# **“COMPUTER GRAPHICS FINAL PAPER”**

**COMPUTER ENGINEERING(CO313)**



## **“SPACE INVADER GAME”**

**Project Report**

|  |  |  |
| --- | --- | --- |
| **NAME** | **ROLL NO** | **EMAIL I’D** |
| **Zishnendu Sarker** | **2K19/CO/450**  **A6** | ([zishnendusarker\_2k19co450@dtu.ac.in](mailto:zishnendusarker_2k19co450@dtu.ac.in)) |

*A comprehensive project report has been submitted in partial fulfillment of the requirements for the degree of*

**Bachelor of Technology**

in

**Computer Engineering**

*Under the supervision*

of

**Sir Raju Kumar**

**Delhi Technological University, formerly**

**Delhi College of Engineering**

**Department of Computer Science Engineering**

Delhi Technological University, Shahbad Daulatpur, **Main Bawana Road**, Delhi-110042. **India**

# 

# **Acknowledgment:**

We have completed our report on “**Space Invaders in Pygame** ”.Our report won’t be possible without the contribution of several individuals from different walks of life. We would like to express our humble gratitude as well as thank our honorable instructor of the course ‘ Computer Graphics’ **Raju Kumar**, COE Department, DTU for her direction, constant and spontaneous support, and constructive suggestions. Without her help, this report could not have been a comprehensive one. We would like to thank some of the individuals for lending their help and giving us their invaluable time to make this report happen. We have also taken help from different websites for collecting information.

# 

# **ABSTRACT:**

An enhancement for the old-style space intruders’ game that includes shooting projectiles at trespassers in a space setting. In our variety, the alien spaceship can likewise discharge shots at the client's spaceship. We need to overcome 5 columns of 10 aliens each, for example, 50 aliens in each round with outsider rates expanding as they are killed. We will likewise add safeguards as insurance for the client's spaceship in specific regions. The safeguards should be annihilated for the projectiles to hit the spaceship. The spaceship will have a specific number of lives. There will be 4 sorts of alien spaceships in the game which will have various varieties of focuses. A secret boat will likewise show up every once in a while, and in case it is obliterated, countless focuses will be given. Be that as it may, it will not remain on the screen for quite a while. After the first round, the client spaceship will actually want to shoot various slugs all the while as trouble increments. The game ends if all the lives are consumed or the alien spaceships arrive at the base.

# **TABLE OF CONTENT:**

|  |  |  |
| --- | --- | --- |
| **S.N** | **TOPICS** | **PAGE NO.** |
| 1. | Acknowledgment | 3 |
| 2. | Abstract | 3 |
| 3. | Introduction   * Objective * History and Background | 4 |
| 4. | Aspect Ratio | 5 |
| 5. | Working Process | 6 |
| 7. | Basic Setup | 7 |
| 8. | Implementation and code:   * Creating the player * Creating the laser for the player * Creating the obstacles * Creating the aliens * Collisions * Adding the health system * Adding the score System * Adding the CRT styling * Adding the music and sound * Adding the victory screen | 8-23 |
| 9 | Conclusion and Future Scope | 24 |
| 10 | References | 24 |

# 

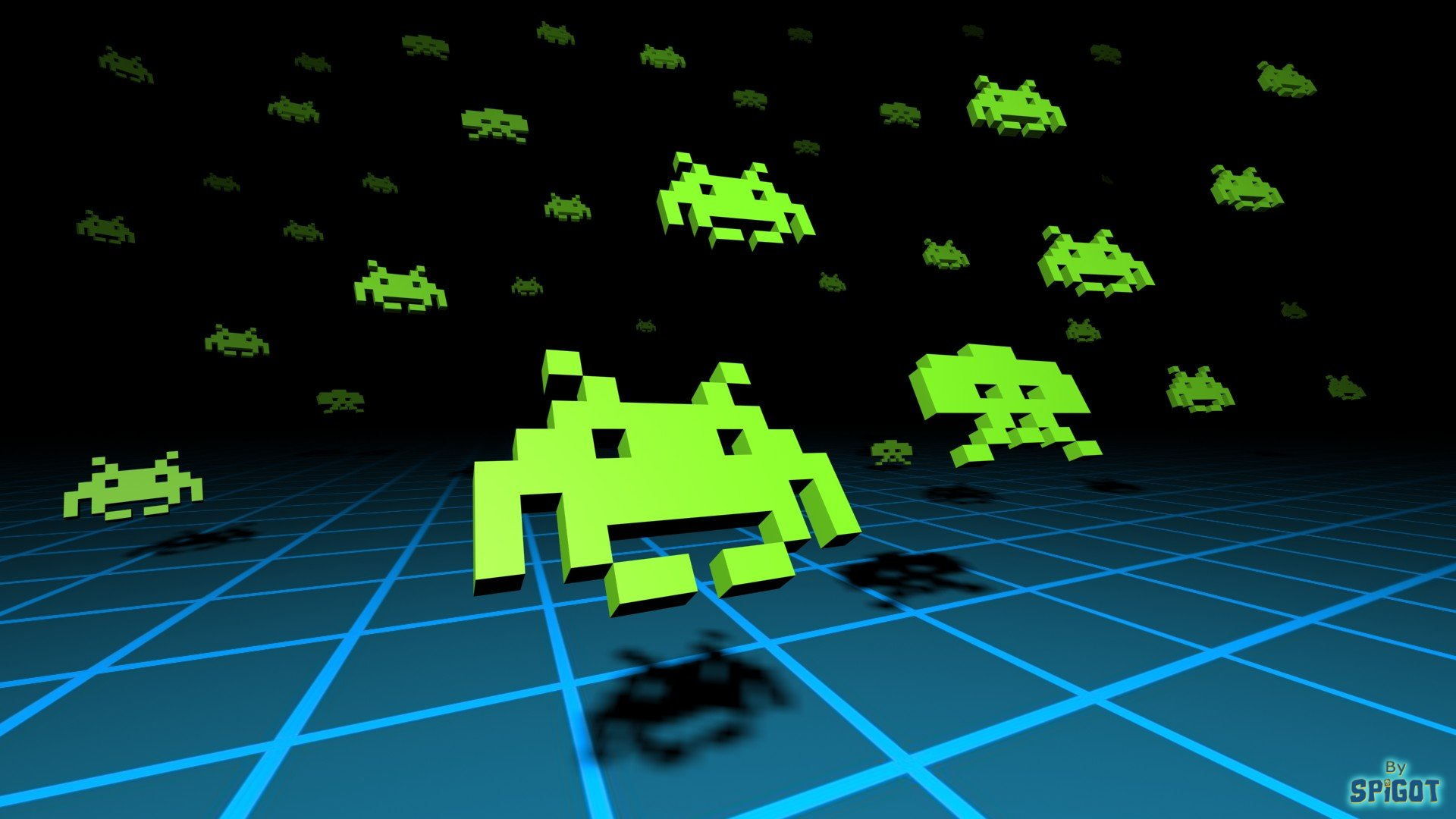
# **INTRODUCTION:**

The game depends on the famous 2D arcade game, Space Invaders, created in the ‘70s by Tomohiro Nishikado. There have been a few executions of the various variants of the game over the long haul. There have not been many 3D executions of this game. The execution examined in this report plans to deliver a 3D adaptation of the game, with a clever UI that carries more authenticity to the player.

## **OBJECTIVE:**

The objective of the game is simply to destroy a formation of advancing enemy spaceships. The player moves on to the next level upon destroying all enemy ships. Each consecutive level gets tougher and tougher with enemies shooting at the player more frequently. The number of points obtained per enemy destroyed increases with every level. The player should aim to maximize his or her score.

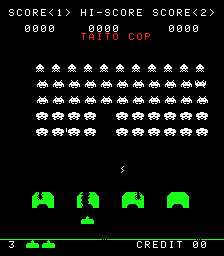
## **BACKGROUND:**



The report will begin with a description of the game, followed by information on how to play it. The talk will next shift to the game's technical features, including the software architecture. The implementation of several elements like collision detection, audio, and procedural modeling will also be presented. To increase the efficiency of rendering the models, optimization techniques were applied in game development. These will be noted as well. The study will finish with suggestions on how the game may be improved in the future.

**ASPECT RATIO:**

In most Space Invaders emulators, one bit of graphical information is drawn as one pixel on the screen.



However, CRT monitors don’t have “pixels” in the same sense. The electron beam is turned on for a period of time as it sweeps horizontally from left to right, illuminating parts of a scanline before it’s turned off again.

Usually, this means that one CRT “pixel” is slightly wider than its tall. The height is one scanline, which can be translated into one pixel, but the width can be more than that. That’s the case for Space Invaders. Because its resolution of 256 x 224 isn’t quite 4:3 like the monitor, the image gets stretched vertically as each pixel occupies a little more of the horizontal scanline.

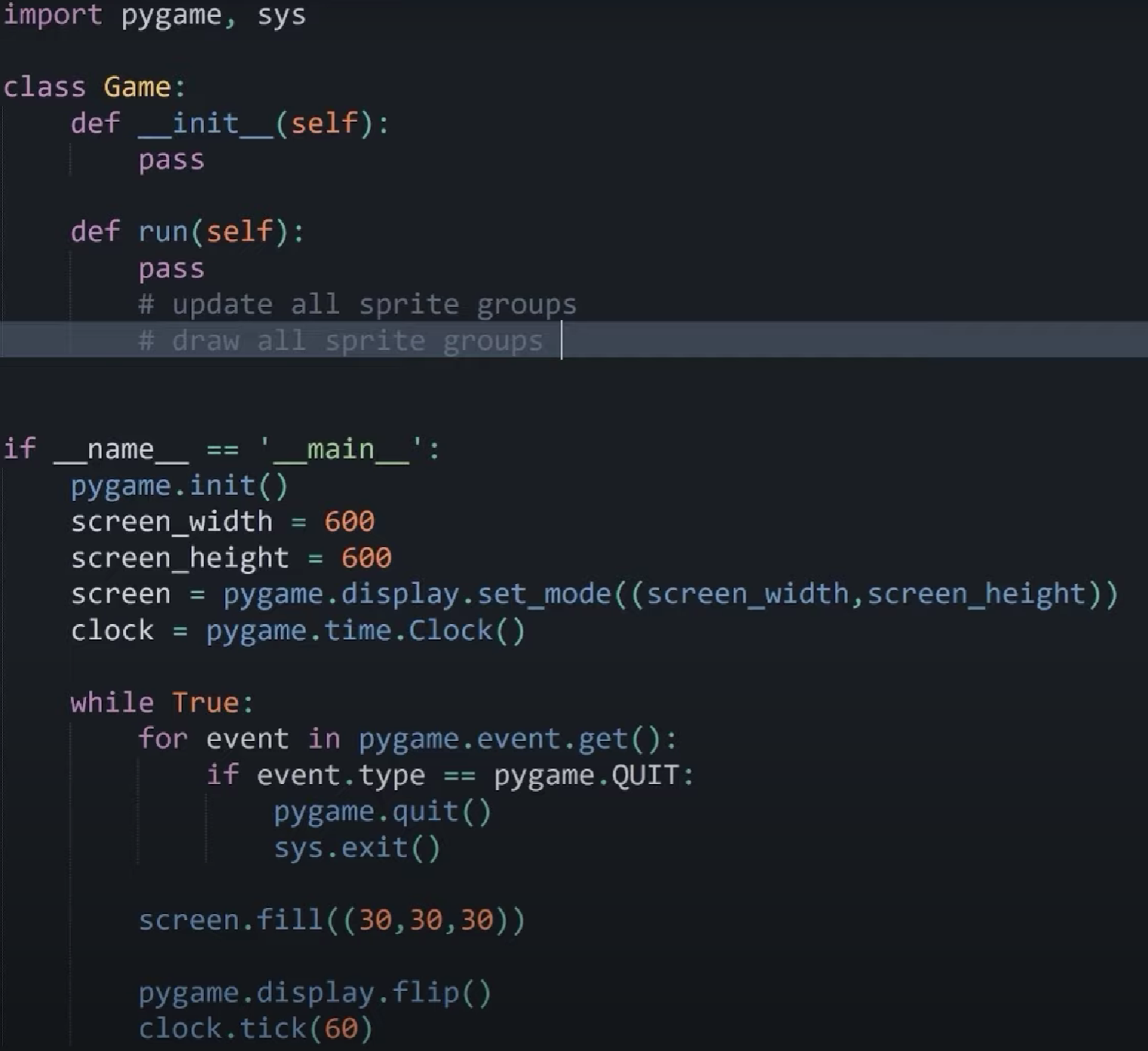
Note that since the CRT monitor in Space Invaders is rotated, one pixel is in fact slightly taller than wide instead, after rotation. This means that the iconic crab-like space invader looks too squat and flat in many emulators, for example. The enemies should look more or less square.

# 

# **Working Process:**

To complete the whole project we divided the total work into different segments. We use Anaconda and pycharm to download all the library and other extension packs to run the game. For the successful output of the game and for getting better results without any problem, firstly, we analyze the game and mark out the points that we need to create in different stages of this game. According to the planning, we started our work and came out with a successful project. The stages or processes that we did in our project are :

* **Intro**
* **Basic setup**
* **Creating the player**
* **Creating the laser for the player**
* **Creating the obstacles**
* **Creating the aliens**
* **Collisions**
* **Adding the health system**
* **Adding the score System**
* **Adding the CRT styling**
* **Adding the music and sound**
* **Adding the victory screen**

Here we start a brief explanation of our work in below :

## **Basic Setup:**

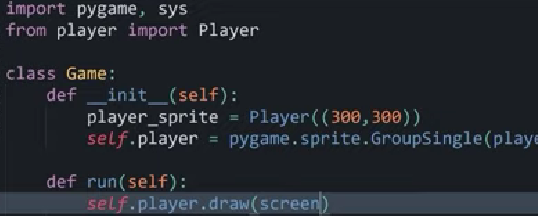
* Firstly, we have the basic pygame setup
* Then the display height and weight is setup
* Using those, we create a display surface
* Then we limit the frame rate
* In the actual game loop, we are checking for the escape button and when we are in the loop, we are drawing the color and anything else drawn in the game.
* Then we are creating a class “Game” where two methods are made. Then we create a reference in the main where we check the if statement for the name to reference main. Then call the game class in the loop which allows it to run.
* In the game class, we called two methods where init was the initial method of the class which contains the sprite groups of the player, obstacle, etc. And this is the framework of our game
* The next is used for updating and drawing all sprite groups
* At last, we created a game instance and run it.  **Fig 2: Drawing player in-game class**

## **Creating Player:**

To set up the full game there must be some player requirements needed to fulfill the game conditions. The requirements are:

### **Show the image of the player**

* Create a class “player.py” and import pygame
* Create the player class which inherits from pygame.sprite.sprite
* We initialize the slf and position of the player and import images and place them in the mid-bottom
* In the game class, we will create player\_sprite with position

Fig 2: Drawing player in game class

* And import the new Player from player.py
* Through the run method we can draw layer in the screen
* player\_sprite= giving screen width/2 and height the image of the player would show in the screen

### **Move the player**

* First, we need to define the inputs and initialize the keys
* Then we gave values to the keys which will take input after being pressed
* For moving right we will increase the speed which is rect.x +=
* For moving right we will decrease the speed which is rect.x -=
* Then we define a function for the self input

### **Constraint player to the window**

* To constrain the player to the window, we have to define constraint and speed to the main method, we have added screen width. We also made a new method named “constraint”.
* And through the constraint method, we control the player inside the window

### **Shoot a laser and recharge**

* Here, we initialize a key which is the space button as the laser shooting button
* We will define a timer for the laser shooting otherwise one tap on the space will continue shooting forever
* We will define a recharge method that will take the current time of the laser
* Every time it will rechange after checking the current time based on laser time and laser cooldown time

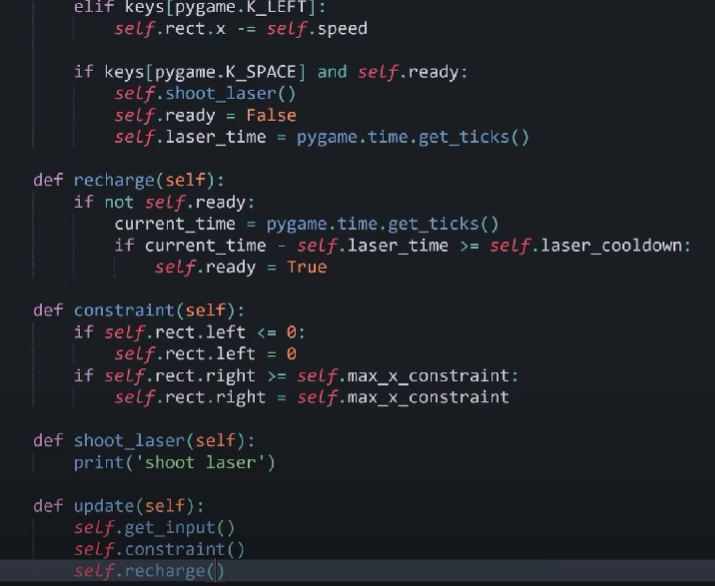


Fig: Shoot and recharge of laser

## **Creating a laser:**

Laser is a sprite class with position and speed. We are making the laser in two different ways; spawn at the player position and move up, and spawn at the alien position and move down.

### **Creating a class called Laser**

* We create an instance of laser
* Self.image and self.rect
* Every time we press space it will laser as an instance
* We took the position of the laser
* We made the laser starting point from the midpoint of the player
* Self.image and self.rect

### **Moving of the Lasers**

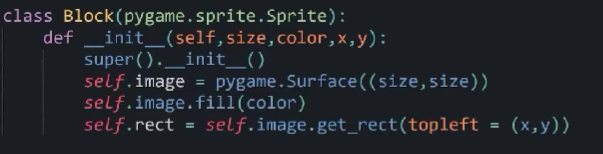
* Firstly, we need to initialize the speed of the laser
* The direction for the moving laser towards up
* Then in the main class, we updated that laser
* Destroying the laser
* In the shoot laser method, we added the laser staring point and speed



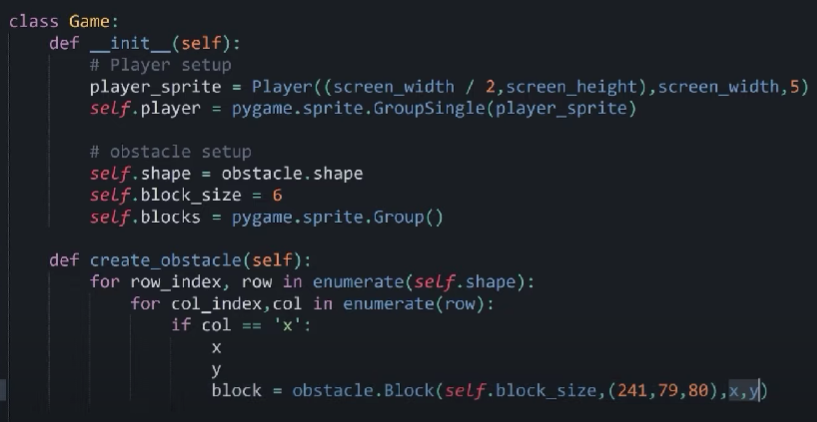
Fig: drawing laser in window Fig: laser class

## **Creating obstacles:**

* Creating an obstacle :
* We will create a class using pygame .
* We initialize image and rect for the obstacles,
* We color the obstacle
* Image must have a size of the images or surface
* Mention the position in the game

Fid : block class(obstacle)

* Shaping of an Obstacle :



* The obstacle may seem like a whole block but it actually consists of many blocks/cells. The cells are named sprite.
* We need to implement the shapes
* We use multidimensional arrays for the obstacle
* In the game class, we will shape the obstacle
* And define the obstacles properly where we will enumerate the obstacle and finalize block size



Fig: the shaping of obstacle

Fig: shape

### **Placements of the obstacles**

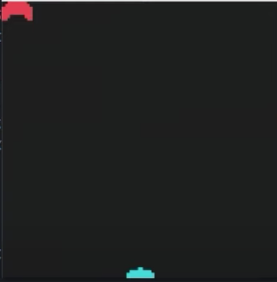
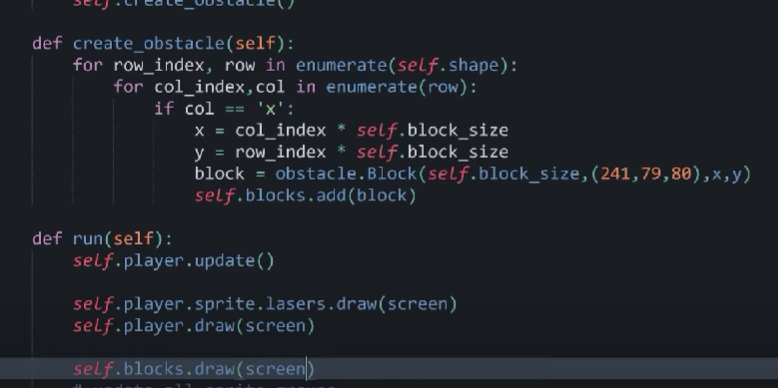
* They are independent of each other
* We initialize column index with the block size
* Also row index with block size
* We have to add self.blocks and add the block we have made into it
* Lastly, we will draw it in Screen 

Fig: Draw Obstacle Fig: First obstacle

### **Create Multiple Obstacle**

* We do almost the same like a single obstacle
* We consider x in offset
* We called here a create\_obstacle instances
* Through this we will pass some arguments
* We will change the position of obstacles by adding x offset

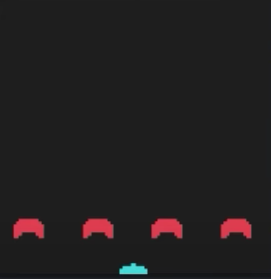
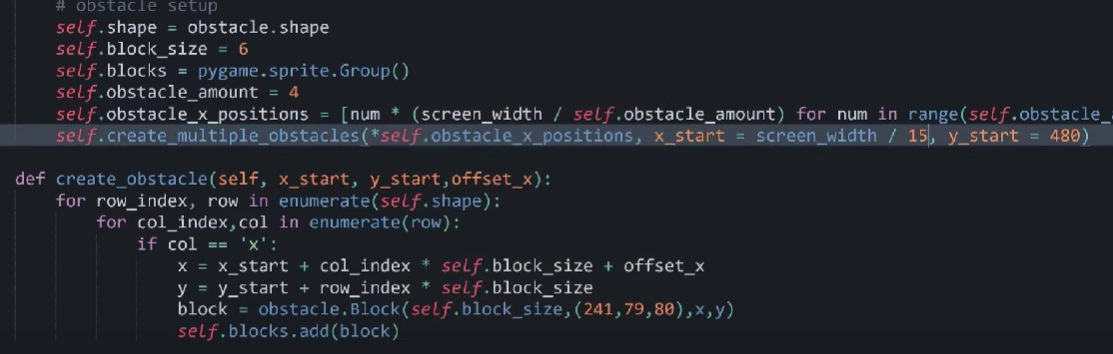


Fig: multiple obstacles placing and defining Fig: Multiple obstacle draws

## **Creating Aliens:**

### **Placements of the aliens:**

* Create a new file saving it as alien.py. Then import pygame.
* Create a class called aliens. It will also be sprite.
* Defining inheritance and in space in between it needs (self)
* Whenever we create an alien we have to specify color(red, yellow or green).

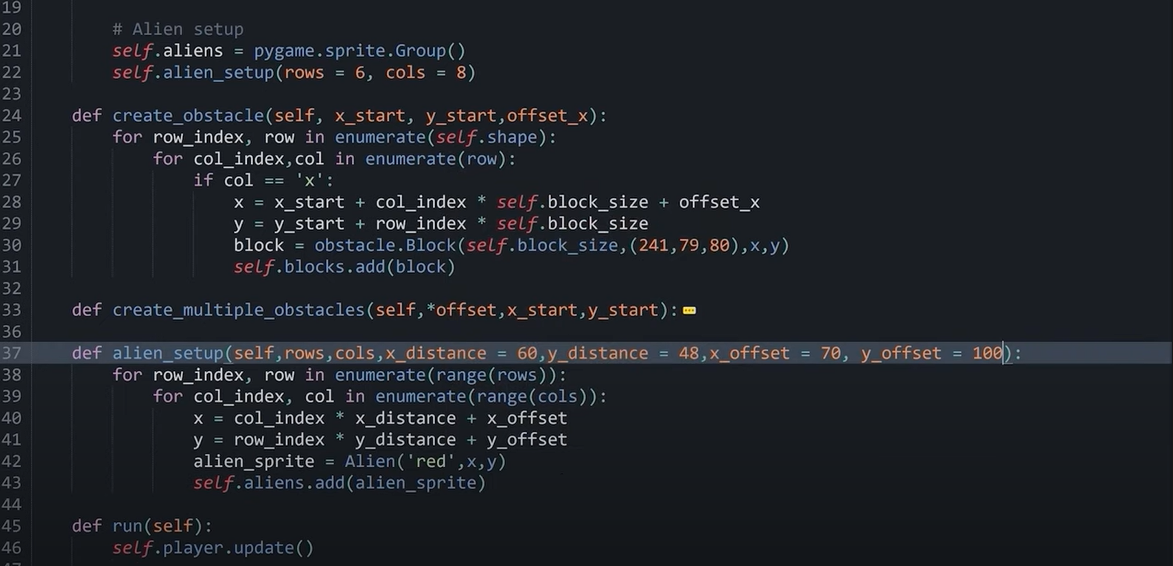
## 

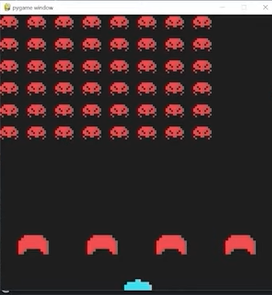
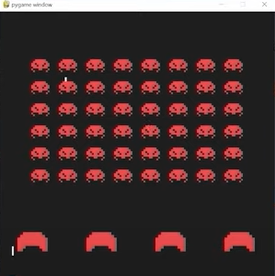
## 

### **Setup of the aliens:**

* Then we have to go to our main file and import the alien new file we created.
* Creating a group where all the aliens will be in the group.
* Creating the function; def alien\_setup
* Now we need rows and columns.

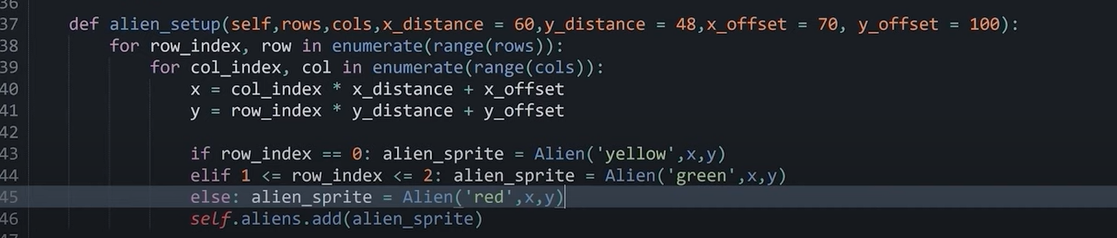
## 

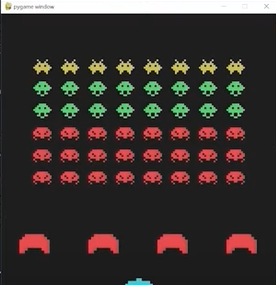


### **The difference in color for the aliens:**

* If the row index is 0, then the color of the alien will be yellow
* If the row index is greater than 1 but less than or equal to 2, then the alien will be green
* Then in the if-else statement, for else the color of the aliens will be red.





### **Making the aliens dynamic:**

* For that, we have to first work on
* Specifying update method
* Specify how fast the aliens will go.
* Then we have to set the direction of the alien on default.
* Then we have to set the direction of the alien to -1; so that they move to the left.



## **Collision:**

For collisions, we can divide this into two parts. they are:

1. Players Lasers
2. Aliens Lasers

We start this operation by defining collision\_checks. The explanation of them are given below :

1. **Players Laser**

* We check the player lasers exist or not,
* and then we check for the lasers towards the player
* Inside it there would be three blocks

1. Obstacle collision
2. Alien Collision
3. Extra collision

* In both places, we will allow laser kill
* If it hits the alien will kill
* If it hits the obstacles, that will kill also

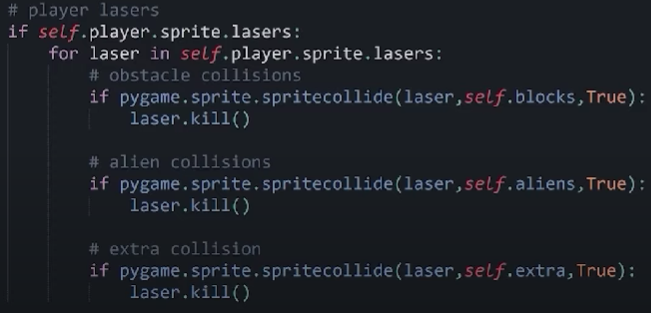
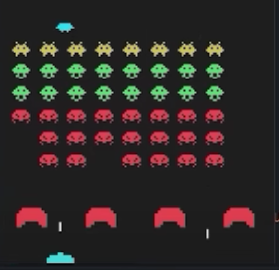


Fig: player laser collision calling Fig: Player laser view

### 2**. Alien Lasers**

* We check the alien lasers exist or not,
* and then we check for the lasers towards alien
* Inside it there would be three blocks

1. Obstacle collision
2. player Collision
3. Aliens

* In both places, we will allow laser kill
* If it hits the player will die
* If it hits the obstacles, that will kill also

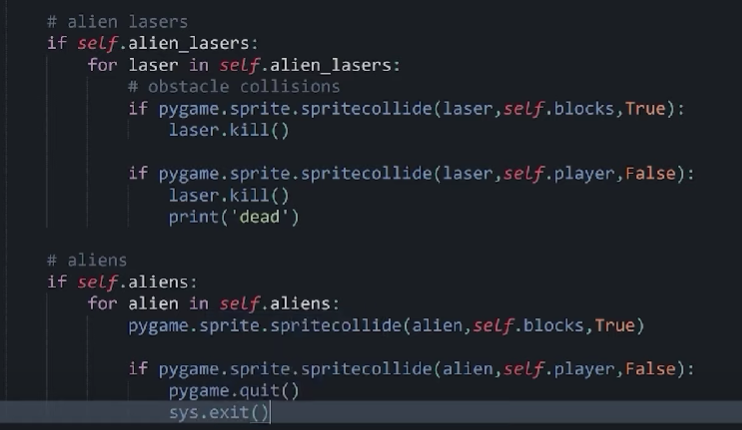
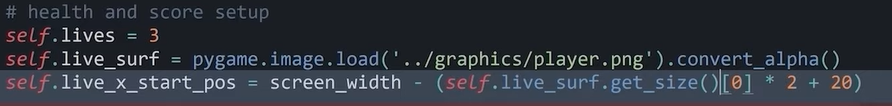


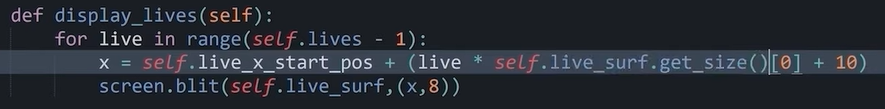
Fig: alien laser collision

## **Adding the health system:**

* Creating a surface for display in game class and We initialize the image.



* And then converting alpha.
* Setting offset to the right side of the screen for displaying all the lives at right side only
* Getting parameters x and y from the method get\_size.
* Setting x =0; and finally setting the offset as 20 for deciding how many players to show.
* Creating another method display lives
  + It doesn’t take any argument and run overs range of lives
  + It displays lives-1; adding the ability to top right i,e. 0 lives.
  + We need x and y positions, it’s values are initialized below:



* Finally, the display\_lives method is called in the run.
* If the live size is not greater than zero then we end the game and call quit.



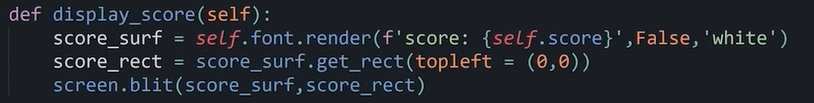
Fig:- Health System view

## **Adding the score System:**

* Before getting started with this; some changes needed to be done
  + Each alien gets a value.
  + We change the collision detection to identify which alien was hit.
  + Update our aliens



* Working on the init method again, setting score =0 and setting font size and size.
* Create a method called display\_socre in-game class; helps to display the score of the player.



* We initialize the image and rectangle for the screen.
* Display the zero at the top left of the screen.
  + Create a score surface and score rect, and set all the values to all the arguments.
  + then screen.blit is called and finally the method is called in run.

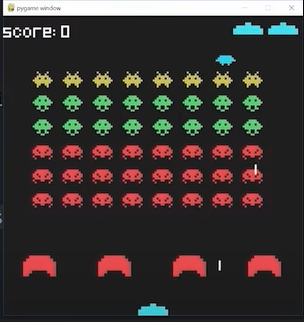
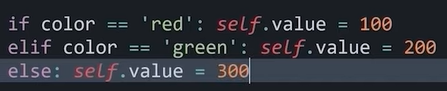
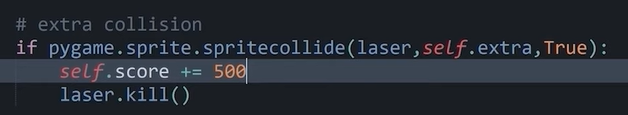


Fig:- Zero Score View

* Setting the offset of x as 10 and y as -10 to make the score look better.
* **Update Score**
  + Go to an alien file and give a score to each alien relative to their color.



* + Now go to check the alien and call the score method and make it 500 initially.



* + Basically, we are storing the hit; checking if it’s an alien’s hit, and copying it as a score.
  + Then simply if its aliens hit, add the number of aliens hit to score.

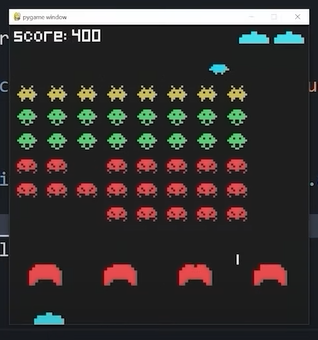


Fig:- Score Update view

## **Adding the CRT styling:**

* Create a class called crt using pygame.
* It’s not inherited from any class.
* Setting up init method
* We initialize image and rect for the CRT
* Created a draw function
* Creating crt as an instance of crt class
* It will draw tv style things around the screen
* Finally styling the screen so that a change in the size of the screen will not affect the moves.
* Setting alpha value, opacity and visibility of the object on the screen.
* Create crt\_lines and setting line\_amount and line\_height.
* Initializing width
  + For start point:-

Needed tuple (x,y) , setting x =0 and y in its position

* + For end point:-

Again needed tuple (x,y) , x will be here screen width and y will be again in its position.

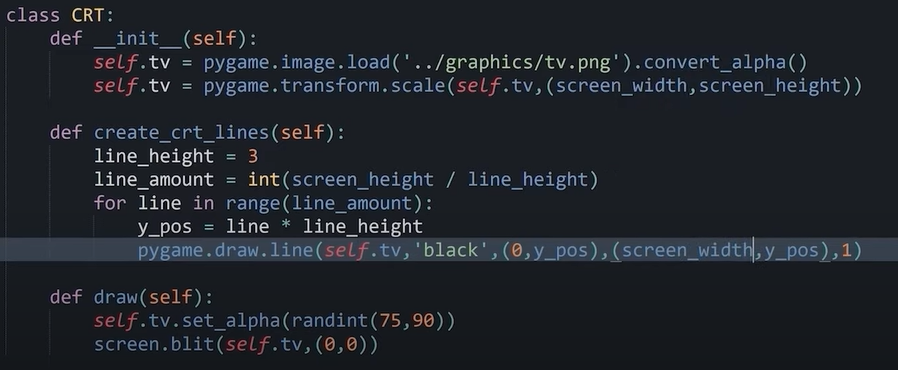
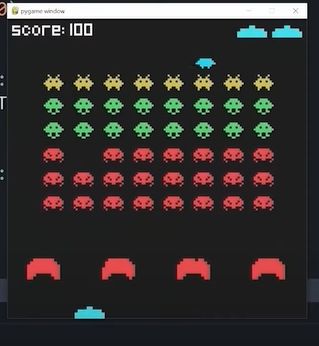
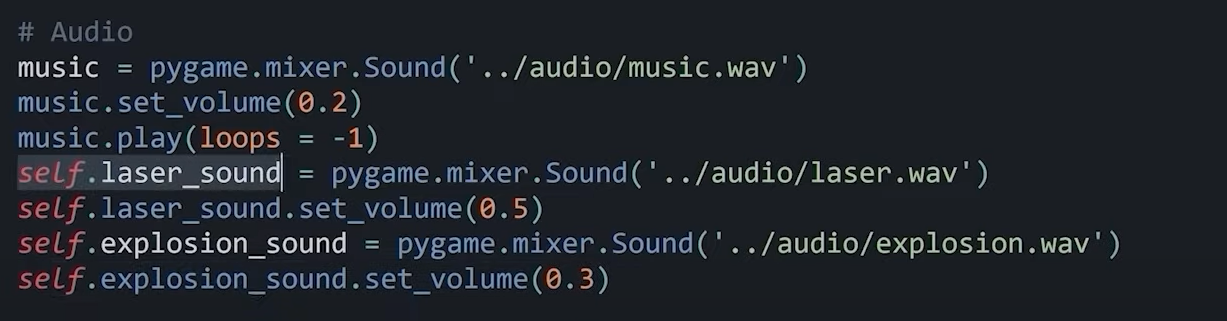
 

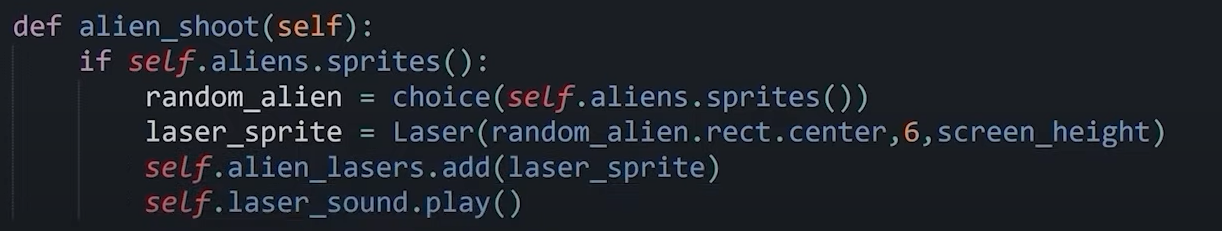
Fig: CRT Class Fig: Crt styling Output

## **Adding music and sounds:**

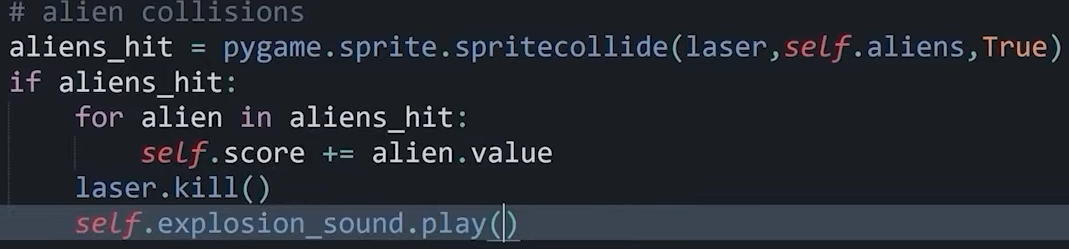
* Import all the audio needed to add in the background by using pygame.mixer.sound



* Setting the volume of this file.
* Play the music and set it at negative 1.
* Add the laser and explosion sound and play them in inappropriate spots.
* Laser sound play when an alien is killed.

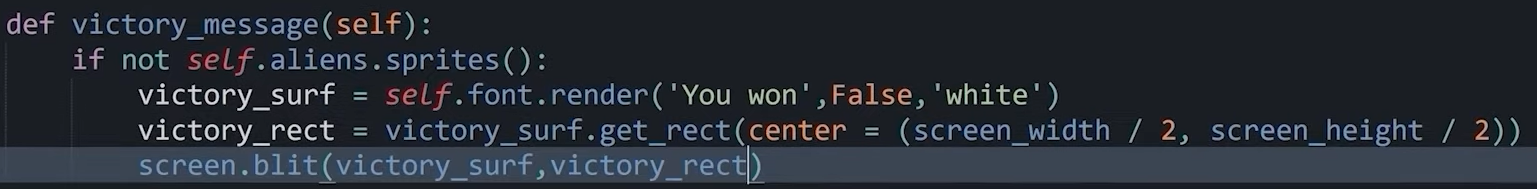


* explosion sound is played at collision check.



* Added laser sound will be played when we press the space button.

## **Adding the victory screen:**

* If no more aliens, show some victory text
* Add a method named victory\_message in game class.
* Doesn’t need any argument, just check if aliens remain or not.
* Creating victory surface, which initially is false and is of white color.
* Add a victory rectangle at the center of the screen.
* And finally, the method is called in the run class.

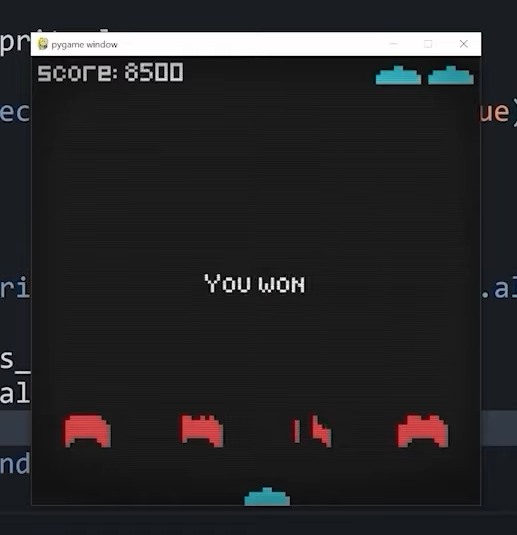


Fig:- Victory message

# 

# **Conclusion:**

If I were to summarize the entire experience of this third-year project in a word, it would be an adventure. Looking back to the past few months, I can truly assert it has been a journey that started slowly, without much progress due to the lack of technical knowledge required for this project but took off while making progress with the research.

I began the project with absolutely no idea on how graphics work, how a collision detection system works, or even how to display an image on the screen and, by the end of the project, I got to a point where we could create our own or modify existing game-specific algorithms to suit my needs. What is more, this was our first attempt at building a graphics-based game and I believe that achieving a single-player space shooter arcade game with a smart power­up module and AI for NPCs is a good start if I decide to go down the game development path.

# **References:**

[1] Pygame. URL: http://www.pygame.org/hifi.html. Last accessed: 04/04/2016

[2] The ESA 2015 Report. URL: http://www.theesa.com/wp­content/uploads/2015/04/ESA­Essential­Facts­2015.pdf. Last accessed: 05/04/2016

[3] Video Games Industry sales. URL: http://vgsales.wikia.com/wiki/Video\_game\_industry. Last accessed: 05/04/2016

[4] Video Game Industry sales in Japan. URL: http://vgsales.wikia.com/wiki/Video\_games\_in\_Japan. Last accessed: 05/04/2016

[5] Unity3D. URL: http://blogs.unity3d.com/2014/09/03/documentation­unity­scripting­languages­and­you/. Last accessed: 28/04/2016