Лабораторная работа 3

Разработка интерактивной игры Agar.io

Томов М.С., Антонова А.В., студенты группы ИД24-1, Финансовый Университет при Правительстве РФ

Название задачи: Agar.io с использованием Pygame и ООП

Дата выполнения: 08 декабря 2025 года

Цель работы

Освоить объектно-ориентированное программирование, работу с графической библиотекой Pygame, реализацию системы управления состояниями игры, динамическую камеру с масштабированием и конфигурируемую архитектуру через JSON.

Описание задания

Разработать игру в жанре «симулятор поедания клеток» на основе концепции Agar.io с использованием Pygame. Игра включает управление ячейкой игрока, систему сложности (easy, medium, hard), динамическую камеру, систему очков и жизней, визуализацию игрового мира с сеткой координат, и условие проигрыша при уменьшении размера игрока.

Математическая модель

Основные параметры

Параметр	Значение	Описание
WORLD_SIZE	2000	Размер игрового мира
SCREEN_WIDTH, HEIGHT	800, 500	Размеры окна
Начальная масса игрока	20	Масса в условных единицах
Масса ячеки	5	Значение для съедания

Функция масштабирования камеры

$$zoom = \frac{100}{mass} + 0.3$$

Чем больше масса игрока, тем больше отдаляется камера (меньше zoom). Это создает динамический визуальный эффект приближения/отдаления.

Функция расстояния между объектами

$$distance = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

Применяется для определения поедания ячеек: если расстояние ≤ радиусу игрока, ячейка поедается.

Граничные условия мира

$$x = \max(radius, \min(WORLD\_SIZE - radius, x))$$

Предотвращает выход игрока за границы мира с учетом его радиуса.

Нормализация вектора движения

$$\text{norm\_vector} = \frac{(v_x, v_y)}{\sqrt{v_x^2 + v_y^2}}$$

Обеспечивает одинаковую скорость движения во всех направлениях.

Структура ООП

Иерархия классов

Базовый класс `Drawable` — определяет интерфейс для рисуемых объектов

- Хранит surface и camera
- Переопределяется в подклассах

Класс `Camera` — управление камерой и масштабированием

- Отслеживание позиции (x, y)
- Динамическое масштабирование (zoom)
- Метод follow(target): следование за игроком

Класс `Grid` — визуализация сетки координат

- Горизонтальные и вертикальные линии
- Учет масштабирования и смещения камеры

Класс `Cell` — отдельная ячейка в мире

- Позиция (x, y) и масса
- Случайный выбор цвета (11 вариантов)

- Отрисовка с трансформацией камеры

Класс `CellList` — управление коллекцией ячеек

- Создание и хранение n ячеек
- Циклическая отрисовка всех ячеек

Класс `Player` — контролируемая ячейка игрока

- Управление позицией, массой, скоростью
- Обработка ввода (WASD/стрелки, мышь)
- Обнаружение коллизий (поедание)
- Вычисляемое свойство radius = mass / 2

Выполненные требования

✅ **ООП-архитектура:** 6 классов с наследованием от Drawable

✅ **Управление игроком:** WASD/стрелки + мышь с нормализацией вектора

✅ **Конфигурация:** game_config.json управляет параметрами

✅ **Динамическая камера:** Масштабирование по формуле zoom = 100/mass + 0.3

✅ **Три уровня сложности:** easy (800 ячеек), medium (500 ячеек), hard (300 ячеек, скорость 5)

✅ **Визуализация:** Сетка координат, цветные ячейки, HUD

✅ **Управление состояниями:** menu → play → game_over

✨ **Дополнительно:** Цветовое разнообразие (11 цветов для ячеек, 7 для игрока), двойное управление, плавная анимация

Ключевые функции

Функция distance(a, b)

```
def distance(a, b):
dx = a[0] - b[0]
dy = a[1] - b[1]
return math.hypot(dx, dy)
```

Расчет евклидова расстояния между двумя точками.

Метод Camera.follow(target)

```
def follow(self, target):
self.zoom = 100 / target.mass + 0.3
self.x = SCREEN_WIDTH / 2 - target.x * self.zoom
self.y = SCREEN_HEIGHT / 2 - target.y * self.zoom
```

Обновление позиции и масштаба камеры для следования за игроком.

Метод Player.move()

Обработка WASD/стрелок с нормализацией вектора или движение к курсору мыши. Применяются граничные условия для удержания в пределах мира.

Метод Player.eat(cells)

Проверка расстояния до всех ячеек. Если distance ≤ radius, ячейка удаляется, масса увеличивается на 0.5, счетчик съеденных ячеек увеличивается.

Функция create_game(difficulty)

Инициализирует игру с параметрами, зависящими от уровня сложности:

- easy: 800 ячеек, скорость 4
- medium: 500 ячеек, скорость 4
- hard: 300 ячеек, скорость 5

Управление состояниями

Три состояния игры

menu — выбор сложности (LEFT/RIGHT стрелки, ENTER для запуска)

play — активная игра с движением, едой, подсчетом очков

game_over — экран окончания с финальным счетом и перезагрузкой (ENTER)

События и их обработка

Событие	Действие
QUIT	Завершение программы
KEYDOWN (WASD/стрелки)	Движение игрока
MOUSEMOTION	Направление к курсору
KEYDOWN (LEFT/RIGHT)	Выбор сложности в меню
KEYDOWN (RETURN)	Запуск игры или перезагрузка
KEYDOWN (ESCAPE)	Выход из программы

Инициализация и конфигурация

game_config.json

{
"game": {
"title": "Agar.io by Tomov & Antonova",
"version": "1.0",
"max_players": 1,
"initial_lives": 3,
"difficulty_levels": ["easy", "medium", "hard"]
}
}

Использование:

- GAME_TITLE загружается при инициализации
- INITIAL_LIVES = 3 (начальное количество жизней)
- DIFFICULTIES используется в меню и для выбора параметров

Инициализация Pygame

import pygame, random, math, json

pygame.init()
screen = pygame.display.set_mode((SCREEN_WIDTH, SCREEN_HEIGHT))
pygame.display.set_caption(GAME_TITLE)
clock = pygame.time.Clock()
font = pygame.font.SysFont("Ubuntu", 20, True)

Основной игровой цикл

while running:
clock.tick(60) # 60 FPS

```
# Обработка событий
for e in pygame.event.get():
    if e.type == pygame.QUIT:
        running = False
    # ... обработка меню, play, game_over ...

# Обновление логики
if state == "play":
    player.move()
    eaten = player.eat(cells.cells)
    score += eaten * 10
    camera.follow(player)
    if player.mass < 10:
```

```
            state = "game_over"

    # Отрисовка
    screen.fill((242, 251, 255))
    if state == "menu":
        # ... отрисовка меню ...
    elif state == "play":
        grid.draw()
        cells.draw()
        player.draw()
        # ... отрисовка HUD ...
    elif state == "game_over":
        # ... отрисовка экрана окончания ...

    pygame.display.flip()
```

pygame.quit()

Особенности реализации

Физическая корректность

Система использует упрощенную модель съедания на основе расстояния. Условие поедания: расстояние между центрами ≤ радиусу игрока. Это качественно отражает поведение в игре Agar.io.

Оптимизация производительности

- Частота кадров ограничена 60 FPS через clock.tick(60)
- Эффективная обработка списка ячеек (удаление при поедании)
- Использование встроенных функций pygame для отрисовки

Интерактивность

- Мгновенная обработка выбора сложности в меню
- Плавное движение с нормализацией вектора
- Двойное управление: клавиши + мышь
- Динамическое масштабирование камеры в реальном времени

Выводы

В ходе выполнения лабораторной работы освоены:

- Объектно-ориентированное программирование с наследованием и полиморфизмом
- Работа с Pygame: инициализация, отрисовка примитивов, обработка событий
- Создание плавной анимации с постоянной частотой кадров
- Реализация математических моделей (расстояния, масштабирование, нормализация)
- Управление состояниями игры (state machine)
- Динамическая система камеры с масштабированием
- Конфигурируемая архитектура через JSON
- Интерактивные элементы интерфейса (меню, HUD)

Программа успешно демонстрирует игровую механику Agar.io с возможностью управления сложностью и интерактивного взаимодействия.

Приложение: Полный исходный код

agar_io.py

import pygame, random, math, json

Загрузка конфигурации

with open("game_config.json", "r", encoding="utf-8") as f:
config = json.load(f)

GAME_TITLE = config["game"]["title"]
INITIAL_LIVES = config["game"]["initial_lives"]
DIFFICULTIES = config["game"]["difficulty_levels"]

SCREEN_WIDTH, SCREEN_HEIGHT = 800, 500
WORLD_SIZE = 2000

pygame.init()
screen = pygame.display.set_mode((SCREEN_WIDTH, SCREEN_HEIGHT))
pygame.display.set_caption(GAME_TITLE)
clock = pygame.time.Clock()
font = pygame.font.SysFont("Ubuntu", 20, True)

def distance(a, b):
return math.hypot(a[0] - b[0], a[1] - b[1])

class Camera:
def **init**(self):
self.x = self.y = self.zoom = 1.0

```
    def follow(self, target):
        self.zoom = 100 / target.mass + 0.3
        self.x = SCREEN_WIDTH / 2 - target.x * self.zoom
        self.y = SCREEN_HEIGHT / 2 - target.y * self.zoom
```

```
class Drawable:
def __init__(self, surface, camera):
self.surface, self.camera = surface, camera
def draw(self): pass

class Grid(Drawable):
def __init__(self, surface, camera):
super().__init__(surface, camera)
self.color = (220, 220, 220)
```

```
    def draw(self):
        z, camx, camy = self.camera.zoom, self.camera.x, self.camera.y
        for i in range(0, WORLD_SIZE + 1, 50):
            pygame.draw.line(self.surface, self.color,
                             (camx, i * z + camy), (WORLD_SIZE * z + camx, i * z + camy), 1)
            pygame.draw.line(self.surface, self.color,
                             (i * z + camx, camy), (i * z + camx, WORLD_SIZE * z + camy), 1)
```

```
class Cell(Drawable):
COLORS = [(80, 252, 54), (36, 244, 255), (243, 31, 46), (4, 39, 243),
(254, 6, 178), (255, 211, 7), (216, 6, 254), (145, 255, 7),
(7, 255, 182), (255, 6, 86), (147, 7, 255)]
```

```
    def __init__(self, surface, camera):
        super().__init__(surface, camera)
        self.x = random.randint(20, WORLD_SIZE - 20)
        self.y = random.randint(20, WORLD_SIZE - 20)
        self.mass = 5
        self.color = random.choice(Cell.COLORS)

    def draw(self):
        z, camx, camy = self.camera.zoom, self.camera.x, self.camera.y
        cx, cy = int(self.x * z + camx), int(self.y * z + camy)
        pygame.draw.circle(self.surface, self.color, (cx, cy), int(self.mass * z))
```

class CellList(Drawable):
def **init**(self, surface, camera, n):
super().**init**(surface, camera)
self.cells = [Cell(surface, camera) for _ in range(n)]

```
def draw(self):
    for c in self.cells:
        c.draw()
```

class Player(Drawable):
COLORS = [(37, 7, 255), (35, 183, 253), (48, 254, 241), (19, 79, 251),
(255, 7, 230), (255, 7, 23), (6, 254, 13)]

```
def __init__(self, surface, camera, name="Player", speed=4):
    super().__init__(surface, camera)
    self.x = random.randint(100, 400)
    self.y = random.randint(100, 400)
    self.mass, self.speed, self.name = 20, speed, name
    self.color = random.choice(Player.COLORS)

@property
def radius(self):
    return self.mass / 2

def move(self):
    keys = pygame.key.get_pressed()
    vx, vy = 0, 0
    if keys[pygame.K_w] or keys[pygame.K_UP]: vy -= 1
    if keys[pygame.K_s] or keys[pygame.K_DOWN]: vy += 1
    if keys[pygame.K_a] or keys[pygame.K_LEFT]: vx -= 1
    if keys[pygame.K_d] or keys[pygame.K_RIGHT]: vx += 1

    if vx or vy:
        dist = math.hypot(vx, vy)
        self.x += (vx / dist) * self.speed
        self.y += (vy / dist) * self.speed
    else:
        mx, my = pygame.mouse.get_pos()
        dx, dy = mx - SCREEN_WIDTH / 2, my - SCREEN_HEIGHT / 2
```

```
            dist = math.hypot(dx, dy)
            if dist:
                self.x += (dx / dist) * self.speed
                self.y += (dy / dist) * self.speed

        self.x = max(self.radius, min(WORLD_SIZE - self.radius, self.x))
        self.y = max(self.radius, min(WORLD_SIZE - self.radius, self.y))

    def eat(self, cells):
        eaten = 0
        for c in cells[:]:
            if distance((self.x, self.y), (c.x, c.y)) <= self.radius:
                self.mass += 0.5
                cells.remove(c)
                eaten += 1
        return eaten

    def draw(self):
        z, camx, camy = self.camera.zoom, self.camera.x, self.camera.y
        cx, cy = int(self.x * z + camx), int(self.y * z + camy)
        pygame.draw.circle(self.surface, (0, 0, 0), (cx, cy), int((self.radius + 3) * z))
        pygame.draw.circle(self.surface, self.color, (cx, cy), int(self.radius * z))
        text = font.render(self.name, True, (50, 50, 50))
        tw, th = text.get_size()
        self.surface.blit(text, (cx - tw // 2, cy - th // 2))
```

```
def create_game(difficulty):
camera = Camera()
cells_count = {"easy": 800, "medium": 500, "hard": 300}[difficulty]
speed = 5 if difficulty == "hard" else 4
return camera, Grid(screen, camera), CellList(screen, camera, cells_count), Player(screen,
camera, "You", speed)

state, running, score, lives, difficulty_index = "menu", True, 0, INITIAL_LIVES, 0
camera, grid, cells, player = create_game(DIFFICULTIES[difficulty_index])

while running:
clock.tick(60)
```

```
    for e in pygame.event.get():
        if e.type == pygame.QUIT:
```

```
            running = False
        if e.type == pygame.KEYDOWN:
            if e.key == pygame.K_ESCAPE:
                running = False
            if state == "menu":
                if e.key == pygame.K_LEFT:
                    difficulty_index = (difficulty_index - 1) % len(DIFFICULTIES)
                if e.key == pygame.K_RIGHT:
                    difficulty_index = (difficulty_index + 1) % len(DIFFICULTIES)
                if e.key == pygame.K_RETURN:
                    score, lives = 0, INITIAL_LIVES
                    camera, grid, cells, player = create_game(DIFFICULTIES[difficulty_inde
                    state = "play"
            if state == "game_over" and e.key == pygame.K_RETURN:
                score, lives = 0, INITIAL_LIVES
                camera, grid, cells, player = create_game(DIFFICULTIES[difficulty_index]
                state = "play"

    if state == "play":
        player.move()
        eaten = player.eat(cells.cells)
        score += eaten * 10
        camera.follow(player)
        if player.mass < 10:
            state = "game_over"

    screen.fill((242, 251, 255))

    if state == "menu":
        title = font.render("Agar.io by Tomov & Antonova", True, (0, 0, 0))
        tw, th = title.get_size()
        screen.blit(title, (SCREEN_WIDTH // 2 - tw // 2, SCREEN_HEIGHT // 2 - 80))
        diff_text = font.render(f"Difficulty: {DIFFICULTIES[difficulty_index]}", True, (
        dw, dh = diff_text.get_size()
        screen.blit(diff_text, (SCREEN_WIDTH // 2 - dw // 2, SCREEN_HEIGHT // 2 - 20))
        hint = font.render("LEFT/RIGHT arrows, ENTER to start", True, (100, 100, 100)
        hw, hh = hint.get_size()
        screen.blit(hint, (SCREEN_WIDTH // 2 - hw // 2, SCREEN_HEIGHT // 2 + 20))
```

```
    elif state == "play":
        grid.draw()
        cells.draw()
        player.draw()
        screen.blit(font.render(f"Score: {score}", True, (0, 0, 0)), (10, 10))
        screen.blit(font.render(f"Lives: {lives}", True, (0, 0, 0)), (10, 35))
        screen.blit(font.render(DIFFICULTIES[difficulty_index], True, (0, 0, 0)), (10, 60
    elif state == "game_over":
        msg = font.render("GAME OVER", True, (200, 0, 0))
        mw, mh = msg.get_size()
        screen.blit(msg, (SCREEN_WIDTH // 2 - mw // 2, SCREEN_HEIGHT // 2 - 40))
        score_text = font.render(f"Final Score: {score}", True, (0, 0, 0))
        sw, sh = score_text.get_size()
        screen.blit(score_text, (SCREEN_WIDTH // 2 - sw // 2, SCREEN_HEIGHT // 2))
        restart = font.render("ENTER to restart", True, (100, 100, 100))
        rw, rh = restart.get_size()
        screen.blit(restart, (SCREEN_WIDTH // 2 - rw // 2, SCREEN_HEIGHT // 2 + 40))

    pygame.display.flip()
```

```
pygame.quit()
```

game_config.json

```
{
"game": {
"title": "Agar.io by Tomov & Antonova",
"version": "1.0",
"max_players": 1,
"initial_lives": 3,
"difficulty_levels": ["easy", "medium", "hard"]
}
}
```