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 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.

## 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.

## Stakeholders

Who would be interested in playing your game?

Everybody who likes retro rpg games and shooters.

Who can help you design the game?

Outsource designer, friends, family or myself.

# Interviews

Ask the stakeholders about what sort of game they would like

## Interview 1:

Q1a

Q1b

Q1c

## Interview 2

Q2a

Q2b

Q2c

# Interview Conclusion

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

# Requirements

List the requirements of you game

Must have:

1. Screen size = 720x1280 pixels
2. Destructible and non-destructible walls
3. Player sprite
4. Enemies with some sort of AI (chasing the player, shooting etc.)
5. Player must move in 4 directions
6. Player can shoot
7. Player can pick up the loot
8. Player can die
9. Loser and Winner window
10. Auto generation of loot on the map
11. Enemy might have random loot, which the player can get after killing him
12. Inventory can be full
13. Adding the items to inventory
14. Displaying the weight of the items in the inventory
15. Player can heal using the medicine kits
16. Player can use the armor
17. Armor adds the armor points, which can be drawn after getting the damage from the enemies
18. 3 types of medicine kits: big (restores 50% of health), medium (restores 20% of health), small (restores 10% of health)
19. 3 types of armor: heavy, medium, light
20. More items the player has in his inventory, the slower he moves
21. Heavy and medium armor can decrease the player’s speed.
22. Player can pick up the weapons
23. Player can pick up the bullets
24. Weapons can’t shoot if there are no bullets
25. 3 types of enemies:
26. Close dist combat enemies (armed with knifes, katanas, axes)
27. Medium dist combat enemies (armed with pistols, bows or rifles)
28. Far dist combat enemies (armed with sniper rifles)
29. 3 Bosses:
30. First boss with a bow (easy)
31. Second Boss with a sniper rifle (medium)
32. Third boss with a rifle and missiles (hard)
33. The camera follows the player
34. Player can drop the weapon
35. Player can select the weapon using buttons 1, 2, 3
36. Player can use other items from inventory using buttons R, T, Y

Should have

1. C
2. D

Could have

1. C
2. D

Won’t have

1. C
2. D

# Hardware Requirements

# Software Requirements

# 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.

Data design:

1. Base class for all players
2. Class for a user player
3. Class for an enemy

**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()

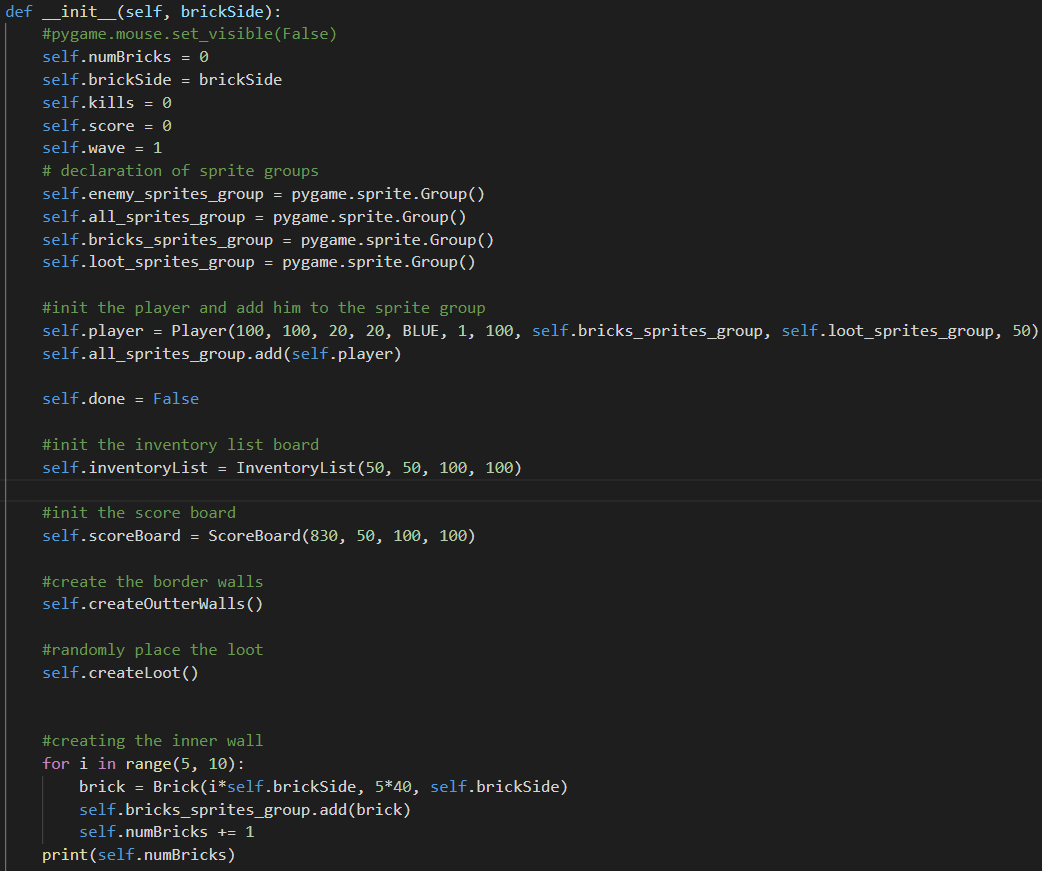
Game Class

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