Министерство образования Республики Беларусь
Учреждение образования
«Белорусский государственный университет
информатики и радиоэлектроники»

Факультет компьютерных систем и сетей
Кафедра вычислительных машин, систем и сетей
Дисциплина: Программирование на языках высокого уровня

# ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
на тему

# Игра “Червячки”

Студент Н.Д. Георгиев
Руководитель Е.В.Богдан

МИНСК 2023

Учреждение образования
«Белорусский государственный университет информатики
и радиоэлектроники»

Факультет компьютерных систем и сетей

УТВЕРЖДАЮ
Заведующий кафедрой
ЭВМ

(подпись)

2023 г.

ЗАДАНИЕ
по курсовому проектированию

Студенту *Георгиев Никита Димитрова*

1. Тема проекта *Игра «Червячки»*

2. Срок сдачи студентом законченного проекта *15 декабря 2023 г.*

3. Исходные данные к проекту: картинки png и звуки игры в папке sounds

4. Содержание расчетно-пояснительной записки (перечень вопросов, которые подлежат разработке)

1. Введение.
2. Задание.
3. Обзор литературы.
 3.1. Обзор методов и алгоритмов решения поставленной задачи.
4. Функциональное проектирование.
 4.1. Структура входных и выходных данных.
 4.2. Разработка диаграммы классов.
 4.3. Описание классов.
5. Разработка программных модулей.
 5.1. Разработка схем алгоритмов (два наиболее важных метода).
 5.2. Разработка алгоритмов (описание алгоритмов по шагам для двух методов).
6. Результаты работы.
7. Заключение
8. Литература
9. Приложения

5. Перечень графического материала (с точным обозначением обязательных чертежей и графиков)

*1. Диаграмма классов.*
*2. Схема алгоритма* `checkCollisionWithFood()`.
*3. Схема алгоритма* `Draw(int i)`.

6. Консультант по проекту (с обозначением разделов проекта) Е.В.Богдан

7. Дата выдачи задания 15 *сентября 2023 г*

8. Календарный график работы над проектом на весь период проектирования (с обозначением сроков выполнения и трудоемкости отдельных этапов):

*1. Выбор задания. Разработка содержания пояснительной записки. Перечень графического материала – 15 %;*

*разделы 2, 3 – 10 %;*

*разделы 4 к –20 %;*

*разделы 5 к – 35 %;*

*раздел 6,7,8 – 5 %;*

*раздел 9 к – 5%;*

*оформление пояснительной записки и графического материала к 15.12.23 – 10 %*

*Защита курсового проекта с 21.12 по 28.12.23г.*

РУКОВОДИТЕЛЬ Е.В.Богдан

_____

Задание принял к исполнению _____ Н.Д. Георгиев

(дата и подпись студента)

**Содержание**

## ВВЕДЕНИЕ

Современные возможности по разработке прикладного программного обеспечения с использованием языка высокого уровня C++ включают в себя следующие аспекты:

Объектно-ориентированное программирование (ООП), которое позволяет создавать программы, основанные на объектах, а не на процедурах. Это позволяет создавать более гибкие и масштабируемые приложения.

Использование стандартных библиотек C++[1], таких как STL (Standard Template Library), которые предоставляют широкий набор готовых компонентов для решения различных задач.

Использование современных инструментов разработки, таких как среды разработки (IDE), отладчики и компиляторы, которые позволяют ускорить процесс разработки и улучшить качество кода.

Применение современных методов разработки, таких как Agile и DevOps, которые позволяют создавать приложения быстрее и более эффективно.

Использование современных практик программирования, таких как TDD (Test-Driven Development) и CI/CD (Continuous Integration/Continuous Deployment), которые позволяют создавать более надежное и качественное программное обеспечение.

Червячки - это игра, в которой игрок управляет змеей, которая растет по мере того, как она съедает “еду”. Цель игры - стать самой большой змеей на поле и уничтожить других игроков, сталкиваясь с ними головой своей змеи. Игрок может управлять змеей, используя клавиатуру.

# 1. ЗАДАНИЕ НА КУРСОВУЮ РАБОТУ

Овладеть практическими навыками проектирования и разработки законченного, отлаженного и протестированного программного продукта с использованием языка высокого уровня C++ , овладеть практическими навыками проектирования и разработки законченного, отлаженного и протестированного программного продукта с использованием языка высокого уровня C++. Разработать игру “Червячки” при помощи библиотеки Raylib.

Понимание Основных Принципов ООП:

Цель: Освоить базовые концепции ООП, такие как инкапсуляция, наследование и полиморфизм. Почему это важно: Это обеспечит более глубокое понимание организации кода и его взаимодействия.

Навыки Проектирования Классов и Объектов:

Цель: уметь создавать классы и объекты, определять их атрибуты и методы. Почему это важно: это является основой ООП и позволяет структурировать код для более легкого понимания и поддержки.

Применение Инкапсуляции:

Цель: использовать инкапсуляцию для скрытия внутренних деталей реализации классов и предоставления публичного интерфейса. Почему это важно: это способствует безопасности кода и облегчает его сопровождение.

Мастерство В Обработке Наследования: Цель: понимать, как использовать наследование для создания иерархии классов и расширения функциональности. Почему это важно: это позволяет эффективно использовать и пере использовать код.

Овладение Исключениями и Обработкой Ошибок: Цель: Знание, как обрабатывать исключения и ошибки в объектно-ориентированных программах. Почему это важно: это повышает устойчивость программы к ошибкам и улучшает ее отказоустойчивость.

Эти цели могут служить отправной точкой для разработки программистом плана обучения и практического применения концепций ООП в реальных проектах.

Преимущество Raylib:

Разработка GUI: Raylib предоставляет широкий спектр компонентов для создания графических пользовательских интерфейсов, включая Rlgl модуль, который позволяет быстро и легко создавать интерфейс с использованием спецификации OpenGL, которая определяет платформонезависимый программный интерфейс для написания приложений, использующих двумерную и трехмерную компьютерную графику.

Простота использования: Raylib[12] имеет простой и интуитивно понятный интерфейс программирования приложений (API), который упрощает разработку игр и графических приложений. Она предоставляет прямой доступ к низкоуровневым графическим и аудиофункциям, что позволяет разработчикам сосредоточиться на создании контента и игровой логики, а не на сложностях низкоуровневого программирования.

Переносимость: Raylib поддерживает несколько платформ, включая Windows, macOS, Linux, Android и iOS. Это позволяет разработчикам создавать приложения, которые могут работать на разных устройствах и операционных системах без необходимости значительных изменений в коде.

Мощные возможности графики: Raylib предлагает широкий спектр функций для работы с 2D и 3D графикой, включая отрисовку примитивов, текстур, спрайтов, анимаций, эффектов частиц и многого другого. Она также поддерживает шейдеры, что позволяет создавать сложные визуальные эффекты и постобработку изображений.

Аудио поддержка: Библиотека Raylib предоставляет простой интерфейс для воспроизведения звуков и музыки. Вы можете загружать аудиофайлы различных форматов и управлять воспроизведением, регулировать громкость и применять эффекты.

В целом, использование библиотеки Raylib в C++ дает много преимуществ для разработчиков, которые хотят создавать игры с графическим пользовательским интерфейсом. Это мощный и гибкий инструмент, который можно использовать в широком спектре приложений, от простых аркадных игр до полноценных 3D игровых проектов.

## 2. ОБЗОР ЛИТЕРАТУРЫ

Перед началом разработки подобия игры "Slither.io" было необходимо ознакомиться с ее правилами.

"Slither.io" - это онлайн-игра, в которой игрок управляет змеей и соревнуется с другими игроками. Цель игры - стать самой длинной змеей, съедая разноцветные точки и других игроков. Змея растет по мере того, как она съедает точки и других игроков.

В начале игры каждый игрок управляет небольшой змеей. Игрок может управлять змеей, используя клавиатуру или мышь. Змея движется в направлении указателя мыши или в заданном направлении, если используется клавиатура.

В игре есть несколько правил и особенностей:

Съедение точек: Змея может съесть разноцветные точки, которые появляются на игровом поле. При этом змея становится длиннее.

Съедение других игроков: Змея может атаковать и съесть других игроков, если они находятся вне защитного круга своей змеи. При этом змея получает очки и становится еще длиннее.

Избегание столкновений: Змея должна избегать столкновений с другими змеями и собственным телом. Если змея столкнулась, она погибает, и игрок начинает заново с маленькой змеей.

Стратегия и тактика: Игрок может использовать различные стратегии и тактики для выживания и съедания других игроков. Некоторые игроки предпочитают активный стиль игры, атакуя других игроков, в то время как другие предпочитают пассивный стиль, избегая столкновений и сосредоточиваясь на росте своей змеи.

Целью данного курсового проекта является разработка игры на подобии браузерной игры “Slitherio” на языке программирования C++. В рамках проекта осуществляется изучение и реализация построения алгоритмов, необходимых для эффективного создания игры. Это обеспечивает более глубокое понимание принципов построения игр.

При разработке игры с использованием библиотеки Raylib, разработчики могут обратиться к официальной документации Raylib, которая предоставляет подробную информацию о различных аспектах использования библиотеки и создания игровых приложений.

Официальная документация Raylib: официальный сайт Raylib содержит обширную документацию, включая руководства, API—справочники, примеры кода, видео—гайды и другие материалы. В документации Raylib присутствуют следующие разделы:

Установка и настройка: Раздел "Getting Started" в документации предоставляет инструкции по установке и настройке Raylib на различные платформы. Здесь разработчики могут найти информацию о необходимых зависимостях, компиляции и запуске проекта.

Основные концепции: В этом разделе описываются основные концепции и принципы работы с Raylib. Включены объяснения о графическом

рендеринге, управлении окнами, обработке событий ввода, загрузке ресурсов и других ключевых аспектах разработки игр.
API-справочник: Документация содержит подробное описание API-функций и структур Raylib. Разработчики могут найти информацию о доступных функциях для работы с графикой, звуком, вводом, физикой и другими аспектами игрового движка.

Примеры кода: Официальная документация Raylib также предлагает набор примеров кода, демонстрирующих использование различных функций и возможностей библиотеки. Эти примеры могут служить вдохновением и помочь разработчикам лучше понять, как применять Raylib в своих проектах.

Форумы и сообщество: Документация также содержит ссылки на форумы и сообщества, где разработчики могут задавать вопросы, делиться опытом и получать поддержку со стороны других пользователей Raylib.

### 2.1 Рассмотрение методов и алгоритмов для решения задачи

Обход в ширину и обход в глубину - два алгоритма, являющиеся основой для большинства сложных алгоритмов, имеющих реальное применение в жизни. Не задумываясь, мы сталкиваемся с графами постоянно. Как самый банальный пример - навигатор. Построение маршрута - уже задача, в которой используются данные алгоритмы.

Обход в глубину (Depth-First Search, DFS)[13]:

Один из основных методов обхода графа, часто используемый для проверки связности графа, поиска цикла и компонент сильной связности и для топологической сортировки. Отбросим все эти сложные слова до поры до времени, а пока посмотрим, что же делает DFS и как он это делает. Будем рассматривать реализацию на списке смежности как самую легкую для понимания. Вкратце напомню, что список смежности - это один из способов представления графа, который выглядит примерно так:

На рисунке 2.1.1 изображен список смежности для данного графа

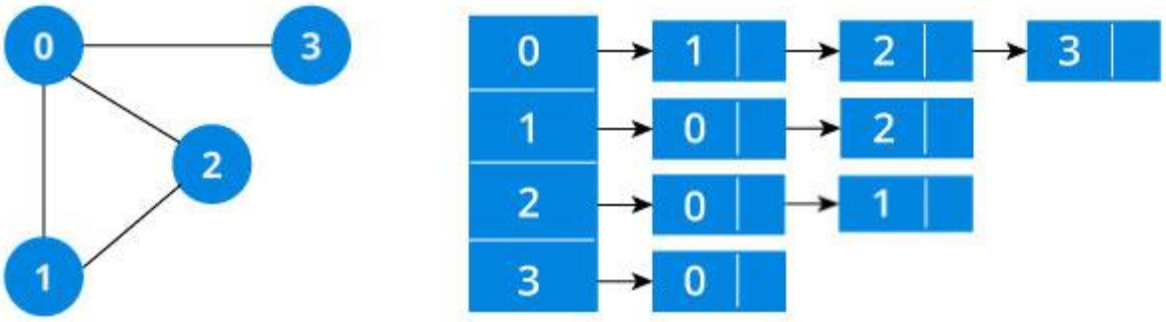


Рисунок 2.1.1 – список смежности графа

Для каждой вершины (первый столбик) мы составляем список смежных ей вершин, то есть список вершин с которыми у данной есть общие ребра(ребро инцидентное данным вершинам). Теперь собственно к самому алгоритму, принцип его работы совпадает с его названием, данный алгоритм идет "внутрь" графа, до того момента как ему становится некуда идти, затем возвращается в предыдущую вершину и снова идет от нее до тех пор, пока есть куда идти. И так далее.

Для понимания данного алгоритма нам потребуются 3 цвета, один будет обозначать, что вершину мы еще не посетили, второй что посетили и ушли, а третий, что посетили и не смогли идти дальше и начинаем возвращаться обратно. Стартуем из любой вершины, например, из первой. Идем по списку смежности. Из 1 вершины попадаем во вторую, переходим в ее список смежности, не забываем красить 1 вершину в серый, так как мы ее посетили и ушли дальше. Из второй вершины идем в любую из списка смежности второй вершины, например в 3. Красим 2 в серый и идем в список смежности 3-й вершины. А в нем ничего нет. В таком случае мы понимаем, что дальше нам идти не куда, пора возвращаться.

На рисунке 2.1.2 изображены первые 3 итерации графа

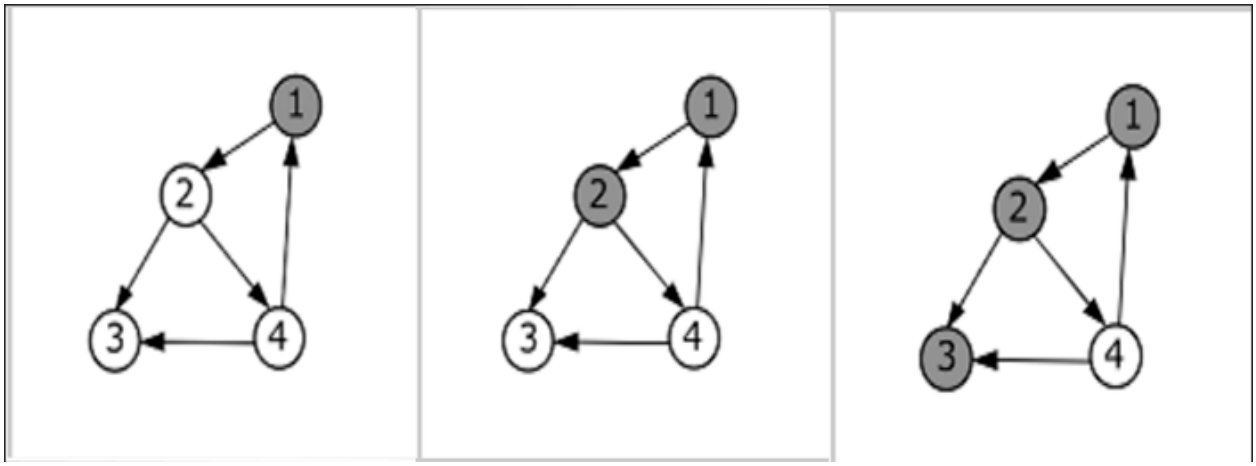


Рисунок 2.1.2 – первые 3 итерации

Красим 3 в черный так как нам идти некуда(нет белых вершин, в которые мы могли бы пойти из 3). Возвращаемся в 2 и в ее список смежности, видим что там еще есть вершина 4, выдвигаемся туда, оттуда можно пойти только в 1, но она уже серая, то есть мы там уже были. Алгоритм начнет рекурсивно откатываться назад перекрашивая все вершины в черный.

На рисунке 2.1.3 изображены 4, 5, 6 итерация графа

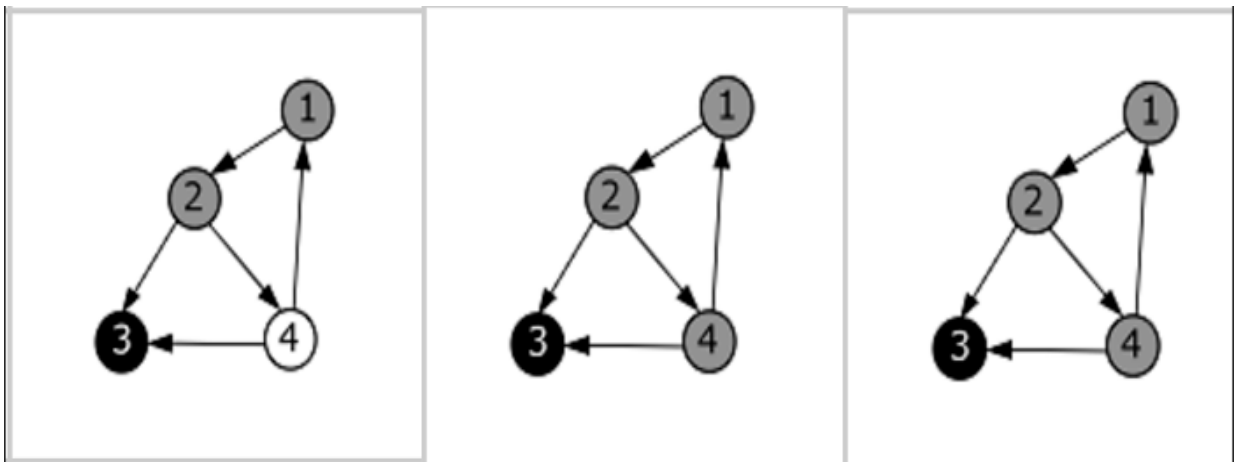


Рисунок 2.1.3 – 4, 5, 6 итерации

Обход в ширину (breadth-first search, BFS):

Систематически обходит все вершины графа. В чем его отличие от обхода в глубину? Обход в глубину "перепрыгивает" между строками списка смежности по 1 вершине за раз, а обход в ширину сразу по всем возможным, посмотрим как он работает на примере:

Первый шаг цикла обхода изображен на рисунке 2.1.4

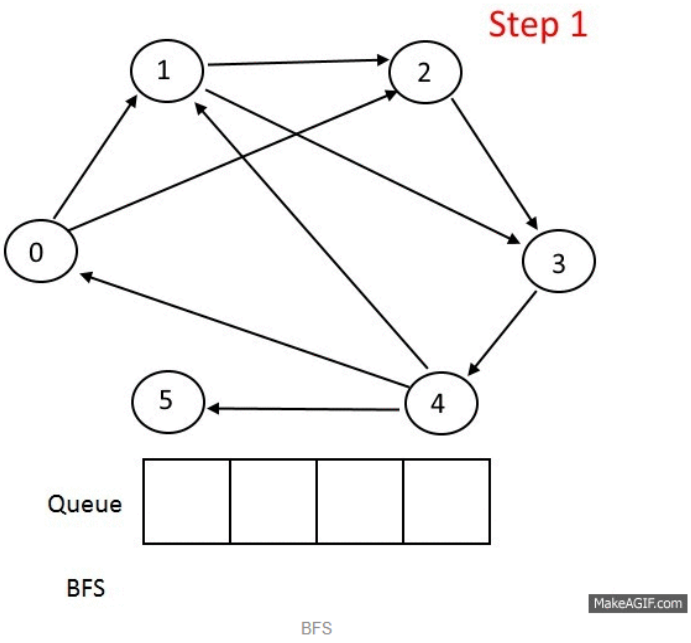


Рисунок 2.1.4 – первый шаг обхода

Второй шаг цикла обхода изображен на рисунке 2.1.5

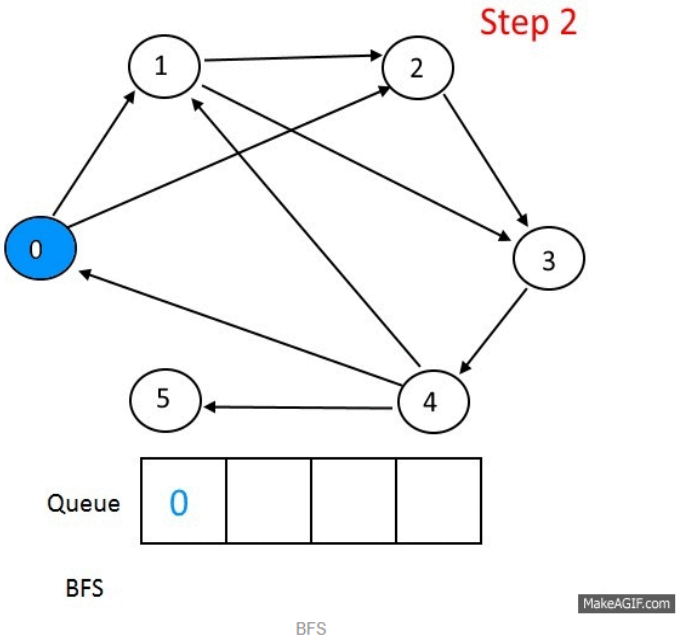


Рисунок 2.1.5 – второй шаг обхода

Третий шаг цикла обхода изображен на рисунке 2.1.6

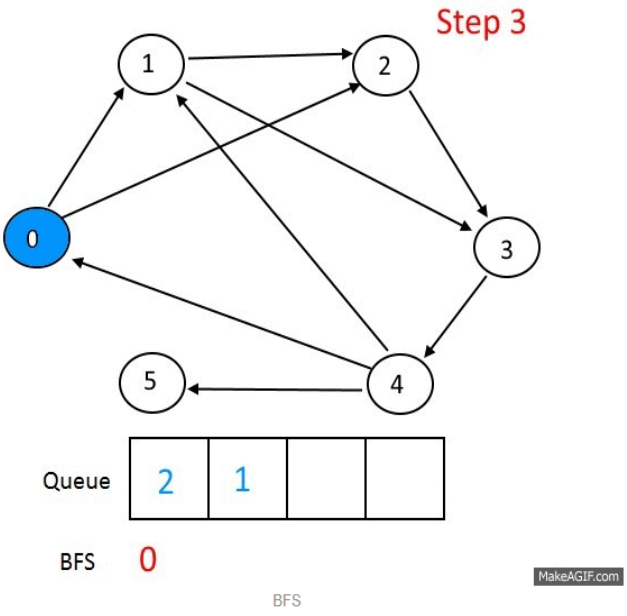


Рисунок 2.1.6

Сначала мы проходимся по всем вершинам смежным со стартовой, потом по всем, смежным со смежными стартовой и так далее. Вот еще один пример, который как мне кажется более наглядно показывает различие обходов в глубину и ширину: в 1-м граф как бы обходится равномерно.

## 2.2 Обзор методов и алгоритмов решения поставленной задачи

Существует множество аналогов игры "Slither.io".

"Agar.io":

Правила: В этой игре вы управляете клеткой, которая должна поедать другие клетки, чтобы стать больше. Цель игры - стать самым большим игроком на игровом поле и избегать попадания в большие клетки.

На рисунке 2.2.1 показан интерфейс игры “agar.io”

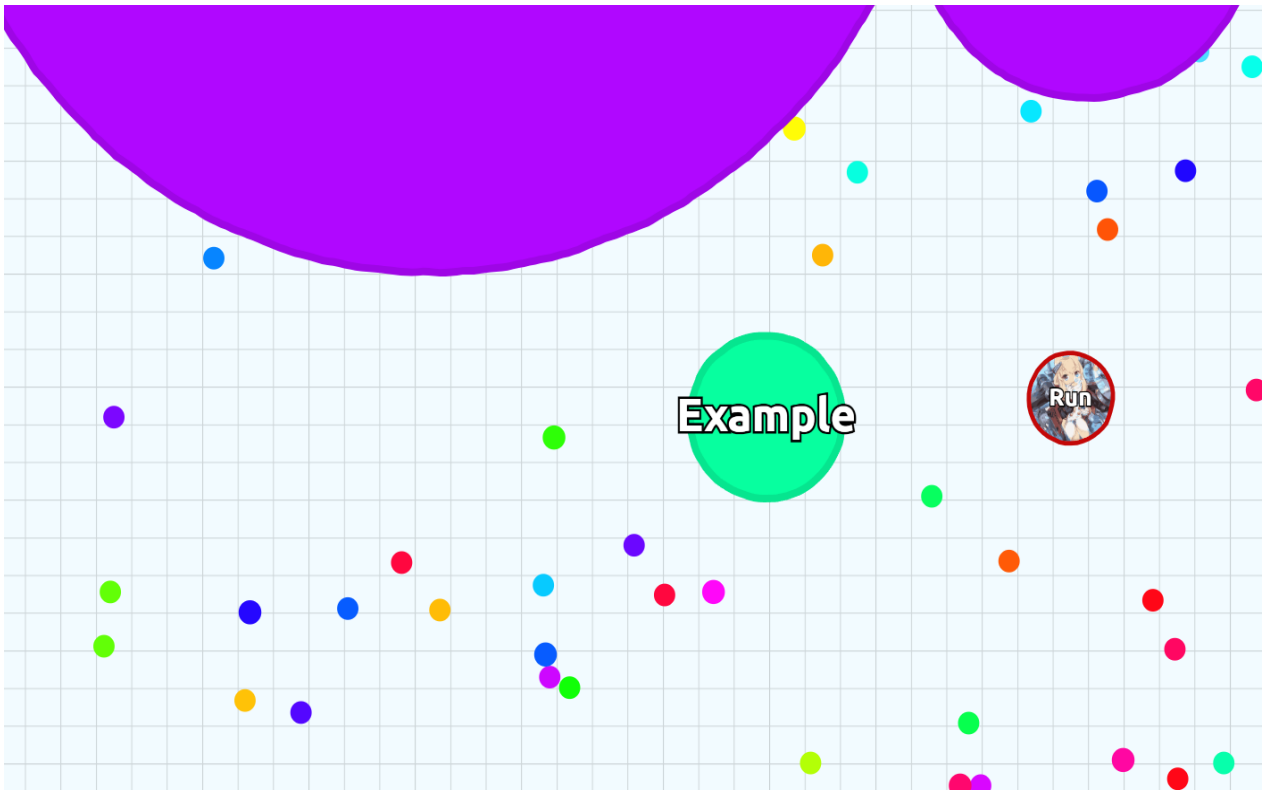


Рисунок 2.2.1 – интерфейс игры “agar.io”

"Wormate.io":

Правила: Вам нужно управлять червем и собирать фрукты, чтобы стать длиннее и сильнее. Цель игры - выживать и избегать столкновений с другими червями, которые могут съесть вас.

На рисунке 2.1.2 показан интерфейс игры “wormate.io”

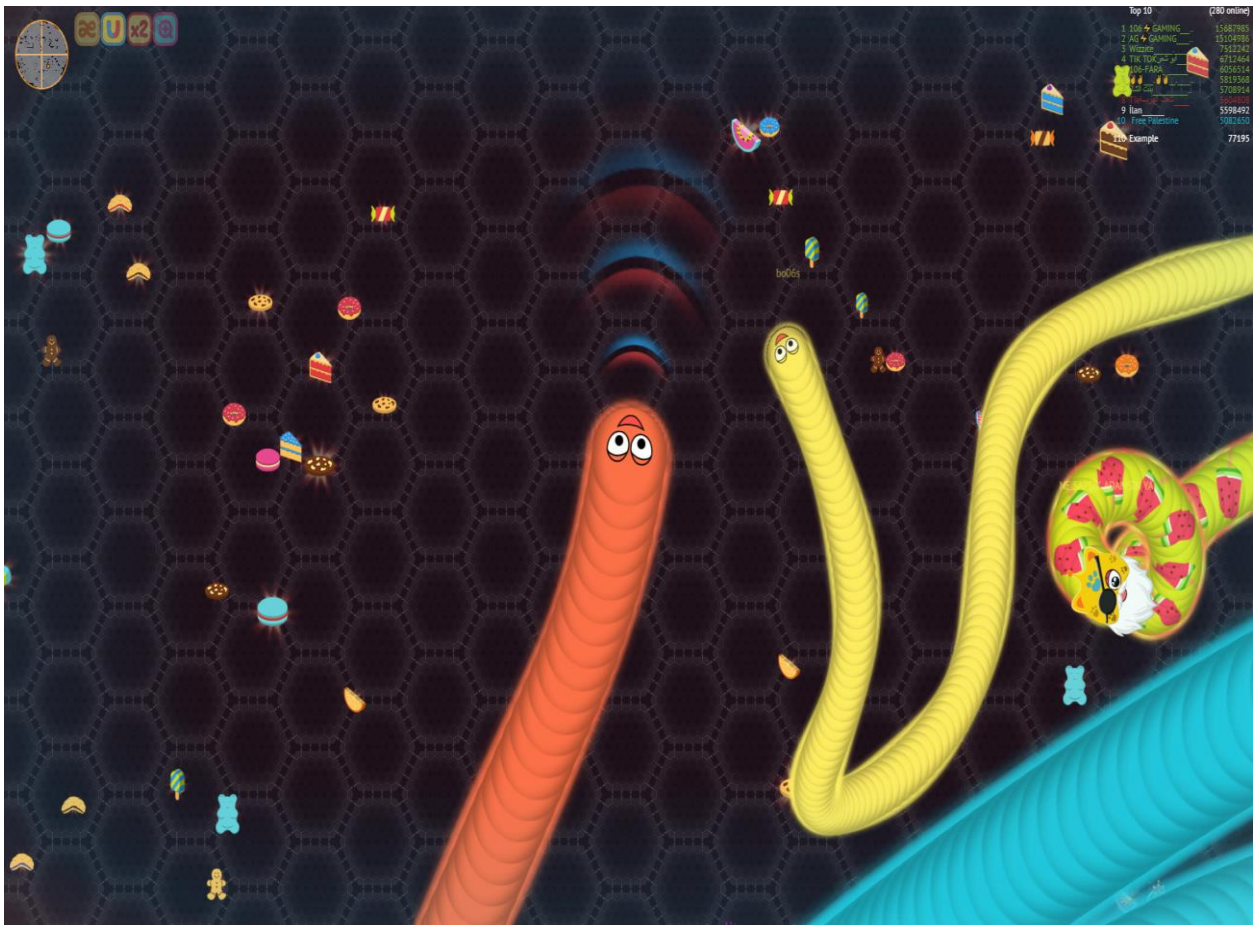

Рисунок 2.2.2 – интерфейс игры “wormate.io”

"Paper.io":

Правила: В этой игре вы управляете квадратом, который должен захватывать территорию, рисуя линии. Цель игры - захватить как можно больше территории и избегать столкновений с другими игроками.

На рисунке 2.1. показан интерфейс игры “paper.io”

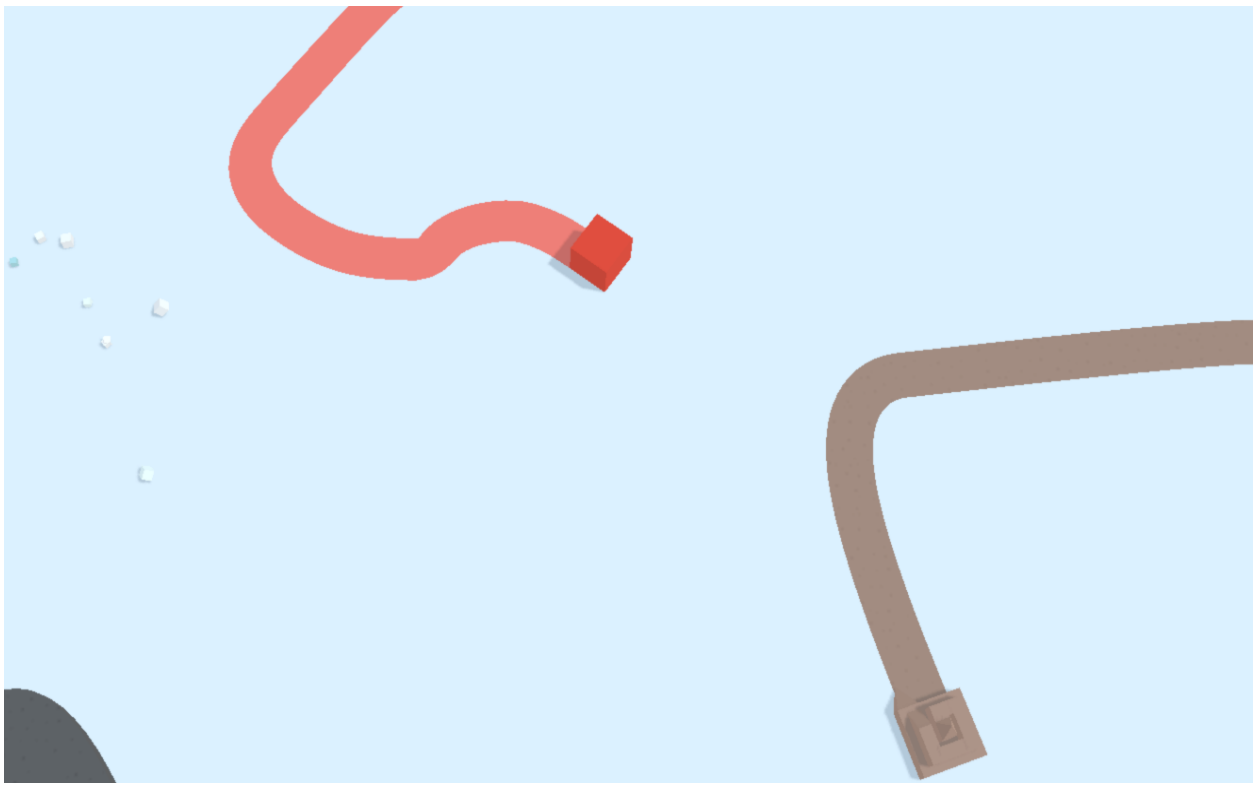

Рисунок 2.2.3 – интерфейс игры “paper.io”

В процессе разработки программы были использованы различные возможности языка C++, которые будут описаны ниже. Для решения задачи был выбран язык программирования C++ и методология объектно-ориентированного программирования.

Для хранения объектов было использовано массивы. Также реализован класс game хранящий информацию о игре для использования в интерфейсе программы.

Для удобства работы с классами snake и snakeBot было использовано наследования, которое позволило использовать методы и поля классов, используемые в двух классах. Так же были использованы такой принцип как инкапсуляция. На практике это означает, что класс должен состоять из двух частей: интерфейса и реализации.

# 3 ФУНКЦИОНАЛЬНОЕ ПРОЕКТИРОВАНИЕ

В данном разделе описываются входные и выходные данные программы, диаграмма классов, а также приводится описание используемых классов и их методов.

## 3.1 Структура входных и выходных данных

В данном проекте был выбран язык программирования C++, а также применена методология объектно-ориентированного программирования (ООП). Далее рассмотрим более подробно основные этапы решения задачи, включая входные и выходные данные.

### 3.1.1 Входные данные

В данном проекте осуществлена возможность для пользователя взаимодействия с приложением через ввод на клавиатуре клавиш W, A, S, D для первого игрока и UP, RIGHT, DOWN, LEFT для второго игрока соответственно (Ц, Ф, Ы, В/ВВЕРХ, ВПРАВО, ВНИЗ, ВЛЕВО). Данные клавиши отвечают за управление движением контроллируемого объекта. Также предусмотрена возможность выйти из игры нажатием кнопки esc, перезапуск игры нажатием клавиши R(К) и включение/выключение звука нажатием кнопки P (З).

### 3.1.2 Выходные данные

На выход программы идет пользовательский интерфейс на котором первоначально видно окно выбора режима игры, затем игровое окно, где отображается игровое поле, игровой персонаж, и счет длины червячка.

Таким образом, разработанное приложение на языке C++ с применением объектно-ориентированного программирования обеспечивает гибкость и универсальность в обработке игрового интерфейса и взаимодействия пользователя с игрой.

## 3.2 Разработка диаграммы классов

Диаграмма классов - это инструмент в языке моделирования UML, который используется для визуализации структуры классов в системе, их атрибутов, методов, интерфейсов и отношений между ними. Эта диаграмма обычно применяется при проектировании архитектуры, документировании системы, уточнении требований, а также для поддержки системы.

Диаграмма классов иллюстрирует модели данных даже для очень сложных информационных систем. На ней представлены в рамках, содержащих три компонента:

В верхней части написано имя класса. Имя класса выравнивается по центру и пишется полужирным шрифтом;

В средней части перечислены атрибуты (поля) класса;

В нижней части перечислены методы класса.

Диаграмма классов служит для визуализации статического представления системы, представляя различные аспекты приложения. Она представляет собой графическое представление статического представления системы и представляет различные аспекты приложения и также может включать в себя классы, их атрибуты, методы и отношения между классами, такие как наследование, агрегация, ассоциация, множественность ассоциации, обобщение, зависимость, использование, реализация и композиция.

Диаграмма классов данной работы показана в приложении А.

### 3.3 Описание классов.

Для создания игры червячки в C++ с интерфейсом через библиотеку Raylib, была реализована следующая структуру классов:

#### 3.3.1 Класс snake

Snake – класс в котором реализуется работа червячка

Описание полей класса:

`int cellSizeSnake` - размер одного деления червячка

`Color darkGreenSnake` - цвет червячка

`deque<Vector2> body` - дэк тела червячка

`Vector2 direction` - направление червячка

`Vector2 directionStart` - стартовое направление червячка

Описание методов класса:

`void drawSnake(deque<Vector2> body)` – метод класса snake рисует червячка на игровом экране

`void Update(Vector2 direction)` – метод класса snake обновляет длину червячка и его положение на поле

`void reset(deque<Vector2> bodyStart, Vector2 directionStart)`- метод класса snake возвращает червячка на стартовое положение

#### 3.3.2 Класс walls

Walls – класс в котором реализуются стены на игровом пространстве

Описание полей класса:

`deque<V> wall` – Содержит в себе данные о стенах

`deque<V> map1` – Содержит в себе координаты стен карты

Описание методов класса:

`void drawWall(int i)` - метод класса walls отрисовывает стены на игровом пространстве

### 3.3.3 Класс food

Food – класс в котором реализуется функционал “еды” и его рандомная генерация

Описание полей класса:

`Vector2 position` – содержит координаты “еды”

`Texture2D texture` – содержит в себе картинку “еды”

Описание методов класса:

`void drawFood()` – метод класса food отрисовывает “еду” на игровом поле

`Vector2 GenerateRandomCell()`- метод класса food генерирует случайное значение координат “еды”

`Vector2GenerateRandomPos(deque<Vector2>snakeBody,deque<Vector2> wall)` – метод класса food изменяет положение еды в случае ее появления в другом игровом объекте

### 3.3.4 Класс game

Game – класс в котором реализуется основная работа игры: обновление положения всех объектов, проверка взаимодействия их между собой, поведение бота-червечка

Описание полей класса:

`bool running` – булевая переменная контролирующая работу игры

`int score` – содержит очки первого игрока

`int score2` – содержит очки второго игрока

`Sound eatFoodSound` – содержит звуковой файл взаимодействия с едой

`Sound wallCollideSound` – содержит звуковой файл взаимодействия с стенами

Описание методов класса:

`void Draw(int i)` – метод класса game отрисовывает все игровые объекты

`void Update()` – метод класса game обновляет положение всех игровых объектов

`void checkCollisionWithFood()` – метод класса game проверяет взаимодействие с “едой”

`void checkCollisionWithEdges()`- метод класса game проверяет взаимодействие с стенами

`void gameOver()` – метод класса game функция окончания игры

`void checkCollisionWithSnake()`- метод класса game функция взаимодействия с червячком

`void checkCollisionWithTail()`- метод класса game функция взаимодействия с хвостом червячка

`void resetSnake()` – метод класса game функция возвращения начальных значений червячка

`void snakeAI()` – метод класса game функция содержащая логику поведения червячка-бота

`void win()` – метод класса game функция условия победы игроков

`void resetSnake()` – метод класса game фунция сброса положения змеи

# 4 РАЗРАБОТКА ПРОГРАММНЫХ МОДУЛЕЙ

## 4.1 Разработка схем алгоритмов

Метод `checkCollisionWithFood()` проверяет на столкновение червячка с едой. Схема метода `checkCollisionWithFood()` показана в приложении Б.

Метод `Draw(int i)` отрисовывает все игровые элементы на игровом поле.

Схема метода `Draw(int i)` показана в приложении В.

## 4.2 Разработка алгоритмов

### 4.2.1 Метод Update() класса game

Метод Update() в классе Game отвечает за обновление состояния игры на каждом шаге. Вот пошаговое описание работы метода Update():

Шаг1. Проверка, запущена ли игра (running). Если игра не запущена, то метод Update() просто завершается.

Шаг2. Обновление состояния змеи snake2 вызовом метода Update2() с передачей текущего направления snake2.direction2. Этот метод обновляет позицию и состояние змеи snake2 в соответствии с выбранным направлением.

Шаг3 .Обновление состояния змеи snake вызовом метода Update() с передачей текущего направления snake.direction. Этот метод обновляет позицию и состояние змеи snake в соответствии с выбранным направлением.

Шаг4. Проверка столкновения с едой вызовом метода checkCollisionWithFood(). Если голова змеи snake.body[0] совпадает с позицией еды food.position, то считается, что змея поймала еду. В этом случае позиция еды обновляется случайным образом, змее добавляется сегмент, и увеличивается счетчик очков score. Также проигрывается звуковой эффект "EatFood.wav", если переменная toggle равна true.

Шаг5. Проверка столкновения с границами поля вызовом метода checkCollisionWithEdges(). Если голова змеи snake.body[0] достигает границы поля (координата x равна cellCount или -1, или координата y равна cellCount или -1), то вызывается метод resetSnake(). Этот метод сбрасывает состояние змеи snake в начальное состояние, обнуляет счетчик очков score и проигрывает звуковой эффект "WallCollide.wav", если toggle равно true.

Шаг6. Проверка столкновения с границами поля для змеи snake2 вызовом метода checkCollisionWithEdges2(). Если голова змеи snake2.body2[0] достигает границы поля (координата x равна cellCount или -1, или координата y равна cellCount или -1), то вызывается метод resetSnake2(). Этот метод сбрасывает состояние

змеи snake2 в начальное состояние, обнуляет счетчик очков score2 и проигрывает звуковой эффект "WallCollide.wav", если toggle равно true.

Шаг7. Проверка столкновения с другой змеей вызовом метода checkCollisionWithSnake(). Если голова змеи snake.body[0] совпадает с любым сегментом змеи snake2.body2, то вызывается метод resetSnake(), сбрасывающий состояние змеи snake в начальное состояние и обнуляющий счетчик очков score. Аналогично, если голова змеи snake2.body2[0] совпадает с любым сегментом змеи snake.body, то вызывается метод resetSnake2(), сбрасывающий состояние змеи snake2 в начальное состояние и обнуляющий счетчик очков score2.

Шаг8. Проверка столкновения с хвостом змеи вызовом методов checkCollisionWithTail() и checkCollisionWithTail2(). Если голова змеи snake.body[0] совпадает с любым сегментом хвоста змеи snake.body, то вызывается метод resetSnake(), сбрасывающий состояние змеи snake в начальное состояние и обнуляющий счетчик очков score. Аналогично, если голова змеи snake2.body2[0] совпадает с любым сегментом хвоста змеи snake2.body2, то вызывается метод resetSnake2(), сбрасывающий состояние змеи snake2 в начальное состояние и обнуляющий счетчик очков score2.

Шаг9. Проверка столкновения с препятствиями вызовом методов checkCollisionWithWalls() и checkCollisionWithWalls2(). Если голова змеи snake.body[0] совпадает с любым сегментом препятствия wall.wall, то вызывается метод resetSnake(), сбрасывающий состояние змеи snake в начальное состояние и обнуляющий счетчик очков score. Аналогично, если голова змеи snake2.body2[0] совпадает с любым сегментом препятствия wall.wall, то вызывается метод resetSnake2(), сбрасывающий состояние змеи snake2 в начальное состояние и обнуляющий счетчик очков score2.

### 4.2.2 Метод snakeAI() класса game

Метод snakeAI() представляет собой часть алгоритма искусственного интеллекта для управления змеей snake2. Вот описание работы данного метода:

Шаг1.Создается двумерный массив used размером 30x30, который используется для отслеживания посещенных клеток поля.

Шаг2.Создается очередь q и инициализируются переменные x0, y0, x1, y1 для хранения координат головы змеи snake2, а также координат еды food.

Шаг3.Голова змеи snake2 добавляется в очередь q, а массив used инициализируется значением 0.

Шаг4.Для каждой клетки препятствия из объекта wall.wall, соответствующий элемент массива used устанавливается в 1, чтобы обозначить его занятость.

Шаг5.Аналогично, для каждого сегмента змеи snake2.body2, соответствующий элемент массива used устанавливается в 1.

Шаг6.Затем происходит аналогичная проверка для каждого сегмента змеи snake.body.

Шаг7.Создается массив p размером 30x30, который будет хранить информацию о предыдущей клетке для каждой посещенной клетки.

Шаг8.Запускается цикл, пока очередь q не станет пустой.

Шаг9.Извлекается первый элемент из очереди q и сохраняются его координаты в переменные x и y.

Шаг10.Если текущие координаты равны координатам еды food, то цикл прерывается.

Шаг11.Для каждого из четырех направлений (влево, вправо, вверх, вниз) проверяется возможность движения в новую клетку.

Шаг12.Если новые координаты находятся в пределах поля и клетка не занята, то текущие координаты становятся предыдущими для новых координат, новые координаты добавляются в очередь q и клетка помечается как посещенная в массиве used.Создается пустой вектор path, который будет содержать последовательность клеток, образующих путь к еде.

Начиная с координат еды food, происходит обратный проход по пути, используя массив p, и клетки добавляются в вектор path.Вектор path инвертируется, чтобы получить правильную последовательность клеток пути.Если первая клетка пути не является начальной позицией змеи snake2, то устанавливаются значения x и y для направления движения змеи snake2, что определяется как разность координат первой клетки пути и текущей головы змеи snake2.

Шаг13.Метод snakeAI() завершается.

## 5 РЕЗУЛЬТАТ РАБОТЫ ПРОГРАММЫ

На рисунке 6.1 изображена начало работы программы. При старте программы открывается окно выбора игрового режима. На данном экране со доступна возможность выбрать 1 из 3 доступных игровых режимов, включить/отключить звук, выйти из игры.

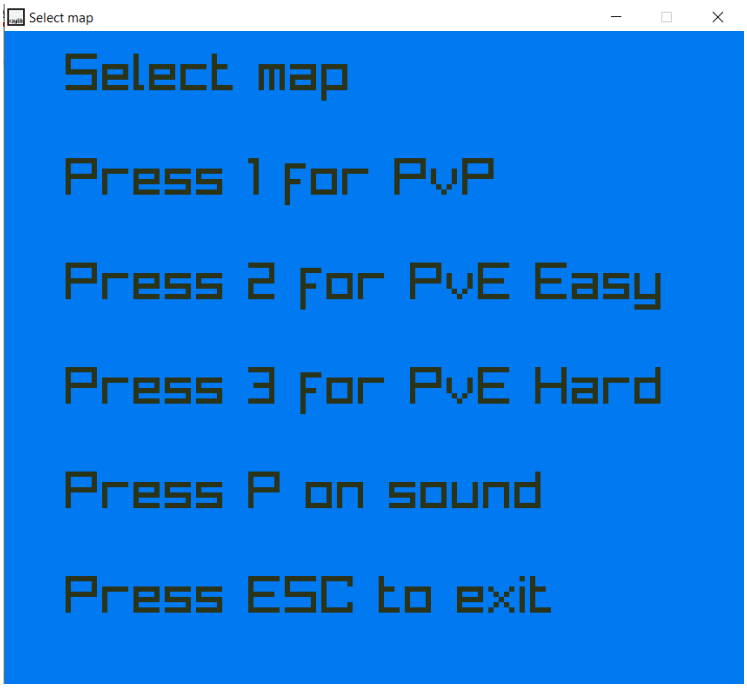


Рисунок 5.1 – окно выбора игрового режима

На рисунке 5.2 показано игровое окно для двух игроков. Первый игрок на игровом поле изображен красным цветом, а второй игрок изображен серым. На карте присутствуют черные перегородки услажняющие передвижение по карте. При нажатии любой из кнопок управления персонажем игра начинается. При смерти игрока игрок возращается в начальное положение и игра продолжается до тех пор пока один из игроков не наберет 20 очков.

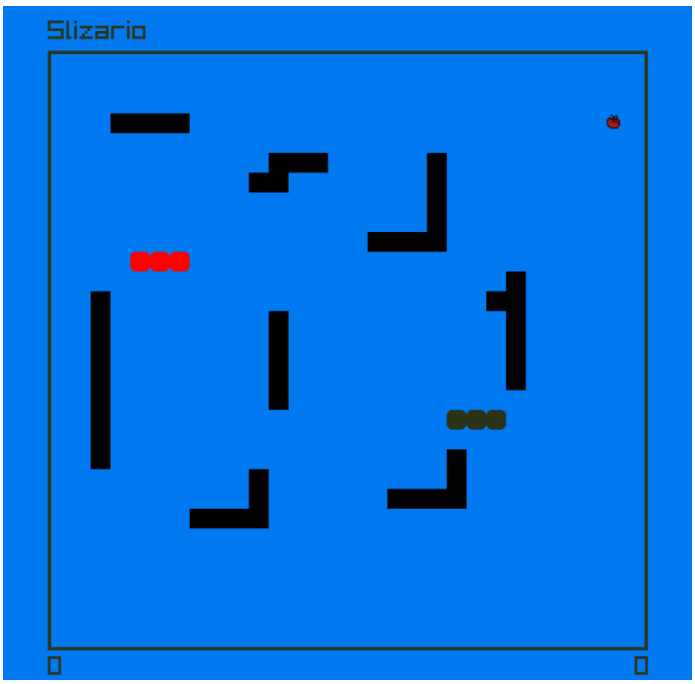


Рисунок 5.2 – окно игры первой карты

На рисунке 5.3 показано игровое окно для одного игрока с ботом. Игрок изображен на игровом поле изображен красным цветом, а бот серым. На карте присутствуют черные перегородки услажняющие передвижение по карте. При смерти игрока игрок возращается в начальное положение и игра продолжается до тех пор игрок или бот не наберет 20 очков.

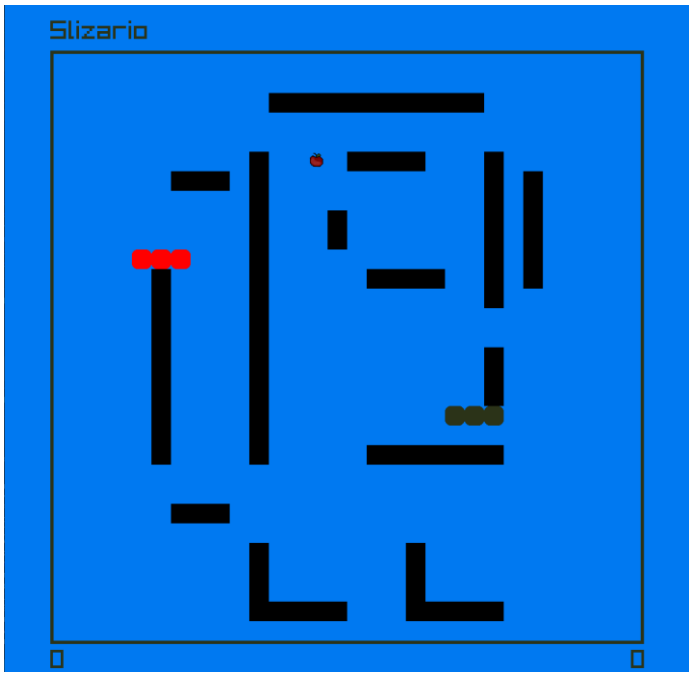


Рисунок 5.3 – окно игры второй карты

# ЗАКЛЮЧЕНИЕ

Итак, изучение объектно-ориентированного программирования (ООП) выдало себя как преобразующий этап в моем понимании и подходе к разработке программного обеспечения. Погружение в мир ООП расширило мои знания и компетенции, обогатив мой инструментарий в области программирования.

Одним из ключевых выводов, которые я сделал в процессе написания курсовой работы, является то, что ООП не только предоставляет эффективные средства организации кода, но и обеспечивает более высокий уровень абстракции, что делает разработку программ более интуитивной и гибкой. Использование классов и объектов позволяет создавать модульные, легко читаемые и поддерживаемые системы.

В наше современное время, когда требования к программному обеспечению постоянно растут, ООП становится неотъемлемой частью профессиональной подготовки разработчика. Адаптивность кода, возможность масштабирования проектов и повторного использования компонентов становятся критическими аспектами успешной разработки. ООП предоставляет технологический фреймворк для разработки сложных, но гибких систем, способных адаптироваться к изменяющимся требованиям бизнеса.

В заключение, при разработке игры “Червячки” был создан мощный бот имитирующий поведения игрока. Это убирает ограничение для игры в надобности второго игрока а также добавляет новое испытание в игру, так как для победы над ботом надо овладеть умениями в данную игру.

## СПИСОК ЛИТЕРАТУРЫ

[1] "Объектно-ориентированное программирование на C++" Бьярн Страуструп

[2] "Язык программирования C++" Герберт Шилдт

[3] "C++ Primer" Липман, Лажойе, Му, Хопкинс

[4] "Алгоритмы. Построение и анализ" Кормен, Лейзерсон, Ривест, Штайн

[5] "Введение в алгоритмы" Кормен, Лейзерсон, Ривест, Штайн

[6] "Алгоритмы на C++" Роберт Седжвик, Кевин Уэйн

[7] "C++: язык и его использование" автор Бьерн Страуструп

[8] "C++: стандарты программирования" автор Герберт Шилдт

[9] "C++: объектно-ориентированное программирование" автор Роберт Лафоре

[10] "C++: программирование на языке высокого уровня" автор Левитин А. В.

[11] Программирование на C++ [Электронный ресурс]. -Электронные данные. Режим доступа: https://metanit.com/cpp/tutorial/ -Дата доступа: 27.11.2023.

[12] Документация Raylib [Электронный ресурс]. –Режим доступа: https://www.raylib.com/cheatsheet/cheatsheet.html –Дата доступа: 27.11.2023.

[13] Обход графа в ширину (BFS) и глубину (DFS) [Электронный ресурс]. – Режим доступа**:** https://habr.com/ru/articles/661577/ –Дата доступа: 27.11.2023.

**ПРИЛОЖЕНИЕ А**

*(обязательное)*

Диаграмма классов

## ПРИЛОЖЕНИЕ Б

*(обязательное)*

Схема метода `checkCollisionWithFood()`

## ПРИЛОЖЕНИЕ В

*(обязательное)*

Схема метода `Draw(int i)`

## ПРИЛОЖЕНИЕ Г

*(обязательное)*

Код программы

**ПРИЛОЖЕНИЕ Д**

*(обязательное)*

Ведомость документов