Space Invaders

CS22B Project

Group 3

Peter Nguyen

Alexis Torres

**Table of contents**

Essay 3 - 7

Instructions for Using the Programs 8 - 9

**Essay**

The focus of our project was to learn and use the pygame library to create a video game that is organized with object-oriented programming. For this project, we decided to create a clone of Space Invaders, as it is a familiar game with only a handful of discrete objects.

Pygame is a set of modules that allows you to implement features such as handling user input, event loops, sounds and music, and graphics in writing a 2-D video game. The basic premise of a pygame consists of initializing pygame, creating a window, creating objects and drawing them to the window so that their top left-corner will be at some given (x, y) location from the top-left corner of the screen. The duration of the game loop is user-controlled and dictates how quickly object interactions occur and how many frames are shown per second. This means that by setting our game’s clock to 60 ticks per second, the display will fresh 60 times per second, and a character moving with a speed of 5 pixels will move 300 pixels across the screen.

While it is possible to write lines for each of the attributes and methods of the unique characters, we were able to implement object-oriented programming to organize all of the items that we wanted to draw on our screen as instances of just a few distinct classes. There are two main classes of objects and Obj.

We separated our whole program into three files: space\_invaders.py, mob.py, and sound.py. The space invaders file is our base file which use the functions contained within the other two modules to run our space invaders game. This file contains the functions to open our game screen and keep it running until the user is finished playing. The Mob file is a module that contains all the classes and subclasses for each of the objects in our program. There are roughly four objects that are displayed on our screen: The space background of the game, the rocket ship that our user can control, the bullets that they fire, and the enemy alien sprites. Together, they all function to recreate a perfect simulation of space invaders. The sound file contains functions that utilize the mixer function from our pygame library. this mixer function allows us to manipulate and play music in our game. There are three functions in the sound file that we utilize. One is the background music which is called at the beginning of the game runtime loop so there will always be music. Another function plays a laser sound whenever we decide to shoot our projectile in the game. The last function called sounds a collision noise whenever a collision is made between our projectile and a random alien.

Our program will open up a screen of a set resolution of 600 by 800 pixels. The increasing direction for the Y-axis would be in the negative direction and vice versa. The increasing direction for the X-axis will be in the right direction and the decreasing direction is the negative. The reason that the enemy and the player mob have boundaries set at 735 instead of 800 is that we must take into account that they are 64 by 64 pixels. The bullet shot is 32 by 32 pixels. Also, the enemy objects are stored within an array and we are able to change the number of enemies created by changing the number of enemies variable. It is currently set at 6 enemies.

Everything in our program can be modified to fit the user’s needs. It is very easy to change the resolution size by changing it in the main space\_invaders file. There are variables that dictate everything in our program and changing them would change the game as a whole. The user would also have to update the boundaries in our mob files so that each object still hits their boundaries.

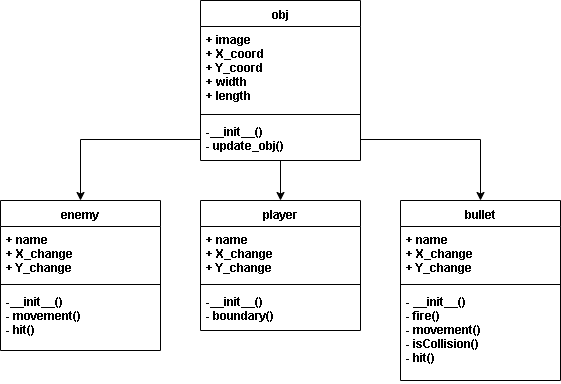
Going in order, our main game loop is set up where our game will quit whenever the user wants to end the program. Firstly in the loop, our background will be rendered into the game screen using the pygame blitting function. Then our program checks if any keys are pressed and acts accordingly. If the left or right key is pressed, it will give our ship object either a positive or negative speed in the X axis. It will traverse left and right until we decide to release our key. This will usually go through a few loops before doing so. Our ship speed will return to 0 whenever the left and right keys are released. If the spacebar is pressed, a projectile will fire from the middle of our of our ship. The reason that we have the projectile’s x and y coordinates be added a certain value is because that would align it with our spaceship sprite.

After checking for keys, our program will check if any enemy alien has reached a Y-coordinate that is too close to our player coordinates. It loops through the array of enemies because there are more than one enemy objects. The enemy objects are stored within a list and our game loop goes through each one of them and checks all of their conditions for each iteration. The enemies will continue moving back and forth and closer to the player if they don’t collide with our bullets.

The last few functions checks the player’s boundaries to see if they are still within the range of the screen and then calls the screen update function. There is also a score function that shows our current score in the game. We increment by one point each time our bullet collides with the enemy aliens. The scoring function utilizes another pygame feature: Font manipulation and font printing. After all of these functions resolve themselves, the whole display screen gets updated by the pygame update function.

If one of the enemy alien objects do end up reaching the zone we set as the player’s field, it would move all of our aliens off of the screen and display a game over screen in bold letters with a 32 font size. We decided on the font freesansbold because it was a font that was easily accessible with the pygame library.

Moving on to the mob.py module file, this file, as stated before, contains classes and subclasses of every object in our game. The parent class, obj, takes in an object’s image file name and x and y coordinates and stores them into their respective variables. The file name is used to invoke the pygame image loading function so that we can have the actual image that we can put into our screen.



All three of the subclasses in this file contain the same variables which are the changes(speeds in pixels) in the X and Y coordinates and everything from the main parent class. Mostly, the child subclasses get called and a super() function is used to assign all of the values to their respective variables. The three subclasses are also very similar as they utilize similar functions to each other but with different number setting such as how the bullet’s boundaries are checked in the Y-direction to see if it has reached the end of its trajectory. This is different from the enemy object because its boundaries are checked on the X coordinate. That is because in the real game, the alien enemies are supposed to descend a certain distance each time they touch each side of the screen. Each function in this file also has a specific action.

The \_\_init\_\_() function is used to initialize each of our object’s attributes into their respective variables. Both parent classes and child classes contain attributes that dictate the characteristics and actions that each mob in the game can do.

The movement() function varies with each of the mobs. For the enemy, it will add position values in the Y axis every time the mob touches the boundary line of our game screen. Our resolution is 600 by 800 pixels so having the condition be when the mob’s position is less than 0 or greater than 735 will cause it to shift directions and move closer towards the player. For the bullet object, the movement function is a function that is activated when the player hits the space key and fires the bullet. A pixelated bullet image is then shot out of our current player’s position. The bullet takes the x coordinate of the player ship and changes its position based on its set speed of 10 pixels.

The boundary() function is used to check boundaries for whenever the mob reaches the end of the screen and then act accordingly to make sure no bugs occur. For the player child class, the boundary function prevents the player from going further than 735 pixels and going less than 0 pixels otherwise the player would traverse off the screen. This would ruin our program because the player must always be on the screen to be able to properly play the game. The boundary function components are also implemented in the bullet class and the enemy class. For the bullet child class, it checks the bullet to see whether a collision has occurred or if the bullet reached the end of the screen in the Y\_axis. After doing one of these conditions, the bullet would reset to the original Y coordinate of the player’s ship. This is so that it can simulate that the player can fire the bullet from the ship again.

For the enemy child class, it will make it so that the enemy will automatically descend towards our player’s ship every time it hits a boundary. This is a core concept in space invaders and it is thoroughly recreated here.

The hit() function is a function that resets the mob position values so that the game can continue. For the bullet, the hit function readies the state attribute which is the condition that allows the bullet to be fired again. It also resets the position of the bullet in the X and Y direction to the player’s so that it may be fired from the player again.

For the enemy, the hit function gives them a new random X and Y coordinate in the ranges of 0-735 for the X direction and 50-150 for the Y-direction. This will allow them to randomly appear at the top of the screen every time they collide with the bullet.

The Collision() function is a condition function that uses a math equation that finds the distance between two objects in motion. It is the Pythagorean equation and we use it to find the distance between the bullet and the enemy alien. If the distance between them is less than 27 then it means that both objects are overlapping on the same space, it would return true as a collision. This is the core function that we use to check if objects do collide.

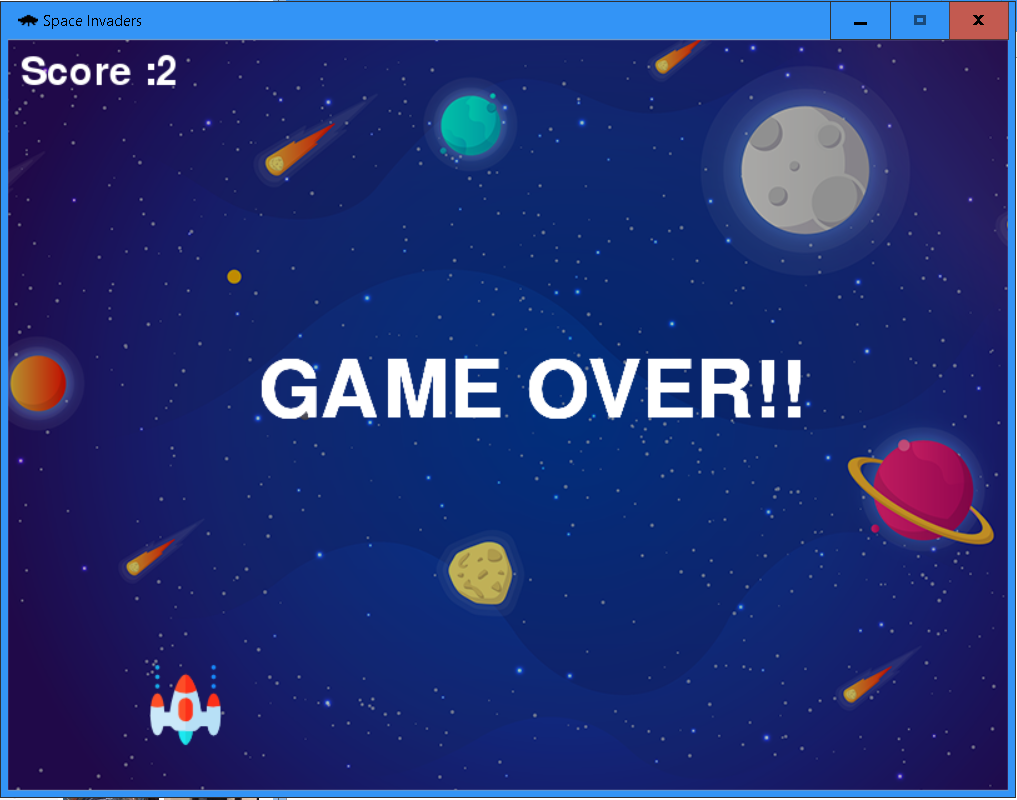
The sound module file was simple in that it was just three functions which implement sound files into variables for us to play using pygame mixer functions. We invoke the functions within our main loop.

Overall, after completing this assignment, we feel that we have developed a good foundation on the pygame library and are confident that we can recreate other games. The benefits of using inherited classes in our program are greater than if we were to create separate classes on their own because most functions are shared between each class.

This project was good in that it taught us how to utilize basic object oriented programming design and graphic user interface. The key components to take away from this assignment is our newly gained knowledge on pygame and the ability to create modules for better organization and readability of our code.

**Sample Output**

****

****

**Instructions for running the program**

1. Open Space\_Invaders.py in an IDE. (File must be in the Space\_Invaders folder)
2. Run the file

**Space\_Invaders Instructions**

Shoot enemies and keep them from reaching you. Destroying a ship increases your score by 1.

**Controls**

* Left arrow: Move the player ship to left
* Right arrow: Move the player ship right
* Space: Fire a bullet