НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ім. ІГОРЯ СІКОРСЬКОГО»
ФАКУЛЬТЕТ ІНФОРМАТИКИ ТА ОБЧИСЛЮВАНОЇ ТЕХНІКИ
КАФЕДРА ОБЧИСЛЮВАНОЇ ТЕХНІКИ

**КУРСОВА РОБОТА**

з дисципліни «Інженерія програмного забезпечення»
на тему: «Прототип гри “Гугл Динозавр”»

Студента 2 курсу групи ІО-12
Напряму підготовки:
123 - Комп’ютерна інженерія
Кравченка Вадима Андрійовича

Керівник:
асистент Русінов В.В.

Національна оцінка _____
Кількість балів _____

Члени комісії: _____ _____
(підпис) (вчене звання, науковий ступінь, прізвище та ініціали)

_____ _____
(підпис) (вчене звання, науковий ступінь, прізвище та ініціали)

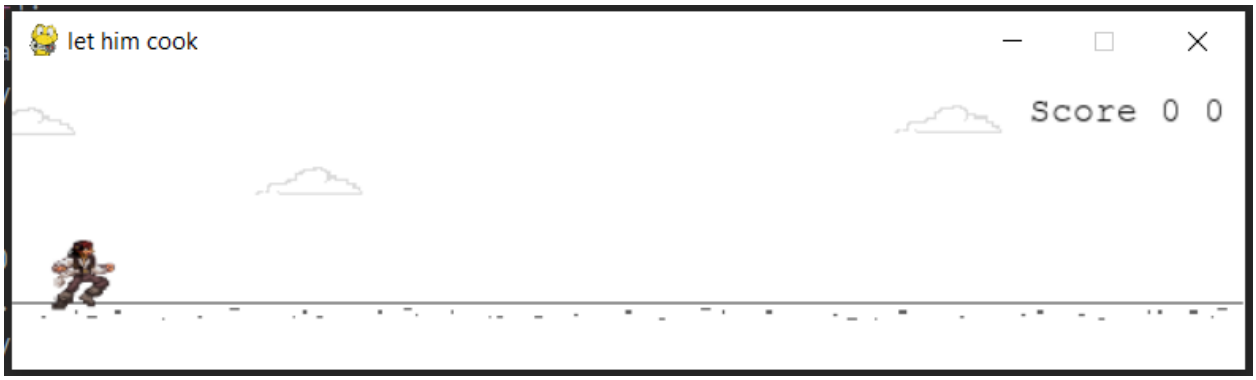
_____ _____
(підпис) (вчене звання, науковий ступінь, прізвище та ініціали)

Київ - 2023 рік

# Зміст

# Вступ

У курсовій роботі описана логіка роботи, тестування та демонстрація прототипа гри **“Гугл динозавр”**. Описана у курсовій роботі програма є досить цікавою, кумедною і простою для користування. Гра привносить новий подих у класичну і всіми нами улюблену гру оскільки головними героями гри виступають **Капітан Джек Горобець**(персонаж) і **Моаї**(перешкода)




*Скріншот того, що бачить користувач після запуску програми.*


*Скріншот того, що бачить користувач після смерті в грі.*

# 1. Технічне завдання

## 1.1. Загальне завдання

Завданням курсової роботи є реалізація прототипу гри “Гугл динозавр”(з’являється коли користувач відкриває Google Chrome без підключення до мережі). Програма написана на мові Python. Розробка програми відбувалася в середовищі PyCharm Community Edition. Для вирішення графічних питань реалізації гри була використана бібліотека PyGame.

## 1.2. Можливості програми

Програма має такі функції:

- Можливість перезапуску гри після смерті без закриття програми;
- Трекінг результату поточного сеансу гри;
- Запис найкращого результату у сесії гри(до закриття програми);
- Користувач має змогу чути звук кожного стрибка у грі;
- Користувач має змогу чути звук кожної смерті у грі.

# 2. Логіка гри

## 2.1. Загальне завдання та вимоги

Програма має містити декілька класів та модулів, які в результаті зроблять гру можливою. Класи мають забезпечити виконання таких дій персонажем гри: пересування, стрибок, падіння, смерть. Також класи реалізують спавн перешкод під час гри, підрахунок балів, яких отримав гравець за час сеансу гри, запис кращого результату сесії і можливість перезапуску гри після смерті без втрати кращого результату та без втрати часу на перезапуск програми.

## 2.2. Функції гри

**У грі реалізовані такі функції:**

- “пересування” об’єкта:
- Стрибок персонажа
- Рандомний спавн перешкод
- Смерть від зіткнення персонажу і перешкоди та пропозиція перезапустити гру на клавішу ESCAPE
- Перезапуск гри після смерті зі збереженням максимального рахунку

## 2.3. Детальне пояснення роботи

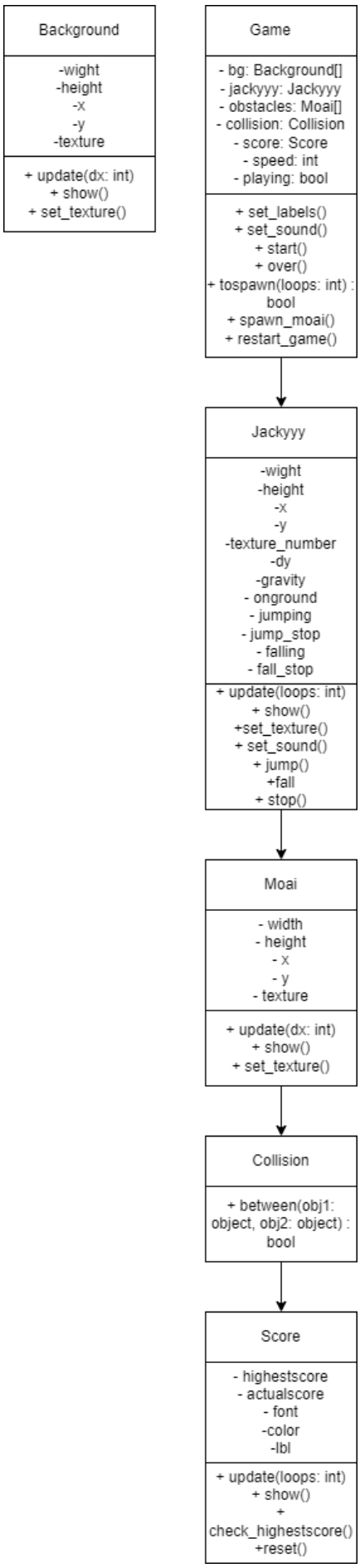



*Рисунок 2.3.1 - Класова діаграма логіки гри*

Клас **Background**: клас реалізує появу фону для гри.

- **def init(self, x)** ініціалізує фон із заданою координатою x.
- **def update(self, dx)** оновлює положення фону шляхом додавання dx до поточної координати x, забезпечує “безкінечність” фону.
- **def show(self)** рендерить фон на екрані гри.
- **def set_texture(self)** завантажує та масштабує фонове зображення.

Клас **Game**: відображає стан і логіку гри

- **init(self, highestscore=0)** ініціалізує ігровий об’єкт із параметром найвищого результату. Він створює екземпляри інших пов’язаних з грою об’єктів, таких як Background, Jackyyy, Collision і Score. Він також встановлює швидкість гри, стан відтворення та ініціалізує звук і мітки.
- **set_labels(self)** встановлює текстові мітки, які використовуються для відображення повідомлень у грі. Він створює два об’єкти pygame.Surface і відображає текст на них.
- **set_sound(self)** завантажує звуковий ефект, який використовується в грі під час смерті персонажа, із шляху до файлу.
- **start(self)** встановлює стан гри на відтворення, дозволяючи головному ігровому циклу виконувати логіку гри.
- **over(self)** обробляє стан завершення гри. Він відтворює звуковий ефект, відображає повідомлення про завершення гри на екрані, використовуючи попередньо встановлені мітки, і встановлює для стану відтворення значення False.
- **tospawn(self, loops)** перевіряє, чи настав час створити нову перешкоду на основі кількості циклів.

- **spawn_moai(self)** створює нову перешкоду (moai) у грі. Він генерує випадкову x-координату для нової перешкоди та створює екземпляр класу Moai з цією x-координатою. Нова перешкода додається до списку перешкод.
- **restart_game(self)** перезапускає гру, повторно ініціалізуючи всі ігрові об’єкти та скидаючи рахунок. Для встановлення найвищого балу потрібен параметр highestscore. Він викликає метод init() із наданим найвищим балом або 0, якщо його немає.

Клас **Jackyyy**: представляє головного героя гри.

- **init(self)** ініціалізує об’єкт головного символу за допомогою таких атрибутів, як ширина, висота, початкова позиція, стан стрибка, стан падіння та інші пов’язані змінні. Він також завантажує текстури персонажа та звуковий ефект.
- **update(self, loops)** оновлює позицію головного героя та анімацію на основі кількості циклів. Якщо персонаж стрибає, він зменшує y-координату, доки не досягне значення jump_stop. Якщо персонаж падає, він збільшує y-координату на основі значення сили тяжіння та dy, доки не досягне значення fall_stop. Якщо персонаж лежить на землі, анімація персонажа оновлюється залежно від кількості петель.
- **show(self)** рендерить текстуру персонажа на екрані в поточній позиції.
- **set_texture(self)** завантажує текстуру персонажа на основі атрибута texture_number і масштабує її до вказаної ширини та висоти.
- **set_sound(self)** завантажує звуковий ефект, який використовується, коли персонаж стрибає.
- **jump(self)** відтворює звуковий ефект стрибка та встановлює для стану стрибка значення True.
- **fall(self)** встановлює стан стрибка на False і стан падіння на True.
- **stop(self)** встановлює стан падіння на False і стан onground на True.

Клас **Moai**: символізує перешкоди в грі.

- **init(self, x)** ініціалізує об’єкт-перешкоду із заданою координатою x. Він встановлює ширину, висоту, атрибути x та y, завантажує та масштабує текстуру перешкоди та викликає метод show(), щоб відобразити її на екрані.
- **update(self, dx)** оновлює положення перешкоди, додаючи вказане значення dx до поточної координати x.
- **show(self)** рендерить текстуру перешкоди на екрані у вказаній позиції.
- **set_texture(self)** завантажує текстуру перешкоди з шляху до файлу та масштабує її до вказаної ширини та висоти.

Клас **Collision**: контролює виявлення зіткнень між об’єктами гри.

- **between(self, obj1, obj2)** обчислює відстань між двома об’єктами за допомогою формули відстані та перевіряє, чи вони стикаються. Він повертає True, якщо відстань менше 35 (попередньо визначене порогове значення вибрано так, щоб два об’єкти не спавнились дуже близько, унеможливлюючи перемогу для клієнта).

Клас **Score:** відображає рахунок гри.

- **init(self, maximumscore)** ініціалізує об’єкт оцінки з наданим найвищим балом за всі ігри, які зіграв користувач не закриваючи гру. Він встановлює найвищий бал, фактичний бал, атрибути шрифту та кольору. Він також викликає метод show() для відтворення партитури на екрані.
- **update(self, loops)** оновлює фактичну оцінку на основі кількості циклів, поділеної на 13(за допомогою цього числа можна міняти швидкість з якою ми отримуємо бали.Чим більше число – тим повільніше нараховуються бали). Потім він викликає метод check_highestscore() для оновлення найвищої оцінки, якщо необхідно.
- **show(self)** відтворює рахунок на екрані за допомогою атрибутів шрифту та кольору.
- check_highestscore(self) порівнює рахунок, який має користувач на даний момент з найвищим рахунком за всю сесію гри та оновлює найвищий рахунок, якщо поточний рахунок більший або дорівнює йому.
- **reset(self)** скидає поточний рахунок до нуля після смерті користувача.

Функція **main**: точка входу в гру. Вона створює екземпляри класу Game та об’єкта Jackyyy. Вона ініціалізує такі змінні, як clock, loops та over (стан завершення гри). Основний ігровий цикл виконує логіку гри, включаючи оновлення фону, персонажа, перешкод, обробку зіткнень, оновлення рахунку та обробку введених користувачем даних. Цикл також обробляє такі події, як вихід з гри або її перезапуск. Годинник забезпечує постійну частоту кадрів, а pygame.display.update() оновлює екран, щоб відобразити зміни в кожному кадрі.

# 3. Графічний інтерфейс

## 3.1. Загальне завдання та вимоги до графічної частини

Має бути реалізоване графічне супроводження програми, яке буде взаємодіяти з логікою гри  та виводити такі функції гри як пересування персонажа, поява перешкод, ефект безкінечного фону, вивід інформації про рахунок в грі та інформацію про кінець гри, можливість її перезапуску . Також має бути звукове супроводження таких дій гравця як стрибок та смерть.

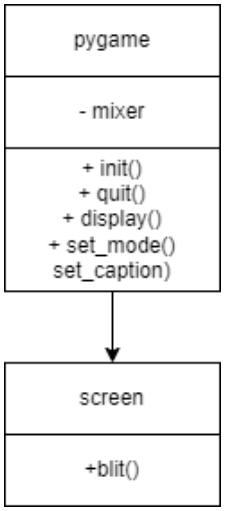



*Рисунок 3.1.1 – класова діаграма графічної частини коду*

### 3.2. Ілюстрації функцій гри:

- **“пересування” об’єкта:**
  На рисунках 3.2.1.1, 3.2.1.2 та 3.2.1.3 ми можемо побачити, що герой приймає різні позиції, що й створює іллюзію руху персонажу. Реалізовано це за допомогою циклічної зміни рисунків, кожен з яких ілюструє певне положення об’єкта.


*Рисунок3.2.1.1*

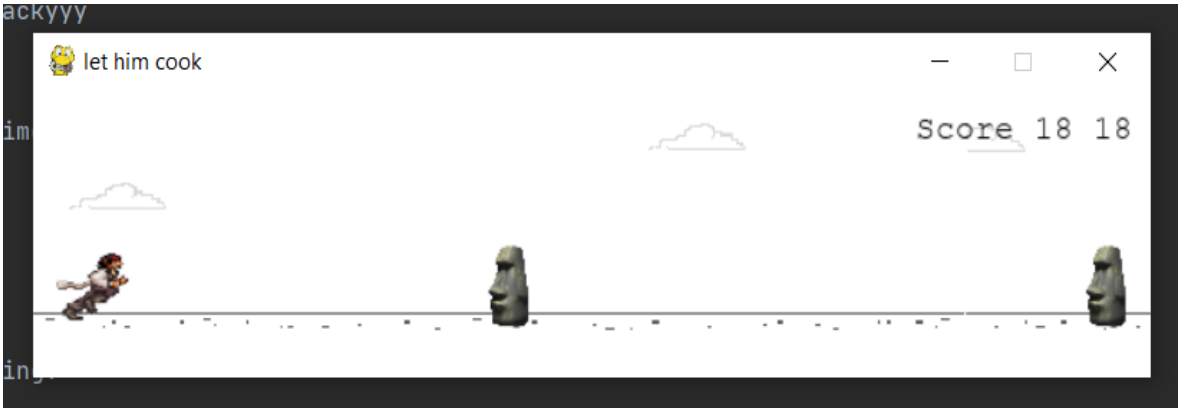



*Рисунок3.2.1.2*


*Рисунок 3.2.1.3*

- **Стрибок персонажа:**

На рисунках 3.2.2.1, 3.2.2.2 та 3.2.2.3 ми можемо пересвідчитися, що при натисканні клавіші SPACE персонаж стрибає, приймаючи останню позицію до виконання цього самого стрибка

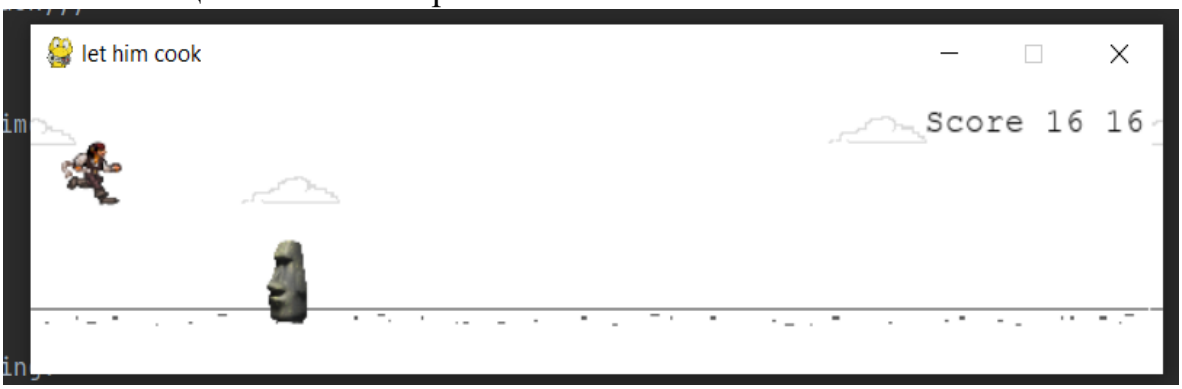



*Рисунок 3.2.2.1*

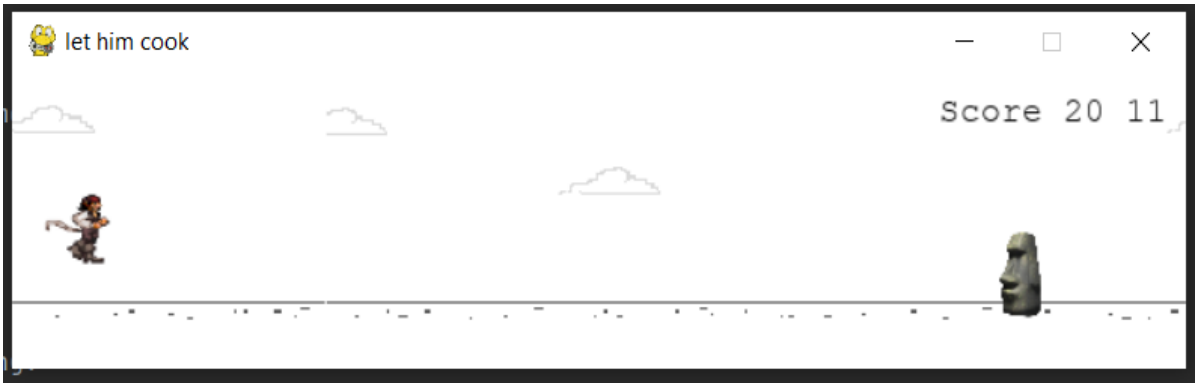



*Рисунок 3.2.2.2*

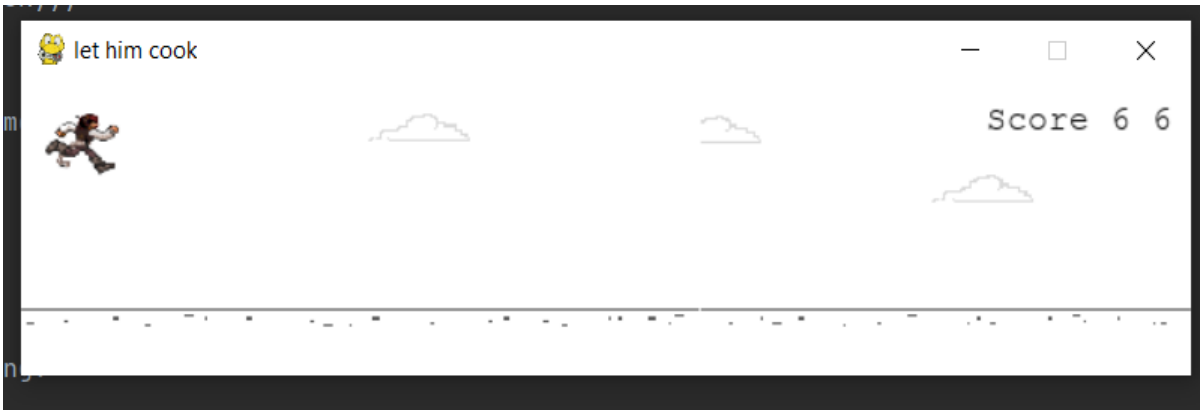



*Рисунок 3.2.2.3*

- **Рандомний спавн перешкод:**
  На рисунках 3.2.3.1, 3.2.3.2 та 3.2.3.3 ми можемо пересвідчитися, що перешкоди дійсно спавнятся і роблять це рандомно(скріншоти взяті з одної сесії гри та з приблизно однаковим рахунком, але з різних спроб, аби пересвідчитися, що спавн працює рандомно)

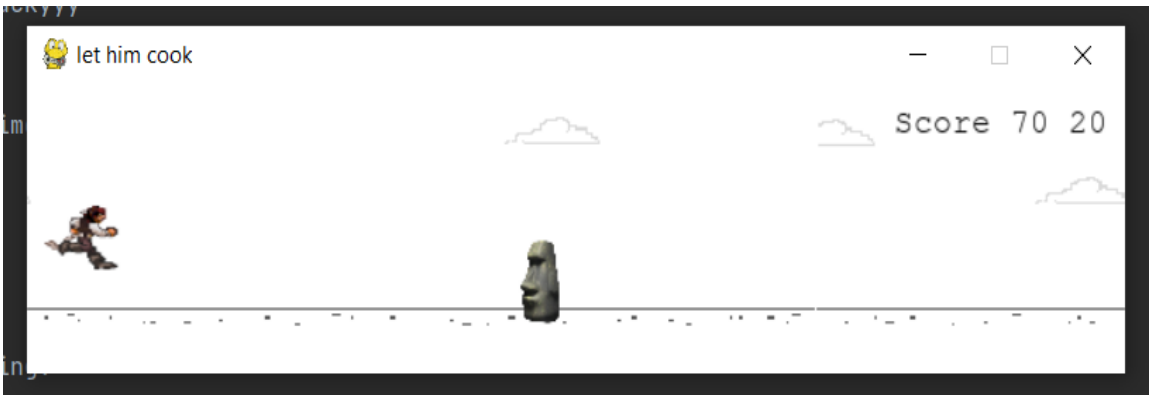



*Рисунок 3.2.3.1*


*Рисунок 3.2.3.2*

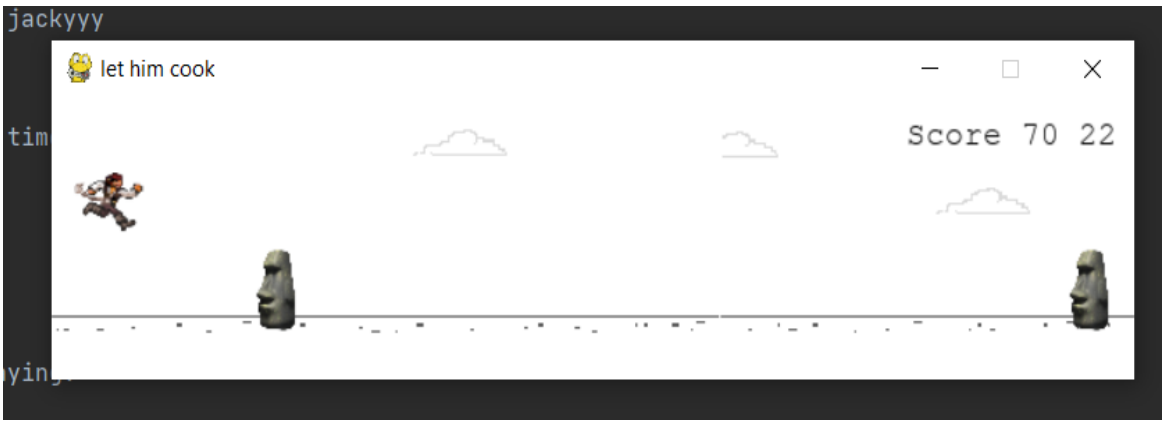



*Рисунок 3.2.3.3*

- **Смерть від зіткнення персонажу і перешкоди та пропозиція перезапустити гру на клавішу ESCAPE:**
  На рисунку 3.2.4.1 можемо побачити, що під час зіткнення Капітана Джека Горобця(персонаж) і Моаї(перешкода) гра зупиняється, виводиться напис JACKYYY DEAD, що сповішає про кінець гри і пропонується перезапустити гру натиснувши на ESCAPE


*Рисунок 3.2.4.1*

- **Перезапуск гри після смерті зі збереженням максимального рахунку:**
  На рисунках 3.2.5.1 ми бачимо як після смерті персонажа програма пропонує перезапустити гру. Після натискання клавіші ESCAPE гра перезапускається зі збереженням максимального рахунку. Рисунок 3.2.5.2 демонструє цю функцію(обидва скріншоти були зроблені в одній сесії, 3.2.5.1 до натискання esc і 3.2.5.2 після)

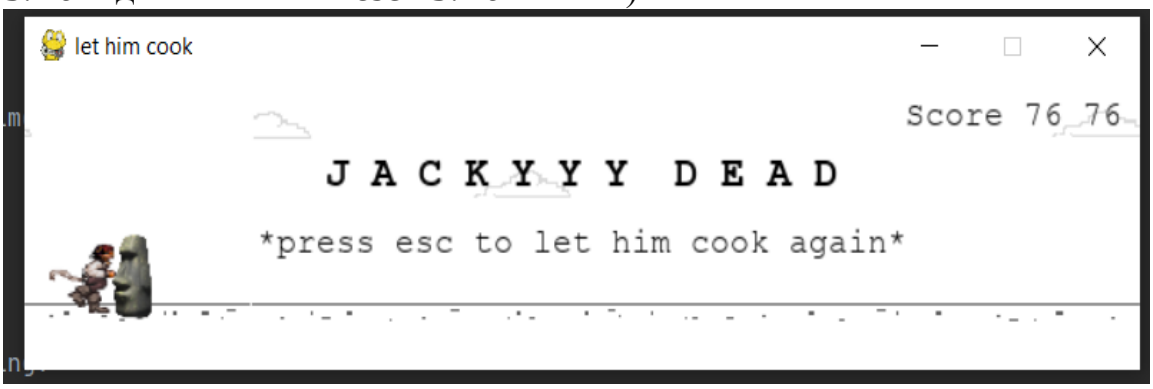



*Рисунок 3.2.5.1*

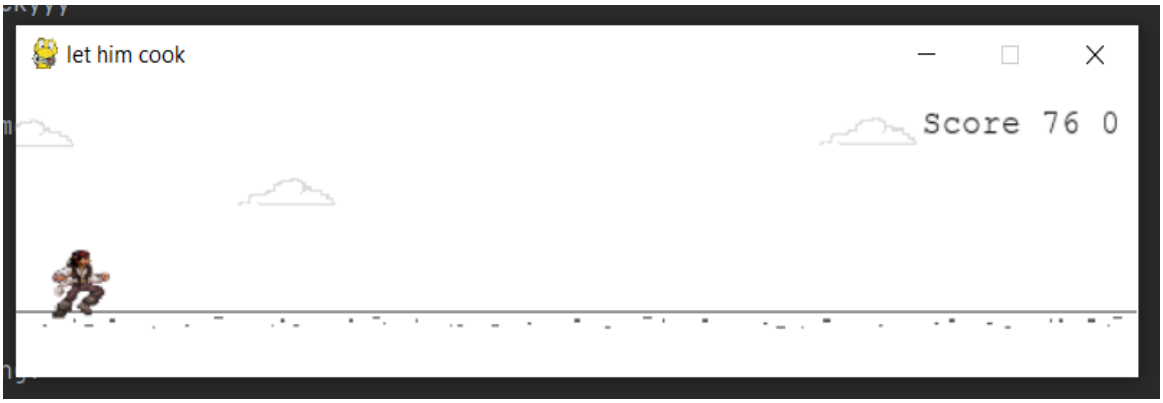



*Рисунок 3.2.5.2*

# 4. Тестування(юніт тести)

## 4.1. Принцип тестування

Для тестування мною був створений файл **test.py**, що зберігає в собі юніт-тести для пеервірки правильності логіки гри. Ці юніт-тести написані з використанням модуля unittest, який надає фреймворк для написання тестів в мові програмування Python. Логіка цих тестів полягає у встановленні певних передумов, виклику методів для тестування, та перевірці очікуваних результатів. Юніт-тести дозволяють автоматично перевірити коректність роботи різних функцій і методів програми, спрощуючи процес виявлення помилок та забезпечуючи більшу впевненість у правильному функціонуванні коду.

**Основні елементи логіки цих тестів:**

- Кожен клас тесту відповідає окремому класу програми, який ми хочемо перевірити. Наприклад, клас TestBackground тестує клас Background.
- Кожен метод в класі тесту відповідає певному методу або властивості, яку потрібно перевірити. Наприклад, метод test_init перевіряє правильність ініціалізації об'єкта Background.
- В методах тесту використовуються функції з модуля unittest, такі як assertEqual, assertNotEqual, assertTrue, assertFalse, для перевірки очікуваних результатів. Наприклад, self.assertEqual(self.bg.width, 623) перевіряє, що ширина об'єкта Background дорівнює 623.
- Для ізоляції певних частин коду та перехоплення викликів до залежних об'єктів, використовуються декоратори patch і об'єкти MagicMock або Mock. Це дозволяє контролювати поведінку залежностей під час тестування. Наприклад, @patch.object(Background, 'show') перехоплює виклик методу show у класі Background.
- В методах тесту використовуються функції з модуля unittest, такі як assertEqual, assertNotEqual, assertTrue, assertFalse, для перевірки очікуваних результатів. Наприклад, self.assertEqual(self.bg.width, 623) перевіряє, що ширина об'єкта Background дорівнює 623.
- Для ізоляції певних частин коду та перехоплення викликів до залежних об'єктів, використовуються декоратори patch і об'єкти MagicMock або Mock. Це дозволяє контролювати поведінку залежностей під час

тестування. Наприклад, @patch.object(Background, 'show') перехоплює виклик методу show у класі Background.

- Методи setUp та tearDown викликаються перед та після кожного тесту відповідно і використовуються для підготовки середовища перед виконанням тестів або для очищення після них.
- На кінці файлу знаходиться умовна конструкція if name == "main", яка запускає всі тести, якщо файл виконується безпосередньо, а не імпортується в інший модуль.

### 4.2. Вихідний код для тестування

***Примітка*- для реалізації юніт тестів у головному коді гри замість main() у останньому рядку я ввів рядки нижче:**

```
#main()
if __name__ == "__main__":
    pygame.init()
    screen = pygame.display.set_mode((WIDTH, HEIGHT))
    game = Game()
    clock = pygame.time.Clock()
    loops = 0
    over = False
    while True:
        # Your game logic here

        clock.tick(77)
        pygame.display.update()
```

*Рисунок 4.2.1 – заміна рядка у головному коді для тестів*

**Код файлу test.py, що виконує тести:**

```
import unittest
from unittest import TestCase
from unittest.mock import patch, MagicMock, Mock
import pygame
import os
from jackyyy import Background, Jackyyy, Moai, Collision, Score

# Test case for the Background class
class TestBackground(unittest.TestCase):
    def setUp(self):
        pygame.init()
        self.screen = pygame.display.set_mode((623, 150))
        self.bg = Background(0)

    def tearDown(self):
        pygame.QUIT

    def test_init(self):
        # Test the initialization of Background object
```

```
        self.assertEqual(self.bg.width, 623)
        self.assertEqual(self.bg.height, 150)
        self.assertEqual(self.bg.x, 0)
        self.assertEqual(self.bg.y, 0)
        self.assertIsNotNone(self.bg.texture)

    def test_update(self):
        # Test the update method of Background object
        self.bg.update(10)
        self.assertEqual(self.bg.x, 10)

    @patch.object(Background, 'show')
    def test_show(self, mock_show):
        # Test the show method of Background object
        self.bg.texture = "mock_texture"  # Set a mock texture for testing
        self.bg.show()
        mock_show.assert_called_once_with()

    def test_set_texture(self):
        # Test the set_texture method of Background object
        with patch("pygame.image.load") as mock_load, \
             patch("pygame.transform.scale") as mock_scale:
            self.bg.set_texture()

mock_load.assert_called_once_with(os.path.join('content/images/bg.png'))
            mock_scale.assert_called_once_with(mock_load.return_value,
(self.bg.width, self.bg.height))

# Test case for the Jackyyy class
class TestJackyyy(unittest.TestCase):
    def setUp(self):
        pygame.init()
        self.screen = pygame.display.set_mode((623, 150))
        self.jackyyy = Jackyyy()

    def tearDown(self):
        pygame.QUIT

    def test_init(self):
        # Test the initialization of Jackyyy object
        self.assertEqual(self.jackyyy.width, 44)
        self.assertEqual(self.jackyyy.height, 44)
        self.assertEqual(self.jackyyy.x, 10)
        self.assertEqual(self.jackyyy.y, 80)
        self.assertEqual(self.jackyyy.texture_number, 0)
        self.assertEqual(self.jackyyy.dy, 3)
        self.assertEqual(self.jackyyy.gravity, 1.23)
        self.assertTrue(self.jackyyy.onground)
        self.assertFalse(self.jackyyy.jumping)
        self.assertEqual(self.jackyyy.jump_stop, 10)
        self.assertFalse(self.jackyyy.falling)
        self.assertEqual(self.jackyyy.fall_stop, self.jackyyy.y)
        self.assertIsNotNone(self.jackyyy.texture)
        self.assertIsNotNone(self.jackyyy.sound)

    def test_update_jumping(self):
        # Test the update method when Jackyyy is jumping
        self.jackyyy.jumping = True
        self.jackyyy.update(0)
        self.assertEqual(self.jackyyy.y, 77)

    def test_update_falling(self):
        # Test the update method when Jackyyy is falling
        self.jackyyy.falling = True
```

```
        self.jackyyy.update(0)
        self.assertEqual(self.jackyyy.y, 83.69)

    def test_update_walking(self):
        # Test the update method when Jackyyy is walking
        self.jackyyy.texture_number = 0
        self.jackyyy.update(0)
        self.assertEqual(self.jackyyy.texture_number, 1)

    @patch.object(Jackyyy, 'show')
    def test_show(self, mock_show):
        # Test the show method of Jackyyy object
        self.jackyyy.texture = "mock_texture"  # Set a mock texture for
testing
        self.jackyyy.show()
        mock_show.assert_called_once_with()

    def test_set_texture(self):
        # Test the set_texture method of Jackyyy object
        with patch("pygame.image.load") as mock_load, \
             patch("pygame.transform.scale") as mock_scale:
            self.jackyyy.set_texture()

mock_load.assert_called_once_with(os.path.join(f'content/images/jack{self.jac
kyyy.texture_number}.png'))
            mock_scale.assert_called_once_with(mock_load.return_value,
(self.jackyyy.width, self.jackyyy.height))

    def test_set_sound(self):
        # Test the set_sound method of Jackyyy object
        with patch("pygame.mixer.Sound") as mock_sound:
            self.jackyyy.set_sound()

mock_sound.assert_called_once_with(os.path.join('content/sounds/jump.wav'))

    @patch("pygame.mixer.Sound")
    def test_jump(self, mock_sound):
        # Test the jump method of Jackyyy object
        jackyyy = Jackyyy()
        mock_play = mock_sound.return_value.play = MagicMock()

        jackyyy.jump()

        mock_play.assert_called_once()

    def test_fall(self):
        # Test the fall method of Jackyyy object
        self.jackyyy.fall()
        self.assertFalse(self.jackyyy.jumping)
        self.assertTrue(self.jackyyy.falling)

    def test_stop(self):
        # Test the stop method of Jackyyy object
        self.jackyyy.stop()
        self.assertFalse(self.jackyyy.falling)
        self.assertTrue(self.jackyyy.onground)

# Test case for the Moai class
class TestMoai(unittest.TestCase):
    def setUp(self):
        pygame.init()
        self.screen = pygame.display.set_mode((623, 150))
        self.moai = Moai(100)
```

```
    def tearDown(self):
        pygame.QUIT

    def test_init(self):
        # Test the initialization of Moai object
        self.assertEqual(self.moai.width, 34)
        self.assertEqual(self.moai.height, 44)
        self.assertEqual(self.moai.x, 100)
        self.assertEqual(self.moai.y, 80)
        self.assertIsNotNone(self.moai.texture)

    def test_update(self):
        # Test the update method of Moai object
        self.moai.update(2)
        self.assertEqual(self.moai.x, 102)

    @patch.object(Moai, 'show')
    def test_show(self, mock_show):
        # Test the show method of Moai object
        self.moai.texture = "mock_texture"  # Set a mock texture for testing
        self.moai.show()
        mock_show.assert_called_once_with()

    def test_set_texture(self):
        # Test the set_texture method of Moai object
        with patch("pygame.image.load") as mock_load, \
             patch("pygame.transform.scale") as mock_scale:
            self.moai.set_texture()

mock_load.assert_called_once_with(os.path.join('content/images/moai.png'))
            mock_scale.assert_called_once_with(mock_load.return_value,
(self.moai.width, self.moai.height))

# Test case for the Collision class
class TestCollision(unittest.TestCase):
    def setUp(self):
        self.collision = Collision()

    def test_between(self):
        # Test the between method of Collision object
        obj1 = Mock(x=10, y=20)
        obj2 = Mock(x=30, y=40)
        self.assertTrue(self.collision.between(obj1, obj2))

        obj1 = Mock(x=10, y=20)
        obj2 = Mock(x=11, y=21)
        self.assertTrue(self.collision.between(obj1, obj2))

        obj1 = Mock(x=10, y=20)
        obj2 = Mock(x=100, y=200)
        self.assertFalse(self.collision.between(obj1, obj2))

# Test case for the Score class
class TestScore(unittest.TestCase):
    def setUp(self):
        pygame.init()
        self.screen = pygame.display.set_mode((623, 150))
        self.score = Score(100)

    def tearDown(self):
        pygame.QUIT

    def test_init(self):
        # Test the initialization of Score object
```

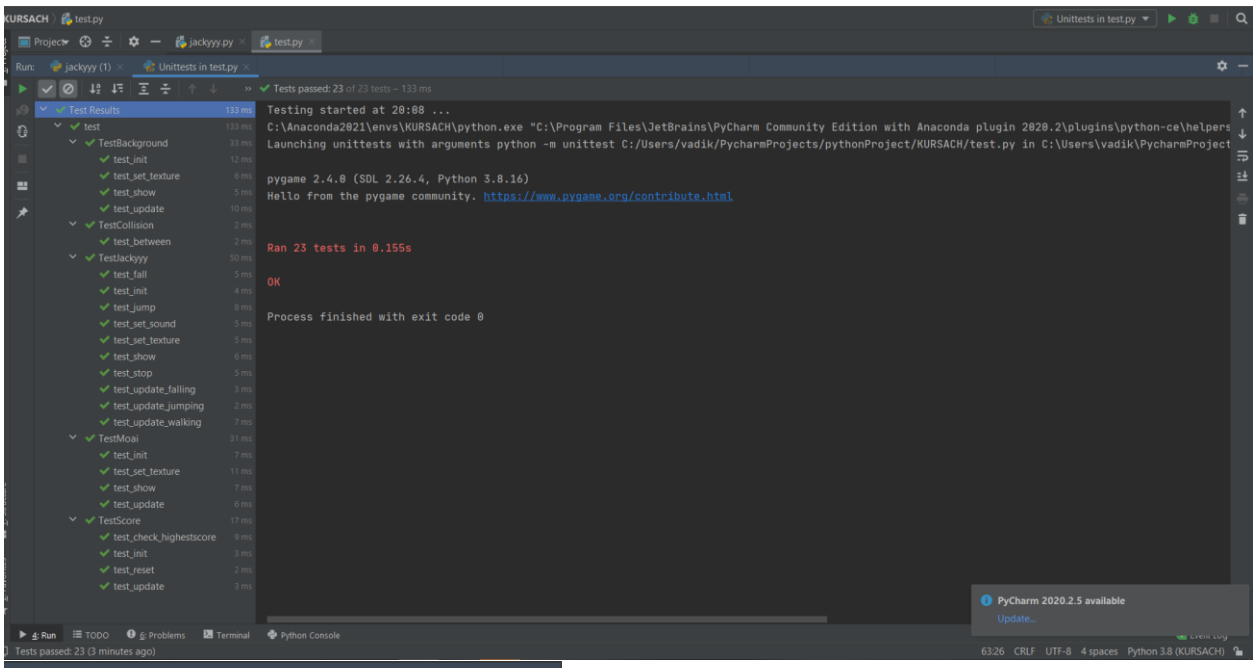
```
        self.assertEqual(self.score.highestscore, 100)
        self.assertEqual(self.score.actualscore, 0)
        self.assertIsNotNone(self.score.font)
        self.assertEqual(self.score.color, (0, 0, 0))

    def test_update(self):
        # Test the update method of Score object
        self.score.update(50)
        self.assertEqual(self.score.actualscore, 3)

    def test_check_highestscore(self):
        # Test the check_highestscore method of Score object
        self.score.actualscore = 50
        self.score.check_highestscore()
        self.assertEqual(self.score.highestscore, 100)

        self.score.actualscore = 150
        self.score.check_highestscore()
        self.assertEqual(self.score.highestscore, 150)

    def test_reset(self):
        # Test the reset method of Score object
        self.score.actualscore = 50
        self.score.reset()
        self.assertEqual(self.score.actualscore, 0)

if __name__ == "__main__":
    unittest.main()
```

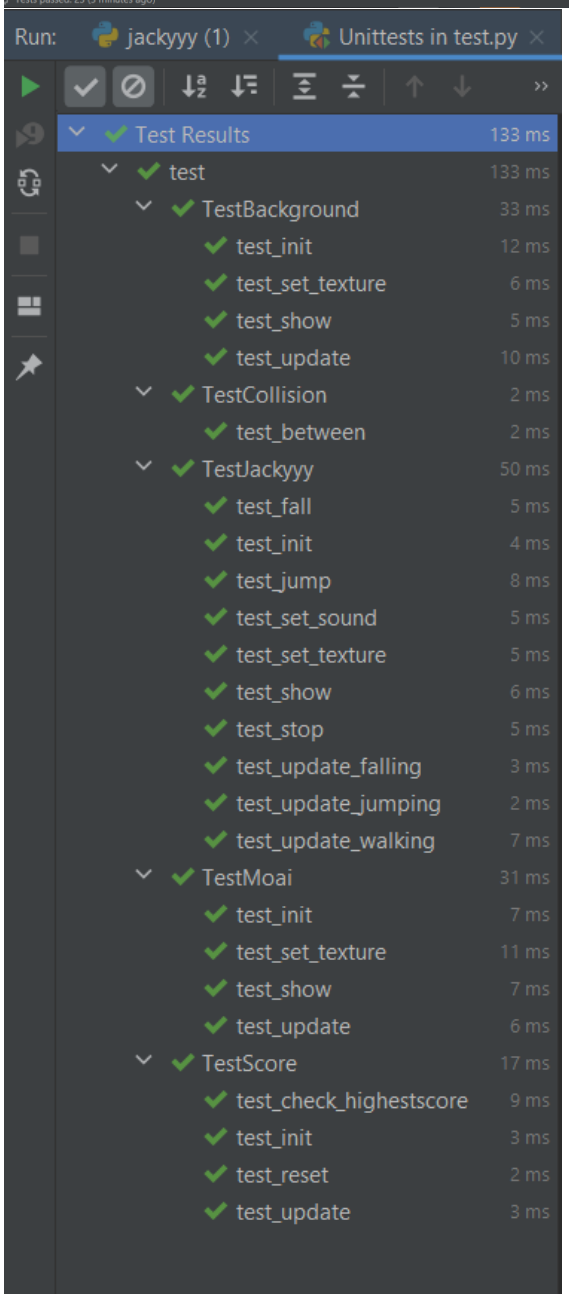
### 4.3. Результати тестування

На скріншотах нижче ми можемо переконатися, що проблем з тестами не виникло і ми маємо коректний код. Для запуску тестів я ввів команду **python -m unittest test.py** у консоль:

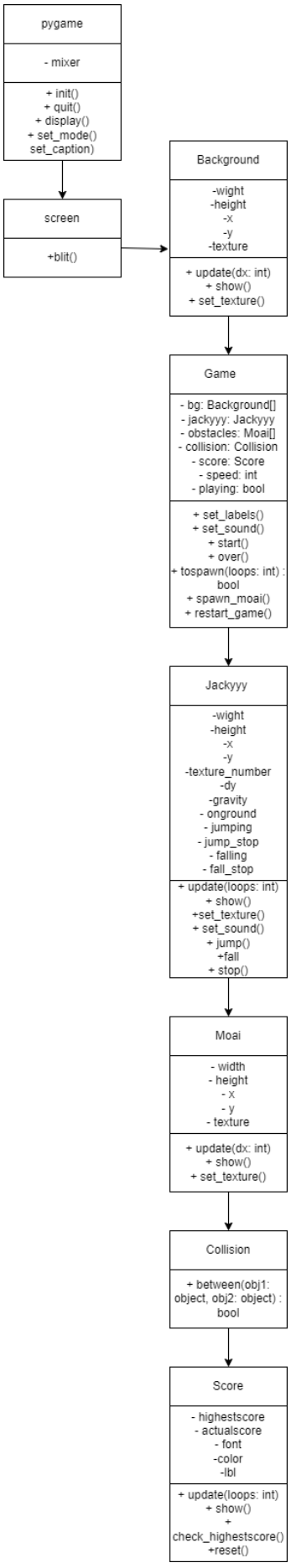
```
(KURSACH) C:\Users\vadik\PycharmProjects\pythonProject\KURSACH>python -m unittest test.py
pygame 2.4.0 (SDL 2.26.4, Python 3.8.16)
Hello from the pygame community. https://www.pygame.org/contribute.html
.......................
----------------------------------------------------------------------
Ran 23 tests in 0.115s

OK

(KURSACH) C:\Users\vadik\PycharmProjects\pythonProject\KURSACH>
```

4: Run | TODO | 6: Problems | Terminal | Python Console

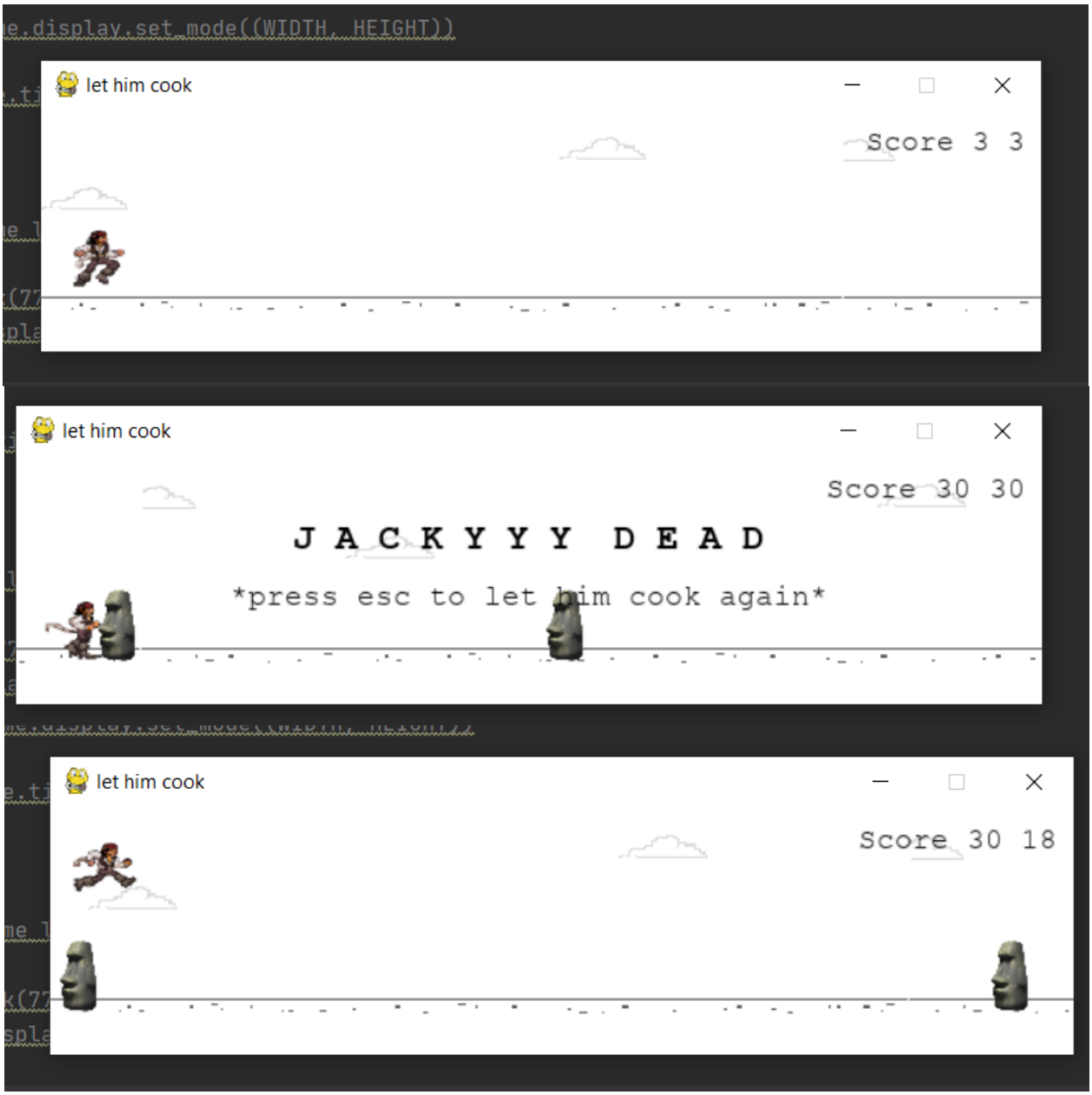

Testing started at 20:08 ...
C:\Anaconda2021\envs\KURSACH\python.exe "C:\Program Files\JetBrains\PyCharm Community Edition with Anaconda plugin 2020.2\plugins\python-ce\helpers
Launching unittests with arguments python -m unittest C:/Users/vadik/PycharmProjects/pythonProject/KURSACH/test.py in C:\Users\vadik\PycharmProject
pygame 2.4.0 (SDL 2.26.4, Python 3.8.16)
Hello from the pygame community. https://www.pygame.org/contribute.html
Ran 23 tests in 0.155s
OK
Process finished with exit code 0
PyCharm 2020.2.5 available
Update...

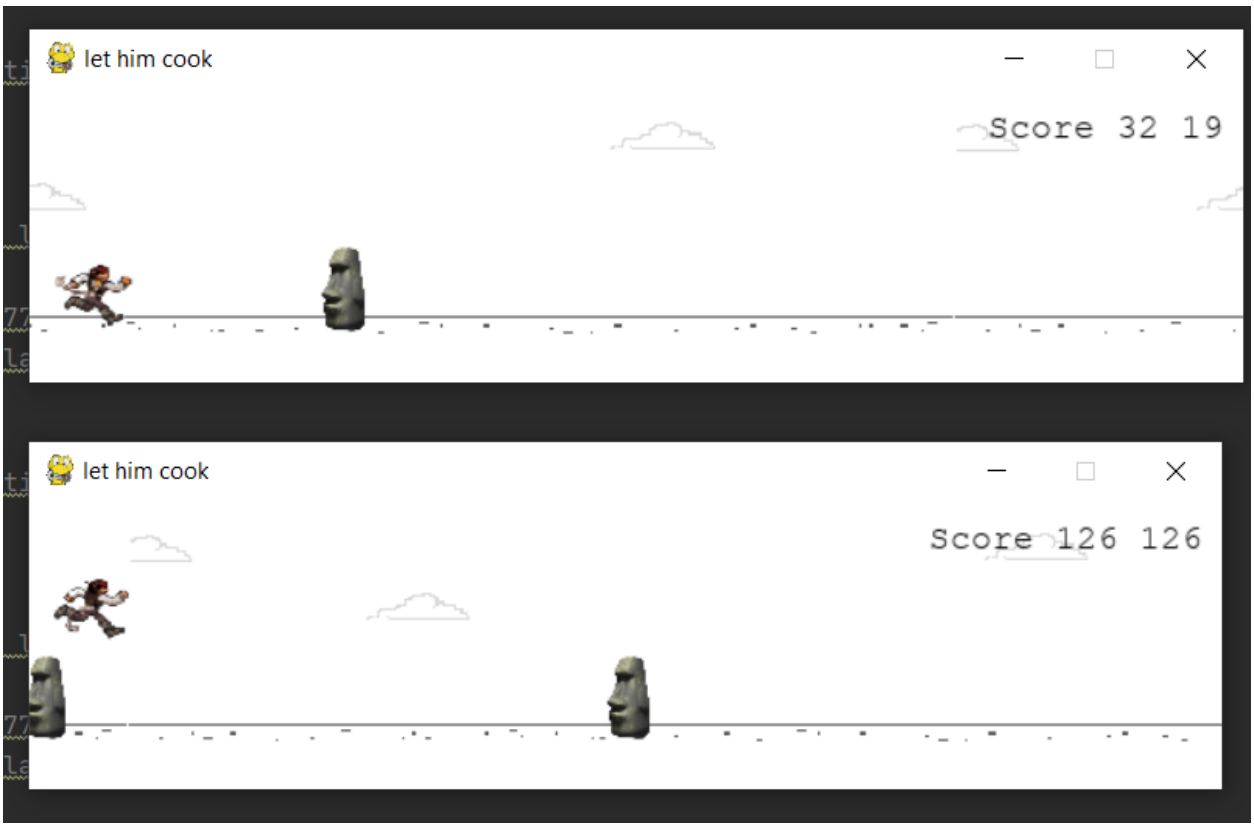

Run:
jackyyy (1)
Unittests in test.py
Test Results 133 ms
test 133 ms
TestBackground 33 ms
test_init 12 ms
test_set_texture 6 ms
test_show 5 ms
test_update 10 ms
TestCollision 2 ms
test_between 2 ms
TestJackyyy 50 ms
test_fall 5 ms
test_init 4 ms
test_jump 8 ms
test_set_sound 5 ms
test_set_texture 5 ms
test_show 6 ms
test_stop 5 ms
test_update_falling 3 ms
test_update_jumping 2 ms
test_update_walking 7 ms
TestMoai 31 ms
test_init 7 ms
test_set_texture 11 ms
test_show 7 ms
test_update 6 ms
TestScore 17 ms
test_check_highestscore 9 ms
test_init 3 ms
test_reset 2 ms
test_update 3 ms

# 5. Загальна діаграма коду

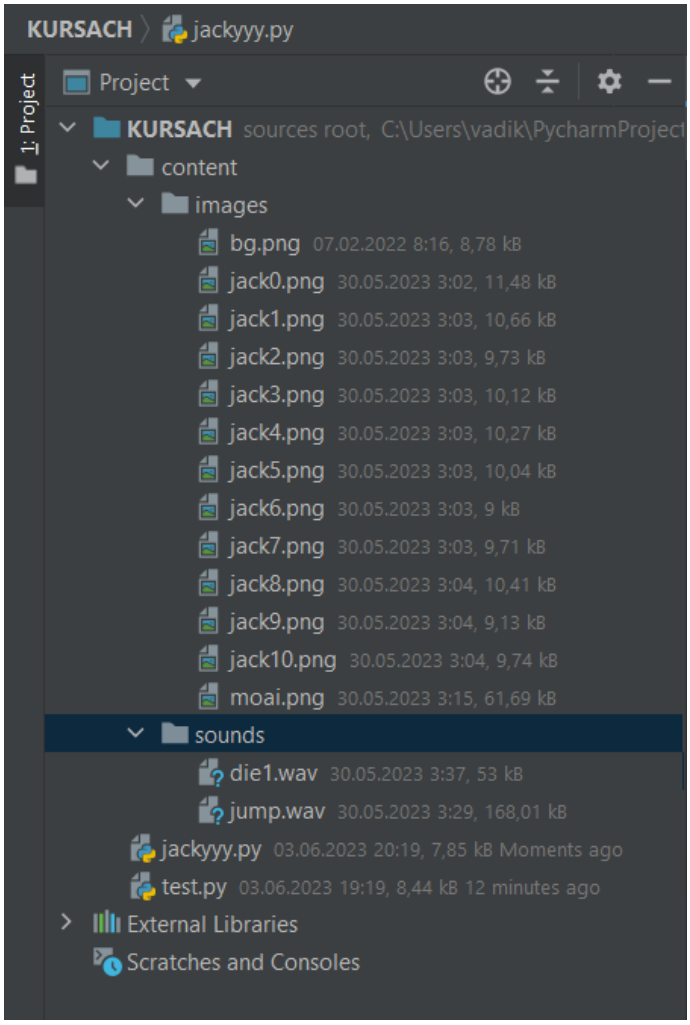


# 6. Скріншоти гри






## 7. Структура проекту



- **bg.png -** картинка бекграунду

bg

- **Jack0.png, Jack1.png…Jack10.png** – 11 положень героя, циклічна зміна яких створює “пересування” персонажа


- **Moai.png** – картинка перешкоди у грі

moai

- **die1.wav** – звук, який чує гравець під час смерті героя

die1.wav

- **jump.wav** – звук, який чує гравець під час стрибка

jump.wav

# 8. Вихідний код проекту та посилання на Github

Додаю посилання на мій Github репозиторій:[https://github.com/vademn/KursachIPZ](https://github.com/vademn/KursachIPZ)

**jackyyy.py:**

```
import sys
import pygame
import os
import random
import math

WIDTH = 623
HEIGHT = 150
pygame.init()
pygame.mixer.init()
screen = pygame.display.set_mode((WIDTH, HEIGHT))
pygame.display.set_caption('let him cook')

class Background:

    def __init__(self, x):
        self.width = WIDTH
        self.height = HEIGHT
        self.x = x
        self.y = 0
        self.set_texture()
        self.show()

    def update(self, dx):
        self.x += dx
        if self.x <= -WIDTH:
            self.x = WIDTH

    def show(self):
        screen.blit(self.texture, (self.x, self.y))

    def set_texture(self):
        path = os.path.join('content/images/bg.png')
        self.texture = pygame.image.load(path)
        self.texture = pygame.transform.scale(self.texture, (self.width,
self.height))

class Game:

    def __init__(self, highestscore=0):
        self.bg = [Background(x=0),Background(x=WIDTH)]
        self.jackyyy = Jackyyy()
        self.obstacles = []
        self.collision = Collision()
        self.score = Score(highestscore)
        self.speed = 3
        self.playing = False
        self.set_sound()
        self.set_labels()

    def set_labels(self):
        big_font = pygame.font.SysFont('monospace', 22, bold = True)
        small_font = pygame.font.SysFont('monospace', 18, bold=False)
        self.big_lbl = big_font.render(f'J A C K Y Y Y  D E A D', 1, (0, 0,
0))
        self.small_lbl = small_font.render(f'*press esc to let him cook
again*', 1, (0, 0, 0))
```

```
    def set_sound(self):
        path = os.path.join('content/sounds/die1.wav')
        self.sound = pygame.mixer.Sound(path)

    def start(self):
        self.playing = True

    def over(self):
        self.sound.play()
        screen.blit(self.big_lbl, (WIDTH // 2 - self.big_lbl.get_width() //
2, HEIGHT // 4))
        screen.blit(self.small_lbl, (WIDTH // 2 - self.small_lbl.get_width()
// 2, HEIGHT // 2))
        self.playing = False

    def tospawn(self, loops):
        return loops % 100 == 0

    def spawn_moai(self):
        # list with moai
        if len(self.obstacles) > 0:
            prev_moai = self.obstacles[-1]
            x = random.randint(prev_moai.x + self.jackyyy.width + 100, WIDTH
+ prev_moai.x + self.jackyyy.width + 100)

        # empty list
        else:
            x = random.randint(WIDTH + 100, 1000)

        # create the new moai
        moai = Moai(x)
        self.obstacles.append(moai)

    def restart_game(self):
        self.__init__(highestscore=self.score.highestscore)

class Jackyyy:

    def __init__(self):
        self.width = 44
        self.height = 44
        self.x = 10
        self.y = 80
        self.texture_number = 0
        self.dy = 3
        self.gravity = 1.23
        self.onground = True
        self.jumping = False
        self.jump_stop = 10
        self.falling = False
        self.fall_stop = self.y
        self.set_texture()
        self.set_sound()
        self.show()

    def update(self, loops):
        #jumping
        if self.jumping:
            self.y -= self.dy
            if self.y <= self.jump_stop:
                self.fall()

        #falling
```

```
        elif self.falling:
            self.y += self.gravity * self.dy
            if self.y >= self.fall_stop:
                self.stop()

        # walking
        elif self.onground and loops % 7 == 0:
           self.texture_number=(self.texture_number+1) % 11 #returns
0,1,2,...,9,10
           self.set_texture()


    def show(self):
        screen.blit(self.texture, (self.x, self.y))

    def set_texture(self):
        path = os.path.join(f'content/images/jack{self.texture_number}.png')
        self.texture = pygame.image.load(path)
        self.texture = pygame.transform.scale(self.texture, (self.width,
self.height))

    def set_sound(self):
        path = os.path.join('content/sounds/jump.wav')
        self.sound = pygame.mixer.Sound(path)

    def jump(self):
        self.sound.play()
        self.jumping = True
        self.onground = False

    def fall(self):
        self.jumping = False
        self.falling = True

    def stop(self):
        self.falling = False
        self.onground = True

class Moai:

    def __init__(self, x):
        self.width = 34
        self.height = 44
        self.x = x
        self.y = 80
        self.set_texture()
        self.show()

    def update(self, dx):
        self.x += dx

    def show(self):
        screen.blit(self.texture, (self.x, self.y))

    def set_texture(self):
        path = os.path.join('content/images/moai.png')
        self.texture = pygame.image.load(path)
        self.texture = pygame.transform.scale(self.texture, (self.width,
self.height))

class Collision:

        def between(self, obj1, obj2):
            distance = math.sqrt((obj1.x - obj2.x) ** 2 + (obj1.y - obj2.y)
```

```
** 2)#формула дистанції d = sqrt((x2-x1)^2 + (y2-y1)^2)
            return distance < 35

class Score:
    def __init__(self, highestscore):
        self.highestscore = highestscore
        self.actualscore = 0
        self.font = pygame.font.SysFont('monospace', 18)
        self.color = (0, 0, 0)
        self.show()

    def update(self, loops):
            self.actualscore = loops // 13 #за допомогою числа міняємо
швидкість нарахування балів
            self.check_highestscore()

    def show(self):
       self.lbl = self.font.render(f'Score {self.highestscore}
{self.actualscore}', 1, self.color)
       lbl_width = self.lbl.get_rect().width
       screen.blit(self.lbl, (WIDTH - lbl_width - 10, 10))

    def check_highestscore(self):
        if self.actualscore >= self.highestscore:
            self.highestscore = self.actualscore

    def reset(self):
        self.actualscore = 0

def main():
    # objects
    game = Game()
    jackyyy = game.jackyyy

    # variables
    clock = pygame.time.Clock()
    loops = 0
    over = False

    # mainloop
    while True:
        if game.playing:

         loops += 1

         # Background
         for bg in game.bg:
            bg.update(-game.speed)
            bg.show()

         # jackyyy
         jackyyy.update(loops)
         jackyyy.show()

         # moai
         if loops == 1:
             game.spawn_moai()

         if game.tospawn(loops):
            game.spawn_moai()

         for moai in game.obstacles:
          moai.update(-game.speed)
          moai.show()
```

```
            # collision
            if game.collision.between(jackyyy, moai):
                over = True

            if over:
                game.over()

            # score
            game.score.update(loops)
            game.score.show()

        # events
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                pygame.quit()
                sys.exit()

            if event.type == pygame.KEYDOWN:
                if event.key == pygame.K_SPACE:
                    if not over:
                        if jackyyy.onground:
                            jackyyy.jump()

                        if not game.playing:
                            game.start()

                    if event.key == pygame.K_ESCAPE:
                        game.restart_game()
                        jackyyy = game.jackyyy
                        loops = 0
                        over = False


        clock.tick(77)
        pygame.display.update()

main()
```

# 9. Список використаних джерел

- Посилання на звук з файлу **die1.wam** - https://youtu.be/xYJ63OTMDL4
- Посилання на звук з файлу **jump.wam** - https://youtu.be/auD_fT0KCQg
- Посилання на образ головного героя **(Капітан Джек Горобець)** - https://www.spriters-resource.com/custom_edited/disneycustoms/sheet/16621/
- Посилання на образ перешкоди **(Моаї)** - https://www.iemoji.com/view/emoji/814/travel-places/moai
- Посилання на вебсайт, що допомагав у розв'язку деяких питань щодо коду - https://stackoverflow.com/
- Посилання на документацію бібліотеки PyGame - https://www.pygame.org
- Посилання на сайт для побудови діаграм - https://app.diagrams.net/