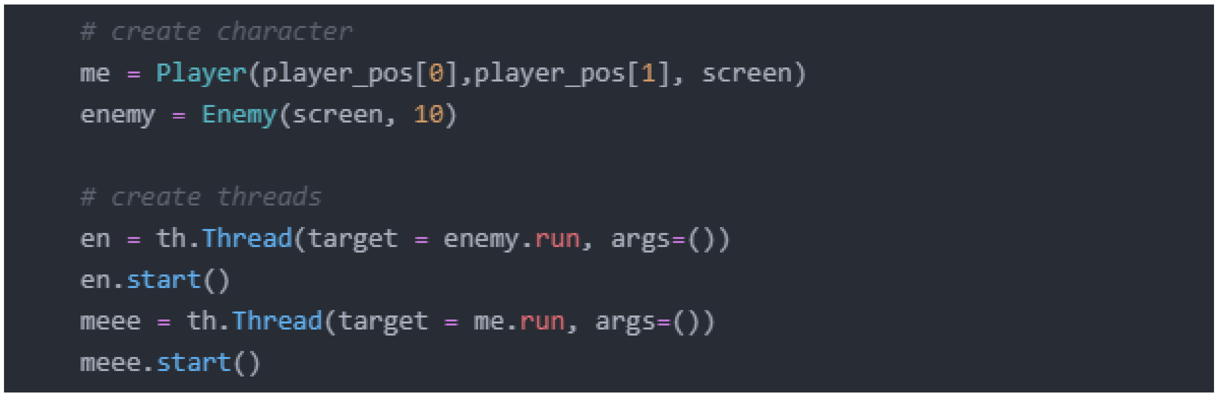
GAME OVER!!

Your Score: 69

Implementation of Multithreading

Multithreading in this program is quite straightforward. Overall, there are three threads running, which is: the main thread, the thread for the player, and the thread for the enemy.

For the main thread, here we run the main game with all its major functionalities including the screen display, setting up of the objects, and the rendering of the sprites. This is the main program of this game. In here, two other threads are created. The thread of the player, and the enemy.



```
# create character
me = Player(player_pos[0],player_pos[1], screen)
enemy = Enemy(screen, 10)

# create threads
en = th.Thread(target = enemy.run, args=())
en.start()
meee = th.Thread(target = me.run, args=())
meee.start()
```

For the threads of the player and the enemy, it is the main thread that renders their sprites towards the screen. What these two threads does is that they assist the main thread in the movement of the player/ enemy. Their main purpose is to do the calculations of the enemy/ player positions on their thread (movement already explained above), and then communicate it with the main thread for it to draw the sprites in the screen at the position calculated by the threads. These threads are useful for dividing the task of calculations because movement (especially the player) involved so many calculations that it is going to be tedious to do by the main thread alone.

To summarize, the purpose of the threading in here is to do the calculations of the movements while the main thread renders what is shown in the screen.

My Notes

Here is the documentation for the program, thank you for reading. I do hope that you enjoy this game, as simple as it is.

This code is somewhat rough around the edges, but it works fine. The main limitation that I have in the development for this is the time constraint. I wish I could develop it further.

Technically, this is the first game I've ever made, so again, I hope it is enjoyed despite how simple it is.