A Level Project

Name: Konstantinos Doran

Centre:

Candidate Number:

Contents

[Analysis 6](#_Toc99355282)

[Introduction 6](#_Toc99355283)

[Computational Suitability 6](#_Toc99355284)

[Research 7](#_Toc99355285)

[Stakeholders 8](#_Toc99355286)

[Interviews 8](#_Toc99355287)

[Interview 1: 8](#_Toc99355288)

[Interview 2 10](#_Toc99355289)

[Interview Conclusion 11](#_Toc99355290)

[Requirements 13](#_Toc99355291)

[Specification 13](#_Toc99355292)

[Must have: 13](#_Toc99355293)

[Should have: 14](#_Toc99355294)

[Could have: 15](#_Toc99355295)

[Will not have: 15](#_Toc99355296)

[Hardware Requirements 16](#_Toc99355297)

[Software Requirements 16](#_Toc99355298)

[Design 17](#_Toc99355299)

[Interface 17](#_Toc99355300)

[Variables and Data Structures 20](#_Toc99355301)

[Class Diagrams: Game Class 20](#_Toc99355302)

[Class Diagrams: Player Class 20](#_Toc99355303)

[Class Diagrams: Player Bullet Class 21](#_Toc99355304)

[Class Diagrams: Enemy Bullet Class 22](#_Toc99355305)

[Class Diagrams: Sword Class 22](#_Toc99355306)

[Class Diagrams: Key Class 22](#_Toc99355307)

[Class Diagrams: Outside Wall Class 22](#_Toc99355308)

[Class Diagrams: Inner Wall Class 23](#_Toc99355309)

[Class Diagrams: Door Class 23](#_Toc99355310)

[Class Diagrams: Spikes Class 23](#_Toc99355311)

[Class Diagrams: Chest Class 24](#_Toc99355312)

[Class Diagrams: Sword Enemy Class 24](#_Toc99355313)

[Class Diagrams: Bow Enemy Class 25](#_Toc99355314)

[Class Diagrams: Boss Enemy Class 25](#_Toc99355315)

[Functionality 27](#_Toc99355316)

[Algorithms: Game Class: 27](#_Toc99355317)

[Algorithms: Player Class: 32](#_Toc99355318)

[Algorithms: Player Bullet Class: 38](#_Toc99355319)

[Algorithms: Enemy Bullet Class: 39](#_Toc99355320)

[Algorithms: SwordClass: 40](#_Toc99355321)

[Algorithms: Key Class: 41](#_Toc99355322)

[Algorithms: Outside Wall Class: 41](#_Toc99355323)

[Algorithms: Inner Wall Class: 41](#_Toc99355324)

[Algorithms: Door Class: 42](#_Toc99355325)

[Algorithms: Spikes Class: 42](#_Toc99355326)

[Algorithms: Chest Class: 44](#_Toc99355327)

[Algorithms: Sword Enemy Class: 45](#_Toc99355328)

[Algorithms: Bow Enemy Class: 48](#_Toc99355329)

[Algorithms: Boss Enemy Class: 49](#_Toc99355330)

[Test Plan 54](#_Toc99355331)

[Implementation 60](#_Toc99355332)

[Iteration 1 – Displaying All Classes: 60](#_Toc99355333)

[Game Loop: 60](#_Toc99355334)

[Game Class: 61](#_Toc99355335)

[Player Class: 67](#_Toc99355336)

[Player Bullet Class: 73](#_Toc99355337)

[Outer Wall Class: 74](#_Toc99355338)

[Inner Wall Class: 76](#_Toc99355339)

[Sword Class: 77](#_Toc99355340)

[Door Class: 79](#_Toc99355341)

[Spike Class: 81](#_Toc99355342)

[Chest Class: 83](#_Toc99355343)

[Sword Enemy Class: 84](#_Toc99355344)

[Key Class: 87](#_Toc99355345)

[Bow Enemy Class: 89](#_Toc99355346)

[Enemy Bullet Class: 93](#_Toc99355347)

[Errors 95](#_Toc99355348)

[Conclusion 99](#_Toc99355349)

[Iteration 2 – Collisions, Level Building and Menu: 100](#_Toc99355350)

[Menu: 100](#_Toc99355351)

[Game Loop Function: 104](#_Toc99355352)

[Game Class: 105](#_Toc99355353)

[Player Class: 114](#_Toc99355354)

[Player Bullet Class: 117](#_Toc99355355)

[Enemy Bullet Class: 118](#_Toc99355356)

[Door Class: 119](#_Toc99355357)

[Spike Class: 120](#_Toc99355358)

[Chest Class: 121](#_Toc99355359)

[Sword Enemy Class: 122](#_Toc99355360)

[Bow Enemy Class: 127](#_Toc99355361)

[Boss Enemy Class: 129](#_Toc99355362)

[Errors 138](#_Toc99355363)

[Conclusion 145](#_Toc99355364)

[Iteration 3 – Sprite sheets and Aesthetics 146](#_Toc99355365)

[Menu: 146](#_Toc99355366)

[Game Class: 149](#_Toc99355367)

[Spritesheet Class: 150](#_Toc99355368)

[Player Class: 151](#_Toc99355369)

[Player Bullet Class: 151](#_Toc99355370)

[Ground Class 152](#_Toc99355371)

[Enemy Bullet Class: 153](#_Toc99355372)

[Key Class 154](#_Toc99355373)

[Outer Wall Class: 154](#_Toc99355374)

[Inner Wall Class: 154](#_Toc99355375)

[Door Class: 155](#_Toc99355376)

[Spike Class: 155](#_Toc99355377)

[Chest Class: 156](#_Toc99355378)

[Sword Enemy Class: 157](#_Toc99355379)

[Bow Enemy Class: 157](#_Toc99355380)

[Boss Enemy Class: 158](#_Toc99355381)

[Errors 158](#_Toc99355382)

[Conclusion 160](#_Toc99355383)

[Testing 161](#_Toc99355384)

[Evaluation 216](#_Toc99355385)

[Specification 216](#_Toc99355386)

[Must Have: 216](#_Toc99355387)

[Should Have: 219](#_Toc99355388)

[Could Have: 220](#_Toc99355389)

[Usability Features 222](#_Toc99355390)

[Maintainability and Future Developments 223](#_Toc99355391)

[Essential Developments: 223](#_Toc99355392)

[Potential Developments: 223](#_Toc99355393)

[Optimisation Developments: 223](#_Toc99355394)

[Appendix 224](#_Toc99355395)

[Bibliography 224](#_Toc99355396)

[Complete Code: 225](#_Toc99355397)

# Analysis

## Introduction

Currently, there are many complex and over complicated forms of relaxation for students interested in playing games in their spare time. Many games have shifted away from a simple yet rewarding arcade-based system of reaching high scores and casual enjoyment into more competitive and violent activities. My current problem is that many people would like to enjoy the nostalgia of a simple arcade game with the benefit of portability.

Therefore, I have provided those interested in a classic dungeon crawler game that takes the nostalgic and casual consumers interests at heart. The basis of the dungeon crawler game is to accommodate casual players by having simple controls and instructions with very little time to become accustomed to the game and to achieve the highest score by proceeding through levels without dying and killing enemies that block your path.

## Computational Suitability

This solution is well suited to being adapted for a computer, especially a laptop, as it requires little hardware requirements, such as standard IO in the form of a keyboard and monitor. Furthermore, the game has been optimized to require little processing power (1 to 2 cores) and can store information locally. This allows users to enjoy the benefits of casual arcade-style games without the need for an internet connection or large and heavy arcade units.

The premise of these games is that they allow the user to become a knight and fight off monsters in a classic Role-playing style without the need for physical interaction other than with the user’s keyboard.

## Research



‘The Binding of Isaac’, is a 2- dimensional, top-down dungeon crawler game where the player controls the character through procedurally generated levels and follows a story, in which the character must fight monsters to continue to the next room. It includes an item system where the player can use equipment to increase the statistics of the character or change how the character can attack the enemies. The game does not accommodate for resuming levels when the game is running, therefore must restart every time the player finishes the game or loses. When the player completes the game, they do not store a score but instead give the player more items or characters based on the achievements they have completed. Thus, compelling the player to repeat the game until they unlock every character to complete every achievement.

Furthermore, the game incorporates a room system, where a ‘Boss’ enemy must be fought after 10 levels. As each level passes, the difficulty increases, and the player is given more abilities and items to choose from. The game does not utilise a mouse and only requires a keyboard to play as the directions that the player faces correspond to the last movement key pressed. Therefore, the player must press the left movement key before they can shoot a bullet left. This increases the difficulty for players to complete the game as it is a key area of the mechanics of the game that the player must learn to use before they will complete the game.



‘Soul Knight’ is another dungeon crawler game, however, has been designed to be used on mobile devices. It similarly uses a health-based system, however, uses a point system, coins which are totalled after the player dies. The game also uses procedural generation of levels, but it does not end until the player loses, with each level progressively increasing in difficulty. This game locks the player in each room until all the enemies have been defeated and presents the player with a chest containing additional items or abilities to be used in future levels. In addition to this, the player is restricted to two weapons which they can use different buttons to use both. Furthermore, weapons and customisations can be unlocked depending on how many points are earned by the player after losing. High scores are also stored remotely and compared on a leader board to other players.

On the other hand, ‘Soul Knight’ does not require the player to aim, as the weapons use an area of effect, that is that each weapon will interact with enemies in a radius rather than at a single point. As well as the area of effect, weapons that would require the player to accurately aim at the enemies automatically aim towards the nearest target without an input from the player, increasing the ease of use of the game as the input required to aim the player on a mobile device would be very difficult to program.

Comparing both these popular games has allowed me to extract the key features that gave them success: First, the sectioning and separating of rooms in to small and compact spaces with quick and fast-paced action has kept the engagement of players and is vital to maintain their interests. Further to this, they both use basic graphics to enable mass adoption on devices and use corresponding intuitive controls dependant on the device they are played on. In addition to this, both include a variety of weapons and mechanics that give the player more options as the difficulty of the game increases, allowing the player to play for longer.

## Stakeholders

My current stakeholders are a group of students aged between 17 and 18 that are interested in arcade-style videogames. They want a game that contains simple game mechanics and an easy to learn system of controls. Furthermore, my stakeholders have proposed the need for a high score system to compare their progress to each other. They would prefer a game that can be played remotely and without the need to have a connection to the internet. In addition to this, my stakeholders are interested in articulated level design that has been developed to both be challenging and fun.

# Interviews

## Interview 1:

First, I will interview two clients, Chris and Deniz, on their thoughts of this genre and style of videogames and the areas which they enjoy or dislike about this style. I will then ask my clients to elaborate on their opinions when asked certain questions.

1. Have you ever played a dungeon crawler game before?
2. What attributes did you like from the games you have played?
3. What attributes did you dislike from dungeon crawler games?
4. Do you enjoy the top-down view of dungeon crawler games?
5. Do you prefer the player to have a variety of items or just one item?
6. What features would you like implemented in a game of this genre?

**Chris:**

1. **Have you ever played a dungeon crawler game before?**

“Yes, I have played Diablo and The Binding of Isaac, both of which I enjoyed.”

1. **What attributes did you like from the games you have played?**

“I really enjoyed the fast-paced gameplay from fighting lots of enemies at once. I also enjoyed the very easy controls that allowed you to enjoy the game quickly.”

1. **What attributes did you dislike from dungeon crawler games?**

“Sometimes I found the rooms a bit too small and the enemies weren’t that difficult to defeat”.

1. **Do you enjoy the top-down view of dungeon crawler games?**

“I really enjoy the top-down view of dungeon crawler games compared to the scrolling style of view as it allows the player to see the entire level, therefore know how to approach the level properly.”

1. **Do you prefer the player to have a variety of items or just one item?**

“I prefer the player having at least two different weapons they can use.”

1. **What features would you like implemented in a game of this genre?**

“I would like to be able to unlock abilities each level to add some more variety rather than be stuck with the same attacks when picking a class at the beginning”.

**Deniz:**

1. **Have you ever played a dungeon crawler game before?**

“Yes, but I have only played one called Pixel Dungeon.”

1. **What attributes did you like from the games you have played?**

“I enjoyed facing different enemies as it adds variety and keeps me more interested while playing, especially if there is a Boss level where the player fights a very tough enemy as that adds more difficulty to the game.”

1. **What attributes did you dislike from dungeon crawler games?**

“Sometimes there are too many mechanics that over complicate how it is played. I really don’t like it when they add lots of different abilities and weapons as many are unnecessary or pointless.”

1. **Do you enjoy the top-down view of dungeon crawler games?**

“I like the scrolling camera view as it is more immersive as the player explores the level, but I don’t mind the top-down view.”

1. **Do you prefer the player to have a variety of items or just one item?**

“I’d rather the player have some variety in how they can play the game rather than just one.”

1. **What features would you like implemented in a game of this genre?**

“A lot of dungeon games don’t allow you to regain health without an ability and I feel that it makes it too hard, so a good feature to add would be to allow the player to regain health over time.”

## Interview 2

My second interview will more directly inform me on requirements for my game as I ask them about specific features they would like included:

1. How many times should the player attack the enemy in order to kill them?
2. Would you like an enemy that shoots bullets at the player?
3. Do you want multiple levels?
4. Should there be a boss level?
5. What should be the health comparison between an enemy and the player?
6. Would you like the player to be able to skip a level?
7. Would you want the player to unlock abilities as the game progresses?
8. Should there be a high score system?
9. Would you prefer to use both a keyboard and cursor control or just use a keyboard to play the game?
10. Should there be a menu before the game starts?

**Chris:**

1. **How many times should the player attack the enemy in order to kill them?**

“I think the player should be able to hit an enemy two or three times to kill them as the difficulty should be determined by how many enemies there are.”

1. **Would you like an enemy that shoots bullets at the player?**

“Yes, as it adds some more problems that the player needs to solve and could change how the player approaches the game.”

1. **Do you want multiple levels?**

“Absolutely, as the game would be very short if it only had one level. There should be 4 or 5 levels to begin with.”

1. **Should there be a boss level?**

“I think a boss level could be a good idea, as long as it isn’t too difficult.”

1. **What should be the health comparison between an enemy and the player?**

“I think the player should have around 100 health while the enemies can have half that of the player.”

1. **Would you like the player to be able to skip a level?**

“No, the player should be able to complete the level first if they think they can beat the proceeding level as they increase in difficulty.”

1. **Would you want the player to unlock abilities as the game progresses?**

“Yes, abilities could help if the levels begin to become too difficult with the starting items of the player and this could maintain a relatively small learning curve.”

1. **Should there be a high score system?**

“I believe that the high score system is not necessary, especially if it an offline game as it is only to compare with others and doesn’t add anything more to the gameplay.”

1. **Would you prefer to use both a keyboard and cursor control or just use a keyboard to play the game?**

“I think that being able to control with both a keyboard and mouse would be easier than using a trackpad and keyboard but if the idea of the game is for portability, then just a keyboard should be enough for the player to move around and aim at enemies.”

1. **Should there be a menu before the game starts?**

“Yes, the menu should have a start game button or exit game button.”

**Deniz:**

1. **How many times should the player attack the enemy in order to kill them?**

“The enemies shouldn’t be too hard to kill so maybe 1 hit up close or 2 hits from a far.”

1. **Would you like an enemy that shoots bullets at the player?**

“If an enemy can shoot bullets, then the player should be able to block them, but adding enemies that can shoot back will increase the variance of enemies and help the game be more challenging.”

1. **Do you want multiple levels?**

“Yes, the more levels the more fun the game will be.”

1. **Should there be a boss level?**

“A boss could make the game more repeatable, especially if it is difficult as it will encourage the player to spend longer trying to beat the game.”

1. **What should be the health comparison between an enemy and the player?**

“The player should have 3 times the health of the enemy and the enemy should take lots of hits to kill the player.”

1. **Would you like the player to be able to skip a level?**

“No, the game should be continuous and not have breaks in between them.”

1. **Would you want the player to unlock abilities as the game progresses?**

“Yes, but the player should start with some abilities and maybe have different abilities in each level.”

1. **Should there be a high score system?**

“Yes, as it encourages the players to play the game more and get the highest score but at the same time a leader board is not compulsory, maybe just what the high score is.”

1. **Would you prefer to use both a keyboard and cursor control or just use a keyboard to play the game?**

“I would prefer to use a keyboard as I use a laptop a lot and it is very difficult to aim using the keyboard and trackpad at the same time.”

1. **Should there be a menu before the game starts?**

“Yes, as it is convenient to not immediately start the game as soon as it loads.”

# Interview Conclusion

What did you gain from the interviews which will inform you game design?

Both these interviews have already determined the main aspects and design for my game: Both interviewees have expressed their similar thoughts and interests for the use of a top-down view and incorporate a variety of enemy classes and abilities. Further to this, my clients would prefer a simple and small learning curve as well as the inclusion of a boss level and varying the difficulty as the levels progress. From these insights into my clients’ thoughts and views I have developed a set of requirements based upon these interviews that lay within the interests of my stakeholders:

# Requirements

## Specification

### Must have:

|  |  |  |
| --- | --- | --- |
| Number | Requirement | Justification |
| 1 | A menu screen where you can choose to start the game or quit the window. | To allow the user to select when to begin playing the game and exit the game. |
| 2 | The player can move up, left, down or right using w,a,s,d keys. | This allows the player to move around each level and avoid enemies, allowing them to proceed to the next level. |
| 3 | Walls should block all bullets from the enemy or player. | To optimise the program to run more optimally and remain the immersive environment of the game. |
| 4 | The player can hold the space key to use his sword for 1 second. | To allow the player to inflict damage on the enemies, which is the main objective of the game. |
| 5 | The enemies are split into two classes, sword, or bow. | To add increased variety to the enemies in each level as requested by the clients. |
| 6 | The player can shoot bullets using the E key. | To add variety to the player’s attacks and to allow the player to inflict damage on enemies. |
| 7 | The player has 100 health. | This is to allow the player to die to an enemy once it runs out of health so that it is possible to lose. |
| 8 | Enemies have 40 health each. | This is to allow the enemies to only require two hits in order to die. |
| 9 | Enemies can attack the player once per second | This is to make it easier for the player as the player would lose health extremely quickly if there was no timer. |
| 10 | The player’s health goes down when attacked by an enemy | This is to allow the player to lose health when attacked and this is to make the player be able to die and therefore lose the game. |
| 11 | The game ends when the player has 0 health, and the player object is removed from the player group. | This is to allow the game to end when the player group has zero elements. |
| 12 | Every enemy killed should increase the score. | This is to allow a score to be tallied to achieve high scores. |
| 13 | Every key collected should increase the score and when all the keys for each room are collected then the doors can open. | This is to prevent player’s skipping a level and avoiding the enemies. |
| 14 | Doors prevent the player from leaving the room without completing it. | This is to prevent level skipping. |
| 15 | 5 levels with each level having more enemies. | This is to increase the length of the game and increase its difficulty per level. |
| 16 | Bow enemies shoot bullets that harm the player but cannot chase the player when in range. | This is to add variety to how the enemies can attack the player. |
| 17 | Sword enemies can chase the player when in range. | This is for the enemies to get close to the player to collide with them and attack the player. |
| 18 | Player can block bullets with his sword or other bullets. | This is to make it easier for the player to avoid taking damage and losing. |
| 19 | Player’s sword attacks all enemies within its radius. | This is to make it easier for the player to attack multiple targets. |
| 20 | Player’s bullets travel in the direction of the player’s last key pressed. | This is to allow the player to shoot bullets in different directions. |
| 21 | Enemy bullets are shot in the direction of the player from the enemy’s position. | This is to add variety to how enemies can attack the player and increase the difficulty of the game. |
| 22 | The 5th level is a boss level. | This is to add more difficulty and repeatability into the game and was requested by my clients. |
| 23 | The boss is larger and deals more damage. | This is to make the boss visually intimidating and increases its difficulty as it is harder to avoid. |
| 24 | The boss has multiple stages of attacks. | This is to increase the difficulty and add more variety to the boss level. |
| 25 | When the boss is killed, the game finishes. | This is to allow the player to finish the game and obtain a score. |
| 26 | The score increases when bullets are blocked using the player’s sword or bullets. | This is to allow more ways for the player to increase their score to achieve a high score. |
| 27 | The player can shoot one bullet per second. | This is to prevent the player from killing enemies too quickly and increases its difficulty |

### Should have:

|  |  |  |
| --- | --- | --- |
| Number | Requirement | Justification |
| 28 | Spikes, which extend or close every 4 seconds which deal damage to the player when stepped on and active. | Increasing the variety of enemies and difficulty as each level progresses. |
| 29 | A high score file that stores the highest score and changes the high score if a larger score is reached. | This is to maintain a create a high score system where only the highest achieved score is stored. |
| 30 | The boss spawns other enemies when low health. | This is to add more stages to my boss and increase its difficulty. |
| 31 | The boss has its own health bar to inform the player how much damage has been inflicted. | This is to inform the player on the health of the boss to make it easier for the player. |
| 32 | The player has a health bar which changes size depending on the amount of health the player has. | This is to help the player know how much health they have. |
| 33 | Chests spawn when each level is completed to unlock a new ability for the player. | This gives the player a new ability each level, which has been requested by my clients. |
| 34 | The player can increase his health after completing a level. | This makes it easier for the player to progress in each level. |
| 35 | The player can increase his sword’s radius after completing a level. | This makes it easier for the player to attack many enemies. |
| 36 | The player can unlock the ability to shoot many bullets every second after completing a level. | This makes it easier for the player to attack enemies and gives them a new ability. |
| 37 | The player cannot shoot until it unlocks the ability after the first level. | This is to add ability progression into the game and adds some difficulty to the first level. |
| 38 | The player can travel to previous levels without the level resetting. | This is to allow the player to travel back a level if they wanted to if I added a different level layout. |
| 39 | The player regains health every 10 seconds. | This is to add passive healing that was requested by one of my clients. |
| 40 | A game over screen that displays the highest score. | This is to inform the player of their score and the high score. |

### Could have:

|  |  |  |
| --- | --- | --- |
| Number | Requirement | Justification |
| 41 | Add images to each class and to give the game a theme, for example, the player is a knight and the enemies are skeletons. | If I have more time, then I could implement more sprite sheets but maintain the same class and add a background. |
| 42 | Main Menu should include text on the side about the controls | The displayed main menu should include information on how to move, shoot and attack with the sword to help the user learn the controls of the game. |
| 43 | Leader board of high scores with names of players. | Given more time I could improve the high score system to store both a name and high score after each game. |
| 44 | A game over screen that allows the player to input a name and save their score. | Given more time, I could improve the game over screen to include this however this is not necessary for my clients. |
| 45 | Use line of sight to allow the enemy to only attack if they can see the player | This improves the behaviour of the enemies to appear more realistic and would be a quality-of-life change if I had enough time. |

### Will not have:

|  |  |  |
| --- | --- | --- |
| 46 | A multiplayer game that is hosted on a server. | This game would not be suited for multiplayer as it would be too easy if there were two players, and the idea is to progress as far as possible without help. |
| 47 | Scrolling Camera | As both my clients have shown their preference for a top-down view instead. |
| 48 | Gravity as this game is not a platformer, therefore the player does not jump over walls or enemies. | This is because jumping would not be a useful feature in my game as it is designed with a different genre and style of play. |

## Hardware Requirements

**A computer capable of running the software, with standard IO peripherals** – The software will need a capable and fast processor. Most laptops and desktop computer bought after 2016 are capable of this. This includes minimum of 4GB of RAM and a single core CPU is necessary to run this game as the RAM is required to store the images and files that are used when the program is running. A keyboard is required to move and shoot, and a mouse is required to click on the buttons within the game, a 1920 x 1080 display is also required as the window size is 1200 x 1000 pixels large.

## Software Requirements

**Windows, Linux or Mac operating systems** – These are the operating systems supported by Python.

**Python interpreter** – The code will be written in Python; therefore, a Python interpreter is needed.

**Pygame for Python** – This is a library for python that is required for the software to execute.

# Design

## Interface

**Main Menu Screen:**

This is the Main Menu design. It includes three buttons, the first button calls the game function, the second calls another menu function and the final button exits the window. This fulfils the **first point in my success criteria**, to create a Menu function with buttons that start the game or exit the window. Furthermore, added features such as a title centred horizontally, and a background image increase the appeal of the Main menu screen.

Graphical user interface, application

Description automatically generated**Game Screen:**

My game screen incorporates **success criteria number 32**, which is to include a health bar for the player on the right-hand side of the screen. I have also included the player’s score and the number of keys the player, to inform them of how well they are progressing and how many more keys they must collect before progressing to each level. The health bar of the player will use the player’s current health attribute to obtain the ratio between the player’s maximum health attribute and the current health to change the width of the health bar. Further to this, both the score and the key attributes will be taken from the game class and player class respectively and displayed on the right-hand side of the screen.

**Game Over Screen:**

The Game over screen incorporates **success criteria number 40**, to draw a game over screen when the player loses the game and displays the high score. This function displays a background image, with the title Game Over as well as the score achieved by the player in this game and the highest score achieved by anyone that has played the game. The score attribute will be pulled from the game class after the player’s health reaches zero and the high score value will be obtained as it is a attribute of the game class that has been allocated the number within the high score text file.

## Variables and Data Structures

### Class Diagrams: Game Class

The Game class has been created as it provides modularity to the program. The game class is vital as it controls all the collisions, levels and what is displayed in its methods. It is used to handle any event, such as a key press or mouse click and is used to update all objects, change each level by removing and adding objects and store the data for each individual level, including the layout and score for each level. Score variable is used to store the total score, the level is used to change between levels, every level exists in a list, with each level being a 2D list that forms a grid of the screen. Load data is used to find the highscore file, level setup and delete are used when changing to a new level, eventprocess checks if the game has been quit and runlogic is used to handle every collision. Display is then used to show everything on the screen.

|  |
| --- |
| **Class: Game** |
| **Attributes:**  score: int  level: int  game\_over: boolean  levels: 1D array  levelcomplete: 1D list  chest\_unlocked: 1D list  level1: 2D list  level2: 2D list  level3: 2D list  level4: 2D list  level5: 2D list  player: object |
| **Methods:**  load\_data()  levelsetup()  leveldelete()  eventprocess()  getscore()  runlogic()  display() |

### Class Diagrams: Player Class

The player class is vital as it is used to create a player object. This class has variables to control how fast the player can move across the screen, the initial and maximum health. The size and colour of the health bar as well as the player score for each level, the size of the player, his direction and number of keys collected for each level as well as Boolean variables used to give the player more abilities after each level.

|  |
| --- |
| **Class: Player** |
| **Attributes:**  Speed\_x: int  Speed\_y: int  Current health: int  Max health: int  Health bar length: int  Target health: int  Health ratio: int  Heat bar color: tuple  Health change speed: int  Score: int  width: int  height: int  Gamekeys: int  directionx: int  direction: int  canshoot: boolean  multishot: boolean  swordradius: boolean  bulletcount: int |
| **Methods:**  Gethealth()  Getdamage()  Healthbar()  Getpos()  Changespeed()  Keyspressed()  Move()  update() |

### Class Diagrams: Player Bullet Class

The player bullet class is used to create an object which the player can shoot at enemies. As such it requires an x and y velocity and also requires variables for its size. Further to this it has a move procedure to allow it to travel across the screen and a collide procedure to handle collisions with other objects.

|  |
| --- |
| **Class: Player Bullet** |
| **Attributes:**  Speedx: int  Speedy: int  Width: int  height: int |
| **Methods:**  Move()  Collide() |

### Class Diagrams: Enemy Bullet Class

The enemy bullet class is used by the enemies to be able to shoot an object towards the player object. Thus it has speed and size attributes, and a move and collide method in order to allow the object to move across the screen and collide with other objects.

|  |
| --- |
| **Class: Enemy Bullet** |
| **Attributes:**  Speedx: int  Speedy: int  Width: int  height: int |
| **Methods:**  Move()  Collide() |

### Class Diagrams: Sword Class

The sword class is used to create an object of a sword to be used by the player object, this will allow the player to inflict damage on enemies and is necessary in order to give the program the intended functionality. It has a radius and centre attributes which are used to position the sword at the player’s location on the screen and set how large the sword can reach. It has one method which is used to handle collisions with other objects.

|  |
| --- |
| **Class: Sword** |
| **Attributes:**  Radius: int  Centre: vector |
| **Methods:**  Collision() |

### Class Diagrams: Key Class

The key class is used to create key objects when an enemy dies. This provides the player with an objective in order to progress to the next level. As such the object requires attributes to control its size and a method to handle collisions with other objects.

|  |
| --- |
| **Class: Key** |
| **Attributes:**  width: int  height: int |
| **Methods:**  Collision() |

### Class Diagrams: Outside Wall Class

The wall class is used to create objects which, through collisions, prevent all objects from passing through them. These objects are used to prevent objects from leaving the screen and keep each level completely encapsulated. Thus, it only requires attributes to describe its heigh and position on the grid.

|  |
| --- |
| **Class: Outside Wall** |
| **Attributes:**  width: int  height: int  positionx: int  positiony: int |
| **Methods:**  Getpos() |

### Class Diagrams: Inner Wall Class

The inner wall class is identical to the outer wall class as it inherits all the attributes and methods from its parent class. This class is separate as it could provide other features in future development such as a destructible environment for the player.

|  |
| --- |
| **Class: Inner Wall** |
| **Attributes:**  width: int  height: int  positionx: int  positiony: int |
| **Methods:**  Getpos() |

### Class Diagrams: Door Class

The door class is used to create an object that can only be passed through if the player has competed the level. This requires attributes to describe its size on the screen and has a single method to prevent other objects from passing through it unless the level is complete.

|  |
| --- |
| **Class: Door** |
| **Attributes:**  width: int  height: int  positionx: int  positiony: int |
| **Methods:**  Collision() |

### Class Diagrams: Spikes Class

The spikes class is used to create a spike object which damages the player object when the spike is “up”. As such it requires a Boolean attribute to determine the action of the spike object if there is a collision. The spike also has attributes to determine its size on the screen and has 3 methods, the first to check for collisions with other objects, the second to alternate between the two modes of the spike and the third to inflict damage on the player object if it is colliding with the spike and is up true.

|  |
| --- |
| **Class: Spikes** |
| **Attributes:**  Is\_up: boolean  Position: vector  width: int  height: int |
| **Methods:**  Collision()  Retracting()  Get\_damage() |

### Class Diagrams: Chest Class

The chest class is used to create a chest object that displays text on the screen and changes one of the player’s attributes. The level attribute distinguishes which player attribute is changed and width and height dictate the size of the chest. It has a collision method to handle collisions with the player object.

|  |
| --- |
| **Class: Chest** |
| **Attributes:**  Level: int  Width: int  Height: int |
| **Methods:**  Collision() |

### Class Diagrams: Sword Enemy Class

The sword enemy class is used to create an object for one type of enemy. It has a health attribute to dictate how many times the player needs to hit the enemy to kill it. It has a direction, speed and move attributes which are used to control the movement of the sword enemy on the screen. The changespeed and move methods are used to move the enemy object on the screen and the collision method is used to handle collisions with other objects. It also has a method that determines if the player is close to it to determine what function is called and has a movetoplayer procedure that tells the enemy to move towards the player when is close is true. The health methods are used when the enemy is attacked by the player to alter the enemy’s health.

|  |
| --- |
| **Class: Sword Enemy** |
| **Attributes:**  health: int  direction: int  speed\_x: int  speed\_y: int  move: int  width: int  height: int |
| **Methods:**  Changespeed()  Collision()  Is\_close()  Move()  Getpos()  Movetoplayer()  Gethealth()  Sethealth() |

### Class Diagrams: Bow Enemy Class

The bow enemy class is used to create an object of the bow enemy type. It has a health attribute to distinguish how many hits the player needs to inflict to kill it, a direction attribute to dictate where it will shoot the bullets and attributes used to change its size. It has an is close function returns a Boolean value that changes how the bow enemy behaves depending on its proximity to the player object. The bow enemy class also has a shoot function that creates an enemy bullet object and collision and health methods used to handle collisions and adjust the enemy’s health.

|  |
| --- |
| **Class: Bow Enemy** |
| **Attributes:**  health: int  direction: int  width: int  height: int |
| **Methods:**  Is\_close()  Shoot()  Collision()  Getpos()  Gethealth()  Sethealth() |

### Class Diagrams: Boss Enemy Class

The Boss enemy class is used to create an object for the final enemy of the game. It shares both the sword and bow enemy methods and attributes and also has health bar colour and size to control the size of the health bar above it on the screen. The boss enemy has damage and health procedures which are used to handle collisions with the player and inflict damage on the player and also has a health bar procedure used to control the health bar of the boss enemy.

|  |
| --- |
| **Class: Boss Enemy** |
| **Attributes:**  Current health: int  Max health: int  Health bar length: int  Target health: int  Health change speed: int  Health bar color: tuple  Health ratio: int  Bosscount: int  direction: int  width: int  height: int |
| **Methods:**  Is\_close()  Collision()  Getpos()  Shoot()  Createenemy()  Gethealth()  Sethealth()  Healthbar()  Getdamage() |

## Functionality

### Algorithms: Game Class:

#### load\_data:

The load data method within the game class is used to load the high score text file and read the value stored within the file, saving this to an attribute within the game class. If no integer value is found within the high score file, then it will set the high score attribute within the game class to zero. This fulfils success criteria 29 as this reads the high score from the text file and is then used within the game class. The flowchart below demonstrates how the method works:

Diagram

Description automatically generated

#### levelsetup:

The levelsetup method is a very complex method that is run as the game object is instantiated. It checks if the level is a multiple of 5 (if it is then it is a boss level), and if not then creates two times the level number plus 1 number of enemies that are given random coordinates on the grid. The level setup is used to accomplish **success criteria 15** as it is used to generate each level correctly. It is represented by the following pseudocode:

function levelsetup():

count = 0

bosscount = 0

if level != 5:

enemies = 0

while enemies != (2 \* level + 1):

xpos = randomint(1,23)

ypos = randomint(1,23)

if levels[level, xpos, ypos] != wall:

levels[level, xpos, ypos] = enemy

enemies = enemies + 1

else:

levels[level, 10, 10] = bossenemy

endif

for j from 0 to 24:

for i from 0 to 24:

index = levels[level,j,i]

Case based on index

Case == wall:

wall = Wall(40,40,i\*40,j\*40)

Case == enemy:

temp = randomint(1,2)

if temp MOD 2 == 0:

enemy = BowEnemy(40,40,i\*40,j\*40)

else:

enemy = SwordEnemy(40,40,i\*40,j\*40)

endif

Case == door:

door = Door(40,40,i\*40,j\*40)

Case == spike:

Spike = Spikes(40,40,i\*40,j\*40)

Case == bossenemy:

bossenemy = BossEnemy(40,40,i\*40,j\*40)

End Case

endfunction

#### leveldelete:

The leveldelete method removes all the objects currently being used in a level but the player. This is to allow a new level to be created using the levelsetup() function every time a player moves to the next level. This helps fulfil **success criteria 15**, in order to include multiple levels. This is shown by the pseudocode below:

function leveldelete():

all\_objects\_group.empty()

all\_objects\_group.add(player)

end function

#### eventprocess:

The eventprocess method’s only task is to check if the event of the window cross being clicked has occurred and if true, then exit game and close the window. This is concisely represented by the following pseudocode:

function eventprocess():

if event == windowclosed:

return True

else:

return False

endif

endfunction

#### getscore:

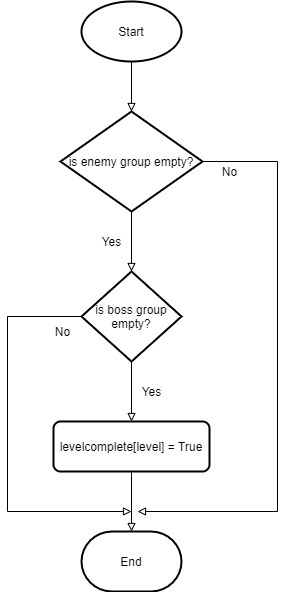
This method helps maintain the current score in the game and completes **success criteria number 29** as it is called in order to compare the score to the highscore attribute within the game class. This is represented by the pseudocode below:

function getscore():

return score

endfunction

#### runlogic:

This method is always checked every frame to see if the player group is empty, or the player is past certain boundaries and if the enemy or boss group are empty and runs following functions as a result. As such this method has been split into 3 different flowcharts, each representing a different requirement. The first flowchart accomplishes **success criteria 13**:

Diagram

Description automatically generatedThe second flowchart fulfils **success criteria 33**:

**Diagram

Description automatically generated**The third flowchart fulfils **success criteria 11 and 38**:

#### display:

The display method updates the screen while the game is running and when the game has finished. It accomplishes **success criteria number 32 and 40** and helps inform the player on how many keys they have, therefore also helps fulfil **success criteria number 13**. This is represented by the following pseudocode:

function display():

display.fill(BLACK)

if game\_over == True:

display(Background\_image)

display(“GAME OVER”,72,centre=(screenwidth//2, screenheight//2))

display(“SCORE:” + str(score),48, centre=(screenwidth//2, 500))

display(“HIGHSCORE: + str(highscore), 48, centre=(screenwidth//2), 700)) elseif game\_over == False:

display(“SCORE: + str(score), 28, centre=(1050,500))

display(“KEYS: + str(keys), 28, centre=(1050,600))

display(str(health), 28, centre=(1050,50))

display(player.health\_bar)

display(all\_objects\_group)

display.update()

endif

endfunction

### Algorithms: Player Class:

#### Gethealth:

This method is used to accomplish **success criteria 39** and increases the player’s current health attribute by the set amount specified in the function until it is at full health. This is shown in the pseudocode below:

function Gethealth(amount):

target\_health = current\_health + amount

if target\_health < maximum\_health:

current\_health = target\_health

elseif target\_health > maximum health:

current\_health = maximum health

endif

endfunction

#### Getdamage:

This method is used to subtract the player’s health by a specified amount when they are attacked by either an enemy or a bullet. Thus, it accomplishes **success criteria number 10, 16 and 23**. This is shown in the pseudocode:

function Getdamage(amount):

target\_health = current\_health - amount

if target\_health > 0:

current\_health = target\_health

elseif target\_health < 0:

current\_health = 0

endif

endfunction

#### Getpos:

This method is used to find the position of the player object which is then used in other methods such as in the BowEnemy class or SwordEnemy class to shoot bullets towards the player or to move towards the player. Thus, it is important for **success criteria number 17 and 21**. This is shown in the following pseudocode:

function getpos():

return (player.x, player.y)

end function

#### healthbar:

This method is used to create a health bar that displays the player’s health. It uses 3 rectangles, the first is an outline around the other two. The second is the main health bar of the current health, and the final is the transition health bar that changes length and connects to the main health bar when health is gained or lost. The main health bar changes colours depending on how high the player health is and all together, they satisfy the **success criteria 32 and 34**. This is described by the following pseudocode:

function health\_bar():

transition\_width = 0

transition\_color = RED

if current\_health < target\_health:

current\_health = current\_health + health\_change\_speed

transition\_width = target\_health-current\_health//health\_ratio

transition\_color = GREEN

elseif current\_health > target\_health:

current\_health = current\_health – health\_change\_speed

transition\_width = targethealth-current\_health//health\_ratio

transition\_color = YELLOW

endif

if current\_health >= 70:

health\_bar\_color = GREEN

if current\_health >= 40 AND < 70:

health\_bar\_color = ORANGE

if current\_health < 40:

health\_bar\_color = RED

endif

health\_bar\_width = current\_health//health\_ratio

health\_bar = rectangle(1005, 45, health\_bar\_width, 25)

transition\_bar = rectangle(health\_bar.right, 45, transition\_width, 25)

display(health\_bar\_color, health\_bar)

display(transition\_color, transition\_bar)

display(WHITE, (1005, 45, health\_bar\_length, 25), 4)

endfunction

#### Changespeed:

This is a basic method that is key to how the player can move, it simply takes the inputs for x and y and adds them to the player’s speed in either direction. This helps fulfil **success criteria number 2** and is show in the pseudocode below:

function Changespeed(x,y):

speed\_x = speed\_x + x

speed\_y = speed\_y + y

endfunction

#### Keyspressed:

This is a vital method as it checks for what keys are being pressed and therefore handles all events within the player class. Thus, it fulfils **success criteria number 2,4,6,20 and 27**. This is shown in the following pseudocode:

function Keyspressed():

key = Key.get\_pressed()

if key[K\_a] == True:

Changespeed(-2,0)

direction\_x = -6

direction\_y = 0

elseif key[K\_d] == True:

Changespeed(2,0)

direction\_x = 6

direction\_y = 0

elseif key[K\_w] == True:

Changespeed(0,-2)

direction\_x = 0

direction\_y = -6

elseif key[K\_s] == True:

Changespeed(0,2)

direction\_x = 0

direction\_y = 6

elseif key[K\_e] == True:

if canshoot == True:

if multishot == False:

if bulletcount > 0:

currenttime = clock.time()

if currenttime – previoustime > 1000:

bullet = Bullet(direction\_x, direction\_y)

bulletcount = bulletcount -1

previoustime = currenttime

endif

endif

else:

if bulletcount > 0:

currenttime = clock.time()

if currenttime – previoustime > 100:

bulletcount = bulletcount – 1

bullet = Bullet(direction\_x, direction\_y)

previoustime = currenttime

endif

endif

endif

endif

elseif key[K\_SPACE] == True:

if len(sword\_group) == 0:

currenttime = clock.time()

if currenttime – previoustime > 1000:

sword = Sword(swordradius)

previoustime = currenttime

endif

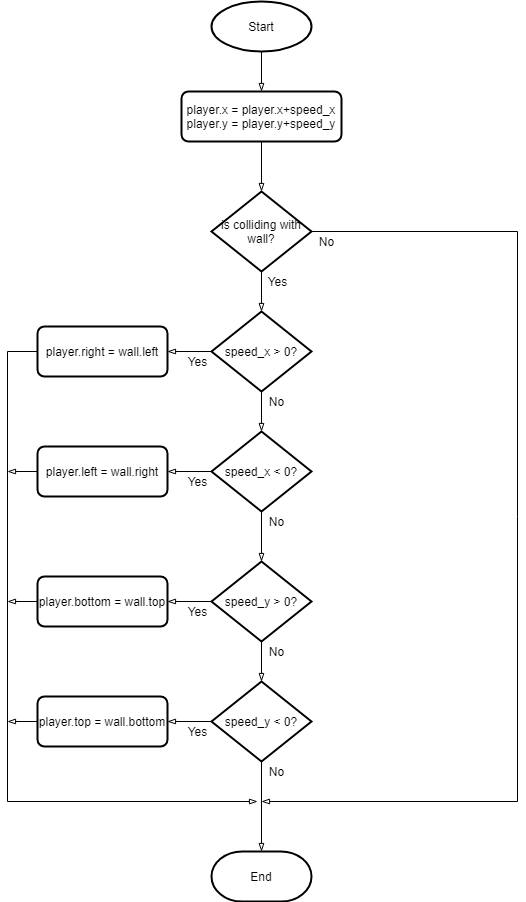
endif

endif

endfunction

#### Move:

This method is used to allow the player to move in a direction as it takes the parameters speedx and speedy and uses these to change the player’s position in the game. It also handles collisions when the player collides with a wall object. It therefore fulfils **success criteria number 2 and 3**. This is shown in the following flowchart:



#### update:

This method combines most of the key methods from the player class into one. It calls the move method and checks whether the player has zero health as well as increasing the player’s current health every 10 seconds. The method accomplishes **success criteria number 11, 36, 37 and 39**. It is shown by the following pseudocode:

function update():

Keyspressed()

currenttime = clock.time()

if currenttime = previoustime > 1000:

if multishot == False:

bulletcount = bulletcount +1

else:

bulletcount = bulletcount + 3

endif

previoustime = currenttime

endif

if bulletcount > 3:

bulletcount = 3:

endif

health\_ratio = maximum\_health//health\_bar\_length

healthtime = clock.time()

if healthtime – previoushealthtime > 10000:

Gethealth(10)

previoushealthtime = healthtime

endif

if current\_health == 0:

score = score + 100

die()

endif

move(speed\_x, speed\_y)

speed\_x = 0

speed\_y = 0

endfunction

### Algorithms: Player Bullet Class:

#### Move:

This method is used to move the bullet object in a set direction at a constant rate. This is used to accomplish **success criteria number 20**. This is shown in the following pseudocode:

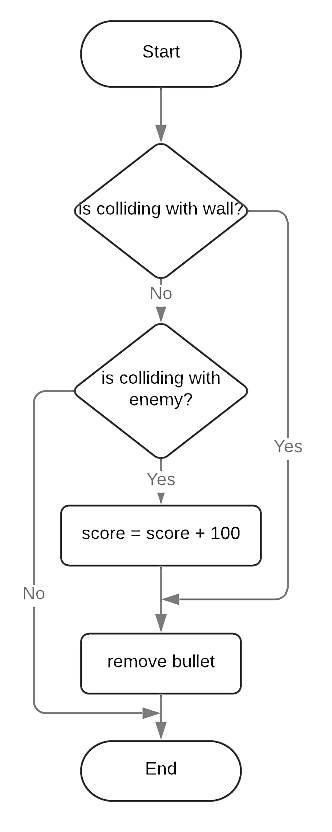
procedure Move(speedx, speedy):

xposition = xposition + speedx

yposition = yposition + speedy

end procedure

**Collide:**

This method checks for collisions for the bullet object in order to check when the bullet hits an enemy or a wall and deletes the bullet object and adding points to the total score. This fulfils **success criteria 3 and 12**. This is shown in the following flowchart:

### Algorithms: Enemy Bullet Class:

#### Move:

This method is used to move the enemy bullet object in the specified direction passed as parameters by the bow enemy. This accomplishes **success criteria number 21** and is shown in the following pseudocode:

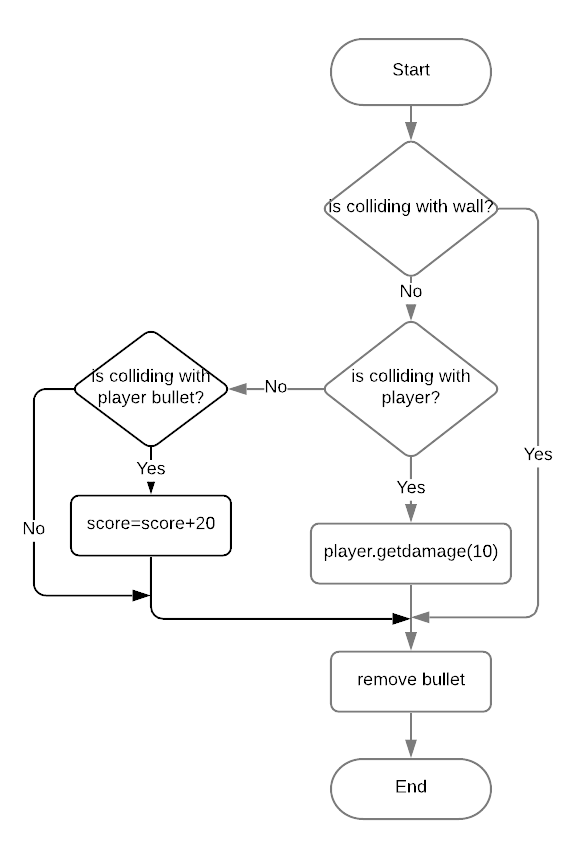
procedure Move(speedx, speedy):

xposition = xposition + speedx

yposition = yposition + speedy

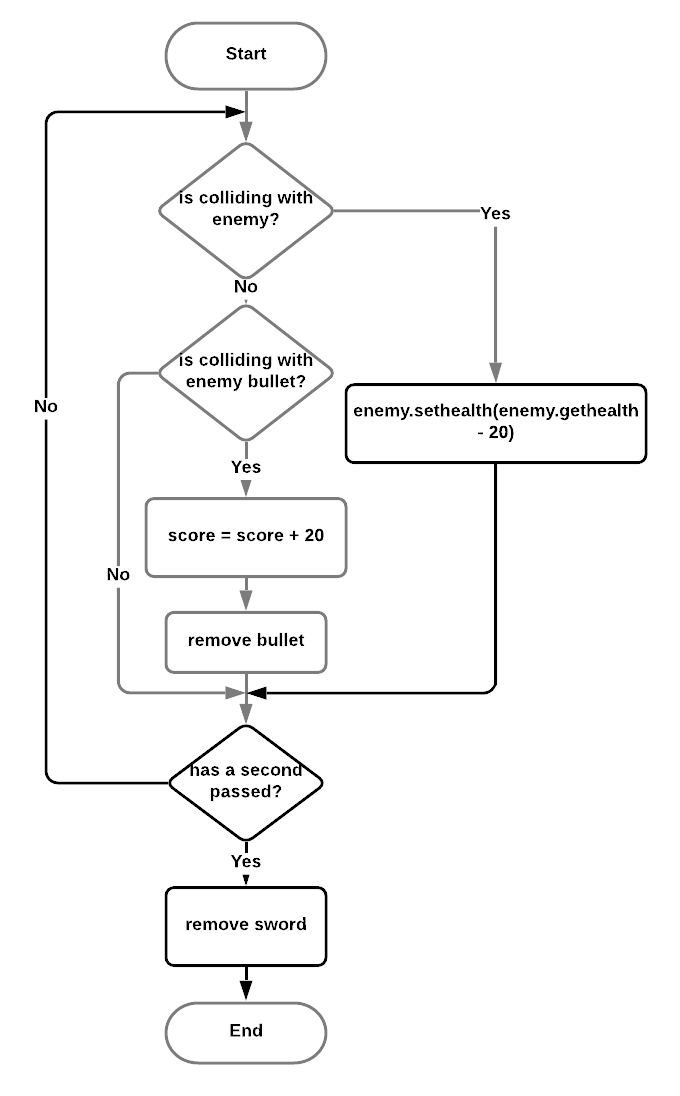
end procedure

#### Collide:

This method checks for collisions for the bullet object in order to check when the bullet hits the player or a wall or a player’s bullet and deletes the bullet object and adding points to the total score if two bullets collide. This fulfils **success criteria 3, 16 and 18**. This is shown in the following flowchart:

### Algorithms: SwordClass:

#### collision:

This method is used to check whether the sword collides with an enemy or an enemy bullet and inflicts damage and adds points once the enemy is killed or the bullet is blocked. Thus, it fulfils **success criterias 12, 18, 19 and 26**. This is shown in the following flowchart:

### Algorithms: Key Class:

#### collision:

This method is used to check if the player collides with a key and increases the score if colliding and increases the number of keys collected by one. This accomplishes **success criteria 13** and is shown in the following flowchart:

### Algorithms: Outside Wall Class:

#### Getpos:

This method is used to find the position of the walls which can then be used in pathfinding for the enemies to check whether they are within range of the player. Thus, this method helps with accomplishing **success criteria 16 and 17**. This is shown in the following pseudocode:

function Getpos():

return (xposition, yposition)

end function

### Algorithms: Inner Wall Class:

#### Getpos:

This method is used to find the position of the walls which can then be used in pathfinding for the enemies to check whether they are within range of the player. Thus, this method helps with accomplishing **success criteria 16 and 17**. This is shown in the following pseudocode:

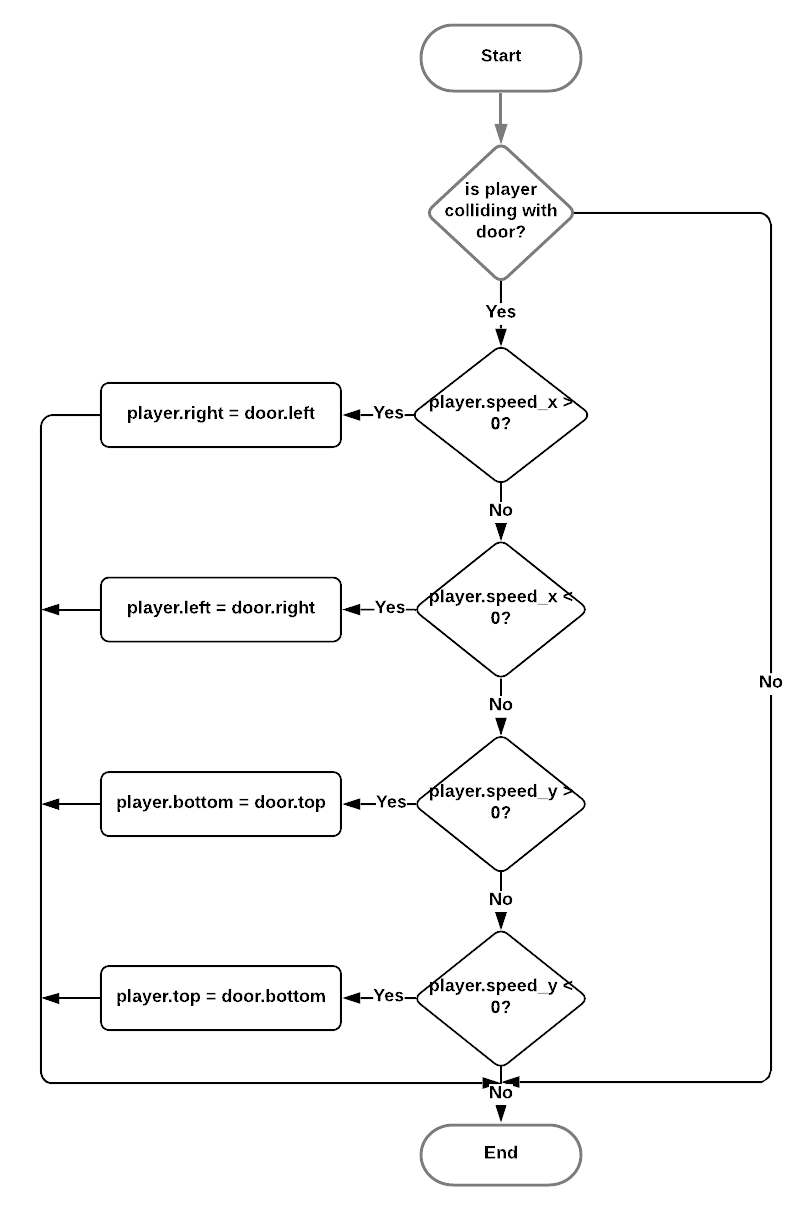
function Getpos():

return (xposition, yposition)

end function

### Algorithms: Door Class:

#### collision:

This method is used to check for a collision with the player and prevent the player from travelling through the door until all the keys are recovered by the player. This fulfils **success criteria number 13 and 14** and is shown in the flowchart below:

### Algorithms: Spikes Class:

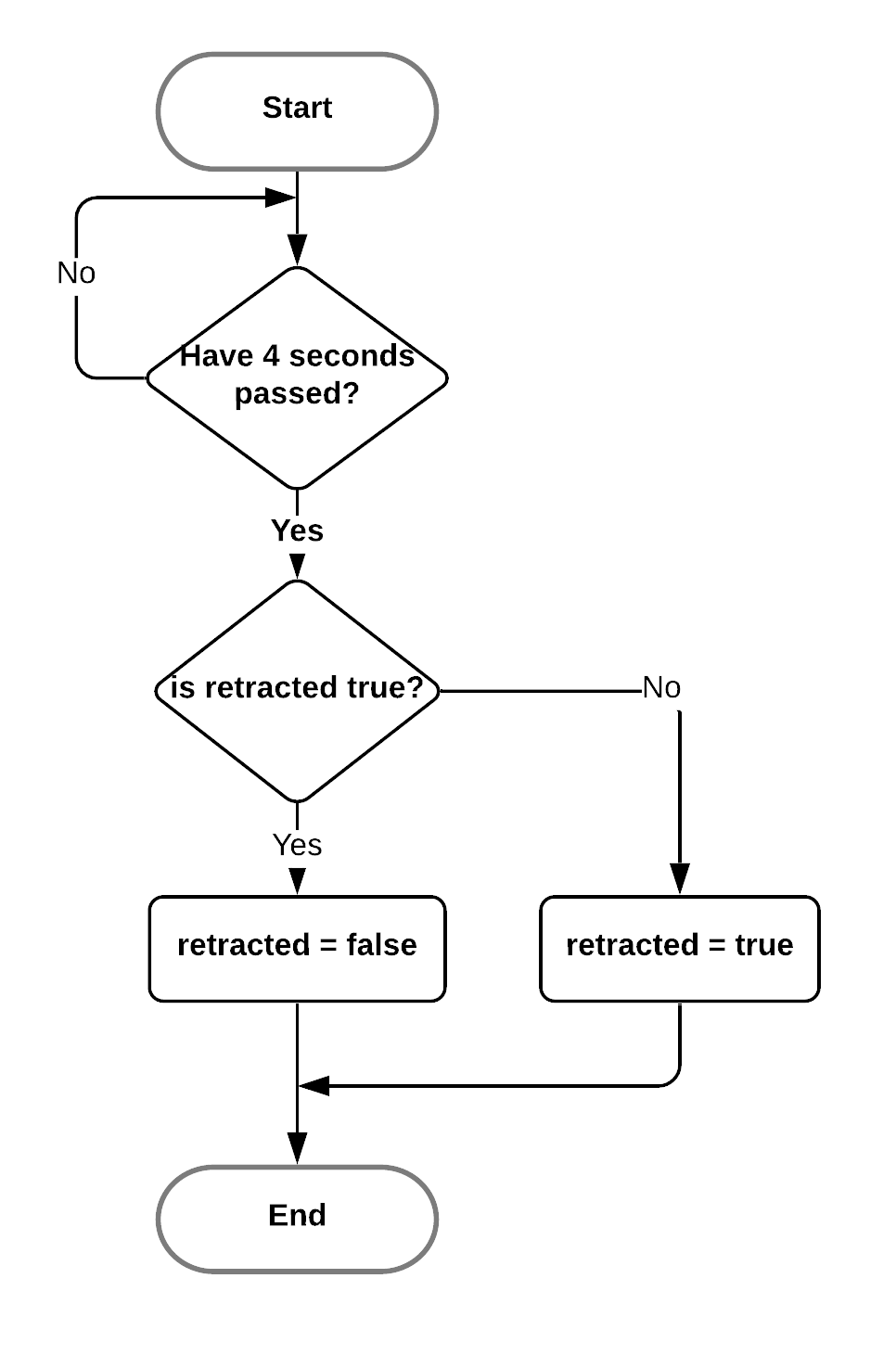
#### collision:

This method is used to check when the player is walking over the spikes and if the spikes are retracted or not. If the spikes are active, then damage is dealt to the player. So, this accomplishes **success criteria 28** and is shown in the following flowchart:

Diagram

Description automatically generated

#### retracting:

This method is used to change the spikes from active to inactive every 4 seconds so that when inactive the player is not damaged. This helps accomplish **success criteria 28** and is shown in the following flowchart:

#### get\_damage:

This method is used to damage the player when colliding with the spikes. So, it finishes **success criteria 28** and is shown in the pseudocode below:

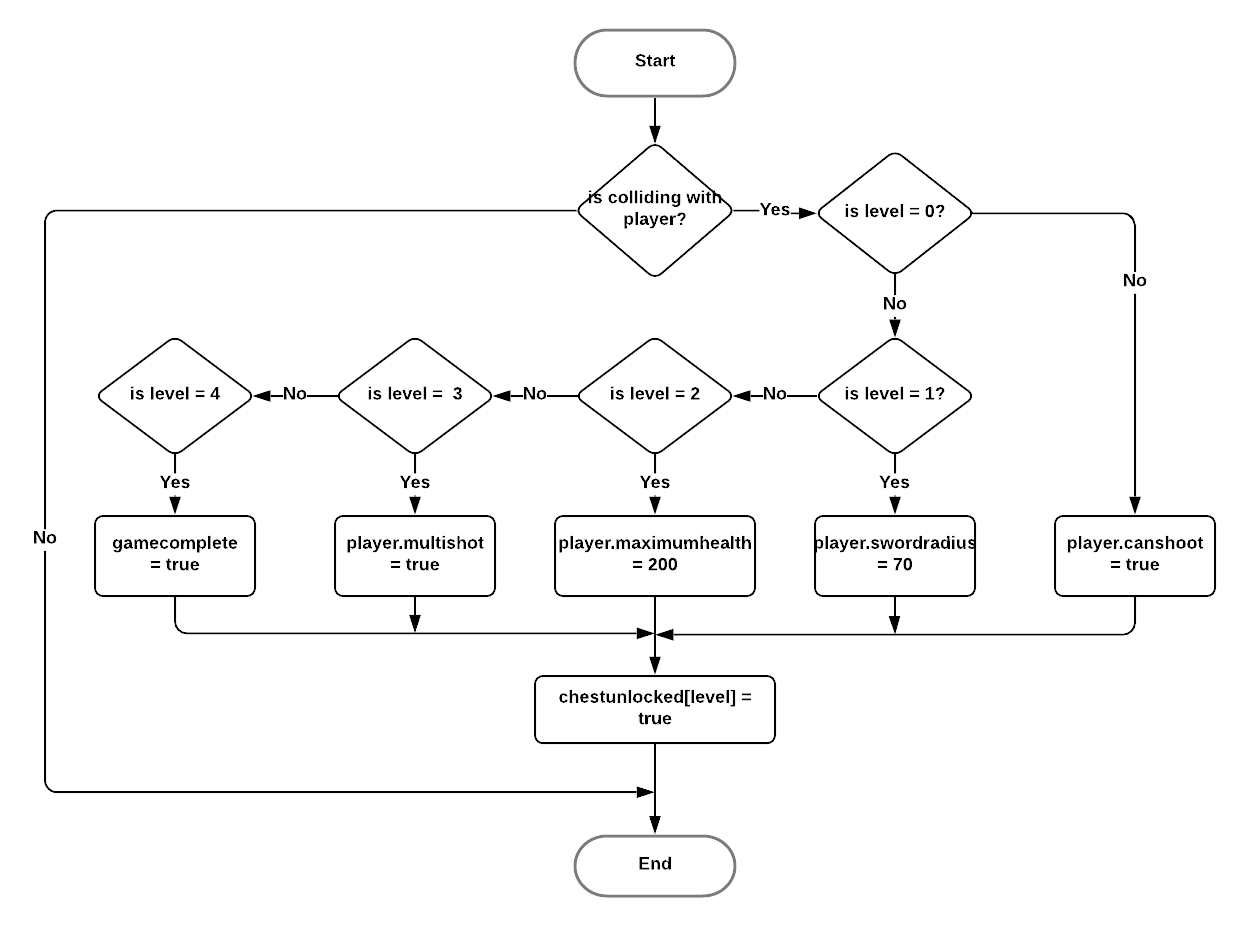
procedure get\_damage(damage):

player.getdamage(damage)

end procedure

### Algorithms: Chest Class:

#### collision:

This method is used to check whether the player collides with the chest in order to unlock a new ability depending on the level. This fulfils **success criteria 6, 7, 8, 9 and 10** and is shown in the following flowchart:

### Algorithms: Sword Enemy Class:

#### changespeed:

This basic method is key to how the enemy can move. It simply takes the inputs for x and y and adds them to the enemy’s speed in either direction. This helps accomplish **success criteria 17** and is shown in the following pseudocode:

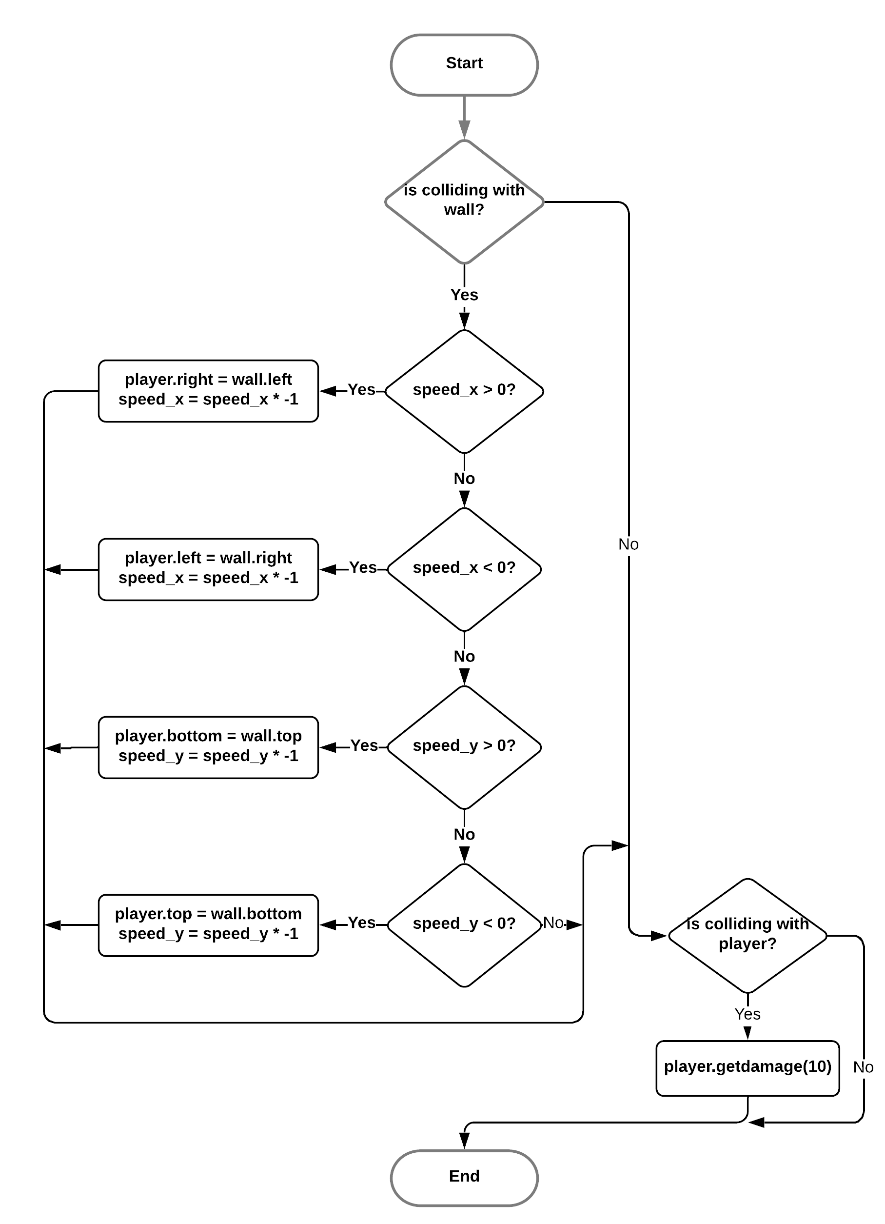
procedure changespeed(x, y):

speed\_x = speed\_x + x

speed\_y = speed\_y + y

end procedure

#### collision:

This method is used to check when the enemy collides with the player and when they're colliding remove health from the player and is also used to bounce off walls. This accomplishes **success criteria number 9** and is shown in the flowchart below:

#### is\_close:

This basic method is used to find the distance between the player and the enemy if below a certain range then a value is returned as true else it is returned as false. This is used to determine when the enemy will chase the player and whether the player is in range. Thus, this fulfils **success criteria number 17**. This is shown by the pseudocode below:

function is\_close():

xpos = player.xposition

ypos = player.yposition

distance = sqrt((xpos-xposition)\*\*2 + (ypos-yposition)\*\*2)

if distance < 300 then

return true

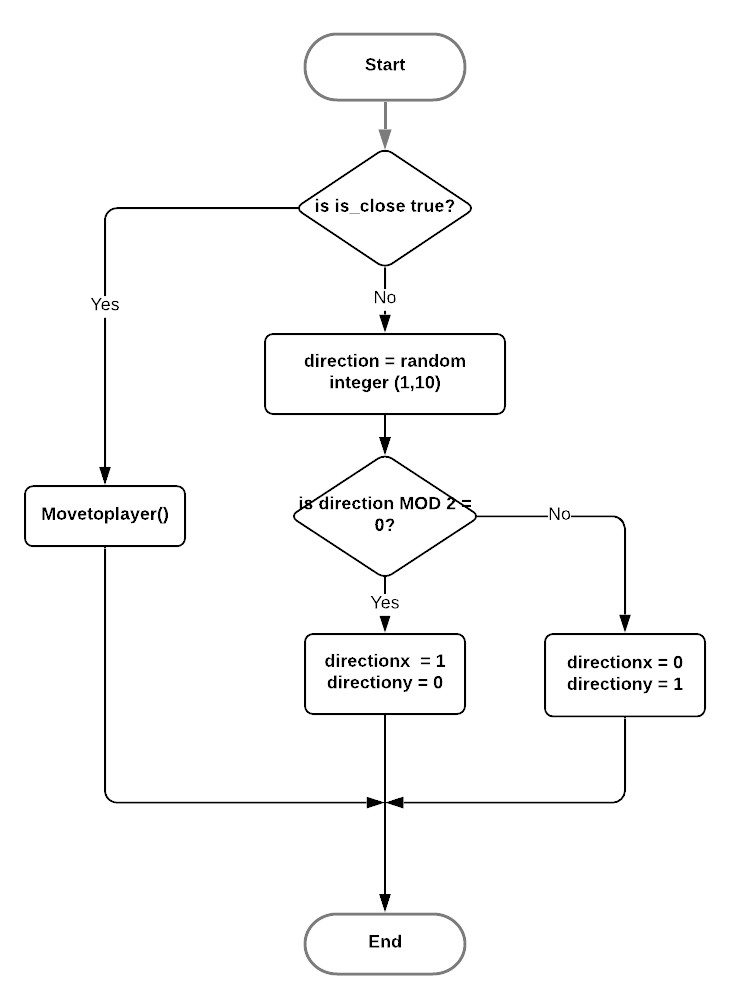
else

return false

endif

end function

#### move:

This method is fundamental to how the enemy will move when the player is not nearby. First, it checks whether the player is in range, then well either execute the procedure to move towards the payer or will bounce back and forth in a random direction. This provides more movement for the enemy when the player is not nearby, and such helps accomplish **success criteria 17**. This is shown by the following flowchart:

#### Movetoplayer:

This function compares the position of the player and the enemy and moves the enemy accordingly until the enemy has the same coordinates as the player. This accomplishes **success criteria 17** as this is what chases the player. This is shown in the following flowchart:

#### Gethealth:

This method is used to find the health of the enemy so that it can be used to set a new health for the enemy when colliding with a bullet or sword. This helps accomplish **success criteria number 19** and is shown in the pseudocode below:

function Gethealth():

return health

end function

#### Sethealth:

This method is used to set the health of the enemy when the enemy collides with a sword or bullet. This helps accomplish **success criteria number 12 and 19**. It is shown in the following pseudocode:

procedure Sethealth(newhealth):

health = newhealth

end procedure

### Algorithms: Bow Enemy Class:

#### is\_close:

This basic method is used to find the distance between the player and the enemy if below a certain range then the values returned as true else it is retained as false. This is used to determine when the enemy will shoot at the player or not. This accomplishes **success criteria 16** and is shown by the pseudo code below:

function is\_close():

xpos = player.xposition

ypos = player.yposition

distance = sqrt((xpos-xposition)\*\*2 + (ypos-yposition)\*\*2)

if distance < 300 then

return true

else

return false

endif

end function

#### Shoot:

This method is used to instantiate an enemy bullet and send it in the direction of the player. It finds the angle between the enemy and the player and sets the speed in the right direction. This accomplishes **success criteria 21** and is shown in the following pseudocode:

procedure Shoot():

xdist = player.xposition - xposition

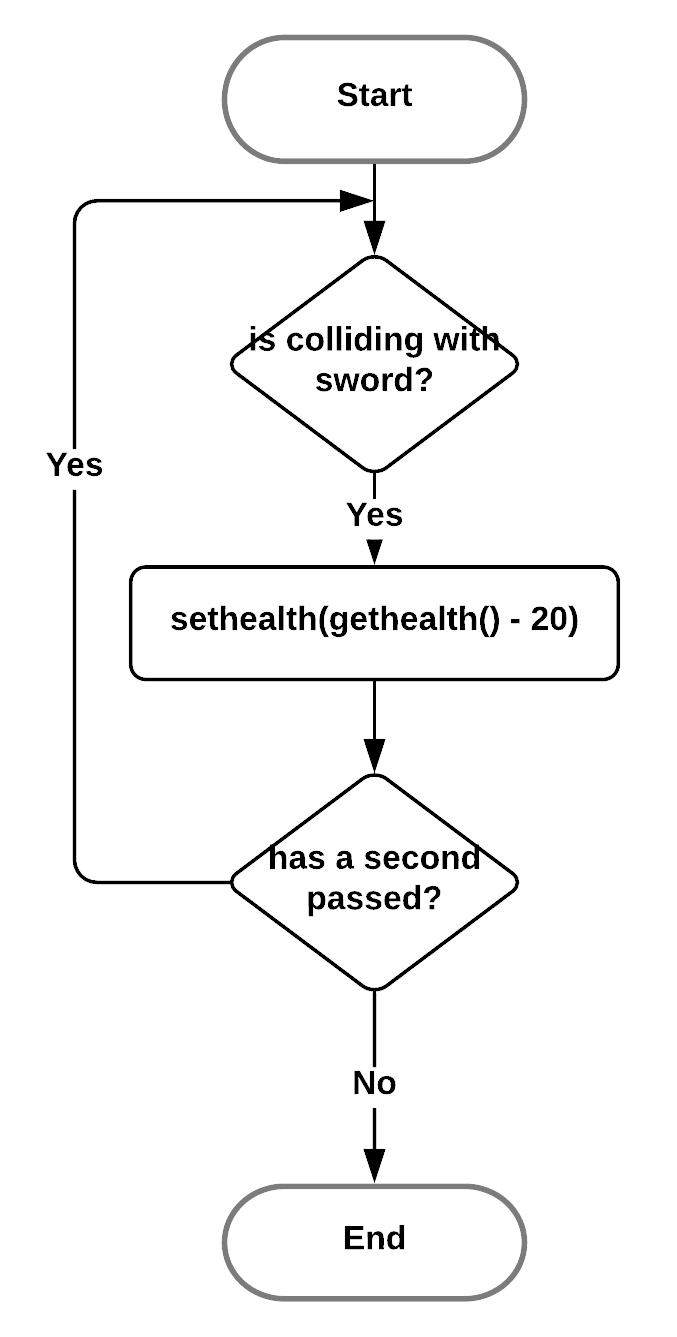
ydist = player.yposition – yposition

angle = tan-1(ydist/xdist)

bullet = new EnemyBullet(cos(angle), sin(angle))

end procedure

#### collision:

This method is used to detect if the enemy is colliding with a sword object. If it is then it will take 20 damage each second. This helps fulfil **success criteria number 19** and is shown by the flowchart below:

#### Gethealth:

This method is used to find the health of the enemy so that it can be used to set a new health for the enemy when colliding with a bullet or sword. This helps accomplish **success criteria number 19** and is shown in the pseudocode below:

function Gethealth():

return health

end function

#### Sethealth:

This method is used to set the health of the enemy when the enemy collides with a sword or bullet. This helps accomplish **success criteria number 12 and 19**. It is shown in the following pseudocode:

procedure Sethealth(newhealth):

health = newhealth

end procedure

### Algorithms: Boss Enemy Class:

#### is\_close:

This basic method is used to find the distance between the player and the enemy if below a certain range then a value is returned as true else it is returned as false. This is used to determine when the enemy will chase the player and whether the player is in range. Thus, this fulfils **success criteria number 17**. This is shown by the pseudocode below:

function is\_close():

xpos = player.xposition

ypos = player.yposition

distance = sqrt((xpos-xposition)\*\*2 + (ypos-yposition)\*\*2)

if distance < 300 then

return true

else

return false

endif

end function

#### collision:

This method is used to check when the enemy collides with the player and when they are colliding remove health from the player and is also used to not pass through the walls. This accomplishes **success criteria number 9 and 23** and is shown in the flowchart below:

#### Shoot:

This method is used to instantiate an enemy bullet and send it in the direction of the player. It finds the angle between the enemy and the player and sets the speed in the right direction. This accomplishes **success criteria 21 and 24**, and is shown in the following pseudocode:

procedure Shoot():

xdist = player.xposition - xposition

ydist = player.yposition – yposition

angle = tan-1(ydist/xdist)

bullet = new EnemyBullet(cos(angle), sin(angle))

end procedure

#### Createenemy:

This method is used to instantiate many enemies around the boss enemy after the boss enemy’s health is below a certain value. This adds more stages to the boss and helps accomplish **success criteria 24 and 30**. It is shown in the flowchart below.

#### Gethealth:

This method is used to find the health of the enemy so that it can be used to set a new health for the enemy when colliding with a bullet or sword and to be displayed by the boss health bar. This helps accomplish **success criteria number 19 and 31** and is shown in the pseudocode below:

function Gethealth():

return health

end function

#### Sethealth:

This method is used to set the health of the enemy when the enemy collides with a sword or bullet and is used to change the size of the health bar. This helps accomplish **success criteria number 12 and 19 and 31**. It is shown in the following pseudocode:

procedure Sethealth(newhealth):

health = newhealth

end procedure

#### Healthbar:

This method is used to display the health of the boss enemy. It uses two rectangles one rectangle is the change in health and the 2nd and the first rectangle is the actual health. The sizes of the rectangles are determined by the ration between the maximum health and the maximum size of the rectangle. This accomplishes **success criteria 31** and is shown in the pseudocode below:

procedure health\_bar():

transition\_width = 0

transition\_color = RED

if current\_health < target\_health:

current\_health = current\_health + health\_change\_speed

transition\_width = target\_health-current\_health//health\_ratio

transition\_color = GREEN

elseif current\_health > target\_health:

current\_health = current\_health – health\_change\_speed

transition\_width = targethealth-current\_health//health\_ratio

transition\_color = YELLOW

endif

if current\_health >= 700:

health\_bar\_color = GREEN

if current\_health >= 400 AND < 700:

health\_bar\_color = ORANGE

if current\_health < 400:

health\_bar\_color = RED

endif

health\_bar\_width = current\_health//health\_ratio

health\_bar = rectangle(1005, 45, health\_bar\_width, 25)

transition\_bar = rectangle(health\_bar.right, 45, transition\_width, 25)

display(health\_bar\_color, health\_bar)

display(transition\_color, transition\_bar)

display(WHITE, (xposition + 10, yposition + 50, health\_bar\_length, 25), 4)

end procedure

#### Getdamage:

This method is used to inflict damage on the boss enemy by changing the health of the boss by an input amount and adjusting the size of the health bar rectangles. This helps fulfil **success criteria 12 and 31** and is shown in the pseudocode below:

procedure Getdamage(amount):

target\_health = current\_health - amount

if target\_health > 0:

current\_health = target\_health

elseif target\_health < 0:

current\_health = 0

endif

end procedure

## Test Plan

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test** | **Description** | **Input** | **Expected Outcome** | **Justification** |
| 1 | Screen appears and is 1200 by 1000 pixels | Run the code | Screen appears on the display | From my research a large window was needed to show the player’s health, score and extra counters as well as showing each level in detail. |
| 2 | Player moves right | Click D key | Player moves right 2 pixels | The game is designed such that a player can move around the screen, which is a requirement given by my stakeholders |
| 3 | Player moves left | Click A key | Player moves left 2 pixels | The game is designed such that a player can move around the screen, which is a requirement given by my stakeholders |
| 4 | Player moves forwards | Click W key | Player moves forward 2 pixels | The game is designed such that a player can move around the screen, which is a requirement given by my stakeholders |
| 5 | Player moves backwards | Click S key | Player moves back 2 pixels | The game is designed such that a player can move around the screen, which is a requirement given by my stakeholders |
| 6 | Player swings sword | Click SPACE key | Player executes swing animation and damages any enemies nearby. | This is required as a success criteria point. |
| 7 | Player shoots bullet | Click E key | Player shoots bullet in direction of last pressed key | This is required by a success criteria given by my stakeholders |
| 8 | Player cannot move through walls | Hold W, A, S or D while next to a wall | Player stops moving when touching wall | This is to ensure that the player does not leave the screen. This is a test that appears in similar games as it creates obstacles for the player. |
| 9 | Bullets cannot pass through walls. | Click E key | Bullet will continue moving and then disappear once it hits a wall | This is used to as in my research this feature prevented bullets from making the game too easy for the player |
| 10 | Enemies spawn in an empty space when a level is created. | Run the code | No walls have been replaced by an enemy. | This is to make sure that the level is created properly and there are no objects missing |
| 11 | The correct number of enemies are created per level | Run the code | There should be 2 times add 1 of the level number to create how many enemies there are per level. | This is to increase the difficulty of each level so that the player has to deal with increasingly more enemies. |
| 12 | Enemies colliding with the wall and outside the range should reflect off the wall | Run the code | As an enemy object collides while travelling vertically or horizontally, the velocity of the enemy is multiplied by -1 causing the enemy to travel in the opposite direction. | This is tested as it confirms whether enemies can move properly. |
| 13 | Enemies that are within range of the player and colliding with a wall cannot move through the wall but around it. | Run the code | Enemies should stop once colliding with a wall in that direction but still retain their perpendicular velocity to where they collided with the wall. | This is used to guarantee that the walls work to stop the player and the enemies. |
| 14 | Health bar should change size when player takes damage | Run the code and move player into enemy object | Bar decreases in size and changes to yellow when below 70 and then change to red when below 30. | This is tested as it is a common test that appears in the other games I researched. |
| 15 | Sword Enemies move towards the player when in range | Run the code and move the player towards an enemy | Sword enemy x and y coordinates should keep changing until they are the same as the players x and y coordinates. | This is tested as it is one of the requirements points to have the enemies only attack the player if they are close enough. |
| 16 | The player can only heal to maximum health | Run the code and take damage | The attribute health of the player does not go past 100 when health is gained without increased maximum health | This guarantees that the player cannot gain more health than originally intended otherwise the player would easily beat the game as their health continues to increase. |
| 17 | The enemies can only attack once per second | Run the code and take damage from a Sword Enemy | The player’s health can only decrease by 10 every second for each enemy colliding with it. | This was included as it is designed to prevent the enemies from inflicting too much damage on the player each second and this test follows on from similar games that use the same limit on the enemies to make it easier to play the game. |
| 18 | When the player has zero health the game ends | Run the code and take damage until the player reaches 0 health. | A screen saying Game Over should appear and all the objects are removed from the screen | This test is used to make sure that the game over screen that was requested by the stakeholders appears. |
| 19 | When an enemy is killed it should increase the score | Run the code and press space until an enemy is killed | The score attribute in the game class increments by 100 when each enemy reaches zero health. | This was introduced to make sure that the player can gain points as this is one of the objectives of the game, which is to have the highest score. |
| 20 | When an enemy is killed it should be removed from the screen and should create a key object | Run the code and kill an enemy | After the enemy object is removed from the all\_sprites group it should no longer be displayed and an instance of the key class is created and in the location of the enemy before it is removed. | This is necessary to the game as without a key object being produced then the player cannot advance to the next level. |
| 21 | Keys collected should increase the key count | Collect a key | Key count on screen should increment by 1 | This test is included to help the player find out how many keys he has in order to complete each level. |
| 22 | Players cannot pass through doors until all keys are collected and chest opened | Run code and collect all keys and collide with chest | Door should open/disappear once all keys for level obtained and chest opened. | This is implemented to prevent the player from trying to skip levels, which increases the difficulty of the game. |
| 23 | When entering a door, the position of the player is changed. | Run the code and walk into open door | the player should be redrawn with the player moving position from xposition = 1000 to xposition = 50 | This is necessary to simulate the player moving from one level to another |
| 24 | When the level changes the levelsetup procedure removes all the objects from the previous level and replaces them with the new objects for the next level. | Run the code and walk into an open door | All objects are the screen are removed and replaced with new objects for the next level, with the doors closed. | This test is used to make sure that each level generates correctly with all the necessary objects |
| 25 | Player bullets should block enemy bullets | Run the code and click E key | Both player and enemy bullets should be removed from the screen when they collide, and the score should increment by 20 | This test is used as it has been requested by the stakeholders and was a feature used in other games. |
| 26 | Player cannot take damage when spikes are retracted | Run the code and walk over spike | Damage will not be taken until the spike changes. Once the spike changes then 5 damage will be inflicted on the player’s health for every spike colliding with the player every second | This test is used to make sure that the player does not take damage when not intended and this feature has been chosen by a stakeholder. |
| 27 | Player should unlock the ability to shoot after completing the first level and colliding with the chest | Run the code and press SPACE then walk over the chest. | Centred text appearing on the screen and are removed from the screen when entering a new level. | This test was included as a suggestion given by a stakeholder which increases the difficulty of the game. |
| 28 | Player can go back a level by going through a door | Run the code and walk through a door they have been through | All objects in the level should be removed and the previous level appears but no enemies should be created, and the doors should remain open. | This was suggested by a stakeholder and could prove to be useful in a future development of the game which includes gathering abilities. |
| 29 | Swords can block bullets | Hold SPACE key | Score should increment by 20 when an enemy bullet collides with a sword. | This was a feature that was recommended from researching other games and improved the game play of the program. |
| 30 | Player Health bar should be on the right-hand side of the screen | Run the code | A small white outlined rectangle with a green rectangle and number inside of it should appear over the rest of the objects. | This is to allow the player to have complete visibility of the level so that the health bar does not obstruct visibility. |
| 31 | The boss health bar follows the boss around | Run the code | The health bar should be centred above the boss object and be overlayed over all other objects. | This provides the player with information about the boss which can be used to deduce how the player can defeat the boss and win the game. This was included as many researched games also included this feature. |
| 32 | The boss health bar should change colour from green to yellow. | Run the code | The health bar should change from green to yellow drops below 70% | This provides the user with a visual indication of the boss health when it has below 70% |
| 33 | The boss health bar should change from yellow to orange | Run the code | The health bar should change from yellow to orange when the boss health is between 70% and 40% | This provide the user with a visual indicator for the percentage of the boss’s health which was recommended by a stakeholder. |
| 34 | The boss health bar should change from orange to red | Run the code | The health bar should change from orange to red when the boss health is below 40% | This provide the user with a visual indicator for the percentage of the boss’s health which was recommended by a stakeholder. |
| 35 | The health bar should be a set size of 100 pixels by 45 pixels. | Run the code | The health bar should appear as 100 pixels across and 45 pixels wide. | This set size is to make the health visible to the user without being too large on the screen. |
| 36 | When the boss is killed, it should say game complete | Run the code and kill the boss. | Text centred in the screen reading “Game Complete” should appear on the screen. | This is used as it was recommended by a stakeholder as it provides a usability for the user to understand what is going on in the game. |
| 37 | Score should increase after completing a level | Run the code | Score should increment by 500 points when a new level is generated. | This increases the score of the player as this is one of the main foci of the game, which is to have the highest score. |
| 38 | When the Quit Button is pressed the window should close | Run the code and press the quit button on the game over screen | Quit button is pressed, the window that the game is running will exit. | A quit button is useful usability feature that is easy to use and provides an easy way to exit the game without having to forcibly close the window. |
| 39 | When the boss health is below 700 then the boss shoots bullets towards the player | Run the code and get the boss to below 700 health | Boss object should follow the player while shooting a bullet towards the player every 1.5 seconds | This test was implemented to ensure the recommendation of having multiple stages for the boss enemy |
| 40 | When the boss health is below 400 the boss shoots 8 bullets in all directions from the boss | Run the code and get the boss below 400 health | Boss object creates 8 instances of bullets each with a velocity that causes them to travel vertically, horizontally, or diagonally. Bullets are instantiated every 3.5 seconds. | This test was implemented to ensure the recommendation of having multiple stages for the boss enemy |
| 41 | When the boss health is below 400 the boss creates 10 instances of the sword enemy class | Run the code and get the boss health below 400 health | 10 sword enemies are created and appear on the screen in random positions on the screen, without removing any walls or spike objects. | This test was implemented to ensure the recommendation of having multiple stages for the boss enemy |
| 42 | Main Menu should include text on the side about the controls | Run the code | The displayed main menu should include information on how to move, shoot and attack with the sword to help the user learn the controls of the game. | This test was introduced to guarantee the inclusion of usability feature in the form of instructions to the controls of the game. |
| 43 | Game over screen displays the score of the current game and the highest score achieved by anyone playing the game | Run the code | The game over menu has text saying game over, the score and the high score. It should also include a quit button to exit the game. | This test was included as a recommendation from a stakeholder to include a high score screen and easily exit the game once the player has finished. |

# Implementation

## Iteration 1 – Displaying All Classes:

Iteration 1 has the main focus of implementing every class such that they are displayed on the screen. To do this, the Game class and game loop need to be completed first so that the screen can update and display each object. **The requirements to be completed in this iteration are 2, 3, 4, 6, 7, 8, 11, 12, 13, 14, 16, 17, 20, 21, 27, 28, 32, 40.**

### Game Loop:

The game loop is used to process and iterate all actions and processes within the game. This is used to glue every procedure together and thus is required for the game to run. Below, the size of the screen is set by the variable size and the variable done is set to false before creating an instance of the game class and iterating through the methods in the game object 60 times per second. When done = True then the pygame window is closed. This is shown in the following code:

1. **import** pygame
3. size = (1200, 1000)
4. screen = pygame.display.set\_mode(size)
5. **pygame.display.set\_caption("Dungeon Escape")**
7. *#Loop until the user clicks the close button.*
8. done = False
10. ***# Used to manage how fast the screen updates***
11. clock = pygame.time.Clock()
12. *# Create instance of Game Class*
13. game = Game()
14. *# -------- Main Program Loop -----------*
15. **while not done:**
16. *# --- Main event loop*
17. done = game.eventprocess()
19. *# --- Game logic should go here*
20. **game.runlogic()**
22. *#draw the screen*
23. game.display(screen)
25. ***# --- Limit to 60 frames per second***
26. clock.tick(60)
28. *# Close the window and quit.*
29. pygame.quit()

Line 17 uses the function eventprocess within the game class to check whether the game is finished or not. In Line 20, all updates, collisions, and procedures performed by other classes are handled within this function. When the eventprocess function returns true, then the variable done causes the while loop to stop and run Line 29 which stops ends the program.

### Game Class:

#### Constructor Function

The constructor function for the Game class is used to initiate every level, sprite group and reset the score and level. The Game class uses the pygame library for python in order to create sprite groups.

1. **def** \_\_init\_\_(self):
2. *#Define all the atributes of the Game class*
3. *#Set the Score of the game to 0*
4. self.score = 0
5. ***#Game is not over so should run***
6. self.game\_over = False
7. *#Set the game level to 0 to start at the first level*
8. self.level = 0
9. *# Create different sprite groups to be used for collisions and updates*
10. **self.all\_sprites\_group = pygame.sprite.Group()**
11. self.outsidewall\_group = pygame.sprite.Group()
12. self.wall\_group = pygame.sprite.Group()
13. self.innerwall\_group = pygame.sprite.Group()
14. self.player\_group = pygame.sprite.Group()
15. **self.bullet\_group = pygame.sprite.Group()**
16. self.enemy\_group = pygame.sprite.Group()
17. self.menemy\_group = pygame.sprite.Group()
18. self.key\_group = pygame.sprite.Group()
19. self.door\_group = pygame.sprite.Group()
20. **self.sword\_group = pygame.sprite.Group()**
21. self.spike\_group = pygame.sprite.Group()
22. self.chest\_group = pygame.sprite.Group()
23. self.boss\_group = pygame.sprite.Group()
24. self.enemybullet\_group = pygame.sprite.Group()

In line 4, the score is set to zero in order to reset any previous score and in line 6 the attribute game\_over is set to false as when set to true then the game loop ends. Lines 10 to 24 create sprite groups which are used to categorise sprites to determine how they collide with other sprites.

The following code is a 2D array used to create the level:

1. *#Create a 2D array of each level with each digit representing a tile of 40 by 40 pixels*
2. self.level1 = [
3. [1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1],
4. [1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1],
5. **[1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1],**
6. [1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1],
7. [1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1],
8. [1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1],
9. [1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1],
10. **[1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1],**
11. [1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1],
12. [1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1],
13. [1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1],
14. [1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1],
15. **[1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1],**
16. [1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1],
17. [1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1],
18. [1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1],
19. [1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1],
20. **[1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1],**
21. [1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1],
22. [1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1],
23. [1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1],
24. [1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1],
25. **[1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1],**
26. [1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1],
27. [1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1]
28. ]

This 2D array directly correlates each element to a 40 by 40 square in the window. Each number represents a different class. With a 1 representing an outside wall and a 0 representing no class at all. This is an easier method to instantiating all the objects at the beginning of the level than a 1D array as it is visually clear where objects will go and will make it easier to make changes to how each level is designed.

As the game class is instantiated, the first level should be created thus the final code of the constructor function is below:

1. *#Create an array of every level*
2. self.levels = [self.level1]
3. *#Run the following function to instantiate all objects needed for the level*
4. self.levelsetup()

line 2 is an array that stores every level within the game. Currently only one level has been made as it will be used to check if all the sprites appear. Line 4 runs the levelsetup procedure which checks the current level and then generates it.

#### Level Setup Procedure

The level setup procedure is used to transform the 2D array for each level into the actual level. Thus, it must read through each item in the array. First a set number of enemies must be created for each level. Line 12 and 13 set the number of enemies created to 0 then will loop through lines 14 to 18 until all the enemies for that level are created. In line 15 and 16 random x and y coordinates are chosen for the enemies to instantiate in and thus replace those elements in the 2D array in line 17 with either a number 3 or 4 to create a random number of Bow and Sword enemies.

1. *#Create a method that creates all the objects according to the 2D array in the game class.*
2. **def** levelsetup(self):
3. *#Set the count to 0*
4. self.count = 0
5. **self.bosscount = 0**
6. *#Spawn the boss enemy when the level is 5*
7. **if** (self.level + 1) % 5 == 0:
8. self.levels[self.level][10][10] = 5
10. **else:**
11. *#When the level is not level 5 then create enemies according to what level the player is on.*
12. enemies = 0
13. **while** enemies != ((2\*(self.level+1)) + 1):
14. *#Find two random numbers between 1 and 23*
15. **xpos = random.randint(1,23)**
16. ypos = random.randint(1,23)
17. self.levels[self.level][xpos][ypos] = random.randint(3,4)
18. enemies = enemies +1

Continuing the Level setup procedure Lines 2 and 3 are used to iterate through each element in the array and create a new object depending on the character stored at that location. Each character is checked using if statements and if, for example, the character is 1 then it will create an instance of a wall and add it to all sprites group and the wall groups. These groups are used for updating all the sprites and for handling collisions. This then checks for all numbers up to 7 and if it is a 0 or another number then no new objects are created.

1. *#Search through the 2D array and create objects that corespond to each coordinate that forms the grid.*
2. **for** j **in** range(len(self.levels[self.level])):
3. **for** i **in** range(len(self.levels[self.level][j])):
4. *#Check what each character is in every index then create objects according to each character*
5. **char = self.levels[self.level][j][i]**
6. **if** char == 1:
7. *#Create outside wall object and add it to corresponding Sprite groups and add the coordinates to the wall list*
8. self.outsidewall = Wall(RED,40,40,i\*40, j\*40, i, j)
9. self.all\_sprites\_group.add(self.outsidewall)
10. **self.wall\_group.add(self.outsidewall)**
11. self.outsidewall\_group.add(self.outsidewall)
12. **if** char == 2:
13. *#Create inner object and add it to corresponding Sprite groups and add the coordinates to the wall list*
14. self.innerwall = InnerWall(RED,40,40,i\*40, j\*40, i, j)
15. **self.all\_sprites\_group.add(self.innerwall)**
16. self.wall\_group.add(self.innerwall)
17. self.innerwall\_group.add(self.innerwall)
18. **if** char == 3:
19. *#Create Bow enemy object and add it to corresponding Sprite groups*
20. **if self.levelcomplete[self.level] == False:**
21. self.benemy = BowEnemy(random.randint(0,10),40,40, i\*40, j\*40, 40)
22. self.all\_sprites\_group.add(self.benemy)
23. self.enemy\_group.add(self.benemy)
24. **if** char == 4:
25. ***#Create Melee enemy object and add it to corresponding Sprite groups***
26. **if** self.levelcomplete[self.level] == False:
27. self.menemy = MeleeEnemy(random.randint(0,10),40,40, i\*40, j\*40, 40)
28. self.all\_sprites\_group.add(self.menemy)
29. self.enemy\_group.add(self.menemy)
30. **self.menemy\_group.add(self.menemy)**
31. **if** char == 5:
32. *#Create Boss object and add it to corresponding Sprite groups*
33. **if** self.levelcomplete[self.level] == False:
34. self.boss = BossEnemy(random.randint(0,10),160,160, i\*40, j\*40, 1000)
35. **self.all\_sprites\_group.add(self.boss)**
36. self.boss\_group.add(self.boss)
37. **if** char == 6:
38. *#Create Door object and add it to corresponding Sprite groups*
39. **if** self.levelcomplete[self.level] == False:
40. **self.door = Door(PURPLE,40,40,i\*40, j\*40, i, j)**
41. self.all\_sprites\_group.add(self.door)
42. self.wall\_group.add(self.door)
43. self.door\_group.add(self.door)
44. **if** char == 7:
45. ***#Create Spike object and add it to corresponding Sprite groups***
46. self.spike = Spikes(GREY, 40,40,i\*40, j\*40, i, j)
47. *#self.all\_sprites\_group.add(self.spike)*
48. self.spike\_group.add(self.spike)

#### Event Process Function

This method is very basic and is only utilised to check whether the user has closed the window or not. It will return True when the close button of the window is pressed to stop the program.

1. *#Method used to check for event if they quit*
2. **def** eventprocess(self):
3. **for** event **in** pygame.event.get():
4. **if** event.type == pygame.QUIT:
5. **return True**
6. **return** False

#### Run Logic Function

The run logic function is used to update all the sprites in the level from lines 5 to 7and also checks whether the player group is empty in lines 9 to 11. If the player group is empty then it will return that the game\_over variable is now true and thus ending the game.

1. *#Method that processes all interactions not within other classes*
2. **def** runlogic(self):
3. **if** **not** self.game\_over:
4. *# Move all the sprites*
5. **self.sword\_group.update()**
6. self.spike\_group.update()
7. self.all\_sprites\_group.update()
8. *#Check if the player has died*
9. **if** len(self.player\_group) == 0:
10. ***#If true, then end the game***
11. self.game\_over = True

#### Display Procedure

The Display procedure is used to display all the sprites and text on screen whilst the game is running and once the game is over. First it sets the background of the window to black in line 3 and if the game is over will then refill the screen black and then add text saying Game Over and the Score achieved in lines 4 to 11. If the game isn’t over then it will display the score and number of keys for each level obtained as well as the health in lines 14 to 21 on the right-hand side of the screen. Finally, lines 23 to 26 display all the sprite groups on the screen in the specified order with the spikes and sword at the bottom, then the display is updated in line 28.

1. **def** display(self, screen):
2. *# background image.*
3. screen.fill(BLACK)
4. **if** self.game\_over:
5. **screen.fill(BLACK)**
6. font1 = pygame.font.Font(None, 74)
7. font2 = pygame.font.Font(None, 48)
8. text = font1.render('GAME OVER', 1, WHITE)
9. score = font2.render('SCORE:'+str(self.getscore()), 1, WHITE)
10. **screen.blit(text, (440,300))**
11. screen.blit(score, (520,500))
12. **if** **not** self.game\_over:
13. *#When the game is running Display text on the right handside of the screen with the score and keys of the player*
14. font = pygame.font.Font(None, 24)
15. **score = font.render('SCORE:'+str(self.getscore()), 1, WHITE)**
16. keys = font.render('KEYS:'+str(self.player.getkeys()), 1, WHITE)
17. *#Draw the text of the player's health over the health bar*
18. health = font.render(str(self.player.current\_health), 1, WHITE)
19. screen.blit(score, (1050,500))
20. **screen.blit(keys, (1050,550))**
21. screen.blit(health, (1081, 51))
22. *# --- Drawing code for sprites*
23. self.sword\_group.draw(screen)
24. self.spike\_group.draw(screen)
25. **self.all\_sprites\_group.draw(screen)**
26. self.boss\_group.draw(screen)
27. *# --- Go ahead and update the screen with what we've drawn.*
28. pygame.display.flip()
29. ***#end procedure***

Text

Description automatically generated with medium confidenceBelow is an image of only the game class running. As only the game class object has been created the screen appears black:

### Player Class:

#### Constructor Function

The constructor function for the player class is used to define all its attributes to be used when creating an object from the class. In Line 2 I defined the parameters used for the player object, asking for its colour, width, height and initial x and y coordinates as well as some other parameters that may be implemented in another iteration. Line 4 we inherit the constructor for sprites and then we use the inherited attributes to define the size of the player, its colour and its coordinates. Lines 11 to 17 are used to define the initial conditions of the health bar for the player and lines 23 to 24 set the initial direction that a player bullet would travel as soon as the player object is created if no movement keys were pressed.

1. *#Method for when the player object is initially created.*
2. **def** \_\_init\_\_(self,color , width, height, x, y, health, score, money, gamekeys):
3. *#call sprite constructor*
4. super().\_\_init\_\_()
5. ***#create a sprite***
6. self.image = pygame.Surface([width,height])
7. self.image.fill(color)
8. *#set the position of the sprite*
9. self.rect = self.image.get\_rect()
10. ***#Set the current and maximum health of the player to be used for health bar***
11. self.current\_health = 50
12. self.maximum\_health = health
13. self.health\_bar\_length = 180
14. self.target\_health = 100
15. **self.health\_change\_speed = 2**
16. self.health\_bar\_color = GREEN
17. self.health\_ratio = self.maximum\_health/ self.health\_bar\_length
18. *#keys used to check to open doors to go to next level*
19. self.gamekeys = gamekeys
20. **self.rect.x = x**
21. self.rect.y = y
22. *#Set original direction for player*
23. self.directionx = 0
24. self.directiony = 5

Below is an image of the player object when it is instantiated:

Graphical user interface, application

Description automatically generated

#### Move Procedure

The move procedure is used to move the player across the screen by changing its x and y coordinates by its speeds. Line 4 adds the player’s horizontal speed to its x coordinate and in line 6 it adds the vertical speed to its y coordinate.

1. *#Method that takes input speeds and moves players position according to speed.*
2. **def** move(self, speedx, speedy):
3. *#move left or right*
4. self.rect.x += self.speed\_x
5. ***#move the player up and down the screen***
6. self.rect.y += self.speed\_y
7. *#end procedure*

Application

Description automatically generated with medium confidenceApplication

Description automatically generated with medium confidenceThe following screenshots below show how the player moves when a key is pressed:

#### Change Speed Procedure

This procedure takes in 2 parameters which are used to change the player’s x and y speeds. This is used alongside the move procedure to allow the player to move across the screen.

1. *#Change the player's speed by the values entered*
2. **def** changespeed(self, x, y):
3. self.speed\_x += x
4. self.speed\_y += y

#### Get Health Procedure

The Get health procedure is used to increase the player’s health when triggered by an event. Line 2 shows that the parameter amount is what is used to increase the player’s health by. As to stop the player’s health from exceeding its maximum, we use the if statement in line 5 to set the target health, the health which will be reached once adding the amount to it to equal the maximum health of the player if it is greater than it.

1. *#Method to increase health of player*
2. **def** gethealth(self, amount):
3. **if** self.target\_health < self.maximum\_health:
4. self.target\_health += amount
5. **if self.target\_health >= self.maximum\_health:**
6. self.target\_health = self.maximum\_health
7. *#endprocedure*

#### Get Damage Procedure

The get damage procedure is used to remove health from the player. Similar to the get health procedure it checks what the target health is before setting the new health of the player so that if the target health of the player is below zero then its will be set to equal zero. This is used to stop the player from having a negative health that could stop the game from ending once the player’s health reaches zero.

1. *#Method to remove health of palyer*
2. **def** getdamage(self,amount):
3. **if** self.target\_health > 0:
4. self.target\_health -= amount
5. **if self.target\_health <=0:**
6. self.target\_health = 0
7. *#end method*

A picture containing application

Description automatically generatedA picture containing graphical user interface

Description automatically generatedBelow are two images showing how inflicting damage changes the health displayed on the screen:

#### Health Bar Procedure

The health bar procedure is used to create and change the rectangles used that form the health bar. The health bar is formed of 3 rectangles, the main health bar, the transition bar and the outline bar. The outline bar outlines the other bars while the transition bar is used to display a change in the player’s health. Line 3 and 4 define the transition bar’s width as zero and its colour as red initially. Line 7 to 10 check whether the target health is larger than the initial health and then change the transition bar’s colour and width so that it matches the difference in size between the initial health and target health. This is then divided by the health ratio which is used to adjust the transition bars width to the correct number of pixels on the screen as the health bar is not the same number of pixels in width as the number of health points the player has. This process is repeated from lines 12 to 15 but in reverse with the transition bar now being yellow as health is deducted. Lines 18 to 23 have several if statements to change the colour of the main health bar depending on how much health the player has. Finally, lines 25 to 32 define the rectangles and draw them on the screen.

1. *#Method used to create health bar*
2. **def** advanced\_health(self):
3. transition\_width = 0
4. transition\_color = RED
6. *#When health is added set the transition bar to green and increase health bar at set speed*
7. **if** self.current\_health < self.target\_health:
8. self.current\_health += self.health\_change\_speed
9. transition\_width = int((self.target\_health - self.current\_health)/ self.health\_ratio)
10. **transition\_color = GREEN**
11. *#When health is removed set the transition bar to yellow and decrease the health bar at a set speed*
12. **if** self.current\_health > self.target\_health:
13. self.current\_health -= self.health\_change\_speed
14. transition\_width = int((self.target\_health - self.current\_health)/ self.health\_ratio)
15. **transition\_color = YELLOW**
17. *#Change the health bar colour depending on the amount of health*
18. **if** self.current\_health >= 70:
19. self.health\_bar\_color = GREEN
20. **if self.current\_health >= 50 and self.current\_health < 70:**
21. self.health\_bar\_color = ORANGE
22. **if** self.current\_health < 30 **and** self.current\_health >=0:
23. self.health\_bar\_color = RED
24. *#set the health bar size equal to the current health divided by max health*
25. **health\_bar\_width = int(self.current\_health/ self.health\_ratio)**
26. health\_bar = pygame.Rect(1005,45, health\_bar\_width, 25)
27. *#transition bar is set to the right of the health bar*
28. transition\_bar = pygame.Rect(health\_bar.right, 45, transition\_width, 25)
29. *#Draw all 3 bars over each other*
30. **pygame.draw.rect(screen, self.health\_bar\_color, health\_bar)**
31. pygame.draw.rect(screen,transition\_color, transition\_bar)
32. pygame.draw.rect(screen, WHITE, (1005, 45, self.health\_bar\_length, 25), 4)
33. *#end method*

A picture containing chart

Description automatically generatedA picture containing graphical user interface

Description automatically generatedBelow is the health bar changing from 100 to 20 health causing the bar to decrease in size and change colour:

#### Update Function

The update function is used to check for inputs from the user and moving the player. Line 4 first defines keys as a list of keys pressed. Thus, lines 5 to 35 use a many if statements to check for keys in the list that have been pressed and associate a direction in which the player moves or an action. This uses the change speed procedure and changes the x and y direction of the player once a movement key is pressed. Line 25 to 32 check from when the E key is pressed and sets a variable to the current time, followed by an if statement to check if 1 second has passed since the previous variable that had been set to the time before. If the statement is true then it will instantiate a player bullet object with the parameters of the direction that the player is facing. Line 33 to 42 check for when the Space key is pressed and then repeat the same timer for the sword once there are no sword sprites on the screen. The sword is then instantiated and then the timer is reset. Lines 44 to 46 use another timer that increases the player’s health by 10 every 10 seconds using the get health procedure and lines 48 to 50 check if the player’s health is zero, and if so, will remove the player sprite and increase the score, which will lead to the game ending. Finally, line 52 uses the move procedure to move the player object by its x and y speeds to move the player across the screen and 53 displays the health bar.

1. *#Update method for player to control movement and damage taken*
2. **def** update(self):
3. *#Check for what keys are pressed*
4. keys = pygame.key.get\_pressed()
5. **if keys[pygame.K\_a]:**
6. *#Move the player to the left and change the direction*
7. self.changespeed(-4,0)
8. self.directionx = -6
9. self.directiony = 0
10. **if keys[pygame.K\_d]:**
11. *#move the player to the right and change the direction*
12. self.changespeed(4,0)
13. self.directionx = 6
14. self.directiony = 0
15. **if keys[pygame.K\_w]:**
16. *#move the player up and change the direction*
17. self.changespeed(0,-4)
18. self.directionx = 0
19. self.directiony = -6
20. **if keys[pygame.K\_s]:**
21. *#move the player down and change the direction*
22. self.changespeed(0,4)
23. self.directionx = 0
24. self.directiony = 6
25. **if keys[pygame.K\_e]:**
26. *#Create a bullet once per second*
27. self.currentbullettime = pygame.time.get\_ticks()
28. **if** self.currentbullettime - self.previousbullettime > 1000:
29. bullet = Bullet(RED, self.directionx, self.directiony)
30. **game.bullet\_group.add(bullet)**
31. game.all\_sprites\_group.add(bullet)
32. self.previousbullettime = self.currentbullettime
33. **if** keys[pygame.K\_SPACE]:
34. *#Check if Space has been pressed and if the sword hasn't already been created*
35. **if len(game.sword\_group) == 0:**
36. *#create a sword with a set radius that lasts for 1 second and add the sword to its sprite groups*
37. self.currentattacktime = pygame.time.get\_ticks()
38. **if** self.currentattacktime - self.previousattacktime > 1000:
39. **sword = Sword(GREEN, self.swordradius)**
40. game.sword\_group.add(sword)
41. game.all\_sprites\_group.add(sword)
42. self.previousattacktime = self.currentattacktime
43. *#Check if 10 seconds have passed to add 10 health to the palyer*
44. **if self.currenthealthtime -self.previoushealthtime > 10000:**
45. self.gethealth(10)
46. self.previoushealthtime = self.currenthealthtime
47. *#Check if the player's health is 0 and kill the player to end the game.*
48. **if** self.current\_health == 0:
49. **game.score += 100**
50. self.kill()
51. *#use the move function by the speed specified according to the key press*
52. self.move(self.speed\_x,self.speed\_y)
53. self.advanced\_health()
54. *#end procedure*

### Player Bullet Class:

#### Constructor Function

The constructor function is used to set the initial conditions for the Player bullet. The parameters it takes are the colour, horizontal and vertical speeds. The bullet is fixed in size as it is 6 pixels across and 4 pixels high. Lines 11 and 12 instantiate the player bullet with initial coordinates in the centre of the player object. Lines 9 and 10 set the attributes to take on the values of their respective parameters.

1. *#Method to initiate bullet objects*
2. **def** \_\_init\_\_(self, color, speedx, speedy):
3. *#Call the sprite constructor*
4. super().\_\_init\_\_()
5. ***#Bullet created in the center of the player object***
6. self.image = pygame.Surface([6,4])
7. self.image.fill(color)
8. self.rect = self.image.get\_rect()
9. self.speedx = speedx
10. **self.speedy = speedy**
11. self.rect.y = game.player.rect.y + 10
12. self.rect.x = game.player.rect.x + 10

#### Update Function

The update function of the player bullet class is used to move the bullet according to its initial speed. Thus, its x and y coordinates are incremented by its set horizontal and vertical speeds. In lines 6 to 13, the update function is checking if the bullet has exceeded the coordinates displayed within the window and if true will then remove the sprite from all sprite groups.

1. *#Update method that moves the bullet by the speed which is defined by the direction that the player last moved in.*
2. **def** update(self):
3. self.rect.y += self.speedy
4. self.rect.x += self.speedx
5. ***#Check if it travel past the boundaries and kills the sprite***
6. **if** self.rect.x > 1000:
7. self.kill()
8. **if** self.rect.x < 0:
9. self.kill()
10. **if self.rect.y > 1000:**
11. self.kill()
12. **if** self.rect.y < 0:
13. self.kill()

A picture containing application

Description automatically generatedApplication

Description automatically generated with medium confidenceBelow shows how the bullet updates after each frame and travels in the direction of the last pressed key:

Moving Bullet

### Outer Wall Class:

#### Constructor Function

The constructor function of the outer wall class is used to determine the coordinates, size and colour of the wall. Line 7 and 8 define the size of the wall and its colour and lines 13 and 14 are parameters used to determine where in the array the wall is located.

1. *#Wall class to create boundaries*
2. **class** Wall(pygame.sprite.Sprite):
3. **def** \_\_init\_\_(self, color, width, height, x, y, posx, posy):
4. *#call sprite constructor*
5. **super().\_\_init\_\_()**
6. *#create a sprite*
7. self.image = pygame.Surface([width,height])
8. self.image.fill(color)
9. *#set the position of the sprite*
10. **self.rect = self.image.get\_rect()**
11. self.rect.x = x
12. self.rect.y = y
13. self.positionx = posx
14. self.positiony = posy
15. ***#end procedure***

#### Get Pos Function

The get pos function is used to find the location of the wall within the grid in a vector format. This can be used to create a path finding algorithm to help with the movement of enemies around each level.

1. *#Method to find the vector position of the wall in the grid*
2. **def** getpos(self):
3. **return** [self.positionx/40, self.positiony/40]
4. ***#end procedure***

Below is the first level displayed with the outer walls:

A picture containing shape

Description automatically generated

### Inner Wall Class:

#### Constructor Function

The inner wall class is currently completely identical to the outer wall class, as such it inherits everything from the Wall class, but can be subject to change in the future.

1. *#Innerwall Class inherits the wall class*
2. **class** InnerWall(Wall):
3. *#Inherits the init method of the wall class*
4. **pass**
5. ***#end class***

Below is an image with both the inner and outer walls:

A picture containing icon

Description automatically generated

### Sword Class:

#### Constructor Function

The constructor function of the sword does not require many parameters as the sword will always follow the player. As such, the radius parameter determines the size of the circle that represents the sword. As shown in lines 7, the sword object creates a circle. Line 10 defines a timer that will later be used in the update function.

1. **def** \_\_init\_\_(self, color, radius):
2. *#call sprite constructor*
3. super().\_\_init\_\_()
4. self.image = pygame.Surface((radius, radius))
5. ***#self.image.fill(BLACK)***
6. *#Create a circle to form the radius that the sword covers*
7. pygame.draw.circle(self.image, (color), (int(radius/2), int(radius/2)), int(radius/2))
8. self.rect = self.image.get\_rect()
9. self.rect.center = (0, 0)
10. **self.previousattacktime = pygame.time.get\_ticks()**
11. self.radius = radius

A picture containing icon

Description automatically generatedIcon

Description automatically generated with medium confidenceBelow shows an image of the sword when the space key is pressed:

#### Update Function

The update function is used to define the coordinates of the sword and how long the sword lasts before being removed. Lines 3 to 8 adjust where the centre of the circle is depending on the circle’s radius in order to make sure that it is always in the centre of the player. Lines 10 to 13 use a timer to check how long the sword has been created and after 0.75 seconds, will remove the sword object from all sprite groups.

1. *#Update function to set the sword to always be centered on the player depending on the size of the radius*
2. **def** update(self):
3. **if** self.radius == 50:
4. self.rect.y = game.player.rect.y -5
5. **self.rect.x = game.player.rect.x -5**
6. **else**:
7. self.rect.y = game.player.rect.y -15
8. self.rect.x = game.player.rect.x -15
9. *#Set the sword to remove itself after 750 milliseconds after Space is pressed*
10. **self.currentattacktime = pygame.time.get\_ticks()**
11. **if** self.currentattacktime - self.previousattacktime > 750:
12. self.kill()
13. self.previousattacktime = self.currentattacktime

### Door Class:

#### Constructor Function

The constructor function of the door class is identical to that of the wall class as it acts identical to a wall until all the keys for the level have been collected.

1. *#Door class is used to prevent the player from proceeding to next level without completing the level.*
2. **class** Door(pygame.sprite.Sprite):
3. **def** \_\_init\_\_(self, color, width, height, x, y, posx, posy):
4. super().\_\_init\_\_()
5. ***#use sprite constructor***
6. self.image = pygame.Surface([width,height])
7. self.image.fill(color)
8. *#set the position of the sprite*
9. self.rect = self.image.get\_rect()
10. **self.rect.x = x**
11. self.rect.y = y
12. self.positionx = posx
13. self.positiony = posy
14. *#end procedure*

Below is an image of the doors at the beginning of the level:

A picture containing icon

Description automatically generated

#### Update Function

The update function of the door class checks if the number of keys collected by the player is greater than ort equal to the number of keys necessary for the level and if true will remove the door from all sprite groups.

1. *#Update function checks what the level is and checks if the level is over before removing the door object*
2. **def** update(self):
3. *#Check all keys have been collected and the chest unlocked*
4. **if** game.player.gamekeys >= ((2\*(game.level+1)) + 1):
5. **self.kill()**
6. *#end function*

Below is an image when all the key value is equal to 3 on the level:

A picture containing icon

Description automatically generated

### Spike Class:

#### Constructor Function

The constructor function of the spike class defines the colour, size and coordinates through parameters and in line 13 and 15 creates 2 timers that will be used to inflict damage and change colour.

1. *#Spike class which covers a full tile and inflicts damage to the player when active*
2. **class** Spikes(pygame.sprite.Sprite):
3. **def** \_\_init\_\_(self, color, width, height, x, y, posx, posy):
4. super().\_\_init\_\_()
5. ***#Use sprite constructor***
6. self.image = pygame.Surface([width,height])
7. self.image.fill(color)
8. self.rect = self.image.get\_rect()
9. *#set the coordinates*
10. **self.rect.x = x**
11. self.rect.y = y
12. *#Set timers to change every 4 seconds between active and inactive states*
13. self.previousattacktime = pygame.time.get\_ticks()
14. self.color = color
15. **self.previousdamagetime = pygame.time.get\_ticks()**
16. *#end procedure*

Below is an image of the spikes initially retracted:

A picture containing application

Description automatically generated

#### Update Function

The update function of the spikes class uses a timer to change the colour of the spike after 4 seconds. In line 6 it checks whether the colour of the spikes were grey and then sets the colour to light blue before resetting the timer. Line 11 checks if the colour of the spikes were blue and sets the colour back to grey before resetting the timer.

1. *#Update function*
2. **def** update(self):
3. *#Check if the time is greater than 4 seconds before changing state*
4. self.currentattacktime = pygame.time.get\_ticks()
5. **if self.currentattacktime - self.previousattacktime > 4000:**
6. **if** self.color == GREY:
7. self.color = LIGHTBLUE
8. self.image.fill(self.color)
9. *#reset the timer*
10. **self.previousattacktime = self.currentattacktime**
11. **else**:
12. *#If they are already active after 4 seconds then switch back to inactive*
13. self.color = GREY
14. self.image.fill(self.color)
15. ***#reset the timer***
16. self.previousattacktime = self.currentattacktime

Below is an image of when the spikes change to “up”:

A picture containing application

Description automatically generated

### Chest Class:

#### Constructor Function

The constructor function of the chest class sets the chests coordinates and the level that the chest was instantiated on. Line 11 is used to check which level it is so that once collisions are added each level should lead to a different outcome when the chest and player collide.

1. *#Chest class used to unlock player's abilities at the end of each level*
2. **class** Chest(pygame.sprite.Sprite):
3. **def** \_\_init\_\_(self,color, width, height, x, y, level):
4. super().\_\_init\_\_()
5. **self.image = pygame.Surface([width,height])**
6. self.image.fill(color)
7. *#set the position of the sprite*
8. self.rect = self.image.get\_rect()
9. self.rect.x = x
10. **self.rect.y = y**
11. self.level = level

Below is an image of the chest class instantiated in the level as there are no enemies in the level:

A picture containing application

Description automatically generated

### Sword Enemy Class:

#### Constructor Function

The constructor function of the enemy class is used to define all the attributes of the enemy when instantiated. Thus, it is used to set the initial health, location, and size of the enemy. Line 7 to 12 are used to define the size, colour, and coordinates of the enemy while lines 13 to 16 define the health, direction, and speeds of the enemy. Line 19 sets the movement speed of the enemy to 3 pixels and lines 20 and 21 are used to start timers for attacking the player.

1. *#Melee Enemy class*
2. **class** MeleeEnemy(pygame.sprite.Sprite):
3. **def** \_\_init\_\_(self, direction, width, height, x, y, health):
4. *#call sprite constructor*
5. **super().\_\_init\_\_()**
6. *#create a sprite*
7. self.image = pygame.Surface([width,height])
8. self.image.fill(YELLOW)
9. *#set the position of the sprite*
10. **self.rect = self.image.get\_rect()**
11. self.rect.x = x
12. self.rect.y = y
13. self.health = health
14. self.direction = direction
15. **self.speed\_x = 0**
16. self.speed\_y = 0
17. *#Set timers to attack the player and set the movement speed to 3 pixels per frame*
18. *#self.previouspathtime = pygame.time.get\_ticks()*
19. self.move = 3
20. **self.previousdamagetime = pygame.time.get\_ticks()**
21. self.previousattacktime = pygame.time.get\_ticks()
22. *#end procedure*

Icon

Description automatically generated with low confidenceA picture containing graphical user interface

Description automatically generatedBelow are two images showing how the enemy moves towards the player:

Enemy moving towards the player

#### Is Close Function

The is close function finds the distance between the player and the enemy. Line 5 calculates the distance between the player and the enemy and lines 6 to 9 return a true or false depending on if the distance is greater or smaller than 300.

1. *#Method using trigonometry to check the distance between the player and the enemy and outputs whether the player is within 300 pixels or not.*
2. **def** is\_close(self):
3. lengthx = self.rect.x - game.player.rect.x
4. lengthy = self.rect.y - game.player.rect.y
5. **distance = math.sqrt((lengthx \*\* 2) + (lengthy \*\* 2))**
6. **if** distance < 300:
7. **return** True
8. **else**:
9. **return** False

#### Change Speed Procedure

The change speed procedure takes in the parameters x and y and increments the speed of the enemy by the parameters.

1. *#Method to add the speed to the object to move up, down and side to side*
2. **def** changespeed(self,x,y):
3. self.speed\_x += x
4. self.speed\_y += y

#### Move To player Procedure

The move to the player procedure is currently very basic and can be changed in the future. It simply compares the player and the enemy’s x and y coordinates and changes the enemy’s speed until both the player and the enemy have the same coordinates. Once the speed of the enemy has been changed then the speeds are added to the enemy’s x and y coordinates to move the enemy.

1. *#Method to change how the enemy moves when the player is within range of the enemy*
2. **def** movetoplayer(self, Player):
3. *#Compare the player's position to the enemy's position and change the speed of the enemy to move it closer to the player.*
4. **if** Player.rect.x - 10 > self.rect.x:
5. **self.speed\_x = 2**
6. **if** Player.rect.x - 10 < self.rect.x:
7. self.speed\_x = -2
8. **if** Player.rect.y - 10 > self.rect.y:
9. self.speed\_y = 2
10. **if Player.rect.y - 10< self.rect.y:**
11. self.speed\_y = -2
12. *# Move along x axis*
13. self.rect.x += self.speed\_x
14. *# Move along y axis*
15. **self.rect.y += self.speed\_y**

#### Update Function

The update function of the enemy is used to change how the enemy moves depending on how close the player is. Line 5 uses an if statement to check if the player is within range, and if true will call the move to player procedure to move the enemy towards the player. Line 10 checks if the enemy’s health is below 1 and if true will create an instance of the key class and then remove the enemy from all sprite groups.

1. *#update function changes behaviour depending on the range of the enemy*
2. **def** update(self):
4. *#Check if the player is 300 pixels away from the enemy*
5. **if self.is\_close() == True:**
6. *#if this is true then use the movetoplayer method to make the enemy move towards the player.*
7. self.movetoplayer(game.player)
9. *#Check if the health is greater than zero. Else it will create an instance of a key and increase the game score by 100 before removing itself.*
10. **if** self.health < 1:
11. game.score += 100
12. gamekey = Key(PINK, self.rect.x + 2, self.rect.y + 9)
13. game.all\_sprites\_group.add(gamekey)
14. game.key\_group.add(gamekey)
15. **self.kill()**
16. *#end procedure*

#### Get Health Function

The get health function returns the enemy’s current health.

1. *#method used to get the enemy's health*
2. **def** gethealth(self):
3. **return** self.health
4. *#endfunction*

#### Set Health Procedure

The set health procedure is used to set a new health for the enemy.

1. *#method to set the enemies health*
2. **def** sethealth(self, newhealth):
3. self.health = newhealth
4. *#endprocedure*

### Key Class:

#### Constructor Function

The constructor function of the key class is very basic. Each key is of a static size of 10 by 10 pixels. The x and y parameters dictate where the key will be created depending on the coordinates of the enemy that was killed to before it was created.

1. **def** \_\_init\_\_(self,color,x,y):
2. *#Initiate sprite constructor*
3. super().\_\_init\_\_()
4. self.image = pygame.Surface([10,10])
5. **self.image.fill(color)**
6. self.rect = self.image.get\_rect()
7. self.rect.y = y
8. self.rect.x = x

A picture containing graphical user interface

Description automatically generatedBelow shows the instantiation of a key after an enemy is killed:

#### Update Function

The update function of the key has only one job; to check for a collision with the player and increase the score and player’s keys once collided. As such Line 4 and 5 check for any key colliding with any player and if true then increase the number of keys and score and remove the key from all sprite groups.

1. *#Update function check for collision and increase game score*
2. **def** update(self):
3. *#Check for collision with player and add one to key attribute in player*
4. key\_hit\_group = pygame.sprite.groupcollide(game.key\_group, game.player\_group, False, False)
5. **for self in key\_hit\_group:**
6. *#increase score and then remove itself.*
7. game.player.gamekeys += 1
8. game.score += 50
9. self.kill()

A picture containing graphical user interface

Description automatically generatedFollowing the creation of a key, when the player collides with the key, the key is removed from the screen and the keys tally increases by 1:

### Bow Enemy Class:

#### Constructor Function

The constructor function of the enemy class is used to define all the attributes of the enemy when instantiated. Thus, it is used to set the initial health, location, and size of the enemy. Line 7 to 12 are used to define the size, colour, and coordinates of the enemy while lines 13 to 14 define the health and direction of the enemy. Lines 15 and 16 are used to start timers for attacking the player.

1. *#Bow enemy class*
2. **class** BowEnemy(pygame.sprite.Sprite):
3. **def** \_\_init\_\_(self, direction, width, height, x, y, health):
4. *#call sprite constructor*
5. **super().\_\_init\_\_()**
6. *#create a sprite*
7. self.image = pygame.Surface([width,height])
8. self.image.fill(ORANGE)
9. *#set the position of the sprite*
10. **self.rect = self.image.get\_rect()**
11. self.rect.x = x
12. self.rect.y = y
13. self.health = health
14. self.direction = direction
15. **self.previousdamagetime = pygame.time.get\_ticks()**
16. self.previousattacktime = pygame.time.get\_ticks()
17. *#end procedure*

A picture containing graphical user interface

Description automatically generatedBelow is an image of what the bow enemy looks like:

#### Update Function

The update function of the bow enemy is used to check if the player is within range and if true to create an enemy bullet which is directed towards the player. Line 3 checks if the player is in range. When the player is in range then the x and y distances are calculated. From this, Line 7 calculates the angle between the enemy and the player. Line 8 converts this into degrees and lines 9 and 10 then set the x and y speeds using cosine and sine for the bullet. Line 12 starts a timer that creates an enemy bullet with these calculated speeds from the location of the centre of the enemy every second in lines 12 to 18. Lines 20 to 25 check if the enemy’s health is below 1 and if true will remove the enemy from all sprite groups and increment the score while instantiating a key.

1. *#Update function used to check if the player is in range and calculate the speeds for the bullet fired*
2. **def** update(self):
3. **if** self.is\_close() == True:
4. *#If player in range then calculate y distance and x distance as well as distance*
5. **xdiff = (game.player.rect.x+20) - (self.rect.x+20)**
6. ydiff = (game.player.rect.y+20) - (self.rect.y+20)
7. self.angle = (180 / math.pi) \* -math.atan2(ydiff, xdiff) - 90
8. self.degrees = math.degrees(self.angle)
9. xspeed = 5 \* math.cos(self.degrees)
10. **yspeed = 5 \* math.sin(self.degrees)**
11. *#Every 1 second an enemy bullet object is created travelling in the direction using the xspeed and yspeed.*
12. self.currentattacktime = pygame.time.get\_ticks()
13. **if** self.currentattacktime - self.previousattacktime > 1000:
14. game.ebullet = EnemyBullet(RED, xspeed, yspeed, self.rect.x+20, self.rect.y+20)
15. **game.all\_sprites\_group.add(game.ebullet)**
16. game.enemybullet\_group.add(game.ebullet)
17. *#reset the timer*
18. self.previousattacktime = self.currentattacktime
19. *#Check if the health is greater than zero. Else it will create an instance of a key and increase the game score by 100 before removing itself.*
20. **if self.health < 1:**
21. game.score += 100
22. gamekey = Key(PINK, self.rect.x + 2, self.rect.y + 9)
23. game.all\_sprites\_group.add(gamekey)
24. game.key\_group.add(gamekey)
25. **self.kill()**
26. *#end procedure*

Below shows how the enemy bullets are fired from the bow enemy towards the player, however, they clearly do not travel directly towards the player but above him:

Graphical user interface

Description automatically generated with low confidence

#### Get Health Function

The get health function returns the enemy’s current health.

1. *#method used to get the enemy's health*
2. **def** gethealth(self):
3. **return** self.health
4. *#endfunction*

#### Set Health Procedure

The set health procedure is used to set a new health for the enemy.

1. *#method to set the enemies health*
2. **def** sethealth(self, newhealth):
3. self.health = newhealth
4. *#endprocedure*

#### Is Close Function

The is close function finds the distance between the player and the enemy. Line 5 calculates the distance between the player and the enemy and lines 6 to 9 return a true or false depending on if the distance is greater or smaller than 400.

1. *#Method using trigonometry to check the distance between the player and the enemy and outputs whether the player is within 300 pixels or not.*
2. **def** is\_close(self):
3. lengthx = self.rect.x - game.player.rect.x
4. lengthy = self.rect.y - game.player.rect.y
5. **distance = math.sqrt((lengthx \*\* 2) + (lengthy \*\* 2))**
6. **if** distance < 400:
7. **return** True
8. **else**:
9. **return** False

A picture containing graphical user interface

Description automatically generatedA picture containing application

Description automatically generatedBelow are images showing how the bow enemy only shoots when the player is in range:

### Enemy Bullet Class:

#### Constructor Function

The constructor function of the enemy bullet class is almost identical to the player bullet class. However, new parameters are used to determine the initial x and y coordinates of the bullet as many bow enemies can be in each level compared to inly one player.

1. *#Method to initiate bullet objects*
2. **def** \_\_init\_\_(self, color, speedx, speedy, x, y):
3. *#call the sprite constructor*
4. super().\_\_init\_\_()
5. ***#set the size of the bullet***
6. self.image = pygame.Surface([6,4])
7. self.image.fill(color)
8. self.rect = self.image.get\_rect()
9. self.speedx = speedx
10. **self.speedy = speedy**
11. self.rect.y = y
12. self.rect.x = x

#### Update Function

The update function is again used to increment the coordinates of the bullet by its initial speeds in lines 4 to 6, while in lines 8 to 15 it checks if the bullet has left the window’s coordinates and removes the bullet from every sprite group.

1. *#Update method that moves the bullet according to the speed set by the bow enemy*
2. **def** update(self):
3. *#move the bullet up and down*
4. self.rect.y += self.speedy
5. ***#move the bullet left or right***
6. self.rect.x += self.speedx
7. *#Check if it travel past the boundaries and kills the sprite*
8. **if** self.rect.x > 1000:
9. self.kill()
10. **if self.rect.x < 0:**
11. self.kill()
12. **if** self.rect.y > 1000:
13. self.kill()
14. **if** self.rect.y < 0:
15. **self.kill()**

Logo

Description automatically generatedBelow is an image of the bullet object compared to the player object:

### Errors

#### Chart Description automatically generated with medium confidenceHealth Bar

The first error with the health bar was that it was that the screen was refreshing over the drawn health bar (as seen above). This is due to the health bar procedure being called within the update function of the player class which is called in the run logic function of the game class, which is called before the display function, thus all objects on the screen are displayed over the health bar. Thus, to fix the health bar from being drawn over, I have amended the code in the following way:

1. *#Create a method to display all sprites and text*
2. **def** display(self, screen):
3. [previous code…]
4. *# background image.*
5. **screen.fill(BLACK)**
6. **if** **not** self.game\_over:
7. *#When the game is running Display text on the right handside of the screen with the score and keys of the player*
8. font = pygame.font.Font(None, 24)
9. score = font.render('SCORE:'+str(self.getscore()), 1, WHITE)
10. **keys = font.render('KEYS:'+str(self.player.getkeys()), 1, WHITE)**
11. *#Draw the player function for the health bar of the player*
12. self.player.advanced\_health()
13. *#Draw the text of the player's health over the health bar*
14. health = font.render(str(self.player.current\_health), 1, WHITE)
15. **screen.blit(score, (1050,500))**
16. screen.blit(keys, (1050,550))
17. screen.blit(health, (1081, 51))
18. *# --- Drawing code for sprites*
19. self.all\_sprites\_group.draw(screen)
20. **[previous code…]**

Graphical user interface

Description automatically generatedThe new code is within line 12 which now allows the health bar to be drawn above the background but still remain below the text that will overlay the health bar. This therefore fixes the first error. As shown in the image below:

Graphical user interface

Description automatically generated

The second error with the health bar is when the max health of the player is changed while the game is running then the health bar will extend outside of the window (as shown above). To prevent this, we must update the health ratio variable while the game is running to prevent this. This is shown in the following code:

1. *#Update method for player to control movement and damage taken*
2. **def** update(self):
3. [previous code...]
4. *#Refresh the health bar so an increase in the maximum health resets the health bar ratio*
5. **self.health\_ratio = self.maximum\_health/ self.health\_bar\_length**
6. [previous code...]

This code is added to the player’s update function and only requires one line. Line 5 updates the health\_ratio variable by defining it every time the update function is called which is every time the game loop executes.

#### Player Movement

When any of the movement keys are pressed, the player moves in that direction. However, what happens is that the player accelerates when a key is pressed instead of moving at a constant speed. This is because the move procedure is called every update of the game loop and the speed of the player is increases every time the key is pressed. This is because the change speed procedure uses addition to change the speed of the player. In order to fix this, we must reset the player’s speed to zero after each update.

1. *#Update method for player to control movement and damage taken*
2. **def** update(self):
3. [previous code...]
4. *#Reset the speeds to prevent the player accelerating*
5. **self.speed\_x = 0**
6. self.speed\_y = 0
7. *#end procedure*

Lines 5 and 6 reset the x and y speed of the player and come at the end of the update function to allow the move function to allow the player to move.

#### Enemy Coordinates During Level Instantiation

The error with the current enemy generation during the level setup procedure is that the enemies are replacing inner wall sprites and should only be replacing empty spaces. In order to stop this from happening we can introduce a new if statement to check if the element in the array they are replacing is another sprite or if it empty space. If it is empty space, then it will replace that element and won’t if it isn’t.

1. *#Create a method that creates all the objects according to the 2D array in the game class.*
2. **def** levelsetup(self):
3. [previous code...]
4. enemies = 0
5. **while enemies != ((2\*(self.level+1)) + 1):**
6. *#Find two random numbers between 1 and 23*
7. xpos = random.randint(1,23)
8. ypos = random.randint(1,23)
9. *#Check if the numbers correspond to coordinates of walls or other sprites*
10. **if self.levels[self.level][xpos][ypos] == 0:**
11. *#set the coordinates to a random number to spawn either a Melee enemy or Bow enemy*
12. self.levels[self.level][xpos][ypos] = random.randint(3,4)
13. enemies = enemies +1
14. [previous code...]

The change has been made to line 10 which checks the element in that index and only creates an enemy if that element is a 0.

#### Direction of Enemy Bullets

The problem with the enemy bullets is that when a bow enemy shoots bullets, the bullets will go in completely the opposite direction to the player’s position. This is shown in the following image:

A picture containing text, first-aid kit

Description automatically generated

The source of the problem comes from the trigonometric functions in the following lines of code:

1. self.angle = (180 / math.pi) \* -math.atan2(ydiff, xdiff) - 90
2. self.degrees = math.degrees(self.angle)
3. xspeed = 5 \* math.cos(self.degrees)
4. **yspeed = 5 \* math.sin(self.degrees)**

I believe the problem arises between lines 1 and 2 where the inverse tan function calculates the angle in radians and is then converted into degrees in line 2. I currently have not found a solution but may require using a different maths library to use only radians when using inverse tan that covers all 360 degrees that the player can be in rather than the limited 90-degree window in which the function works. This will be improved upon in iteration 2.

### Conclusion

Iteration 1 successfully implemented each primary class and completed the task of moving the player and enemies around as well as displaying them correctly. As such iteration one has completed **success criteria 2, 3, 4, 6, 7, 8, 11, 12, 13, 14, 17, 20, 27,28, 32 and 40.** Only one major error remaining, the inability for the bow enemies to shoot at the player, that needs to be solved in the following iteration. Following the completion of displaying all the classes, the next step is to now add collisions and further features to improve the enjoyment of the game.

## Iteration 2 – Collisions, Level Building and Menu:

Iteration 2 has the main focus of implementing all collisions and making sure that each objects updates properly on screen. Iteration 2 also adds a new menu screen and game over screen with the inclusion of high scores. The majority of classes will be completed in this iteration, with any requirements not completed in iteration 1 to be completed in this iteration. **The requirements that should be completed in this iteration are 1, 5, 9, 10, 15, 16, 21, 22, 23, 24, 25, 26, 29, 30, 31, 33, 34, 35, 36, 37, 38, 39.**

### Menu:

#### Libraries and constants

The following constants in line 10 to 22 are used for the colours of various sprites and text throughout the game, including in the menu screen. Lines 1 to 5 are used to import python libraries that are used to simplify certain functions and procedures by not having to define them from scratch. Line 7 and 9 use the os library to find the directory of the python file and find the folder in the same directory as the python file named images. These will later be used to organise and locate the sprite sheets that will be used for our sprites. Lines 24, 26 and 28 are used to identify the background image, the high score text file and the high score folder in which the text file is stored. Lines 30 to 39 initiate the pygame library as well as set the dimensions of the window and set the window name before setting up the clock that will be used to refresh the screen at a constant rate.

1. **import** pygame
2. **import** random
3. **import** os
4. **import** sys
5. **import math**
6. *#use os to find the path of this file:*
7. current\_path = os.path.dirname(\_\_file\_\_)
8. *#use os to find the folder called 'images' in the same folder as this file*
9. image\_path = os.path.join(current\_path, 'images')
10. ***# Define colours as tuples***
11. BLACK = (0, 0, 0)
12. WHITE = (255, 255, 255)
13. GREEN = (0, 255, 0)
14. RED = (255, 0, 0)
15. **YELLOW = (255, 255, 0)**
16. PINK = (255,20,147)
17. PURPLE = (75,0,130)
18. BROWN = (150, 75, 0)
19. ORANGE = (230,165,0)
20. **BLUE = (30,144,255)**
21. LIGHTBLUE = (173, 216, 240)
22. GREY = (180,180,180)
23. *#use os to find the menu background image in the images folder*
24. BACKGROUND\_IMAGE = pygame.image.load(os.path.join(image\_path, 'Menu background.png'))
25. ***#Define the highscore file as highscore.txt***
26. HS\_FILE = "highscore.txt"
27. *#Use os to find the highscore file*
28. hs\_path = os.path.join(current\_path, 'highscore')
29. *#Initiate pygame module*
30. **pygame.init()**
31. *# Set the width and height of the screen [width, height]*
32. size = (1200, 1000)
33. screen\_width = 1200
34. screen\_height = 1000
35. **screen = pygame.display.set\_mode(size)**
36. *#Set the window name to DUNGEON ESCAPE*
37. pygame.display.set\_caption("Dungeon Escape")
38. *#Define the clock used for what refresh rate the game runs at.*
39. clock = pygame.time.Clock()

#### Text Function

The text function is used to create text that will appear on the screen. As such, line 4 takes the input text and displays the text on the surface and returns the text position on the surface.

1. *#Create a function used to format text used in buttons*
2. **def** text\_objects(text,font):
3. *#Render the text input as white and return the position of the surface.*
4. textSurface = font.render(text, True, WHITE)
5. **return textSurface, textSurface.get\_rect()**
6. *#end procedure*

#### Menu Procedure

The menu procedure is vital as it fulfils success criteria number 1. The menu procedure must create a menu screen that involves buttons to start and quit the game as well as a title. Lines 4 to 12 are used to create a loop that refreshes 60 times a second. This loop is used to display a background image, as in line 17, as well as a title in lines 19 to 21. Two buttons are created in lines 24 and 25, with the first button used to start the game and the second to quit the window. Lines 27 and 29 are used to refresh the screen at 60hz.

1. *#Create a function for the Main Menu of the game.*
2. *#Display the buttons for Starting the game and Quiting.*
3. **def** game\_intro():
4. intro = True
5. ***#Use while loop so that the Menu refreshes 60 times per second***
6. **while** intro:
7. **for** event **in** pygame.event.get(): *# User did something*
8. **if** event.type == pygame.QUIT: *# If user clicked close*
9. intro = False *# Flag that we are done so we exit this loop and quit the game*
10. **elif event.type==pygame.KEYDOWN:**
11. **if** event.key==pygame.K\_ESCAPE:
12. intro=False
14. *#Drawing the menu screen*
15. **screen.fill(BLACK)**
16. *#Draw the Background image for the background*
17. screen.blit(BACKGROUND\_IMAGE, [0,0])
18. *#Set the font and text for the title and center it.*
19. font = pygame.font.Font('freesansbold.ttf', 84)
20. **text = font.render(str("DUNGEON ESCAPE"), 1, WHITE)**
21. text\_rect = text.get\_rect(center=(screen\_width/2, screen\_height/6))
22. screen.blit(text, text\_rect)
23. *#Draw the buttons for starting the game and quiting*
24. button\_1("START GAME",475,420,250,60,WHITE,GREY,"1")
25. **button\_1("QUIT",475,490,250,60,WHITE,GREY,"Q")**
26. *#Display the image in the window*
27. pygame.display.flip()
28. *#Refresh at 60Hz*
29. clock.tick(60)
30. ***#end procedure***

Below is an image of the menu function:

A brick wall with a sign on it

Description automatically generated with medium confidence

#### Button Function

The button function is a complex function that checks for the user’s cursor position and changes colour when the cursor hovers over the button. This function also uses an identification parameter which it uses to execute different instructions depending on which button is pressed. This is demonstrated in line 13 and 15 where if the button is left clicked on by the mouse and the action of the button is 1 then the button will execute the gameloop function. Line 25 to 28 are used to take the message parameter and display it as text centred in the middle of the button.

1. *#Create a button function that interacts with the mouse. When hovered over by the mouse change colour and define the x position, y position and width and height of the Button.*
2. *#Also specify what the button does when it is pressed and what text is in the button*
3. **def** button\_1(message,xpos,ypos,width,height,inactivecolor,activecolor,action1=None):
4. *#Find the position of the mouse*
5. **mouse = pygame.mouse.get\_pos()**
6. *#Check if the mouse is being clicked*
7. click = pygame.mouse.get\_pressed()
8. *#Check if the mouse's position is the same as the button's position*
9. **if** xpos+width >mouse[0] > xpos **and** ypos+height > mouse[1] > ypos:
10. ***#Draw the button with its inactive colour***
11. pygame.draw.rect(screen, inactivecolor,(xpos,ypos,width,height),5)
12. *#Check if the mouse is clicked:*
13. **if** click[0] == 1 **and** action1 !=None:
14. *#Check what the action of the button does.*
15. **if action1 == "1":**
16. *#If the button has action 1 then it will start the gameloop when pressed.*
17. gameloop()
18. **elif** action1 == "Q":
19. *#if the button has the action Q then exit the window*
20. **pygame.quit()**
21. **else**:
22. *#When the button is not hovered over by the mouse then draw the button with it's active colour*
23. pygame.draw.rect(screen, activecolor,(xpos,ypos,width,height),5)
24. *#Draw the text of the buutton in the middle of the button*
25. **smallText = pygame.font.Font("freesansbold.ttf",30)**
26. textSurf, textRect = text\_objects(message, smallText)
27. textRect.center = ( (xpos+(width/2)), (ypos+(height/2)) )
28. screen.blit(textSurf, textRect)
29. *#end function*

Below are two images showing how the colour of the button changes when hovered over by a mouse:

Graphical user interface

Description automatically generatedGraphical user interface

Description automatically generated

### Game Loop Function:

This function is called by the button in the menu screen. Thus, it must contain everything that is used for the game. Line 4 sets up the game loop that will repeat at 60 times per second. Line 8 is a placeholder that contains all the classes in this iteration. The rest of the game loop remains identical to the previous iteration. Outside of the game loop and at line 24 the menu procedure is called as the second to last line in the code. Finally, Line 26 is the pygame.quit function that exits the window after the game loop is finished.

1. *#Create a game loop function that is called after the Menu*
2. **def** gameloop():
3. *#Loop until the user clicks the close button.*
4. done = False
5. ***# Used to manage how fast the screen updates***
6. clock = pygame.time.Clock()
8. [All classes]
10. ***# -------- Main Program Loop -----------***
11. **while** **not** done:
12. *# --- Main event loop*
13. *#Declare done = True when the game\_over is true returning the player to the Menu*
14. done = game.eventprocess()
15. ***# --- Game logic should go here***
16. *#Runs all the logic for the game loop*
17. game.runlogic()
18. *#draw the screen*
19. *#Draws everything for the game and updates the screen*
20. **game.display(screen)**
21. *# --- Limit to 60 frames per second*
22. clock.tick(60)
23. *#Initiate the Menu function to start with the menu*
24. game\_intro()
25. ***# Close the window and quits.***
26. pygame.quit()

### Game Class:

#### Constructor Function

The constructor function of the game class has had many elements added to it. It now includes the addition in line 8 used to load the high score text file, and line 10 is an array used to check in which levels chests have been unlocked. This is useful as it is used as a trigger to display text when one of the elements is set to true. Line 14 is also used to allow for the player to travel back a level without needing to have to recomplete the level that they returned to and as such the array allows the level setup function to know when to create enemies.

1. *#Create A Game class that holds all the Sprite groups and level information*
2. *#Use a game object to establish a game loop that is used to setup the game*
3. **class** Game(object):
4. **def** \_\_init\_\_(self):
5. ***#Define all the atributes of the Game class***
6. [previous code...]
7. *#Call the load data method to be used to find and store high scores*
8. self.load\_data()
9. *#Create a list of which chests have been unlocked*
10. **self.chestunlocked = [False, False, False, False, False]**
11. *#Set a timer for how long to display text*
12. self.previoustexttime = pygame.time.get\_ticks()
13. *#Create a list of all the levels that have been completed*
14. self.levelcomplete = [False, False, False, False, False]
15. **[previous code...]**

Further to these changes, more levels have been included within the constructor function for the game class. All the levels follow the same square format of a 25 by 25 grid with the outer grid elements consisting of walls while the inside of the grid will vary from each level. Below are shown the next 4 levels included within the game:

1. self.level2 = [
2. [1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1],
3. [1,0,0,0,0,0,0,0,0,2,0,0,0,0,0,0,0,0,2,0,0,0,0,0,1],
4. [1,0,0,0,0,0,0,0,0,2,0,0,0,0,0,0,0,0,2,0,0,0,0,0,1],
5. **[1,0,0,0,2,2,2,2,2,2,0,0,0,0,0,0,0,0,2,0,0,2,2,2,1],**
6. [1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1],
7. [1,0,0,0,0,0,0,0,0,2,2,2,2,0,0,0,0,0,0,0,0,0,0,0,1],
8. [1,0,0,0,0,0,0,0,0,2,0,0,0,0,0,0,0,0,2,0,0,0,0,0,1],
9. [1,0,0,0,0,0,2,2,2,2,0,0,0,0,0,0,0,0,2,0,0,2,2,2,1],
10. **[1,0,0,0,0,0,2,0,0,0,0,0,0,0,0,0,0,2,2,0,0,0,0,0,1],**
11. [1,0,0,0,0,0,2,0,2,2,2,2,2,2,2,2,0,2,0,0,0,0,0,0,1],
12. [1,0,0,0,0,0,2,2,2,2,0,0,0,0,2,2,2,2,0,0,0,0,0,0,1],
13. [6,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,6],
14. [6,0,0,0,0,2,2,2,2,2,0,0,0,0,2,2,2,2,2,2,2,2,0,0,6],
15. **[6,0,0,0,0,2,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,2,0,0,6],**
16. [1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,2,0,0,1],
17. [1,2,0,2,2,2,0,0,0,0,0,0,0,0,0,0,0,2,2,2,2,2,0,0,1],
18. [1,0,0,0,0,2,0,0,0,0,0,0,0,0,0,0,0,2,0,0,0,0,0,0,1],
19. [1,0,0,0,0,2,0,0,0,0,0,0,0,0,0,2,2,2,0,0,0,2,2,2,1],
20. **[1,0,0,0,0,2,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,2,0,0,1],**
21. [1,0,0,0,0,2,0,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,0,0,1],
22. [1,0,0,0,0,2,0,0,0,0,0,0,0,0,2,0,0,0,0,0,0,0,0,0,1],
23. [1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1],
24. [1,0,0,0,0,2,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1],
25. **[1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1]**
27. ]
29. self.level3 = [
30. **[1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1],**
31. [1,0,0,0,0,0,0,0,0,2,0,0,0,0,0,0,2,0,0,0,0,0,0,0,1],
32. [1,0,0,0,0,0,0,0,0,2,0,0,0,0,0,0,2,0,0,0,0,0,0,0,1],
33. [1,0,0,0,0,0,0,0,0,2,0,0,0,0,0,0,2,0,0,0,0,0,0,0,1],
34. [1,0,0,0,0,0,0,0,0,2,0,0,0,0,0,0,2,0,0,0,0,0,0,0,1],
35. **[1,0,0,0,0,0,2,2,2,2,0,0,0,0,0,0,7,0,0,0,0,0,0,0,1],**
36. [1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,7,0,0,0,0,0,0,0,1],
37. [1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,2,0,0,0,0,0,0,0,1],
38. [1,0,0,0,0,0,0,0,0,2,0,0,0,0,0,0,2,2,2,7,7,2,2,2,1],
39. [1,0,0,0,0,0,0,0,0,2,0,0,0,0,0,0,2,0,0,0,0,0,0,0,1],
40. **[1,0,0,0,0,0,0,0,0,2,2,2,2,2,2,2,2,0,0,0,0,0,0,0,1],**
41. [6,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,7,0,0,0,0,0,0,0,6],
42. [6,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,7,0,0,0,0,0,0,0,6],
43. [6,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,7,0,0,0,0,0,0,0,6],
44. [1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,7,0,0,0,0,0,0,0,1],
45. **[1,2,2,2,2,0,0,0,2,2,2,2,2,2,2,2,2,0,0,0,0,0,0,0,1],**
46. [1,0,0,0,0,0,0,0,2,0,0,0,0,0,0,0,2,0,0,0,0,0,0,0,1],
47. [1,0,0,0,0,0,0,0,2,0,0,0,0,0,0,0,7,0,0,0,0,0,0,0,1],
48. [1,0,0,0,0,2,2,2,2,0,0,0,0,0,0,0,7,0,0,0,0,0,0,0,1],
49. [1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,2,2,7,7,2,2,2,2,1],
50. **[1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,2,0,0,0,0,0,0,0,1],**
51. [1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,2,0,0,0,0,0,0,0,1],
52. [1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,2,0,0,0,0,0,0,0,1],
53. [1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1]
55. **]**
57. self.level4 = [
58. [1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1],
59. [1,0,0,0,0,0,2,0,0,0,0,7,0,0,0,0,0,0,0,0,0,0,0,0,1],
60. **[1,0,0,0,0,0,2,0,0,0,0,7,0,0,0,0,0,0,0,0,0,0,0,0,1],**
61. [1,0,0,0,0,0,2,0,0,0,0,2,0,0,0,0,0,0,0,0,0,0,0,0,1],
62. [1,0,0,0,0,0,2,0,0,0,0,2,0,0,0,0,0,2,2,0,0,0,0,0,1],
63. [1,0,0,0,0,0,7,0,0,0,0,2,0,0,0,0,0,2,2,0,0,0,0,0,1],
64. [1,0,0,0,0,0,7,0,0,0,0,2,0,0,0,0,0,0,0,0,0,0,0,0,1],
65. **[1,0,0,0,0,0,7,0,0,0,0,2,0,0,0,0,0,0,0,0,0,0,0,0,1],**
66. [1,0,0,0,0,0,2,0,0,0,0,2,0,0,0,0,0,0,0,0,0,0,0,0,1],
67. [1,0,0,0,0,0,2,0,0,0,0,2,0,0,0,0,0,0,0,0,0,0,0,0,1],
68. [1,2,2,2,2,2,2,0,0,0,0,2,2,2,2,2,2,2,2,2,2,2,2,2,1],
69. [6,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,2,0,0,0,0,0,6],
70. **[6,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,2,0,0,0,0,0,6],**
71. [6,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,2,0,0,0,0,0,6],
72. [1,2,2,2,2,2,2,0,0,0,0,0,0,0,0,0,0,0,2,0,0,0,0,0,1],
73. [1,0,0,0,0,0,2,0,0,0,0,0,0,0,0,0,2,2,2,7,7,7,7,7,1],
74. [1,0,0,0,0,0,2,0,0,0,0,0,0,0,0,0,0,0,7,0,0,0,0,0,1],
75. **[1,0,0,0,0,0,7,0,0,0,0,0,0,0,0,0,0,0,7,0,0,0,0,0,1],**
76. [1,0,0,0,0,0,7,0,0,0,0,0,0,0,0,0,0,0,7,0,0,0,0,0,1],
77. [1,0,0,0,0,0,7,0,0,0,0,0,0,0,0,0,0,0,7,0,0,0,0,0,1],
78. [1,0,0,0,0,0,2,0,0,0,0,0,0,0,0,0,0,0,2,0,0,0,0,0,1],
79. [1,0,0,0,0,0,2,0,0,0,0,0,0,0,0,0,0,0,2,0,0,0,0,0,1],
80. **[1,0,0,0,0,0,2,0,0,0,0,0,0,0,0,0,0,0,2,0,0,0,0,0,1],**
81. [1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1]
83. ]
85. **self.level5 = [**
86. [1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1],
87. [1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1],
88. [1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1],
89. [1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1],
90. **[1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1],**
91. [1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1],
92. [1,0,0,0,0,0,2,2,7,7,7,7,7,7,7,7,7,2,2,0,0,0,0,0,1],
93. [1,0,0,0,0,0,2,0,0,0,0,0,0,0,0,0,0,0,2,0,0,0,0,0,1],
94. [1,0,0,0,0,0,7,0,0,0,0,0,0,0,0,0,0,0,7,0,0,0,0,0,1],
95. **[1,0,0,0,0,0,7,0,0,0,0,0,0,0,0,0,0,0,7,0,0,0,0,0,1],**
96. [1,0,0,0,0,0,7,0,0,0,0,0,0,0,0,0,0,0,7,0,0,0,0,0,1],
97. [1,0,0,0,0,0,7,0,0,0,0,0,0,0,0,0,0,0,7,0,0,0,0,0,6],
98. [1,0,0,0,0,0,7,0,0,0,0,0,0,0,0,0,0,0,7,0,0,0,0,0,6],
99. [1,0,0,0,0,0,7,0,0,0,0,0,0,0,0,0,0,0,7,0,0,0,0,0,6],
100. **[1,0,0,0,0,0,7,0,0,0,0,0,0,0,0,0,0,0,7,0,0,0,0,0,1],**
101. [1,0,0,0,0,0,7,0,0,0,0,0,0,0,0,0,0,0,7,0,0,0,0,0,1],
102. [1,0,0,0,0,0,7,0,0,0,0,0,0,0,0,0,0,0,7,0,0,0,0,0,1],
103. [1,0,0,0,0,0,2,0,0,0,0,0,0,0,0,0,0,0,2,0,0,0,0,0,1],
104. [1,0,0,0,0,0,2,2,7,7,7,7,7,7,7,7,7,2,2,0,0,0,0,0,1],
105. **[1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1],**
106. [1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1],
107. [1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1],
108. [1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1],
109. [1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1],
110. **[1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1]**
112. ]

In addition to the extra 2D arrays for more levels, each level is then added to the levels array which is used to increment after the completion of each level. Lines 6 to 8 are added to create the player for the first level, which will then continue to traverse each level after.

1. *#Create a list of all the levels*
2. self.levels = [self.level1, self.level2, self.level3,self.level4, self.level5]
3. *#Initiate the level setup for the first level*
4. self.levelsetup()
5. ***#Create an object of the player class and add him to Sprite groups***
6. self.player = Player(WHITE, 40, 40,40,500,100,0,0,0)
7. self.all\_sprites\_group.add(self.player)
8. self.player\_group.add(self.player)
9. *#end method*

A picture containing text, hand, clock, scoreboard

Description automatically generatedA screenshot of a game

Description automatically generated with medium confidenceA screenshot of a computer

Description automatically generated with low confidenceA picture containing graphical user interface

Description automatically generatedImages of each level are shown below:

#### Load Data Procedure

The load data procedure is a procedure used to open and read from the high score text file stored in a folder in the same directory as the python file. Line 4 is used to open this file and is set to read only. Line 5 to 7 are used to check if the text within the file can be converted into an integer and if not, line 8 to 10 will reset the high score to 0 if that is not possible.

1. *#Create a method to read the highscore file and set the highscore in the game*
2. **def** load\_data(self):
3. *#load high score file*
4. f = open(os.path.join(hs\_path, HS\_FILE), "r")
5. **try:**
6. *#If file has data in it then set the highscore*
7. self.highscore = int(f.readline())
8. **except**:
9. *#Else if this doesn't work then set the highscore to 0*
10. **self.highscore = 0**
11. *#end method*

#### Level Delete Procedure

The level delete procedure is used to remove all sprites from the screen once the player has finished a level. As such it empties every sprite group before adding back the player to the all sprites group and updating every group in Lines 18 and 19.

1. *#Method to delete each level when the player completes it*
2. **def** leveldelete(self):
3. *#Remove all sprites and readd the player*
4. self.all\_sprites\_group.empty()
5. **self.wall\_group.empty()**
6. self.spike\_group.empty()
7. self.enemy\_group.empty()
8. self.bullet\_group.empty()
9. self.menemy\_group.empty()
10. **self.key\_group.empty()**
11. self.portal\_group.empty()
12. self.door\_group.empty()
13. self.sword\_group.empty()
14. self.chest\_group.empty()
15. **self.boss\_group.empty()**
16. self.enemybullet\_group.empty()
17. self.ground\_group.empty()
18. self.all\_sprites\_group.add(self.player)
19. self.all\_sprites\_group.update()

#### Run Logic Function

The run logic function has been adapted to now include the boss level and the chest unlocked as well as the level complete arrays. Line 5 checks if all the enemies in the level have been defeated and if true will then check if the number of boss enemies is zero. If both are true, then the level complete array will be set to true for that level and then lines 8 to 17 are used to create 1 chest once the level complete element for that level is true. Lines 25 to 30 check if the player has moved through the door to the next level and when true, will check that this is not the last level and then set that level’s chest unlocked element to false, in order to stop displaying text on the screen, before incrementing the level by 1. Lines 32 to 36 are used to reposition the player to the correct location before the new level is generated and reset the number of keys the player has. Lines 41 to 47 check if the player has gone back to a previous level and if true then relocate the player, reset the keys and perform the level delete and level setup procedures.

1. *#Method that processes all interactions not within other classes*
2. **def** runlogic(self):
3. **if** **not** self.game\_over:
4. *#When the game is running and all the enemies and boss are dead create a chest*
5. **if len(self.enemy\_group) == 0:**
6. **if** len(self.boss\_group) == 0:
7. self.levelcomplete[self.level] = True
8. **if** self.levelcomplete[self.level] == True:
9. **if** len(self.chest\_group) == 0:
10. ***#Use count to only create one chest per level***
11. **if** self.count == 0:
12. self.count+=1
13. **print**(self.count)
14. *#create chest object and add it to sprite groups*
15. **self.chest = Chest(BROWN, 40, 40, 460,440, self.level)**
16. self.all\_sprites\_group.add(self.chest)
17. self.chest\_group.add(self.chest)
18. *# Move all the sprites*
19. self.all\_sprites\_group.update()
20. ***#Check if the player has died***
21. **if** len(self.player\_group) == 0:
22. *#If true, then end the game*
23. self.game\_over = True
24. *#Change the level when the player goes past the boundaries(goes through the doors)*
25. **if self.player.rect.x > 1000:**
26. *#Check if this is the last level and if not set to true and increase the level*
27. **if** self.level != (len(self.levels)-1):
28. self.levelcomplete[self.level] = True
29. self.chestunlocked[self.level] = False
30. **self.level += 1**
31. *#Reset the room keys needed to proceed to next levels*
32. self.player.gamekeys = 0
33. self.player.rect.x = 40
34. *#Run level delete then create the next level*
35. **self.leveldelete()**
36. self.levelsetup()
37. **else**:
38. *#If the final level then end the game*
39. self.game\_over = True
40. ***#Check if the player goes back a level***
41. **elif** self.player.rect.x < 0:
42. *#set the level back one and then setup the new level*
43. self.chestunlocked[self.level] = False
44. self.level -= 1
45. **self.leveldelete()**
46. self.player.rect.x = 960
47. self.levelsetup()
48. *#end method*

#### Display Procedure

The display procedure remains mostly unchanged, however, now includes the addition of the high score text. Line 8 checks if the current score is greater than the high score and if true will set the new high score to the current score. Lines 11 to 13 are then used to change the high score within the text file before lines 14 to 16 display the new high score on the screen. Lines 24 to 48 are all the same in that they check which level it is and then display differing text when the chest unlocked element for that level is set to true. Finally, line 49 updates the screen.

1. *#Create a method to display all sprites and text*
2. **def** display(self, screen):
3. *# background image.*
4. screen.fill(BLACK)
5. **if self.game\_over:**
6. [previous code...]
7. *#Check if the score is greater than the old highscore and set the highscore as the score*
8. **if** self.score > self.highscore:
9. self.highscore = self.score
10. ***#Open the highscore file***
11. f = open(os.path.join(hs\_path, HS\_FILE), "w")
12. *#Change the file to have new high score*
13. f.write(str(self.score))
14. screen.blit(self.hs, self.text\_rect)
15. **screen.blit(text, text\_rect2)**
16. screen.blit(score, (520,500))
17. **if** **not** self.game\_over:
18. [previous code...]
19. *# --- Drawing code for sprites*
20. **self.all\_sprites\_group.draw(screen)**
21. *#Check which chest has been unlocked and display text until player moves to next level*
22. **if** self.chestunlocked[self.level] == True:
23. **if** self.level == 0:
24. **font3 = pygame.font.Font(None, 48)**
25. msg = font3.render('FIREBALL UNLOCKED', 1, BLUE)
26. text\_rect = msg.get\_rect(center=((screen\_width-200)/2, screen\_height/6))
27. screen.blit(msg, text\_rect)
28. **if** self.level == 1:
29. **font3 = pygame.font.Font(None, 48)**
30. msg = font3.render('SWORD RADIUS INCREASED', 1, BLUE)
31. text\_rect = msg.get\_rect(center=((screen\_width-200)/2, screen\_height/6))
32. screen.blit(msg, text\_rect)
33. **if** self.level == 2:
34. **font3 = pygame.font.Font(None, 48)**
35. msg = font3.render('HEALTH INCREASED', 1, BLUE)
36. text\_rect = msg.get\_rect(center=((screen\_width-200)/2, screen\_height/6))
37. screen.blit(msg, text\_rect)
38. **if** self.level == 3:
39. **font3 = pygame.font.Font(None, 48)**
40. msg = font3.render('MULTISHOT UNLOCKED', 1, BLUE)
41. text\_rect = msg.get\_rect(center=((screen\_width-200)/2, screen\_height/6))
42. screen.blit(msg, text\_rect)
43. **if** self.level == 4:
44. **font3 = pygame.font.Font(None, 48)**
45. msg = font3.render('BOSS COMPLETE', 1, BLUE)
46. text\_rect = msg.get\_rect(center=((screen\_width-200)/2, screen\_height/6))
47. screen.blit(msg, text\_rect)
48. pygame.display.flip()
49. ***#end procedure***

Below are images of the game over screen created by the display procedure:

Graphical user interface

Description automatically generated

Graphical user interface

Description automatically generated with low confidenceA picture containing text, hand, scoreboard

Description automatically generatedFurther to this, here are images showing the text that appears when a collision occurs with a chest object:

### Player Class:

#### Constructor Function

The player constructor function has new variables which are used to determine what the player can do depending on which levels the player has completed. These new variables are seen in lines 6 to 9 with can shoot and multi shot set to false, preventing the player from using either of these abilities until unlocked. Lines 11 to 15 are all timers which are later used to regenerate the players health, reset the bullet timer, limit how often the player can inflict damage with his sword and how quickly the player gains bullets.

1. **def** \_\_init\_\_(self,color , width, height, x, y, health, score, money, gamekeys):
2. *#call sprite constructor*
3. super().\_\_init\_\_()
4. [previous code...]
5. ***#Set player attribute that are unlocked after opening chests***
6. self.canshoot = False
7. self.multishot = False
8. self.swordradius = 50
9. self.bulletcount = 3
10. ***#Set timers to be used for attacks***
11. self.previoushealthtime = pygame.time.get\_ticks()
12. self.previousbullettime = pygame.time.get\_ticks()
13. self.previousdamagetime = pygame.time.get\_ticks()
14. self.previousattacktime = pygame.time.get\_ticks()
15. **self.previousbulletaddtime = pygame.time.get\_ticks()**
16. *#end procedure*

#### Update Function

The update function has had some addition to the keys section, in that when the e key is pressed, no function is executed unless can shoot is true. If true then the player can only shoot 1 bullet a second, but if multi shot is true then the player can shoot three bullets in bursts every second. This multi shot ability is shown in lines 20 to 29 where an instance of a player bullet is created every 100 milliseconds and the bullet count of the player is incremented down by 1. When 3 bullets have been created in quick succession, then the bullet count of the player will equal 0, thus no further bullets can be created until the bullet count is greater than zero. Lines 32 to 41 are used to increase the bullet count of the player by one each second when the variable multi shot is false or increment the bullet count of the player by 3 if true. This therefore allows the player to shoot 3 bullets before having to wait a second until they can shoot again.

1. *#Update method for player to control movement and damage taken*
2. **def** update(self):
3. [previous code...]
4. **if** keys[pygame.K\_e]:
5. ***#When the E key is pressed and bullet shooting is unlocked then create a fireball that travel in the direction set by the player from the previous key pressed***
6. **if** self.canshoot == True:
7. *#Check if multishot is unlocked*
8. **if** self.multishot == False:
9. **if** self.bulletcount > 0:
10. ***#Create a bullet once per second***
11. self.currentbullettime = pygame.time.get\_ticks()
12. **if** self.currentbullettime - self.previousbullettime > 1000:
13. bullet = Bullet(RED, self.directionx, self.directiony)
14. game.bullet\_group.add(bullet)
15. **game.all\_sprites\_group.add(bullet)**
16. *#remove 1 from the bullet count*
17. self.bulletcount -= 1
18. self.previousbullettime = self.currentbullettime
19. **else**:
20. ***#If multishot is active then produce a bullet every 100 milliseconds only if bulletcount is greater than zero***
21. **if** self.bulletcount > 0:
22. self.currentbullettime = pygame.time.get\_ticks()
23. **if** self.currentbullettime - self.previousbullettime > 100:
24. *#remove 1 from bullet count so bullets can only be produced in maximum of 3 per second*
25. **self.bulletcount -=1**
26. bullet = Bullet(RED, self.directionx, self.directiony)
27. game.bullet\_group.add(bullet)
28. game.all\_sprites\_group.add(bullet)
29. self.previousbullettime = self.currentbullettime
30. **[previous code...]**
31. *#Check if the timer is greater than one to add bullets to the bulletcount*
32. self.currentbulletaddtime = pygame.time.get\_ticks()
33. **if** self.currentbulletaddtime - self.previousbulletaddtime > 1000:
34. **if** self.multishot == False:
35. **self.bulletcount += 1**
36. **else**:
37. self.bulletcount +=3
38. self.previousbulletaddtime = self.currentbulletaddtime
39. *#Bulletcount is limited to 3*
40. **if self.bulletcount > 3:**
41. self.bulletcount = 3
42. [previous code...]
43. *#end procedure*

Below are images showing how the player can shoot 3 bullets at a time, demonstrating how the multishot ability works for the player:

A picture containing qr code

Description automatically generatedA picture containing qr code

Description automatically generated

#### Move Procedure

The move procedure of the player has been adjusted to prevent the player from being able to move through walls. Lines 6 to 8 first check if the player is colliding with a wall. If the player has a speed in the x direction greater than zero then the player’s right side will be equal the wall’s left side. Thus, when the player tries to move through a wall, the player is moved back to the edge of the wall. Lines 11 to 12 repeat this but instead for the other x direction. Line 16 to 17 check if the player is colliding with a wall again but in line 19 if the player has a positive y direction, then the bottom of the player now is set to the top of the wall that the player is colliding with, and then is repeated but for the negative y direction. This now prevents the player from moving through any wall sprites.

1. *#Method that takes input speeds and moves players position according to speed.*
2. **def** move(self, speedx, speedy):
3. *#move left or right*
4. self.rect.x += self.speed\_x
5. ***#Check if the player collides with the wall***
6. wallcollision = pygame.sprite.spritecollide(self,game.wall\_group, False)
7. **for** wall **in** wallcollision:
8. **if** self.speed\_x > 0:
9. *#stop the player moving into the wall*
10. **self.rect.right = wall.rect.left**
11. **else**:
12. self.rect.left = wall.rect.right
13. *#move the player up and down the screen*
14. self.rect.y += self.speed\_y
15. ***#check for collision***
16. wallcollision = pygame.sprite.spritecollide(self, game.wall\_group, False)
17. **for** wall **in** wallcollision:
18. *#if there is a collision while moving up then set the speed to 0*
19. **if** self.speed\_y > 0:
20. ***#stop the player moving into the wall***
21. self.rect.bottom = wall.rect.top
22. **else**:
23. self.rect.top = wall.rect.bottom
24. *#end procedure*

A picture containing graphical user interface

Description automatically generatedA picture containing graphical user interface

Description automatically generatedThe following images demonstrate how when the player is colliding and moving towards a wall, the player can only move along the wall and not through it:

### Player Bullet Class:

#### Update Function

The update function for the player bullet has been updated to now handle collisions with walls. This is shown in line 5 where if the bullet collides with a wall, then the sprite is then removed from all sprite groups. From line 9 onwards, the code is the same in that any bullets outside of the window are also removed from all sprite groups.

1. *#Update method that moves the player by the speed which is defined by the direction that the player last moved in.*
2. **def** update(self):
3. self.rect.y += self.speedy
4. *#Check if it collides with a wall and kills the sprite*
5. **if pygame.sprite.groupcollide(game.bullet\_group, game.wall\_group, True, False) == True:**
6. self.kill()
7. self.rect.x += self.speedx
8. *#Check if it travel past the boundaries and kills the sprite*
9. **if** self.rect.x > 1000:
10. **self.kill()**
11. **if** self.rect.x < 0:
12. self.kill()
13. if self.rect.y > 1000:
14. self.kill()
15. **if self.rect.y < 0:**
16. self.kill()

A picture containing graphical user interface

Description automatically generatedA picture containing icon

Description automatically generatedBelow are two images that demonstrate bullets being deleted on collision with a wall or from exceeding the limits of the screen:

### Enemy Bullet Class:

#### Update Function

The update function for enemy bullets has been changed to now handle collisions with walls, players and other bullets. Line 5 is used to check if the bullet has collided with a wall and if true will then remove the bullet from all sprite groups. Lines 8 to 10 check if the bullet has collided with a player and if true will inflict 10 damage on the player’s health. Lines 12 to 20 check if the bullets have collided with the player’s bullets or sword and if true will remove the both bullets from all sprite groups and increment the score by 20.

1. *#Update method that moves the bullet according to the speed set by the bow enemy*
2. **def** update(self):
3. [previous code...]
4. *#check for wall collision and kill the bullet*
5. **if pygame.sprite.groupcollide(game.enemybullet\_group, game.wall\_group, True, False) == True:**
6. self.kill()
7. *#Check if it collides with the player and inflict 10 damage on the player*
8. player\_hit\_group = pygame.sprite.groupcollide(game.player\_group, game.enemybullet\_group, False, True)
9. **for** game.player **in** player\_hit\_group:
10. **game.player.getdamage(10)**
11. *#Check for collision with player bullet and kill the bullet and then kill itself and add points to game score*
12. enemybullet\_hit\_group = pygame.sprite.groupcollide(game.enemybullet\_group, game.bullet\_group, False, True)
13. **for** self **in** enemybullet\_hit\_group:
14. self.kill()
15. **game.score +=20**
16. *#Check for collision with player's sword and kill itself when colliding and add points to game score*
17. enemysword\_hit\_group = pygame.sprite.groupcollide(game.enemybullet\_group, game.sword\_group, False, False)
18. **for** self **in** enemysword\_hit\_group:
19. self.kill()
20. **game.score +=20**

A picture containing text, scoreboard

Description automatically generatedA picture containing text, hand, scoreboard

Description automatically generatedBelow is an image demonstrating the collision between the player and the bullet and how the health of the player is reduced:

### Door Class:

#### Update Function

The update function for the door has been slightly adapted to now check if the boss has been defeated and the chest has been unlocked before the door is removed for level 5.

1. *#Update function checks what the level is and checks if the level is over before removing the door object*
2. **def** update(self):
3. **if** (game.level + 1) % 5 != 0:
4. *#Check if the level is a multiple of 5 and if not then check all keys have been collected and the chest unlocked*
5. **if game.player.gamekeys >= ((2\*(game.level+1)) + 1):**
6. **if** game.chestunlocked[game.level] == True:
7. self.kill()
8. **else**:
9. *#If the level is a multiple of 5 and the boss has been defeated then open the doors.*
10. **if len(game.boss\_group) == 0:**
11. **if** game.chestunlocked[game.level] == True:
12. self.kill()

### Spike Class:

#### Update Function

The update function for the spikes now include collision checking with the player. Line 5 to 7 check if the spikes are active and if the player is colliding with the spikes and if both are true, then will inflict 5 damage on the player. To prevent the player from taking too much damage, Line 9 and 10 use a timer of 4 seconds to prevent the spikes from registering a collision with the player more than once.

1. *#Update function*
2. **def** update(self):
3. [previous code...]
4. *#If the spikes are active then check for player collision*
5. **if self.color == LIGHTBLUE:**
6. player\_hit\_group = pygame.sprite.spritecollide(self, game.player\_group, False, False)
7. **for** game.player **in** player\_hit\_group:
8. *#If player collision is true then inflict 5 damage every 4 seconds (this limits the spike to only attack the player once when activated)*
9. self.currentdamagetime = pygame.time.get\_ticks()
10. **if self.currentdamagetime - self.previousdamagetime > 4000:**
11. game.player.getdamage(5)
12. *#reset attack timer*
13. self.previousdamagetime = self.currentdamagetime

A picture containing graphical user interface

Description automatically generatedA picture containing graphical user interface

Description automatically generatedThe following images show the collision between the spike class and the player class and the inflicted damage:

### Chest Class:

#### Update Function

The update function of the chest class has been added to check for player collision and the current level. Line 4 and 5 are used to check when the player collides with the chest object. When true, the current level is checked and depending on the level, the chest will alter one of the player’s variables, such as the can shoot or multi shot variables, that give the player more abilities. After colliding with the player, lines 15 and 16 are used to remove the sprite from all sprite groups.

1. *#Update function used to check for player collision*
2. **def** update(self):
3. *#Check what the level is so when the chest collides with the player then unlock an ability for each level.*
4. chest\_hit\_group = pygame.sprite.groupcollide(game.chest\_group, game.player\_group, False, False)
5. **for game.chest in chest\_hit\_group:**
6. **if** self.level == 0:
7. game.player.canshoot = True
8. **if** self.level == 1:
9. game.player.swordradius = 70
10. **if self.level == 2:**
11. game.player.maximum\_health = 200
12. **if** self.level == 3:
13. game.player.multishot = True
14. *#When the player collides then remove the chest and set the chest unlocked to true for that level.*
15. **game.chestunlocked[game.level] = True**
16. self.kill()

Qr code

Description automatically generatedA picture containing graphical user interface

Description automatically generatedBelow are images of the different abilities triggered when colliding with the chest class:

A screenshot of a computer

Description automatically generated with low confidenceA screenshot of a computer

Description automatically generated with low confidence

### Sword Enemy Class:

#### Update Function

The update function for the sword enemy class now includes collisions that check when the player is in contact with a player bullet or sword. This is shown in lines 11 to 13 which check for a collision with a player bullet and subtract 20 health from the enemy if true. Lines 15 to 20 check for the collision with a sword object and if true will also subtract 20 health from the enemy. In lines 22 to 28, a collision is checked between the player and the enemy and when true, 20 damage is inflicted on the player’s health every 2 seconds.

1. *#update function handles collisions and changes behaviour depending on the range of the enemy*
2. **def** update(self):
3. *#Check if the player is 300 pixels away from the enemy*
4. **if** self.is\_close() == True:
5. ***#if this is true then use the movetoplayer method to make the enemy move towards the player.***
6. self.movetoplayer(game.player)
7. *#If False then use the move method instead which makes the enmey bounce off walls.*
8. **elif** self.is\_close() == False:
9. self.MOVE()
10. ***#if the enemy collides with a bullet then subtract 20 health from the enemy.***
11. enemybullet\_hit\_group = pygame.sprite.groupcollide(game.enemy\_group, game.bullet\_group, False, True)
12. **for** self **in** enemybullet\_hit\_group:
13. self.health -= 20
14. *#check for a collision with the sword and only allow the sword to inflict 20 damage per second on the enemy.*
15. **enemysword\_hit\_group = pygame.sprite.groupcollide(game.enemy\_group, game.sword\_group, False, False)**
16. **for** self **in** enemysword\_hit\_group:
17. self.currentdamagetime = pygame.time.get\_ticks()
18. **if** self.currentdamagetime - self.previousdamagetime > 1000:
19. self.health -= 20
20. **self.previousdamagetime = self.currentdamagetime**
21. *#Check for collision with player.*
22. player\_hit\_group = pygame.sprite.groupcollide(game.player\_group, game.enemy\_group, False, False)
23. **for** game.player **in** player\_hit\_group:
24. *#if the player collides with the enemy then subtract 10 health from the player every 2 seconds*
25. **self.currentattacktime = pygame.time.get\_ticks()**
26. **if** self.currentattacktime - self.previousattacktime > 2000:
27. game.player.getdamage(10)
28. self.previousattacktime = self.currentattacktime
29. *#Check if the health is greater than zero. Else it will create an instance of a key and increase the game score by 100 before removing itself.*
30. **if self.health < 1:**
31. game.score += 100
32. gamekey = Key(PINK, self.rect.x + 2, self.rect.y + 9)
33. game.all\_sprites\_group.add(gamekey)
34. game.key\_group.add(gamekey)
35. **self.kill()**

Logo, qr code

Description automatically generatedBelow are two images demonstrating the collision between the sword enemy and player:

Logo

Description automatically generated

#### Move Procedure

This new procedure for the sword enemy class was created to allow the enemies to move whenever the player is not within range. Thus, line 4 checks whether the direction of the enemy object, which is random is divisible by 2. If true, then the enemy will move horizontally across the screen. When the enemy collides with a wall, shown in line 7, then the enemy will be reflected off the wall and move in the opposite direction. If the direction of the enemy is not divisible by two, then it will move vertically instead and also reflect off the wall, as shown in lines 19 to 31.

1. *#Method used to control how the enemy moves when the player is not in range*
2. **def** MOVE(self):
3. *#Using a random number between 1 and 10 and check if it divisible by 2. this provides a 50 50 chance of the enemy bouncing up and down or side to side.*
4. **if** self.direction % 2 == 0:
5. **self.rect.x += self.move**
6. *#If divisible by 2 then move to the right by 3 pixels.*
7. wallcollision = pygame.sprite.groupcollide(game.enemy\_group,game.wall\_group, False,False)
8. **for** wall **in** wallcollision:
9. *#if it collides to the left of a wall then bounce of the wall and move in the opposite direction*
10. **if self.move > 0:**
11. self.rect.right = wall.rect.left
12. self.move = self.move \* -1
13. **else**:
14. *#if it collides to the left of a wall then bounce of the wall and move in the opposite direction*
15. **self.rect.left = wall.rect.right**
16. self.move = self.move \* -1
17. *#move the player up and down the screen*
18. **else**:
19. self.rect.y += self.move
20. ***#check for collision***
21. wallcollision = pygame.sprite.groupcollide(game.enemy\_group, game.wall\_group, False, False)
22. **for** wall **in** wallcollision:
23. *#if there is a collision while moving up then set the speed to 0*
24. **if** self.move > 0:
25. ***#if it collides with the top of the wall then reflect the speed and bounce of the wall***
26. self.rect.bottom = wall.rect.top
27. self.move = self.move \* -1
28. **else**:
29. *#if it collides with the bottom of the wall then reflect the speed and bounce of the wall*
30. **self.rect.top = wall.rect.bottom**
31. self.move = self.move \* -1

A picture containing graphical user interface

Description automatically generatedA picture containing graphical user interface

Description automatically generatedThe following two images demonstrate how the enemy reflects off the wall when not near the player:

#### Move to Player Procedure

The move to player procedure has now been updated to prevent the enemy from moving through walls as it attempts to pursue the player. In lines 15 and 16, if a collision occurs between the wall and the enemy and it is moving in a horizontal direction (line 18), then the right of the enemy is set to the left of the wall and vice versa for a negative x direction. This is then repeated from lines 26 onwards but instead in the vertical direction.

1. *#Method to change how the enemy moves when the player is within range of the enemy*
2. **def** movetoplayer(self, Player):
3. *#Compare the player's position to the enemy's position and change the speed of the enemy to move it closer to the player.*
4. **if** Player.rect.x - 10 > self.rect.x:
5. **self.speed\_x = 2**
6. **if** Player.rect.x - 10 < self.rect.x:
7. self.speed\_x = -2
8. **if** Player.rect.y - 10 > self.rect.y:
9. self.speed\_y = 2
10. **if Player.rect.y - 10< self.rect.y:**
11. self.speed\_y = -2
12. *# Move along x axis*
13. self.rect.x += self.speed\_x
14. *# Did enemy hit a wall*
15. **block\_hit\_list = pygame.sprite.spritecollide(self, game.wall\_group, False) *# false so it doesn't remove the wall, true would***
16. **for** wall **in** block\_hit\_list:
17. *# If moving right, place enemy to the left side of wall*
18. **if** self.speed\_x > 0:
19. self.rect.right = wall.rect.left
20. **else:**
21. *# if moving left, do the opposite.*
22. self.rect.left = wall.rect.right
23. *# Move along y axis*
24. self.rect.y += self.speed\_y
25. ***# Did enemy hit a wall***
26. block\_hit\_list = pygame.sprite.spritecollide(self, game.wall\_group, False)
27. **for** wall **in** block\_hit\_list:
28. *# Do same as above but on the y axis*
29. **if** self.speed\_y > 0:
30. **self.rect.bottom = wall.rect.top**
31. **else**:
32. self.rect.top = wall.rect.bottom

Logo, qr code

Description automatically generatedThe following image demonstrates the enemy not being able to pass through the wall:

### Bow Enemy Class:

#### Update Function

The update function is used to check how close the player is to the bow enemy and also check for collisions. Lines 3 to 8 check if the player is close and calculate the x and y differences. Lines 9 to 14 check if the magnitude is greater than 60 and if true will adjust the x and y speeds so that the enemy bullets do not move to quickly depending on the distance of the player from the bow enemy. Line 16 to 22 are used to create an instance of the enemy bullet class every second with the given speeds. Then in lines 24 to 33 it checks for collisions with the player bullet and checks if the health of the bow enemy is below zero to then create a key object and increment the score.

1. *#Update function used to check if the player is in range and calculate the speeds for the bullet fired*
2. **def** update(self):
3. **if** self.is\_close() == True:
4. *#If player in range then calculate y distance and x distance as well as distance*
5. **xdiff = (game.player.rect.x-5) - (self.rect.x+20)**
6. ydiff = (game.player.rect.y-5) - (self.rect.y+20)
7. magnitude = math.hypot(xdiff,ydiff)
8. *#If the distance is greater than 60 pixels then change the speed off the bullet*
9. **if** magnitude > 60:
10. **xspeed = xdiff \* 0.01**
11. yspeed = ydiff \* 0.01
12. **else**:
13. xspeed = xdiff \* 0.05
14. yspeed = ydiff \* 0.05
15. ***#Every 1 second an enemy bullet object is created travelling in the direction using the xspeed and yspeed.***
16. self.currentattacktime = pygame.time.get\_ticks()
17. **if** self.currentattacktime - self.previousattacktime > 1000:
18. game.ebullet = EnemyBullet(RED, xspeed, yspeed, self.rect.x+20, self.rect.y+20)
19. game.all\_sprites\_group.add(game.ebullet)
20. **game.enemybullet\_group.add(game.ebullet)**
21. *#reset the timer*
22. self.previousattacktime = self.currentattacktime
23. *#if the enemy collides with a bullet then subtract 20 health from the enemy.*
24. enemybullet\_hit\_group = pygame.sprite.groupcollide(game.enemy\_group, game.bullet\_group, False, True)
25. **for self in enemybullet\_hit\_group:**
26. self.health -= 20
27. *#Check if the health is greater than zero. Else it will create an instance of a key and increase the game score by 100 before removing itself.*
28. **if** self.health < 1:
29. game.score += 100
30. **gamekey = Key(PINK, self.rect.x + 2, self.rect.y + 9)**
31. game.all\_sprites\_group.add(gamekey)
32. game.key\_group.add(gamekey)
33. self.kill()

A picture containing qr code

Description automatically generatedA screenshot of a computer

Description automatically generated with low confidenceThe following image demonstrates the key being produced when the bow enemy is killed:

### Boss Enemy Class:

#### Constructor Function

1. *#Boss Enemy Class*
2. **class** BossEnemy(pygame.sprite.Sprite):
3. **def** \_\_init\_\_(self, direction, width, height, x, y, health):
4. *#call sprite constructor*
5. **super().\_\_init\_\_()**
6. *#create a sprite*
7. self.image = pygame.Surface([width,height])
8. self.image.fill(YELLOW)
9. *#set the position of the sprite*
10. **self.rect = self.image.get\_rect()**
11. self.rect.x = x
12. self.rect.y = y
13. *#Define attributes for health to be used in boss health bar*
14. self.current\_health = 999
15. **self.maximum\_health = health**
16. self.health\_bar\_length = 200
17. self.target\_health = 1000
18. self.health\_change\_speed = 2
19. self.health\_bar\_color = GREEN
20. **self.health\_ratio = self.maximum\_health/ self.health\_bar\_length**
21. self.direction = direction
22. *#Set timers to be used for attacks.*
23. self.previoushealthtime = pygame.time.get\_ticks()
24. self.previousdamagetime = pygame.time.get\_ticks()
25. **self.previousattacktime = pygame.time.get\_ticks()**

A picture containing graphical user interface

Description automatically generatedThe following image shows the appearance of the boss compared to the player:

#### Get Health Function

The Get health procedure is used to increase the boss’s health when triggered by an event. Line 2 shows that the parameter amount is what is used to increase the boss’s health by. As to stop the boss’s health from exceeding its maximum, we use the if statement in line 5 to set the target health, the health which will be reached once adding the amount to it to equal the maximum health of the boss if it is greater than it.

1. *#method used to increase the health of the boss by an amount*
2. **def** gethealth(self, amount):
3. **if** self.target\_health < self.maximum\_health:
4. self.target\_health += amount
5. **if self.target\_health >= self.maximum\_health:**
6. self.target\_health = self.maximum\_health
7. *#endprocedure*

#### Get Damage Procedure

The get damage procedure is used to remove health from the boss. Similar to the get health procedure it checks what the target health is before setting the new health of the boss so that if the target health of the boss is below zero then its will be set to equal zero. This is used to stop the boss from having a negative health that could cause error to occur.

1. *#Method used to remove health from the boss*
2. **def** getdamage(self,amount):
3. **if** self.target\_health > 0:
4. self.target\_health -= amount
5. **if self.target\_health <=0:**
6. self.target\_health = 0

#### Health Bar Procedure

The health bar procedure is used to create and change the rectangles used that form the health bar. The health bar is formed of 3 rectangles, the main health bar, the transition bar and the outline bar. The outline bar outlines the other bars while the transition bar is used to display a change in the boss’s health. Line 3 and 4 define the transition bar’s width as zero and its colour as red initially. Line 7 to 10 check whether the target health is larger than the initial health and then change the transition bar’s colour and width so that it matches the difference in size between the initial health and target health. This is then divided by the health ratio which is used to adjust the transition bars width to the correct number of pixels on the screen as the health bar is not the same number of pixels in width as the number of health points the boss has. This process is repeated from lines 12 to 15 but in reverse with the transition bar now being yellow as health is deducted. Lines 18 to 23 have several if statements to change the colour of the main health bar depending on how much health the boss has. Finally, lines 25 to 32 define the rectangles and draw them on the screen, with the health bar always above the boss object.

1. *#Method used to create helath bar*
2. **def** advanced\_health(self):
3. transition\_width = 0
4. transition\_color = RED
5. ***#When health is added set the transition bar to green and increase health bar at set speed***
6. **if** self.current\_health < self.target\_health:
7. self.current\_health += self.health\_change\_speed
8. transition\_width = int((self.target\_health - self.current\_health)/ self.health\_ratio)
9. transition\_color = GREEN
10. ***#When health is removed set the transition bar to yellow and decrease the health bar at a set speed***
11. **if** self.current\_health > self.target\_health:
12. self.current\_health -= self.health\_change\_speed
13. transition\_width = int((self.target\_health - self.current\_health)/ self.health\_ratio)
14. transition\_color = YELLOW
15. ***#Change the health bar colour depending on the amount of health***
16. **if** self.current\_health >= 700:
17. self.health\_bar\_color = GREEN
18. **if** self.current\_health >= 500 **and** self.current\_health < 700:
19. self.health\_bar\_color = ORANGE
20. **if self.current\_health < 300 and self.current\_health >=0:**
21. self.health\_bar\_color = RED
22. *#set the health bar size equal to the current health divided by max health*
23. health\_bar\_width = int(self.current\_health/ self.health\_ratio)
24. health\_bar = pygame.Rect(self.rect.x-20,self.rect.y-40, health\_bar\_width, 25)
25. ***#transition bar is set to the right of the health bar***
26. transition\_bar = pygame.Rect(health\_bar.right, self.rect.y-40, transition\_width, 25)
27. **if** self.current\_health > 0:
28. *#Draw all 3 bars over each other*
29. pygame.draw.rect(screen, self.health\_bar\_color, health\_bar)
30. **pygame.draw.rect(screen,transition\_color, transition\_bar)**
31. pygame.draw.rect(screen, WHITE, (self.rect.x-20, self.rect.y-40, self.health\_bar\_length, 25), 4)

#### Update Function

The update function is used to change the behaviour of the boss depending on its health and also check for collisions. Lines 4 to 9 are used to define the health ratio and increase the boss’s health by 10 every 10 seconds. Lines 11 to 13 check if the boss’s health is below zero and if true then increment the score and remove the boss object from all sprite groups. The boss has 3 stages, the first just follows the player around as shown in lines 15 to 18. The second stage follows the player and shoots bullets towards the player in lines 19 to 39.

1. *#Update function to change behaviour of Boss depending on its health.*
2. **def** update(self):
3. *#Every 10 seconds the boss regains health*
4. self.health\_ratio = self.maximum\_health/ self.health\_bar\_length
5. **self.currenthealthtime = pygame.time.get\_ticks()**
6. **if** self.currenthealthtime - self.previoushealthtime > 10000:
7. self.gethealth(10)
8. *#reset the timer*
9. self.previoushealthtime = self.currenthealthtime
10. ***#Check if the health is zero then add to the game score and kill the boss***
11. **if** self.current\_health < 1:
12. game.score += 1000
13. self.kill()
14. *#When the health of the boss is greater than 70% then just move towards the player*
15. **if self.current\_health >= 700:**
16. *#move towards the player no matter the range*
17. self.movetoplayer(game.player)
18. *#When the health is lower than 70% but above 30% then move around and shoot at the player*
19. **if** self.current\_health < 700 **and** self.current\_health >= 400:
20. ***#move towards the player and shoot projectiles***
21. *#Calculate y distance and x distance as well as distance*
22. self.movetoplayer(game.player)
23. xdiff = (game.player.rect.x-5) - (self.rect.x+80)
24. ydiff = (game.player.rect.y-5) - (self.rect.y+80)
25. **magnitude = math.hypot(xdiff,ydiff)**
26. *#If the distance is greater than 60 pixels then change the speed off the bullet*
27. **if** magnitude > 60:
28. xspeed = xdiff \* 0.01
29. yspeed = ydiff \* 0.01
30. **else:**
31. xspeed = xdiff \* 0.05
32. yspeed = ydiff \* 0.05
33. *#Every 1 second an enemy bullet object is created travelling in the direction using the xspeed and yspeed.*
34. self.currentattacktime = pygame.time.get\_ticks()
35. **if self.currentattacktime - self.previousattacktime > 1500:**
36. game.ebullet = EnemyBullet(RED, xspeed, yspeed, self.rect.x+80, self.rect.y+80)
37. game.all\_sprites\_group.add(game.ebullet)
38. game.enemybullet\_group.add(game.ebullet)
39. self.previousattacktime = self.currentattacktime

The following is still part of the update function and is used for the final stage of the boss. Lines 1 to 4 are used to move the boss towards the player and lines 5 to 32 are used to create 8 instances of the enemy bullet class that all travel in different directions. Line 7 then repeats this instantiation every second. Further to this, lines 33 to 36 are used to create 5 random coordinates. Lines 37 to 42 then use these coordinates to create instances of sword enemies using the 5 coordinates.

1. **if** self.current\_health < 300:
2. *#If the boss is less than 30% then shoot 8 bullets travelling up, down, left, right and all diagonal directions*
3. *#shoot projectiles in all directions and spawn in enemies*
4. self.movetoplayer(game.player)
5. **self.currentattacktime = pygame.time.get\_ticks()**
6. *#Repeat this every second*
7. **if** self.currentattacktime - self.previousattacktime > 1000:
8. game.ebullet = EnemyBullet(RED, 3, 0, self.rect.x+80, self.rect.y+80)
9. game.all\_sprites\_group.add(game.ebullet)
10. **game.enemybullet\_group.add(game.ebullet)**
11. game.ebullet = EnemyBullet(RED, 3, 3, self.rect.x+80, self.rect.y+80)
12. game.all\_sprites\_group.add(game.ebullet)
13. game.enemybullet\_group.add(game.ebullet)
14. game.ebullet = EnemyBullet(RED, 3, -3, self.rect.x+80, self.rect.y+80)
15. **game.all\_sprites\_group.add(game.ebullet)**
16. game.enemybullet\_group.add(game.ebullet)
17. game.ebullet = EnemyBullet(RED, -3, 0, self.rect.x+80, self.rect.y+80)
18. game.all\_sprites\_group.add(game.ebullet)
19. game.enemybullet\_group.add(game.ebullet)
20. **game.ebullet = EnemyBullet(RED, -3, 3, self.rect.x+80, self.rect.y+80)**
21. game.all\_sprites\_group.add(game.ebullet)
22. game.enemybullet\_group.add(game.ebullet)
23. game.ebullet = EnemyBullet(RED, -3, -3, self.rect.x+80, self.rect.y+80)
24. game.all\_sprites\_group.add(game.ebullet)
25. **game.enemybullet\_group.add(game.ebullet)**
26. game.ebullet = EnemyBullet(RED, 0, 3, self.rect.x+80, self.rect.y+80)
27. game.all\_sprites\_group.add(game.ebullet)
28. game.enemybullet\_group.add(game.ebullet)
29. game.ebullet = EnemyBullet(RED, 0, -3, self.rect.x+80, self.rect.y+80)
30. **game.all\_sprites\_group.add(game.ebullet)**
31. game.enemybullet\_group.add(game.ebullet)
32. self.previousattacktime = self.currentattacktime
33. **while** game.bosscount != 5:
34. *#Create 5 melee enemies that randomly spawn in the level*
35. **xpos = random.randint(1,23)**
36. ypos = random.randint(1,23)
37. *#Check if the random number has the same coordinates as a wall or spike and if not then create an enemy*
38. **if** game.levels[game.level][xpos][ypos] !=1 **and** game.levels[game.level][xpos][ypos] != 2 **and** game.levels[game.level][xpos][ypos] != 7:
39. game.menemy = MeleeEnemy(random.randint(0,10),40,40, xpos\*40, ypos\*40, 40)
40. **game.all\_sprites\_group.add(game.menemy)**
41. game.enemy\_group.add(game.menemy)
42. game.bosscount = game.bosscount +1

The update function also checks for collisions. Lines 2 to 4 check for collisions with a player bullet object and subtract 5 health from the boss. Lines 6 to 11 are used to check for collision with the sword object and if true subtract 20 health from the boss. Lines 13 to 19 are then used to check if the boss collides with the player. If true then subtract 30 health from the player every 2 seconds.

1. *#Check if the boss collides with a player bullet then subtract 5 health from the boss.*
2. enemybullet\_hit\_group = pygame.sprite.groupcollide(game.boss\_group, game.bullet\_group, False, True)
3. **for** self **in** enemybullet\_hit\_group:
4. self.getdamage(5)
5. ***#Check for a collision with the sword and only allow the sword to inflict 20 damage per second on the boss.***
6. enemysword\_hit\_group = pygame.sprite.groupcollide(game.boss\_group, game.sword\_group, False, False)
7. **for** self **in** enemysword\_hit\_group:
8. self.currentdamagetime = pygame.time.get\_ticks()
9. **if** self.currentdamagetime - self.previousdamagetime > 1000:
10. **self.getdamage(20)**
11. self.previousdamagetime = self.currentdamagetime
12. *#Check for collision with player.*
13. player\_hit\_group = pygame.sprite.groupcollide(game.player\_group, game.boss\_group, False, False)
14. **for** game.player **in** player\_hit\_group:
15. **self.currentattacktime = pygame.time.get\_ticks()**
16. *#if the player collides with the boss then subtract 30 health from the player every 2 seconds*
17. **if** self.currentattacktime - self.previousattacktime > 2000:
18. game.player.getdamage(30)
19. self.previousattacktime = self.currentattacktime

Logo

Description automatically generatedThe image below demonstrates the boss’ behaviour after it has reduced its health to 70%:

The following image demonstrates the boss’ health when it reaches 30%:

Graphical user interface, application

Description automatically generated

#### Move to Player Procedure

The move to player procedure is identical to that in the sword enemy class. In lines 15 and 16, if a collision occurs between the wall and the boss and it is moving in a horizontal direction (line 18), then the right of the boss is set to the left of the wall and vice versa for a negative x direction. This is then repeated from lines 26 onwards but instead in the vertical direction.

1. *#Method used to make the boss move towards the player*
2. **def** movetoplayer(self, Player):
3. *#compare the player's position to the boss' position and change the speed of the boss to move it towards the player*
4. **if** Player.rect.x > self.rect.x:
5. **self.speed\_x = 1**
6. **if** Player.rect.x < self.rect.x:
7. self.speed\_x = -1
8. **if** Player.rect.y > self.rect.y:
9. self.speed\_y = 1
10. **if Player.rect.y < self.rect.y:**
11. self.speed\_y = -1
12. *# Move along x axis*
13. self.rect.x += self.speed\_x
14. *# Did enemy hit a wall*
15. **block\_hit\_list = pygame.sprite.spritecollide(self, game.wall\_group, False) *# false so it doesn't remove the wall, true would***
16. **for** wall **in** block\_hit\_list:
17. *# If moving right, place enemy to the left side of wall*
18. **if** self.speed\_x > 0:
19. self.rect.right = wall.rect.left
20. **else:**
21. *# if moving left, do the opposite.*
22. self.rect.left = wall.rect.right
23. *# Move along y axis*
24. self.rect.y += self.speed\_y
25. ***# Did enemy hit a wall***
26. block\_hit\_list = pygame.sprite.spritecollide(self, game.wall\_group, False)
27. **for** wall **in** block\_hit\_list:
28. *# Do same as above but on the y axis*
29. **if** self.speed\_y > 0:
30. **self.rect.bottom = wall.rect.top**
31. **else**:
32. self.rect.top = wall.rect.bottom

### Errors

#### Graphical user interface Description automatically generatedDirection of Enemy Bullets

The above image shows that the new code used to control the direction of the bullets for the bow and boss enemies still does not work, and thus needs to be corrected again so that the bullets travel in the correct direction. To fix this I have replaced the code again using the functions from the library but this time I have not converted into degrees, which has solved the problem in firing the bullets towards the player. This can be seen in the following code:

1. *#Update function used to check if the player is in range and calculate the speeds for the bullet fired*
2. **def** update(self):
3. [previous code...]
4. *#If player in range then calculate y distance and x distance*
5. **xdiff = (game.player.rect.x) - (self.rect.x)**
6. ydiff = (game.player.rect.y) - (self.rect.y)
7. *#find the angle from the enemy to the player*
8. self.angle = math.atan2(ydiff,xdiff)
9. *#set the x and y speed of the bullet towards the player*
10. **xspeed = 3 \* math.cos(self.angle)**
11. yspeed = 3 \* math.sin(self.angle)
12. [previous code...]

Graphical user interface, application

Description automatically generatedBelow is an image demonstrating the corrected code, causing the bow enemy to more accurately shoot at the player:

#### Logo Description automatically generatedBoss Enemy Health Bar

Again, the same issue of the health bar being drawn over again but this time with the boss enemy class. In order to fix this I required to call the method as the last object to be drawn in the game.display method. This is shown in the following code:

1. *#Create a method to display all sprites and text*
2. **def** display(self, screen):
3. [previous code...]
4. self.boss.advanced\_health()
5. **[previous code...]**

However, what happens when the code is run is that this error occurs:

This is due to the fact that the boss enemy object is not created until level 4, so produces an error as the object is yet to be created within the game class and it cannot be called before it has been defined. To fix this I introduced another line of code to prevent this:

1. *#Create a method to display all sprites and text*
2. **def** display(self, screen):
3. [previous code...]
4. **if** (self.level + 1) % 5 == 0:
5. **self.boss.advanced\_health()**
6. [previous code...]

A picture containing shape

Description automatically generatedThus, solving the problem as shown:

#### Chart Description automatically generated with medium confidenceSpike objects are appearing over the boss Enemy Group

The above image shows that the spikes are either drawn on top of or below the boss enemy group even though they should be below. To fix this a new line of code was added to the game.display procedure:

1. *#Create a method to display all sprites and text*
2. **def** display(self, screen):
3. [previous code...]
4. *# --- Drawing code for sprites*
5. **self.all\_sprites\_group.draw(screen)**
6. self.boss\_group.draw(screen)
7. [previous code...]

This ensures that the boss enemy is always drawn over the spikes such as in the image below:

#### A picture containing text, screenshot, clipart, vector graphics Description automatically generatedEnemy glitches inside wall

Logo

Description automatically generatedAll melee enemies start to glitch inside of walls due to the move function if one of them moves between two walls with a 1 block gap between them as shown below:

This is due to the code using groupcollide rather than spritecollide, which makes all the objects glitch if one of the objects has a problem:

1. *#Method used to control how the enemy moves when the player is not in range*
2. **def** MOVE(self):
3. *#Using a random number between 1 and 10 and check if it divisible by 2. this provides a 50 50 chance of the enemy bouncing up and down or side to side.*
4. **if** self.direction % 2 == 0:
5. **self.rect.x += self.move**
6. *#If divisible by 2 then move to the right by 3 pixels.*
7. wallcollision = pygame.sprite.groupcollide(game.enemy\_group,game.wall\_group, False,False)
8. **for** wall **in** wallcollision:
9. *#if it collides to the left of a wall then bounce of the wall and move in the opposite direction*
10. **if self.move > 0:**
11. self.rect.right = wall.rect.left
12. self.move = self.move \* -1
13. **else**:
14. *#if it collides to the left of a wall then bounce of the wall and move in the opposite direction*
15. **self.rect.left = wall.rect.right**
16. self.move = self.move \* -1
17. *#move the player up and down the screen*
18. **else**:
19. self.rect.y += self.move
20. ***#check for collision***
21. wallcollision = pygame.sprite.groupcollide(game.enemy\_group, game.wall\_group, False, False)
22. **for** wall **in** wallcollision:
23. *#if there is a collision while moving up then set the speed to 0*
24. **if** self.move > 0:
25. ***#if it collides with the top of the wall then reflect the speed and bounce of the wall***
26. self.rect.bottom = wall.rect.top
27. self.move = self.move \* -1
28. **else**:
29. *#if it collides with the bottom of the wall then reflect the speed and bounce of the wall*
30. **self.rect.top = wall.rect.bottom**
31. self.move = self.move \* -1

This code was first changed to the following to stop the glitching:

1. **def** MOVE(self):
2. *#Using a random number between 1 and 10 and check if it divisible by 2. this provides a 50 50 chance of the enemy bouncing up and down or side to side.*
3. **if** self.direction % 2 == 0:
4. self.movey = 0
5. **self.rect.x += self.movex**
6. *#If divisible by 2 then move to the right by 3 pixels.*
7. **else**:
8. self.movex = 0
9. self.rect.y += self.movey
10. ***#check for collision***
11. wallcollision = pygame.sprite.spritecollide(self, game.wall\_group, False, False)
12. **for** wall **in** wallcollision:
13. *#if there is a collision while moving up then set the speed to 0*
14. **if** self.movey > 0:
15. ***#if it collides with the top of the wall then reflect the speed and bounce of the wall***
16. self.rect.bottom = wall.rect.top
17. self.movey = self.movey \* -1
18. **elif** self.movey < 0:
19. *#if it collides with the bottom of the wall then reflect the speed and bounce of the wall*
20. **self.rect.top = wall.rect.bottom**
21. self.movey = self.movey \* -1
22. *#if it collides to the left of a wall then bounce of the wall and move in the opposite direction*
23. **if** self.movex > 0:
24. self.rect.right = wall.rect.left
25. **self.movex = self.movex \* -1**
26. **elif** self.movex <0:
27. *#if it collides to the left of a wall then bounce of the wall and move in the opposite direction*
28. self.rect.left = wall.rect.right
29. self.movex = self.movex \* -1

However, this new code had a problem in that when the enemy had used the movetoplayer procedure, the next time the move procedure was used, the enemy would not collide with the wall and go straight through it. In order to solve this problem, I simplified the code again in the move procedure:

1. **def** MOVE(self):
2. *#Using a random number between 1 and 10 and check if it divisible by 2. this provides a 50 50 chance of the enemy bouncing up and down or side to side.*
3. **if** self.direction % 2 == 0:
4. self.movey = 0
5. **self.rect.x += self.movex**
6. *#If divisible by 2 then move to the right by 3 pixels.*
7. **else**:
8. self.movex = 0
9. self.rect.y += self.movey

I also added new code into the update function in order to fix the collisions not being handled in time:

1. **def** update(self):
2. *#Check if the player is 300 pixels away from the enemy*
3. **if** self.is\_close() == True:
4. *#if this is true then use the movetoplayer method to make the enemy move towards the player.*
5. **self.movetoplayer(game.player)**
6. *#If False then use the move method instead which makes the enmey bounce off walls.*
7. **if** self.is\_close() == False:
8. self.MOVE()
9. wallcollision = pygame.sprite.spritecollide(self, game.wall\_group, False)
10. **if wallcollision:**
11. *#if there is a collision then reflect the speeds*
12. self.movex = self.movex \* -1
13. self.movey = self.movey \* -1

This solved the issue and now the enemies behave as normal in the following image:

A picture containing graphical user interface

Description automatically generated

### Conclusion

Iteration 2 was very successful in handling all the errors faced with collisions and any errors found from the previous iteration. Thus, iteration 2 has completed **success criteria 1, 5, 9, 10, 15, 16, 21, 22, 23, 24, 25, 26, 29, 30, 31, 33, 34, 35, 36, 37, 38, 39.** Since finishing creating all the levels and resolving collisions in this iteration, there will be a larger focus on adding aesthetic changes to the game and increased functionality to the game over screen as well as resolving smaller issues. Considering that most of my requirements are complete I will be adding sprite sheets and images to make the game more enticing to play.

## Iteration 3 – Sprite sheets and Aesthetics

Iteration 3 has the focus of implementing aesthetic and usability features. A lot of the changes are small tweaks that will make the game look better or improve the usability of the game. As such **the requirements to be completed in this iteration are 41 and 42.**

### Menu:

#### Button Function

The button function has added functionality to include an extra operation used to call the new instruction procedure. This is used to include more usability features such as an instructions menu that gives the user the important aspects and keys needed to play the game. This assists with **success criteria number 42.**

1. **def** button\_1(message,xpos,ypos,width,height,inactivecolor,activecolor,action1=None):
2. [previous code...]
3. **if** click[0] == 1 **and** action1 !=None:
4. *#Check what the action of the button does.*
5. **if action1 == "1":**
6. *#If the button has action 1 then it will start the gameloop when pressed.*
7. gameloop()
8. **elif** action1 == "Q":
9. *#if the button has the action Q then exit the window*
10. **exit()**
11. **elif** action1 == "I":
12. game\_instruct()
13. [previous code...]
14. *#end function*

#### Menu Procedure

The menu procedure has been adjusted to include a new button for the instruction menu, which is included so the program has more usability features that instruct the player on how to play. This assists in completing **success criteria number 42.**

1. **def** game\_intro():
2. intro = True
3. *#Use while loop so that the Menu refreshes 60 times per second*
4. **while** intro:
5. **for event in pygame.event.get(): *# User did something***
6. **if** event.type == pygame.QUIT: *# If user clicked close*
7. intro = False *# Flag that we are done so we exit this loop and quit the game*
8. **elif** event.type==pygame.KEYDOWN:
9. **if** event.key==pygame.K\_ESCAPE:
10. **intro=False**
11. *#Drawing the menu screen*
12. screen.fill(BLACK)
13. *#Draw the Background image for the background*
14. screen.blit(BACKGROUND\_IMAGE, [0,0])
15. ***#Set the font and text for the title and center it.***
16. font = pygame.font.Font('freesansbold.ttf', 84)
17. text = font.render(str("DUNGEON ESCAPE"), 1, WHITE)
18. text\_rect = text.get\_rect(center=(screen\_width//2, screen\_height//6))
19. screen.blit(text, text\_rect)
20. ***#Draw the buttons for starting the game and quiting***
21. button\_1("START GAME",475,420,250,60,WHITE,GREY,"1")
22. button\_1("Instructions",475,490,250,60,WHITE,GREY,"I")
23. button\_1("QUIT",475,560,250,60,WHITE,GREY,"Q")
24. *#Display the image in the window*
25. **pygame.display.flip()**
26. *#Refresh at 60Hz*
27. clock.tick(60)
28. *#end procedure*

Graphical user interface

Description automatically generatedThis code produces the following screen:

#### Instruction Procedure

The instruction procedure adds a new menu which shows the user the controls used to play the game and which keys perform what operation. This is done by displaying an image and including 2 buttons, one to start the game and the other to exit the window. This completes **success criteria number 42** and uses the following code:

1. **def** game\_instruct():
2. running = True
3. **while** running:
4. **for** event **in** pygame.event.get(): *# User did something*
5. **if event.type == pygame.QUIT: *# If user clicked close***
6. running = False *# Flag that we are done so we exit this loop and quit the game*
7. **elif** event.type==pygame.KEYDOWN:
8. **if** event.key==pygame.K\_ESCAPE:
9. running=False
11. *#Drawing the menu screen*
12. screen.fill(BLACK)
13. *#Draw the Background image for the background*
14. screen.blit(BACKGROUND\_IMAGE, [0,0])
15. **screen.blit(INSTRUCT\_IMG, [250,225])**
16. *#Set the font and text for the title and center it.*
17. font = pygame.font.Font('freesansbold.ttf', 84)
18. text = font.render(str("INSTRUCTIONS"), 1, WHITE)
19. text\_rect = text.get\_rect(center=(screen\_width//2, screen\_height//6))
20. **screen.blit(text, text\_rect)**
21. *#Draw the buttons for starting the game and quiting*
22. button\_1("START GAME",475,720,250,60,WHITE,GREY,"1")
23. button\_1("QUIT",475,790,250,60,WHITE,GREY,"Q")
24. *#Display the image in the window*
25. **pygame.display.flip()**
26. *#Refresh at 60Hz*
27. clock.tick(60)
28. *#end procedure*

A screenshot of a video game

Description automatically generatedThis produces the following screen:

### Game Class:

#### Display Procedure

The display function now defines more attributes that are part of a new class called Spritesheet. Lines 4 to 15 are used to create a new object of the Spritesheet class and then define using the os library where the file for the image is. Furthermore, a new group has also been added on line 16 called the ground group which is used for a new class that just adds tiles with an image to every empty space.

1. **def** display(self, screen):
2. [previous code...]
3. *#define spritesheets in game class as attributes*
4. self.wallspritesheet = SpriteSheet(os.path.join(image\_path, TILESSHEET))
5. **self.playerspritesheet = SpriteSheet(os.path.join(image\_path, PLAYERSHEET))**
6. self.invertedplayerspritesheet = SpriteSheet(os.path.join(image\_path,"BlueKnightSpriteSheetflipped.png"))
7. self.cratespritesheet = SpriteSheet(os.path.join(image\_path, "crate.png"))
8. self.doorspritesheet = SpriteSheet(os.path.join(image\_path, "doorsheet.png"))
9. self.floorspritesheet = SpriteSheet(os.path.join(image\_path, "floor.png"))
10. **self.benemyspritesheet = SpriteSheet(os.path.join(image\_path, "bow enemy.png"))**
11. self.menemyspritesheet = SpriteSheet(os.path.join(image\_path, "menemy.png"))
12. self.bossspritesheet = SpriteSheet(os.path.join(image\_path, "boss enemy.png"))
13. self.keyspritesheet = SpriteSheet(os.path.join(image\_path, "silverkey.png"))
14. self.bulletspritesheet = SpriteSheet(os.path.join(image\_path, "fireball.png"))
15. **self.enemybulletspritesheet = SpriteSheet(os.path.join(image\_path, "enemyfireball.png"))**
16. self.ground\_group = pygame.sprite.Group()

#### QUIT Button

I added a new line of code to add a quit button on the game over screen so that it is easier to exit the game and fulfils success criteria 1. This is shown in line 12 in the following code below:

1. **def** display(self, screen):
2. [previous code...]
3. **if** self.score > self.highscore:
4. self.highscore = self.score
5. ***#Open the highscore file***
6. f = open(os.path.join(hs\_path, HS\_FILE), "w")
7. *#Change the file to have new high score*
8. f.write(str(self.score))
9. screen.blit(self.hs, self.text\_rect)
10. **screen.blit(text, text\_rect2)**
11. screen.blit(score, (520,500))
12. button\_1("QUIT",475,590,250,60,WHITE,GREY,"Q")

A brick wall with white text

Description automatically generated with low confidenceThis produces the following screen:

### Spritesheet Class:

#### Constructor Function

The constructor function of the spritesheet class is used to intake an image and convert it into an appropriate format to be used as a sprite, by removing transparent backgrounds.

1. **class** SpriteSheet(object):
2. *#Utility class required to pass and load spritesheets*
3. **def** \_\_init\_\_(self, filename):
4. *#image loaded has transparent background removed*
5. **self.spritesheet = pygame.image.load(filename).convert\_alpha()**

#### Get image function

The get image function is used in order to find the exact position within the image that is used as the sprite image. It takes in several parameters that determine where in the image, the size of which it will use from the image and the new size of the sprite image. It will then return the corrected sprite image.

1. **def** get\_image(self, x, y, width, height,newwidth, newheight):
2. *#grab image inside spritesheet*
3. image = pygame.Surface([width,height])
4. image.blit(self.spritesheet, (0,0), (x, y, width, height))
5. **image = pygame.transform.scale(image, (newwidth,newheight))**
6. **return** image

### Player Class:

#### Constructor Function

The constructor function has been adjusted to change the image of the player object to the image defined by the sprite-sheet class in which it uses the get\_image function to return the image with the correct coordinates and dimensions.

1. **def** \_\_init\_\_(self,color , width, height, x, y, health, score, money, gamekeys):
2. *#call sprite constructor*
3. super().\_\_init\_\_()
4. *#create a sprite*
5. **self.image = game.playerspritesheet.get\_image(82,1200,80,100,40,40)**
6. [previous code...]

A picture containing text

Description automatically generatedThe player now has this appearance:

### Player Bullet Class:

#### Constructor Function

The constructor function has been adjusted to change the image of the bullet object to the image defined by the sprite-sheet class in which it uses the get\_image function to return the image with the correct coordinates and dimensions.

1. **def** \_\_init\_\_(self, color, speedx, speedy):
2. *#Call the sprite constructor*
3. super().\_\_init\_\_()
4. *#Bullet created in the center of the player object*
5. **self.image = game.bulletspritesheet.get\_image(0,0,22,24,12,12)**
6. [previous code...]

The bullet has the following appearance:

### Ground Class

#### Constructor Function

The ground class was added in order to remove the background image of the game and instead replace it with a tiled, randomised pattern, which looks better. It only consists of a constructor function as its only purpose is as a background, thus it uses the sprite-sheet class and randomly rotates the image to generate a unique pattern for each level.

1. **def** \_\_init\_\_(self, x, y):
2. super().\_\_init\_\_()
3. self.id = random.randint(1,2)
4. **if** self.id == 1:
5. **self.image = game.floorspritesheet.get\_image(960,160,32,32,40,40)**
6. **else**:
7. self.image = pygame.transform.rotate(Game.floorspritesheet.get\_image(992,160,32,32,40,40), 90\*random.randint(0,3))
8. *#self.image.fill(color)*
9. *#set the position of the sprite*
10. **self.rect = self.image.get\_rect()**
11. self.rect.x = x
12. self.rect.y = y

A screenshot of a computer

Description automatically generated with medium confidenceThe ground class adds the following background to each level and helps distinguish the edge of the screen in comparison to the right hand side which includes the health bar and scores:

### Enemy Bullet Class:

#### Constructor Function

The constructor function has been adjusted to change the image of the bullet object to the image defined by the sprite-sheet class in which it uses the get\_image function to return the image with the correct coordinates and dimensions.

1. **def** \_\_init\_\_(self, color, speedx, speedy, x, y):
2. super().\_\_init\_\_()
3. self.image = game.enemybulletspritesheet.get\_image(0,0,22,24,12,12)
4. [previous code...]

A picture containing text

Description automatically generatedEnemy bullets have this appearance:

### Key Class

#### Constructor Function

The constructor function has been adjusted to change the image of the key object to the image defined by the sprite-sheet class in which it uses the get\_image function to return the image with the correct coordinates and dimensions.

1. **def** \_\_init\_\_(self,color,x,y):
2. *#Initiate sprite constructor*
3. super().\_\_init\_\_()
4. self.image = game.keyspritesheet.get\_image(0,0,38,22,25,15)
5. **[previous code...]**

A picture containing graphical user interface

Description automatically generatedThe key has the following appearance:

### Outer Wall Class:

#### Constructor Function

The constructor function has been adjusted to change the image of the wall object to the image defined by the sprite-sheet class in which it uses the get\_image function to return the image with the correct coordinates and dimensions.

1. **def** \_\_init\_\_(self, color, width, height, x, y, posx, posy):
2. *#call sprite constructor*
3. super().\_\_init\_\_()
4. *#create a sprite*
5. **self.image = game.wallspritesheet.get\_image(479,576,32,32,40,40)**
6. [previous code...]

A picture containing text, electronics, keyboard

Description automatically generatedOuter walls now display as the following:

### Inner Wall Class:

#### Constructor Function

The constructor function has been adjusted to change the image of the wall object to the image defined by the sprite-sheet class in which it uses the get\_image function to return the image with the correct coordinates and dimensions.

1. **def** \_\_init\_\_(self, color, width, height, x, y, posx, posy):
2. super().\_\_init\_\_(color, width, height, x, y, posx, posy)
3. self.image = game.cratespritesheet.get\_image(0,0,32,32,40,40)
4. [previous code...]

A picture containing text

Description automatically generatedInner walls now appear as follows:

### Door Class:

#### Constructor Function

The constructor function has been adjusted to change the image of the door object. Now the door’s image depends on its position in the level so that the image is correctly oriented depending if the door is on the left side or right side of the level.

1. **def** \_\_init\_\_(self, color, width, height, x, y, posx, posy):
2. super().\_\_init\_\_()
3. *#use sprite constructor*
4. self.angle = 0
5. **self.positionx = posx**
6. self.positiony = posy
7. **if** self.positionx == 0:
8. self.angle = 90
9. **elif** self.positionx == 24:
10. **self.angle = -90**
11. self.image = pygame.transform.rotate(game.doorspritesheet.get\_image(64,96,16,16,40,40), self.angle)
12. A picture containing qr code

    Description automatically generated [previous code...]

Doors appear as the following image:

### Spike Class:

#### Constructor Function

The constructor function has been adjusted to change the image of the spike object to the image defined by the sprite-sheet class in which it uses the get\_image function to return the image with the correct coordinates and dimensions and instead of using colours, we have two different images which it alternates between instead.

1. **def** \_\_init\_\_(self, color, width, height, x, y, posx, posy):
2. super().\_\_init\_\_()
3. *#Use sprite constructor*
4. self.image = pygame.image.load(os.path.join(image\_path, "spike down.png"))
5. **[previous code...]**

Spikes appearing in the following image when they are down:

#### Update Function

The update function still uses the colours to change but a new line is added to also update the spike object image.

1. **def** update(self):
2. *#Check if the time is greater than 4 seconds before changing state*
3. self.currentattacktime = pygame.time.get\_ticks()
4. **if** self.currentattacktime - self.previousattacktime > 4000:
5. **if self.color == GREY:**
6. self.color = LIGHTBLUE
7. self.image = pygame.image.load(os.path.join(image\_path, "spike up.png"))
8. *#reset the timer*
9. self.previousattacktime = self.currentattacktime
10. **else:**
11. *#If they are already active after 4 seconds then switch back to inactive*
12. self.color = GREY
13. self.image = pygame.image.load(os.path.join(image\_path, "spike down.png"))
14. *#reset the timer*
15. **self.previousattacktime = self.currentattacktime**
16. [previous code...]

Spikes appear as follows when they are up:

### Chest Class:

#### Constructor Function

The constructor function has been adjusted to change the image of the chest to the image defined by the sprite-sheet class in which it uses the get\_image function to return the image with the correct coordinates and dimensions.

1. **def** \_\_init\_\_(self,color, width, height, x, y, level):
2. super().\_\_init\_\_()
3. self.image = game.wallspritesheet.get\_image(224,0,32,32,40,40)
4. [previous code...]

Chests appear as follows:

### Sword Enemy Class:

#### Constructor Function

The constructor function has been adjusted to change the image of the enemy to the image defined by the sprite-sheet class in which it uses the get\_image function to return the image with the correct coordinates and dimensions.

1. **def** \_\_init\_\_(self, direction, width, height, x, y, health):
2. *#call sprite constructor*
3. super().\_\_init\_\_()
4. *#create a sprite*
5. **self.image = game.menemyspritesheet.get\_image(19,21,43,55,40,40)**
6. [previous code...]

A picture containing text, automaton, control panel

Description automatically generatedSword enemy now appears as the following:

### Bow Enemy Class:

#### Constructor Function

The constructor function has been adjusted to change the image of the bow enemy object to the image defined by the sprite-sheet class in which it uses the get\_image function to return the image with the correct coordinates and dimensions.

1. **def** \_\_init\_\_(self, direction, width, height, x, y, health):
2. *#call sprite constructor*
3. super().\_\_init\_\_()
4. *#create a sprite*
5. **self.image = game.benemyspritesheet.get\_image(19,21,43,55,40,40)**
6. [previous code...]

A picture containing text

Description automatically generatedBow enemy now appears as follows:

### Boss Enemy Class:

#### Constructor Function

The constructor function has been adjusted to change the image of the boss enemy to the image defined by the sprite-sheet class in which it uses the get\_image function to return the image with the correct coordinates and dimensions.

1. **def** \_\_init\_\_(self, direction, width, height, x, y, health):
2. *#call sprite constructor*
3. super().\_\_init\_\_()
4. *#create a sprite*
5. **self.image = game.bossspritesheet.get\_image(23,21,36,53,width,height)**
6. [previous code...]

### Errors

#### Spritesheets within the Game Class

The following error occurs when the code is run:

This error arises as each class that uses a constructor function attempts to locate the attribute within the game object which has not been created yet. As such I need to redefine these attributes such that they exist as variables that can be accessed before instantiation yet still exist within the game class.

As such I required to change the original code within the class to allow the variables to be called before the instantiation. I concluded that the code should form a variable of the class as shown below:

1. **def** display(self, screen):
2. [previous code...]
3. Game.wallspritesheet = SpriteSheet(os.path.join(image\_path, TILESSHEET))
4. Game.playerspritesheet = SpriteSheet(os.path.join(image\_path, PLAYERSHEET))
5. **Game.invertedplayerspritesheet = SpriteSheet(os.path.join(image\_path,"BlueKnightSpriteSheetflipped.png"))**
6. Game.cratespritesheet = SpriteSheet(os.path.join(image\_path, "crate.png"))
7. Game.doorspritesheet = SpriteSheet(os.path.join(image\_path, "doorsheet.png"))
8. Game.floorspritesheet = SpriteSheet(os.path.join(image\_path, "floor.png"))
9. Game.benemyspritesheet = SpriteSheet(os.path.join(image\_path, "bow enemy.png"))
10. **Game.menemyspritesheet = SpriteSheet(os.path.join(image\_path, "menemy.png"))**
11. Game.bossspritesheet = SpriteSheet(os.path.join(image\_path, "boss enemy.png"))
12. Game.keyspritesheet = SpriteSheet(os.path.join(image\_path, "silverkey.png"))
13. Game.bulletspritesheet = SpriteSheet(os.path.join(image\_path, "fireball.png"))
14. Game.enemybulletspritesheet = SpriteSheet(os.path.join(image\_path, "enemyfireball.png"))

Further to this, each line of code that called each sprite sheet also had to be changed so that it now used the following prefix:

self.image = Game.playerspritesheet.get\_image(82,1200,80,100,40,40)

This changed to lowercase game – which would call an attribute from the game object to an uppercase Game, which would call for the variable within the game class.

#### Images Cause Serious Performance Issues.

The use of the spritesheets cause the frames per second of the game to drop considerably from a constant 60 to a mere 10 frames per second. This is due to the spritesheet object constantly being defined within the display procedure which runs every time the game loop iterates. As a result, the spritesheets have been moved into the constructor function of the game class as shown below:

1. **def** \_\_init\_\_(self):
2. [previous code...]
3. Game.wallspritesheet = SpriteSheet(os.path.join(image\_path, TILESSHEET))
4. Game.playerspritesheet = SpriteSheet(os.path.join(image\_path, PLAYERSHEET))
5. **Game.invertedplayerspritesheet = SpriteSheet(os.path.join(image\_path,"BlueKnightSpriteSheetflipped.png"))**
6. Game.cratespritesheet = SpriteSheet(os.path.join(image\_path, "crate.png"))
7. Game.doorspritesheet = SpriteSheet(os.path.join(image\_path, "doorsheet.png"))
8. Game.floorspritesheet = SpriteSheet(os.path.join(image\_path, "floor.png"))
9. Game.benemyspritesheet = SpriteSheet(os.path.join(image\_path, "bow enemy.png"))
10. **Game.menemyspritesheet = SpriteSheet(os.path.join(image\_path, "menemy.png"))**
11. Game.bossspritesheet = SpriteSheet(os.path.join(image\_path, "boss enemy.png"))
12. Game.keyspritesheet = SpriteSheet(os.path.join(image\_path, "silverkey.png"))
13. Game.bulletspritesheet = SpriteSheet(os.path.join(image\_path, "fireball.png"))
14. Game.enemybulletspritesheet = SpriteSheet(os.path.join(image\_path, "enemyfireball.png"))

### Conclusion

Iteration 3 involved cleaning up any previous issues and adding textures and images to my classes in order to make the game more immersive. From this iteration, only **success criteria 41** was completed, but has really changed the appearance of the project. This final iteration has led to the conclusion of all the necessary requirements and accomplishes a requirement that was not necessary but severely developed the usability of the program. Following this, testing will now occur to test everything so that the game can be given to the selected users.

# Testing

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Test** | **Description** | **Input** | **Expected Outcome** | **Outcome** | **Pass/Fail** |
| 1 | Screen appears and is 1200 by 1000 pixels | Run the code | Window appears on the display with dimensions 1200 by 1000 pixels |  | Pass |
|  | | | | | |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | | | | | |
| 2 | Player moves right | Click D key | Player moves right 2 pixels |  | Pass |
|  | | | | | |
| 3 | Player moves left | Click A key | Player moves left 2 pixels |  | Pass |
|  | | | | | |
| 4 | Player moves forwards | Click W key | Player moves forward 2 pixels |  | Pass |
|  | | | | | |
| 5 | Player moves backwards | Click S key | Player moves back 2 pixels |  | Pass |
|  | | | | | |
| 6 | Player swings sword | Click SPACE key | Player executes swing animation and damages any enemies nearby. |  | Pass |
|  | | | | | |
| 7 | Player shoots bullet | Click E key | Player shoots bullet in direction of last pressed key |  | Pass |
|  | | | | | |
| 8 | Player cannot move through walls | Hold W, A, S or D while next to a wall | Player stops moving when touching wall |  | Pass |
|  | | | | | |
| 9 | Bullets cannot pass through walls. | Click E key | Bullet will continue moving and then disappear once it hits a wall |  | Pass |
|  | | | | | |
| 10 | Enemies spawn in an empty space when a level is created. | Run the code | No walls have been replaced by an enemy. |  | Pass |
|  | | | | | |
| 11 | The correct number of enemies are created per level | Run the code | There should be 2 times add 1 of the level number to create how many enemies there are per level. |  | Pass |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | | | | | |
| 12 | Enemies colliding with the wall and outside the range should reflect off the wall | Run the code | As an enemy object collides while travelling vertically or horizontally, the velocity of the enemy is multiplied by -1 causing the enemy to travel in the opposite direction. |  | Pass |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | | | | | |
| 13 | Enemies that are within range of the player and colliding with a wall cannot move through the wall but around it. | Run the code | Enemies should stop once colliding with a wall in that direction but still retain their perpendicular velocity to where they collided with the wall. |  | Pass |
|  | | | | | |
| 14 | Health bar should change size when player takes damage | Run the code and move player into enemy object | Bar decreases in size and changes to yellow when below 70 and then change to red when below 30. |  | Pass |
|  | | | | | |
| 15 | Sword Enemies move towards the player when in range | Run the code and move the player towards an enemy | Sword enemy x and y coordinates should keep changing until they are the same as the players x and y coordinates. |  | Pass |
|  | | | | | |
| 16 | The player can only heal to maximum health and heals every 10 seconds | Run the code and take damage | The attribute health of the player does not go past 100 when health is gained without increased maximum health |  | Pass |
|  | | | | | |
| 17 | The enemies can only attack once per second | Run the code and take damage from a Sword Enemy | The player’s health can only decrease by 10 every second for each enemy colliding with it. |  | Pass |
|  | | | | | |
| 18 | When the player has zero health the game ends | Run the code and take damage until the player reaches 0 health. | A screen saying Game Over should appear and all the objects are removed from the screen |  | Pass |
|  | | | | | |
| 19 | When an enemy is killed it should increase the score | Run the code and press space until an enemy is killed | The score attribute in the game class increments by 100 when each enemy reaches zero health. |  | Pass |
|  | | | | | |
| 20 | When an enemy is killed it should be removed from the screen and should create a key object | Run the code and kill an enemy | After the enemy object is removed from the all\_sprites group it should no longer be displayed and an instance of the key class is created and in the location of the enemy before it is removed. |  | Pass |
|  | | | | | |
| 21 | Keys collected should increase the key count | Collect a key | Key count on screen should increment by 1 |  | Pass |
|  | | | | | |
| 22 | Players cannot pass through doors until all keys are collected and chest opened | Run code and collect all keys and collide with chest | Door should open/disappear once all keys for level obtained and chest opened. |  | Pass |
|  | | | | | |
| 23 | When entering a door, the position of the player is changed. | Run the code and walk into open door | the player should be redrawn with the player moving position from xposition = 1000 to xposition = 50 |  | Pass |
|  | | | | | |
| 24 | When the level changes the levelsetup procedure removes all the objects from the previous level and replaces them with the new objects for the next level. | Run the code and walk into an open door | All objects are the screen are removed and replaced with new objects for the next level, with the doors closed. |  | Pass |
|  | | | | | |
| 25 | Player bullets should block enemy bullets | Run the code and click E key | Both player and enemy bullets should be removed from the screen when they collide, and the score should increment by 20 |  | Pass |
|  | | | | | |
| 26 | Player cannot take damage when spikes are retracted | Run the code and walk over spike | Damage will not be taken until the spike changes. Once the spike changes then 5 damage will be inflicted on the player’s health for every spike colliding with the player every second |  | Pass |
|  | | | | | |
| 27 | Player should unlock a new ability after completing each level and colliding with the chest | Run the code and press SPACE then walk over the chest. | Centred text appearing on the screen and are removed from the screen when entering a new level. |  | Pass |
|  | | | | | |
| 28 | Player can go back a level by going through a door | Run the code and walk through a door they have been through | All objects in the level should be removed and the previous level appears but no enemies should be created, and the doors should remain open. |  | Pass |
|  | | | | | |
| 29 | Swords can block bullets | Hold SPACE key | Score should increment by 20 when an enemy bullet collides with a sword. |  | Pass |
|  | | | | | |
| 30 | Player Health bar should be on the right-hand side of the screen | Run the code | A small white outlined rectangle with a green rectangle and number inside of it should appear over the rest of the objects. |  | Pass |
|  | | | | | |
| 31 | The boss health bar follows the boss around | Run the code | The health bar should be centred above the boss object and be overlayed over all other objects. |  | Pass |
|  | | | | | |
| 32 | The boss health bar should change colour from green to orange | Run the code | The health bar should change from green to orange drops below 70% |  | Pass |
|  | | | | | |
| 33 | The boss health bar should change from orange to red | Run the code | The health bar should change from orange to red when the boss health is below 30% |  | Pass |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | | | | | |
| 34 | The health bar should be a set size of 100 pixels by 45 pixels. | Run the code | The health bar should appear as 100 pixels across and 45 pixels wide. |  | Pass |
|  | | | | | |
| 35 | When the boss is killed, it should say game complete | Run the code and kill the boss and collide with the chest | Text centred in the screen reading “Boss Complete” should appear on the screen. |  | Pass |
|  | | | | | |
| 36 | Score should increase after completing a level | Run the code | Score should increment by 500 points when a new level is generated. |  | Pass |
|  | | | | | |
| 37 | When the Quit Button is pressed the window should close | Run the code and press the quit button on the game over screen | Quit button is pressed, the window that the game is running will exit. |  | Pass |
|  | | | | | |
| 38 | When the boss health is below 700 then the boss shoots bullets towards the player | Run the code and get the boss to below 700 health | Boss object should follow the player while shooting a bullet towards the player every 1.5 seconds |  | Pass |
|  | | | | | |
| 39 | When the boss health is below 400 the boss shoots 8 bullets in all directions from the boss | Run the code and get the boss below 400 health | Boss object creates 8 instances of bullets each with a velocity that causes them to travel vertically, horizontally or diagonally. Bullets are instantiated every 3.5 seconds. |  | Pass |
|  | | | | | |
| 40 | When the boss health is below 400 the boss creates 10 instances of the sword enemy class | Run the code and get the boss health below 400 health | 10 sword enemies are created and appear on the screen in random positions on the screen, without removing any walls or spike objects. |  | Pass |
|  | | | | | |
| 41 | Main Menu should include text on the side about the controls | Run the code | The displayed main menu should include information on how to move, shoot and attack with the sword to help the user learn the controls of the game. |  |  |
|  | | | | | |
| 42 | Game over screen displays the score of the current game and the highest score achieved by anyone playing the game | Run the code | The game over menu has text saying game over, the score and the high score. It should also include a quit button to exit the game. |  |  |

# Evaluation

## Specification

### Must Have:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Number | Requirement | Justification | Test Number | Requirement met? |
| 1 | A menu screen where you can choose to start the game or quit the window. | To allow the user to select when to begin playing the game and exit the game. | 1 | Yes – This requirement was completed and fully functional |
| 2 | The player can move up, left, down or right using w,a,s,d keys. | This allows the player to move around each level and avoid enemies, allowing them to proceed to the next level. | 2,3,4,5 | Yes – This requirement was completed and fully functional |
| 3 | Walls should block all bullets from the enemy or player and stop enemies or players | To optimise the program to run more optimally and remain the immersive environment of the game. | 9, 12, 13 | Yes – This requirement was completed and fully functional |
| 4 | The player can hold the space key to use his sword for 1 second. | To allow the player to inflict damage on the enemies, which is the main objective of the game. | 6 | Yes – This requirement was completed; however, the sword is not visible to the user as there was insufficient time to implement an animation that would make it visible. |
| 5 | The enemies are split into two classes, sword, or bow. | To add increased variety to the enemies in each level as requested by the clients. | 10, 11 | Yes – This requirement was completed and fully functional |
| 6 | The player can shoot bullets using the E key. | To add variety to the player’s attacks and to allow the player to inflict damage on enemies. | 7 | Yes – This requirement was completed and fully functional |
| 7 | The player has 100 health. | This is to allow the player to die to an enemy once it runs out of health so that it is possible to lose. | 16 | Yes – This requirement was completed and fully functional and the player’s maximum health also increases. |
| 8 | Enemies have 40 health each. | This is to allow the enemies to only require two hits in order to die. | 10, 11 | Yes – This requirement was completed and fully functional. |
| 9 | Enemies can attack the player once per second | This is to make it easier for the player as the player would lose health extremely quickly if there was no timer. | 17 | Yes – This requirement was completed and fully functional |
| 10 | The player’s health goes down when attacked by an enemy | This is to allow the player to lose health when attacked and this is to make the player be able to die and therefore lose the game. | 14 | Yes – This requirement was completed and fully functional |
| 11 | The game ends when the player has 0 health, and the player object is removed from the player group. | This is to allow the game to end when the player group has zero elements. | 18 | Yes – This requirement was completed and fully functional |
| 12 | Every enemy killed should increase the score. | This is to allow a score to be tallied to achieve high scores. | 19 | Yes – This requirement was completed and fully functional |
| 13 | Every key collected should increase the score and when all the keys for each room are collected then the doors can open. | This is to prevent player’s skipping a level and avoiding the enemies. | 20, 21, 22 | Yes – This requirement was completed and fully functional |
| 14 | Doors prevent the player from leaving the room without completing it. | This is to prevent level skipping. | 22, 23 | Yes – This requirement was completed and fully functional |
| 15 | 5 levels with each level having more enemies. | This is to increase the length of the game and increase its difficulty per level. | 11,24 | Yes – This requirement was completed and fully functional and more levels can be easily added in the game class. |
| 16 | Bow enemies shoot bullets that harm the player but cannot chase the player when in range. | This is to add variety to how the enemies can attack the player. | 15, 25 | Yes – This requirement was completed and fully functional |
| 17 | Sword enemies can chase the player when in range. | This is for the enemies to get close to the player to collide with them and attack the player. | 15 | Yes – This requirement was completed and fully functional |
| 18 | Player can block bullets with his sword or other bullets. | This is to make it easier for the player to avoid taking damage and losing. | 29 | Yes – This requirement was completed and fully functional however the sword swing is not visible. |
| 19 | Player’s sword attacks all enemies within its radius. | This is to make it easier for the player to attack multiple targets. | 6 | Yes – This requirement was completed and fully functional however the sword swing is not visible. |
| 20 | Player’s bullets travel in the direction of the player’s last key pressed. | This is to allow the player to shoot bullets in different directions. | 7 | Yes – This requirement was completed and fully functional |
| 21 | Enemy bullets are shot in the direction of the player from the enemy’s position. | This is to add variety to how enemies can attack the player and increase the difficulty of the game. | 15, 25 | Yes – This requirement was completed and fully functional |
| 22 | The 5th level is a boss level. | This is to add more difficulty and repeatability into the game and was requested by my clients. | 31, 38, 39, 40 | Yes – This requirement was completed and fully functional |
| 23 | The boss is larger and deals more damage. | This is to make the boss visually intimidating and increases its difficulty as it is harder to avoid. | 31, 38, 39, 40 | Yes – This requirement was completed and fully functional |
| 24 | The boss has multiple stages of attacks. | This is to increase the difficulty and add more variety to the boss level. | 38, 39, 40 | Yes – This requirement was completed and fully functional |
| 25 | When the boss is killed, the game finishes. | This is to allow the player to finish the game and obtain a score. | 35 | Yes – This requirement was completed and fully functional |
| 26 | The score increases when bullets are blocked using the player’s sword or bullets. | This is to allow more ways for the player to increase their score to achieve a high score. | 6, 29 | Yes – This requirement was completed and fully functional |
| 27 | The player can shoot one bullet per second. | This is to prevent the player from killing enemies too quickly and increases its difficulty | 7 | Yes – This requirement was completed and fully functional |

Every criterion in this section was met. Although some of the criteria required slightly different forms of completing compared with the design plan, each criterion was successfully implemented with complete functionality. Although not clearly state requirement 22 was met as the level was designed in the game class and is visible across the associated test numbers designated.

### Should Have:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Number | Requirement | Justification | Test Number | Requirement met? |
| 28 | Spikes, which extend or close every 4 seconds which deal damage to the player when stepped on and active. | Increasing the variety of enemies and difficulty as each level progresses. | 26 | Yes – This requirement was completed and fully functional |
| 29 | A high score file that stores the highest score and changes the high score if a larger score is reached. | This is to maintain a create a high score system where only the highest achieved score is stored. | 43 | Yes – This requirement was completed and fully functional |
| 30 | The boss spawns other enemies when low health. | This is to add more stages to my boss and increase its difficulty. | 41 | Yes – This requirement was completed and fully functional |
| 31 | The boss has its own health bar to inform the player how much damage has been inflicted. | This is to inform the player on the health of the boss to make it easier for the player. | 31,32,33,34,35 | Yes – This requirement was completed and fully functional |
| 32 | The player has a health bar which changes size depending on the amount of health the player has. | This is to help the player know how much health they have. | 14, 26, 30 | Yes – This requirement was completed and fully functional |
| 33 | Chests spawn when each level is completed to unlock a new ability for the player. | This gives the player a new ability each level, which has been requested by my clients. | 22 | Yes – This requirement was completed and fully functional |
| 34 | The player can increase his health after completing a level. | This makes it easier for the player to progress in each level. | 27 | Yes – This requirement was completed and fully functional |
| 35 | The player can increase his sword’s radius after completing a level. | This makes it easier for the player to attack many enemies. | 27 | Yes – This requirement was completed and fully functional |
| 36 | The player can unlock the ability to shoot many bullets every second after completing a level. | This makes it easier for the player to attack enemies and gives them a new ability. | 27 | Yes – This requirement was completed and fully functional |
| 37 | The player cannot shoot until it unlocks the ability after the first level. | This is to add ability progression into the game and adds some difficulty to the first level. | 27 | Yes – This requirement was completed and fully functional |
| 38 | The player can travel to previous levels without the level resetting. | This is to allow the player to travel back a level if they wanted to if I added a different level layout. | 28 | Yes – This requirement was completed however the chest objects still appear in completed levels, although they do not affect the game in anyway if they do appear. |
| 39 | The player regains health every 10 seconds. | This is to add passive healing that was requested by one of my clients. | 16 | Yes – This requirement was completed and fully functional |
| 40 | A game over screen that displays the highest score. | This is to inform the player of their score and the high score. | 43 | Yes – This requirement was completed and fully functional |

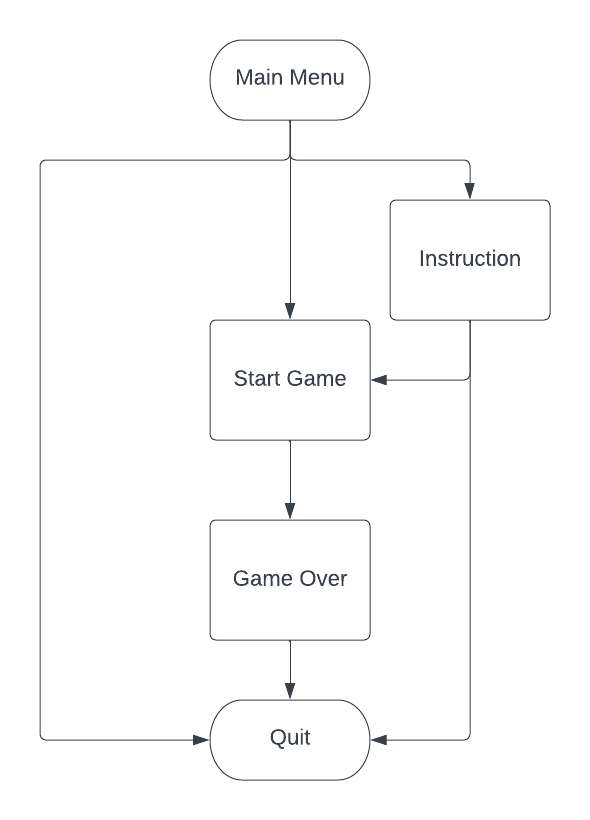
All the criterion in the should have section, although not integral to the game helped implement further depth to further entice the user to play the game and developed the game more. Almost every criterion in this section was completely met with a few criteria that have been partially met have very minor issues which do not affect the game significantly.

### Could Have:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Number | Requirement | Justification | Test Number | Requirement met? |
| 41 | Add images to each class and to give the game a theme, for example, the player is a knight, and the enemies are skeletons. | If I have more time, then I could implement more sprite sheets but maintain the same class and add a background. | 41, 42 | Yes – This requirement was partially completed as animations have not been added to every object due to a lack of time for completion however this does not retract from the playability of the game. |
| 42 | Main Menu should include text on the side about the controls | The displayed main menu should include information on how to move, shoot and attack with the sword to help the user learn the controls of the game. | 42 | Yes – This requirement was completed and fully functional through the use of a separate instruction menu. |
| 43 | Leader board of high scores with names of players. | Given more time I could improve the high score system to store both a name and high score after each game. | Not tested | No – This requirement was not attempted due to a lack of time |
| 44 | A game over screen that allows the player to input a name and save their score. | Given more time, I could improve the game over screen to include this however this is not necessary for my clients. | Not tested | No – This requirement was not attempted due to a lack of time |
| 45 | Use line of sight to allow the enemy to only attack if they can see the player | This improves the behaviour of the enemies to appear more realistic and would be a quality-of-life change if I had enough time. | Not tested | No – This requirement was not attempted due to a lack of time |

The “could have” section includes criterion that had the possibility to be completed in the given time frame. Only two of the 5 criteria in this section were completed as they provided the most impact on the appearance and playability of the game. This is due to the simplicity of implementing **criterion 41** which involved creating a new class and then adding every image for each object. Further to this **number 42** was very important as it introduced usability features in the form of an instruction menu. The final criterion in this section was very difficult to attempt to implement when attempting to utilise in-built functions of the pygame library and the included disadvantage of a lack of time prevent the implementation of the criterion in the project, however, could be implemented in a future development of the game. This also applies to the other criteria that were not completed in this section, which could all be included in a future iteration of the game.

## Usability Features

The implementation of usability features in the form of an instruction menu proved to be extremely advantageous should the game have future development. This is due to the instruction menu existing as an isolated function that adds modularity to the game menu and can easily be altered individually. Further to this, the button function also added further modularity as separate cases were included that would call on separate functions, either to start, quit or enter the instruction menu that produced the following menu branching:

The instruction menu consists of details which describe which keys can be pressed, along with all the other menu screens below:Graphical user interface

Description automatically generatedA brick wall with white text

Description automatically generated with low confidenceA screenshot of a video game

Description automatically generated

## Maintainability and Future Developments

### Maintenance:

The game uses an object-oriented approach which provides encapsulation through classes and methods. This allows for new functions to be easily implemented as amending the code will only require a change to the sub routine for the program to function. Should I want to add new sub-classes, code would not have to be adjusted significantly apart from introducing the new class and inheriting the methods and attributes of a previous class. For example, adding destructible walls could inherit the wall group but include new procedures that change the effect of the wall’s interaction with bullets or sword objects. Furthermore, my game loop and menu loop are called via functions and procedures which adds further modularity to my program. This allows for easier debugging and correcting of issues within my code or implementing new features. However, I have used global variables to define certain images and variables, to initiate the pygame library or assist in later stages of the program, that if adjusted could cause many errors.

### Essential Developments:

Essential changes that need to be made are adjusting how the sword enemies behave. Currently, they have a very simple function that uses if statements to check the range and simply adjusts the enemy’s position in the direction of the player. This causes enemies to easily become stuck attempting to move through a wall, which prevents the enemy from moving through. Instead, the sword enemies require a far more complex movement system using a line of site feature and a shortest path algorithm, which uses the grid with each square existing as a node. This suggestion comes from one of my clients, Chris, who found it easy to complete each level as the enemy objects could be forced into a position where they become stuck. Chris suggested this change to improve the experience of the game and fixing this issue would further increase the difficulty of the game. This will prevent the enemies from being unable to move towards the player if they collide with a wall. Further to this I need to add animations to every object class. This is to add further aesthetic adjustments that provide a more refined game. This can be easily sorted by creating a list formed of images which can be cycled through in an iterative loop when an action is performed. For example, when the player holds the A key, the player object not only moves to the left but also cycles through images to appear as if the player is walking. Deniz suggested this as it would add further finish to the game’s design and appearance that would improve the appearance of the game.

### Potential Developments:

Future changes that could be introduced are the inclusion of a leader board. This would require the file reading code to be adjusted slightly to read each line as a separate score alongside the name chosen by the player. A player could enter their name when the game ends with an input box on the game over screen. The score of the user will first be written to a line in the text file followed by the name of the player. The game over screen would require another function that would display all the users that have their names and scores on the leader board. This could be done using the text object function that already is used for buttons and create another function that will organise each score and name correctly on the screen. This suggestion was provided by Chris, so that he could compare scores with his friends, however, this was not a necessary change that he thought would greatly improve the experience.

In addition to this, I could adjust how the player shoots bullets. Instead of checking the last key pressed, I could use the mouse position on the screen, calculate the angle between the player and the mouse position and use this angle to adjust the direction which the player’s bullets are fired. The change would be made in the player class by adjusting the xspeed and yspeed variables before they are passed as parameters in the instantiation of the bullet class. Another possibility is creating a player-centric camera, which keeps the player in the centre of the screen. For this to work, I would need to create a new class for the camera that actively checks the player’s position on the screen and displaces the position of the camera to keep the player centralised. Each level would also require adjustments to increase the size of each object such that only parts of each level are visible on screen and not the entire level. Deniz suggested this change as he felt it was difficult and unintuitive to shoot bullets from the player. Using a mouse to direct the player’s bullets would allow the player to move in a different direction to where he shoots.

### Optimisation Developments:

Although the program would require re-implementing, I could export the program to a different language such as C++, although a lower-level language would prove to be more efficient than python and would be easier to distribute as python uses an interpreter while C++ uses a compiler which would produce an executable file that is smaller and faster to use. Further to this, some classes require their methods to be broken down into further sub procedures that would improve the modularity of the code, particularly in update functions, where all collisions and decisions are handled in for every object. Both my clients have suggested that this is not necessary for the game to run and continuing to use Python will function well as the game will not be mass distributed. Further to this, Deniz, my client suggested using more efficient functions and libraries to improve performance without changing to a compiled language.

# Appendix

## Bibliography

Stack Overflow: <https://stackoverflow.com/>

Program Arcade Games: <http://programarcadegames.com/>

Pygame Documentation: <https://www.pygame.org/docs/>

W3 Schools File handling: <https://www.w3schools.com/python/python_file_handling.asp>

## Complete Code:

1. **import** pygame
2. **import** random
3. **import** os
4. **import** sys
5. **import numpy as np**
6. **import** math
7. **from** collections **import** deque
8. vec = pygame.math.Vector2

11. *#use os to find the path of this file:*
12. current\_path = os.path.dirname(\_\_file\_\_)
13. *#use os to find the folder called 'images' in the same folder as this file*
14. image\_path = os.path.join(current\_path, 'images')
15. ***# Define colours as tuples***
16. BLACK = (0, 0, 0)
17. WHITE = (255, 255, 255)
18. GREEN = (0, 255, 0)
19. RED = (255, 0, 0)
20. **YELLOW = (255, 255, 0)**
21. PINK = (204,102,153)
22. PURPLE = (75,0,130)
23. BROWN = (150, 75, 0)
24. ORANGE = (230,165,0)
25. **BLUE = (30,144,255)**
26. PERFECTBLUE = (0,0,255)
27. LIGHTBLUE = (173, 216, 240)
28. GREY = (180,180,180)
29. *#use os to find the menu background image in the images folder*
30. **BACKGROUND\_IMAGE = pygame.image.load(os.path.join(image\_path, 'Menu background.png'))**
31. INSTRUCT\_IMG = pygame.image.load(os.path.join(image\_path, 'Instructions.png'))
32. *#Define constants for grid class for pathfinding:s*
33. *#Define the size of each tile (size of walls/ player)*
34. TILESIZE = 40
35. ***#Specify how many tiles form the width of the grid and the height of the grid.***
36. GRIDWIDTH = 25
37. GRIDHEIGHT = 25
38. WIDTH = TILESIZE \* GRIDWIDTH
39. HEIGHT = TILESIZE \* GRIDHEIGHT
40. ***#Define the highscore file as highscore.txt***
41. HS\_FILE = "highscore.txt"
42. PLAYERSHEET = "BlueKnightSpriteSheet.png"
43. TILESSHEET = "tilessheet.png"
44. *#Use os to find the highscore file*
45. **hs\_path = os.path.join(current\_path, 'highscore')**
46. *#Use os to find the folder called icons to use when showing the start and end points in the pathfinding algorithm*
47. icon\_dir = os.path.join(current\_path, 'icons')
48. *#Initiate pygame module*
49. pygame.init()
51. *# Set the width and height of the screen [width, height]*
52. size = (1200, 1000)
53. screen\_width = 1200
54. screen\_height = 1000
55. **screen = pygame.display.set\_mode(size)**
56. *#Set the window name to DUNGEON ESCAPE*
57. pygame.display.set\_caption("Dungeon Escape")
58. *#Define the clock used for what refresh rate the game runs at.*
59. clock = pygame.time.Clock()
60. ***#Create a function used to format text used in buttons***
61. **def** text\_objects(text,font):
62. *#Render the text input as white and return the position of the surface.*
63. textSurface = font.render(text, True, WHITE)
64. **return** textSurface, textSurface.get\_rect()
65. ***#end procedure***
67. *#Create a button function that interacts with the mouse. When hovered over by the mouse change colour and define the x position, y position and width and height of the Button.*
68. *#Also specify what the button does when it is pressed and what text is in the button*
69. **def** button\_1(message,xpos,ypos,width,height,inactivecolor,activecolor,action1=None):
70. ***#Find the position of the mouse***
71. mouse = pygame.mouse.get\_pos()
72. *#Check if the mouse is being clicked*
73. click = pygame.mouse.get\_pressed()
74. *#Check if the mouse's position is the same as the button's position*
75. **if xpos+width >mouse[0] > xpos and ypos+height > mouse[1] > ypos:**
76. *#Draw the button with its inactive colour*
77. pygame.draw.rect(screen, inactivecolor,(xpos,ypos,width,height),5)
78. *#Check if the mouse is clicked:*
79. **if** click[0] == 1 **and** action1 !=None:
80. ***#Check what the action of the button does.***
81. **if** action1 == "1":
82. *#If the button has action 1 then it will start the gameloop when pressed.*
83. gameloop()
84. **elif** action1 == "Q":
85. ***#if the button has the action Q then exit the window***
86. exit()
87. **elif** action1 == "I":
88. game\_instruct()
89. **else**:
90. ***#When the button is not hovered over by the mouse then draw the button with it's active colour***
91. pygame.draw.rect(screen, activecolor,(xpos,ypos,width,height),5)
92. *#Draw the text of the buutton in the middle of the button*
93. smallText = pygame.font.Font("freesansbold.ttf",30)
94. textSurf, textRect = text\_objects(message, smallText)
95. **textRect.center = ( (xpos+(width/2)), (ypos+(height/2)) )**
96. screen.blit(textSurf, textRect)
97. *#end function*



102. *#Create a function for the Main Menu of the game.*
103. *#Display the buttons for Starting the game and Quiting.*
104. **def** game\_intro():
105. **intro = True**
106. *#Use while loop so that the Menu refreshes 60 times per second*
107. **while** intro:
108. **for** event **in** pygame.event.get(): *# User did something*
109. **if** event.type == pygame.QUIT: *# If user clicked close*
110. **intro = False *# Flag that we are done so we exit this loop and quit the game***
111. **elif** event.type==pygame.KEYDOWN:
112. **if** event.key==pygame.K\_ESCAPE:
113. intro=False
115. ***#Drawing the menu screen***
116. screen.fill(BLACK)
117. *#Draw the Background image for the background*
118. screen.blit(BACKGROUND\_IMAGE, [0,0])
119. *#Set the font and text for the title and center it.*
120. **font = pygame.font.Font('freesansbold.ttf', 84)**
121. text = font.render(str("DUNGEON ESCAPE"), 1, WHITE)
122. text\_rect = text.get\_rect(center=(screen\_width//2, screen\_height//6))
123. screen.blit(text, text\_rect)
124. *#Draw the buttons for starting the game and quiting*
125. **button\_1("START GAME",475,420,250,60,WHITE,GREY,"1")**
126. button\_1("Instructions",475,490,250,60,WHITE,GREY,"I")
127. button\_1("QUIT",475,560,250,60,WHITE,GREY,"Q")
128. *#Display the image in the window*
129. pygame.display.flip()
130. ***#Refresh at 60Hz***
131. clock.tick(60)
132. *#end procedure*
134. **def** game\_instruct():
135. **running = True**
136. **while** running:
137. **for** event **in** pygame.event.get(): *# User did something*
138. **if** event.type == pygame.QUIT: *# If user clicked close*
139. running = False *# Flag that we are done so we exit this loop and quit the game*
140. **elif event.type==pygame.KEYDOWN:**
141. **if** event.key==pygame.K\_ESCAPE:
142. running=False
144. *#Drawing the menu screen*
145. **screen.fill(BLACK)**
146. *#Draw the Background image for the background*
147. screen.blit(BACKGROUND\_IMAGE, [0,0])
148. screen.blit(INSTRUCT\_IMG, [250,225])
149. *#Set the font and text for the title and center it.*
150. **font = pygame.font.Font('freesansbold.ttf', 84)**
151. text = font.render(str("INSTRUCTIONS"), 1, WHITE)
152. text\_rect = text.get\_rect(center=(screen\_width//2, screen\_height//6))
153. screen.blit(text, text\_rect)
154. *#Draw the buttons for starting the game and quiting*
155. **button\_1("START GAME",475,720,250,60,WHITE,GREY,"1")**
156. button\_1("QUIT",475,790,250,60,WHITE,GREY,"Q")
157. *#Display the image in the window*
158. pygame.display.flip()
159. *#Refresh at 60Hz*
160. **clock.tick(60)**
161. *#end procedure*



166. *#Create a game loop function that is called after the Menu*
167. **def** gameloop():
168. *#Create a function that draws the icons used in the shortest path and center them*
169. **def** draw\_icons():
170. **start\_center = (game.menemy.goal.x \* TILESIZE + TILESIZE / 2, game.menemy.goal.y \* TILESIZE + TILESIZE / 2)**
171. screen.blit(home\_img, home\_img.get\_rect(center=start\_center))
172. goal\_center = (game.menemy.start.x \* TILESIZE + TILESIZE / 2, game.menemy.start.y \* TILESIZE + TILESIZE / 2)
173. screen.blit(cross\_img, cross\_img.get\_rect(center=goal\_center))








183. *#Loop until the user clicks the close button.*
184. done = False
186. *# Used to manage how fast the screen updates*
187. clock = pygame.time.Clock()


191. *#Create A Game class that holds all the Sprite groups and level information*
192. *#Use a game object to establish a game loop that is used to setup the game*
193. **class** Game(object):
194. **def** \_\_init\_\_(self):
195. ***#Define all the atributes of the Game class***
196. *#Set the Score of the game to 0*
197. self.score = 0
198. *#define spritesheets in game class as attributes*
199. Game.wallspritesheet = SpriteSheet(os.path.join(image\_path, TILESSHEET))
200. **Game.playerspritesheet = SpriteSheet(os.path.join(image\_path, PLAYERSHEET))**
201. Game.invertedplayerspritesheet = SpriteSheet(os.path.join(image\_path,"BlueKnightSpriteSheetflipped.png"))
202. Game.cratespritesheet = SpriteSheet(os.path.join(image\_path, "crate.png"))
203. Game.doorspritesheet = SpriteSheet(os.path.join(image\_path, "doorsheet.png"))
204. Game.floorspritesheet = SpriteSheet(os.path.join(image\_path, "floor.png"))
205. **Game.benemyspritesheet = SpriteSheet(os.path.join(image\_path, "bow enemy.png"))**
206. Game.menemyspritesheet = SpriteSheet(os.path.join(image\_path, "menemy.png"))
207. Game.bossspritesheet = SpriteSheet(os.path.join(image\_path, "boss enemy.png"))
208. Game.keyspritesheet = SpriteSheet(os.path.join(image\_path, "silverkey.png"))
209. Game.bulletspritesheet = SpriteSheet(os.path.join(image\_path, "fireball.png"))
210. **Game.enemybulletspritesheet = SpriteSheet(os.path.join(image\_path, "enemyfireball.png"))**
211. *#Use an attribute to tell whether the game has finished*
212. self.game\_over = False
213. *#Set the game level to 0*
214. self.level = 0
215. ***# Create a different sprite groups to be used for collisions and updates***
216. self.all\_sprites\_group = pygame.sprite.Group()
217. self.outsidewall\_group = pygame.sprite.Group()
218. self.wall\_group = pygame.sprite.Group()
219. self.innerwall\_group = pygame.sprite.Group()
220. **self.player\_group = pygame.sprite.Group()**
221. self.bullet\_group = pygame.sprite.Group()
222. self.enemy\_group = pygame.sprite.Group()
223. self.menemy\_group = pygame.sprite.Group()
224. self.key\_group = pygame.sprite.Group()
225. **self.portal\_group = pygame.sprite.Group()**
226. self.door\_group = pygame.sprite.Group()
227. self.sword\_group = pygame.sprite.Group()
228. self.spike\_group = pygame.sprite.Group()
229. self.chest\_group = pygame.sprite.Group()
230. **self.boss\_group = pygame.sprite.Group()**
231. self.enemybullet\_group = pygame.sprite.Group()
232. self.ground\_group = pygame.sprite.Group()
233. *#Call the load data method to be used to find and store high scores*
234. self.load\_data()
235. ***#Create a list of which chests have been unlocked***
236. self.chestunlocked = [False, False, False, False, False]
237. *#Set a timer for how long to display text*
238. self.previoustexttime = pygame.time.get\_ticks()
239. *#Create a list of all the levels that have been completed*
240. **self.levelcomplete = [False, False, False, False, False]**
241. *#Create a 2D array of each level with each digit representing a tile of 40 by 40 pixels*
242. self.level1 = [
243. [1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1],
244. [1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1],
245. **[1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1],**
246. [1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1],
247. [1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1],
248. [1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1],
249. [1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1],
250. **[1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1],**
251. [1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1],
252. [1,0,0,0,0,0,0,0,0,2,2,2,2,2,2,0,0,0,0,0,0,0,0,0,1],
253. [1,0,0,0,0,0,0,0,2,2,0,0,0,0,2,2,0,0,0,0,0,0,0,0,1],
254. [1,0,0,0,0,0,0,0,7,0,0,0,0,0,0,7,0,0,0,0,0,0,0,0,6],
255. **[1,0,0,0,0,0,0,0,2,2,0,0,0,0,2,2,0,0,0,0,0,0,0,0,1],**
256. [1,0,0,0,0,0,0,0,0,2,2,2,2,2,2,0,0,0,0,0,0,0,0,0,1],
257. [1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1],
258. [1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1],
259. [1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1],
260. **[1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1],**
261. [1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1],
262. [1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1],
263. [1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1],
264. [1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1],
265. **[1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1],**
266. [1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1],
267. [1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1]
269. ]
271. self.level2 = [
272. [1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1],
273. [1,0,0,0,0,0,0,0,0,2,0,0,0,0,0,0,0,0,2,0,0,0,0,0,1],
274. [1,0,0,0,0,0,0,0,0,2,0,0,0,0,0,0,0,0,2,0,0,0,0,0,1],
275. **[1,0,0,0,2,2,2,2,2,2,0,0,0,0,0,0,0,0,2,0,0,2,2,2,1],**
276. [1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1],
277. [1,0,0,0,0,0,0,0,0,2,2,2,2,0,0,0,0,0,0,0,0,0,0,0,1],
278. [1,0,0,0,0,0,0,0,0,2,0,0,0,0,0,0,0,0,2,0,0,0,0,0,1],
279. [1,0,0,0,0,0,2,2,2,2,0,0,0,0,0,0,0,0,2,0,0,2,2,2,1],
280. **[1,0,0,0,0,0,2,0,0,0,0,0,0,0,0,0,0,2,2,0,0,0,0,0,1],**
281. [1,0,0,0,0,0,2,0,2,2,2,2,2,2,2,2,0,2,0,0,0,0,0,0,1],
282. [1,0,0,0,0,0,2,2,2,2,0,0,0,0,2,2,2,2,0,0,0,0,0,0,1],
283. [6,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,6],
284. [1,0,0,0,0,2,2,2,2,2,0,0,0,0,2,2,2,2,2,2,2,2,0,0,1],
285. **[1,0,0,0,0,2,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,2,0,0,1],**
286. [1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,2,0,0,1],
287. [1,2,0,2,2,2,0,0,0,0,0,0,0,0,0,0,0,2,2,2,2,2,0,0,1],
288. [1,0,0,0,0,2,0,0,0,0,0,0,0,0,0,0,0,2,0,0,0,0,0,0,1],
289. [1,0,0,0,0,2,0,0,0,0,0,0,0,0,0,2,2,2,0,0,0,2,2,2,1],
290. **[1,0,0,0,0,2,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,2,0,0,1],**
291. [1,0,0,0,0,2,0,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,0,0,1],
292. [1,0,0,0,0,2,0,0,0,0,0,0,0,0,2,0,0,0,0,0,0,0,0,0,1],
293. [1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1],
294. [1,0,0,0,0,2,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1],
295. **[1,0,0,0,0,2,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1],**
296. [1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1]
298. ]
300. **self.level3 = [**
301. [1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1],
302. [1,0,0,0,0,0,0,0,0,2,0,0,0,0,0,0,2,0,0,0,0,0,0,0,1],
303. [1,0,0,0,0,0,0,0,0,2,0,0,0,0,0,0,2,0,0,0,0,0,0,0,1],
304. [1,0,0,0,0,0,0,0,0,2,0,0,0,0,0,0,2,0,0,0,0,0,0,0,1],
305. **[1,0,0,0,0,0,0,0,0,2,0,0,0,0,0,0,2,0,0,0,0,0,0,0,1],**
306. [1,0,0,0,0,0,2,2,2,2,0,0,0,0,0,0,7,0,0,0,0,0,0,0,1],
307. [1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,7,0,0,0,0,0,0,0,1],
308. [1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,2,0,0,0,0,0,0,0,1],
309. [1,0,0,0,0,0,0,0,0,2,0,0,0,0,0,0,2,2,2,7,7,2,2,2,1],
310. **[1,0,0,0,0,0,0,0,0,2,0,0,0,0,0,0,2,0,0,0,0,0,0,0,1],**
311. [1,0,0,0,0,0,0,0,0,2,2,2,2,2,2,2,2,0,0,0,0,0,0,0,1],
312. [6,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,7,0,0,0,0,0,0,0,6],
313. [1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,7,0,0,0,0,0,0,0,1],
314. [1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,7,0,0,0,0,0,0,0,1],
315. **[1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,7,0,0,0,0,0,0,0,1],**
316. [1,2,2,2,2,0,0,0,2,2,2,2,2,2,2,2,2,0,0,0,0,0,0,0,1],
317. [1,0,0,0,0,0,0,0,2,0,0,0,0,0,0,0,2,0,0,0,0,0,0,0,1],
318. [1,0,0,0,0,0,0,0,2,0,0,0,0,0,0,0,7,0,0,0,0,0,0,0,1],
319. [1,0,0,0,0,2,2,2,2,0,0,0,0,0,0,0,7,0,0,0,0,0,0,0,1],
320. **[1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,2,2,7,7,2,2,2,2,1],**
321. [1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,2,0,0,0,0,0,0,0,1],
322. [1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,2,0,0,0,0,0,0,0,1],
323. [1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,2,0,0,0,0,0,0,0,1],
324. [1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,2,0,0,0,0,0,0,0,1],
325. **[1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1]**
327. ]
329. self.level4 = [
330. **[1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1],**
331. [1,0,0,0,0,0,2,0,0,0,0,7,0,0,0,0,0,0,0,0,0,0,0,0,1],
332. [1,0,0,0,0,0,2,0,0,0,0,7,0,0,0,0,0,0,0,0,0,0,0,0,1],
333. [1,0,0,0,0,0,2,0,0,0,0,2,0,0,0,0,0,0,0,0,0,0,0,0,1],
334. [1,0,0,0,0,0,2,0,0,0,0,2,0,0,0,0,0,2,2,0,0,0,0,0,1],
335. **[1,0,0,0,0,0,7,0,0,0,0,2,0,0,0,0,0,2,2,0,0,0,0,0,1],**
336. [1,0,0,0,0,0,7,0,0,0,0,2,0,0,0,0,0,0,0,0,0,0,0,0,1],
337. [1,0,0,0,0,0,7,0,0,0,0,2,0,0,0,0,0,0,0,0,0,0,0,0,1],
338. [1,0,0,0,0,0,2,0,0,0,0,2,0,0,0,0,0,0,0,0,0,0,0,0,1],
339. [1,0,0,0,0,0,2,0,0,0,0,2,0,0,0,0,0,0,0,0,0,0,0,0,1],
340. **[1,2,2,2,2,2,2,0,0,0,0,2,2,2,2,2,2,2,2,2,2,2,2,2,1],**
341. [6,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,2,0,0,0,0,0,6],
342. [1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,2,0,0,0,0,0,1],
343. [1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,2,0,0,0,0,0,1],
344. [1,2,2,2,2,2,2,0,0,0,0,0,0,0,0,0,0,0,2,0,0,0,0,0,1],
345. **[1,0,0,0,0,0,2,0,0,0,0,0,0,0,0,0,2,2,2,7,7,7,7,7,1],**
346. [1,0,0,0,0,0,2,0,0,0,0,0,0,0,0,0,0,0,7,0,0,0,0,0,1],
347. [1,0,0,0,0,0,7,0,0,0,0,0,0,0,0,0,0,0,7,0,0,0,0,0,1],
348. [1,0,0,0,0,0,7,0,0,0,0,0,0,0,0,0,0,0,7,0,0,0,0,0,1],
349. [1,0,0,0,0,0,7,0,0,0,0,0,0,0,0,0,0,0,7,0,0,0,0,0,1],
350. **[1,0,0,0,0,0,2,0,0,0,0,0,0,0,0,0,0,0,2,0,0,0,0,0,1],**
351. [1,0,0,0,0,0,2,0,0,0,0,0,0,0,0,0,0,0,2,0,0,0,0,0,1],
352. [1,0,0,0,0,0,2,0,0,0,0,0,0,0,0,0,0,0,2,0,0,0,0,0,1],
353. [1,0,0,0,0,0,2,0,0,0,0,0,0,0,0,0,0,0,2,0,0,0,0,0,1],
354. [1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1]
356. ]
358. self.level5 = [
359. [1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1],
360. **[1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1],**
361. [1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1],
362. [1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1],
363. [1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1],
364. [1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1],
365. **[1,0,0,0,0,0,7,7,7,7,7,7,7,7,7,7,7,7,7,0,0,0,0,0,1],**
366. [1,0,0,0,0,0,7,0,0,0,0,0,0,0,0,0,0,0,7,0,0,0,0,0,1],
367. [1,0,0,0,0,0,7,0,0,0,0,0,0,0,0,0,0,0,7,0,0,0,0,0,1],
368. [1,0,0,0,0,0,7,0,0,0,0,0,0,0,0,0,0,0,7,0,0,0,0,0,1],
369. [1,0,0,0,0,0,7,0,0,0,0,0,0,0,0,0,0,0,7,0,0,0,0,0,1],
370. **[1,0,0,0,0,0,7,0,0,0,0,0,0,0,0,0,0,0,7,0,0,0,0,0,6],**
371. [1,0,0,0,0,0,7,0,0,0,0,0,0,0,0,0,0,0,7,0,0,0,0,0,1],
372. [1,0,0,0,0,0,7,0,0,0,0,0,0,0,0,0,0,0,7,0,0,0,0,0,1],
373. [1,0,0,0,0,0,7,0,0,0,0,0,0,0,0,0,0,0,7,0,0,0,0,0,1],
374. [1,0,0,0,0,0,7,0,0,0,0,0,0,0,0,0,0,0,7,0,0,0,0,0,1],
375. **[1,0,0,0,0,0,7,0,0,0,0,0,0,0,0,0,0,0,7,0,0,0,0,0,1],**
376. [1,0,0,0,0,0,7,0,0,0,0,0,0,0,0,0,0,0,7,0,0,0,0,0,1],
377. [1,0,0,0,0,0,7,7,7,7,7,7,7,7,7,7,7,7,7,0,0,0,0,0,1],
378. [1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1],
379. [1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1],
380. **[1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1],**
381. [1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1],
382. [1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1],
383. [1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1]
385. **]**
386. *#Create a list of all the levels*
387. self.levels = [self.level1, self.level2, self.level3,self.level4, self.level5]
388. *#Initiate the level setup for the first level*
389. self.levelsetup()
390. ***#Create an object of the player class and add him to Sprite groups***
391. self.player = Player(WHITE, 40, 40,40,500,100,0,0,0)
392. self.all\_sprites\_group.add(self.player)
393. self.player\_group.add(self.player)
394. *#end method*
396. *#Create a method to read the highscore file and set the highscore in the game*
397. **def** load\_data(self):
398. *#load high score file*
399. f = open(os.path.join(hs\_path, HS\_FILE), "r")
400. **try:**
401. *#If file has data in it then set the highscore*
402. self.highscore = int(f.readline())
403. *#print(self.highscore)*
404. **except**:
405. ***#Else if this doesn't work then set the highscore to 0***
406. self.highscore = 0
407. *#print(self.highscore)*
408. *#load spritesheet*
410. ***#end method***
412. *#Create a method that creates all the objects according to the 2D array in the game class.*
413. **def** levelsetup(self):
414. *#Set the count to 0*
415. **self.count = 0**
416. self.bosscount = 0
417. *#Spawn the boss enemy when the level is 5*
418. **if** (self.level + 1) % 5 == 0:
419. self.levels[self.level][10][10] = 5
421. **else**:
422. *#When the level is not level 5 then create enemies according to what level the player is on.*
423. enemies = 0
424. **while** enemies != ((2\*(self.level+1)) + 1):
425. ***#Find two random numbers between 1 and 23***
426. xpos = random.randint(1,23)
427. ypos = random.randint(1,23)
428. *#Check if the numbers correspond to coordinates of walls or other sprites*
429. **if** self.levels[self.level][xpos][ypos] !=1 **and** self.levels[self.level][xpos][ypos] != 2 **and** self.levels[self.level][xpos][ypos] != 7:
430. ***#set the coordinates to a random number to spawn either a Melee enemy or Bow enemy***
431. self.levels[self.level][xpos][ypos] = random.randint(3,4)
432. enemies = enemies +1
434. *#Search through the 2D array and create objects that corespond to each coordinate that forms the grid.*
435. **for j in range(len(self.levels[self.level])):**
436. **for** i **in** range(len(self.levels[self.level][j])):
437. *#print(i,j)*
438. *#Check what each character is in every index then create objects according to each character*
439. char = self.levels[self.level][j][i]
440. **if char == 1:**
441. *#Create outside wall object and add it to corresponding Sprite groups and add the coordinates to the wall list*
442. self.outsidewall = Wall(RED,40,40,i\*40, j\*40, i, j)
443. self.all\_sprites\_group.add(self.outsidewall)
444. self.wall\_group.add(self.outsidewall)
445. **self.outsidewall\_group.add(self.outsidewall)**
446. *#g.walls.append(vec((self.outsidewall.rect.x/40), (self.outsidewall.rect.y/40)))*
447. **if** char == 2:
448. *#Create inner object and add it to corresponding Sprite groups and add the coordinates to the wall list*
449. self.innerwall = InnerWall(RED,40,40,i\*40, j\*40, i, j)
450. **self.all\_sprites\_group.add(self.innerwall)**
451. self.wall\_group.add(self.innerwall)
452. self.innerwall\_group.add(self.innerwall)
453. *#g.walls.append(vec((self.innerwall.rect.x/40), (self.innerwall.rect.y/40)))*
454. **if** char == 3:
455. ***#Create Bow enemy object and add it to corresponding Sprite groups***
456. **if** self.levelcomplete[self.level] == False:
457. self.benemy = BowEnemy(random.randint(0,10),40,40, i\*40, j\*40, 40)
458. self.all\_sprites\_group.add(self.benemy)
459. self.enemy\_group.add(self.benemy)
460. **if char == 4:**
461. *#Create Melee enemy object and add it to corresponding Sprite groups*
462. **if** self.levelcomplete[self.level] == False:
463. self.menemy = MeleeEnemy(random.randint(0,10),40,40, i\*40, j\*40, 40)
464. self.all\_sprites\_group.add(self.menemy)
465. **self.enemy\_group.add(self.menemy)**
466. self.menemy\_group.add(self.menemy)
467. **if** char == 5:
468. *#Create Boss object and add it to corresponding Sprite groups*
469. **if** self.levelcomplete[self.level] == False:
470. **self.boss = BossEnemy(random.randint(0,10),160,160, i\*40, j\*40, 1000)**
471. self.all\_sprites\_group.add(self.boss)
472. self.boss\_group.add(self.boss)
473. **if** char == 6:
474. *#Create Door object and add it to corresponding Sprite groups*
475. **if self.levelcomplete[self.level] == False:**
476. self.door = Door(PURPLE,40,40,i\*40, j\*40, i, j)
477. self.all\_sprites\_group.add(self.door)
478. self.wall\_group.add(self.door)
479. self.door\_group.add(self.door)
480. **if char == 7:**
481. *#Create Spike object and add it to corresponding Sprite groups*
482. self.spike = Spikes(GREY, 40,40,i\*40, j\*40, i, j)
483. *#self.all\_sprites\_group.add(self.spike)*
484. self.spike\_group.add(self.spike)
486. self.ground = Ground(i\*40,j\*40)
487. self.ground\_group.add(self.ground)
488. *#end method*
490. ***#Method to delete each level when the player completes it***
491. **def** leveldelete(self):
492. *#Remove all sprites and readd the player*
493. self.all\_sprites\_group.empty()
494. self.wall\_group.empty()
495. **self.spike\_group.empty()**
496. self.enemy\_group.empty()
497. self.bullet\_group.empty()
498. self.menemy\_group.empty()
499. self.key\_group.empty()
500. **self.portal\_group.empty()**
501. self.door\_group.empty()
502. self.sword\_group.empty()
503. self.chest\_group.empty()
504. self.boss\_group.empty()
505. **self.enemybullet\_group.empty()**
506. self.ground\_group.empty()
507. self.all\_sprites\_group.add(self.player)
508. self.all\_sprites\_group.update()
509. self.score += 500
511. *#endprocess*
512. *#Method used to check for event if they quit*
513. **def** eventprocess(self):
514. **for** event **in** pygame.event.get():
515. **if event.type == pygame.QUIT:**
516. **return** True
517. **return** False
519. *#Method to get score of the game*
520. **def getscore(self):**
521. **return** self.score
522. *#endprocess*
524. *#Method that processes all interactions not within other classes*
525. **def runlogic(self):**
526. **if** **not** self.game\_over:
527. *#When the game is running and all the enemies and boss are dead create a chest*
528. **if** len(self.enemy\_group) == 0:
529. **if** len(self.boss\_group) == 0:
530. **self.levelcomplete[self.level] = True**
531. **if** self.levelcomplete[self.level] == True:
532. **if** len(self.chest\_group) == 0:
533. *#Use count to only create one chest per level*
534. **if** self.count == 0:
535. **self.count+=1**
536. **print**(self.count)
537. *#create chest object and add it to sprite groups*
538. self.chest = Chest(BROWN, 40, 40, 460,440, self.level)
539. self.all\_sprites\_group.add(self.chest)
540. **self.chest\_group.add(self.chest)**
541. *#print(self.levelcomplete)*
542. *# Move all the sprites*
543. self.sword\_group.update()
544. self.spike\_group.update()
545. **self.all\_sprites\_group.update()**
546. *#Check if the player has died*
547. **if** len(self.player\_group) == 0:
548. *#If true, then end the game*
549. self.game\_over = True
550. ***#print(len(game.enemy\_group))***
551. *#Change the level when the player goes past the boundaries(goes through the doors)*
552. **if** self.player.rect.x > 1000:
553. *#Check if this is the last level and if not set to true and increase the level*
554. **if** self.level != (len(self.levels)-1):
555. **self.levelcomplete[self.level] = True**
556. self.chestunlocked[self.level] = False
557. self.level += 1
558. *#Reset the room keys needed to proceed to next levels*
559. self.player.gamekeys = 0
560. **self.player.rect.x = 40**
561. *#Run level delete then create the next level*
562. self.leveldelete()
563. self.levelsetup()
565. **else:**
566. *#If the final level then end the game*
567. self.game\_over = True
568. *#Check if the player goes back a level*
569. **elif** self.player.rect.x < 0:
570. ***#set the level back one and then setup the new level***
571. self.chestunlocked[self.level] = False
572. self.level -= 1
573. self.leveldelete()
574. self.player.rect.x = 960
575. **self.levelsetup()**
576. *#end method*
578. *#Create a method to display all sprites and text*
579. **def** display(self, screen):
580. ***# background image.***
581. screen.fill(BLACK)
582. *#screen.blit(BACKGROUND\_IMAGE, (0,0))*
583. **if** self.game\_over:
584. *#If the game is over then create a game over screen*
585. ***#First set the screen black***
586. screen.fill(BLACK)
587. screen.blit(BACKGROUND\_IMAGE, (0,0))
588. *#Then draw text of Game over, score and Highscore*
589. font1 = pygame.font.Font(None, 74)
590. **font2 = pygame.font.Font(None, 48)**
591. font = pygame.font.Font(None, 48)
592. text = font1.render('GAME OVER', 1, WHITE)
593. score = font2.render('SCORE:'+str(self.getscore()), 1, WHITE)
594. self.hs = font.render('HIGHSCORE:'+str(self.highscore),1, WHITE)
595. **self.text\_rect = self.hs.get\_rect(center=((screen\_width)/2, 700))**
596. text\_rect2 = text.get\_rect(center=((screen\_width)/2, 300))
597. *#Check if the score is greater than the old highscore and set the highscore as the score*
598. **if** self.score > self.highscore:
599. self.highscore = self.score
600. ***#Open the highscore file***
601. f = open(os.path.join(hs\_path, HS\_FILE), "w")
602. *#Change the file to have new high score*
603. f.write(str(self.score))
604. screen.blit(self.hs, self.text\_rect)
605. **screen.blit(text, text\_rect2)**
606. screen.blit(score, (520,500))
607. button\_1("QUIT",475,590,250,60,WHITE,GREY,"Q")
608. **if** **not** self.game\_over:
609. *#When the game is running Display text on the right handside of the screen with the score and keys of the player*
610. **font = pygame.font.Font(None, 24)**
611. score = font.render('SCORE:'+str(self.getscore()), 1, WHITE)
612. keys = font.render('KEYS:'+str(self.player.getkeys()), 1, WHITE)
613. *#move = font.render('TO MOVE')*
614. *#attack1 = font.render('TO SWING SWORD')*
615. ***#attack2 = font.render('TO SHOOT FIREBALL')***
616. *#Draw the player function for the health bar of the player*
617. self.player.advanced\_health()
618. *#Draw the text of the player's health over the health bar*
619. health = font.render(str(self.player.current\_health), 1, WHITE)
620. **screen.blit(score, (1050,500))**
621. screen.blit(keys, (1050,550))
622. screen.blit(health, (1081, 51))
624. *# --- Drawing code for sprites*
625. **self.sword\_group.draw(screen)**
626. self.ground\_group.draw(screen)
627. self.spike\_group.draw(screen)
628. self.all\_sprites\_group.draw(screen)
629. self.boss\_group.draw(screen)
630. ***#Check which chest has been unlocked and display text until player moves to next level***
631. **if** self.chestunlocked[self.level] == True:
632. **if** self.level == 0:
633. font3 = pygame.font.Font(None, 48)
634. msg = font3.render('FIREBALL UNLOCKED', 1, BLUE)
635. **text\_rect = msg.get\_rect(center=((screen\_width-200)/2, screen\_height/6))**
636. screen.blit(msg, text\_rect)
637. **if** self.level == 1:
638. font3 = pygame.font.Font(None, 48)
639. msg = font3.render('SWORD RADIUS INCREASED', 1, BLUE)
640. **text\_rect = msg.get\_rect(center=((screen\_width-200)/2, screen\_height/6))**
641. screen.blit(msg, text\_rect)
642. **if** self.level == 2:
643. font3 = pygame.font.Font(None, 48)
644. msg = font3.render('HEALTH INCREASED', 1, BLUE)
645. **text\_rect = msg.get\_rect(center=((screen\_width-200)/2, screen\_height/6))**
646. screen.blit(msg, text\_rect)
647. **if** self.level == 3:
648. font3 = pygame.font.Font(None, 48)
649. msg = font3.render('MULTISHOT UNLOCKED', 1, BLUE)
650. **text\_rect = msg.get\_rect(center=((screen\_width-200)/2, screen\_height/6))**
651. screen.blit(msg, text\_rect)
652. **if** self.level == 4:
653. font3 = pygame.font.Font(None, 48)
654. msg = font3.render('BOSS COMPLETE', 1, BLUE)
655. **text\_rect = msg.get\_rect(center=((screen\_width-200)/2, screen\_height/6))**
656. screen.blit(msg, text\_rect)
657. *#Check if the level is multiple of 5. If true then display the boss health bar.*
658. **if** (self.level + 1) % 5 == 0:
659. self.boss.advanced\_health()
660. ***#endif***
661. *# --- Go ahead and update the screen with what we've drawn.*
662. pygame.display.flip()
663. *#end procedure*



668. **class** SpriteSheet(object):
669. *#Utility class required to pass and load spritesheets*
670. **def \_\_init\_\_(self, filename):**
671. *#image loaded has transparent background removed*
672. self.spritesheet = pygame.image.load(filename).convert\_alpha()
674. **def** get\_image(self, x, y, width, height,newwidth, newheight):
675. ***#grab image inside spritesheet***
676. image = pygame.Surface([width,height])
677. image.blit(self.spritesheet, (0,0), (x, y, width, height))
678. image = pygame.transform.scale(image, (newwidth,newheight))
679. **return** image





686. *# Create a Player Class*
687. **class** Player(pygame.sprite.Sprite):
688. *#define constants for speed of player*
689. speed\_x = 0
690. **speed\_y = 0**
691. *#Method for when the player object is initially created.*
692. **def** \_\_init\_\_(self,color , width, height, x, y, health, score, money, gamekeys):
693. *#call sprite constructor*
694. super().\_\_init\_\_()
695. ***#create a sprite***
696. self.walkimagearray = [Game.playerspritesheet.get\_image(82,1200,80,100,40,40),Game.playerspritesheet.get\_image(103,907,80,100,40,40),Game.playerspritesheet.get\_image(291,908,87,97,40,40), Game.playerspritesheet.get\_image(388,908,88,98,40,40), Game.playerspritesheet.get\_image(485,908,90,99,40,40),Game.playerspritesheet.get\_image(590,908,82,99,40,40), Game.playerspritesheet.get\_image(688,908,81,99,40,40),Game.playerspritesheet.get\_image(786,908,80,98,40,40),Game.invertedplayerspritesheet.get\_image(1917,908,80,99,40,40),Game.invertedplayerspritesheet.get\_image(1820,908,80,99,40,40),Game.invertedplayerspritesheet.get\_image(1723,908,86,97,40,40),]
697. self.attackarray = []
698. self.image = self.walkimagearray[0]
699. self.image.set\_colorkey(GREEN)
700. **self.numberarray = 0**
701. *#self.image.fill(color)*
702. *#set the position of the sprite*
703. self.rect = self.image.get\_rect()
704. *#Set the current and maximum health of the player to be used for health bar*
705. **self.current\_health = 50**
706. self.maximum\_health = health
707. self.health\_bar\_length = 180
708. self.target\_health = 100
709. self.health\_change\_speed = 2
710. **self.health\_bar\_color = GREEN**
711. self.health\_ratio = self.maximum\_health/ self.health\_bar\_length
712. *#Player score is tallied and added to game score*
713. self.score = score
714. *#Money attribute currently not used but can be used in a future update*
715. **self.money = money**
716. *#keys used to check to open doors to go to next level*
717. self.gamekeys = gamekeys
718. self.rect.x = x
719. self.rect.y = y
720. ***#Set original direction for player***
721. self.directionx = 0
722. self.directiony = 5
723. *#Set player attribute that are unlocked after opening chests*
724. self.canshoot = False
725. **self.multishot = False**
726. self.swordradius = 50
727. self.bulletcount = 3
728. *#Set timers to be used for attacks*
729. self.previoushealthtime = pygame.time.get\_ticks()
730. **self.previousbullettime = pygame.time.get\_ticks()**
731. self.previousdamagetime = pygame.time.get\_ticks()
732. self.previousattacktime = pygame.time.get\_ticks()
733. self.previousbulletaddtime = pygame.time.get\_ticks()
735. ***#end procedure***
737. *#Method to increase health of player*
738. **def** gethealth(self, amount):
739. **if** self.target\_health < self.maximum\_health:
740. **self.target\_health += amount**
741. **if** self.target\_health >= self.maximum\_health:
742. self.target\_health = self.maximum\_health
743. *#endprocedure*
745. ***#Method to remove health of palyer***
746. **def** getdamage(self,amount):
747. **if** self.target\_health > 0:
748. self.target\_health -= amount
749. **if** self.target\_health <=0:
750. **self.target\_health = 0**
751. *#end method*
753. *#Method used to create helath bar*
754. **def** advanced\_health(self):
755. **transition\_width = 0**
756. transition\_color = RED
758. *#When health is added set the transition bar to green and increase health bar at set speed*
759. **if** self.current\_health < self.target\_health:
760. **self.current\_health += self.health\_change\_speed**
761. transition\_width = int((self.target\_health - self.current\_health)/ self.health\_ratio)
762. transition\_color = GREEN
763. *#When health is removed set the transition bar to yellow and decrease the health bar at a set speed*
764. **if** self.current\_health > self.target\_health:
765. **self.current\_health -= self.health\_change\_speed**
766. transition\_width = int((self.target\_health - self.current\_health)/ self.health\_ratio)
767. transition\_color = YELLOW
769. *#Change the health bar colour depending on the amount of health*
770. **if self.current\_health >= 70:**
771. self.health\_bar\_color = GREEN
772. **if** self.current\_health >= 50 **and** self.current\_health < 70:
773. self.health\_bar\_color = ORANGE
774. **if** self.current\_health < 30 **and** self.current\_health >=0:
775. **self.health\_bar\_color = RED**
776. *#set the health bar size equal to the current health divided by max health*
777. health\_bar\_width = int(self.current\_health/ self.health\_ratio)
778. health\_bar = pygame.Rect(1005,45, health\_bar\_width, 25)
779. *#transition bar is set to the right of the health bar*
780. **transition\_bar = pygame.Rect(health\_bar.right, 45, transition\_width, 25)**
781. *#Draw all 3 bars over each other*
782. pygame.draw.rect(screen, self.health\_bar\_color, health\_bar)
783. pygame.draw.rect(screen,transition\_color, transition\_bar)
784. pygame.draw.rect(screen, WHITE, (1005, 45, self.health\_bar\_length, 25), 4)
785. ***#end method***
787. *#Get player score*
788. **def** getscore(self):
789. **return** self.score
790. ***#endprocedure***
792. *#get player's node position vector on grid*
793. **def** getpos(self):
794. **return** vec(self.rect.x/40, self.rect.y/40)
795. ***#endfunction***
797. *#set player score*
798. **def** setscore(self, score):
799. self.score = score
800. ***#endfunction***
801. *#get player's key number*
802. **def** getkeys(self):
803. **return** self.gamekeys
804. *#endprocedure*
806. *#set player's key number*
807. **def** setkeys(self, keys):
808. self.gamekeys = keys
809. *#endfunction*
811. *#get player's money*
812. **def** getmoney(self):
813. **return** self.money
814. *#endprocedure*
816. *#set player's money*
817. **def** setmoney(self, money):
818. self.money = money
819. *#endfunction*
821. *#Change the player's speed by the values entered*
822. **def** changespeed(self, x, y):
823. self.speed\_x += x
824. self.speed\_y += y
826. *#Update method for player to control movement and damage taken*
827. **def** update(self):
828. **if** self.speed\_x ==0 **and** self.speed\_y == 0:
829. self.image = Game.playerspritesheet.get\_image(82,1200,80,100,40,40)
830. **self.image.set\_colorkey(GREEN)**
831. *#Check for what keys are pressed*
832. keys = pygame.key.get\_pressed()
833. **if** keys[pygame.K\_a]:
834. *#Move the player to the left and change the direction*
835. **self.changespeed(-2,0)**
836. self.directionx = -4
837. self.directiony = 0
838. self.numberarray += 0.2
839. self.image = self.walkimagearray[int(self.numberarray)]
840. **self.image.set\_colorkey(GREEN)**
841. **if** self.numberarray > 7:
842. self.numberarray = 0
843. **if** keys[pygame.K\_d]:
844. *#move the player to the right and change the direction*
845. **self.changespeed(2,0)**
846. self.directionx = 4
847. self.directiony = 0
848. **if** keys[pygame.K\_w]:
849. *#move the player up and change the direction*
850. **self.changespeed(0,-2)**
851. self.directionx = 0
852. self.directiony = -4
853. **if** keys[pygame.K\_s]:
854. *#move the player down and change the direction*
855. **self.changespeed(0,2)**
856. self.directionx = 0
857. self.directiony = 4
858. **if** keys[pygame.K\_e]:
859. *#When the E key is pressed and bullet shooting is unlocked then create a fireball that travel in the direction set by the player from the previous key pressed*
860. **if self.canshoot == True:**
861. *#Check if multishot is unlocked*
862. **if** self.multishot == False:
863. **if** self.bulletcount > 0:
864. *#Create a bullet once per second*
865. **self.currentbullettime = pygame.time.get\_ticks()**
866. **if** self.currentbullettime - self.previousbullettime > 1000:
867. bullet = Bullet(RED, self.directionx, self.directiony)
868. game.bullet\_group.add(bullet)
869. game.all\_sprites\_group.add(bullet)
870. ***#remove 1 from the bullet count***
871. self.bulletcount -= 1
872. self.previousbullettime = self.currentbullettime
873. **else**:
874. *#If multishot is active then produce a bullet every 100 milliseconds only if bulletcount is greater than zero*
875. **if self.bulletcount > 0:**
876. self.currentbullettime = pygame.time.get\_ticks()
877. **if** self.currentbullettime - self.previousbullettime > 100:
878. *#remove 1 from bullet count so bullets can only be produced in maximum of 3 per second*
879. self.bulletcount -=1
880. **bullet = Bullet(RED, self.directionx, self.directiony)**
881. game.bullet\_group.add(bullet)
882. game.all\_sprites\_group.add(bullet)
883. self.previousbullettime = self.currentbullettime
884. **if** keys[pygame.K\_SPACE]:
885. ***#Check if Space has been pressed and if the sword hasn't already been created***
886. **if** len(game.sword\_group) == 0:
887. *#create a sword with a set radius that lasts for 1 second and add the sword to its sprite groups*
888. self.currentattacktime = pygame.time.get\_ticks()
889. **if** self.currentattacktime - self.previousattacktime > 1000:
890. **sword = Sword(BLACK, self.swordradius)**
891. game.sword\_group.add(sword)
892. *#game.all\_sprites\_group.add(sword)*
893. self.previousattacktime = self.currentattacktime
894. *#Check if the timer is greater than one to add bullets to the bulletcount*
895. **self.currentbulletaddtime = pygame.time.get\_ticks()**
896. **if** self.currentbulletaddtime - self.previousbulletaddtime > 1000:
897. **if** self.multishot == False:
898. self.bulletcount += 1
899. **else**:
900. **self.bulletcount +=3**
901. self.previousbulletaddtime = self.currentbulletaddtime
903. *#Bulletcount is limited to 3*
904. **if** self.bulletcount > 3:
905. **self.bulletcount = 3**
906. *#Refresh the health bar so an increase in the maximum health resets the health bar ratio*
907. self.health\_ratio = self.maximum\_health/ self.health\_bar\_length
908. self.currenthealthtime = pygame.time.get\_ticks()
909. *#Check if 10 seconds have passed to add 10 health to the palyer*
910. **if self.currenthealthtime -self.previoushealthtime > 10000:**
911. self.gethealth(10)
912. self.previoushealthtime = self.currenthealthtime
913. *#Check if the player's health is 0 and kill the player to end the game.*
914. **if** self.current\_health < 1:
915. **game.score += 100**
916. self.kill()
917. *#use the move function by the speed specified according to the key press*
918. self.move(self.speed\_x,self.speed\_y)
919. *#Reset the speeds to prevent the player accelerating*
920. **self.speed\_x = 0**
921. self.speed\_y = 0
922. *#end procedure*

925. ***#Method that takes input speeds and moves players position according to speed.***
926. **def** move(self, speedx, speedy):
927. *#move left or right*
928. self.rect.x += self.speed\_x
929. *#Check if the player collides with the wall*
930. **wallcollision = pygame.sprite.spritecollide(self,game.wall\_group, False)**
931. **for** wall **in** wallcollision:
932. **if** self.speed\_x > 0:
933. *#stop the player moving into the wall*
934. self.rect.right = wall.rect.left
935. **else:**
936. self.rect.left = wall.rect.right
937. *#move the player up and down the screen*
938. self.rect.y += self.speed\_y
939. *#check for collision*
940. **wallcollision = pygame.sprite.spritecollide(self, game.wall\_group, False)**
941. **for** wall **in** wallcollision:
942. *#if there is a collision while moving up then set the speed to 0*
943. **if** self.speed\_y > 0:
944. *#stop the player moving into the wall*
945. **self.rect.bottom = wall.rect.top**
946. **else**:
947. self.rect.top = wall.rect.bottom
948. *#end procedure*
949. *#end class*
951. *#Class for Player bullets*
952. **class** Bullet(pygame.sprite.Sprite):
953. *#Method to initiate bullet objects*
954. **def** \_\_init\_\_(self, color, speedx, speedy):
955. ***#Call the sprite constructor***
956. super().\_\_init\_\_()
957. *#Bullet created in the center of the player object*
958. self.image = Game.bulletspritesheet.get\_image(0,0,22,24,12,12)
959. self.image.set\_colorkey(BLACK)
960. **self.rect = self.image.get\_rect()**
961. self.speedx = speedx
962. self.speedy = speedy
963. self.rect.y = game.player.rect.y + 10
964. self.rect.x = game.player.rect.x + 10
966. *#Update method that moves the player by the speed which is defined by the direction that the player last moved in.*
967. **def** update(self):
968. self.rect.y += self.speedy
969. *#Check if it collides with a wall and kills the sprite*
970. **if pygame.sprite.groupcollide(game.bullet\_group, game.wall\_group, True, False) == True:**
971. self.kill()
972. self.rect.x += self.speedx
973. *#Check if it collides with a wall and kills the sprite*
974. **if** pygame.sprite.groupcollide(game.bullet\_group, game.wall\_group, True, False) == True:
975. **self.kill()**
976. *#Check if it travel past the boundaries and kills the sprite*
977. **if** self.rect.x > 1000:
978. self.kill()
979. **if** self.rect.x < 0:
980. **self.kill()**
982. **class** Ground(pygame.sprite.Sprite):
983. **def** \_\_init\_\_(self, x, y):
984. super().\_\_init\_\_()
985. **self.id = random.randint(1,2)**
986. **if** self.id == 1:
987. self.image = Game.floorspritesheet.get\_image(960,160,32,32,40,40)
988. **else**:
989. self.image = pygame.transform.rotate(Game.floorspritesheet.get\_image(992,160,32,32,40,40), 90\*random.randint(0,3))
990. ***#self.image.fill(color)***
991. *#set the position of the sprite*
992. self.rect = self.image.get\_rect()
993. self.rect.x = x
994. self.rect.y = y
995. ***#end procedure***
996. *#No updates needed for wall class*
997. **def** update(self):
998. **pass**
999. *#end procedure*
1001. *#Enemy Bullet Class shot from Bow enemy*
1002. **class** EnemyBullet(pygame.sprite.Sprite):
1003. **def** \_\_init\_\_(self, color, speedx, speedy, x, y):
1004. super().\_\_init\_\_()
1005. **self.image = Game.enemybulletspritesheet.get\_image(0,0,22,24,12,12)**
1006. self.image.set\_colorkey(BLACK)
1007. self.rect = self.image.get\_rect()
1008. self.speedx = speedx
1009. self.speedy = speedy
1010. **self.rect.y = y**
1011. self.rect.x = x
1013. *#Update method that moves the bullet according to the speed set by the bow enemy*
1014. **def** update(self):
1015. ***#move the bullet up and down***
1016. self.rect.y += self.speedy
1017. *#check for wall collision and kill the bullet*
1018. **if** pygame.sprite.groupcollide(game.enemybullet\_group, game.wall\_group, True, False) == True:
1019. self.kill()
1021. *#move the bullet left or right*
1022. self.rect.x += self.speedx
1023. *#check for wall collision and kill the bullet*
1024. **if** pygame.sprite.groupcollide(game.enemybullet\_group, game.wall\_group, True, False) == True:
1025. **self.kill()**
1026. *#Check if it travel past the boundaries and kills the sprite*
1027. **if** self.rect.x > 1000:
1028. self.kill()
1029. **if** self.rect.x < 0:
1030. **self.kill()**
1032. *#Check if it collides with the player and inflict 10 damage on the player*
1033. player\_hit\_group = pygame.sprite.groupcollide(game.player\_group, game.enemybullet\_group, False, True)
1034. **for** game.player **in** player\_hit\_group:
1035. **game.player.getdamage(10)**
1036. *#Check for collision with player bullet and kill the bullet and then kill itself and add points to game score*
1037. enemybullet\_hit\_group = pygame.sprite.groupcollide(game.enemybullet\_group, game.bullet\_group, False, True)
1038. **for** self **in** enemybullet\_hit\_group:
1039. self.kill()
1040. **game.score +=20**
1041. *#Check for collision with player's sword and kill itself when colliding and add points to game score*
1042. enemysword\_hit\_group = pygame.sprite.groupcollide(game.enemybullet\_group, game.sword\_group, False, False)
1043. **for** self **in** enemysword\_hit\_group:
1044. self.kill()
1045. **game.score +=20**

1048. *#Sword Class that is used with the player*
1049. **class** Sword(pygame.sprite.Sprite):
1050. **def \_\_init\_\_(self, color, radius):**
1051. *#call sprite constructor*
1052. super().\_\_init\_\_()
1053. self.image = pygame.Surface((radius, radius))
1054. *#self.image.fill(BLACK)*
1055. ***#Create a circle to form the radius that the sword covers***
1056. pygame.draw.circle(self.image, (color), (int(radius/2), int(radius/2)), int(radius/2))
1057. self.rect = self.image.get\_rect()
1058. self.rect.center = (0, 0)
1059. self.previousattacktime = pygame.time.get\_ticks()
1060. **self.radius = radius**
1062. *#Update function to set the sword to always be centered on the player depending on the size of the radius*
1063. **def** update(self):
1064. **if** self.radius == 50:
1065. **self.rect.y = game.player.rect.y -5**
1066. self.rect.x = game.player.rect.x -5
1067. **else**:
1068. self.rect.y = game.player.rect.y -15
1069. self.rect.x = game.player.rect.x -15
1070. ***#Set the sword to remove itself after 750 milliseconds after Space is pressed***
1071. self.currentattacktime = pygame.time.get\_ticks()
1072. **if** self.currentattacktime - self.previousattacktime > 750:
1073. self.kill()
1074. self.previousattacktime = self.currentattacktime



1079. *#Key class that is created after every enemy is killed*
1080. **class Key(pygame.sprite.Sprite):**
1081. **def** \_\_init\_\_(self,color,x,y):
1082. *#Initiate sprite constructor*
1083. super().\_\_init\_\_()
1084. self.image = Game.keyspritesheet.get\_image(0,0,38,22,25,15)
1085. **self.image.set\_colorkey(BLACK)**
1086. self.rect = self.image.get\_rect()
1087. self.rect.y = y
1088. self.rect.x = x
1090. ***#Update function check for collision and increase game score***
1091. **def** update(self):
1092. *#Check for collision with player and add one to key attribute in player*
1093. key\_hit\_group = pygame.sprite.groupcollide(game.key\_group, game.player\_group, False, False)
1094. **for** self **in** key\_hit\_group:
1095. ***#increase score and then remove itself.***
1096. game.player.gamekeys += 1
1097. game.score += 50
1098. self.kill()


1102. *#Wall class to create boundaries*
1103. **class** Wall(pygame.sprite.Sprite):
1104. **def** \_\_init\_\_(self, color, width, height, x, y, posx, posy):
1105. ***#call sprite constructor***
1106. super().\_\_init\_\_()
1107. *#create a sprite*
1108. self.image = Game.wallspritesheet.get\_image(479,576,32,32,40,40)
1109. *#self.image.fill(color)*
1110. ***#set the position of the sprite***
1111. self.rect = self.image.get\_rect()
1112. self.rect.x = x
1113. self.rect.y = y
1114. self.positionx = posx
1115. **self.positiony = posy**
1116. *#end procedure*
1117. *#No updates needed for wall class*
1118. **def** update(self):
1119. **pass**
1120. ***#end procedure***
1122. *#Method to find the vector position of the wall in the grid*
1123. **def** getpos(self):
1124. **return** [self.positionx/40, self.positiony/40]
1126. *#Innerwall Class inherits the wall class*
1127. **class** InnerWall(Wall):
1128. **def** \_\_init\_\_(self, color, width, height, x, y, posx, posy):
1129. super().\_\_init\_\_(color, width, height, x, y, posx, posy)
1130. **self.image = Game.cratespritesheet.get\_image(0,0,32,32,40,40)**

1133. *#endprocedure*
1135. ***#Update function for the inner wall could be used in the future to make them destructible***
1136. **def** update(self):
1137. **pass**
1139. *#Door class is used to prevent the player from proceeding to next level without completing the level.*
1140. **class Door(pygame.sprite.Sprite):**
1141. **def** \_\_init\_\_(self, color, width, height, x, y, posx, posy):
1142. super().\_\_init\_\_()
1143. *#use sprite constructor*
1144. self.angle = 0
1145. **self.positionx = posx**
1146. self.positiony = posy
1147. **if** self.positionx == 0:
1148. self.angle = 90
1149. **elif** self.positionx == 24:
1150. **self.angle = -90**
1151. self.image = pygame.transform.rotate(Game.doorspritesheet.get\_image(64,96,16,16,40,40), self.angle)
1152. self.image.set\_colorkey(ORANGE)
1153. *#self.image.fill(color)*
1154. *#set the position of the sprite*
1155. **self.rect = self.image.get\_rect()**
1156. self.rect.x = x
1157. self.rect.y = y


1161. *#end procedure*
1163. *#Update function checks what the level is and checks if the level is over before removing the door object*
1164. **def** update(self):
1165. **if (game.level + 1) % 5 != 0:**
1166. *#Check if the level is a multiple of 5 and if not then check all keys have been collected and the chest unlocked*
1167. **if** game.player.gamekeys >= ((2\*(game.level+1)) + 1):
1168. **if** game.chestunlocked[game.level] == True:
1169. self.kill()
1171. **else**:
1172. *#If the level is a multiple of 5 and the boss has been defeated then open the doors.*
1173. **if** len(game.boss\_group) == 0:
1174. **if** game.chestunlocked[game.level] == True:
1175. **self.kill()**
1177. *#Spike class which covers a full tile and inflicts damage to the player when active*
1178. **class** Spikes(pygame.sprite.Sprite):
1179. **def** \_\_init\_\_(self, color, width, height, x, y, posx, posy):
1180. **super().\_\_init\_\_()**
1181. *#Use sprite constructor*
1182. self.image = pygame.image.load(os.path.join(image\_path, "spike down.png"))
1183. *#self.image.fill(color)*
1184. self.rect = self.image.get\_rect()
1185. ***#set the coordinates***
1186. self.rect.x = x
1187. self.rect.y = y
1188. *#Set timers to change every 4 seconds between active and inactive states*
1189. self.previousattacktime = pygame.time.get\_ticks()
1190. **self.color = color**
1191. self.previousdamagetime = pygame.time.get\_ticks()
1193. *#end procedure*
1195. ***#Update function***
1196. **def** update(self):
1197. *#Check if the time is greater than 4 seconds before changing state*
1198. self.currentattacktime = pygame.time.get\_ticks()
1199. **if** self.currentattacktime - self.previousattacktime > 4000:
1200. **if self.color == GREY:**
1201. self.color = LIGHTBLUE
1202. self.image = pygame.image.load(os.path.join(image\_path, "spike up.png"))
1203. *#reset the timer*
1204. self.previousattacktime = self.currentattacktime
1205. **else:**
1206. *#If they are already active after 4 seconds then switch back to inactive*
1207. self.color = GREY
1208. self.image = pygame.image.load(os.path.join(image\_path, "spike down.png"))
1209. *#reset the timer*
1210. **self.previousattacktime = self.currentattacktime**
1211. *#If the spikes are active then check for player collision*
1212. **if** self.color == LIGHTBLUE:
1213. player\_hit\_group = pygame.sprite.spritecollide(self, game.player\_group, False, False)
1214. **for** game.player **in** player\_hit\_group:
1215. ***#If player collision is true then inflict 5 damage every 4 seconds (this limits the spike to only attack the player once when activated)***
1216. self.currentdamagetime = pygame.time.get\_ticks()
1217. **if** self.currentdamagetime - self.previousdamagetime > 4000:
1218. game.player.getdamage(5)
1219. *#reset attack timer*
1220. **self.previousdamagetime = self.currentdamagetime**

1223. *#Chest class used to unlock player's abilities at the end of each level*
1224. **class** Chest(pygame.sprite.Sprite):
1225. **def \_\_init\_\_(self,color, width, height, x, y, level):**
1226. super().\_\_init\_\_()
1227. self.image = Game.wallspritesheet.get\_image(224,0,32,32,40,40)
1228. self.image.set\_colorkey(GREEN)
1229. *#set the position of the sprite*
1230. **self.rect = self.image.get\_rect()**
1231. self.rect.x = x
1232. self.rect.y = y
1233. self.level = level
1235. ***#Update function used to check for player collision***
1236. **def** update(self):
1237. *#Check what the level is so when the chest collides with the player then unlock an ability for each level.*
1238. chest\_hit\_group = pygame.sprite.groupcollide(game.chest\_group, game.player\_group, False, False)
1239. **for** game.chest **in** chest\_hit\_group:
1240. **if self.level == 0:**
1241. game.player.canshoot = True
1243. **if** self.level == 1:
1244. game.player.swordradius = 70
1246. **if** self.level == 2:
1247. game.player.maximum\_health = 200
1249. **if** self.level == 3:
1250. **game.player.multishot = True**
1251. *#When the player collides then remove the chest and set the chest unlocked to true for that level.*
1252. game.chestunlocked[game.level] = True
1253. self.kill()
1255. ***#Melee Enemy class***
1256. **class** MeleeEnemy(pygame.sprite.Sprite):
1257. **def** \_\_init\_\_(self, direction, width, height, x, y, health):
1258. *#call sprite constructor*
1259. super().\_\_init\_\_()
1260. ***#create a sprite***
1261. self.image = Game.menemyspritesheet.get\_image(19,21,43,55,40,40)
1262. self.image.set\_colorkey(GREEN)
1263. *#set the position of the sprite*
1264. self.rect = self.image.get\_rect()
1265. **self.rect.x = x**
1266. self.rect.y = y
1267. self.health = health
1268. self.direction = direction
1269. self.speed\_x = 0
1270. **self.speed\_y = 0**
1271. *#self.goal = vec(13,3)*
1272. *#self.start = vec(self.rect.x, self.rect.y)*
1273. *#Set timers to attack the player and set the movement speed to 3 pixels per frame*
1274. *#self.previouspathtime = pygame.time.get\_ticks()*
1275. **self.movex = 1**
1276. self.movey = 1
1277. self.previousdamagetime = pygame.time.get\_ticks()
1278. self.previousattacktime = pygame.time.get\_ticks()
1279. *#end procedure*
1281. *#Method to add the speed to the object to move up, down and side to side*
1282. **def** changespeed(self,x,y):
1283. self.speed\_x += x
1284. self.speed\_y += y

1287. *#update function handles collisions and changes behaviour depending on the range of the enemy*
1288. **def** update(self):
1290. ***#Check if the player is 300 pixels away from the enemy***
1291. **if** self.is\_close() == True:
1292. *#if this is true then use the movetoplayer method to make the enemy move towards the player.*
1293. self.movetoplayer(game.player)
1294. *#If False then use the move method instead which makes the enmey bounce off walls.*
1295. **if self.is\_close() == False:**
1296. self.MOVE()
1297. wallcollision = pygame.sprite.spritecollide(self, game.wall\_group, False)
1298. **if** wallcollision:
1299. *#if there is a collision then reflect the speeds*
1300. **self.movex = self.movex \* -1**
1301. self.movey = self.movey \* -1
1303. *#if the enemy collides with a bullet then subtract 20 health from the enemy.*
1304. enemybullet\_hit\_group = pygame.sprite.groupcollide(game.enemy\_group, game.bullet\_group, False, True)
1305. **for self in enemybullet\_hit\_group:**
1306. self.health -= 20
1307. *#print(self.health)*
1308. *#Check if the health is greater than zero. Else it will create an instance of a key and increase the game score by 100 before removing itself.*
1309. **if** self.health < 1:
1310. **game.score += 100**
1311. gamekey = Key(PINK, self.rect.x + 2, self.rect.y + 9)
1312. game.all\_sprites\_group.add(gamekey)
1313. game.key\_group.add(gamekey)
1314. self.kill()
1315. ***#check for a collision with the sword and only allow the sword to inflict 20 damage per second on the enemy.***
1316. enemysword\_hit\_group = pygame.sprite.groupcollide(game.enemy\_group, game.sword\_group, False, False)
1317. **for** self **in** enemysword\_hit\_group:
1318. self.currentdamagetime = pygame.time.get\_ticks()
1319. **if** self.currentdamagetime - self.previousdamagetime > 1000:
1320. **self.health -= 20**
1321. self.previousdamagetime = self.currentdamagetime
1323. *#Check if the health is greater than zero. Else it will create an instance of a key and increase the game score by 100 before removing itself.*
1324. **if** self.health < 1:
1325. **game.score += 100**
1326. gamekey = Key(PINK, self.rect.x + 2, self.rect.y + 9)
1327. game.all\_sprites\_group.add(gamekey)
1328. game.key\_group.add(gamekey)
1329. self.kill()
1330. ***#endif***
1331. *#next*
1333. *#Check for collision with player.*
1334. player\_hit\_group = pygame.sprite.groupcollide(game.player\_group, game.menemy\_group, False, False)
1335. **for game.player in player\_hit\_group:**
1336. *#if the player collides with the enemy then subtract 10 health from the player every 2 seconds*
1337. self.currentattacktime = pygame.time.get\_ticks()
1338. **if** self.currentattacktime - self.previousattacktime > 2000:
1339. game.player.getdamage(10)
1340. **self.previousattacktime = self.currentattacktime**
1342. *#self.currentpathtime = pygame.time.get\_ticks()*
1343. *#if self.currentpathtime - self.previouspathtime > 500:*
1344. *#self.g = SquareGrid(GRIDWIDTH, GRIDHEIGHT)*
1345. ***#self.goal = vec((game.player.rect.x/40),(game.player.rect.y/40))***
1346. *#self.start = vec(self.rect.x, self.rect.y)*
1347. *#print(self.start)*
1348. *#self.path = breadth\_first\_search(g,self.goal, self.start)*
1349. *#self.previouspathtime = self.currentpathtime*
1350. ***#pass***


1354. *#Method to find vector position of the enemy on the grid*
1355. **def getpos(self):**
1356. **return** vec(self.rect.x/40, self.rect.y/40)
1358. *#Method used to control how the enemy moves when the player is not in range*
1359. **def** MOVE(self):
1360. ***#Using a random number between 1 and 10 and check if it divisible by 2. this provides a 50 50 chance of the enemy bouncing up and down or side to side.***
1361. **if** self.direction % 2 == 0:
1362. self.movey = 0
1363. self.rect.x += self.movex
1364. *#If divisible by 2 then move to the right by 3 pixels.*
1365. **else:**
1366. self.movex = 0
1367. self.rect.y += self.movey
1369. *#Method to change how the enemy moves when the player is within range of the enemy*
1370. **def movetoplayer(self, Player):**
1371. *#Compare the player's position to the enemy's position and change the speed of the enemy to move it closer to the player.*
1372. **if** Player.rect.x > self.rect.x:
1373. self.speed\_x = 1
1374. **if** Player.rect.x < self.rect.x:
1375. **self.speed\_x = -1**
1376. **if** Player.rect.y > self.rect.y:
1377. self.speed\_y = 1
1378. **if** Player.rect.y < self.rect.y:
1379. self.speed\_y = -1
1380. **if Player.rect.x == self.rect.x:**
1381. self.speed\_x = 0
1382. **if** Player.rect.y == self.rect.y:
1383. self.speed\_y = 0
1385. ***# Move along x axis***
1386. self.rect.x += self.speed\_x
1388. *# Did enemy hit a wall*
1389. block\_hit\_list = pygame.sprite.spritecollide(self, game.wall\_group, False) *# false so it doesn't remove the wall, true would*
1390. **for wall in block\_hit\_list:**
1391. *# If moving right, place enemy to the left side of wall*
1393. **if** self.speed\_x > 0:
1394. self.rect.right = wall.rect.left
1396. **else**:
1397. *# if moving left, do the opposite.*
1398. self.rect.left = wall.rect.right


1402. *# Move along y axis*
1403. self.rect.y += self.speed\_y
1405. ***# Did enemy hit a wall***
1406. block\_hit\_list = pygame.sprite.spritecollide(self, game.wall\_group, False)
1407. **for** wall **in** block\_hit\_list:
1408. *# Do same as above but on the y axis*
1409. **if** self.speed\_y > 0:
1410. **self.rect.bottom = wall.rect.top**
1412. **else**:
1413. self.rect.top = wall.rect.bottom

1416. *#Method using trigonometry to check the distance between the player and the enemy and outputs whether the player is within 300 pixels or not.*
1417. **def** is\_close(self):
1418. lengthx = self.rect.x - game.player.rect.x
1419. lengthy = self.rect.y - game.player.rect.y
1420. **distance = math.sqrt((lengthx \*\* 2) + (lengthy \*\* 2))**
1421. **if** distance < 300:
1422. **return** True
1423. **else**:
1424. **return** False
1426. *#method used to get the enemy's health*
1427. **def** gethealth(self):
1428. **return** self.health
1429. *#endprocedure*
1431. *#method to set the enemies health*
1432. **def** sethealth(self, newhealth):
1433. self.health = newhealth
1434. *#endfunction*
1436. *#Bow enemy class*
1437. **class** BowEnemy(pygame.sprite.Sprite):
1438. **def** \_\_init\_\_(self, direction, width, height, x, y, health):
1439. *#call sprite constructor*
1440. **super().\_\_init\_\_()**
1441. *#create a sprite*
1442. self.image = Game.benemyspritesheet.get\_image(19,21,43,55,40,40)
1443. self.image.set\_colorkey(PERFECTBLUE)
1444. *#set the position of the sprite*
1445. **self.rect = self.image.get\_rect()**
1446. self.rect.x = x
1447. self.rect.y = y
1448. self.health = health
1449. self.direction = direction
1450. **self.previousdamagetime = pygame.time.get\_ticks()**
1451. self.previousattacktime = pygame.time.get\_ticks()
1452. *#self.angle = 180*
1453. *#end procedure*
1455. ***#Update function used to check if the player is in range and calculate the speeds for the bullet fired***
1456. **def** update(self):
1458. **if** self.is\_close() == True:
1459. *#If player in range then calculate y distance and x distance as well as distance*
1460. **xdiff = (game.player.rect.x) - (self.rect.x)**
1461. ydiff = (game.player.rect.y) - (self.rect.y)
1462. self.angle = math.atan2(ydiff,xdiff)
1463. xspeed = 3 \* math.cos(self.angle)
1464. yspeed = 3 \* math.sin(self.angle)
1465. ***#Every 1 second an enemy bullet object is created travelling in the direction using the xspeed and yspeed.***
1466. self.currentattacktime = pygame.time.get\_ticks()
1467. **if** self.currentattacktime - self.previousattacktime > 2000:
1468. game.ebullet = EnemyBullet(RED, xspeed, yspeed, self.rect.x+20, self.rect.y+20)
1469. game.all\_sprites\_group.add(game.ebullet)
1470. **game.enemybullet\_group.add(game.ebullet)**
1471. *#reset the timer*
1472. self.previousattacktime = self.currentattacktime
1474. *#Create a line for testing purposes to be used to check for line of sight. If line of sight is true then the enemy can shoot.*
1475. **line = pygame.draw.line(screen, RED, (game.player.rect.x,game.player.rect.y), (self.rect.x,self.rect.y))**


1479. *#if the enemy collides with a bullet then subtract 20 health from the enemy.*
1480. **enemybullet\_hit\_group = pygame.sprite.groupcollide(game.enemy\_group, game.bullet\_group, False, True)**
1481. **for** self **in** enemybullet\_hit\_group:
1482. self.health -= 20
1483. *#print(self.health)*
1485. ***#Check if the health is greater than zero. Else it will create an instance of a key and increase the game score by 100 before removing itself.***
1486. **if** self.health < 1:
1487. game.score += 100
1488. gamekey = Key(PINK, self.rect.x + 2, self.rect.y + 9)
1489. game.all\_sprites\_group.add(gamekey)
1490. **game.key\_group.add(gamekey)**
1491. self.kill()
1493. *#check for a collision with the sword and only allow the sword to inflict 20 damage per second on the enemy.*
1494. enemysword\_hit\_group = pygame.sprite.groupcollide(game.enemy\_group, game.sword\_group, False, False)
1495. **for self in enemysword\_hit\_group:**
1496. self.currentdamagetime = pygame.time.get\_ticks()
1497. **if** self.currentdamagetime - self.previousdamagetime > 1000:
1498. self.health -= 20
1499. self.previousdamagetime = self.currentdamagetime
1501. *#Check if the health is greater than zero. Else it will create an instance of a key and increase the game score by 100 before removing itself.*
1502. **if** self.health < 1:
1503. game.score += 100
1504. gamekey = Key(PINK, self.rect.x + 2, self.rect.y + 9)
1505. **game.all\_sprites\_group.add(gamekey)**
1506. game.key\_group.add(gamekey)
1507. self.kill()

1510. ***#def line(self):***
1511. *# pygame.draw.line(screen, RED, (game.player.rect.x,game.player.rect.y), (self.rect.x,self.rect.y))*
1513. *#end procedure*
1515. ***#Get enemy's health***
1516. **def** gethealth(self):
1517. **return** self.health
1518. *#endprocedure*
1520. ***#Set enemy's health***
1521. **def** sethealth(self, newhealth):
1522. self.health = newhealth
1523. *#endfunction*
1525. ***#Method using trigonometry to check the distance between the player and the enemy and outputs whether the player is within 400 pixels or not.***
1526. **def** is\_close(self):
1527. *#Calculate x distance and y distance to player*
1528. lengthx = self.rect.x - game.player.rect.x
1529. lengthy = self.rect.y - game.player.rect.y
1530. ***#find total distance/ hypotenouse***
1531. distance = math.sqrt((lengthx \*\* 2) + (lengthy \*\* 2))
1532. *#If distance is less than 400 then return True*
1533. **if** distance < 400:
1534. **return** True
1535. **else:**
1536. **return** False

1539. *#Boss Enemy Class*
1540. **class BossEnemy(pygame.sprite.Sprite):**
1541. **def** \_\_init\_\_(self, direction, width, height, x, y, health):
1542. *#call sprite constructor*
1543. super().\_\_init\_\_()
1544. *#create a sprite*
1545. **self.image = Game.bossspritesheet.get\_image(23,21,36,53,width,height)**
1546. self.image.set\_colorkey(PERFECTBLUE)
1547. *#set the position of the sprite*
1548. self.rect = self.image.get\_rect()
1549. self.rect.x = x
1550. **self.rect.y = y**
1551. *#Define attributes for health to be used in boss health bar*
1552. self.current\_health = 999
1553. self.maximum\_health = health
1554. self.health\_bar\_length = 200
1555. **self.target\_health = 1000**
1556. self.health\_change\_speed = 2
1557. self.health\_bar\_color = GREEN
1558. self.health\_ratio = self.maximum\_health/ self.health\_bar\_length
1559. self.direction = direction
1560. ***#Set timers to be used for attacks.***
1561. self.previoushealthtime = pygame.time.get\_ticks()
1562. self.previousdamagetime = pygame.time.get\_ticks()
1563. self.previousattacktime = pygame.time.get\_ticks()

1566. *#method used to increase the health of the boss by an amount*
1567. **def** gethealth(self, amount):
1568. **if** self.target\_health < self.maximum\_health:
1569. self.target\_health += amount
1570. **if self.target\_health >= self.maximum\_health:**
1571. self.target\_health = self.maximum\_health
1572. *#endprocedure*
1574. *#Method used to remove health from the boss*
1575. **def getdamage(self,amount):**
1576. **if** self.target\_health > 0:
1577. self.target\_health -= amount
1578. **if** self.target\_health <=0:
1579. self.target\_health = 0
1581. *#Method used to create helath bar*
1582. **def** advanced\_health(self):
1583. transition\_width = 0
1584. transition\_color = RED
1586. *#When health is added set the transition bar to green and increase health bar at set speed*
1587. **if** self.current\_health < self.target\_health:
1588. self.current\_health += self.health\_change\_speed
1589. transition\_width = int((self.target\_health - self.current\_health)/ self.health\_ratio)
1590. **transition\_color = GREEN**
1591. *#When health is removed set the transition bar to yellow and decrease the health bar at a set speed*
1592. **if** self.current\_health > self.target\_health:
1593. self.current\_health -= self.health\_change\_speed
1594. transition\_width = int((self.target\_health - self.current\_health)/ self.health\_ratio)
1595. **transition\_color = YELLOW**
1597. *#Change the health bar colour depending on the amount of health*
1598. **if** self.current\_health >= 700:
1599. self.health\_bar\_color = GREEN
1600. **if self.current\_health >= 500 and self.current\_health < 700:**
1601. self.health\_bar\_color = ORANGE
1602. **if** self.current\_health < 300 **and** self.current\_health >=0:
1603. self.health\_bar\_color = RED
1604. *#set the health bar size equal to the current health divided by max health*
1605. **health\_bar\_width = int(self.current\_health/ self.health\_ratio)**
1606. health\_bar = pygame.Rect(self.rect.x-20,self.rect.y-40, health\_bar\_width, 25)
1607. *#transition bar is set to the right of the health bar*
1608. transition\_bar = pygame.Rect(health\_bar.right, self.rect.y-40, transition\_width, 25)
1609. **if** self.current\_health > 0:
1610. ***#Draw all 3 bars over each other***
1611. pygame.draw.rect(screen, self.health\_bar\_color, health\_bar)
1612. pygame.draw.rect(screen,transition\_color, transition\_bar)
1613. pygame.draw.rect(screen, WHITE, (self.rect.x-20, self.rect.y-40, self.health\_bar\_length, 25), 4)

1616. *#Update function to change behaviour of Boss depending on its health.*
1617. **def** update(self):
1619. *#Every 10 seconds the boss regains health*
1620. **self.health\_ratio = self.maximum\_health/ self.health\_bar\_length**
1621. self.currenthealthtime = pygame.time.get\_ticks()
1622. **if** self.currenthealthtime - self.previoushealthtime > 10000:
1623. self.gethealth(10)
1624. *#reset the timer*
1625. **self.previoushealthtime = self.currenthealthtime**
1627. *#Check if the health is zero then add to the game score and kill the boss*
1628. **if** self.current\_health < 1:
1629. game.score += 1000
1630. **self.kill()**
1632. *#When the health of the boss is greater than 60% then just move towards the player*
1633. **if** self.current\_health >= 700:
1634. *#move towards the player no matter the range*
1635. **self.movetoplayer(game.player)**
1636. *#When the health is lower than 60% but above 30% then move around and shoot at the player*
1637. **if** self.current\_health < 700 **and** self.current\_health >= 400:
1638. *#move towards the player and shoot projectiles*
1639. *#Calculate y distance and x distance as well as distance*
1640. **self.movetoplayer(game.player)**
1641. xdiff = (game.player.rect.x) - (self.rect.x+80)
1642. ydiff = (game.player.rect.y) - (self.rect.y+80)
1643. self.angle = math.atan2(ydiff,xdiff)
1644. xspeed = 3 \* math.cos(self.angle)
1645. **yspeed = 3 \* math.sin(self.angle)**
1646. *#Every 1 second an enemy bullet object is created travelling in the direction using the xspeed and yspeed.*
1647. self.currentattacktime = pygame.time.get\_ticks()
1648. **if** self.currentattacktime - self.previousattacktime > 1500:
1649. game.ebullet = EnemyBullet(RED, xspeed, yspeed, self.rect.x+80, self.rect.y+80)
1650. **game.all\_sprites\_group.add(game.ebullet)**
1651. game.enemybullet\_group.add(game.ebullet)
1652. self.previousattacktime = self.currentattacktime
1653. **if** self.current\_health < 400:
1654. *#If the boss is less than 30% then shoot 8 bullets travelling up, down, left, right and all diagonal directions*
1655. ***#shoot projectiles in all directions and spawn in enemies***
1656. self.movetoplayer(game.player)
1657. self.currentattacktime = pygame.time.get\_ticks()
1658. *#Repeat this every second*
1659. **if** self.currentattacktime - self.previousattacktime > 3500:
1660. **if len(game.enemybullet\_group) == 0:**
1661. game.ebullet = EnemyBullet(RED, 3, 0, self.rect.x+80, self.rect.y+80)
1662. game.all\_sprites\_group.add(game.ebullet)
1663. game.enemybullet\_group.add(game.ebullet)
1664. game.ebullet = EnemyBullet(RED, 3, 3, self.rect.x+80, self.rect.y+80)
1665. **game.all\_sprites\_group.add(game.ebullet)**
1666. game.enemybullet\_group.add(game.ebullet)
1667. game.ebullet = EnemyBullet(RED, 3, -3, self.rect.x+80, self.rect.y+80)
1668. game.all\_sprites\_group.add(game.ebullet)
1669. game.enemybullet\_group.add(game.ebullet)
1670. **game.ebullet = EnemyBullet(RED, -3, 0, self.rect.x+80, self.rect.y+80)**
1671. game.all\_sprites\_group.add(game.ebullet)
1672. game.enemybullet\_group.add(game.ebullet)
1673. game.ebullet = EnemyBullet(RED, -3, 3, self.rect.x+80, self.rect.y+80)
1674. game.all\_sprites\_group.add(game.ebullet)
1675. **game.enemybullet\_group.add(game.ebullet)**
1676. game.ebullet = EnemyBullet(RED, -3, -3, self.rect.x+80, self.rect.y+80)
1677. game.all\_sprites\_group.add(game.ebullet)
1678. game.enemybullet\_group.add(game.ebullet)
1679. game.ebullet = EnemyBullet(RED, 0, 3, self.rect.x+80, self.rect.y+80)
1680. **game.all\_sprites\_group.add(game.ebullet)**
1681. game.enemybullet\_group.add(game.ebullet)
1682. game.ebullet = EnemyBullet(RED, 0, -3, self.rect.x+80, self.rect.y+80)
1683. game.all\_sprites\_group.add(game.ebullet)
1684. game.enemybullet\_group.add(game.ebullet)
1685. **self.previousattacktime = self.currentattacktime**
1686. **while** game.bosscount != 10:
1687. *#Create 5 melee enemies that randomly spawn in the level*
1688. xpos = random.randint(1,23)
1689. ypos = random.randint(1,23)
1690. ***#Check if the random number has the same coordinates as a wall or spike and if not then create an enemy***
1691. **if** game.levels[game.level][xpos][ypos] !=1 **and** game.levels[game.level][xpos][ypos] != 2 **and** game.levels[game.level][xpos][ypos] != 7:
1692. game.menemy = MeleeEnemy(random.randint(0,10),40,40, xpos\*40, ypos\*40, 40)
1693. game.all\_sprites\_group.add(game.menemy)
1694. game.enemy\_group.add(game.menemy)
1695. **game.bosscount = game.bosscount +1**
1697. *#Check if the boss collides with a player bullet then subtract 5 health from the boss.*
1698. enemybullet\_hit\_group = pygame.sprite.groupcollide(game.boss\_group, game.bullet\_group, False, True)
1699. **for** self **in** enemybullet\_hit\_group:
1700. **self.getdamage(5)**
1702. *#Check for a collision with the sword and only allow the sword to inflict 20 damage per second on the boss.*
1703. enemysword\_hit\_group = pygame.sprite.groupcollide(game.boss\_group, game.sword\_group, False, False)
1704. **for** self **in** enemysword\_hit\_group:
1705. **self.currentdamagetime = pygame.time.get\_ticks()**
1706. **if** self.currentdamagetime - self.previousdamagetime > 1000:
1707. self.getdamage(20)
1708. self.previousdamagetime = self.currentdamagetime
1710. ***#Check for collision with player.***
1711. player\_hit\_group = pygame.sprite.groupcollide(game.player\_group, game.boss\_group, False, False)
1712. **for** game.player **in** player\_hit\_group:
1713. self.currentattacktime = pygame.time.get\_ticks()
1714. *#if the player collides with the boss then subtract 30 health from the player every 2 seconds*
1715. **if self.currentattacktime - self.previousattacktime > 1000:**
1716. game.player.getdamage(30)
1717. self.previousattacktime = self.currentattacktime


1721. *#Method used to make the boss move towards the player*
1722. **def** movetoplayer(self, Player):
1723. *#compare the player's position to the boss' position and change the speed of the boss to move it towards the player*
1724. **if** Player.rect.x > self.rect.x:
1725. **self.speed\_x = 1**
1726. **if** Player.rect.x < self.rect.x:
1727. self.speed\_x = -1
1728. **if** Player.rect.y > self.rect.y:
1729. self.speed\_y = 1
1730. **if Player.rect.y < self.rect.y:**
1731. self.speed\_y = -1
1732. **if** Player.rect.x == self.rect.x:
1733. self.speed\_x = 0
1734. **if** Player.rect.y == self.rect.y:
1735. **self.speed\_y = 0**
1737. *# Move along x axis*
1738. self.rect.x += self.speed\_x
1740. ***# Did enemy hit a wall***
1741. block\_hit\_list = pygame.sprite.spritecollide(self, game.wall\_group, False) *# false so it doesn't remove the wall, true would*
1742. **for** wall **in** block\_hit\_list:
1743. *# If moving right, place enemy to the left side of wall*
1745. **if self.speed\_x > 0:**
1746. self.rect.right = wall.rect.left
1748. **else**:
1749. *# if moving left, do the opposite.*
1750. **self.rect.left = wall.rect.right**


1754. *# Move along y axis*
1755. **self.rect.y += self.speed\_y**
1757. *# Did enemy hit a wall*
1758. block\_hit\_list = pygame.sprite.spritecollide(self, game.wall\_group, False)
1759. **for** wall **in** block\_hit\_list:
1760. ***# Do same as above but on the y axis***
1761. **if** self.speed\_y > 0:
1762. self.rect.bottom = wall.rect.top
1764. **else**:
1765. **self.rect.top = wall.rect.bottom**




1771. *#Create an instance of the Game class used for the game loop*
1772. game = Game()
1774. *#walls = [(10, 7), (11, 7), (12, 7), (13, 7), (14, 7), (15, 7), (16, 7), (7, 7), (6, 7), (5, 7), (5, 5), (5, 6), (1, 6), (2, 6), (3, 6), (5, 10), (5, 11), (5, 12), (5, 9), (5, 8), (12, 8), (12, 9), (12, 10), (12, 11), (15, 14), (15, 13), (15, 12), (15, 11), (15, 10), (17, 7), (18, 7), (21, 7), (21, 6), (21, 5), (21, 4), (21, 3), (22, 5), (23, 5), (24, 5), (25, 5), (18, 10), (20, 10), (19, 10), (21, 10), (22, 10), (23, 10), (14, 4), (14, 5), (14, 6), (14, 0), (14, 1), (9, 2), (9, 1), (7, 3), (8, 3), (10, 3), (9, 3), (11, 3), (2, 5), (2, 4), (2, 3), (2, 2), (2, 0), (2, 1), (0, 11), (1, 11), (2, 11), (21, 2), (20, 11), (20, 12), (23, 13), (23, 14), (24, 10), (25, 10), (6, 12), (7, 12), (10, 12), (11, 12), (12, 12), (5, 3), (6, 3), (5, 4)]*
1775. ***#for wall in walls:***
1776. *#g.walls.append(vec(wall))*
1777. *#goal = vec(15, 8)*
1778. *#start = vec(20,1)*
1779. *#path = breadth\_first\_search(g, goal, start)*

1782. *# -------- Main Program Loop -----------*
1783. **while** **not** done:
1784. *# --- Main event loop*
1785. ***#Declare done = True when the game\_over is true returning the player to the Menu***
1786. done = game.eventprocess()
1788. *# --- Game logic should go here*
1789. *#Runs all the logic for the game loop*
1790. **game.runlogic()**
1792. *#draw the screen*
1793. *#Draws everything for the game and updates the screen*
1794. game.display(screen)
1796. *# --- Limit to 60 frames per second*
1797. clock.tick(60)
1799. *#Initiate the Menu function to start with the menu*
1800. **game\_intro()**
1801. *# Close the window and quits.*
1802. pygame.quit()