Space Pirate

Project Summary and Design Report

Christopher Guerra, Marc Jimenez, David Tu

Table of Contents

[1 Introduction 2](#_Toc482118246)

[2 Design 2](#_Toc482118247)

[2.1 Rules 2](#_Toc482118248)

[2.2 Sources of Uncertainty 2](#_Toc482118249)

[2.3 How to Win 2](#_Toc482118250)

[2.4 Player Skills Prerequisite(s) 2](#_Toc482118251)

[2.5 Controls 2](#_Toc482118252)

[2.6 Game Duration 2](#_Toc482118253)

[2.7 Scoring 3](#_Toc482118254)

[2.8 Game State Flow Diagram (Tree) 3](#_Toc482118255)

[3 Software Architecture Detail 3](#_Toc482118256)

[3.1 Algorithms 3](#_Toc482118257)

[3.2 Software Organization 4](#_Toc482118258)

[3.3 Classes 5](#_Toc482118259)

[4 Game Demonstration 7](#_Toc482118260)

[5 Bibliography 17](#_Toc482118261)

# 1 Introduction

In this game, you will assume the role of a space pirate captain cruising along the galaxy when suddenly, you get attacked by other space pirates trying to steal your loot! As captain, it is your job to show who’s the best pirate in the whole galaxy by engaging in a little intergalactic warfare!

In this space pirate themed shooter game, you get the command a pirate ship and shoot at your enemies! Watch out though, there’s more than one fleet in the galaxy, so be prepared for anything!

# 2 Design

## 2.1 Rules

* There are three types of attacks that correspond to different colors: Red, Green, and Yellow
* There are three different types of enemies that correspond to those colors as well
* The weakness of every enemy is the determined by the color of the enemy. For example, a red-colored enemy is weak against red-type attacks from the player
* Exploiting your enemies’ weaknesses will defeat them and is the key to victory
* Every ship can only take one hit and that includes yours. Stay sharp and avoid your enemies’ maneuvers and their projectiles!
* After 1 minute, you would have survived the battle and you be declared the winner

## 2.2 Sources of Uncertainty

The enemy spawns are a source of uncertainty. This is due to the algorithm for spawning an enemy being based on the global variable, ALIEN\_ODDS. Also, the spawnRandomEnemy function randomly chooses one of the three enemy types to spawn, which is another source of uncertainty

Enemy bombing patterns are another source of uncertainty, since two of the enemy types will randomly drop bombs as they move through their pathing sequences

## 2.3 How to Win

To win the game, the player must last for 1 minute without getting hurt. If they do, then the player will lose and it’ll be game over

## 2.4 Player Skills Prerequisite(s)

Players are expected to be familiar with the keyboard layout since the game will be controlled from it

## 2.5 Controls

* 1 – 3 Keys: Switch to Green, Red, and Yellow Weapons, respectively
* SPACEBAR: Shoot with the equipped weapon
* LEFT: Move left
* RIGHT: Move right
* ESC: Exit the screen

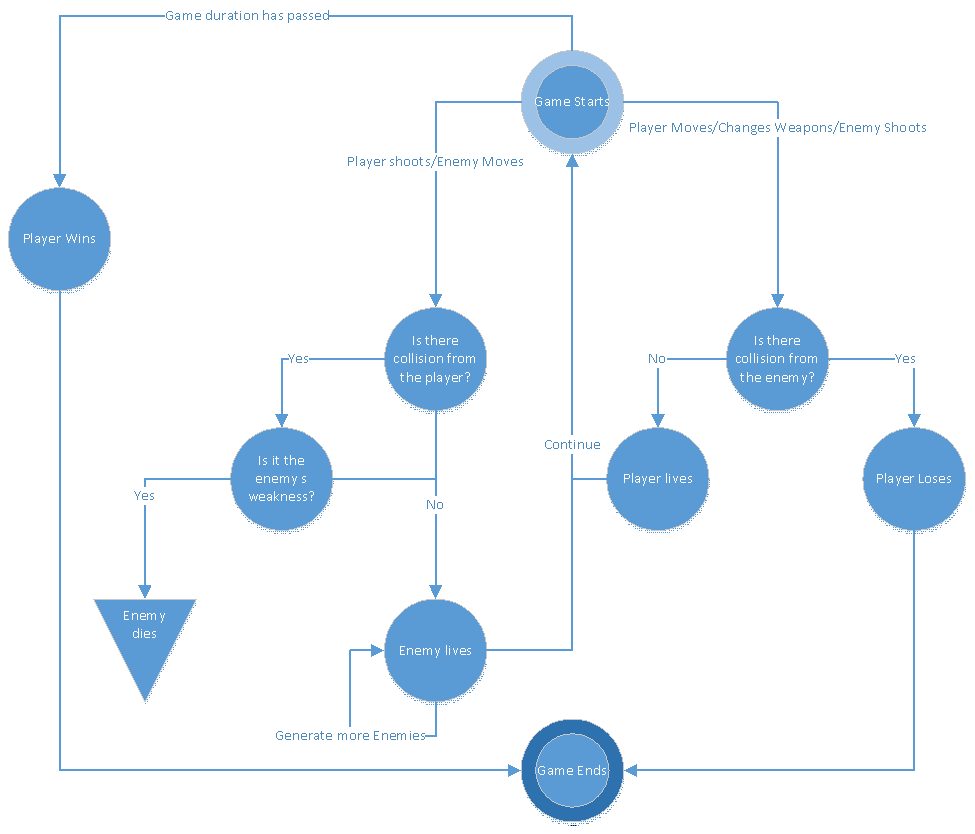
## 2.6 Game Duration

Battles will last for 1 minute in which afterwards, the player will be declared the winner

## 2.7 Scoring

Players will get 1 point added to their total score when they defeat an enemy

## 2.8 Game State Flow Diagram (Tree)



# 3 Software Architecture Detail

## 3.1 Algorithms

The algorithms required to make this game where mostly just simple control statements such as while and for loops

While loops where used to control entry into each game module. In this game, we have three: Title Screen, Game Loop, and Game Over Screen. When a player exits from one module, the current while loop he is in breaks due to the falsification of the while loop’s conditional statement and goes onto the next

For loops are used for collision detection for each type of collision. In general, they check for collisions between the player’s projectiles and the enemies, the player and the enemies themselves, and the player and the enemies’ projectiles. This is done by iterating through a list of game objects and checking for collisions with any members of another game object list

Similar loops are used for spawning enemy bombs. A loop iterates through a list of enemy objects and activates the function to spawn a bomb. The function itself compares the one of the variables in the enemy’s class definition to one of the global variables to determine if the enemy object shall shoot or not

For enemy spawning, the game iterates through based on a random range. Each iteration activates the spawn function, which randomizes a variable to determine which kind of enemy should spawn

## 3.2 Software Organization

Within the source code, our software’s logical flow is as follows:

Game setup: This part initializes the game, sets up the screen, defines game constants and variables, and prepares the game objects such as the backgrounds, sounds, the player and his enemies, game texts, and a timer

Title Screen Loop: This loop is then executed to render the title screen. While within this screen, the game checks if the player inputs ENTER or RETURN to start the game. If the player presses ESC, then the game will take the player directly to the game over screen

Game Loop: This loop is where the gameplay gets simulated and as a result, where most of the logic occurs. First, it checks for player input:

1. ESC to exit gameplay and go to the game over screen
2. LEFT or RIGHT keys to move the character
3. 1 – 3 keys to equip a Green, Red, and Yellow Weapon, respectively
4. SPACEBAR to shoot with the equipped weapon

Then, it updates the game state by performing calculations for rendering onto the next frame. It does things such as determining a sprite’s next position, whether it’s firing a weapon, and enemy spawning. Collisions are also calculated in this step. Finally, it renders everything on the screen based on the results of those calculations

Game Over Loop: This is the final part of the game that gets executed. While within this loop, it renders the game over screen and checks if the player presses ESC, which exits the application

## 3.3 Classes





















# 4 Game Demonstration

In this section, the most important aspects of the game will be discussed: Player movement/shooting, weapon changing, enemy movement/shooting/spawning, and collision detection

Player Movement: To implement movement we first need to determine the direction the player is going. While in the game loop, we can determine the direction with: direction = keystate[K\_RIGHT] – keystate[K\_LEFT]. Then we call the player function, move(), and pass the direction as a parameter. This will call the pygame function, move\_ip(), in order to move the player in the direction based on the player’s speed



Player’s starting position\*

****

Player moving right from the start position\*

Player Shooting: To implement shooting we first need to determine if there is player input (firing = keystate[K\_SPACE]). If there is, then we also need determine what kind of weapon the player should be firing. We can find that out by checking for the value of player1.weapon variable. If it is a ‘Green Projectile’, then a green projectile object will be constructed, passing along in its constructor the position of the player. This will then create a green projectile that will shoot vertically along the screen. A similar process occurs if player1.weapon is a ‘Red Projectile’ and ‘Yellow Projectile’ instead

****

Player shooting Green Projectiles\*

Weapon Change: Weapon change is an important part of the game since certain enemies can only die from certain weapon attacks. To implement the weapon change, within the game loop, we needed to determine if the player has pressed any of the 1 – 3 keys. If the 1 key is pressed then that changes the player1.weapon variable into ‘Green Projectile’ by using simple assignment. If 2 key is pressed, then by a similar process, player1.weapon changes into ‘Red Projectile’. The same applies for the ‘Yellow Projectile’, if the 3 key is pressed

****

Player switching to a Red Projectile and using it to shoot. Note that when the player switches weapons, the weapon icon on the top left corner changes\*

Enemy Movement: Enemy movement is classified within the three enemy classes we have made. Each enemy was meant to have different patterns. Green enemies will spawn at the top of the game screen and move left and right. Red enemies will spawn at the top of the screen and move down to the bottom of the screen to collide with the player and otherwise will move up when it reaches the bottom of the screen. Yellow enemies will spawn anywhere at the top of the screen and move diagonally within the top 2/3rds of the game screen



Note the initial positions of the red and green enemies\*



After some time, the red enemy has moved down, while the green on has moved right\*

Enemy Shooting: Enemy shooting will only occur within two types of enemies - green and yellow. This is to combine with their patterns as they will never directly collide with the player. Each instance of the enemy has their own internal timer. Due to our algorithm, there will be random period it will take for the enemies to shoot



Enemies shooting projectiles at the player. Their projectiles are the bombs with the skull design\*

Enemy Spawning: Enemies will spawn on a random timer when the game starts. Random in this case is a base timer plus random.randint. The game will also check how many enemies are currently alive to determine the number of enemies to spawn. If the number of enemies on the screen is below a certain number (in this case, four), the game will spawn either two or three enemies. Otherwise, the game will spawn one enemy. The function to spawn the enemy will also use random.randint in order to determine the type of enemy that will spawn



At initial startup, there were two enemies\*



After some time, more enemies have appeared\*

Collision Detection: Collision detection was implemented by checking if an object has intersected with any other object. This is done by using the pygame function, groupcollide, which requires two important parameters of the type, Group. Groups are simply pygame container types that pygame uses to manage their sprites. When groupcollide is invoked, it checks if each object within one group has intersected with any of the objects of another. If it has, then it returns a boolean indicating that a collision has occurred. With pygame’s groupcollide function, we simply put in multiple loops to apply to these cases: The player’s weapons kill the correct colored enemy, the red enemy killing the player, and the enemy’s bombs killing the player

****

Player colliding with an enemy projectile. The source of the collision is the explosion\*

\*Screenshots here reflect a work in progress

# 5 Bibliography

"Pygame Front Page." Pygame Front Page — Pygame V1.9.2 Documentation. Pygame, n.d. Web.

17 April 2017. <https://www.pygame.org/docs/>.

"Pygame.examples.aliens.main." Pygame.examples.aliens.main - Nullege Python Samples.

Pygame, n.d. Web. 24 April 2017.

<http://nullege.com/codes/search?cq=pygame.examples.aliens.main>.