Дніпровський державний технічний університет

Кафедра Програмне забезпечення систем

КУРСОВА РОБОТА

з дисципліни *Об’єктно-орієнтоване програмування*

на тему  *Розробка інформаційного та програмного*

*забезпечення задачі комп’ютерної гри <Шутера>*

**Студента** ІI курсу ПЗ-19-1ду

Спеціальності 121 – Інженерія програмного

забезпечення

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ **Боюн А.В.**

Кандидат технічних наук, доцент

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ **Яшина К.В.**

Національна шкала \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Кількість балів: \_\_\_\_\_\_\_\_\_\_\_\_ Оцінка: ECTS \_\_\_\_\_\_\_

Члени комісії \_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ проф., д.т.н. Шумейко О. О.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ к.т.н. Дранишников Л.В.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ к.т.н. Бабенко М. В.

м. Кам`янське – 2020 р.

Дніпровський державний технічний університет

Факультет *Електроніки та комп’ютерної техніки* Кафедра *ПЗ*

Галузь знань *0501 – Інформатика та обчислювальна техніка*

Спеціальності *121 – Інженерія програмного забезпечення*

Затверджую

Завідувач кафедри

\_\_\_\_\_\_\_\_\_ Шумейко О.О.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

ЗАВДАННЯ

на курсову роботу

Студенту групи  *ПЗ-19-1ду Боюну Артуру Валерыйовичу*

Тема курсової роботи  *Розробка інформаційного та програмного*

*забезпечення задачі комп’ютерної гри <Шутера>*

затверджено наказом по ДДТУ від \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ № \_\_\_\_\_\_\_\_\_

Строк подання студентом закінченої роботи \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Завдання на окремі частини роботи

|  |  |  |  |
| --- | --- | --- | --- |
| Найменування частини | Завдання | Консультант | Підпис |
| Загальна | *Аналіз предметної області* | К.В. Яшина |  |
|  |  |  |  |
| Спеціальна | *Опис програми* | К.В. Яшина |  |
|  |  |  |  |

Перелік ілюстрованих матеріалів \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Календарний план**

|  |  |
| --- | --- |
| Етап роботи | Строк виконання |
| *Аналіз предметної області і складання аналітичного огляду* | *2020-03-20* |
| *Розробка архітектури програмного забезпечення* | *2020-03-25* |
| *Розробка інтерфейсної частини* | *2020-04-05* |
| *Написання програмного коду* | *2020-04-30* |
| *Оформлення роботи* | *2020-05-10* |
| *Підготовка ілюстративних матеріалів* | *2020-05-24* |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

Дата видачі завдання *2020-03-01*

Завдання видав

керівник курсової роботи \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ К.В. Яшина

Завдання прийняв до виконання

Студент \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Боюн А.В.

Дніпровський державний технічний університет

**Студент**  Боюн А.В. **групи** ПЗ-19-1ду

**Тема роботи**  *Розробка інформаційного та програмного*

*забезпечення задачі комп’ютерної гри<Шутера>*

Оцінка за роботу

|  |  |  |
| --- | --- | --- |
| Критерії оцінки | Максимальна оцінка | Оцінка, балів |
| **ОЦІНКА ЕТАПУ ПІДГОТОВКИ РОБОТИ** | **20 б.** |  |
| Дотримання календарного плану | 5 б. |  |
| Відповідність роботи постановці задачі | 5 б. |  |
| Самостійність | 5 б. |  |
| Ініціативність | 5 б. |  |
| **ОЦІНКА ЗА ЗВІТ І** | **20 б.** |  |
| Структура, об’єм, оформлення | 5 б. |  |
| Реферат | 5 б. |  |
| Вступ, постановка задачі, висновки, перелік посилань | 5 б. |  |
| Опис предметної області (задачі) | 5 б. |  |
| **ОЦІНКА ЗА ЗВІТ ІІ** | **40 б.** |  |
| Логічна структура (класи, об’єкти) | 10 б. |  |
| Опис програмного продукту | 10 б. |  |
| Функціональна повнота | 5 б. |  |
| Інтерфейсна частина | 5 б. |  |
| Опис тестування | 5 б. |  |
| Керівництво користувача | 5 б. |  |
| **ОЦІНКА ЗА ПРАКТИЧНУ РЕАЛІЗАЦІЮ** | **10 б.** |  |
| Степінь закінченості | 5 б. |  |
| Демонстрація | 5 б. |  |
| **РІВЕНЬ СКЛАДНОСТІ (наявність спеціалізації)** | **10 б.** |  |
|  |  |  |
| **РАЗОМ** | **100 б.** |  |

Захист відбувся \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Загальна оцінка \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Керівник курсової роботи \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Яшина К.В

**ЗМІСТ**

**Вступ**

1. **АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ**

2. **ОПИС ПРОГРАМИ**

2.1. Система керування версіями

2.2. Постановка задачі

2.3. Діаграми ієрархії класів

2.4. Опис класів і об’єктів

2.5. Початок гри

2.6. Керівництво користувача

**Висновок**

**Перелік посилань**

**Додатки**

ДОДАТОК А

ДОДАТОК Б

Пояснювальна записка 55с., 6рис., 2 додатки, 5 джерел.

**Об’єкт дослідження**: об’єктно-орієнтоване програмування, мова програмування Python.

**Мета роботи**: закріплення, поглиблення та узагальнення знань, які були здобуті під час вивчення курсу, набуття навичок використання основ алгоритмізації та програмування на алгоритмічних мовах високого рівня з використанням принципів об’єктно-орієнтованого програмування.

Написання цієї курсової роботи мало на меті закріплення знань, отриманих при вивченні теоретичної частини курсу і набуття практичних навичок розробки об’єктно-орієнтованих програм. В результаті виконання курсової роботи студент повинен засвоїти:

* Концепції об’єктно-орієнтованого програмування;
* Спеціалізовані засоби розробки програмного забезпечення мовою Python;
* Аналіз вимог до програмного забезпечення;
* Основи уніфікованої мови моделювання UML.

Курсова робота складається з двох частин. В першій частині побудована детальна об’єктна модель гри. Друга частина курсової роботи містить безпосередньо розробку об’єктно-орієнтованого програмування програмного забезпечення та розроблену об’єктну модель.

Додаток розроблений за допомогою середовища розробки JetBrains PyCharm Professional 2020 та бібліотеки PyGame.

Вступ

Метою данного курсового проєкту є створення програмного продукту, що представляє собою гру. Область застосування: дозвілля. Гра дає можливість чудово продемонструвати всі можливості сучасного підходу об’єктно-орієнтованого програмування на най популярнішій мові програмування Python, основою якої є ООП.   
Перед тим, як почасти працювати над програмою, необхідно сформулювати основні вимоги до програми.

Програмний продукт реалізує такі вимоги:

* Створення стейт менеджера для;
* Перемикати сцени за допомогою стейт менеджера;
* Генерація ворогів;
* Створення гравця;
* Реалізацію руху гравця та ворогів;
* Можливість стріляти у ворогів;
* Обробка ігрових ситуацій.

Таким чином наша програма не призначена для гри користувача зі штучним інтелектом. За допомогою цієї програми можна грати, та вдосконалювати реакцію.  
Розробка велась мовою Python. за допомогою та з використанням бібліотеки pygame.

**Pygame —** набір крос-платформових модулів для Python, призначених для створення відеоігор. Включає в себе бібліотеки комп'ютерної графіки і звуку.

Підтримує Linux (pygame входить до складу найпопулярніших його дистрибутивів), Windows (95, 98, me, 2000, XP, Vista, 7, 8, 10), Windows CE, BeOS, MacOS, Mac OS X, FreeBSD, NetBSD, OpenBSD, BSD/OS, Solaris, IRIX, та QNX.

Функції ядра бібліотеки написані на С та Assembly. Це робить бібліотеку ще швидшою, через те, що С код зазвичай в 10-20 разів швидший за Python, а Assembly в свою чергу в 100 раз швидший, ніж Python.

Також pygame не потребує наявності OpenGL. Тому, що OpenGL створює більше проблем, ніж вирішує.

1. **Аналіз предметної області**

Тип цієї гри відомий як Top-Down Shooter. Шу́тер (англ. shooter game, від shooter — «стрілець») — жанр відеоігор, піджанр action, основу ігрового процесу якого складає стрільба зі зброї по цілях, зазвичай вороже налаштованих. Часто основним завданням гравця є знищення усіх цілей на рівні, або/та набір найбільшої кількості очок, а не проходження сюжету.

1. **Опис програми**
   1. *Система керування версіями*

Під час розробки даного проекту використовувалася система керування версіями Git.

Git — розподілена система керування версіями файлів та спільної роботи. Проект створив Лінус Торвальдс для управління розробкою ядра Linux, а сьогодні підтримується Джуніо Хамано. Git є однією з найефективніших, надійних і високопродуктивних систем керування версіями, що надає гнучкі засоби нелінійної розробки, що базуються на відгалуженні і злитті гілок. Для забезпечення цілісності історії та стійкості до змін заднім числом використовуються криптографічні методи, також можлива прив'язка цифрових підписів розробників до тегів і комітів.

Система спроектована як набір програм, спеціально розроблених з врахуванням їхнього використання у скриптах. Це дозволяє зручно створювати спеціалізовані системи управління версіями на базі Git або користувацькі інтерфейси.

* 1. *Постановка задачі*

Гра: шутер   
Мета гри: знищення усіх цілей на рівні.

*2.3. Діаграми ієрархії класів*

Провівши аналіз залежностей між класами на предмет виявлення зв’язків типу «це є узагальненням того» можна побудувати діаграму успадкування (ієрархію) та список класів, представлену на рис. 2.1.

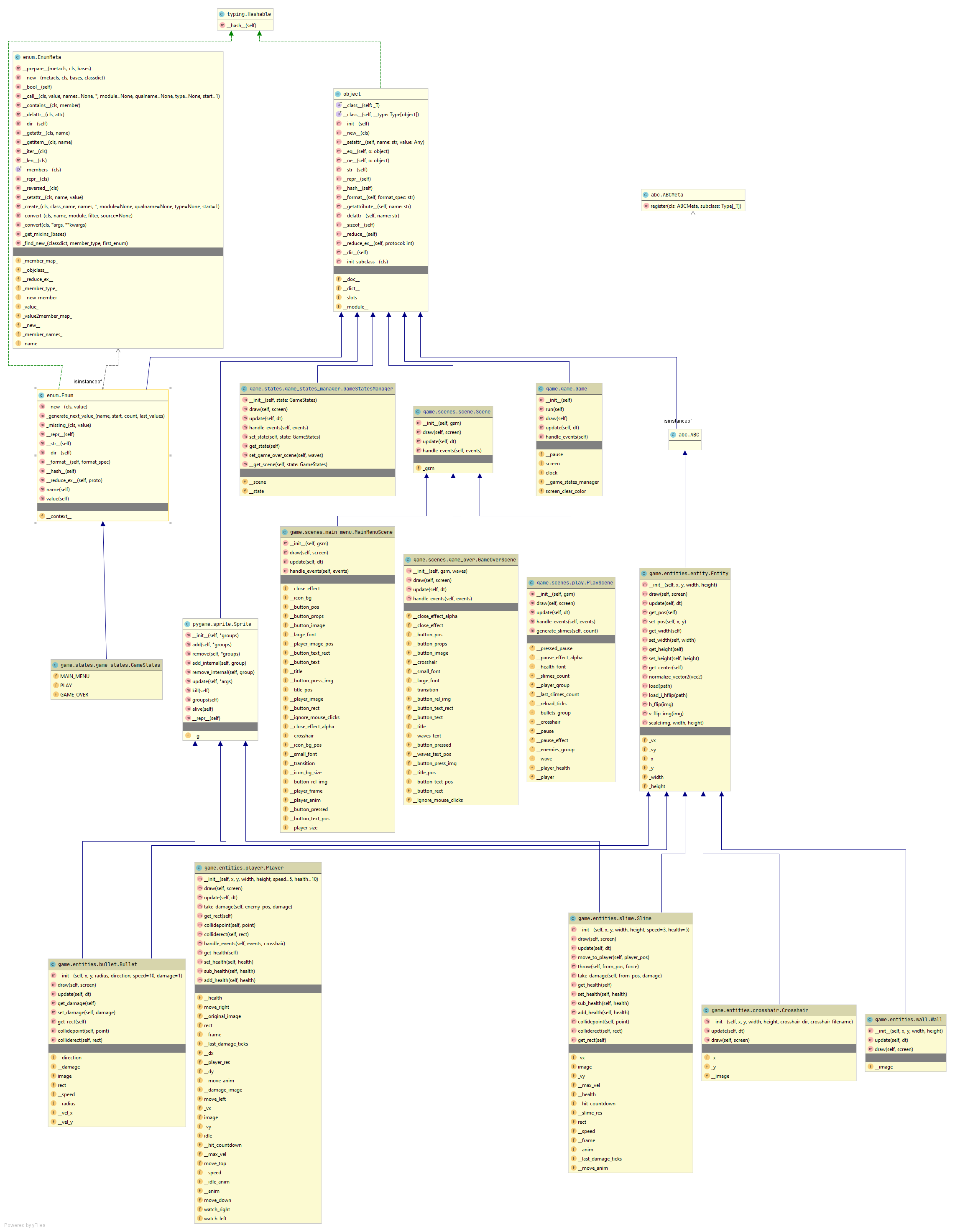


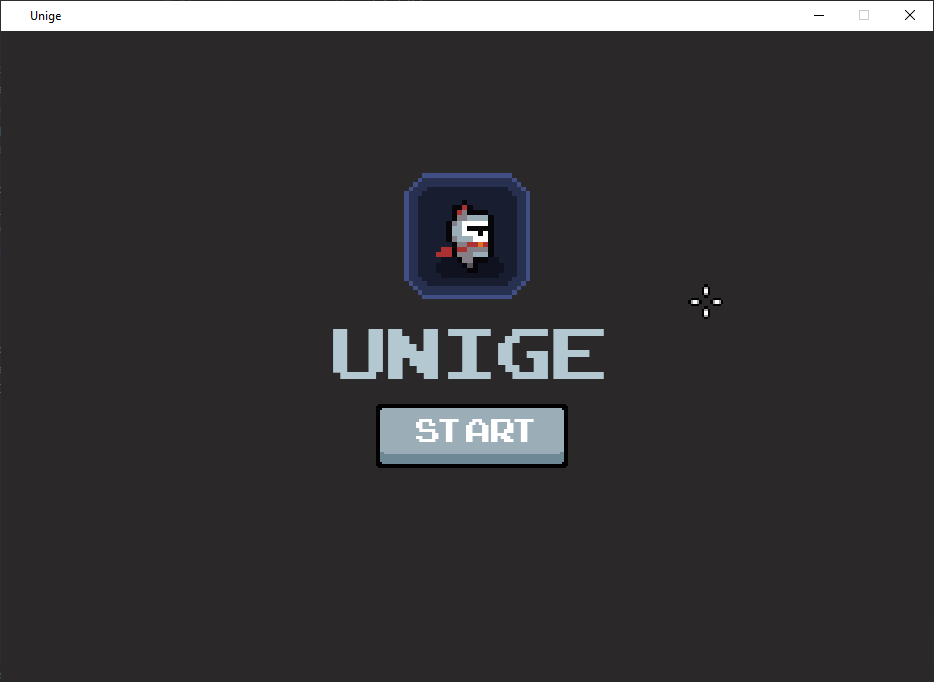
Рис. 2.1. - Діаграма та список класів

* 1. *Опис класів та об’єктів*

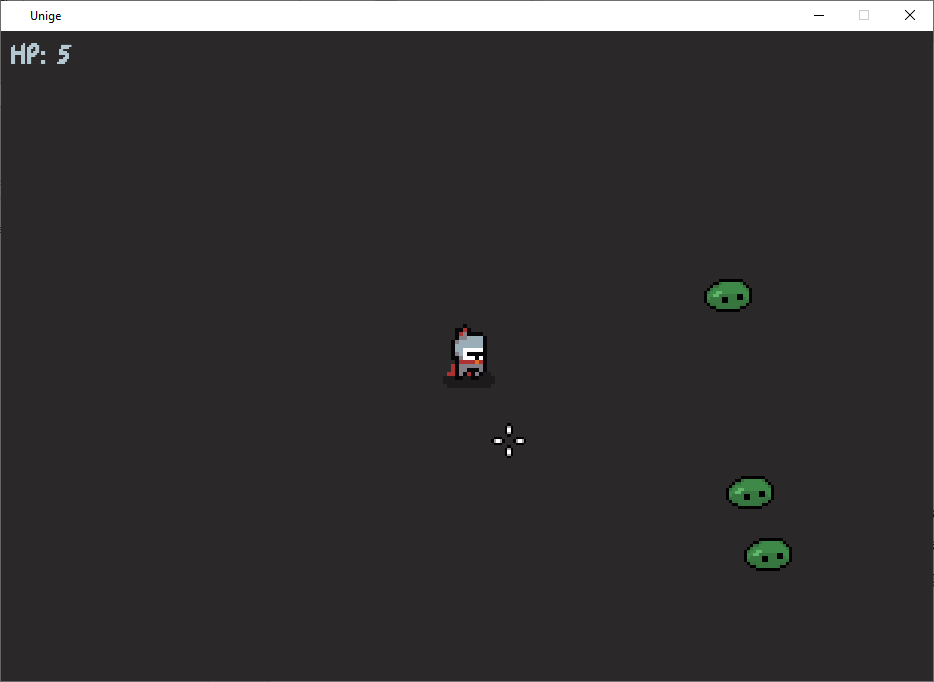
Було виділено наступні класи:

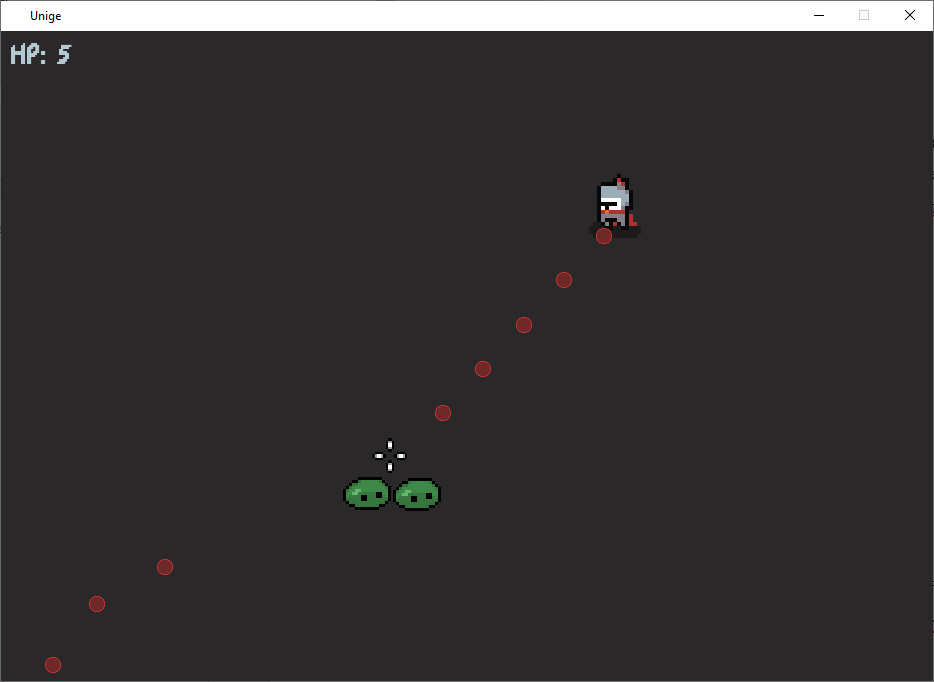
* Game – основний клас, який ініціює початок гри, та стейт менеджера.
* GameStatesManager – красс за допомогою якого перемикаються сцени згідно трейту.
* Entity – клас на від якого наслідуються інщі динамічні класи. На приклад Player
* Player – клас гравця, в якому оброблюється все що стосується гравця.
* Bullet – клас кулі яку гравець создає, коли зажимає луву кнопку мищі, на PlayScene.
* Crosshair – клас курсору в якому оброблюється все, що його стосується.
* Slime - клас слизня, який э ворогом гравця, в ньому оброблюється все що стосується слизня.
* Scene – це абстрактний клас сцени гри, якого наслідують інщі сцени.
* MainMenuScene – клас головного меню гри.
* PlayScene – клас який наслідує клас Scene, та виконує функцію основної сцени гри.
* GameOverScene – клас кінця гри, який наслідує клас Scene, та демонструє гравцю його результат.
  1. *Початок гри*

На початку гри завантажується головне меню.



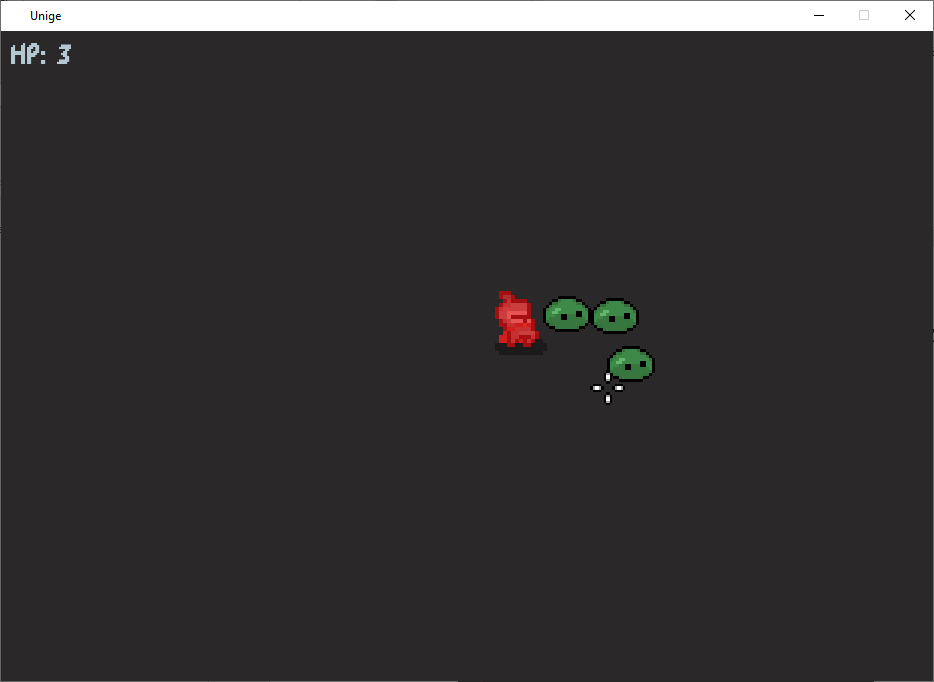
У головному меню гравець бачить анімацію гравця, курсор, назву гри та кнопку початку гри. Після натиску на кнопку Start гравця буде плавно перенесено до головної сцени гри, де майже від разу почнется генерація ворогів.





У правому лівому краю екрану гри, відображуеться, кількість здоровя гравця (далі HP).

Кожен раз, коли гравець торкається до ворога, його HP зменшується на одиницю.



Коли HP гравця закінчуються, гра відправляє гравця до сцени кінця гри (GameOverScene).

* 1. *Керівництво користувача*
* Запустити програму.
* Натиснути кнопку “START”.

**ВИСНОВОК**

В результаті виконаних робіт був отриманий програмний продукт комп’ютерної гри “Шутера” і комплект документації до нього. Програма реалізовувалася повністю з відповідністю технічного завдання.

У процесі виконання робіт були закріплені навички роботи з мовою високого рівня програмування Python та бібліотеки PyGame і отриманий досвід розробки програмного продукту та документації до нього.

Під час виконання курсової роботи було продемонстровано:

* вміння збирати і аналізувати відповідні матеріали про об’єкт дослідження, використовуючи сучасні джерела інформації, включаючи Інтернет ресурси;
* спроможність проводити необхідні обґрунтування для розробки програмних засобів систем управляння різного призначення;
* здатність доводити розв’язання поставленої задачі до кінця;
* вміння аналізувати отримані результати і робити з них висновки.

**ПЕРЕЛІК ПОСИЛАНЬ**

1. Python CookbookDavid Beazley & Brian K. Jones (O’Reilly, 3rd edition, 2013)
2. Making Games with Python & Pygame By Al Sweigart (2012 by Albert Sweigart)
3. Cracking Codes with Python: An Introduction to Building and Breaking Ciphers (No Starch Press January 23, 2018)
4. Python Crash Course, 2nd Edition: A Hands-On, Project-Based Introduction to Programming (Englisch) Taschenbuch – 3. Mai 2019
5. Линейная алгебра для разработчиков игр (<https://habr.com/ru/post/131931/>)
6. Посилання на вихідний код на платформі GitHub: <https://github.com/archieruin/Unige>

**Додаток А**

Лістинг програми

**\_\_main\_\_.py**

from game.game import Game

if \_\_name\_\_ == '\_\_main\_\_':

game = Game()

**settings.py**

from pathlib import Path

GAME\_TITLE = 'Unige'

FPS = 60

SCREEN\_WIDTH = 932

SCREEN\_HEIGHT = 650

root\_dir = Path.cwd()

src\_dir = root\_dir / 'game/'

res\_dir = root\_dir / 'res/'

ui\_res = res\_dir / 'ui/'

fonts\_res = res\_dir / 'fonts/'

player\_res = res\_dir / 'player/'

bullets\_res = res\_dir / 'bullets/'

slime\_res = res\_dir / 'enemies/' / 'slime/'

**Додаток Б**

Лістинг класів

**game.py**

import pygame

from game import settings

from game.states.game\_states\_manager import GameStatesManager

from game.states.game\_states import GameStates

class Game:

"""Класс отвечающий за запуск игры и игровой цикл."""

def \_\_init\_\_(self):

pygame.init()

pygame.mouse.set\_visible(False)

self.screen\_clear\_color = (42, 40, 40)

self.clock = pygame.time.Clock()

self.screen = pygame.display.set\_mode((settings.SCREEN\_WIDTH, settings.SCREEN\_HEIGHT))

pygame.display.set\_caption(settings.GAME\_TITLE)

game\_icon = pygame.image.load(str(settings.ui\_res / 'icon.png')).convert()

pygame.display.set\_icon(game\_icon)

# Inti game states manager

self.\_\_game\_states\_manager = GameStatesManager(GameStates.MAIN\_MENU)

self.\_\_pause = False

# Run game loop

self.run()

def run(self):

while True:

dt = self.clock.tick(settings.FPS) / 1000

self.update(dt)

self.draw()

def draw(self):

pygame.display.update()

self.screen.fill(self.screen\_clear\_color)

self.\_\_game\_states\_manager.draw(self.screen)

def update(self, dt):

self.handle\_events()

self.\_\_game\_states\_manager.update(dt)

def handle\_events(self):

events = pygame.event.get()

for event in events:

if event.type == pygame.QUIT:

pygame.quit()

quit()

self.\_\_game\_states\_manager.handle\_events(events)

if \_\_name\_\_ == '\_\_main\_\_':

game = Game()

**game\_states\_manager.py**

from game.scenes.game\_over import GameOverScene

from game.scenes.main\_menu import MainMenuScene

from game.scenes.play import PlayScene

from game.scenes.scene import Scene

from game.states.game\_states import GameStates

class GameStatesManager:

def \_\_init\_\_(self, state: GameStates):

self.\_\_state = state

self.\_\_scene = self.set\_state(self.\_\_state)

def draw(self, screen):

self.\_\_scene.draw(screen)

def update(self, dt):

self.\_\_scene.update(dt)

def handle\_events(self, events):

self.\_\_scene.handle\_events(events)

def set\_state(self, state: GameStates):

self.\_\_scene = self.\_\_get\_scene(state)

return self.\_\_scene

def get\_state(self):

return self.\_\_state

def set\_game\_over\_scene(self, waves):

self.\_\_state = GameStates.GAME\_OVER

self.\_\_scene = GameOverScene(self, waves)

def \_\_get\_scene(self, state: GameStates):

if state == GameStates.MAIN\_MENU:

self.\_\_state = GameStates.MAIN\_MENU

return MainMenuScene(self)

elif state == GameStates.PLAY:

self.\_\_state = GameStates.PLAY

return PlayScene(self)

**game\_states.py**

from enum import Enum

class GameStates(Enum):

MAIN\_MENU = 0

PLAY = 1

GAME\_OVER = 3

**scene.py**

class Scene(object):

def \_\_init\_\_(self, gsm):

self.\_gsm = gsm

def draw(self, screen):

raise NotImplementedError

def update(self, dt):

raise NotImplementedError

def handle\_events(self, events):

raise NotImplementedError

**main\_menu.py**

import pygame

from game import settings

from game.entities.crosshair import Crosshair

from game.scenes.scene import Scene

from game.states.game\_states import GameStates

class MainMenuScene(Scene):

def \_\_init\_\_(self, gsm):

super().\_\_init\_\_(gsm)

# Crosshair

self.\_\_crosshair = Crosshair(settings.SCREEN\_WIDTH / 2,

settings.SCREEN\_HEIGHT / 2,

35, 35,

settings.ui\_res,

'crosshair.png')

# Player icon

self.\_\_player\_size = 84

self.\_\_icon\_bg\_size = 144

self.\_\_icon\_bg = pygame.transform.scale(

pygame.image.load(str(settings.ui\_res / 'icon\_bg.png')), (self.\_\_icon\_bg\_size, self.\_\_icon\_bg\_size))

self.\_\_player\_anim = [

pygame.image.load(str(settings.player\_res / 'knight\_run\_anim\_f0.png')),

pygame.image.load(str(settings.player\_res / 'knight\_run\_anim\_f1.png')),

pygame.image.load(str(settings.player\_res / 'knight\_run\_anim\_f2.png')),

pygame.image.load(str(settings.player\_res / 'knight\_run\_anim\_f3.png')),

pygame.image.load(str(settings.player\_res / 'knight\_run\_anim\_f4.png')),

pygame.image.load(str(settings.player\_res / 'knight\_run\_anim\_f5.png'))

]

for i, img in enumerate(self.\_\_player\_anim):

self.\_\_player\_anim[i] = pygame.transform.scale(img, (self.\_\_player\_size, self.\_\_player\_size))

self.\_\_player\_frame = 0

self.\_\_player\_image = self.\_\_player\_anim[int(self.\_\_player\_frame)]

# Title and Font

font\_path = str(settings.fonts\_res / 'Arcade Classic.ttf')

self.\_\_large\_font = pygame.font.Font(font\_path, 100)

self.\_\_title = self.\_\_large\_font.render("Unige", True, (180, 200, 210))

# Start text

self.\_\_small\_font = pygame.font.Font(font\_path, 45)

self.\_\_button\_text = self.\_\_small\_font.render("Start", True, (255, 255, 255))

# Play Button

button\_press\_path = str(settings.ui\_res / "menu\_button\_press.png")

button\_rel\_path = str(settings.ui\_res / "menu\_button.png")

self.\_\_button\_props = (192, 64)

self.\_\_button\_press\_img = pygame.transform.scale(pygame.image.load(button\_press\_path), self.\_\_button\_props)

self.\_\_button\_rel\_img = pygame.transform.scale(pygame.image.load(button\_rel\_path), self.\_\_button\_props)

self.\_\_button\_image = self.\_\_button\_rel\_img

self.\_\_button\_pressed = False

# Positions

self.\_\_icon\_bg\_pos = (settings.SCREEN\_WIDTH / 2 - self.\_\_icon\_bg\_size / 2,

settings.SCREEN\_HEIGHT / 2 - self.\_\_icon\_bg\_size / 2 - 120)

self.\_\_player\_image\_pos = (settings.SCREEN\_WIDTH / 2 - self.\_\_player\_size / 2,

settings.SCREEN\_HEIGHT / 2 - self.\_\_player\_size / 2 - 120)

self.\_\_title\_pos = (settings.SCREEN\_WIDTH / 2 - self.\_\_title.get\_width() / 2 + 4,

settings.SCREEN\_HEIGHT / 2 - self.\_\_title.get\_height() / 2)

self.\_\_button\_pos = (settings.SCREEN\_WIDTH / 2 - self.\_\_button\_props[0] / 2 + 5,

settings.SCREEN\_HEIGHT / 2 - self.\_\_button\_props[1] / 2 + 80)

self.\_\_button\_text\_pos = (self.\_\_button\_pos[0] + self.\_\_button\_text.get\_width() / 2 - 22,

self.\_\_button\_pos[1] + 4)

# Rects

self.\_\_button\_rect = self.\_\_button\_image.get\_rect(topleft=self.\_\_button\_pos)

self.\_\_button\_text\_rect = self.\_\_button\_text.get\_rect(center=self.\_\_button\_text\_pos)

# Effects

self.\_\_ignore\_mouse\_clicks = False

self.\_\_transition = False

self.\_\_close\_effect = pygame.Surface((settings.SCREEN\_WIDTH, settings.SCREEN\_HEIGHT))

self.\_\_close\_effect\_alpha = 0

self.\_\_close\_effect.set\_alpha(self.\_\_close\_effect\_alpha)

self.\_\_close\_effect.fill((10, 10, 10))

def draw(self, screen):

screen.blit(self.\_\_icon\_bg, self.\_\_icon\_bg\_pos)

screen.blit(self.\_\_player\_image, self.\_\_player\_image\_pos)

screen.blit(self.\_\_title, self.\_\_title\_pos)

screen.blit(self.\_\_button\_image, self.\_\_button\_pos)

screen.blit(self.\_\_button\_text, self.\_\_button\_text\_pos)

self.\_\_crosshair.draw(screen)

if self.\_\_transition:

screen.blit(self.\_\_close\_effect, (0, 0))

def update(self, dt):

self.\_\_player\_frame += 0.15

if self.\_\_player\_frame >= len(self.\_\_player\_anim):

self.\_\_player\_frame = 0

self.\_\_player\_image = self.\_\_player\_anim[int(self.\_\_player\_frame)]

self.\_\_crosshair.update(dt)

if self.\_\_transition:

self.\_\_close\_effect\_alpha += 10

self.\_\_close\_effect.set\_alpha(self.\_\_close\_effect\_alpha)

if self.\_\_close\_effect\_alpha > 255 + 300:

self.\_gsm.set\_state(GameStates.PLAY)

def handle\_events(self, events):

keys = pygame.key.get\_pressed()

if keys[pygame.K\_SPACE] == 1:

if not self.\_\_button\_pressed:

self.\_\_ignore\_mouse\_clicks = True

self.\_\_button\_image = self.\_\_button\_press\_img

self.\_\_button\_pressed = True

self.\_\_button\_text\_pos = (self.\_\_button\_pos[0] + self.\_\_button\_text.get\_width() / 2 - 22,

self.\_\_button\_pos[1] + 10)

self.\_\_transition = True

if not self.\_\_ignore\_mouse\_clicks:

press = pygame.mouse.get\_pressed()

mouse\_pos = pygame.mouse.get\_pos()

collude\_button = self.\_\_button\_rect.collidepoint(mouse\_pos) or \

self.\_\_button\_text\_rect.collidepoint(mouse\_pos)

if press[0]:

if collude\_button and not self.\_\_button\_pressed:

self.\_\_button\_image = self.\_\_button\_press\_img

self.\_\_button\_pressed = True

self.\_\_button\_text\_pos = (self.\_\_button\_pos[0] + self.\_\_button\_text.get\_width() / 2 - 22,

self.\_\_button\_pos[1] + 10)

else:

if self.\_\_button\_pressed:

self.\_\_button\_image = self.\_\_button\_rel\_img

self.\_\_button\_pressed = False

self.\_\_button\_text\_pos = (self.\_\_button\_pos[0] + self.\_\_button\_text.get\_width() / 2 - 22,

self.\_\_button\_pos[1] + 5)

if collude\_button:

self.\_\_transition = True

**play.py**

import random

import pygame

from pygame.sprite import Group

from game import settings

from game.entities.bullet import Bullet

from game.entities.crosshair import Crosshair

from game.entities.player import Player

from game.entities.slime import Slime

from game.scenes.scene import Scene

class PlayScene(Scene):

def \_\_init\_\_(self, gsm):

super().\_\_init\_\_(gsm)

self.\_\_reload\_ticks = pygame.time.get\_ticks()

# Pause

self.\_\_pressed\_pause = False

self.\_\_pause = False

self.\_\_pause\_effect = pygame.Surface((settings.SCREEN\_WIDTH, settings.SCREEN\_HEIGHT))

self.\_\_pause\_effect\_alpha = 0

self.\_\_pause\_effect.set\_alpha(self.\_\_pause\_effect\_alpha)

self.\_\_pause\_effect.fill((0, 0, 0))

# Crosshair

self.\_\_crosshair = Crosshair(settings.SCREEN\_WIDTH / 2,

settings.SCREEN\_HEIGHT / 2,

35, 35,

settings.ui\_res,

'crosshair.png')

# Groups

self.\_\_player\_group = Group()

self.\_\_enemies\_group = Group()

self.\_\_bullets\_group = Group()

# Player

self.\_\_player = Player(settings.SCREEN\_WIDTH / 2 - 32,

settings.SCREEN\_HEIGHT / 2 - 32,

64, 64,

speed=6,

health=5)

self.\_\_player\_group.add(self.\_\_player)

# Font

font\_path = str(settings.fonts\_res / 'dpcomic.ttf')

self.\_\_health\_font = pygame.font.Font(font\_path, 32)

health\_text = "HP: %d" % self.\_\_player.get\_health()

self.\_\_player\_health = self.\_\_health\_font.render(health\_text, True, (180, 200, 210))

# Game

self.\_\_wave = 1

self.\_\_slimes\_count = 3

self.\_\_last\_slimes\_count = self.\_\_slimes\_count

self.generate\_slimes(self.\_\_slimes\_count)

def draw(self, screen):

self.\_\_enemies\_group.draw(screen)

self.\_\_player\_group.draw(screen)

self.\_\_bullets\_group.draw(screen)

# UI

screen.blit(self.\_\_player\_health, (10, 10))

if self.\_\_pause:

screen.blit(self.\_\_pause\_effect, (0, 0))

self.\_\_crosshair.draw(screen)

def update(self, dt):

if not self.\_\_pause:

# Update Groups

for enemy in self.\_\_enemies\_group.sprites():

enemy.move\_to\_player(self.\_\_player.get\_pos())

for other\_enemy in self.\_\_enemies\_group.sprites():

if enemy.colliderect(other\_enemy.get\_rect()):

enemy.throw(other\_enemy.get\_pos(), 1)

for bullet in self.\_\_bullets\_group.sprites():

if bullet.colliderect(enemy.get\_rect()):

enemy.take\_damage(self.\_\_player.get\_center(), bullet.get\_damage())

bullet.kill()

# Check collides between slimes and player

if self.\_\_player.colliderect(enemy.get\_rect()):

self.\_\_player.take\_damage(enemy.get\_pos(), 1)

self.\_\_bullets\_group.update(dt)

self.\_\_player\_group.update(dt)

self.\_\_enemies\_group.update(dt)

# Update player health points

player\_hp = self.\_\_player.get\_health()

health\_text = f"HP: {player\_hp}"

self.\_\_player\_health = self.\_\_health\_font.render(health\_text, True, (180, 200, 210))

self.\_\_slimes\_count = len(self.\_\_enemies\_group.sprites())

if self.\_\_slimes\_count <= 0:

self.generate\_slimes(self.\_\_last\_slimes\_count + 3)

self.\_\_wave += 1

if player\_hp <= 0:

self.\_gsm.set\_game\_over\_scene(self.\_\_wave)

else:

# Pause Effect

if self.\_\_pause\_effect\_alpha <= 128:

self.\_\_pause\_effect\_alpha += 5

self.\_\_pause\_effect.set\_alpha(self.\_\_pause\_effect\_alpha)

self.\_\_crosshair.update(dt)

def handle\_events(self, events):

if not self.\_\_pause:

self.\_\_player.handle\_events(events, self.\_\_crosshair)

for event in events:

if event.type == pygame.KEYUP:

if event.key == pygame.K\_p:

if self.\_\_pause:

self.\_\_pause = False

self.\_\_pause\_effect\_alpha = 0

else:

self.\_\_pause = True

if event.type == pygame.MOUSEBUTTONUP:

player\_x = self.\_\_player.get\_center()[0]

player\_y = self.\_\_player.get\_center()[1]

mouse\_pos = pygame.mouse.get\_pos()

direction = (player\_x - mouse\_pos[0], player\_y - mouse\_pos[1])

bullet = Bullet(player\_x, player\_y, 16, direction, speed=10, damage=1)

self.\_\_bullets\_group.add(bullet)

mouse\_pressed = pygame.mouse.get\_pressed()

if mouse\_pressed[0]:

reload\_time = (pygame.time.get\_ticks() - self.\_\_reload\_ticks) / 1000

if reload\_time > 0.1:

self.\_\_reload\_ticks = pygame.time.get\_ticks()

player\_x = self.\_\_player.get\_center()[0]

player\_y = self.\_\_player.get\_center()[1]

mouse\_pos = pygame.mouse.get\_pos()

direction = (player\_x - mouse\_pos[0], player\_y - mouse\_pos[1])

bullet = Bullet(player\_x, player\_y, 16, direction, speed=10, damage=1)

self.\_\_bullets\_group.add(bullet)

def generate\_slimes(self, count):

slime\_size = 48

self.\_\_slimes\_count = count

self.\_\_last\_slimes\_count = count

pos\_choices = 'top', 'down', 'left', 'right'

pos\_choice = random.choice(pos\_choices)

for i in range(self.\_\_slimes\_count):

rand\_x = 0

rand\_y = 0

if pos\_choice == 'top':

rand\_x = random.randint(-slime\_size, settings.SCREEN\_WIDTH - slime\_size)

rand\_y = -slime\_size

elif pos\_choice == 'down':

rand\_x = random.randint(-slime\_size, settings.SCREEN\_WIDTH - slime\_size)

rand\_y = slime\_size + settings.SCREEN\_HEIGHT

elif pos\_choice == 'left':

rand\_x = -slime\_size

rand\_y = random.randint(-slime\_size, settings.SCREEN\_HEIGHT + slime\_size)

elif pos\_choice == 'right':

rand\_x = slime\_size + settings.SCREEN\_WIDTH

rand\_y = random.randint(-slime\_size, settings.SCREEN\_HEIGHT + slime\_size)

slime = Slime(rand\_x, rand\_y, slime\_size, slime\_size, 2, health=3)

self.\_\_enemies\_group.add(slime)

**game\_over.py**

import pygame

from game import settings

from game.entities.crosshair import Crosshair

from game.scenes.scene import Scene

from game.states.game\_states import GameStates

class GameOverScene(Scene):

def \_\_init\_\_(self, gsm, waves):

super().\_\_init\_\_(gsm)

# Crosshair

self.\_\_crosshair = Crosshair(settings.SCREEN\_WIDTH / 2,

settings.SCREEN\_HEIGHT / 2,

35, 35,

settings.ui\_res,

'crosshair.png')

# Fonts

font\_path = str(settings.fonts\_res / 'dpcomic.ttf')

self.\_\_small\_font = pygame.font.Font(font\_path, 45)

self.\_\_large\_font = pygame.font.Font(font\_path, 100)

# Title

self.\_\_title = self.\_\_large\_font.render("Game Over", True, (180, 200, 210))

# Start text

self.\_\_button\_text = self.\_\_small\_font.render("Restart", True, (255, 255, 255))

# Waves text

waves\_text = f"Yor record: {waves} waves."

self.\_\_waves\_text = self.\_\_small\_font.render(waves\_text, True, (235, 235, 235))

# Play Button

button\_press\_path = str(settings.ui\_res / "menu\_button\_press.png")

button\_rel\_path = str(settings.ui\_res / "menu\_button.png")

self.\_\_button\_props = (192, 64)

self.\_\_button\_press\_img = pygame.transform.scale(pygame.image.load(button\_press\_path), self.\_\_button\_props)

self.\_\_button\_rel\_img = pygame.transform.scale(pygame.image.load(button\_rel\_path), self.\_\_button\_props)

self.\_\_button\_image = self.\_\_button\_rel\_img

self.\_\_button\_pressed = False

# Positions

self.\_\_title\_pos = (settings.SCREEN\_WIDTH / 2 - self.\_\_title.get\_width() / 2,

settings.SCREEN\_HEIGHT / 2 - self.\_\_title.get\_height() / 2 - 80)

self.\_\_button\_pos = (settings.SCREEN\_WIDTH / 2 - self.\_\_button\_props[0] / 2,

settings.SCREEN\_HEIGHT / 2 - self.\_\_button\_props[1] / 2)

self.\_\_button\_text\_pos = (self.\_\_button\_pos[0] + self.\_\_button\_text.get\_width() / 2 - 25,

self.\_\_button\_pos[1] + self.\_\_button\_text.get\_height() / 2 - 15)

self.\_\_waves\_text\_pos = (settings.SCREEN\_WIDTH / 2 - self.\_\_waves\_text.get\_width() / 2,

settings.SCREEN\_HEIGHT - 50)

# Rects

self.\_\_button\_rect = self.\_\_button\_image.get\_rect(topleft=self.\_\_button\_pos)

self.\_\_button\_text\_rect = self.\_\_button\_text.get\_rect(center=self.\_\_button\_text\_pos)

# Effects

self.\_\_ignore\_mouse\_clicks = False

self.\_\_transition = False

self.\_\_close\_effect = pygame.Surface((settings.SCREEN\_WIDTH, settings.SCREEN\_HEIGHT))

self.\_\_close\_effect\_alpha = 0

self.\_\_close\_effect.set\_alpha(self.\_\_close\_effect\_alpha)

self.\_\_close\_effect.fill((10, 10, 10))

def draw(self, screen):

screen.blit(self.\_\_title, self.\_\_title\_pos)

screen.blit(self.\_\_button\_image, self.\_\_button\_pos)

screen.blit(self.\_\_button\_text, self.\_\_button\_text\_pos)

screen.blit(self.\_\_waves\_text, self.\_\_waves\_text\_pos)

self.\_\_crosshair.draw(screen)

if self.\_\_transition:

screen.blit(self.\_\_close\_effect, (0, 0))

def update(self, dt):

self.\_\_crosshair.update(dt)

if self.\_\_transition:

self.\_\_close\_effect\_alpha += 10

self.\_\_close\_effect.set\_alpha(self.\_\_close\_effect\_alpha)

if self.\_\_close\_effect\_alpha > 255 + 300:

self.\_gsm.set\_state(GameStates.PLAY)

def handle\_events(self, events):

keys = pygame.key.get\_pressed()

if keys[pygame.K\_SPACE] == 1:

if not self.\_\_button\_pressed:

self.\_\_ignore\_mouse\_clicks = True

self.\_\_button\_image = self.\_\_button\_press\_img

self.\_\_button\_pressed = True

self.\_\_button\_text\_pos = (self.\_\_button\_pos[0] + self.\_\_button\_text.get\_width() / 2 - 25,

self.\_\_button\_pos[1] + self.\_\_button\_text.get\_height() / 2 - 8)

self.\_\_transition = True

if not self.\_\_ignore\_mouse\_clicks:

press = pygame.mouse.get\_pressed()

mouse\_pos = pygame.mouse.get\_pos()

collude\_button = self.\_\_button\_rect.collidepoint(mouse\_pos) or \

self.\_\_button\_text\_rect.collidepoint(mouse\_pos)

if press[0]:

if collude\_button and not self.\_\_button\_pressed:

self.\_\_button\_image = self.\_\_button\_press\_img

self.\_\_button\_pressed = True

self.\_\_button\_text\_pos = (self.\_\_button\_pos[0] + self.\_\_button\_text.get\_width() / 2 - 25,

self.\_\_button\_pos[1] + self.\_\_button\_text.get\_height() / 2 - 8)

else:

if self.\_\_button\_pressed:

self.\_\_button\_image = self.\_\_button\_rel\_img

self.\_\_button\_pressed = False

self.\_\_button\_text\_pos = (self.\_\_button\_pos[0] + self.\_\_button\_text.get\_width() / 2 - 25,

self.\_\_button\_pos[1] + self.\_\_button\_text.get\_height() / 2 - 15)

if collude\_button:

self.\_\_transition = True

**entity.py**

from abc import ABC, abstractmethod

from math import sqrt

import pygame

class Entity(ABC):

def \_\_init\_\_(self, x, y, width, height):

self.\_x = x

self.\_y = y

self.\_vx = 0

self.\_vy = 0

self.\_width = width

self.\_height = height

@abstractmethod

def draw(self, screen):

raise NotImplementedError

@abstractmethod

def update(self, dt):

raise NotImplementedError

def get\_pos(self):

return self.\_x, self.\_y

def set\_pos(self, x, y):

self.\_x = x

self.\_y = y

def get\_width(self):

return self.\_width

def set\_width(self, width):

self.\_width = width

def get\_height(self):

return self.\_height

def set\_height(self, height):

self.\_height = height

def get\_center(self):

return self.\_x + self.\_width / 2, self.\_y + self.\_height / 2

@staticmethod

def normalize\_vector2(vec2):

dir\_len = sqrt(pow(vec2[0], 2) + pow(vec2[1], 2))

dir\_x = vec2[0] / dir\_len

dir\_y = vec2[1] / dir\_len

return dir\_x, dir\_y

@staticmethod

def load(path):

return pygame.image.load(path)

@staticmethod

def load\_i\_hflip(path):

return Entity.h\_flip(pygame.image.load(path))

@staticmethod

def h\_flip(img):

return pygame.transform.flip(img, True, False)

@staticmethod

def v\_flip\_img(img):

return pygame.transform.flip(img, False, True)

@staticmethod

def scale(img, width, height):

return pygame.transform.scale(img, (width, height))

**crosshair.py**

import pygame

from .entity import Entity

class Crosshair(Entity):

def \_\_init\_\_(self, x, y, width, height, crosshair\_dir, crosshair\_filename):

super().\_\_init\_\_(x, y, width, height)

self.\_\_image = self.scale(self.load(str(crosshair\_dir / crosshair\_filename)), width, height)

def update(self, dt):

self.\_x = pygame.mouse.get\_pos()[0] - self.\_width / 2

self.\_y = pygame.mouse.get\_pos()[1] - self.\_height / 2

def draw(self, screen):

screen.blit(self.\_\_image, (self.\_x, self.\_y))

**player.py**

from math import sqrt

import pygame

from pygame.sprite import Sprite

from .entity import Entity

from .. import settings

class Player(Entity, Sprite):

def \_\_init\_\_(self, x, y, width, height, speed=5, health=10):

super().\_\_init\_\_(x, y, width, height)

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

self.\_\_player\_res = settings.player\_res

self.\_\_health = health

self.\_\_hit\_countdown = 0

self.\_\_last\_damage\_ticks = 0

self.\_\_speed = speed

self.\_\_max\_vel = speed \* 2

self.\_\_dx = 0

self.\_\_dy = 0

self.watch\_left = False

self.watch\_right = True

self.move\_top = False

self.move\_down = False

self.move\_left = False

self.move\_right = False

self.idle = True

self.\_\_idle\_anim = [

self.scale(self.load(str(self.\_\_player\_res / 'knight\_idle\_anim\_f0.png')), width, height),

self.scale(self.load(str(self.\_\_player\_res / 'knight\_idle\_anim\_f1.png')), width, height),

self.scale(self.load(str(self.\_\_player\_res / 'knight\_idle\_anim\_f2.png')), width, height),

self.scale(self.load(str(self.\_\_player\_res / 'knight\_idle\_anim\_f3.png')), width, height),

self.scale(self.load(str(self.\_\_player\_res / 'knight\_idle\_anim\_f4.png')), width, height),

self.scale(self.load(str(self.\_\_player\_res / 'knight\_idle\_anim\_f5.png')), width, height)

]

self.\_\_move\_anim = [

self.scale(self.load(str(self.\_\_player\_res / 'knight\_run\_anim\_f0.png')), width, height),

self.scale(self.load(str(self.\_\_player\_res / 'knight\_run\_anim\_f1.png')), width, height),

self.scale(self.load(str(self.\_\_player\_res / 'knight\_run\_anim\_f2.png')), width, height),

self.scale(self.load(str(self.\_\_player\_res / 'knight\_run\_anim\_f3.png')), width, height),

self.scale(self.load(str(self.\_\_player\_res / 'knight\_run\_anim\_f4.png')), width, height),

self.scale(self.load(str(self.\_\_player\_res / 'knight\_run\_anim\_f5.png')), width, height)

]

self.\_\_damage\_image = self.scale(

self.load(str(self.\_\_player\_res / 'knight\_got\_damage.png')), width, height)

self.\_\_anim = self.\_\_idle\_anim

self.\_\_frame = 0

self.image = self.\_\_anim[int(self.\_\_frame)]

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

self.\_\_original\_image = self.image

def draw(self, screen):

screen.blit(self.image, (self.\_x, self.\_y))

def update(self, dt):

self.\_\_frame += 0.2

if self.\_\_frame >= len(self.\_\_anim):

self.\_\_frame = 0

if self.watch\_left:

self.image = self.h\_flip(self.\_\_anim[int(self.\_\_frame)])

else:

self.image = self.\_\_anim[int(self.\_\_frame)]

self.\_\_original\_image = self.image

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

if self.\_\_hit\_countdown:

if self.\_\_hit\_countdown:

self.image = self.\_\_damage\_image

else:

self.image = self.\_\_original\_image

self.\_\_hit\_countdown = max(0, self.\_\_hit\_countdown - 1)

else:

self.\_vx = self.\_\_dx \* self.\_\_speed

self.\_vy = self.\_\_dy \* self.\_\_speed

self.\_x += self.\_vx

self.\_y += self.\_vy

if not self.move\_top and not self.move\_down and not self.move\_left and not self.move\_right:

self.\_\_anim = self.\_\_idle\_anim

else:

self.\_\_anim = self.\_\_move\_anim

def take\_damage(self, enemy\_pos, damage):

seconds\_passed = (pygame.time.get\_ticks() - self.\_\_last\_damage\_ticks) / 1000

if seconds\_passed > 0.1:

self.\_\_health -= damage

self.\_\_last\_damage\_ticks = pygame.time.get\_ticks()

self.\_\_hit\_countdown = 5

pos = self.get\_pos()

dir\_x = pos[0] - enemy\_pos[0]

dir\_y = pos[1] - enemy\_pos[1]

dir\_x, dir\_y = self.normalize\_vector2((dir\_x, dir\_y))

if self.\_vx < self.\_\_max\_vel and self.\_vy < self.\_\_max\_vel:

self.\_vx += dir\_x \* 2

self.\_vy += dir\_y \* 2

def get\_rect(self):

return self.rect

def collidepoint(self, point):

if self.rect.collidepoint(point):

return True

return False

def colliderect(self, rect):

if self.rect.colliderect(rect):

return True

return False

def handle\_events(self, events, crosshair):

keys = pygame.key.get\_pressed()

mouse\_x = crosshair.get\_center()[0]

player\_x = self.get\_center()[0]

if mouse\_x > player\_x:

self.watch\_right = True

self.watch\_left = False

if mouse\_x < player\_x:

self.watch\_left = True

self.watch\_right = False

if keys[pygame.K\_w]:

self.\_\_dy = -1

self.move\_top = True

self.move\_down = False

self.idle = False

elif keys[pygame.K\_s]:

self.\_\_dy = 1

self.move\_down = True

self.move\_top = False

self.idle = False

else:

self.\_\_dy = 0

self.idle = True

self.move\_down = False

self.move\_top = False

if keys[pygame.K\_a]:

self.\_\_dx = -1

self.move\_left = True

self.move\_right = False

self.idle = False

elif keys[pygame.K\_d]:

self.\_\_dx = 1

self.move\_right = True

self.move\_left = False

self.idle = False

else:

self.\_\_dx = 0

self.idle = True

self.move\_right = False

self.move\_left = False

def get\_health(self):

return self.\_\_health

def set\_health(self, health):

self.\_\_health = health

def sub\_health(self, health):

self.\_\_health -= health

def add\_health(self, health):

self.\_\_health += health

**slime.py**

import random

from math import sqrt

import pygame

from pygame.sprite import Sprite

from game import settings

from game.entities.entity import Entity

class Slime(Entity, Sprite):

def \_\_init\_\_(self, x, y, width, height, speed=3, health=5):

super().\_\_init\_\_(x, y, width, height)

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

self.\_\_slime\_res = settings.slime\_res

self.\_\_speed = speed

self.\_\_max\_vel = speed \* 5

self.\_\_health = health

self.\_\_hit\_countdown = 0

self.\_\_last\_damage\_ticks = 0

self.\_\_move\_anim = [

self.scale(self.load(str(self.\_\_slime\_res / 'slime\_run\_anim\_f0.png')), width, height),

self.scale(self.load(str(self.\_\_slime\_res / 'slime\_run\_anim\_f1.png')), width, height),

self.scale(self.load(str(self.\_\_slime\_res / 'slime\_run\_anim\_f2.png')), width, height),

self.scale(self.load(str(self.\_\_slime\_res / 'slime\_run\_anim\_f3.png')), width, height),

self.scale(self.load(str(self.\_\_slime\_res / 'slime\_run\_anim\_f4.png')), width, height),

self.scale(self.load(str(self.\_\_slime\_res / 'slime\_run\_anim\_f5.png')), width, height)

]

self.\_\_anim = self.\_\_move\_anim

self.\_\_frame = 0

self.image = self.\_\_anim[int(self.\_\_frame)]

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

def draw(self, screen):

screen.blit(self.image, (self.\_x, self.\_y))

pygame.draw.rect(screen, (0, 200, 0), self.rect)

def update(self, dt):

self.\_x += self.\_vx

self.\_y += self.\_vy

self.\_\_frame += 0.2

if self.\_\_frame >= len(self.\_\_anim):

self.\_\_frame = 0

self.image = self.\_\_anim[int(self.\_\_frame)]

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

if self.\_\_health <= 0:

self.kill()

def move\_to\_player(self, player\_pos):

self\_pos = self.get\_pos()

dir\_x = self\_pos[0] - player\_pos[0]

dir\_y = self\_pos[1] - player\_pos[1]

dir\_len = sqrt(pow(dir\_x, 2) + pow(dir\_y, 2))

dir\_x = dir\_x / dir\_len

dir\_y = dir\_y / dir\_len

self.\_vx = -dir\_x \* self.\_\_speed

self.\_vy = -dir\_y \* self.\_\_speed

def throw(self, from\_pos, force):

self\_pos = self.get\_pos()

dir\_x = self\_pos[0] - from\_pos[0]

dir\_y = self\_pos[1] - from\_pos[1]

dir\_len = sqrt(pow(dir\_x, 2) + pow(dir\_y, 2))

if dir\_x or dir\_y:

dir\_x = dir\_x / dir\_len

dir\_y = dir\_y / dir\_len

self.\_vx += dir\_x \* force

self.\_vy += dir\_y \* force

def take\_damage(self, from\_pos, damage):

seconds\_passed = (pygame.time.get\_ticks() - self.\_\_last\_damage\_ticks) / 1000

if seconds\_passed > 0.1:

self.\_\_health -= damage

self.\_\_last\_damage\_ticks = pygame.time.get\_ticks()

self.\_\_hit\_countdown = 5

pos = self.get\_pos()

dir\_x = pos[0] - from\_pos[0]

dir\_y = pos[1] - from\_pos[1]

dir\_x, dir\_y = self.normalize\_vector2((dir\_x, dir\_y))

# if self.\_vx < self.\_\_max\_vel and self.\_vy < self.\_\_max\_vel:

# self.\_vx = dir\_x \* self.\_\_speed

# self.\_vy = dir\_y \* self.\_\_speed

self.throw(from\_pos, 40)

def get\_health(self):

return self.\_\_health

def set\_health(self, health):

self.\_\_health = health

def sub\_health(self, health):

self.\_\_health -= health

def add\_health(self, health):

self.\_\_health += health

def collidepoint(self, point):

if self.rect.collidepoint(point):

return True

return False

def colliderect(self, rect):

if self.rect.colliderect(rect):

return True

return False

def get\_rect(self):

return self.rect

**bullet.py**

import pygame

from pygame.sprite import Sprite

from game import settings

from game.entities.entity import Entity

class Bullet(Entity, Sprite):

def \_\_init\_\_(self, x, y, radius, direction, speed=10, damage=1):

super().\_\_init\_\_(x, y, width=radius, height=radius)

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

self.\_\_radius = radius

self.\_\_direction = self.normalize\_vector2(direction)

self.\_\_speed = speed

self.\_\_damage = damage

self.\_\_vel\_x = speed \* -self.\_\_direction[0]

self.\_\_vel\_y = speed \* -self.\_\_direction[1]

self.image = self.load(str(settings.bullets\_res / 'bullet.png'))

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

def draw(self, screen):

screen.blit(self.image, (self.\_x, self.\_y))

# pygame.draw.circle(screen, self.\_\_color, (self.\_x, self.\_y), self.\_\_radius)

def update(self, dt):

if self.\_x < settings.SCREEN\_WIDTH and self.\_x > 0 and self.\_y < settings.SCREEN\_HEIGHT and self.\_y > 0:

self.\_x += self.\_\_vel\_x

self.\_y += self.\_\_vel\_y

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

else:

self.kill()

def get\_damage(self):

return self.\_\_damage

def set\_damage(self, damage):

self.\_\_damage = damage

def get\_rect(self):

return self.rect

def collidepoint(self, point):

if self.rect.collidepoint(point):

return True

return False

def colliderect(self, rect):

if self.rect.colliderect(rect):

return True

return False