

**Маланова Татьяна Денисовна**

242080

Практикум по программированию

Лабораторная работа 3

Простые астероиды

Роль: Разработчик игровой логики и системных компонентов

Состав команды: Маланова Татьяна Денисовна , Гарифуллин Тимур Альбертович.

**Гарифуллин Тимур Альбертович**

245114

Практикум по программированию

Лабораторная работа 3

Простые астероиды

Роль: Разработчик интерфейса и игрового движка

Состав команды: Гарифуллин Тимур Альбертович, Маланова Татьяна Денисовна .

## **Описание игры**

### **Простые астероиды**

Основная цель – игрок управляет кораблем и стреляет в астероиды , чтобы разбить их.

Изменения: Корабль расположен внизу экрана , астероиды разных классов – большие , средние , маленькие , есть сохранение рекордов , а также повышение уровней сложности.

### **Используемые инструменты:**

- Библиотека PyGame
- Встроенная оптимизированная проверка пересечения(столкновения) `collidect()`
- Сохранение данных в виде json
- Модули `math` и `random` для математических операций
- Модули `os` для работы с файлами и `sys` для создания путей поиска файлов

## **Разделения ролей и задач**

### **Подробное описание роли каждого участника**

#### **Маланова Татьяна Денисовна:**

Основные обязанности:

- Разработка базовых классов игровых объектов (`GameObject`, `MovingObject`)
- Реализация игровой механики (система столкновений, движение объектов)
- Создание классов `Player` и `Projectile`
- Настройка конфигурационных файлов (JSON)
- Реализация системы подсчёта очков
- Отладка и тестирование игрового процесса

#### **Гарифуллин Тимур Альбертович:**

Основные обязанности:

- Создание главного игрового движка (`GameEngine`)
- Разработка пользовательского интерфейса (UI, Menu)
- Реализация класса `Asteroid` с механикой разрушения
- Система уровней сложности и спавна волн астероидов
- Работа с файловой системой (сохранение рекордов)
- Оптимизация производительности и графического рендеринга

## **Методы сотрудничества и коммуникации**

Коммуникационные каналы

- Личные встречи для обсуждения архитектуры
- Мессенджеры для оперативных вопросов
- Совместное тестирование и отладка

## **Распределение задач по созданию графических элементов**

Маланова Т.Д.:

- Графическое представление игрока (корабль)
- Визуализация снарядов (`Projectile`)
- Цветовая схема игровых объектов

- Эффекты мигания при неуязвимости

Гарифуллин Т.А.:

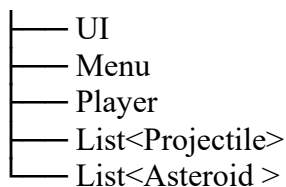
- Графическое представление астероидов
- Разработка пользовательского интерфейса:
- Меню игры (Menu)
- Интерфейс во время игры (UI)
- Таблица рекордов
- Анимация вращения астероидов
- Эффекты разрушения астероидов

### Архитектура проекта

**Полная диаграмма классов со связями:**

Game object <- Moving\_object <- Player/Asteroid/Projectile

Game\_engine



**Ключевые компоненты и объяснения:**

GameEngine : Управляет игровым циклом, координирует все объекты, обрабатывает состояния игры.

GameObject : Определяет общий интерфейс, реализует систему коллизий, база для всех игровых объектов.

MovingObject :Добавляет механику движения, наследуют: Player, Asteroid, Projectile.

Player : Обработка ввода и стрельбы, управление жизнями и состоянием

Asteroid : Движение и вращение астероидов, система разбивания на части

Projectile : Движение по прямой пуль , автоуничтожение по времени

**Использованные шаблоны проектирования:**

Методы ООП ,такие как : Инкапсуляция , наследование , полиморфизм , композиция .

В проекте используется шаблонный метод - то есть базовый класс GameObjectопределяет общие методы update() и draw(), которые переопределяются в дочерних классах для конкретной реализации.Также используется фабричный метод - то есть класс Asteroid имеет метод split(), который создает новые объекты астероидов при разрушении.Игровой цикл организует последовательную обработку ввода, обновления состояния и отрисовки кадров.

### Реализованный функционал

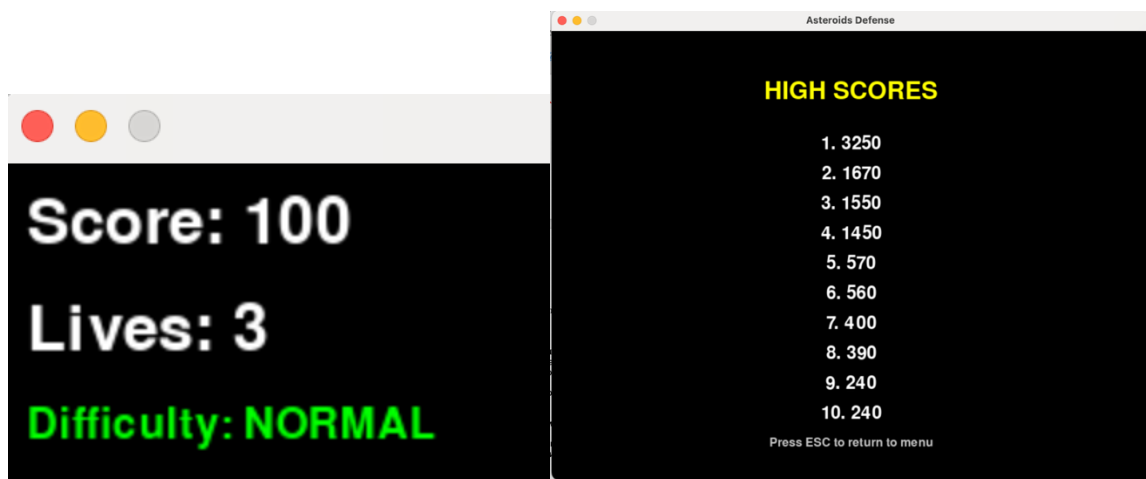
Требования:

- Реализация классического варианта игры
- Основная игровая механика и правила

Игровая механика выполнена : корабль может двигаться вправо/влево , стрелять по астероидам, получать очки .

- Система подсчета очков с отображением в реальном времени

За каждый разбитый астероид даются игровые очки , то есть за большой – 20 ,за средний – 35, за маленький – 50. Также сделана таблица рекордов , 10 самых высоких очков сохраняются в файл json и отображаются в игре .



- Управление с клавиатуры/мыши

Управление кораблем–право/лево. Начать игру заново–R.Поставить игру на паузу–P.Таблица рекордов – T. Все это также прописано в окне игры.

- Базовый пользовательский интерфейс
- Система обнаружения столкновений

Использована встроенная оптимизированная проверка пересечения(столкновения) colliderect()

- Игровые состояния (меню, игра, игра окончена)





## Инструкции по запуску и игре

В нашей версии игры:

Корабль находится внизу поля и двигается влево/вправо, стреляя по астероидам, летящим сверху вниз. Большие астероиды разбиваются на маленький и средний астероиды. За каждый разбитый астероид даются игровые очки, то есть за большой – 20, за средний – 35, за маленький – 50. У игрока есть три жизни, каждая из которых отнимается, когда корабль сталкивается с астероидом. В игре есть разные уровни сложности, то есть если игрок достигает 2000 очков, скорость астероидов увеличивается, если игрок достигает 5000 очков, скорость снова увеличивается. Цель игры – заработать как можно больше очков, избегая и разбивая астероиды.

Чтобы запустить файлы с игрой в терминале пишем `python src/main.py`

Далее появляется экран меню игры. В начале игры меню, из которого можно попасть в игру либо таблицу рекордов.

← / → - Движение корабля влево/вправо

P - Пауза

R - Начать новую игру

T - Просмотреть таблицу рекордов

ESC - Выйти в главное меню

SPACE – Стрельба

Уровень сложности и количество очков во время игры написаны на экране.

## Системные требования

Операционная система: Windows 10+, macOS 10.15+

Процессор: 1.5 GHz или выше

Оперативная память: 2 GB RAM

Место на диске: 50 MB свободного места

Python: Версия 3.8 или выше

PyGame: Версия 2.6.1

## Полный исходный код

Основные модули с пояснениями

1. Класс **game\_object** , от которого наследуются все игровые объекты .

Скриншот :

```
src > objects > game_object.py > ...
1  # Класс , от которого наследуются все игровые объекты/общий интерфейс: обновление, отрисовка, коллизии
2  import pygame
3
4  class GameObject:
5      def __init__(self, x, y, width, height):
6          # Позиция объекта в игровом мире
7          self.x = x # Координата X центра объекта
8          self.y = y # Координата Y центра объекта
9          # Размеры объекта (используются для коллизий)
10         self.width = width # Ширина объекта
11         self.height = height # Высота объекта
12         # если False, объект будет удален из игры
13         self.active = True
14
15     def update(self):
16         # Вызывается каждый кадр для обновления состояния объекта
17         pass
18
19     def draw(self, surface):
20         # Отрисовывает объект на переданной поверхности Pygame
21         pass
22
23     def get_rect(self):
24         # Возвращает прямоугольник Pygame для обнаружения столкновений
25         # Расчет от центра к краям (x - width/2)
26         return pygame.Rect(self.x - self.width/2, self.y - self.height/2,
27                             self.width, self.height)
28
29     def collides_with(self, other):
30         # Проверяет столкновение с другим игровым объектом
31         return self.get_rect().colliderect(other.get_rect())
```

Фрагмент кода:

```
# # Класс , от которого наследуются все игровые объекты/общий
# интерфейс: обновление, отрисовка, коллизии
# import pygame
#
# class GameObject:
#     def __init__(self, x, y, width, height):
#         # Позиция объекта в игровом мире
#         self.x = x # Координата X центра объекта
#         self.y = y # Координата Y центра объекта
#         # Размеры объекта (используются для коллизий)
#         self.width = width # Ширина объекта
#         self.height = height # Высота объекта
#         # если False, объект будет удален из игры
#         self.active = True
#
#     def update(self):
#         # Вызывается каждый кадр для обновления состояния
# объекта
#         pass
#
#     def draw(self, surface):
#         # Отрисовывает объект на переданной поверхности Pygame
#         pass
#
#     def get_rect(self):
#         # Возвращает прямоугольник Pygame для обнаружения
# столкновений
#         # Расчет от центра к краям (x - width/2)
```

```
#         return pygame.Rect(self.x - self.width/2, self.y -
self.height/2,
#                               self.width, self.height)

#     def collides_with(self, other):
#         # Проверяет столкновение с другим игровым объектом
#         return self.get_rect().colliderect(other.get_rect())
```

2. Класс движущихся объектов **moving\_object** , который наследует функцию `__init__` из класса `game_object` и задает движение объектам.

Скриншот :

```
src > objects > moving_object.py > ...
1  # Класс движущихся объектов
2  from src.objects.game_object import GameObject
3
4  class MovingObject(GameObject):
5      def __init__(self, x, y, width, height, speed):
6          # Вызов конструктора родительского класса
7          super().__init__(x, y, width, height)
8          # Скорость движения (пикселей за кадр)
9          self.speed = speed
10         # Направление движения по осям (-1, 0, 1)
11         self.dx = 0 # Направление по X: -1=влево, 0=стоп, 1=вправо
12         self.dy = 0 # Направление по Y: -1=вверх, 0=стоп, 1=вниз
13
14     def move(self):
15         # Обновляет позицию объекта на основе направления и скорости
16         self.x += self.dx * self.speed # Движение по горизонтали
17         self.y += self.dy * self.speed # Движение по вертикали
```

Фрагмент кода:

```
# # Класс движущихся объектов
# from src.objects.game_object import GameObject

# class MovingObject(GameObject):
#     def __init__(self, x, y, width, height, speed):
#         # Вызов конструктора родительского класса
#         super().__init__(x, y, width, height)
#         # Скорость движения (пикселей за кадр)
#         self.speed = speed
#         # Направление движения по осям (-1, 0, 1)
#         self.dx = 0 # Направление по X: -1=влево, 0=стоп,
1=вправо
#         self.dy = 0 # Направление по Y: -1=вверх, 0=стоп,
1=вниз

#     def move(self):
```

```

#           # Обновляет позицию объекта на основе направления и
скорости
#           self.x += self.dx * self.speed # Движение по
горизонтали
#           self.y += self.dy * self.speed # Движение по
вертикали

```

3. Класс player , который создает игровой корабль , жизни игрока , управление движением(в том числе с помощью кнопок право/лево на клавиатуре) , проверяет уязвимость и наследует движение объектов из moving\_object.

Скриншот :

```

src > objects > player.py > Player > __init__
1  import pygame
2  from src.objects.moving_object import MovingObject
3  from src.objects.projectile import Projectile
4
5  class Player(MovingObject):
6      def __init__(self, x, y, initial_lives=3, graphics_config=None):
7          player_width = 50
8          player_height = 30
9          if graphics_config and 'sizes' in graphics_config:
10             sizes = graphics_config['sizes']
11             player_width = sizes.get('player_width', 50)
12             player_height = sizes.get('player_height', 30)
13
14             super().__init__(x, y, player_width, player_height, 5)
15             self.lives = initial_lives
16             self.score = 0
17             self.shoot_cooldown = 0
18             self.invincible = 0
19             self.screen_width = 800
20         def update(self):
21             keys = pygame.key.get_pressed()
22             self.dx = keys[pygame.K_RIGHT] - keys[pygame.K_LEFT]
23             self.move()
24             self.x = max(self.width/2, min(self.screen_width - self.width/2, self.x))
25
26             if self.shoot_cooldown > 0:
27                 self.shoot_cooldown -= 1
28
29             if self.invincible > 0:
30                 self.invincible -= 1

```



```

32     def draw(self, surface):
33         if self.invincible > 0 and self.invincible % 10 < 5:
34             return
35
36         ship_rect = pygame.Rect(self.x - self.width/2, self.y - self.height/2,
37                                 self.width, self.height)
38         pygame.draw.rect(surface, (0, 255, 0), ship_rect)
39
40         points = [
41             (self.x - self.width/2, self.y - self.height/2),
42             (self.x + self.width/2, self.y - self.height/2),
43             (self.x, self.y - self.height)
44         ]
45         pygame.draw.polygon(surface, (0, 200, 0), points)
46
47     def shoot(self, graphics_config=None):
48         if self.shoot_cooldown == 0:
49             self.shoot_cooldown = 20
50             return Projectile(self.x, self.y - self.height, graphics_config)
51         return None
52
53     def take_damage(self):
54         if self.invincible == 0:
55             self.lives -= 1
56             self.invincible = 120
57             return True
58         return False

```

Фрагмент кода:

```

# import pygame
# from src.objects.moving_object import MovingObject
# from src.objects.projectile import Projectile

# class Player(MovingObject):
#     def __init__(self, x, y, initial_lives=3,
# graphics_config=None):
#         player_width = 50
#         player_height = 30
#         if graphics_config and 'sizes' in graphics_config:
#             sizes = graphics_config['sizes']
#             player_width = sizes.get('player_width', 50)
#             player_height = sizes.get('player_height', 30)

#         super().__init__(x, y, player_width, player_height, 5)
#         self.lives = initial_lives
#         self.score = 0
#         self.shoot_cooldown = 0
#         self.invincible = 0
#         self.screen_width = 800
#     def update(self):
#         keys = pygame.key.get_pressed()
#         self.dx = keys[pygame.K_RIGHT] - keys[pygame.K_LEFT]

```

```

#         self.move()
#         self.x = max(self.width/2, min(self.screen_width -
self.width/2, self.x))

#         if self.shoot_cooldown > 0:
#             self.shoot_cooldown -= 1

#         if self.invincible > 0:
#             self.invincible -= 1

#     def draw(self, surface):
#         if self.invincible > 0 and self.invincible % 10 < 5:
#             return

#         ship_rect = pygame.Rect(self.x - self.width/2, self.y
- self.height/2,
#                                 self.width, self.height)
#         pygame.draw.rect(surface, (0, 255, 0), ship_rect)

#         points = [
#             (self.x - self.width/2, self.y - self.height/2),
#             (self.x + self.width/2, self.y - self.height/2),
#             (self.x, self.y - self.height)
#         ]
#         pygame.draw.polygon(surface, (0, 200, 0), points)

#     def shoot(self, graphics_config=None):
#         if self.shoot_cooldown == 0:
#             self.shoot_cooldown = 20
#             return Projectile(self.x, self.y - self.height,
graphics_config)
#         return None

#     def take_damage(self):
#         if self.invincible == 0:
#             self.lives -= 1
#             self.invincible = 120
#             return True
#         return False

```

4. Класс projectile , который наследует движение из moving\_object и создает снаряд.

Скриншот:

```

src > objects > projectile.py > Projectile > __init__
1  from src.objects.moving_object import MovingObject
2  import pygame
3
4  class Projectile(MovingObject):
5      def __init__(self, x, y, graphics_config=None):
6          projectile_width = 8
7          projectile_height = 16
8
9          if graphics_config and 'sizes' in graphics_config:
10             sizes = graphics_config['sizes']
11             projectile_width = sizes.get('projectile_width', 8)
12             projectile_height = sizes.get('projectile_height', 16)
13
14             super().__init__(x, y, projectile_width, projectile_height, 8)
15             self.dy = -1
16             self.lifetime = 90
17
18     def update(self):
19         self.move()
20         self.lifetime -= 1
21         if self.lifetime <= 0 or self.y < 0:
22             self.active = False
23
24     def draw(self, surface):
25         bullet_rect = pygame.Rect(self.x - self.width/2, self.y - self.height/2,
26                                   self.width, self.height)
27         pygame.draw.rect(surface, (255, 255, 0), bullet_rect)
28
29         glow_rect = pygame.Rect(self.x - self.width/2 - 2, self.y - self.height/2 - 2,
30                                 self.width + 4, self.height + 4)
31         pygame.draw.rect(surface, (255, 255, 100), glow_rect, 1)

```

Фрагмент кода:

```

# from src.objects.moving_object import MovingObject
# import pygame

# class Projectile(MovingObject):
#     def __init__(self, x, y, graphics_config=None):
#         projectile_width = 8
#         projectile_height = 16

#         if graphics_config and 'sizes' in graphics_config:
#             sizes = graphics_config['sizes']
#             projectile_width = sizes.get('projectile_width',
# 8)
#             projectile_height = sizes.get('projectile_height',
# 16)

#             super().__init__(x, y, projectile_width,
# projectile_height, 8)
#             self.dy = -1
#             self.lifetime = 90

#     def update(self):

```

```

#         self.move()
#         self.lifetime -= 1
#         if self.lifetime <= 0 or self.y < 0:
#             self.active = False

#     def draw(self, surface):
#         bullet_rect = pygame.Rect(self.x - self.width/2,
self.y - self.height/2,
#                                     self.width, self.height)
#         pygame.draw.rect(surface, (255, 255, 0), bullet_rect)

#         glow_rect = pygame.Rect(self.x - self.width/2 - 2,
self.y - self.height/2 - 2,
#                                     self.width + 4, self.height +
4)
#         pygame.draw.rect(surface, (255, 255, 100), glow_rect,
1)

```

5. Класс `asteroid` , который задает параметры астероида , очки за уничтожение , наследует движение из `moving_object`,разделяет большой астероид на средний и маленький и рисует его.

Скриншот :

```

src > objects > asteroid.py > Asteroid > __init__
1  import pygame
2  import math
3  import random
4  from src.objects.moving_object import MovingObject
5  class Asteroid(MovingObject):
6      def __init__(self, x, y, size=3, can_split=True, graphics_config=None):
7          sizes = {3: 50, 2: 35, 1: 20}
8          if graphics_config and 'asteroid_sizes' in graphics_config.get('sizes', {}):
9              asteroid_sizes = graphics_config['sizes']['asteroid_sizes']
10             sizes = {3: asteroid_sizes.get('3', 50), 2: asteroid_sizes.get('2', 35), 1: asteroid_sizes.get('1', 20)}
11             speed = {3: 1.5, 2: 2.0, 1: 2.5}
12             super().__init__(x, y, sizes[size], sizes[size], speed[size])
13             self.size = size
14             self.can_split = can_split
15             self.base_speed = speed[size]
16             self.points = {3: 20, 2: 30, 1: 50}[size]
17             self.dy = 1
18             self.dx = random.uniform(-0.3, 0.3)
19             self.rotation = random.uniform(0, 360)
20             self.rotation_speed = random.uniform(-3, 3)
21         def update(self):
22             self.move()
23             self.rotation += self.rotation_speed
24             if self.y > 800 + self.height:
25                 self.active = False
26         def draw(self, surface):
27             points = []
28             for i in range(8):
29                 angle = 2 * math.pi * i / 8 + math.radians(self.rotation)
30                 radius = self.width / 2 * random.uniform(0.8, 1.2)
31                 points.append((
32                     self.x + math.cos(angle) * radius,
33                     self.y + math.sin(angle) * radius
34                 ))
35             pygame.draw.polygon(surface, (255, 0, 0), points)
36             pygame.draw.polygon(surface, (200, 0, 0), points, 2)
37         def split(self):
38             if self.size == 3 and self.can_split:
39                 return [
40                     Asteroid(self.x, self.y, 2, False),
41                     Asteroid(self.x, self.y, 1, False)
42                 ]
43             return []

```

Фрагмент кода:

```

# import pygame
# import math
# import random
# from src.objects.moving_object import MovingObject
# class Asteroid(MovingObject):
#     def __init__(self, x, y, size=3, can_split=True,
graphics_config=None):
#         sizes = {3: 50, 2: 35, 1: 20}
#         if graphics_config and 'asteroid_sizes' in
graphics_config.get('sizes', {}):
#             asteroid_sizes =
graphics_config['sizes']['asteroid_sizes']
#             sizes = {3: asteroid_sizes.get('3', 50), 2:
asteroid_sizes.get('2', 35), 1: asteroid_sizes.get('1', 20)}
#             speed = {3: 1.5, 2: 2.0, 1: 2.5}
#             super().__init__(x, y, sizes[size], sizes[size],
speed[size])
#             self.size = size
#             self.can_split = can_split

```

```

#         self.base_speed = speed[size]
#         self.points = {3: 20, 2: 30, 1: 50}[size]
#         self.dy = 1
#         self.dx = random.uniform(-0.3, 0.3)
#         self.rotation = random.uniform(0, 360)
#         self.rotation_speed = random.uniform(-3, 3)
#     def update(self):
#         self.move()
#         self.rotation += self.rotation_speed
#         if self.y > 800 + self.height:
#             self.active = False
#     def draw(self, surface):
#         points = []
#         for i in range(8):
#             angle = 2 * math.pi * i / 8 +
math.radians(self.rotation)
#             radius = self.width / 2 * random.uniform(0.8, 1.2)
#             points.append((
#                 self.x + math.cos(angle) * radius,
#                 self.y + math.sin(angle) * radius
#             ))
#         pygame.draw.polygon(surface, (255, 0, 0), points)
#         pygame.draw.polygon(surface, (200, 0, 0), points, 2)
#     def split(self):
#         if self.size == 3 and self.can_split:
#             return [
#                 Asteroid(self.x, self.y, 2, False),
#                 Asteroid(self.x, self.y, 1, False)
#             ]
#         return []

```

6. Класс ui , который отображает счет игрока , жизни , уровень сложности ,состояние игры ,настройки управления, таблицу рекордов.

Скриншот :

```

src > ui > hud.py > UI > draw
1  import pygame
2
3  class UI:
4      def __init__(self):
5          self.font = pygame.font.Font(None, 36)
6          self.small_font = pygame.font.Font(None, 24)
7
8      def draw(self, surface, player, game_state, difficulty_level=1):
9          score_text = self.font.render(f"Score: {player.score}", True, (255, 255, 255))
10         surface.blit(score_text, (10, 10))
11
12         lives_text = self.font.render(f"Lives: {player.lives}", True, (255, 255, 255))
13         surface.blit(lives_text, (10, 50))
14
15         difficulty_names = {1: "NORMAL", 2: "HARD", 3: "EXTREME"}
16         difficulty_color = {1: (0, 255, 0), 2: (255, 165, 0), 3: (255, 0, 0)}
17         diff_text = self.small_font.render(f"Difficulty: {difficulty_names[difficulty_level]}",
18         True, difficulty_color[difficulty_level])
19         surface.blit(diff_text, (10, 90))
20         if game_state == "game_over":
21             self._draw_centered(surface, "GAME OVER - Press R to restart", self.font)
22             self._draw_centered(surface, "Press T to view high scores", self.small_font, 40)
23         elif game_state == "paused":
24             self._draw_centered(surface, "PAUSED - Press P to continue", self.font)
25         controls = "Controls: LEFT/RIGHT - Move, SPACE - Shoot, P - Pause"
26         controls_text = self.small_font.render(controls, True, (200, 200, 200))
27         surface.blit(controls_text, (10, 570))
28     def _draw_centered(self, surface, text, font, offset=0):
29         text_surface = font.render(text, True, (255, 255, 255))
30         rect = text_surface.get_rect(center=(400, 300 + offset))
31         surface.blit(text_surface, rect)
32     def draw_high_scores(self, surface, high_scores):
33         title = pygame.font.Font(None, 48).render("HIGH SCORES", True, (255, 255, 0))
34         surface.blit(title, title.get_rect(center=(400, 80)))
35         if not high_scores:
36             self._draw_centered(surface, "No scores yet!", self.font)
37         else:
38             for i, score in enumerate(high_scores):
39                 score_text = self.font.render(f"{i+1}. {score}", True, (255, 255, 255))
40                 rect = score_text.get_rect(center=(400, 150 + i * 40))
41                 surface.blit(score_text, rect)
42         back_text = self.small_font.render("Press ESC to return to menu", True, (200, 200, 200))
43         surface.blit(back_text, back_text.get_rect(center=(400, 550)))

```

Фрагмент кода:

```
# import pygame
```

```
# class UI:
```

```
#     def __init__(self):
```

```
#         self.font = pygame.font.Font(None, 36)
```

```
#         self.small_font = pygame.font.Font(None, 24)
```

```
#     def draw(self, surface, player, game_state,
# difficulty_level=1):
```

```
#         score_text = self.font.render(f"Score:
# {player.score}", True, (255, 255, 255))
```

```
#         surface.blit(score_text, (10, 10))
```

```

#         lives_text = self.font.render(f"Lives:
{player.lives}", True, (255, 255, 255))
#         surface.blit(lives_text, (10, 50))

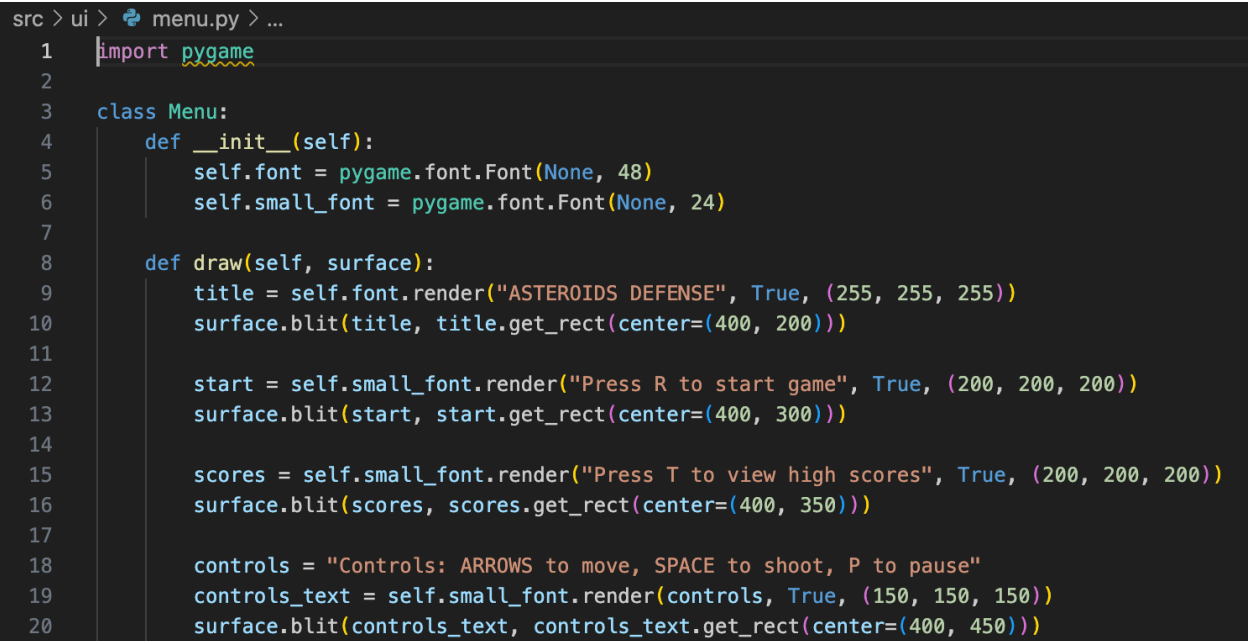
#         difficulty_names = {1: "NORMAL", 2: "HARD", 3:
"EXTREME"}
#         difficulty_color = {1: (0, 255, 0), 2: (255, 165, 0),
3: (255, 0, 0)}
#         diff_text = self.small_font.render(f"Difficulty:
{difficulty_names[difficulty_level]}",
#                                           True,
difficulty_color[difficulty_level])
#         surface.blit(diff_text, (10, 90))
#         if game_state == "game_over":
#             self._draw_centered(surface, "GAME OVER – Press R
to restart", self.font)
#             self._draw_centered(surface, "Press T to view high
scores", self.small_font, 40)
#         elif game_state == "paused":
#             self._draw_centered(surface, "PAUSED – Press P to
continue", self.font)
#         controls = "Controls: LEFT/RIGHT – Move, SPACE –
Shoot, P – Pause"
#         controls_text = self.small_font.render(controls, True,
(200, 200, 200))
#         surface.blit(controls_text, (10, 570))
#         def _draw_centered(self, surface, text, font, offset=0):
#             text_surface = font.render(text, True, (255, 255,
255))
#             rect = text_surface.get_rect(center=(400, 300 +
offset))
#             surface.blit(text_surface, rect)
#         def draw_high_scores(self, surface, high_scores):
#             title = pygame.font.Font(None, 48).render("HIGH
SCORES", True, (255, 255, 0))
#             surface.blit(title, title.get_rect(center=(400, 80)))
#             if not high_scores:
#                 self._draw_centered(surface, "No scores yet!",
self.font)
#             else:
#                 for i, score in enumerate(high_scores):
#                     score_text = self.font.render(f"{i+1}.
{score}", True, (255, 255, 255))
#                     rect = score_text.get_rect(center=(400, 150 +
i * 40))
#                     surface.blit(score_text, rect)
#             back_text = self.small_font.render("Press ESC to
return to menu", True, (200, 200, 200))
#             surface.blit(back_text,
back_text.get_rect(center=(400, 550)))

```



7. Класс меню отображает меню(заголовок, переход к игре и таблице рекордов, информацию о управлении)

Скриншот :



```
src > ui > menu.py > ...
1  import pygame
2
3  class Menu:
4      def __init__(self):
5          self.font = pygame.font.Font(None, 48)
6          self.small_font = pygame.font.Font(None, 24)
7
8      def draw(self, surface):
9          title = self.font.render("ASTEROIDS DEFENSE", True, (255, 255, 255))
10         surface.blit(title, title.get_rect(center=(400, 200)))
11
12         start = self.small_font.render("Press R to start game", True, (200, 200, 200))
13         surface.blit(start, start.get_rect(center=(400, 300)))
14
15         scores = self.small_font.render("Press T to view high scores", True, (200, 200, 200))
16         surface.blit(scores, scores.get_rect(center=(400, 350)))
17
18         controls = "Controls: ARROWS to move, SPACE to shoot, P to pause"
19         controls_text = self.small_font.render(controls, True, (150, 150, 150))
20         surface.blit(controls_text, controls_text.get_rect(center=(400, 450)))
```

Фрагмент кода:

```
# import pygame

# class Menu:
#     def __init__(self):
#         self.font = pygame.font.Font(None, 48)
#         self.small_font = pygame.font.Font(None, 24)

#     def draw(self, surface):
#         title = self.font.render("ASTEROIDS DEFENSE", True,
# (255, 255, 255))
#         surface.blit(title, title.get_rect(center=(400, 200)))

#         start = self.small_font.render("Press R to start
game", True, (200, 200, 200))
#         surface.blit(start, start.get_rect(center=(400, 300)))

#         scores = self.small_font.render("Press T to view high
scores", True, (200, 200, 200))
#         surface.blit(scores, scores.get_rect(center=(400,
350)))

#         controls = "Controls: ARROWS to move, SPACE to shoot,
P to pause"
#         controls_text = self.small_font.render(controls, True,
(150, 150, 150))
#         surface.blit(controls_text,
controls_text.get_rect(center=(400, 450)))
```

8. Класс `game_engine` наследует классы `player`, `asteroid`, `projectile`, `ui`, `menu` , загружает рекорды в json файл, сохраняет новый рекорд , обновляет уровни сложности по очкам, создает новые астероиды, обрабатывает события ввода , обновляет игровое состояние , отрисовывает все в игре и запускает главный игровой цикл.

Скриншот :

src > engine >  game\_engine.py >  GameEngine >  update\_difficulty

```
1  import pygame
2  import random
3  import json
4  import os
5  from src.objects.player import Player
6  from src.objects.asteroid import Asteroid
7  from src.objects.projectile import Projectile
8  from src.ui.hud import UI
9  from src.ui.menu import Menu
10
11 class GameEngine:
12     def __init__(self):
13         self.game_config = self.load_config('src/config/game_config.json')
14         self.graphics_config = self.load_config('src/config/graphics.json')
15
16         screen_width = self.game_config['game']['screen_width']
17         screen_height = self.game_config['game']['screen_height']
18         title = self.game_config['game']['title']
19
20         self.screen = pygame.display.set_mode((screen_width, screen_height))
21         pygame.display.set_caption(title)
22         self.clock = pygame.time.Clock()
23         self.running = True
24         self.game_state = "menu"
25
26         initial_lives = self.game_config['game']['initial_lives']
27         self.player = Player(screen_width // 2, screen_height - 50,
28                               initial_lives, self.graphics_config)
29         self.player.screen_width = screen_width
30
31         self.projectiles = []
32         self.asteroids = []
33         self.ui = UI()
34         self.menu = Menu()
35
36         self.spawn_timer = 0
37         self.spawn_interval = 180
38         self.wave_count = 0
39         self.max_asteroids = 5
40
41         self.high_scores = self.load_high_scores()
42         self.difficulty_level = 1
43         self.speed_multiplier = 1.0
44
45         self.spawn_wave(2)
```

```

47     def load_config(self, file_path):
48         try:
49             with open(file_path, 'r') as f:
50                 return json.load(f)
51         except Exception as e:
52             print(f"Ошибка загрузки {file_path}: {e}")
53             return {}
54
55     def load_high_scores(self):
56         try:
57             if os.path.exists('high_scores.json'):
58                 with open('high_scores.json', 'r') as f:
59                     return json.load(f)
60         except:
61             pass
62         return []
63
64     def save_high_score(self, score):
65         self.high_scores.append(score)
66         self.high_scores.sort(reverse=True)
67         self.high_scores = self.high_scores[:10]
68         with open('high_scores.json', 'w') as f:
69             json.dump(self.high_scores, f)
70
71     def update_difficulty(self):
72         if self.player.score >= 5000:
73             self.difficulty_level, self.speed_multiplier = 3, 1.8
74         elif self.player.score >= 2000:
75             self.difficulty_level, self.speed_multiplier = 2, 1.4
76         else:
77             self.difficulty_level, self.speed_multiplier = 1, 1.0
78
79         for asteroid in self.asteroids:
80             asteroid.speed = asteroid.base_speed * self.speed_multiplier
81
82     def spawn_asteroid(self):
83         screen_width = self.game_config['game']['screen_width']
84         x = random.uniform(50, screen_width - 50)
85         asteroid = Asteroid(x, -30, 3, True, self.graphics_config)
86         asteroid.speed = asteroid.base_speed * self.speed_multiplier
87         self.asteroids.append(asteroid)
88

```

```

89     def spawn_wave(self, count):
90         for _ in range(min(count, self.max_asteroids - len(self.asteroids))):
91             self.spawn_asteroid()
92         self.wave_count += 1
93
94     def handle_events(self):
95         for event in pygame.event.get():
96             if event.type == pygame.QUIT:
97                 self.running = False
98             elif event.type == pygame.KEYDOWN:
99                 if event.key == pygame.K_ESCAPE:
100                     if self.game_state == "high_scores":
101                         self.game_state = "menu"
102                     else:
103                         self.running = False
104                 elif event.key == pygame.K_p and self.game_state in ["playing", "paused"]:
105                     self.game_state = "paused" if self.game_state == "playing" else "playing"
106                 elif event.key == pygame.K_r and self.game_state in ["menu", "game_over"]:
107                     self.start_game()
108                 elif event.key == pygame.K_t and self.game_state in ["menu", "game_over"]:
109                     self.game_state = "high_scores"
110                 elif event.key == pygame.K_SPACE and self.game_state == "playing":
111                     # NEPEQAEM graphics_config & shoot()
112                     if projectile := self.player.shoot(self.graphics_config):
113                         self.projectiles.append(projectile)
114
115     def start_game(self):
116         screen_width = self.game_config['game']['screen_width']
117         screen_height = self.game_config['game']['screen_height']
118         initial_lives = self.game_config['game']['initial_lives']
119
120         self.player = Player(screen_width // 2, screen_height - 50,
121                               initial_lives, self.graphics_config)
122         self.player.screen_width = screen_width
123         self.projectiles.clear()
124         self.asteroids.clear()
125         self.game_state = "playing"
126         self.wave_count = 0
127         self.difficulty_level = 1
128         self.speed_multiplier = 1.0
129         self.spawn_wave(2)

```

```

131     def update(self):
132         if self.game_state != "playing":
133             return
134
135         self.update_difficulty()
136         self.player.update()
137
138         for projectile in self.projectiles[:]:
139             projectile.update()
140             if not projectile.active:
141                 self.projectiles.remove(projectile)
142
143         for asteroid in self.asteroids[:]:
144             asteroid.update()
145             if not asteroid.active:
146                 self.asteroids.remove(asteroid)
147                 continue
148
149             if asteroid.collides_with(self.player) and self.player.take_damage():
150                 if self.player.lives <= 0:
151                     if self.player.score > 0:
152                         self.save_high_score(self.player.score)
153                         self.game_state = "game_over"
154
155             for projectile in self.projectiles[:]:
156                 if asteroid.collides_with(projectile):
157                     self.player.score += asteroid.points
158                     self.asteroids.remove(asteroid)
159                     self.projectiles.remove(projectile)
160
161                     for new_asteroid in asteroid.split():
162                         new_asteroid.speed = new_asteroid.base_speed * self.speed_multiplier
163                         self.asteroids.append(new_asteroid)
164                     break
165
166         self.spawn_timer += 1
167         if (self.spawn_timer >= self.spawn_interval and
168             len(self.asteroids) < self.max_asteroids - 1):
169             self.spawn_wave(random.randint(1, min(3, 1 + self.wave_count // 5)))
170             self.spawn_timer = 0

```

```

172     def draw(self):
173         self.screen.fill((0, 0, 0))
174
175         if self.game_state == "menu":
176             self.menu.draw(self.screen)
177         elif self.game_state == "high_scores":
178             self.ui.draw_high_scores(self.screen, self.high_scores)
179         else:
180             for obj in self.asteroids + self.projectiles:
181                 obj.draw(self.screen)
182             self.player.draw(self.screen)
183             self.ui.draw(self.screen, self.player, self.game_state, self.difficulty_level)
184
185         pygame.display.flip()
186
187     def run(self):
188         fps = self.game_config['game']['fps']
189         while self.running:
190             self.handle_events()
191             self.update()
192             self.draw()
193             self.clock.tick(fps)

```

Фрагмент кода:

```

# import pygame
# import random
# import json
# import os
# from src.objects.player import Player
# from src.objects.asteroid import Asteroid
# from src.objects.projectile import Projectile
# from src.ui.hud import UI
# from src.ui.menu import Menu

# class GameEngine:
#     def __init__(self):
#         self.game_config =
self.load_config('src/config/game_config.json')
#         self.graphics_config =
self.load_config('src/config/graphics.json')

#         screen_width = self.game_config['game']['screen_width']
#         screen_height =
self.game_config['game']['screen_height']
#         title = self.game_config['game']['title']

#         self.screen = pygame.display.set_mode((screen_width,
screen_height))
#         pygame.display.set_caption(title)
#         self.clock = pygame.time.Clock()
#         self.running = True
#         self.game_state = "menu"

```

```

#         initial_lives =
self.game_config['game']['initial_lives']
#         self.player = Player(screen_width // 2, screen_height
- 50,
#                                 initial_lives, self.graphics_config)
#         self.player.screen_width = screen_width

#         self.projectiles = []
#         self.asteroids = []
#         self.ui = UI()
#         self.menu = Menu()

#         self.spawn_timer = 0
#         self.spawn_interval = 180
#         self.wave_count = 0
#         self.max_asteroids = 5

#         self.high_scores = self.load_high_scores()
#         self.difficulty_level = 1
#         self.speed_multiplier = 1.0

#         self.spawn_wave(2)

#     def load_config(self, file_path):
#         try:
#             with open(file_path, 'r') as f:
#                 return json.load(f)
#         except Exception as e:
#             print(f"Ошибка загрузки {file_path}: {e}")
#             return {}

#     def load_high_scores(self):
#         try:
#             if os.path.exists('high_scores.json'):
#                 with open('high_scores.json', 'r') as f:
#                     return json.load(f)
#         except:
#             pass
#         return []

#     def save_high_score(self, score):
#         self.high_scores.append(score)
#         self.high_scores.sort(reverse=True)
#         self.high_scores = self.high_scores[:10]
#         with open('high_scores.json', 'w') as f:
#             json.dump(self.high_scores, f)

#     def update_difficulty(self):
#         if self.player.score >= 5000:
#             self.difficulty_level, self.speed_multiplier = 3,
1.8
#             elif self.player.score >= 2000:

```





```

#     def start_game(self):
#         screen_width = self.game_config['game']['screen_width']
#         screen_height =
self.game_config['game']['screen_height']
#         initial_lives =
self.game_config['game']['initial_lives']

#         self.player = Player(screen_width // 2, screen_height
- 50,
#                               initial_lives, self.graphics_config)
#         self.player.screen_width = screen_width
#         self.projectiles.clear()
#         self.asteroids.clear()
#         self.game_state = "playing"
#         self.wave_count = 0
#         self.difficulty_level = 1
#         self.speed_multiplier = 1.0
#         self.spawn_wave(2)

#     def update(self):
#         if self.game_state != "playing":
#             return

#         self.update_difficulty()
#         self.player.update()

#         for projectile in self.projectiles[:]:
#             projectile.update()
#             if not projectile.active:
#                 self.projectiles.remove(projectile)

#         for asteroid in self.asteroids[:]:
#             asteroid.update()
#             if not asteroid.active:
#                 self.asteroids.remove(asteroid)
#             continue

#             if asteroid.collides_with(self.player) and
self.player.take_damage():
#                 if self.player.lives <= 0:
#                     if self.player.score > 0:
#                         self.save_high_score(self.player.score)
#                         self.game_state = "game_over"

#                 for projectile in self.projectiles[:]:
#                     if asteroid.collides_with(projectile):
#                         self.player.score += asteroid.points
#                         self.asteroids.remove(asteroid)
#                         self.projectiles.remove(projectile)

#                 for new_asteroid in asteroid.split():

```

```

#             new_asteroid.speed =
new_asteroid.base_speed * self.speed_multiplier
#             self.asteroids.append(new_asteroid)
#             break

#         self.spawn_timer += 1
#         if (self.spawn_timer >= self.spawn_interval and
#             len(self.asteroids) < self.max_asteroids - 1):
#             self.spawn_wave(random.randint(1, min(3, 1 +
self.wave_count // 5)))
#             self.spawn_timer = 0

#     def draw(self):
#         self.screen.fill((0, 0, 0))

#         if self.game_state == "menu":
#             self.menu.draw(self.screen)
#         elif self.game_state == "high_scores":
#             self.ui.draw_high_scores(self.screen,
self.high_scores)
#         else:
#             for obj in self.asteroids + self.projectiles:
#                 obj.draw(self.screen)
#             self.player.draw(self.screen)
#             self.ui.draw(self.screen, self.player,
self.game_state, self.difficulty_level)

#         pygame.display.flip()

#     def run(self):
#         fps = self.game_config['game']['fps']
#         while self.running:
#             self.handle_events()
#             self.update()
#             self.draw()
#             self.clock.tick(fps)

```

### Конфигурационные файлы

Файлы game\_config.json и graphics.json параметры игры :

Фрагмент кода : game\_config.json:

```

{
    "game": {
        "title": "Asteroids Defense",
        "initial_lives": 3,
        "screen_width": 800,
        "screen_height": 600,
        "fps": 60
    }
}

```

Фрагмент кода : graphics.json:

```
{
  "sizes": {
    "player_width": 50,
    "player_height": 30,
    "projectile_width": 8,
    "projectile_height": 16,
    "asteroid_sizes": {
      "3": 50,
      "2": 35,
      "1": 20
    }
  }
}
```

### Структура организации ресурсов

- asteroid\_project/
  - |— src/
    - |— main.py
    - |— config/
      - |— game\_config.json
      - |— graphics.json
    - |— engine/
      - |— game\_engine.py
    - |— objects/
      - |— asteroid.py
      - |— game\_object.py
      - |— moving\_object.py
      - |— player.py
      - |— projectile.py
    - |— ui/
      - |— menu.py
      - |— hud.py
  - |— docs/
    - |— пак\_3
  - |— high\_scores.json
- |— requirements.txt