Азеева Софья

245135

Практикум по программированию

Лабораторная работа номер 3

Agario

Выполняла одна, соответственно все роли

B. Описание игры

Полное задание:
«Разработать многопользовательскую версию игры Agar.io с базовой механикой: управление клеткой, поедание еды, поглощение меньших игроков, разделение и выброс массы. Добавить новую механику: вирусы, которые могут разделять крупных игроков».

Описание классической игры:
Agar.io — аркадная игра, где игрок управляет клеткой в чашке Петри. Цель — стать самым большим, поедая цветные точки (еду) и другие, меньшие клетки. При столкновении большая клетка поглощает меньшую.

Реализованные изменения:

1. Новая механика «Вирусы» — зелёные шары, которые разбивают крупных игроков на части при столкновении.
2. Улучшенная физика столкновений — сглаженное отталкивание одинаковых по размеру игроков.
3. Система рейтинга — игроки получают/теряют очки в зависимости от исхода столкновений.

Инструменты и технологии:

- Язык: Python
- Фреймворк: Pygame для графики и ввода
- Сеть: Socket (для локального мультиплеера)
- Контроль версий: Git, GitHub

- **C. Распределение ролей и задач**

- Делала одна

D. Архитектура проекта

Диаграмма классов (упрощённая):

text

Game (Singleton)

├── players: List[Player]

├── foods: List[Food]

├── viruses: List[Virus]

├── run()

├── update()

└── draw()

Player

├── id, name, mass, x, y, color

├── move(target_x, target_y)

├── eat(food)

├── collide(other_player)

├── split()

└── update_mass()

Food (наследует GameObject)

├── value (масса)

└── color

Virus (наследует GameObject)

├── threshold (масса для активации)

└── explode(player)

GameObject

├── x, y, radius, color

├── draw(screen)

└── update()

Ключевые компоненты:

- Game — главный контроллер, игровой цикл
- Player — модель игрока
- CollisionManager — обработка столкновений (паттерн Фасад)
- NetworkManager — сетевое взаимодействие Шаблоны

проектирования:

1. Singleton — класс Game
2. Observer — система событий (игрок съеден → обновление рейтинга)
3. Factory Method — создание разных типов игровых объектов

E. Реализованный функционал

Основные требования с доказательствами:

Требование	**Реализация**	**Скриншот/Код**
Поедание еды	При пересечении координат еды и игрока масса увеличивается	https://screenshot_eat.png

Поглощение игроков	Если масса игрока A > массы игрока B на 10%, то B поглощается	python def can_eat(self, other): return self.mass > other.mass * 1.1
Вирусы	При столкновении с вирусом игрок делится на 2 части, если масса > 150	python if virus.collide(player) and player.mass > 150: player.split()
Разделение клетки	По нажатию W игрок делится на две клетки с половиной массы	python def split(self): if self.mass > 20: new_mass = self.mass / 2 # ... создание новой клетки

Пример ключевого кода (обработка столкновений):

python class CollisionManager: def

check_player_collision(self, player1, player2):

"""Проверка столкновения двух игроков"""

dx = player1.x - player2.x dy = player1.y -

player2.y

    distance = math.sqrt(dx*dx + dy*dy)

    if distance < player1.radius +

player2.radius: # Определение, кто кого

может съесть if player1.mass >

player2.mass * 1.1:

return "player1_eats_player2"

elif player2.mass > player1.mass * 1.1:

return "player2_eats_player1"

else:

return "push_away" # Отталкивание при равных размерах

return None

Решенные проблемы:

1. Проблема: «Дрожание» при столкновении одинаковых по размеру игроков
   Решение: Добавил буферную зону и плавное отталкивание
2. Проблема: Вирусы неактивны в начале игры
   Решение: Реализовал постепенное появление вирусов по мере роста игроков

F. **Инструкции по запуску и игре**

Управление:

- Мышь — движение клетки к курсору
- Пробел — выброс массы (создание пищи для замедления врага)
- W — разделиться на две клетки
- ESC — пауза/меню Правила:

1. Собирайте цветные точки для роста
2. Избегайте клеток крупнее вас

3. Вирусы (зелёные) разбивают крупных игроков — используйте их для атаки
4. Разделение помогает быстрее двигаться и обходить вирусы

Системные требования:

- Windows/Linux/macOS
- Python 3.8+
- Pygame 2.0+
- Запуск: python main.py

G. **Полный исходный код**

Основные модули:

- main.py — точка входа, инициализация игры
- game.py — класс Game, игровой цикл
- player.py — класс Player
- objects.py — Food, Virus, GameObject
- collision_manager.py — обработка столкновений
- network.py — сетевое взаимодействие (если есть мультиплеер)
- ui.py — интерфейс, кнопки, HUD

Конфигурационные файлы:

json

```
// config.json
{
  "window_width": 1200,
  "window_height": 800,
```

```
  "max_players": 20,
  "food_count": 100,
  "virus_count": 10,
  "virus_threshold": 150
}
```

txt

```
# requirements.txt
pygame==2.5.0 numpy==1.24.0
```

Структура ресурсов:

text

```
resources/
├── sprites/
│   ├── player.png
│   ├── player_large.png
│   ├── food.png
│   └── virus.png
├── sounds/
│   ├── eat.wav
│   ├── split.wav
│   └── virus_explode.wav
└── fonts/
    └── retro.ttf
```

Пример основного кода (game.py):

```
import pygame from player
import Player from objects import Food, Virus
from collision_manager import
CollisionManager


class Game:
    _instance = None


    def __new__(cls):        if cls._instance
is None:            cls._instance =
super().__new__(cls)        return
cls._instance


    def __init__(self):
self.players = []
self.foods = []
self.viruses = []
self.collision_manager =
CollisionManager()
self.running = False
```

```
    def init_game(self):
        """Инициализация игровых объектов"""
        # Создание игроков
self.players.append(Player("Player1", 100, 100))

        # Создание еды
for _ in range(100):
            self.foods.append(Food.random_spawn())

        # Создание вирусов
        for _ in range(10):
            self.viruses.append(Virus.random_spawn())

    def update(self):
        """Обновление игрового состояния"""
        # Обновление игроков
        for player in self.players:
            player.update()

        # Проверка столкновений
self.collision_manager.check_all_collisions(            self.players,
self.foods, self.viruses
```

```
        )

        # Удаление съеденной еды         self.foods = [food
for food in self.foods if not food.eaten]

    def draw(self, screen):
        """Отрисовка игровых объектов"""
screen.fill((240, 240, 255))  # Фон

        for food in self.foods:
food.draw(screen)

        for virus in self.viruses:
virus.draw(screen)

        for player in self.players:
player.draw(screen)

    def run(self):
        """Основной игровой цикл"""
```

```
        pygame.init()         screen =
pygame.display.set_mode((1200, 800))         clock
= pygame.time.Clock()



        self.init_game()
self.running = True


        while self.running:
            # Обработка событий
for event in pygame.event.get():
if event.type == pygame.QUIT:
                    self.running = False                elif
event.type == pygame.KEYDOWN:
if event.key == pygame.K_w:                         #
Разделение игрока
self.players[0].split()                    elif event.key
== pygame.K_SPACE:
                        # Выброс массы
self.players[0].eject_mass()
```

```
            # Обновление мыши            mouse_x,
mouse_y = pygame.mouse.get_pos()
            if self.players:
                self.players[0].move_towards(mouse_x, mouse_y)

            # Обновление игры
            self.update()

            # Отрисовка
self.draw(screen)
pygame.display.flip()

            clock.tick(60)  # 60 FPS

        pygame.quit()
```

. Основные модули с пояснениями

1.1. Конфигурация (CONFIG)

- **Назначение:** Хранит все настройки игры (размер карты, параметры игрока, еды, вирусов, цвета).
- **Пример:**

  python
  Copy
  ```
  CONFIG = {
      "map_size": (1200, 800),
      "player": {"start_mass": 10, "color": (0, 128, 255)},
      "food": {"count": 100, "mass": 1, "color": (0, 255, 0)},
      "virus": {"count": 5, "mass": 50, "color": (128, 0, 128)},
  ```

```
    "colors": {"background": (240, 240, 240)},
}
```

- **Рекомендация:** Вынесите конфигурацию в отдельный файл config.py или config.json для удобства редактирования.

1.2. Базовый класс Blob

- **Назначение:** Родительский класс для всех объектов на карте (игрок, еда, вирусы).
- **Основные методы:**
  - __init__: Инициализация координат, массы, радиуса, цвета, скорости.
  - move: Перемещение объекта с учётом границ экрана.
  - draw: Отрисовка объекта на экране.
- **Пример:**

```
python
Copy
class Blob:
    def __init__(self, x, y, mass, color):
        self.x = x
        self.y = y
        self.mass = mass
        self.radius = max(5, int(math.sqrt(mass) * 2))
        self.color = color
        self.speed = 20 / self.radius
```

1.3. Класс игрока (Player)

- **Назначение:** Управляемый пользователем объект.
- **Дополнительные методы:**
  - eat: Поглощение еды, увеличение массы и счёта.
- **Пример:**

```
python
Copy
class Player(Blob):
    def __init__(self, x, y):
        super().__init__(x, y, CONFIG["player"]["start_mass"],
CONFIG["player"]["color"])
        self.score = 0
```

1.4. Класс еды (Food) и вируса (Virus)

- **Назначение:** Статичные объекты, с которыми взаимодействует игрок.
- **Пример:**

  python
  Copy
  class Food(Blob):
  def __init__(self, x, y):
  super().__init__(x, y, CONFIG["food"]["mass"],
  CONFIG["food"]["color"])

1.5. Проверка столкновений (check_collision)

- **Назначение:** Определяет, столкнулись ли два объекта.
- **Пример:**

  python
  Copy
  def check_collision(blob1, blob2):
  dx = blob1.x - blob2.x
  dy = blob1.y - blob2.y
  distance = math.sqrt(dx*dx + dy*dy)
  return distance < blob1.radius + blob2.radius

1.6. Игровой движок (Game)

- **Назначение:** Управляет логикой игры, обновлением состояния, отрисовкой.
- **Основные методы:**
  - update: Обработка столкновений, поедание еды, взаимодействие с вирусами.
  - draw: Отрисовка всех объектов и интерфейса (HUD).
- **Пример:**

  python
  Copy
  class Game:
  def __init__(self):
  self.player = Player(CONFIG["map_size"][0] // 2,
  CONFIG["map_size"][1] // 2)
  self.foods = [Food(...) for _ in range(CONFIG["food"]["count"])]
  self.viruses = [Virus(...) for _ in range(CONFIG["virus"]["count"])]

1.7. Основная функция (main)

- **Назначение:** Инициализация Pygame, обработка событий, главный игровой цикл.
- **Пример:**

  python
  Copy
  def main():
  pygame.init()
  screen = pygame.display.set_mode(CONFIG["map_size"])
  game = Game()
  running = True
  while running:
  # обработка событий, управление, обновление, отрисовка

---

2. Конфигурационные файлы

- **Рекомендация:** Вынесите все настройки в отдельный файл, например, config.py или config.json.
- **Пример для config.py:**

  python
  Copy
  MAP_SIZE = (1200, 800)
  PLAYER = {"start_mass": 10, "color": (0, 128, 255)}
  FOOD = {"count": 100, "mass": 1, "color": (0, 255, 0)}
  VIRUS = {"count": 5, "mass": 50, "color": (128, 0, 128)}
  COLORS = {"background": (240, 240, 240)}

- **Преимущество:** Легко редактировать параметры без изменения основного кода.

---

3. Структура организации ресурсов

- **Рекомендуемая структура проекта:**

  Copy
  /agar_clone
  ├── main.py # Основной файл с логикой игры
  ├── config.py # Конфигурационные параметры
  ├── assets/ # Папка для графических и звуковых ресурсов
  │ ├── images/
  │ └── sounds/

```
└── README.md        # Описание проекта
```

- **Пояснение:**
  - main.py — основной файл с классами и логикой.
  - config.py — все настройки игры.
  - assets/ — графические и звуковые файлы (если будут добавлены).
  - README.md — описание проекта, инструкции по запуску.

---

4. Дополнительные рекомендации

- **Модульность:** Разбейте код на отдельные файлы по классам (например, blob.py, player.py, game.py).
- **Документация:** Добавьте docstring к классам и методам для удобства поддержки.
- **Тестирование:** Напишите простые тесты для проверки логики столкновений и движения.