IT 19-204-0713 MINI PROJECT - MULTIMEDIA PROJECT

PROJECT REPORT

On

STRANGER THINGS - GAME

Submitted by

Anupriya Asok - 20419506

Anzil Nizar - 20419507

Neeraj N - 20419520

Nived P - 20419522

Syam Suresh P C - 20419536

BACHELOR IN TECHNOLOGY

in

INFORMATION TECHNOLOGY

DIVISION OF INFORMATION TECHNOLOGY

COCHIN UNIVERSITY COLLEGE OF ENGINEERING KUTTANAD

PULINCUNNOO, ALAPPUZHA

CERTIFICATE

This is to certify that the project report entitled “STRANGER THINGS” submitted by Anupriya Asok (20419506), Anzil Nizar (20419507), Neeraj N (20419520), Nived P (20519522) and Syam Suresh P C (20419536) of Cochin University of Science and Technology towards partial fulfillment of requirements for the award of the degree of Bachelor of Technology in Information Technology is a bonafide record work carried out by them under our supervision and guidance during the academic year 2022-2023.

Project Guide

Jabir K V T

Head of Department

Dr. Harikrishnan D

ACKNOWLEDGEMENT

The project on the topic “Stranger Things - Game” was taken as a part of the curriculum for the award of B.Tech degree in Information Technology Engineering. At the outset, we thank almighty God for making our endeavor a success. We also express our gratitude to Dr. Harikrishnan D, Head of the Department for providing us with adequate facilities, ways and means by which we were able to complete this project. We also express our sincere gratitude to our faculty Jabir K V T, Lekshmi Rajagopal (Information Technology Division) for their constant support and valuable suggestions without which the successful completion of this project would not have been possible. We express our immense pleasure and thankfulness to all the teachers and staff of the Department of Information Technology, CUSAT for their cooperation and support. We would also like to thank some of our seniors, who helped us in all ways they could when we were stuck on important conjectures during this effort of ours. Last but not the least, we thank all others, especially our classmates and our family members who in one way or another helped us in the successful completion of this work.

CONTENTS

ABSTRACT

Stranger Things - game is a first person shooter (FPS) game in which the player assume the role of a survivor known as Eleven fighting its way through hordes of invading creatures from stranger things universe (upside down). This science fiction game is inspired from popular stranger things television series. Enemies often appear in large groups thus increasing the quantity and damage done by enemies. The monsters have very simple behavior, consisting of either moving towards their opponent or by shooting them. The game utilizes the multimedia aspect by incoperating high quality MP3 audios and detailed visuals of each monsters and also the same goes with the background images and map. On attack, the life of the player decreases gradually and keeps on increasing while not in combat mode.

1. INTRODUCTION

Stranger Things game is a fun and addictive first person shooter game in which enemies often appear in large groups, thus increasing the quantity and damage done by enemies. The monsters have very simple behavior, consisting of either moving towards their opponent or by shooting. The life of the player decreases as the intensity of attack increases and at last the player is greeted by Victory screen once all the enemies are defeated.

1.1 AIM

To build a first person shooter (FPS) game using python and pygame library.

1.2 OBJECTIVES

- The game is developed for full-time entertainment and enthusiasm. The main objective of this game is to hit the enemies and to survive.
- Easy gameplay and simple GUI is provided.
- Kid friendly approach is implemented due to game simplistic design.
- Controls of the game is very easy.

1.3 SCOPE OF THE PROJECT

This game will be implemented using Python language and Pycharm IDE.Python is a high-level, general purpose programming language. Whereas Pycharm is an integrated development environment (IDE) used in programming & developed specifically for python. The game will ensure that the UI is understandable and user can play the game without any prior interpretations.

2. SOFTWARE REQUIREMENT ANALYSIS

2.1 SYSTEM REQUIREMENTS

- RAM: 256 MB
- CPU: Intel Pentium Dual core
- Storage: 100 MB of free disk space
- Display: 800x600 (min)
- Adobe Flash Player

Recommended Hardware:

- RAM > 256 MB
- CPU: Intel Pentium I11 1.0 GHz
- Storage: 100 MB of free disk space
- Display: 1080x720

2.2 SOFTWARE REQUIREMENTS

- Operating System: Windows 7 or above
- Softwares: Python, Pycharm

2.3 FEATURES AND CONSTRAINTS

- The game is built using Python, hence platform independent.
- Pygame library is implemented.
- Requires low amount of RAM and storage.
- Ray casting method is used for the game development.
- Controls are user friendly.
- Smooth scrolling and accurate shooting.
- Different levels of motions and animations are visualized for in-game characterstics.
- A clear and accurate health bar is displayed for the player which decreases upon attack and increases upon shealth.

2.4 BASIS OF DESIGN

The main plot of the game heavily relies on a fictional world in which the main player acts as a slayer against the demons in upside down. The ultimate aim of the game is to confront terrifying supernatural forces. The game consists of the design elements in a constrained world such as pixelated png images with set of movements. The game includes a main background score and a defined sounds for various in-game actions such as attacks, pain, etc.The player stays always at the center of the display, hence the aiming is improved. The design also incorporates a accurate health bar located on the left side which increases and decreases upon attack. Depending upon the game outcome the player is greeted with either a “Victory” screen or a “Game Over “screen.

2.5 BASIS OF TESTING

In the testing phase, the controls as well as the completeness of the animation is being tested. When the controls are pressed, for eg: when the control ‘W’ is pressed the main player will move forward, when ‘A’ key is pressed the main player will move to the left, when ‘D’ is presssed the player will move right side and when ‘S’ key is pressed the main player will move backwards. The mouse right click will activate firing. Throughout the testing phase, the animation as well as the controls are being succesfully tested.

3. SYSTEM DESIGN

3.1 DESIGN

The game relies on a fictional world in which the main player acts as a slayer against the demons in upside down. The ultimate aim of the game is to confront terrifying supernatural forces. The game consists of the design elements in a constrained world such as pixelated png images with set of movements. the game includes a main background score and a defined sounds for various in-game actions such as attacks pain etc. The design also incorporates a accurate health bar located on the left side which increases and decreases upon attack. Depending upon the game outcome the player is greeted with either a “Victory” or a “Game Over” screen.

3.2 DESIGN DECISION

We have documented the design and design decisions in order to provide the basis for implementation and unit. Initially, when the team decided to go forward with an first person 3D shooting game, the idea was yet to be finalized. The team came up with the very idea of a "Stranger Things" shooting game. This idea is an inspiration of the famous television series "Stranger Things" which is the world of darkness (upside down). The ultimate aim of the game is to confront terrifying supernatural forces. The game consists of the design elements in a constrained world such as pixelated png images with set of movements. The game includes a main background score and a defined sounds for various in-game actions such as attacks, pain, etc. The UI of the game was designed using the **Figma** tool. Since this game is a First person 3D shooting game based on Stranger things, it comprises with a dark world environment to navigate through. Hence the UI has been designed taking these things into consideration.

3.3 USER INTERFACE DESIGN

3.4 SOFTWARE TEST DOCUMENTATION (STD)

TESTING METHODOLOGY

- In the testing phase, the control as well as the completeness of the animation is being tested.
- When the controls are pressed, for example: when the control ‘W’ is pressed the main player will move forward, when ‘A’ key is pressed the main player will move to the left, when ‘D’ is presssed the player will move right side and when ‘S’ key is pressed the main player will move backwards. The mouse right click will activate firing.
- Throughout the testing phase, the animation as well as the controls are being succesfully tested.
- It is checked whether the computer system has been installed correctly.
- Software can accurately and consistently produce results that meet predetermined guidelines for compliance and quality management.
- **Design Qualification (DQ):** This is typically supplied by the software vendor; it documents design specifications, software requirements, functional specifications, operational specifications and vendor attributes.
- **Installation Qualification (IQ):** At this stage, your tests and documentation will confirm that the software has been installed correctly according to your company's specifications and user requirements, the vendor's recommendations and the FDA's guidance. This goes for all hardware, software, equipment and systems.
- **Operational Qualification (OQ):** These tests establish confidence that the software will consistently perform the way it is supposed to when operating within expected ranges. These tests and results can be supplied by the vendor, since they involve standard features and security capabilities.
- **Performance Qualification (PQ):** This stage confirms that the software, as it was installed, will perform the way your company needs it to. Based on the processes and specifications outlined in the previous stages, your tests and documentation validate that the product being produced will meet EDA requirements for functionality and safety.

4. IMPLEMENTATION

4.1 BASIS OF IMPLEMENTATION

During the implementation, there should be strict adherence to standards. If some tasks that have been overlooked in the earlier phases, they must be corrected during this phase. A proper implementation schedule should be created and followed for timely achievement of goals in the design.

4.2 CODE

main.py

```
import sys

import pygame as pg

from database import getScores, loadScores
from fonts import FontRenderer
from mainmenu import MainMenu
from map import *
from object_handler import *
from object_renderer import *
from pathfinding import *
from player import *
from raycasting import *
from settings import *
from sound import *
from sprite_object import *
from weapon import *


class Game:
    def _init_(self):
        pg.init()
        self.screen = pg.display.set_mode(RES)
        self.font_renderer = FontRenderer(self)
        self.clock = pg.time.Clock()
        self.delta_time = 1
        self.global_trigger = False
        self.global_event = pg.USEREVENT + 0
        pg.time.set_timer(self.global_event, 40)
        self.level = 1
        self.score = 0
        self.enemies = 5
```

```
        self.MAIN_MENU = 0
        self.GAME = 1
        self.CUR_MENU = self.MAIN_MENU
        loadScores()
        self.name = 'no name'
        if 'current' in getScores():
            self.name = getScores()['current']

        self.main_menu_scene = MainMenu(self)

    def start_game(self):
        pg.mouse.set_visible(False)
        self.game_started = True
        self.new_game()

    def new_game(self):
        self.map = Map(self)
        self.player = Player(self)
        self.object_renderer = ObjectRenderer(self)
        self.raycasting = RayCasting(self)
        self.object_handler = ObjectHandler(self)
        self.weapon = Weapon(self)
        self.sound = Sound(self)
        self.sound.loadGameSounds()
        self.pathfinding = PathFinding(self)
        pg.mixer.music.play(-1)
        self.level = 1
        self.score = 0
        self.enemies = 5

    def update(self):
        self.player.update()
        self.raycasting.update()
        self.object_handler.update()
        self.weapon.update()
        pg.display.set_caption(f'{self.clock.get_fps() :.1f}')
        pg.display.flip()
        self.delta_time = self.clock.tick(FPS)

    def draw(self):
        self.object_renderer.draw()
        self.weapon.draw()

    def check_events(self):
        self.global_trigger = False
        for event in pg.event.get():
            if event.type == pg.QUIT or (
                    event.type == pg.KEYDOWN and event.key == pg.K_ESCAPE):
```

```
            if self.CUR_MENU == self.GAME:
                self.CUR_MENU = self.MAIN_MENU
            else:
                pg.quit()
                sys.exit()
        elif event.type == self.global_event:
            self.global_trigger = True
        self.player.single_fire_event(event)

    def main_menu(self):
        self.main_menu_scene.update()

    def game(self):
        self.check_events()
        self.update()
        self.draw()

    def run(self):
        while True:
            if self.CUR_MENU == self.MAIN_MENU:
                self.main_menu()
            elif self.CUR_MENU == self.GAME:
                self.game()


if _name_ == '_main_':
    game = Game()
    game.run()
```

map.py

main map.py

```
from button import Button
import sys
import pygame as pg
from leaderboards import Leaderboard
from profile import ProfileMenu
from settings import *
from sound import Sound


class MainMenu:
    def _init_(self, game):
        self.game = game
        self.screen = game.screen
        self.font_renderer = game.font_renderer
        self.buttons = []
        self.initButtons()
        self.mouseDown = False
        self.sound = Sound(self)
        self.sound.loadMenuSounds()
        pg.mixer.music.play(-1)
        self.image = self.get_texture('resources/textures/stranger.jpg', RES)
        self.show_leaderboard = False
        self.show_profile = False
        self.leaderboard = Leaderboard(self)
        self.profile = ProfileMenu(self)
        self.keys = []

    def initButtons(self):
        self.buttons = []
        start = Button(
            self.game,
            20,
            HEIGHT // 2,
            'Start Game', 'white',
            self.start_game)

        leaderboards = Button(
            self.game,
            20,
            HEIGHT // 2 + 100,
            'Leader boards', 'white',
            self.on_leaderboard)

        credits = Button(
         self.game,
            20,
            HEIGHT // 2 + 200,
            'Credits', 'white',
            self.start_game)

        profile = Button(
            self.game,
```

```
WIDTH // 2 - 100,
        20,
        f"Profile - {self.game.name}", 'white',
        self.on_profile)

    self.add_button(leaderboards)
    self.add_button(start)
    self.add_button(credits)
    self.add_button(profile)

def start_game(self):
    self.game.CUR_MENU = self.game.GAME
    self.game.new_game()

def on_leaderboard(self):
    self.show_leaderboard = True

def on_profile(self):
    self.show_profile = True

def check_events(self):
    self.mouseDown = False
    self.keys = []
    for event in pg.event.get():
        if event.type == pg.KEYDOWN:
            self.keys.append(event)
        if event.type == pg.QUIT or (
                event.type == pg.KEYDOWN and event.key == pg.K_ESCAPE):
            pg.quit()
            sys.exit()
        if event.type == pg.MOUSEBUTTONDOWN:
            self.mouseDown = True

def update(self):
    self.game.screen.blit(self.image, (0, 0))
    for button in self.buttons:
        button.draw()
        if self.mouseDown:
            button.clicked()

    self.check_events()
    if self.show_leaderboard:
        self.leaderboard.draw()

    if self.show_profile:
        self.profile.draw()

    pg.display.flip()

def add_button(self, button):
    self.buttons.append(button)

@staticmethod
def get_texture(path, res=(TEXTURE_SIZE, TEXTURE_SIZE)):
    texture = pg.image.load(path).convert_alpha()
    return pg.transform.scale(texture, res)
```

```
import pygame as pg

_ = False
mini_map = [
    [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1],
    [1, _, _, _, _, _, _, _, _, _, _, _, _, _, _, 1],
    [1, _, _, 3, 3, 3, 3, _, _, _, 2, 2, 2, _, _, 1],
    [1, _, _, _, _, _, 4, _, _, _, _, _, 2, _, _, 1],
    [1, _, _, _, _, _, 4, _, _, _, _, _, 2, _, _, 1],
    [1, _, _, 3, 3, 3, 3, _, _, _, _, _, _, _, _, 1],
    [1, _, _, _, _, _, _, _, _, _, _, _, _, _, _, 1],
    [1, _, _, _, 4, _, _, _, 4, _, _, _, _, _, _, 1],
    [1, 1, 1, 3, 1, 3, 1, 1, 1, 3, _, _, 3, 1, 1, 1],
    [1, 1, 1, 1, 1, 1, 1, 1, 1, 3, _, _, 3, 1, 1, 1],
    [1, 1, 1, 1, 1, 1, 1, 1, 1, 3, _, _, 3, 1, 1, 1],
    [1, 1, 3, 1, 1, 1, 1, 1, 1, 3, _, _, 3, 1, 1, 1],
    [1, 4, _, _, _, _, _, _, _, _, _, _, _, _, _, 1],
    [3, _, _, _, _, _, _, _, _, _, _, _, _, _, _, 1],
    [1, _, _, _, _, _, _, _, _, _, _, _, _, _, _, 1],
    [1, _, _, 2, _, _, _, _, 3, 4, _, 4, 3, _, 1],
    [1, _, _, 5, _, _, _, _, _, 3, _, 3, _, _, 1],
    [1, _, _, 2, _, _, _, _, _, _, _, _, _, _, 1],
    [1, _, _, _, _, _, _, _, _, _, _, _, _, _, _, 1],
    [3, _, _, _, _, _, _, _, _, _, _, _, _, _, _, 1],
    [1, 4, _, _, _, _, _, 4, _, _, 4, _, _, _, 1],
    [1, 1, 3, 3, _, _, 3, 3, 1, 3, 3, 1, 3, 1, 1, 1],
    [1, 1, 1, 3, _, _, 3, 1, 1, 1, 1, 1, 1, 1, 1, 1],
    [1, 3, 3, 4, _, _, 4, 3, 3, 3, 3, 3, 3, 3, 3, 1],
    [3, _, _, _, _, _, _, _, _, _, _, _, _, _, _, 3],
    [3, _, _, _, _, _, _, _, _, _, _, _, _, _, _, 3],
    [3, _, _, _, _, _, _, _, _, _, _, _, _, _, _, 3],
    [3, _, _, 5, _, _, _, 5, _, _, _, 5, _, _, _, 3],
    [3, _, _, _, _, _, _, _, _, _, _, _, _, _, _, 3],
    [3, _, _, _, _, _, _, _, _, _, _, _, _, _, _, 3],
    [3, _, _, _, _, _, _, _, _, _, _, _, _, _, _, 3],
    [3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3],
]


class Map:
    def _init_(self, game):
        self.game = game
        self.mini_map = mini_map
        self.world_map = {}
        self.rows = len(self.mini_map)
        self.cols = len(self.mini_map[0])
        self.get_map()
        self.minimap_texture = pg.Surface((300, 300))
        self.minimap_texture.fill('green')
        self.gen_minimap_texture()

    def gen_minimap_texture(self):
        rectSizeX = 300 / len(self.mini_map[0])
        rectSizeY = 300 / len(self.mini_map)
```

```
    for i in range(len(self.mini_map)):
        for j in range(len(self.mini_map[i])):
            if self.mini_map[i][j]:
                pg.draw.rect(
                    self.minimap_texture,
                    'darkgray',
                    (int(rectSizeX * j),
                     int(rectSizeY * i),
                     int(rectSizeX),
                     int(rectSizeY))
                )

def get_map(self):
    for j, row in enumerate(self.mini_map):
        for i, value in enumerate(row):
            if value:
                self.world_map[(i, j)] = value
    print(len(self.mini_map), len(self.mini_map[0]))

def draw(self):
    [pg.draw.rect(self.game.screen, 'darkgray', (pos[0] * 100,
                 pos[1] * 100, 100, 100), 2) for pos in self.world_map]
```

objecthandler.py

```
from sprite_object import *
from npc import *
from random import choices, randrange
class ObjectHandler:
    def _init_(self, game):
        self.game = game
        self.sprite_list = []
        self.npc_list = []
        self.npc_sprite_path = 'resources/sprites/npc/'
        self.static_sprite_path = 'resources/sprites/static_sprites/'
        self.anim_sprite_path = 'resources/sprites/animated_sprites/'
        add_sprite = self.add_sprite
        add_npc = self.add_npc
        self.npc_positions = {}

        # spawn npc
        self.npc_types = [SoldierNPC, CacoDemonNPC, CyberDemonNPC]
        self.weights = [70, 20, 10]
        self.restricted_area = {(i, j) for i in range(10) for j in range(10)}
        self.spawn_npc()

        # sprite map
        add_sprite(AnimatedSprite(game))
        add_sprite(AnimatedSprite(game, pos=(1.5, 1.5)))
        add_sprite(AnimatedSprite(game, pos=(1.5, 7.5)))
        add_sprite(AnimatedSprite(game, pos=(5.5, 3.25)))
        add_sprite(AnimatedSprite(game, pos=(5.5, 4.75)))
        add_sprite(AnimatedSprite(game, pos=(7.5, 2.5)))
        add_sprite(AnimatedSprite(game, pos=(7.5, 5.5)))
        add_sprite(AnimatedSprite(game, pos=(14.5, 1.5)))
        add_sprite(AnimatedSprite(game, pos=(14.5, 4.5)))
        add_sprite(
            AnimatedSprite(
                game,
                path=self.anim_sprite_path +
                'red_light/0.png',
                pos=(
                    14.5,
                    5.5)))
        add_sprite(
            AnimatedSprite(
                game,
                path=self.anim_sprite_path +
                'red_light/0.png',
                pos=(
                    14.5,
                    7.5)))
        add_sprite(
            AnimatedSprite(
                game,
                path=self.anim_sprite_path +
                'red_light/0.png',
                pos=(
                    12.5,
```

```
            7.5)))
        add_sprite(
            AnimatedSprite(
                game,
                path=self.anim_sprite_path +
                'red_light/0.png',
                pos=(
                    9.5,
                    7.5)))
        add_sprite(
            AnimatedSprite(
                game,
                path=self.anim_sprite_path +
                'red_light/0.png',
                pos=(
                    14.5,
                    12.5)))
        add_sprite(
            AnimatedSprite(
                game,
                path=self.anim_sprite_path +
                'red_light/0.png',
                pos=(
                    9.5,
                    20.5)))
        add_sprite(
            AnimatedSprite(
        game,
                path=self.anim_sprite_path +
                'red_light/0.png',
                pos=(
                    9.5,
                    20.5)))
        add_sprite(
            AnimatedSprite(
                game,
                path=self.anim_sprite_path +
                'red_light/0.png',
                pos=(
                    10.5,
                    20.5)))
        add_sprite(
            AnimatedSprite(
                game,
                path=self.anim_sprite_path +
                'red_light/0.png',
                pos=(
                    3.5,
                    14.5)))
        add_sprite(
            AnimatedSprite(
                game,
                path=self.anim_sprite_path +
                'red_light/0.png',
                pos=(
                    3.5,
```

```
                18.5)))
        add_sprite(AnimatedSprite(game, pos=(14.5, 24.5)))
        add_sprite(AnimatedSprite(game, pos=(14.5, 30.5)))
        add_sprite(AnimatedSprite(game, pos=(1.5, 30.5)))
        add_sprite(AnimatedSprite(game, pos=(1.5, 24.5)))

        # npc map
        # add_npc(SoldierNPC(game, pos=(11.0, 19.0)))
        # add_npc(SoldierNPC(game, pos=(11.5, 4.5)))
        # add_npc(SoldierNPC(game, pos=(13.5, 6.5)))
        # add_npc(SoldierNPC(game, pos=(2.0, 20.0)))
        # add_npc(SoldierNPC(game, pos=(4.0, 29.0)))
        # add_npc(CacoDemonNPC(game, pos=(5.5, 14.5)))
        # add_npc(CacoDemonNPC(game, pos=(5.5, 16.5)))
        # add_npc(CyberDemonNPC(game, pos=(14.5, 25.5)))

    def spawn_npc(self):
        for i in range(self.game.enemies):
            npc = choices(self.npc_types, self.weights)[0]
            pos = x, y = randrange(
                self.game.map.cols), randrange(
                self.game.map.rows)
while (
                    pos in self.game.map.world_map) or (
                    pos in self.restricted_area):
                pos = x, y = randrange(
                    self.game.map.cols), randrange(
                    self.game.map.rows)
            self.add_npc(npc(self.game, pos=(x + 0.5, y + 0.5)))

    def check_win(self):
        if not len(self.npc_positions):
            text = f"Level {self.game.level} completed"
            self.game.font_renderer.addTimeText(
                text, (WIDTH / 2, HEIGHT / 2), 2, (255, 255, 255), 120)
            pg.display.flip()
            count = 5 + self.game.level * 2
            self.game.enemies = min(20, count)
            self.game.level += 1
            self.npc_list = []
            self.spawn_npc()

    def update(self):
        self.npc_positions = {
            npc.map_pos for npc in self.npc_list if npc.alive}
        [sprite.update() for sprite in self.sprite_list]
        [npc.update() for npc in self.npc_list]
        self.check_win()

    def add_npc(self, npc):
        self.npc_list.append(npc)

    def add_sprite(self, sprite):
        self.sprite_list.append(sprite)
```

npc.py

```
from sprite_object import *
from random import randint, random


class NPC(AnimatedSprite):
    def _init_(
            self,
            game,
            path='resources/sprites/npc/soldier/0.png',
            pos=(
                10.5,
                5.5),
            scale=0.6,
            shift=0.38,
            animation_time=180):
        super()._init_(game, path, pos, scale, shift, animation_time)
        self.attack_images = self.get_images(self.path + '/attack')
        self.death_images = self.get_images(self.path + '/death')
        self.idle_images = self.get_images(self.path + '/idle')
        self.pain_images = self.get_images(self.path + '/pain')
        self.walk_images = self.get_images(self.path + '/walk')

        self.attack_dist = randint(3, 6)
        self.speed = 0.03
        self.size = 20
        self.health = 100
        self.attack_damage = 10
        self.accuracy = 0.15
        self.alive = True
        self.pain = False
        self.ray_cast_value = False
        self.frame_counter = 0
        self.player_search_trigger = False
        self.points = 100

    def update(self):
        self.check_animation_time()
        self.get_sprite()
        self.run_logic()
        # self.draw_ray_cast()

    def check_wall(self, x, y):
        return (x, y) not in self.game.map.world_map

    def check_wall_collision(self, dx, dy):
        if self.check_wall(int(self.x + dx * self.size), int(self.y)):
            self.x += dx
        if self.check_wall(int(self.x), int(self.y + dy * self.size)):
            self.y += dy
```

```
def movement(self):
    next_pos = self.game.pathfinding.get_path(
        self.map_pos, self.game.player.map_pos)
    next_x, next_y = next_pos

    # pg.draw.rect(self.game.screen, 'blue', (100 * next_x, 100 * next_y, 100, 100))
    if next_pos not in self.game.object_handler.npc_positions:
        angle = math.atan2(next_y + 0.5 - self.y, next_x + 0.5 - self.x)
        dx = math.cos(angle) * self.speed
        dy = math.sin(angle) * self.speed
        self.check_wall_collision(dx, dy)

def attack(self):
    if self.animation_trigger:
        self.game.sound.npc_shot.play()
        if random() < self.accuracy:
            self.game.player.get_damage(self.attack_damage)

def animate_death(self):
    if not self.alive:
        if self.game.global_trigger and self.frame_counter < len(
                self.death_images) - 1:
            self.death_images.rotate(-1)
            self.image = self.death_images[0]
            self.frame_counter += 1

def animate_pain(self):
    self.animate(self.pain_images)
    if self.animation_trigger:
        self.pain = False

def check_hit_in_npc(self):
    if self.ray_cast_value and self.game.player.shot:
        if HALF_WIDTH - self.sprite_half_width < self.screen_x < HALF_WIDTH + \
                self.sprite_half_width:
            self.game.sound.npc_pain.play()
            self.game.player.shot = False
            self.pain = True
            self.health -= self.game.weapon.damage
            self.check_health()

def check_health(self):
    if self.health < 1:
        self.alive = False
        self.game.sound.npc_death.play()
        self.game.score += self.points
        self.game.enemies -= 1

def run_logic(self):
```

```
        if self.alive:
            self.ray_cast_value = self.ray_cast_player_npc()
            self.check_hit_in_npc()

            if self.pain:
                self.animate_pain()

            elif self.ray_cast_value:
                self.player_search_trigger = True

                if self.dist < self.attack_dist:
                    self.animate(self.attack_images)
                    self.attack()
                else:
                    self.animate(self.walk_images)
                    self.movement()

            elif self.player_search_trigger:
                self.animate(self.walk_images)
                self.movement()

            else:
                self.animate(self.idle_images)
        else:
            self.animate_death()

    @property
    def map_pos(self):
        return int(self.x), int(self.y)

    def ray_cast_player_npc(self):
        if self.game.player.map_pos == self.map_pos:
            return True

        wall_dist_v, wall_dist_h = 0, 0
        player_dist_v, player_dist_h = 0, 0

        ox, oy = self.game.player.pos
        x_map, y_map = self.game.player.map_pos

        ray_angle = self.theta

        sin_a = math.sin(ray_angle)
        cos_a = math.cos(ray_angle)

        # horizontals
        y_hor, dy = (y_map + 1, 1) if sin_a > 0 else (y_map - 1e-6, -1)

        depth_hor = (y_hor - oy) / sin_a
```