­­A Level Project

Name: **Maxim Ladoshin**

Centre:

Candidate Number:

Contents

[Analysis 3](#_Toc58567207)

[Introduction 3](#_Toc58567208)

[Computational Suitability 3](#_Toc58567209)

[Research 3](#_Toc58567210)

[Stakeholders 3](#_Toc58567211)

[Interviews 3](#_Toc58567212)

[Interview 1: 3](#_Toc58567213)

[Interview 2 3](#_Toc58567214)

[Interview Conclusion 3](#_Toc58567215)

[Requirements 3](#_Toc58567216)

[Hardware Requirements 4](#_Toc58567217)

[Software Requirements 4](#_Toc58567218)

[Design 5](#_Toc58567219)

[Interface 5](#_Toc58567220)

[Variables 5](#_Toc58567221)

[Functionality 5](#_Toc58567222)

[Test Plan 5](#_Toc58567223)

[Implementation 6](#_Toc58567224)

[Iteration 1 6](#_Toc58567225)

[Requirements being developed 6](#_Toc58567226)

[Errors 6](#_Toc58567227)

[Conclusion 6](#_Toc58567228)

[Iteration 2 6](#_Toc58567229)

[Requirements being developed 6](#_Toc58567230)

[Errors 6](#_Toc58567231)

[Conclusion 6](#_Toc58567232)

[Iteration 3 6](#_Toc58567233)

[Requirements being developed 6](#_Toc58567234)

[Errors 6](#_Toc58567235)

[Conclusion 6](#_Toc58567236)

[Testing 7](#_Toc58567237)

[Evaluation 8](#_Toc58567238)

# Analysis

## Introduction

Nowadays the game industry becomes more and more popular amongst people of all ages. Computer games have been historically the type of entertainment when people can forget about all their problems in real life and dive into the game reality. For some people games have become a process of relaxation after a hard day, or just a sort of funny entertainment when they are bored (e.g. while travelling in a bus, train etc). Moreover, some games can develop person’s mind, improve his reaction and cognitive skills which is sometimes quiet useful for small children. Another games are used to educate children and teach the everything about the world (Animal Jam – zoology, Endless Alphabet – alphabet, Starfall Kids Games). There is an enormous amount of various types of games accessible today, which can educate you, develop some of your skills or just entertain you.

Rapid technology changes in mobile devices have revealed the whole new mobile gaming platform, which now is gaining more and more new players due to its convenience and cheap price. Almost every person in the world have access to a mobile phone, so mobile games are accessible to an extremely wide audience.

Most of the popular games in 21 century try to reach the maximum possible realism, using quiet a lot of hardware resources. Every year there is a new graphic technology invented, which improves the games’ graphic such as 3D, Ray Tracing, DLSS and so on and so forth. All this technologies make the games really close to the real life. However, less and less people can afford to play this games each year because such AAA games require an expensive hardware (personal computers or laptops). Such high-end games as Cyberpunk, GTA 5, Red Dead Redemption 2 are one of the most graphic demanding games out now (in 2021). Moreover, the development of such high-end games is quiet costly, which make their final price in the market higher. Therefore, the gaming industry tends to be more and more expensive for the players.

Besides that, the other problem of modern games is that they require the internet connection which is not always the case. For example, if you have a long 10 hour flight and you are very bored, you can’t play GTA 5, Fortnite or any other multiplayer game. So, the only choice is to play arcade games or any other game which doesn’t require an internet connection.

I personally like RPG and strategy games such as Age of Empire, GTA 2 and Terraria. These games are relatively simple and straight forward in terms of graphics as they are all 2D games. Even though, I enjoyed these games so much in childhood. These games are extremely profound and exciting to play.

That is why I decided to develop a game which will combine some elements from arcade 2D games, shooter games and RPG games. My game will also not require any internet connection, so it can be played anywhere. In addition, my game will be suitable for low-end devices, so more stakeholders could appreciate it. The primary input devices will be the keyboard and a mouse/touchpad.

## Computational Suitability

The problem I will be solving will require to use computer calculations such as checking for collisions, player’s health, enemies’ health and so on. The game I will develop is an RPG game with the elements of a shooter, so it obviously cannot be played in real life. This game would provide a virtual world where you will be battling against enemies, searching for new Weapons, armor, medicine kits and so forth.

Moreover, the computational power of a device allows me to implement enemy AI, so the enemies won’t just rest at one place, but try to attack the player. The enemy AI will make a game more realistic and more fun. It will be more difficult to survive in a game where all the enemies are trying to terminate you meaning the user will have to come up with various strategies to cope with enemies (such as hiding, running away, killing them etc). Finally, the implementation of a simple AI will make the enemies similar to real life players, which will definitely make a game more exciting to play.

Trying to solve such a problem in a real-life world is impossible as you obviously have a virtual map with obstacles and randomly located enemies. The number of enemies will increase over time as well as the player’s score and number of kills.

* Encapsulation may help me to develop a code which will be more secure, and players will have to try considerably harder to cheat during the game. Furthermore, I could use encapsulation to encourage other programmers or users to modify the source code and add new feature. In this case, encapsulation will make sure that class properties are not accessed or changed in the way they should not be. This will prevent program crashes and make the development process clearer and pleasing.
* Abstraction will help me to develop my game without huge troubles by focusing on core functionality of my game and excluding all the unessential details which will only make the code longer and more complicated. In my game I will use pygame library which is already pre-build to help python programmers build games and significantly reduce the amount of code. The programmer does not really have to know how the pygame methods work to create a game. he just only needs to know how to use these methods.
* Data Visualization

My game will use different data such as score, player’s health, screen size constant, color constants, player’s inventory list, and so on. The GUI (graphic user interface) will be rendered according to all this data and it will allow the stakeholders to better navigate and control inside the game, which will make my game user-friendly.

## Research

I was inspired to develop my own 2D RPG game by such games as “Enter The Gungeon”, “Nuclear Throne”, and “Grand Theft Auto 2”. All these games have a decent UI which is very comfortable and easy to use.

* **Enter the Gungeon**

Enter the Gungeon is a single/cooperation 2D RPG shooter game. Player should kill the enemies and complete the levels to reach the final prize – the weapon that can demolish the past. This game includes different types of weapons, characters, enemies, NPCs and so on. The user should look for better ammunition that can be found in different places as a loot. The final mission is to kill the main boss and the player will be awarded a main prize as he has done that successfully.

A picture containing text, indoor

Description automatically generated

Figure 1 - Enter the Gungeon - game screenshot

**A screen shot of a video game

Description automatically generated with medium confidence**

Figure 2 - "Enter The Gungeon" - game screenshot

**I was inspired by this game because:**

* Enter the Gungeon has a great UI design.
* The game has lots of content with different bosses, enemies, NPCs and other cool stuff such as various weapons, spells, types of armor etc.
* The game has nice animations and is smooth.
* It reminds me of retro 2d arcades, though it is a modern and refined version of old games.
* **Nuclear Throne**

Nuclear Throne is a top-down shooter game which is very dynamic and intensive. There is so many things going on in the game, so you become a bit lost in the beginning but get used to it a bit later. The game mimics retro 2D arcades, although it has improved design and looks quite nice.

Map

Description automatically generatedThe player in “Nuclear Throne” should always be moving and cope with all types of enemies, such as mice, monsters, mutants and bosses. The game also includes elements of an RPG such as looting, character customization and dialogs with NPCs which can help you progress faster in the virtual world.

Figure 3 - "Nuclear Throne" - game screenshot

I was inspired by this game because it has character customization, in-game progression, different difficulty levels and advanced enemy AI which makes this game more realistic and fun to play. I would like to add some of game’s features into my own game. For instance, combat system, looting systems, some UI elements (health bar, menus)

Map

Description automatically generated

Figure 4 - "Nuclear Throne" - game screenshot

## Grand Theft Auto 2 (GTA 2)

Almost everyone knows about GTA games. These are the open-world games, where the player have minimum limitations. This game is a virtual world where you can do whatever you want. This fact seems quite appealing to me. GTA series started with top down 2D games and the became one of the best 3D AAA games (GTA 5). Even though this game was launched in 1991, it still astonishes me and I enjoy spending some time playing this legendary game.

The things I like about GTA 2 are:

* The game provide an open world with NPCs and enemies
* The player has lots of choice
* The game includes elements from RPG, shooter and vehicle driving games

A picture containing text, control panel

Description automatically generated

Figure 5 - "Grand Theft Auto 2" - game screenshot

Grand Theft Auto 2 had one of the best graphics at that time in 1999 and I quiet like it. That’s why it would be ideal if I could implement similar design in my game. Moreover, I enjoy this game because it has lots of freedom for a player – players can make and spend money, drive different vehicles, shoot enemies, run away from the police, complete missions and so many more. This wide variety of options makes the game more interesting and satisfying to play.

## Stakeholders

My game is going to be a simple indie arcade which will provide the player with a wide variety of paths to progress in game (such as get better weapons, armor). So, my stakeholders will be people who like playing arcade games, top-down shooters, or retro 2D RPG games. Moreover, most of the people who doesn’t have a stable internet connection would appreciate my game, because it won’t require any internet to play it. Besides that, the owners of low-end devices can benefit from my game because it will suit their low performance devices and they will be able to play it without lags.

Finally, the students in the boarding house will appreciate this game because sometimes the internet in the boarding house goes down and these guys don’t know how to entertain themselves.

However the target audience for 2D retro game is not wide, I could add some modern features in it such as refined and new-looking graphics and textures, better ambient sound etc. Besides that, it will make my game to stand out from other arcades if I distribute my game for free. This implies, more people will afford it and play. Even If they don’t like it, they won’t regret because they haven’t paid a penny for it. I guess this game won’t suit most of the players who only values astonishing 3D graphics in games and plays only AAA games such as GTA5, Cyberpunk and Call Of Duty.

Who can help you design the game?

To help me with the game’s design I can ask my friends from a boarding house. Some of them have designed games before, so they can give me a helpful device. Moreover, I could ask computer science teachers to help me with game level design and maybe UI design. Finally, my family may help me with designing my game. They could choose the best design for in-game elements (such as textures and objects)

# Interviews

I have surveyed several friends of mine in the boarding house and my parents.

I have collected some useful responses that helped me design the game and its structure to suit the most of my stakeholders.

I decided to survey 3 people with different interests and different ages to get unbiased results.

I have interviewed:

1. Shiwei Hangs (16 years old) – he is a boarder student in my school and he is passionate about physics. He loves computer games, however he is interested in mobile games (such as Subway Surfers and Crossy Road) and AAA games (such as Assasin’s Creed, Cyberpunk)
2. Martin Kozon (19 years old) – hard-working student who is interested in arts and design. Doesn’t mind playing some strategies on his laptop sometimes: Age of Empires
3. Makar Ladoshin (10 years old) – he likes playing games on his tablet. He also plays computer games such as GTA5 and Minecraft on the weekends.

## Interview Questions

Q1: Do you play computer games?

Q2: How often do you play them?

Q3: What is your main gaming device?

Q4: Which types of games do you like the most?

Q5: What are your favorite games?

Q6: Would you try playing 2D RPG shooter game?

Q7: When and where would you most likely play it?

Q8: What sorts of weapons and battle techniques would you like to see in the game?

Q9: Do you want to pick up random loot and to look for rare items in a game?

## Interview 1 (Shiwei Hangs):

Q1: Do you play computer games?

“Yes, I do”

Q2: How often do you play them?

“I usually play in the evenings when I have some free time after school, or on the weekends when I have a lot of spare time. I spend about 2 hours in average playing games a day.”

Q3: What is your main gaming device?

“Laptop and phone”

Q4: Which types of games do you like the most?

“AAA games, Shooters and mobile arcades”

Q5: What are your favorite games?

“Far cry 4, Grand Theft Auto 5”

Q6: Would you try playing 2D RPG shooter game?

“I won’t mind trying”

Q7: When and where would you most likely play it?

“I would most likely play it when I don’t have internet connection, while travelling for example.

Q8: What sorts of weapons and battle techniques would you like to see in the game?

“Perhaps, it will be cool if you can add firearms and some knifes into your game”

Q9: Do you want to pick up random loot and to look for rare items in a game?

“Yes, definitely”

## Interview 2 (Martin Kozon):

Q1: Do you play computer games?

“Yes, I sometimes do”

Q2: How often do you play them?

“I don’t have time to play games during the week, so I usually game during weekends about 2 hour every weekend”

Q3: What is your main gaming device?

“Laptop”

Q4: Which types of games do you like the most?

“Strategy games”

Q5: What are your favorite games?

“Age of Empires, HearthStone”

Q6: Would you try playing 2D RPG shooter game?

“I will try if there are some strategy elements or it is just fun to play.”

Q7: When and where would you most likely play it?

“If I like it I can play it on weekends to relax and have a great time.”

Q8: What sorts of weapons and battle techniques would you like to see in the game?

“I don’t really mind. It can be whatever.”

Q9: Do you want to pick up random loot and to look for rare items in a game?

“Yes, why not. This may make a game more interesting to play.”

## Interview 1 (Makar Ladoshin):

Q1: Do you play computer games?

“Yes”

Q2: How often do you play them?

“I play games every day after school or in the evening to rest after a school day.”

Q3: What is your main gaming device?

“Ipad”

Q4: Which types of games do you like the most?

“RPG shooters, strategy games”

Q5: What are your favorite games?

“Brawl Stars, Clash Royal, Minecraft”

Q6: Would you try playing 2D RPG shooter game?

“I really enjoy playing Brawl Stars, so I am excited to try new 2D RPG games.”

Q7: When and where would you most likely play it?

“I can play it even instead of Brawl Stars and play about 1 hour every evening”

Q8: What sorts of weapons and battle techniques would you like to see in the game?

“I would enjoy various guns and missiles, maybe some sniper rifles. Also, it will be great if there are some different combat styles: hiding, rushing or sneaking”

Q9: Do you want to pick up random loot and to look for rare items in a game?

“Yes, I guess it will be amazing!”

# Interview Conclusion

The interview has given me essential information about how my future game will look like. I will try to reasonably satisfy most demands of my stakeholders in my game.

* most of my stakeholders play computer games quiet often and have access to laptops or mobile devices
* All my stakeholders are willing to try out my game, which is quiet good
* Some of my stakeholders stated they enjoy shooters, others mentioned they like strategies. This means my game should be a combination of shooter and strategy game. So, I could add some elements and techniques from shooter games – shooting, weapons, bullets, inventory etc. And I could add some elements of a strategy game – various paths to kill enemies and get better loot or even add some missions.
* Most of my stakeholders said they would use the game during rest time after school, so I should make the game simple and the UI – user-friendly. If the UI is simple to use, stakeholders won’t be distracted from the gaming process.
* Most of my stakeholders have already played games on a pc, so I can easily use keyboard and a mouse as a primary control devices for this game.

All in all, according to the results I have received, it is reasonable to make a 2D top-down shooter with elements of RPG game. Moreover, the player should be able to pick up the loot and use different types of weapons to battle the enemies. This type of game would be relatively popular among people that I have interviewed.

# Requirements (Success Criteria)

List the requirements of you game.

Must have:

1. Screen size = 800x600 pixels

*The screen size may be changed in the code (change the constant)*

1. Destructible and non-destructible walls

*The player and enemies can destruct the walls and shoot through the holes in walls. This will make a game more realistic and fun to play.*

1. Player sprite

*Use sprite for easier collision detection with bullets, wall bricks and enemies.*

1. Enemies with some sort of AI (chasing the player, shooting etc.). Most of the modern games ave some sort of AI, so the game is interesting to play.
2. Player must move in 4 directions.

*Up, Down, Left, and right. Moreover, the user can press the different keys simultaneously and move diagonally. This is the most suitable type of movement for this game.*

1. Player can shoot.

*Player can click the right mouse button and release the bullets from the weapon. As I am developing a shooter RPG game, the shooting is essential for player.*

1. Player can pick up the loot.

*Player will be able to collect the loot and store the items of the loot in the inventory. My game is an RPG game, so it should have a looting system.*

1. Player can die.

*If the health of a player is 0 or below, he dies. This will stop the game, so the user can restart the game.*

1. Loser and Winner window
2. Auto generation of loot on the map

*The loot will be randomly generated, so it will be a bit more interesting.*

1. Enemy might have random loot, which the player can get after killing him.
2. Inventory can be full.

*If the weight of the items in the inventory at the limit, the inventory is going to be full, so the player will not be able to pick up the loot. This will make the game more realistic and a bit harder to play.*

1. Adding the items to inventory
2. Displaying the weight of the items in the inventory
3. Player can heal using the medicine kits.
4. Player can use the armor.
5. Armor adds the armor points, which can be drawn after getting the damage from the enemies.
6. 3 types of medicine kits: big (restores 50% of health), medium (restores 20% of health), small (restores 10% of health)
7. 3 types of armor: heavy, medium, light
8. More items the player has in his inventory, the slower he moves.
9. Heavy and medium armor can decrease the player’s speed.
10. Player can pick up the weapons.
11. Player can pick up the bullets.
12. Weapons cannot shoot if there are no bullets.
13. 3 types of enemies:
14. 3 Bosses:
15. First boss with a bow (easy)
16. Second Boss with a sniper rifle (medium)
17. Third boss with a rifle and missiles (hard)
18. Player can select the weapon using buttons 1, 2, 3
19. Player can use other items from inventory using buttons R, T, Y

**Should have:**

1. Close distance combat enemies (armed with knives, katanas, axes)
2. Medium distance combat enemies (armed with pistols, bows or rifles)
3. Far distance combat enemies (armed with sniper rifles)
4. Enemies could shoot the player.

**Could have:**

1. Player can drop the weapon.
2. Player can drop the items in inventory.
3. Player can use spells to increase his speed.
4. The camera follows the player.

**Won’t have:**

1. Internet connection
2. 3D

# Hardware Requirements

1. Screen with minimum size of 800x600 px
2. Keyboard
3. Mouse or touchpad
4. Free space on disk
5. Dual core processor

# Software Requirements

1. Operating system (Windows, Linux, Mac os)
2. Python (version 3.7.6 or later)
3. PyGame library (version 1.9.3 or later)

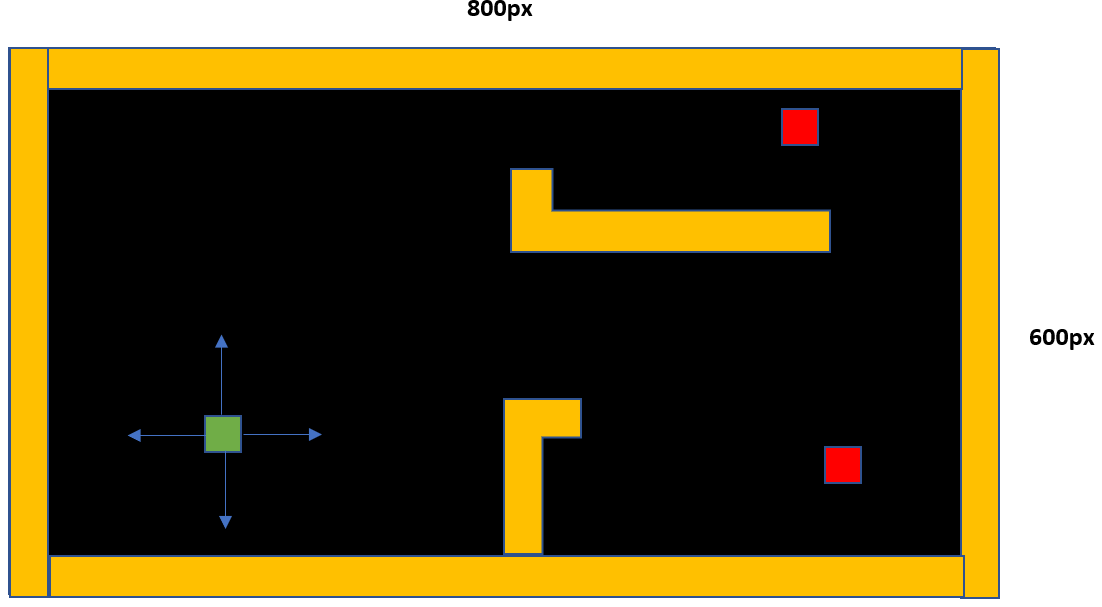
# Design

## Interface

Hand drawn diagrams of what the game looks like. Label the important aspects.

Basic requirements:

1. 800x600px window
2. Player moves in all 4 directions (up, down, left, and right)
3. The player moves when the keys WASD are pressed
4. The player can boost their speed if the shift key is pressed
5. The player can shoot if the left mouse button is clicked
6. The player can select weapons using keys 1, 2 or 3.

Interface design:

*Player can move in four directions up, down right, and left.*

A picture containing diagram

Description automatically generated

A picture containing graphical user interface

Description automatically generated**HealthBars:**

A picture containing diagram

Description automatically generatedInventory Design:

## Variables and Data Structures

List all classes and methods clearly.

Inheritance diagram.

The game generates the map, consisting of blocks 16x16 pixels. The player is a class with its own functionality – methods. There is also an enemy class with its unique methods. Enemy class and player class are both children of a parent class Person which has basic methods like move, shoot etc.

**Global Variables:**

Colors:

BLACK = (0,0, 0)

WHITE = (255, 255,255)

BLUE = (50, 50, 255)

YELLOW = (255, 255, 0)

GREEN = (50, 255, 50)

RED = (255, 0, 0)

Loot types:

LOOT\_TYPES = ["weapon", "bullets", "paramedics", "armour"]

Weapon types:

WEAPON\_TYPES = ["glock", "ak47", "shotgun"]

Bullet types:

BULLET\_TYPES = ["pistols", "rifles", "shotguns"]

Armour types:

ARMOUR\_TYPES = ["heavy", "medium", "light"]

Medicine kit types:

PARAMEDIC\_TYPES = ["heavy", "medium", "light"]

Screen size:

size = (1000, 1000)

Data design:

1. Base class for all both enemies and player
2. Class for a user’s player
3. Class for an enemy

Graphical user interface

Description automatically generated with medium confidence**class diagram for players and enemies:**

**Description:**

People class is the base class for Player class and Enemy class. People classs will contain basic methods (such as move, heal, checkCollisions etc.) and attributes such as position, speed, health and so on. The Inheritance will allow to reuse much of the code and in effect reduce the amount of code.

**Justification:**

This inheritance structure is best suited for my case because both player and enemies share some same properties such as position, speed, and health. Moreover, this structure allows me to add some other types of NPCs in the future without copying the code.

Diagram

Description automatically generated**Class diagram for loot:**

**Description:**

Loot class is the base class for all other loot specific classes. Loot class will contain basic attributes such as position, weight, dimensions, loot type and so on. Other child classes can have additional attributes and methods or rather override parent attributes or classes. The Inheritance will allow to reuse much of the code and in effect reduce the amount of code.

**Justification:**

This inheritance structure is best suited for my case because all types of loot share several same properties such as name, weight, position etc. Moreover, this structure allows me to add some other types loot into my game in the future without copying the code.

# OOP Structure:

**Base class for player and enemies**

|  |
| --- |
| **class People** |
| **Attributes:**  *- width (integer)*  *- height (integer)*  *- health (integer)*  *- speed (integer)*  *- color (tuple)*  *- bricks (list)*  *- bullets\_list (list)*  *- health\_bar (healthbar object)* |
| **Methods:**   * updatePlayerPosition() * getXPosition() * getYPosition() * isCollision() * move() * setSpeed() * shoot() |

**Algorithms:**

1. updatePlayerPosition(x, y)

Diagram

Description automatically generatedUpdates the player coordinates x and y. The arguments are new x and y coordinates.

1. getXPosition()

Method to get the x coordinate of the player.

Diagram, text

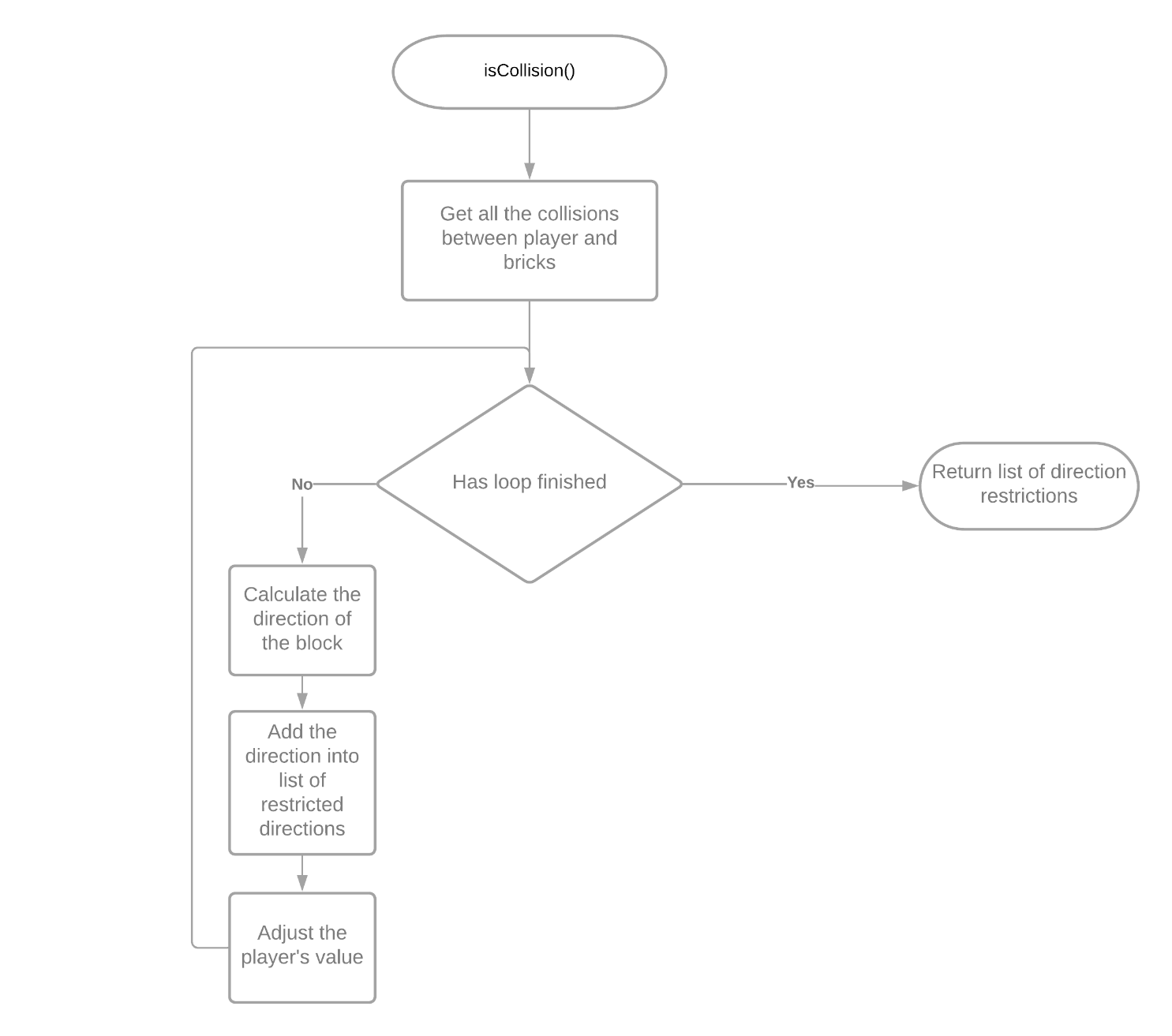
Description automatically generated

1. getYPosition()

Diagram, text

Description automatically generatedMethod for getting the y coordinate of the player.

1. isCollision()

Check if the player has collided with any wall bricks and restrain its movement in that direction. The algorithm uses pygame.sprite.spritecollide function to determine the collisions with walls. Then it checks the direction of the brick compared to the player or enemy (up, down, left or right). The method returns the list of restraint directions (e.g. [“up”, “”, “”, “”] means there is a wall on top of the player)

1. move()

pass the method to children classes.

1. setSpeed(newSpeed)

Diagram

Description automatically generatedThis method sets the speed for the player: speed is equal to newSpeed.

1. shoot()

This method creates a bullet and releases it. Then, it adds the bullet sprite to the bullet sprite list (bullets\_list)

**pass the method to child classes, where it can be overwritten**

|  |
| --- |
| **class Player** |
| **Attributes:**  *- width (integer)*  *- height (integer)*  *- health (integer)*  *- speed (integer)*  *- color (tuple)*  *- playerX (integer)*  *- playerY (integer)*  *- weight\_capacity (integer)*  *- inventory (list)*  *- selectedWeapon (integer)*  *- bullets (list)*  *- weapons (list)*  *- max\_amount\_weapons (integer)*  *- loot\_group (list)*  *- health\_bar (helathbar object)* |
| **Methods:**   * updatePlayerPosition() * getXPosition() * getYPosition() * isCollision() * move() * setSpeed() * shoot() * getInventoryWeight() * setSelectedWeapon() * heal() * getWeaponsList() * getBulletsList() * checkLootCollision() * getInventory() * getMedicineKitsAmount() * isBulletCollisionWithEnemy() * isHitByEnemy() |

**Algorithms:**

1. getInventoryWeight()

Method for calculating the weight of items in the inventory. Loops through all the items in the inventory and calculates the sum of their weights. and returns it.

Diagram

Description automatically generatedNew algorithms can be implemented: when new item is added, the current inventory weight will be incremented by the item’s weight. The attribute of player class will store the current inventory weight.

Figure 3 - getInventoryWeight() flowchart

1. setSelectedWeapon(value)

Method that sets the selected weapon for player accordin to the key the player pressed. If val is less than number of all weapons, than selectedWeapon = val-1.

Diagram

Description automatically generated

1. heal()

Diagram

Description automatically generatedIf the user inputted T, Y, or U the user can use 3 types of medicine kits: large, medium and small. Different kits have different weight and healing effect. So, the largest kit has the biggest healing effect and it is the largest one of all 3 types of medicine kits. The algorithm then checks if the player has corresponding medicine kits in his inventory. If yes, it will use it to heal the player (add health point to player) and delete from the medicine kit from inventory. Also, this method validates the health points, so the health level can’t rise above 100 points.

1. getWeaponsList()

Diagram, text

Description automatically generatedReturns the list of weapons the player possess.

1. getBulletsList()

Diagram, text

Description automatically generatedMethod that returns the list of player’s bullets in the inventory (type and amount of bullets)

1. checkLootCollision()

Diagram

Description automatically generatedThe method that checks if the user has collide with the loot boxes. It uses pygame.sprite.spritecollide function to get all the collisions. Then it checks the type of the loot; if the loot type is weapon, the weapon is added to the weapons list. If the loot type is bullets, then the bullets\_list is updated.

1. getInventory()

Returns the player’s inventory (list)

Diagram, text, application

Description automatically generated

1. getMedicineKitsAmount()

Diagram

Description automatically generatedReturns the list of the medicine kits the player posseses.

1. isHitByEnemy(enemies)

The method that checks if any of the enemies collided with the player. If it is true the enemy is killed and the player is damaged by 10 points. The score is also incremented by 10 points.

1. isBulletCollisionWithEnemy(enemies, incrementKills, incrementScore)

The method which checks if any of the player’s bullets have hit the enemis. If the bullet has hit the enemy, the enemy looses health points and the bullet is removed from the screen (removed from the bullets\_list as well). The kills counter and score counter are incremented by a certain amount.

1. shoot()

The method which allows the player to shoot particular types of bullets which correspond to weapons possessed by the player. So, the methos checks if the bullet type matches the type of the selected weapon and if the number of bullets is more than zero. If all of this is true, the bullet is released and the number of bullets is decremented.

1. move(direction)

This method allows the player to move in 4 directions on the screen: up, down, right and left. It checks if the player can move in that direction, and if yes, then it changes the corresponding coordinates by speed of the player.

1. Graphical user interface, text, application, chat or text message

   Description automatically generatedgetXPosition()

Returns player’s x coordinate

1. getYPosition()

**Graphical user interface, text, application, chat or text message

Description automatically generated**Return the player’s y coordinate

|  |
| --- |
| **class Enemy** |
| **Attributes:**  *- fieldView (integer)*  *- isAttacking (Boolean)*  - attackVector (array of integers) |
| **Methods:**   * drawHealthBar() * isCollision() * move() * getVector() * update() |

**Algorithms:**

1. drawHealthBar()

Method for drawing the healthbar for the enemy. It calls the update method of healthbar class, which update the value of the healthbar and redraws it on the screen.

1. isCollision()

Method that checks if there is any collision between the enemy and a wall. If yes, it restrains enemy’s movement, so it cant move in the direction where the wall is.

1. move()

The method which allows enemies to go around obstacles when chasing the player.

1. getVector()

The getter method which returns the attackVector of the enemy. The attack vector is the direction vector from the enemy to the player.

1. update()

The method for updating the position of the enemy and has the logic for chasing the player if the player is in its field of view.

|  |
| --- |
| **class Game** |
| **Attributes:**  *- numBricks (integer)*  *- brickSide (integer)*  *- kills (integer)*  *- score (integer)*  *- wave (integer)*  *- enemy\_sprites\_group (list)*  *- all\_sprites\_group (list)*  *- bricks\_sprites\_group (list)*  *- loot\_sprites\_group (list)*  *- player (player object)*  *- done (Boolean)*  *- inventoryList (inventory object)*  *- scoreboard (scoreboard object)* |
| **Methods:**   * incrementKills() * incrementScore() * createLoot() * createOutterWalls() * start() * end() * createEnemies() * reRender() * mainLoop() |

1. incrementKills()

Diagram

Description automatically generatedThe method for incrementing the current player’s number of enemies killed.

1. incrementScore()

Diagram

Description automatically generatedThe method for incrementing the player’s score by some amount.

1. Diagram

   Description automatically generatedcreateLoot()

The method for generating random loot and placing it in random places on the map. The loot can be weapons, medicine kits, armor or bullets. Bullets loot also comes in random number of bullets.

1. createOutterWalls()

Diagram

Description automatically generatedThe method for generating and drawing the outer walls for the game. It creates instances of Brick class and places them on the edges of the screen. It also adds all these blocks to the brick list and all sprites list.

1. start()

Diagram

Description automatically generatedThe method for starting the game. It creates an enemy on the map and calls the main game loop.

1. end()

The method for ending the game. It changes the DONE attribute to false and the main game loop stops.

Diagram

Description automatically generated

1. createEnemies()

The method for generating enemies on the game map in random position. It takes in the number of enemies to be generated to be spawn on the game.

Diagram

Description automatically generated

1. reRender()

The method which is in charge of updating all the sprites in the game (like enemis, walls, player, scoreboard). This method is called each game clock tick and reRenders the game frame.

Diagram

Description automatically generated

1. mainLoop()

The method which includes a main game loop. The main game loop calls a reRender method and check for the user input events such as mouse click and key press.

**Diagram

Description automatically generated**

|  |
| --- |
| **class Bullet** |
| **Attributes:**  *- name (string)*  *- width (integer)*  - height (integer)  - speed (integer)  - image  - rect: rect.x, rect.y (integer) |
| **Methods:**   * move() * draw() * update() |

1. move()

Method for moving the bullet by its speed. It updates the current bullet’s position.

Diagram

Description automatically generated

1. Diagram

   Description automatically generateddraw()

Method for drawing the bullet on the screen on the current bullet’s position.

1. update()

Method which is called every game tick and is meant to update bullet’s behavior. When the bullet escapes the game map it is removed from the screen.

Diagram

Description automatically generated

**Base loot class for all loot types:**

|  |
| --- |
| **class Loot** |
| **Attributes:**  *- weight (integer)*  *- name (string)*  - loot\_type (string)  - width (integer)  - height (integer)  - rect: rect.x, rect.y (integer) |
| **Methods:**  *none* |

1. Class constructor

It Initializes the attributes of loot such as name, weight, type, position, dimension to the values passed into the constructor.

## Test Plan

Fill in the table below with at least 30 tests:

|  |  |  |  |
| --- | --- | --- | --- |
| **Test** | **Description** | **Input** | **Expected Outcome** |
| 1 | Screen appears and is 800 by 600 pixels | Run the code | Screen appears on the display |
| 2 | Player moves right | Click D key | Player moves right 5 pixels |
| 3 | Player moves left | Click A key |  |
| 4 | Player moves up | Click W key |  |
| 5 | Player moves down | Click S key |  |
| 6 | Shooting | Left click | If the player has the weapon with appropriate bullets, he will shoot the bullet. |
| 7 | Looting | Player collides with loot box | If the player has enough weight capacity in his inventory, he can loot the item and the item will be added to inventory |
| 8 | Collision with walls | Player collides with walls | The player stops, he can’t move any farther in this direction. |
| 9 | Player hits the enemy with bullet | The bullet hits the enemy | If the player’s bullet hits the enemy, the enemy health is decremented. If enemy’s health <= 0, then the enemy dies (disappears) |
| 10 | Enemies attacking the player. | The player is in enemy’s field of view | The enemy should attack the player and try to kill him. |
| 11 | Selecting the weapon. | the user presses keys 1, 2 or 3 | If the user has a weapon in his inventory, then he selects this weapon. |
| 12 | The player kills the enemy. | The player hits the enemy and it dies | The player’s score is incremented, the kills value is increased by 1 as well. |
| 13 | The player kills all enemies on the screen. | No enemies in enemy group | Increase the wave value by 1. Spawn more new enemies on the screen (1 more than in last wave) |
| 14 | Enemies colliding with walls. | The collision list is not empty | The enemy should stop moving in that direction, where the wall is |
| 15 | Enemies colliding with the player. | The collision list is not empty | The enemy should damage player and the player’s health should decrease |
| 16 | Player hits the wall with a bullet. | The collision between bullets and bricks list is not empty | The walls which are destructive should be destroyed and the bullet should be remove from the player |
| 17 | The enemies chasing the player. | The player is close enough to the enemy | The enemy should be able to chase a player. |
| 18 | Increasing the number of enemies each wave. | When the player kills all the enemies. | Each wave there are more enemies than previously (increased by 1) |
| 19 | Outer walls generated. |  | Outer walls are generated in the beginning of the game |
| 20 | Inner walls generated. |  | Inner walls are generated inside outer walls in the beginning of the game |
| 21 | The loot boxes generated and placed randomly. |  | The loot boxes are generated on the map in random places |
| 22 | Display the healthbar under the player. |  | The healthbar is displayed under the player |
| 23 | Stick the healthbar position to the player. |  | The healbar of the players follows its movements |
| 24 | Display the healthbars over all the enemies. |  | The healthbars are drawn on top of all enemies |
| 25 | Update the healthbars when the health of the player/enemy changes. |  | When the health changes the value in the healthbar changes as well and it updates (shows different health level) |
| 26 | The inventory list is displayed in the top left corner of the game screen. |  | The inventory list is shown in the top left corner with white letters |
| 27 | New items are added to the inventory list. | If the player collects the loot | When the player collects loot, the new items should be added to the end of inventory list. |
| 28 | New items can’t be added to the inventory list when the inventory is full. | The inventory weight is full | The player can’t collect loot if his inventory weight is full |
| 29 | The player can heal, using the medicine kits. |  | The player should be able to use medicine kits and increase its health |
| 30 | The medicine kits are removed from the inventory as the player use it for healing. | Player using the medicine kit to heal | The player’s health is increased, and the medicine kit is removed from the inventory |

# Implementation

/\* Plan \*/

Iteration 1 (displaying all objects on the screen)

* Display black screen
* Player
* Player movement (up, down, left and right)

Iteration 2 (animating all objects on the screen):

* Outer walls
* Inner walls
* Collisions with walls

Iteration 3 (adding some advanced logic and AI):

* Refactoring Object Structure.
* Adding People base class
* Changing player class
* Adding Enemy class

*/\* Iteration 4:*

* *Adding base loot class and child classes (weapon, bullets, armor etc)*
* *Loot random generation*
* *Loot collisions*
* *Implementing an inventor*

*Iteration 5:*

* *Implementing player’s shooting logic*
* *Bullet collisions with enemies*
* *Bullet collisions with walls*
* *Killing enemies*
* *New enemies are generated each wave*

*Iteration 6:*

* *Implementing enemy movement and simple AI*
* *Enemies attacking a player*
* *Implementing healthbars for both player and enemies*
* *Implementing the scoreboard for score and kills*

*Iteration 7:*

* *Adding a game menu \*/*

/\* Plan \*/

## Iteration 1

### Game UI

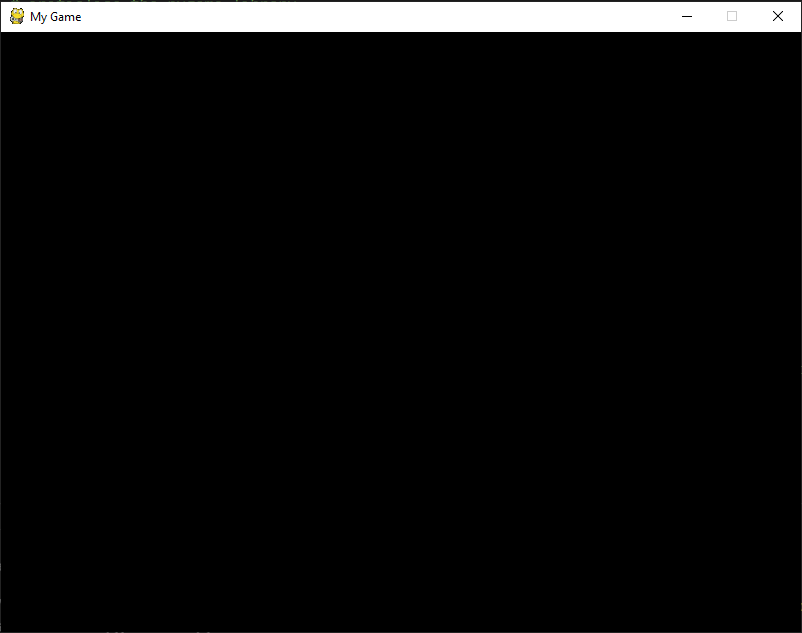
I started this iteration with creating a basic game interface such as black screen with dimensions of 800 by 600 px.

1. *#import pygame library*
2. **import** pygame
3. *#initialise the pygame library*
4. **pygame.init()**
5. *# setting black color hex to BLACK variable*
6. BLACK = (0, 0, 0)
7. *#set the size tuple to be 1000 by 1000 px*
8. size = (800, 600)
9. *#create a screen with a size of 800 by 600 px*
10. **screen = pygame.display.set\_mode(size)**
12. *#set the game caption to "My game"*
13. pygame.display.set\_caption("My Game")

This piece of code imports the pygame library which is used for building games in python. Then I initialise this library, so I can use all its methods. Then on line 8 I create a tuple called size to store a screen dimensions. On line 10 I create a screen with defined dimensions (800x600 px). On line 13 I set the game caption (it is displayed on top of the window) to “My game”. It can be further changed at any stage of development.

1. *# setting clock to pygame clock for rerendering the game frame*
2. clock = pygame.time.Clock()
4. *# setting the flag to False. If the flag is true the program exits the main loop and the game stops*
5. **done = False**
7. *# main loop*
8. **while** **not** done:
9. *#checking for events in the game*
10. **for event in pygame.event.get():**
11. *# if user want to exit the program (press the close button on the top of the window)*
12. *# then the program should breal out of the loop*
13. **if** event.type == pygame.QUIT:
14. done = True
16. *# filling all the screen with black color*
17. screen.fill(BLACK)
19. *# tick the clock 60 times per second*
20. **clock.tick(60)**
22. *# flip the display*
23. pygame.display.flip()

Here, on the line number 2 I declared the clock variable to control the clock in my game. Also, I defined the “done” variable and set it to False. This is some sort of a flag which If changed to True can stop the main game loop. On line 8 I put a while loop which breaks out when the done variable is True. From line 10 to 14 I check for events in my game and check if the user exits the program. If so, I change “done” flag to True and the main game loop breaks out and the game ends. On line number 17 I fill the screen with black colour (clear the screen). On line number 20 I say that the game loop should be executed 60 times per second. On line number 23 I use the flip method of the display. This method is responsible for updating all the content on the screen in my game.

Then I run my game and we can see only a black screen with size 800 by 600 px. I can also exit the game by clicking the close button in the top right corner of the window. There’s nothing more yet.

The source cose of my game looks like this so far:

1. *# import pygame library*
2. **import** pygame
3. *# initialise the pygame library*
4. pygame.init()
6. *# setting black color hex to BLACK variable*
7. BLACK = (0, 0, 0)
9. *# set the size tuple to be 1000 by 1000 px*
10. **size = (800, 600)**
11. *# create a screen with a size of 800 by 600 px*
12. screen = pygame.display.set\_mode(size)
14. *# set the game caption to "My game"*
15. **pygame.display.set\_caption("My Game")**
16. *# setting clock to pygame clock for rerendering the game frame*
17. clock = pygame.time.Clock()
19. *# setting the flag to False. If the flag is true the program exits the main loop and the game stops*
20. **done = False**
22. *# main loop*
23. **while** **not** done:
24. *#checking for events in the game*
25. **for event in pygame.event.get():**
26. *# if user want to exit the program (press the close button on the top of the window)*
27. *# then the program should breal out of the loop*
28. **if** event.type == pygame.QUIT:
29. done = True
31. *# filling all the screen with black color*
32. screen.fill(BLACK)
34. *# tick the clock 60 times per second*
35. **clock.tick(60)**
37. *# flip the display*
38. pygame.display.flip()
40. ***# exit the game when the loop is not running***
41. pygame.quit()

Then I proceed to testing If the screen size is what we expect (800 by 600 px) by logging the screen dimensions to the console.

I added these two lines of code to test it:

1. width, height = pygame.display.get\_surface().get\_size()
2. **print**("Width: "+str(width)+"**\n**Height: "+str(height))

Then there’s a result in the console:



### Displaying walls:

In my game I would like to have outer walls on the edges of the game window and inner walls inside. I also want inner walls to be partly destructible, meaning when the player shoots a bullet into the wall the wall has a gap in it. That’s why it is reasonable to construct a wall using square bricks. In my case I created a class Brick.

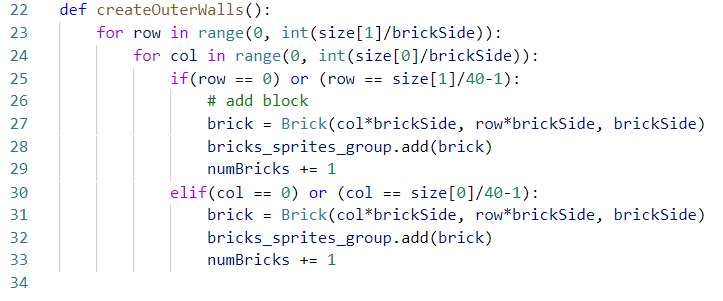
|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | **class** **Brick**(pygame.sprite.Sprite):  **def** **\_\_init\_\_**(self, x, y, brickSide):  super().\_\_init\_\_()  self.side = brickSide  self.image = pygame.Surface([self.side, self.side])  self.image.fill(YELLOW)  self.rect = self.image.get\_rect()  self.rect.x = x  self.rect.y = y |

So, this is a brick class. Each brick is a sprite because we need to display it on the screen and check for collisions. In line 1 I declare class Brick which inherits from pygame sprite class. Next there’s is a class constructor in line 2 which initialises the instances of this class. In line 3 we call a constructor of the parent class – pygame.sprite.Sprite. Then we initialise some attributes for the brick instance such as side length, image, rect and position (self.rect.x and self.rect.y)

Then we need to place all these bricks on the screen. In order to do that we need to create sprite groups for them. So, we add these code into our body of the code.

1. *#brick counter*
2. self.numBricks = 0
3. *# the length of the side of each brick*
4. self.brickSide = brickSide
5. ***# creating the sprite group for all the sprites***
6. self.all\_sprites\_group = pygame.sprite.Group()
7. *# creating the sprite group for brick sprites*
8. self.bricks\_sprites\_group = pygame.sprite.Group()

Next, I wrote a function called “createOuterWalls”, which is meant to generate all the outer walls and place them on the edges of the screen.

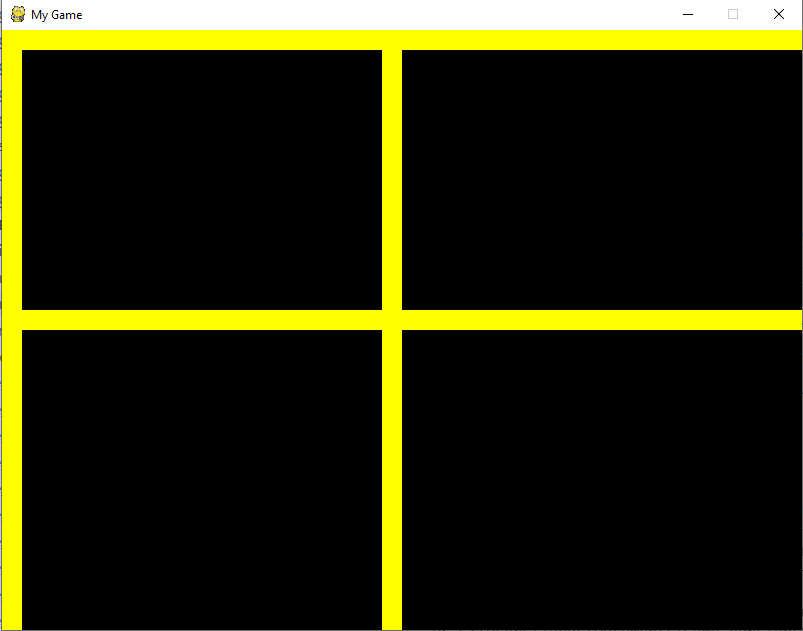
1. **def** createOuterWalls():
2. *# starting the loop which loops for every cell on the map and places blocks on the edges*
3. **for** row **in** range(0, int(size[1]/brickSide)) :
4. **for** col **in** range(0, int(size[0]/brickSide)):
5. **if(row == 0) or (row == size[1]/40-1):**
6. *# add block on the top and bottom rows*
7. brick = Brick(col\*brickSide, row\*brickSide, brickSide)
8. bricks\_sprites\_group.add(brick)
9. *# increment the number of bricks by 1*
10. **numBricks += 1**
11. **elif**(col == 0) **or** (col == size[0]/40-1):
12. *# add block on the right and left columns*
13. brick = Brick(col\*brickSide, row\*brickSide, brickSide)
14. bricks\_sprites\_group.add(brick)
15. ***# increment the number of bricks by 1***
16.  numBricks += 1

This algorithm loops through all the cells of size brickSide on the map and places Bricks on the top and bottom rows as well as on the most left and right columns.



Now the outer walls are properly generated. The length of each side is 40 px.

### Errors:

If we try to change the length of the side of a brick by changing the constant brickside, than the outer walls are not generated correctly. This happens because when the algorithm checks for the most right column it divides the width by 40 and then subtracts 1. This is similar for the most bottom row. See line 5 and 11 in code. Here’s the screenshot of a bug:

The solution is just to change 40 to brickSide. Now it works perfectly fine:

### Adding the player:

The player in my game is a sprite which will have some dimensions and it will be able to move in all 4 directions. We can represent a player as a square for simplicity for now. I started with adding a player class to my game:

1. **class** Player(pygame.sprite.Sprite):
2. **def** \_\_init\_\_(self, x, y, width, height, speed, health, bricks):
3. super().\_\_init\_\_()
4. self.bricks = bricks
5. **self.width = width**
6. self.height = height
7. self.health = health
8. self.speed = speed
10. **self.image = pygame.Surface([self.width, self.height])**
11. self.image.fill(BLUE)
12. self.rect = self.image.get\_rect()
13. self.rect.x = x
14. self.rect.y = y

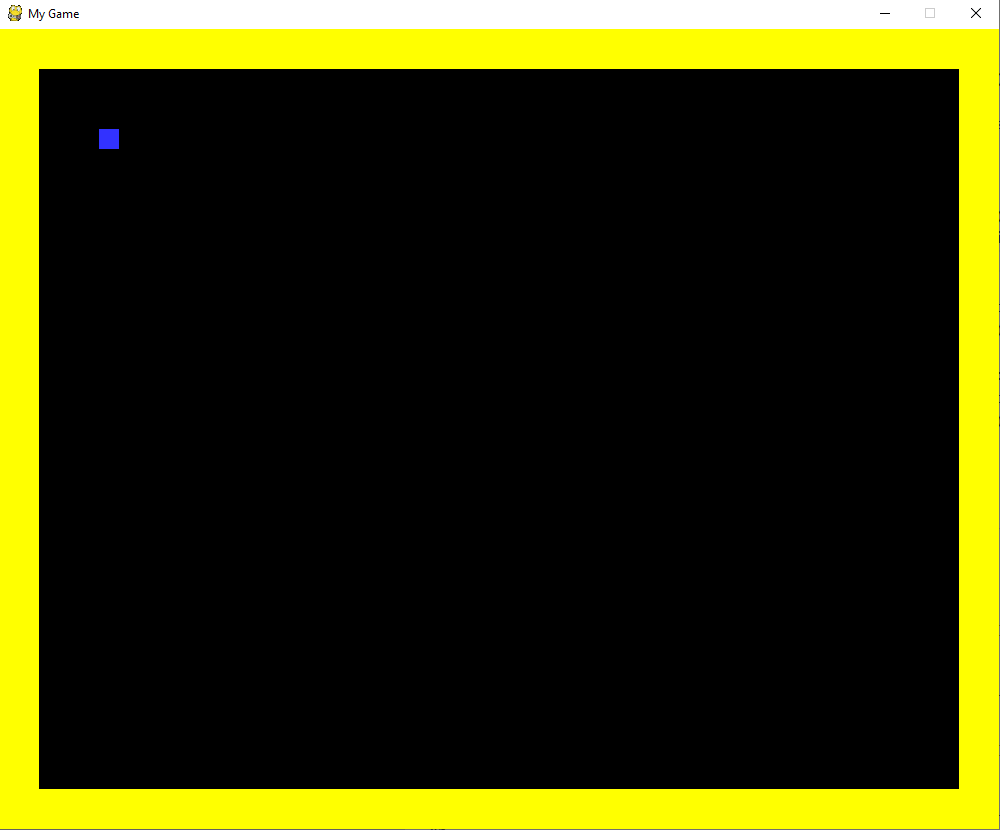
Then we create a player instance in our program by writing this line of code:

player = Player(100, 100, 20, 20, 5, 100, bricks\_sprites\_group)

And then it is necessary to add a player sprite to the sprite group all\_sprites\_group:

all\_sprites\_group.add(player)

Now when we call update and draw methods on all\_sprites\_group the player’s sprite will also be updated and drawn on the screen.

The player is a blue square by default.

The code looks like this so far:

1. *# import pygame library*
2. **import** pygame
3. *# initialise the pygame library*
4. pygame.init()
6. *# defining all colors as tuples*
7. BLACK = (0, 0, 0)
8. WHITE = (255, 255,255)
9. BLUE = (50, 50, 255)
10. **YELLOW = (255, 255, 0)**
11. GREEN = (50, 255, 50)
13. *# set the size tuple to be 1000 by 1000 px*
14. size = (800, 600)
15. ***# create a screen with a size of 800 by 600 px***
16. screen = pygame.display.set\_mode(size)
18. *# set the game caption to "My game"*
19. pygame.display.set\_caption("My Game")
20. ***# setting clock to pygame clock for rerendering the game frame***
21. clock = pygame.time.Clock()
23. *# setting the flag to False. If the flag is true the program exits the main loop and the game stops*
24. done = False

27. *#classes*
28. **class** Brick(pygame.sprite.Sprite):
29. **def** \_\_init\_\_(self, x, y, brickSide):
30. ***# calling the parent class constructor***
31. super().\_\_init\_\_()
33. self.side = brickSide
34. self.image = pygame.Surface([self.side, self.side])
35. **self.image.fill(YELLOW)**
36. self.rect = self.image.get\_rect()
37. self.rect.x = x
38. self.rect.y = y

41. **class** Player(pygame.sprite.Sprite):
42. **def** \_\_init\_\_(self, x, y, width, height, speed, health, bricks):
43. super().\_\_init\_\_()
44. self.bricks = bricks
45. **self.width = width**
46. self.height = height
47. self.health = health
48. self.speed = speed
50. **self.image = pygame.Surface([self.width, self.height])**
51. self.image.fill(BLUE)
52. self.rect = self.image.get\_rect()
53. self.rect.x = x
54. self.rect.y = y

57. **def** createOuterWalls():
58. **for** row **in** range(0, int(size[1]/brickSide)):
59. **for col in range(0, int(size[0]/brickSide)):**
60. **if**(row == 0) **or** (row == size[1]/brickSide-1):
61. *# add block to the most top and bottom rows*
62. brick = Brick(col\*brickSide, row\*brickSide, brickSide)
63. bricks\_sprites\_group.add(brick)
64. **all\_sprites\_group.add(brick)**
66. **elif**(col == 0) **or** (col == size[0]/brickSide-1):
67. *# add block to the most left and right columns*
68. brick = Brick(col\*brickSide, row\*brickSide, brickSide)
69. **bricks\_sprites\_group.add(brick)**
70. all\_sprites\_group.add(brick)

73. *# brick counter*
74. numBricks = 0
75. *# the length of the side of each brick*
76. brickSide = 20
77. ***# creating the sprite group for all the sprites***
78. all\_sprites\_group = pygame.sprite.Group()
79. *# creating the sprite group for brick sprites*
80. bricks\_sprites\_group = pygame.sprite.Group()
82. ***# create a player instance of a Player class***
83. player = Player(100, 100, 20, 20, 5, 100, bricks\_sprites\_group)
85. *# add a player to all sprites group*
86. all\_sprites\_group.add(player)
88. *# calling the function which generates the outer walls on the game map*
89. createOuterWalls()
91. *# main loop*
92. **while not done:**
93. *#checking for events in the game*
94. **for** event **in** pygame.event.get():
95. *# if user want to exit the program (press the close button on the top of the window)*
96. *# then the program should breal out of the loop*
97. **if event.type == pygame.QUIT:**
98. done = True
100. *# filling all the screen with black color*
101. screen.fill(BLACK)
103. *# updating all the sprites*
104. all\_sprites\_group.update()
106. *# drawing all the sprites on the screen*
107. **all\_sprites\_group.draw(screen)**
109. *# tick the clock 60 times per second*
110. clock.tick(60)
112. ***# flip the display***
113. pygame.display.flip()
115. *# exit the game when the loop is not running*
116. pygame.quit()

### Adding and displaying enemies:

I would like to add enemies which will eventually be able to attack a player. For simplicity, I decided to show all enemies as red squares. This is going to help me focus more on logic rather than the actual game design. I can change the enemy sprite at any time, but I will abstract from it for now.

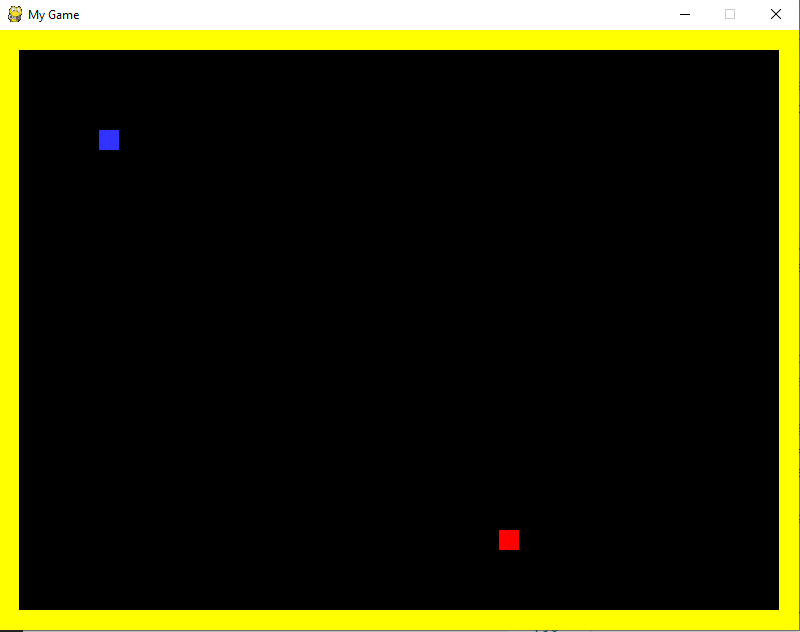
So, I started with implementing an Enemy class:

1. *# Enemy class*
2. **class** Enemy(pygame.sprite.Sprite):
3. **def** \_\_init\_\_(self, x, y, width, height, color, speed, health, bricks, player):
4. super().\_\_init\_\_()
5. **self.bricks = bricks**
6. self.width = width
7. self.height = height
8. self.health = health
9. self.speed = speed
11. self.attackVector = [0, 0, 0]
12. self.player = player
13. self.fieldView = 400
15. **self.image = pygame.Surface([self.width, self.height])**
16. self.image.fill(color)
17. self.rect = self.image.get\_rect()
18. self.rect.x = x
19. self.rect.y = y

Class Enemy inherits from pygame.sprite.Sprite class. This allows to use an Enemy as a sprite in our game. In line 3 we define the class constructor which initialises the instance of this class and sets all the attributes to passed values. In lines 15-19 we define how the enemy sprite will look like on the screen.

Next we create an enemy instance called “enemy” and add it to our all\_sprites\_group:

1. *#adding enemy*
2. enemy = Enemy(500, 500, 20, 20, RED, 5, 100, bricks\_sprites\_group, player)
3. all\_sprites\_group.add(enemy)

Here’s the screenshot of a result. The blue square is the player and a red square is the enemy.

The next step is to add inventory list and the scoreboard for the player.

### Adding the inventory list and a scoreboard:

Firstly, my scoreboard will display the number of kills and the total score of a player. I can declare this variable in my main program and not in the player class. So, I added the code:

1. *# defining game variables*
2. kills = 0
3. score = 0

Then I proceed and create a Scoreboard class which has a constructor and only one method draw:

1. *# Scoreboard class*
2. **class** ScoreBoard():
3. *#constructor (initialisation)*
4. **def** \_\_init\_\_(self, x, y, width, height):
5. **self.width = width**
6. self.height = height
7. self.x = x
8. self.y = y
10. ***# custom draw method***
11. **def** draw(self, kills, score):
12. *# kills label*
13. kills\_label = mainFont.render("Kills: "+str(kills), 1, WHITE)
14. *# score label*
15. **score\_label = mainFont.render("Score: "+str(score), 1, WHITE)**
16. *# displaying both labels on the screen*
17. screen.blit(kills\_label, (self.x, self.y))
18. screen.blit(score\_label, (self.x, self.y+40))

In line 4 there’s a class constructor where we instantiate all the needed attributes such as scoreboard width, hight and its position. In line 11 there is a custom draw method, which takes in 2 parameters: the number of kills and the score. Both of these parameters are integers. In order to display the scoreboard we first need to make an instance of this class called “scoreboard” for instance:

1. *# Initialising the scoreboard*
2. scoreboard = ScoreBoard(670, 30, 200, 100)

Then we need to call a custom draw method of our new scoreboard instance from our main game loop because we eventually want to update its content:

1. *# draw a scoreboard*
2. scoreboard.draw(kills, score)

Next step is to add an inventory list for a player. It is going be a list, which will store all the items in player’s inventory and it is going to be displayed in the top left corner. First I created InventoryList class. The code is given below:

1. **class** InventoryList():
2. *# constructor to initialise all the attributes*
3. **def** \_\_init\_\_(self, x, y, width, height):
4. self.x = x
5. **self.y = y**
6. self.width = width
7. self.height = height
9. *# a custommethod for drawing the inventory on the screen*
10. **def draw(self, inventory, weight, maxWeight):**
11. *# header label*
12. header = mainFont.render("Inventory("+str(weight)+"/"+str(maxWeight)+"): ", 1, WHITE)
13. *# number of items in the inventory*
14. counter = 0
16. i = 0
18. *# loop through all the items in the inventory*
19. **for** item **in** inventory:
20. ***# increment a counter***
21. counter+=1
23. *# check if the loot in the inventory is a bullet*
24. **if** (item['loot\_type'] == "bullet"):
25. **if(item['name'] == "bullet glock"):**
26. i = 0
27. **elif**(item['name'] == "bullet rifles"):
28. i = 1
29. **elif**(item['name'] == "bullet shotguns"):
30. **i = 2**
32. *# label for bullets (name and amount)*
33. item\_label = secondaryFont.render(item['name']+" ("+str(item.amount)+")", 1, WHITE)
34. **else**:
35. ***# a label for other items (just name)***
36. item\_label = secondaryFont.render(item['name'], 1, WHITE)
38. *# display the label on top of each other (vertical list)*
39. screen.blit(item\_label, (self.x, self.y+counter\*20))
41. *# display the header label*
42. screen.blit(header, (self.x, self.y))

Next we need to add the inventory attribute to the player class, so the player can have a list of items in the inventory. We should define this attribute in the constructor of Player class.

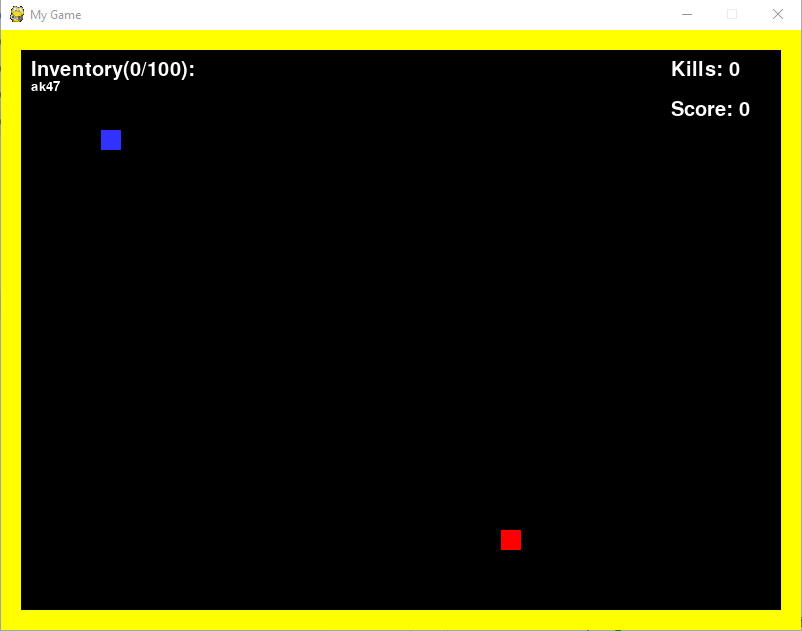
1. self.inventory = []

Now the Player class looks like this:

1. *# Player class*
2. **class** Player(pygame.sprite.Sprite):
3. **def** \_\_init\_\_(self, x, y, width, height, speed, health, bricks):
4. super().\_\_init\_\_()
5. **self.bricks = bricks**
6. self.width = width
7. self.height = height
8. self.health = health
9. self.speed = speed
11. self.image = pygame.Surface([self.width, self.height])
12. self.image.fill(BLUE)
13. self.rect = self.image.get\_rect()
14. self.rect.x = x
15. **self.rect.y = y**
17. *# list if items in the inventory*
18. self.inventory = []

Let’s also add 1 item to the inventory. I just added this code to the main program:

1. *# declare an item in the inventory list*
2. item = {
3. 'name': 'ak47',
4. 'loot\_type': 'weapon',
5. **}**
7. player.inventory.append(item)

This code adds the weapon “ak47” to the player’s inventory. Here’s the screenshot of the game on this stage.

The next step is to add healthbars to both player and enemy.

### Adding healthbars:

As shown in the design section the healthbars are going to be rectangular bars, where green rectangle represents the health, and the red rectangle represents the overall health. The green rectangle overlaps the red rectangle, so it looks like the lost health is red on the healthbar. In order to make healthbars I created the healthbar class:

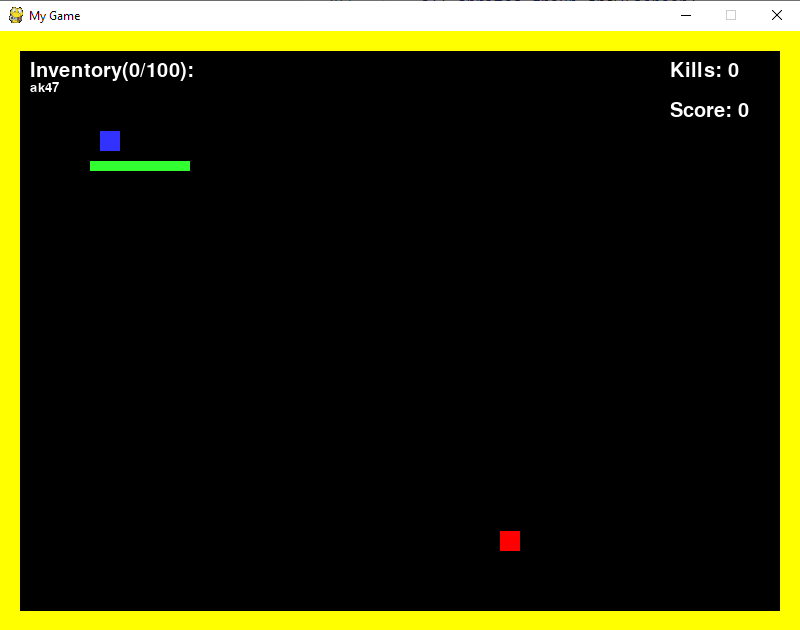
1. *#class for healthbar*
2. **class** HealthBar():
3. **def** \_\_init\_\_(self, objX, objY, width, height, initHealth):
4. self.maxHealth = initHealth
5. **self.outterContainer = pygame.Surface([width, height])**
6. self.outterContainer.fill(WHITE)
7. self.rectOutter = self.outterContainer.get\_rect()
8. self.rectOutter.x = objX
9. self.rectOutter.y = objY
11. self.innerContainer = pygame.Surface([width, height])
12. self.innerContainer.fill(GREEN)
13. self.rectInner = self.innerContainer.get\_rect()
14. self.rectInner.x = objX
15. **self.rectInner.y = objY**
16. self.maxWidth = width
17. self.height = height
19. *# update method for healthbar (the healthbar green rectangle changes in length)*
20. **def update(self, player, health, isPlayer):**
21. percent = health/self.maxHealth
22. newWidth = int(self.maxWidth \* percent)
24. **if** (newWidth <= 0):
25. **newWidth = 0**
27. self.innerContainer = pygame.Surface([newWidth, self.height])
28. self.innerContainer.fill(GREEN)
29. self.rectInner = self.innerContainer.get\_rect()
30. **self.rectInner.x = player.rect.x-player.width/2**

33. self.rectOutter.x = player.rect.x-player.width/2
34. **if** (isPlayer):
35. **self.rectOutter.y = player.rect.y + player.height + 10**
36. self.rectInner.y = player.rect.y + player.height + 10
37. **else**:
38. **print**(self)
40. **self.rectOutter.y = player.rect.y - 10**
41. self.rectInner.y = player.rect.y - 10
43. self.draw()
45. ***# custom method which draws the healthbar***
46. **def** draw(self):
47. screen.blit(self.outterContainer, (self.rectOutter.x, self.rectOutter.y))
48. screen.blit(self.innerContainer, (self.rectInner.x, self.rectInner.y))

Then I created the instance of the HealthBar class in the player class, in the constructor:

1. *# healthbar initialisation*
2. self.healthbar = HealthBar(self.rect.x, self.rect.y, 100, 10, health)

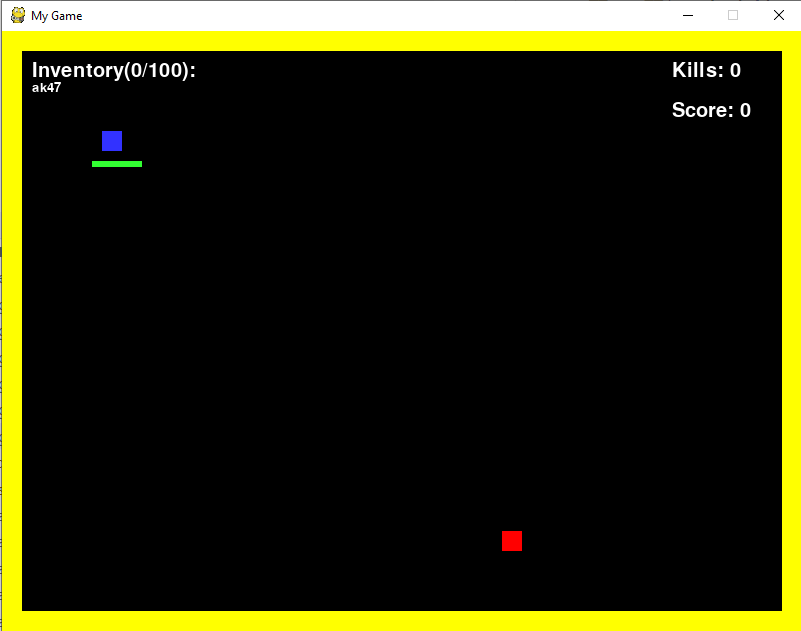
Finally, I had to call the update method in my main game loop to update and display player’s healthbar:

1. *# update and draw a player's healthbar*
2. player.healthbar.update(player, player.health, True)

Now, as shown on the screenshot above, the healthbar is not perfectly cantered and is a bit too long and wide, so I decide to change the dimensions of the healthbar when initialising the healthbar in the player’s class:

1. *# healthbar initialisation*
2. self.healthbar = HealthBar(self.rect.x, self.rect.y, 50, 6, health)

Now, the length of the healthbar is 50px and its height is just 6 px.

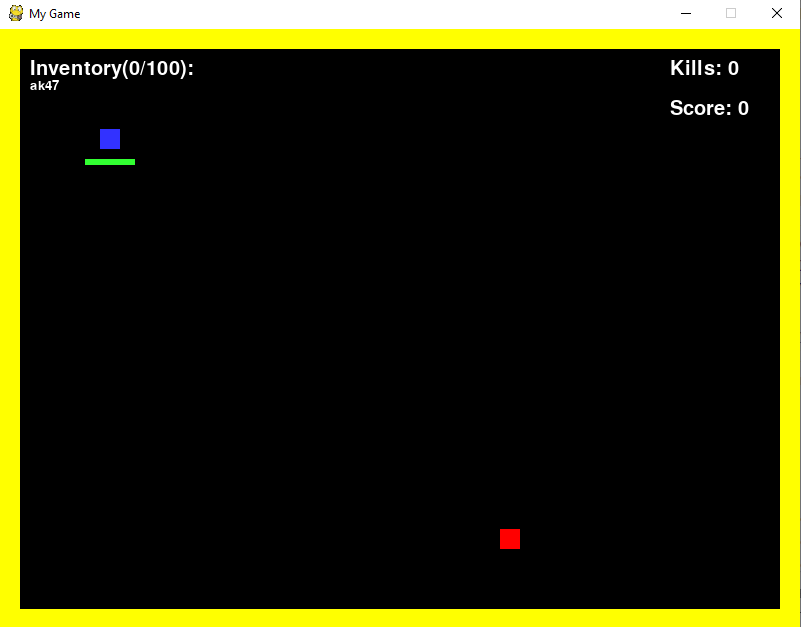


The next step is to place the healthbar in the centre relatively to a player.

1. *# setting the x coordinate of outer and inner rectangles to be centered*
2. self.rectInner.x = player.rect.x-self.maxWidth/2+player.width/2
3. self.rectOutter.x = player.rect.x-self.maxWidth/2+player.width/2

The x coordinate is the same as the x coordinate of the left side of the player. Then the half of the width of the healthbar is subtracted. Then the offset is added (offset equals half of length of the player).

Now, here’s the result:



My code looks like this so far:

1. *# In this Iteration I added healthbars*
3. *# import pygame library*
4. **import** pygame
5. ***# initialise the pygame library***
6. pygame.init()
8. *# defining all colors as tuples*
9. BLACK = (0, 0, 0)
10. **WHITE = (255, 255,255)**
11. BLUE = (50, 50, 255)
12. YELLOW = (255, 255, 0)
13. GREEN = (50, 255, 50)
14. RED = (255, 0, 0)
16. *#declare font for the game ui*
17. mainFont = pygame.font.SysFont("comicsans", 30)
18. secondaryFont = pygame.font.SysFont("comicsans", 20)
20. ***# set the size tuple to be 1000 by 1000 px***
21. size = (800, 600)
22. *# create a screen with a size of 800 by 600 px*
23. screen = pygame.display.set\_mode(size)
25. ***# set the game caption to "My game"***
26. pygame.display.set\_caption("My Game")
27. *# setting clock to pygame clock for rerendering the game frame*
28. clock = pygame.time.Clock()
30. ***# setting the flag to False. If the flag is true the program exits the main loop and the game stops***
31. done = False

34. *#classes*
35. **class Brick(pygame.sprite.Sprite):**
36. **def** \_\_init\_\_(self, x, y, brickSide):
37. *# calling the parent class constructor*
38. super().\_\_init\_\_()
40. **self.side = brickSide**
41. self.image = pygame.Surface([self.side, self.side])
42. self.image.fill(YELLOW)
43. self.rect = self.image.get\_rect()
44. self.rect.x = x
45. **self.rect.y = y**
47. *# Enemy class*
48. **class** Enemy(pygame.sprite.Sprite):
49. **def** \_\_init\_\_(self, x, y, width, height, color, speed, health, bricks, player):
50. **super().\_\_init\_\_()**
51. self.bricks = bricks
52. self.width = width
53. self.height = height
54. self.health = health
55. **self.speed = speed**
57. self.attackVector = [0, 0, 0]
58. self.player = player
59. self.fieldView = 400
61. self.image = pygame.Surface([self.width, self.height])
62. self.image.fill(color)
63. self.rect = self.image.get\_rect()
64. self.rect.x = x
65. **self.rect.y = y**
67. *# Player class*
68. **class** Player(pygame.sprite.Sprite):
69. **def** \_\_init\_\_(self, x, y, width, height, speed, health, bricks):
70. **super().\_\_init\_\_()**
71. self.bricks = bricks
72. self.width = width
73. self.height = height
74. self.health = health
75. **self.speed = speed**
77. self.image = pygame.Surface([self.width, self.height])
78. self.image.fill(BLUE)
79. self.rect = self.image.get\_rect()
80. **self.rect.x = x**
81. self.rect.y = y
83. *# list if items in the inventory*
84. self.inventory = []
86. *# healthbar initialisation*
87. self.healthbar = HealthBar(self.rect.x, self.rect.y, 50, 6, health)

90. ***# Scoreboard class***
91. **class** ScoreBoard():
92. *#constructor (initialisation)*
93. **def** \_\_init\_\_(self, x, y, width, height):
94. self.width = width
95. **self.height = height**
96. self.x = x
97. self.y = y
99. *# custom draw method*
100. **def draw(self, kills, score):**
101. *# kills label*
102. kills\_label = mainFont.render("Kills: "+str(kills), 1, WHITE)
103. *# score label*
104. score\_label = mainFont.render("Score: "+str(score), 1, WHITE)
105. ***# displaying both labels on the screen***
106. screen.blit(kills\_label, (self.x, self.y))
107. screen.blit(score\_label, (self.x, self.y+40))
109. **class** InventoryList():
110. ***# constructor to initialise all the attributes***
111. **def** \_\_init\_\_(self, x, y, width, height):
112. self.x = x
113. self.y = y
114. self.width = width
115. **self.height = height**
117. *# a custommethod for drawing the inventory on the screen*
118. **def** draw(self, inventory, weight, maxWeight):
119. *# header label*
120. **header = mainFont.render("Inventory("+str(weight)+"/"+str(maxWeight)+"): ", 1, WHITE)**
121. *# number of items in the inventory*
122. counter = 0
124. i = 0
126. *# loop through all the items in the inventory*
127. **for** item **in** inventory:
128. *# increment a counter*
129. counter+=1
131. *# check if the loot in the inventory is a bullet*
132. **if** (item['loot\_type'] == "bullet"):
133. **if**(item['name'] == "bullet glock"):
134. i = 0
135. **elif(item['name'] == "bullet rifles"):**
136. i = 1
137. **elif**(item['name'] == "bullet shotguns"):
138. i = 2
140. ***# label for bullets (name and amount)***
141. item\_label = secondaryFont.render(item['name']+" ("+str(item.amount)+")", 1, WHITE)
142. **else**:
143. *# a label for other items (just name)*
144. item\_label = secondaryFont.render(item['name'], 1, WHITE)
146. *# display the label on top of each other (vertical list)*
147. screen.blit(item\_label, (self.x, self.y+counter\*20))
149. *# display the header label*
150. **screen.blit(header, (self.x, self.y))**

153. *#class for healthbar*
154. **class** HealthBar():
155. **def \_\_init\_\_(self, objX, objY, width, height, initHealth):**
156. self.maxHealth = initHealth
157. self.outterContainer = pygame.Surface([width, height])
158. self.outterContainer.fill(WHITE)
159. self.rectOutter = self.outterContainer.get\_rect()
160. **self.rectOutter.x = objX**
161. self.rectOutter.y = objY
163. self.innerContainer = pygame.Surface([width, height])
164. self.innerContainer.fill(GREEN)
165. **self.rectInner = self.innerContainer.get\_rect()**
166. self.rectInner.x = objX
167. self.rectInner.y = objY
168. self.maxWidth = width
169. self.height = height
171. *# update method for healthbar (the healthbar green rectangle changes in length)*
172. **def** update(self, player, health, isPlayer):
173. percent = health/self.maxHealth
174. newWidth = int(self.maxWidth \* percent)
176. **if** (newWidth <= 0):
177. newWidth = 0
179. self.innerContainer = pygame.Surface([newWidth, self.height])
180. **self.innerContainer.fill(GREEN)**
181. self.rectInner = self.innerContainer.get\_rect()
183. *# setting the x coordinate of outer and inner rectangles to be centered*
184. self.rectInner.x = player.rect.x-self.maxWidth/2+player.width/2
185. **self.rectOutter.x = player.rect.x-self.maxWidth/2+player.width/2**
187. **if** (isPlayer):
188. self.rectOutter.y = player.rect.y + player.height + 10
189. self.rectInner.y = player.rect.y + player.height + 10
190. **else:**
191. **print**(self)
193. self.rectOutter.y = player.rect.y - 10
194. self.rectInner.y = player.rect.y - 10
196. self.draw()
198. *# custom method which draws the healthbar*
199. **def** draw(self):
200. **screen.blit(self.outterContainer, (self.rectOutter.x, self.rectOutter.y))**
201. screen.blit(self.innerContainer, (self.rectInner.x, self.rectInner.y))

204. **def createOuterWalls():**
205. **for** row **in** range(0, int(size[1]/brickSide)):
206. **for** col **in** range(0, int(size[0]/brickSide)):
207. **if**(row == 0) **or** (row == size[1]/brickSide-1):
208. *# add block to the most top and bottom rows*
209. **brick = Brick(col\*brickSide, row\*brickSide, brickSide)**
210. bricks\_sprites\_group.add(brick)
211. all\_sprites\_group.add(brick)
213. **elif**(col == 0) **or** (col == size[0]/brickSide-1):
214. ***# add block to the most left and right columns***
215. brick = Brick(col\*brickSide, row\*brickSide, brickSide)
216. bricks\_sprites\_group.add(brick)
217. all\_sprites\_group.add(brick)
219. *# brick counter*
220. numBricks = 0
221. ***# the length of the side of each brick***
222. brickSide = 20
223. *# creating the sprite group for all the sprites*
224. all\_sprites\_group = pygame.sprite.Group()
225. *# creating the sprite group for brick sprites*
226. **bricks\_sprites\_group = pygame.sprite.Group()**
228. *# create a player instance of a Player class*
229. player = Player(100, 100, 20, 20, 1, 100, bricks\_sprites\_group)
231. ***# add a player to all sprites group***
232. all\_sprites\_group.add(player)
234. *# adding an enemy to the scene*
235. enemy = Enemy(500, 500, 20, 20, RED, 5, 100, bricks\_sprites\_group, player)
236. **all\_sprites\_group.add(enemy)**
238. *# Initialising the scoreboard*
239. scoreboard = ScoreBoard(670, 30, 200, 100)
241. ***# Initialising the inventory list***
242. inventoryList = InventoryList(30, 30, 200, 400)
244. *# calling the function which generates the outer walls on the game map*
245. createOuterWalls()
247. *# defining game variables*
248. kills = 0
249. score = 0
251. ***# declare an item in the inventory list***
252. item = {
253. 'name': 'ak47',
254. 'loot\_type': 'weapon',
255. }
257. player.inventory.append(item)
259. *# main loop*
260. **while** **not** done:
261. ***#checking for events in the game***
262. **for** event **in** pygame.event.get():
263. *# if user want to exit the program (press the close button on the top of the window)*
264. *# then the program should breal out of the loop*
265. **if** event.type == pygame.QUIT:
266. **done = True**
268. *# filling all the screen with black color*
269. screen.fill(BLACK)
271. ***# updating all the sprites***
272. all\_sprites\_group.update()
274. *# drawing all the sprites on the screen*
275. all\_sprites\_group.draw(screen)
277. *# draw a scoreboard*
278. scoreboard.draw(kills, score)
280. *# update and draw a player's healthbar*
281. **player.healthbar.update(player, player.health, True)**
283. *# draw an inventory list*
284. inventoryList.draw(player.inventory, 0, 100)
286. ***# tick the clock 60 times per second***
287. clock.tick(60)
289. *# flip the display*
290. pygame.display.flip()
292. *# exit the game when the loop is not running*
293. pygame.quit()

## Iteration 2

I decided to start Iteration 2 with restructuring my code and adding game class, so it is easier to handle all other sprites on my screen and animate them. So, I wrote this game class:

[game class]

**class** Game():

**def** \_\_init\_\_(self, brickSide):

*#pygame.mouse.set\_visible(False)*

self.numBricks = 0

self.brickSide = brickSide

self.kills = 0

self.score = 0

self.wave = 1

*# declaration of sprite groups*

self.enemy\_sprites\_group = pygame.sprite.Group()

self.all\_sprites\_group = pygame.sprite.Group()

self.bricks\_sprites\_group = pygame.sprite.Group()

self.loot\_sprites\_group = pygame.sprite.Group()

*#init the player and add him to the sprite group*

self.player = Player(100, 100, 20, 20, BLUE, 2, 100, self.bricks\_sprites\_group, self.loot\_sprites\_group, 50, self.all\_sprites\_group)

self.all\_sprites\_group.add(self.player)

self.done = False

self.gameover = False

*#init the inventory list board*

self.inventoryList = InventoryList(50, 50, 100, 100)

*#init the score board*

self.scoreBoard = ScoreBoard(830, 50, 100, 100)

*#create the border walls*

self.createOutterWalls()

*#randomly place the loot*

self.createLoot()

self.isMenu = True

*#creating the inner wall*

**for** i **in** range(5, 10):

brick = Brick(i\*self.brickSide, 5\*40, self.brickSide)

self.bricks\_sprites\_group.add(brick)

self.all\_sprites\_group.add(brick)

self.numBricks += 1

**print**(self.numBricks)

*#function for rendering outer walls on the window*

**def** createOutterWalls(self):

**for** row **in** range(0, int(1000/self.brickSide)) :

**for** col **in** range(0, int(1000/self.brickSide)):

**if**(row == 0) **or** (row == 1000/40-1):

*#add block*

brick = Brick(col\*self.brickSide, row\*self.brickSide, self.brickSide)

self.bricks\_sprites\_group.add(brick)

self.all\_sprites\_group.add(brick)

self.numBricks += 1

**elif**(col == 0) **or** (col == 1000/40-1):

brick = Brick(col\*self.brickSide, row\*self.brickSide, self.brickSide)

self.bricks\_sprites\_group.add(brick)

self.all\_sprites\_group.add(brick)

self.numBricks += 1

Then I proceeded and added the start method to the game class (start method should start the game when called):

**def** start(self):

self.done = False

enemy = Enemy(600, 600, 20, 20, RED, 1, 100, self.bricks\_sprites\_group, self.player)

self.enemy\_sprites\_group.add(enemy)

self.all\_sprites\_group.add(enemy)

*#self.mainLoop()*

self.mainMenu()

**def** end(self):

self.done = True

At this stage I needed to implement a game main loop. In order to accomplish this I decided to write a mainLoop method in game class:

**def** mainLoop(self):

**while** **not** self.done:

screen.fill(BLACK)

self.reRender()

**for** event **in** pygame.event.get():

**if** event.type == pygame.QUIT:

self.end()

**if** (event.type == pygame.MOUSEBUTTONDOWN) **and** (event.button == 1):

**print**("Left click!")

self.player.shoot()

keys = pygame.key.get\_pressed()

**if** keys[pygame.K\_a]:

*#move the player to the right*

self.player.move("left")

**if** keys[pygame.K\_d]:

*#move the player to the left*

self.player.move("right")

**if** keys[pygame.K\_w]:

*#move the player up*

self.player.move("up")

**if** keys[pygame.K\_s]:

*#move the player down*

self.player.move("down")

*#selecting the weapon*

**if** keys[pygame.K\_1]:

self.player.setSelectedWeapon(1)

**if** keys[pygame.K\_2]:

self.player.setSelectedWeapon(2)

**if** keys[pygame.K\_3]:

self.player.setSelectedWeapon(3)

**if** keys[pygame.K\_t]:

self.player.heal(1)

**if** keys[pygame.K\_y]:

self.player.heal(2)

**if** keys[pygame.K\_u]:

self.player.heal(3)

*#if keys[pygame.K\_LSHIFT]:*

*#move the player down*

*#self.player.setSpeed(10)*

*#else:*

*#self.player.setSpeed(1)*

clock.tick(60)

*#EndWhile*

At each clock of the game (or each iteration of the main game loop I should call rerender function, which will refresh the current display and show next game frame). So, I added reRender method to game class as well:

1. **def** reRender(self):
2. playerX = self.player.getXPosition()
3. playerY = self.player.getYPosition()
5. self.player.isBulletCollisionWithEnemy(self.enemy\_sprites\_group, self.incrementKills, self.incrementScore)
6. self.enemy\_sprites\_group.update(playerX, playerY, self.enemy\_sprites\_group)
7. self.player.bullets\_list.update(self.player.bullets\_list, self.all\_sprites\_group)
8. *#render the player*
10. collision\_with\_enemy = pygame.sprite.spritecollide(self.player, self.enemy\_sprites\_group, True)
12. **for** hit **in** collision\_with\_enemy:
13. self.player.health -= 50
15. self.all\_sprites\_group.draw(screen)
16. self.player.update()
18. self.player.health\_bar.update(playerX, playerY, self.player.rect.width, self.player.rect.height, self.player.health, True)
19. **self.scoreBoard.draw(self.kills, self.score)**
20. self.inventoryList.draw(self.player.getInventory(), self.player.getInventoryWeight(), self.player.getWeightCapacity(), self.player.getBulletsList(), self.player.getWeaponsList())
22. **if** (self.player.health <= 0):
23. self.player.kill()
24. **self.gameover = True**
25. self.done = True

28. **if** (len(self.loot\_sprites\_group)==0):
29. **self.createLoot()**
31. **if**(len(self.enemy\_sprites\_group) == 0):
32. self.wave += 1
33. self.createEnemies(self.wave)
35. pygame.display.update()

### Animating the player movement

### Animating the enemy movement

### Adding Collisions

### Adding shooting functionality

### Animating the inventory list

### Animating the scoreboard

## Iteration 3

### Implementing simple AI for enemies

***/\*Draft\*/***

**class People:**

***Attributes:***

*- width*

*- height*

*- health*

*- speed*

*- color*

*- playerX*

*- player*

***Methods:***

def \_\_init(self):

self.bricks = bricks

self.width = width

self.height = height

self.health = health

self.speed = speed

self.color = color

self.bullets\_list = pygame.sprite.Group()

self.image = pygame.Surface([self.width, self.height])

self.image.fill(self.color)

self.rect = self.image.get\_rect()

self.rect.x = x

self.rect.y = y

self.playerX = 0

self.playerY = 0

#health bar component

self.health\_bar = HealthBar(self.rect.x, self.rect.y, self.width\*2, self.height/3, self.health)

Plus:

* updatePlayerPosition()
* getXPosition()
* getYPosition()
* isCollision()
* move()
* setSpeed()
* shoot()
* drawHealthBar()

**class Player:**

Attributes:

* weight\_capacity
* inventory
* selectedWeapon
* bullets
* weapons
* loot\_group

Methods:

def \_\_init\_\_(self, x, y, width, height, color, speed, health, bricks, loot, inventory\_capacity):

        super().\_\_init\_\_(x, y, width, height, color, speed, health, bricks)

        pygame.sprite.Sprite.\_\_init\_\_(self)

        self.weight\_capacity = inventory\_capacity

        self.inventory = []

        self.selectedWeapon = -1

        #declare the list of the number of bullets, where 0 - pistols bullets, 1 - rifles bullet, 2 - gunshot bullets

        self.bullets = [0, 0, 0]

        #weapons[0] for glocks, 1 for ak47, 2 for shotguns

        self.weapons = [False, False, False]

        self.max\_amount\_weapons = 3

        self.loot\_group = loot

def getInventoryWeight(self):

        weight = 0

        for item in self.inventory:

            weight += item.weight

        return weight

    def setSelectedWeapon(self, val):

        if(val <= len(self.weapons)):

            self.selectedWeapon = val-1

        print(self.selectedWeapon)

def heal(self, indx):

        medicine = self.getMedicineKitsAmount()

        val = 0

        if (len(medicine[indx-1]) > 0):

            arr = medicine[indx-1]

            val = arr[len(arr)-1].healing

            print(val)

            self.inventory.remove(arr[len(arr)-1])

        if (self.health + val >= 100):

            self.health = 100

        else:

            self.health += val

def getWeaponsList(self):

        return self.weapons

def getBulletsList(self):

        return self.bullets

Plus:

* checkLootCollision()
* getInventory()
* getWeightCapacity()
* getMousePosition()
* getPlayerDirection()
* getPlayerBearing()
* getMedicineKitsAmount()
* isHitByEnemy()
* isBulletCollisionWithEnemy()
* shoot()
* move()

**class Enemy:**

**Attributes:**

* attackVector
* player
* fieldView

**Methods:**

* constructor:

def \_\_init\_\_(self, x, y, width, height, color, speed, health, bricks, player):

        super().\_\_init\_\_(x, y, width, height, color, speed, health, bricks)

        pygame.sprite.Sprite.\_\_init\_\_(self)

        self.attackVector = [0, 0, 0]

        self.player = player

        self.fieldView = 400

* attack()
* def attack(self):
* if (self.attackVector[2] <= self.fieldView):
* self.move()
* move()

def move(self):

        no\_direction=self.isCollision()

        if (self.attackVector[0] == 0):

            fraction = 0

        else:

            fraction = self.attackVector[1] / self.attackVector[0]

        xSpeed = self.speed/(math.sqrt(1+pow(fraction, 2)))

        ySpeed = xSpeed\*fraction

        #print(ySpeed)

        if (self.attackVector[0] < 0):

            #left

            self.rect.x -= math.ceil(xSpeed)

        else:

            #right

            self.rect.x += math.ceil(xSpeed)

        if (self.attackVector[1] < 0):

            #down

            self.rect.y += math.ceil(ySpeed)

        else:

            #up

            self.rect.y -= (-1)\*math.floor(ySpeed)

* getVector()
* def getVector(self):
* return self.attackVector
* update()

def update(self):

        self.attackVector[0] = self.player.rect.x-self.rect.x

        self.attackVector[1] = self.rect.y-self.player.rect.y

        self.attackVector[2] = math.sqrt(pow(self.attackVector[0], 2)+pow(self.attackVector[1], 2))

        #print(self.attackVector)

        #print("playerX: "+str(self.player.rect.x)+"  enemyX: "+str(self.rect.x))

        if (self.attackVector[2] <= self.fieldView):

            self.attack()

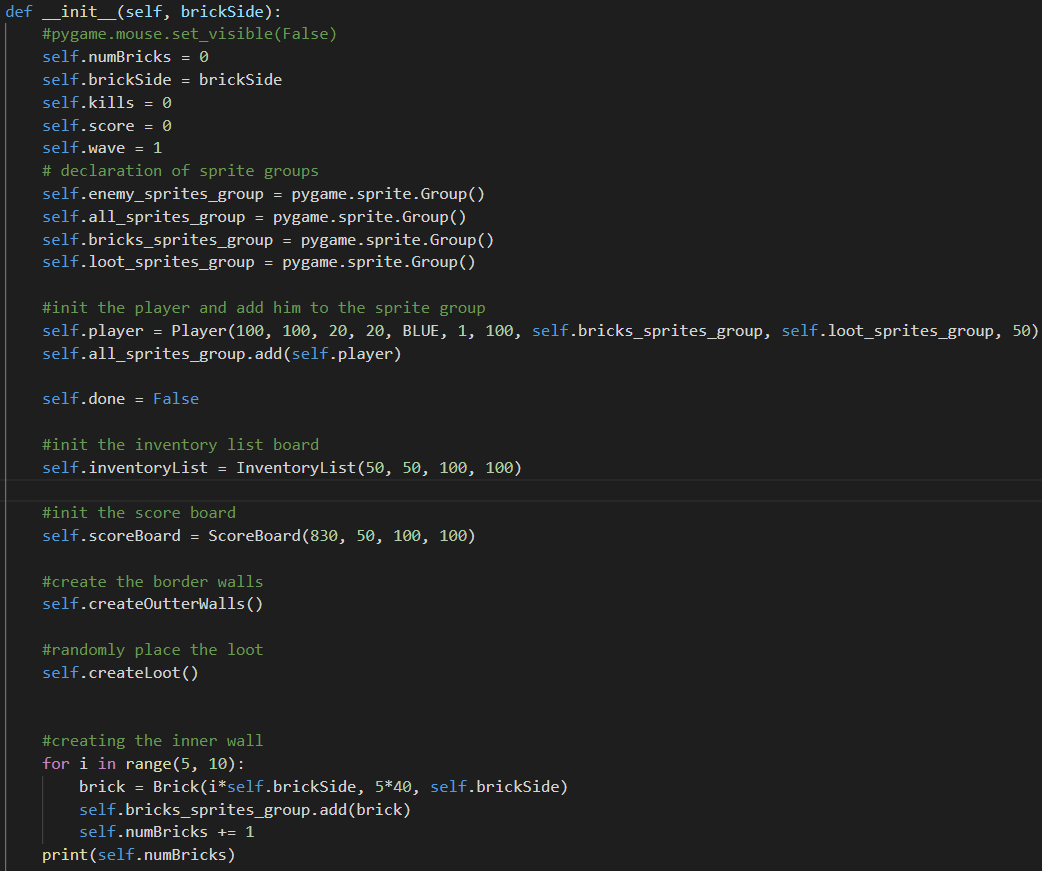
Game Class

Attributes:

* numBricks
* brickSide
* kills
* score
* wave
* enemy\_sprites\_group
* all\_sprites\_group
* bricks\_sprites\_group
* loot\_sprites\_group
* player
* done
* inventoryList
* scoreboard

Methods:

* constructor



* incrementKills()
* def incrementKills(self):
* self.kills += 1
* decrementKills()

def incrementScore(self, val):

        self.score += val

* createLoot()
* def createLoot(self):
* x = random.randint(40, 960)
* y = random.randint(40, 960)
* lootType = LOOT\_TYPES[random.randint(0, len(LOOT\_TYPES)-1)]
* if (lootType == "weapon"):
* weapon\_type = WEAPON\_TYPES[random.randint(0, len(WEAPON\_TYPES)-1)]
* loot = Weapon(x, y, 20, 20, GREEN, weapon\_type)
* print("The weapon "+weapon\_type+"was added!")
* elif(lootType == "bullets"):
* bullet\_type = BULLET\_TYPES[random.randint(0, len(BULLET\_TYPES)-1)]
* print("Bullets " + bullet\_type + " were added!")
* loot = BulletsLoot(x, y, 20, 20, GREEN, bullet\_type)
* elif(lootType == "paramedics"):
* paramedic\_type = PARAMEDIC\_TYPES[random.randint(0, len(PARAMEDIC\_TYPES)-1)]
* print("Paramedic was added!")
* loot = Paramedic(x, y, 20, 20, GREEN, paramedic\_type)
* elif(lootType == "armour"):
* armour\_type = ARMOUR\_TYPES[random.randint(0, len(ARMOUR\_TYPES)-1)]
* print("Armour "+armour\_type+" was added!")
* loot = Armour(x, y, 20, 20, GREEN, armour\_type)
* #self.all\_sprites\_group.add(loot)
* self.loot\_sprites\_group.add(loot)
* createOutterWalls()

def createOutterWalls(self):

        for row in range(0, int(1000/self.brickSide)) :

            for col in range(0, int(1000/self.brickSide)):

                if(row == 0) or (row == 1000/40-1):

                    #add block

                    brick = Brick(col\*self.brickSide, row\*self.brickSide, self.brickSide)

                    self.bricks\_sprites\_group.add(brick)

                    self.numBricks += 1

                elif(col == 0) or (col == 1000/40-1):

                    brick = Brick(col\*self.brickSide, row\*self.brickSide, self.brickSide)

                    self.bricks\_sprites\_group.add(brick)

                    self.numBricks += 1

* start()
* def start(self):
* self.done = False
* enemy = Enemy(600, 600, 20, 20, RED, 1, 100, self.bricks\_sprites\_group, self.player)
* self.enemy\_sprites\_group.add(enemy)
* self.all\_sprites\_group.add(enemy)
* self.mainLoop()
* end()

def end(self):

        self.done = True

* createEnemies()
* def createEnemies(self, quantity):
* for i in range(quantity):
* x = random.randint(40, 940)
* y = random.randint(40, 940)
* enemy = Enemy(x, y, 20, 20, RED, 1, 100, self.bricks\_sprites\_group, self.player)
* self.enemy\_sprites\_group.add(enemy)
* self.all\_sprites\_group.add(enemy)
* reRender()



* mainLoop()
* def mainLoop(self):
* while not self.done:
* screen.fill(BLACK)
* self.reRender()
* for event in pygame.event.get():
* if event.type == pygame.QUIT:
* self.end()
* if (event.type == pygame.MOUSEBUTTONDOWN) and (event.button == 1):
* print("Left click!")
* self.player.shoot()
* keys = pygame.key.get\_pressed()
* if keys[pygame.K\_a]:
* #move the player to the right
* self.player.move("left")
* if keys[pygame.K\_d]:
* #move the player to the left
* self.player.move("right")
* if keys[pygame.K\_w]:
* #move the player up
* self.player.move("up")
* if keys[pygame.K\_s]:
* #move the player down
* self.player.move("down")
* #selecting the weapon
* if keys[pygame.K\_1]:
* self.player.setSelectedWeapon(1)
* if keys[pygame.K\_2]:
* self.player.setSelectedWeapon(2)
* if keys[pygame.K\_3]:
* self.player.setSelectedWeapon(3)
* if keys[pygame.K\_t]:
* self.player.heal(1)
* if keys[pygame.K\_y]:
* self.player.heal(2)
* if keys[pygame.K\_u]:
* self.player.heal(3)
* if keys[pygame.K\_LSHIFT]:
* #move the player down
* self.player.setSpeed(10)
* else:
* self.player.setSpeed(1)
* clock.tick(240)
* #EndWhile

Functionality

Describe the game loop and any global variables or constants

For each method:

* Explain what it does and how which requirement it fulfils
* Write out pseudo code
* Draw a flowchart

game loop:

* rerender the scene
* listen for events (key press or mouse button click)

Iteration 1

Requirements being developed

Errors

Conclusion

Iteration 2

Requirements being developed

Errors

Conclusion

Iteration 3

Requirements being developed

Errors

Conclusion

# Testing

# Evaluation