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

The game will be RPG with a map, different types of enemies, loot boxes etc. The main purpose of the game is to go through several waves of enemies and kill the boss. The player may choose to rush the game and go straight to the boss, but it will be almost impossible to kill him, as the player wouldn’t have strong weapons, armor and so on. The player can pick up the loot boxes (with common weapons, medicine kits, and light armor) or can kill the enemies to get the rarer and better ammunition.

## Computational Suitability

Why is suited to being played on a computer?

The game doesn’t require the internet connection, so many people can p[lay it when the are board. This game is an RPG game with the elements of a shooter, so it obviously can’t 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 on.

## 

## Research

What type of games have you looked at and drawn inspiration from e.g. Pacman, platform games, side scrolling games etc.

This game was inspired by retro 2D RPG games.

A screenshot of a video game

Description automatically generated

## Map Description automatically generated

## Stakeholders

Who would be interested in playing your game?

Everybody who likes retro rpg games and shooters.

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. Also, this game is a good game for people travelling without the internet (e.g. by bus, train or airplane). This people will probably like this game as well as it can make the journey more comfortable and funnier.

Who can help you design the game?

I can ask help of my friends in the boarding house, teachers, and my family.

# Interviews

Ask the stakeholders about what sort of game they would like

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

I have collected some useful responses that helped me to align my game’s goals for the future players.

## Interview 1 (Friends):

Q1a: What would you like to see in the 2D rpg game?

Q2a: If you had a chance, what feature you would add into this game?

Q3a: How do you want the player in the game to move?

Q4a: How do you want to control the player (basic keys)?

Q5a: Do you want to have various levels of difficulty in the game (e.g. different enemies – bosses, half-bosses and usual enemies)?

Q6a: What sorts of weapons and battle techniques would you like to see in the game?

## Interview 2 (Family)

Q1a: What would you like to see in the 2D rpg game?

Q2a: If you had a chance, what feature you would add into this game?

Q3a: How do you want the player in the game to move?

Q4a: How do you want to control the player (basic keys)?

Q5a: Do you want to have various levels of difficulty in the game (e.g. different enemies – bosses, half-bosses and usual enemies)?

Q6a: What sorts of weapons and battle techniques would you like to see in the game?

# Interview Conclusion

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

* Most of the interviewers have said that they would like to see a player moving around a map with randomly located enemies.
* There is a big group of people who would like to use keys WASD to move the plyer, while others want to use Arrow keys.
* The interviewers told me the would really appreciate if the player could pick up the random loot and have an inventory to see what items he has.
* For shooting, everybody agreed to use right mouse button
* Finally, everybody agreed to have various levels of difficulty in the game.

# 

# 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.*

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.)
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.*

1. Player can pick up the loot

*Player will be able to collect the loot and store the items of the loot in the inventoy*

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 won’t 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 can’t 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 dist combat enemies (armed with knifes, katanas, axes)
2. Medium dist combat enemies (armed with pistols, bows or rifles)
3. Far dist 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

# Software Requirements

1. python

# 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:

**800px**

**600px**

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

## 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

**class diagram for players and enemies:**

class People

class Player

class Enemy

**Class diagram for loot:**

class Loot

class Paramedic

class Weapon

class BulletsLoot

class Armour

Base class for player and enemies

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

**Algorithms:**

1. updatePlayerPosition(x, y)

Updates 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.

1. getYPosition()

Method 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)

This 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)

|  |
| --- |
| **class Player** |
| **Attributes:**  *- width*  *- height*  *- health*  *- speed*  *- color*  *- playerX*  *- player*  *- weight\_capacity*  *- inventory*  *- selectedWeapon*  *- bullets*  *- weapons*  *- max\_amount\_weapons*  *- loot\_group*  *- health\_bar* |
| **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.

New 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.

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.

1. heal()

If 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()

Returns the list of weapons the player possess.

1. getBulletsList()

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

1. checkLootCollision()

The 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)

1. getMedicineKitsAmount()

Returns 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 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. getXPosition()

Returns player’s x coordinate

1. getYPosition()

Return the player’s y coordinate

|  |
| --- |
| **class Enemy** |
| **Attributes:**  *- fieldView*  *- isAttacking*  - attackVector |
| **Methods:**   * drawHealthBar() * isCollision() * move() * getVector() * update() |

**Algorithms:**

1. drawHealthBar()

Method for drawing the healthbar for the enemy. It calls the method of healthbar class

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.

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

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

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

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

**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()

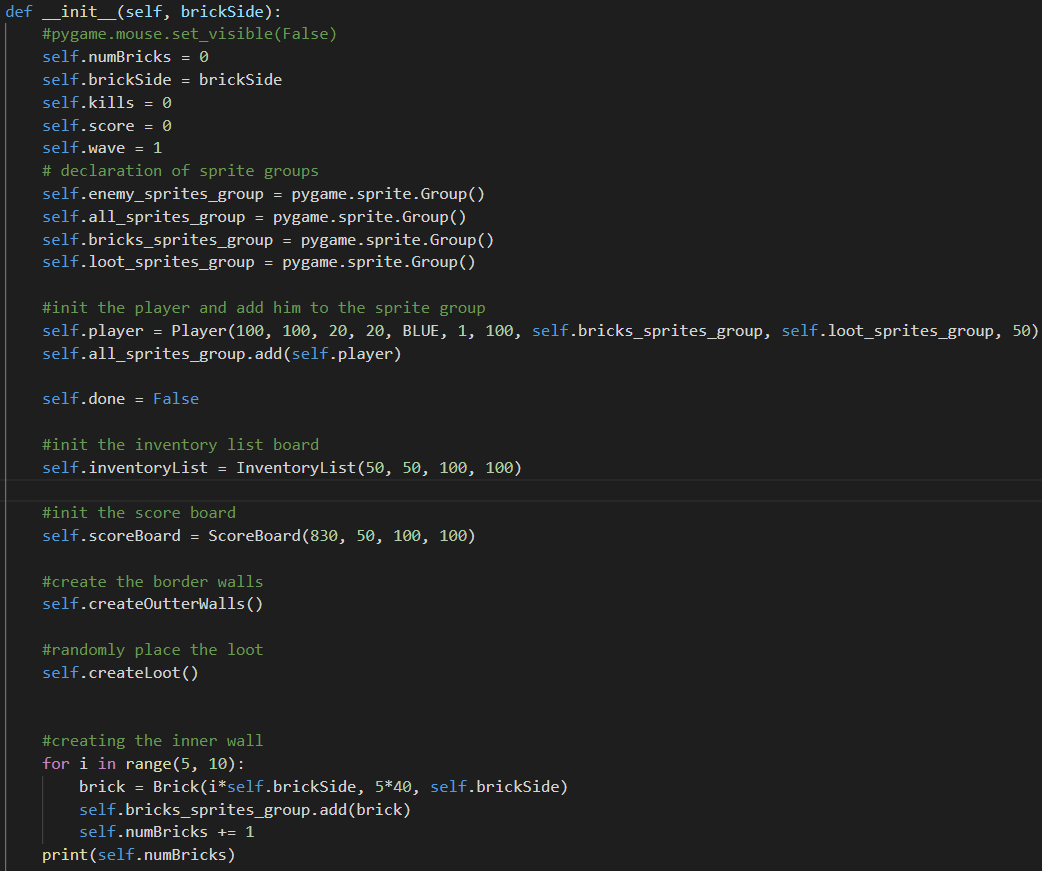
Game Class

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)

## 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) |  |

# Implementation

## Iteration 1

### Requirements being developed

### Errors

### Conclusion

## Iteration 2

### Requirements being developed

### Errors

### Conclusion

## Iteration 3

### Requirements being developed

### Errors

### Conclusion

# Testing

# Evaluation