**Министерство науки и высшего образования Российской Федерации**

федеральное государственное автономное образовательное учреждение  
высшего образования

«Национальный исследовательский Томский политехнический университет»

|  |  |
| --- | --- |
| Школа | Инженерная школа информационных технологий и робототехники |
| Обеспечивающее подразделение | Отделение информационных технологий |
| Направление подготовки / специальность | 09.03.02 Информационные системы и технологии |
| Образовательная программа (направленность (профиль)) | Искусственный интеллект и машинное обучение |

**ОТЧЕТ О ПРАКТИКЕ**

|  |  |
| --- | --- |
| Вид практики | Учебная практика |
| Тип практики | Учебная практика по развитию цифровых компетенций |
| Место практики | Отделение информационных технологий |

|  |  |
| --- | --- |
| Выполнил обучающийся | Михайлов Глеб Алексеевич |
| Группа | 8И32 |

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(подпись обучающегося)

Руководитель практикиТПУ:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(степень, звание, должность) (Ф. И. О.)

Дата проверки \_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_ 20\_\_г.

Допустить / не допустить к защите

Подпись \_\_\_\_\_\_\_\_\_\_\_\_

Итоговая оценка по практике \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(традиционная оценка, балл)

Томск 2024

Оглавление

[Индивидуальное задание на практику 3](#_Toc172224420)

[Введение 3](#_Toc172224421)

[Актуальность 4](#_Toc172224422)

[Цель работы 4](#_Toc172224423)

[Задачи 4](#_Toc172224424)

[Понедельный план разработки игры и распределение ролей 5](#_Toc172224425)

[Анализ и выбор средств реализации программы 6](#_Toc172224426)

[Описание алгоритма работы программы 7](#_Toc172224427)

[Описание полученных результатов 8](#_Toc172224428)

[Работа программы 8](#_Toc172224429)

[Дополнительный функционал 10](#_Toc172224430)

[Графический интерфейс 12](#_Toc172224431)

[Плохо работающие моменты 12](#_Toc172224432)

[Заключение 13](#_Toc172224433)

[Дальнейшее развития проекта 13](#_Toc172224434)

[Список источников 15](#_Toc172224435)

[ПРИЛОЖЕНИЕ А 16](#_Toc172224436)

**Индивидуальное задание на практику**

Разработать игру «Змейка» на языке программирования C++ без использования сторонних библиотек и графического интерфейса (т.е игра должна быть реализована непосредственно в консоле).

Введение

«Змейка» — это классическая аркадная игра, в которой игрок управляет длинной, тонкой линией, напоминающей змею, которая движется по игровому полю. Цель игры заключается в том, чтобы управлять змейкой, собирая различные объекты (обычно это еда), которые появляются на игровом поле. Каждый собранный объект увеличивает длину змейки. Игрок должен избегать столкновений с границами игрового поля и собственным телом змейки.

Принцип игры

1. Управление: игрок управляет змейкой с помощью клавиш направления (вверх, вниз, влево, вправо).

2. Сбор объектов: на игровом поле появляются объекты (еда), которые змейка должна собирать. Каждый собранный объект увеличивает длину змейки.

3. Избегание столкновений: игрок должен избегать столкновений с границами игрового поля и собственным телом змейки. Если змейка сталкивается с чем-либо, игра заканчивается.

4. Увеличение сложности: по мере того как змейка становится длиннее, управлять ею становится сложнее, так как увеличивается вероятность столкновения с собственным телом.

Игра заканчивается, когда змейка сталкивается с границей игрового поля или своим телом.

Актуальность

Игра "Змейка" является классическим примером аркадной игры, которая позволяет изучить и применить на практике основные концепции программирования, такие как работа с массивами, управление вводом и выводом, обработка событий и реализация алгоритмов. Реализация данной игры на языке C++ без использования сторонних библиотек и графического интерфейса позволяет углубить знания в области программирования и алгоритмизации, а также развить навыки работы с консольными приложениями.

Цель работы

Цель работы: разработать консольную игру «Змейка» посредством языка программирования C++.

****Задачи****

1. Изучить основные принципы работы с консольным вводом и выводом в C++;
2. Разработать алгоритм управления змейкой с помощью клавиш направления;
3. Реализовать генерацию объектов (еды) на игровом поле;
4. Обеспечить увеличение длины змейки при сборе объектов;
5. Реализовать проверку столкновений змейки с границами игрового поля и собственным телом;
6. Обеспечить вывод текущего счета игрока на экран;
7. Разработать функциональное и понятное меню;
8. Реализовать возможность выбора карты и уровня сложности;
9. Добавить сохранение рекордов;
10. Провести тестирование и отладку программы.

Понедельный план разработки игры и распределение ролей

Таблица 1. – Понедельный план разработки игры

|  |  |  |
| --- | --- | --- |
| Неделя 1 (24 – 30 июня) | Проектирование основы и каркаса игры | 1. Анализ требований и проектирование.  2. Создание базового каркаса игры: реализация основного игрового цикла.  3. Разработка функции обновления и отображения экрана.  4. Проектирование классов Game, Snake и Food. |
| Неделя 2 (1 – 7 июля) | Разработка функциональности и логики игры | 1. Реализация пользовательского ввода для управления змейкой.  2. Доработка механики управления.  3. Обработка столкновений игровых объектов.  4. Реализация выбора сложности игры и изменения скорости змейки. |
| Неделя 3 (8 – 14 июля) | Завершение и тестирование | 1. Реализация функции загрузки игровых карт.  2. Позиционирование и отрисовка препятствий на игровом поле.  3. Тестирование и отладка.  4. Обнаружение и исправление багов. |

Исходя из набора уникальных навыков и опыта каждого из участников проекта, роли в проекте были распределены следующим образом:

* Михайлов Глеб Алексеевич — аналитик, разработчик;
* Билык Владислав Сергеевич — главный разработчик.

Анализ и выбор средств реализации программы

Данная работа подразумевает, что в процессе создания не будут использоваться сторонние библиотеки и графический интерфейс, поэтому было принято решение использовать библиотеки и модули, интегрированные в язык программирования C++. Основные средства реализации программы включают:

* **Стандартная библиотека C++**: используется для работы с вводом и выводом данных, генерации случайных чисел и управления временем выполнения программы. Стандартная библиотека C++ предоставляет широкий набор функций и классов, которые облегчают разработку программного обеспечения и позволяют сосредоточиться на логике приложения.
* **<iostream>**: подключается для ввода и вывода данных в консоль. Этот заголовочный файл предоставляет функции для работы с потоками ввода и вывода, что позволяет легко взаимодействовать с пользователем через консольное окно.
* **<conio.h>**: задействуется для работы с функциями \_kbhit() и \_getch(), которые позволяют обрабатывать нажатия клавиш. Эти функции необходимы для реализации интерактивного управления змейкой в реальном времени.
* **<windows.h>**: применяется для использования функции Sleep(), которая позволяет приостанавливать выполнение программы на заданное время. Эта функция используется для создания задержек между обновлениями экрана, что делает игру более плавной и управляемой.
* <string>: используется для возможности работать со строковыми типами данных.
* <stdlib.h>: подключается для распределения памяти компьютера, управлением процессами и преобразованием.
* <fstream>: вызывается для работы с файлами (чтение файла, запись данных в файл).

В качестве среды программирования был выбран текстовый редактор CodeBlocks, поскольку он весьма легок в использовании и имеет интуитивно понятный интерфейс. Помимо этого, для более удобной работы в команде было решено использовать сервис GitHub. GitHub позволяет нескольким разработчикам работать над одним и тем же проектом одновременно. Система pull requests и возможность легко просматривать изменения позволяют командам эффективно сотрудничать. Также GitHub использует Git – мощную систему контроля версий, которая позволяет отслеживать все изменения в проекте. Это делает возможным возврат к любой предыдущей версии кода при необходимости. Для удобной работы с GitHub, было решено использовать программу GitHub Desktop.

Описание алгоритма работы программы

Программа реализует классическую игру «Змейка», в которой игрок управляет змейкой, собирая объекты (еду) и избегая столкновений с границами игрового поля и собственным телом. Основные моменты алгоритма включают:

1. **Инициализация игры**: установка начальных значений переменных, таких как положение змейки, положение еды, счет и направление движения. На этом этапе создаются все необходимые объекты и переменные, которые будут использоваться в ходе игры.
2. **Отрисовка игрового поля**: вывод границ игрового поля, змейки и еды на экран. Этот шаг включает в себя обновление экрана и отображение текущего состояния игры, что позволяет игроку видеть, где находится змейка и еда.
3. **Обработка ввода**: проверка нажатий клавиш и изменение направления движения змейки в зависимости от нажатой клавиши. Этот шаг обеспечивает интерактивность игры, позволяя игроку управлять змейкой с помощью клавиш направления.
4. **Логика игры**: обновление положения змейки, проверка столкновений с границами игрового поля и собственным телом, увеличение длины змейки при сборе еды и обновление счета. Этот шаг включает в себя основные правила игры и определяет, что происходит при каждом обновлении экрана.
5. **Цикл игры**: повторение шагов отрисовки, обработки ввода и логики игры до тех пор, пока игра не закончится. Этот цикл продолжается до тех пор, пока игрок не проиграет, столкнувшись с границей или собственным телом.

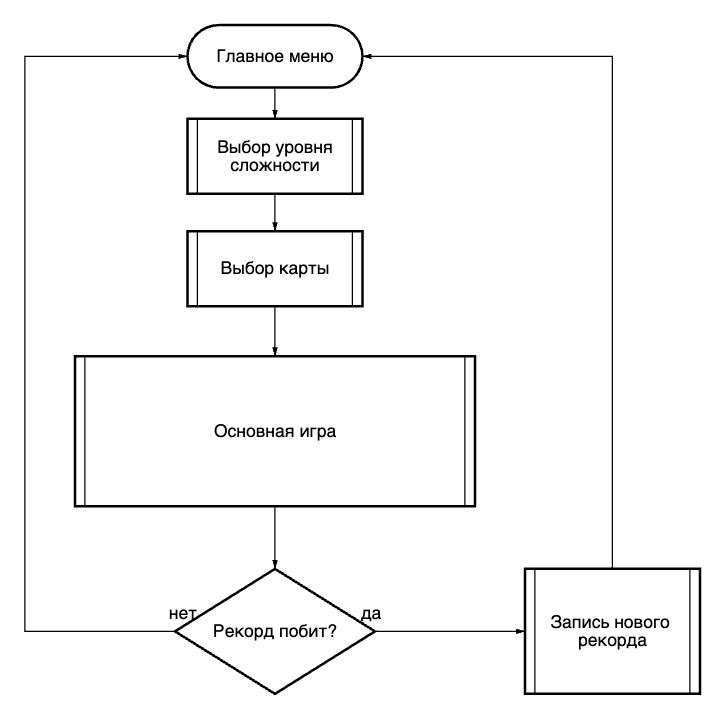


Рисунок 1 – схема работы программы

Описание полученных результатов

Программа успешно реализована и работает корректно. Основные результаты включают:

**Работа программы**

Игра запускается в консоли, змейка управляется с помощью клавиш направления, Объекты еды генерируется случайным образом на игровом поле. Собирая объекты еды, игрок может увеличивать счет. По мере сбора еды, увеличивается и скорость змейки, что усложняет игру. Программа демонстрирует стабильную работу и корректное выполнение всех реализованных функций. При запуске игры на экране отображается игровое поле с границами, змейкой и едой. Игрок может управлять змейкой с помощью клавиатуры, собирая еду и избегая столкновений. Программа корректно обрабатывает все действия игрока и обновляет состояние игры в реальном времени.



Рисунок 2 – Игровой процесс



Рисунок 3 – Окно окончания игры

Дополнительный функционал

Для более удобного использования программы, было реализовано меню, в котором можно узнать текущие игровую карту и уровень сложности.



Рисунок 4 – Меню игры

При любом запуске игры у игрока есть возможность выбрать уровень сложности (чем сложнее уровень — тем быстрее двигается змейка, всего уровней 3)



Рисунок 5 – Меню выбора сложности игры

Помимо этого, игрок может выбрать карту игрового поля из трех предложенных, для этого также было реализовано удобное меню.

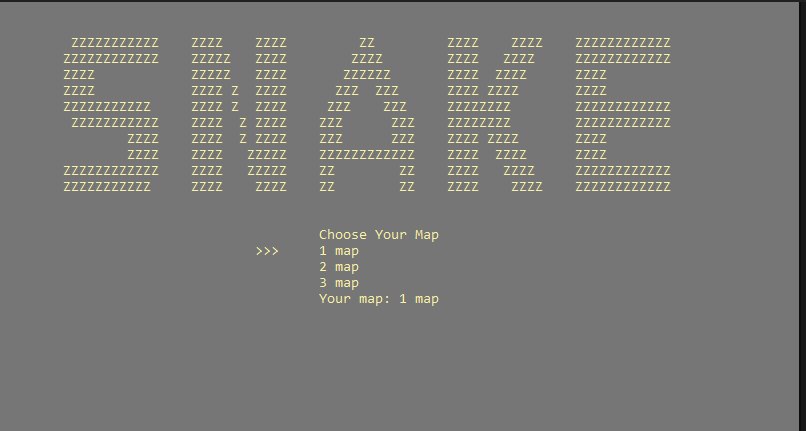


Рисунок 6 – Меню выбора карты

Также, в игре реализована система сохранения лучших попыток. Для этого, при завершении ваш результат сравнивается с лучшим, а в главном меню есть возможность посмотреть лучшие результаты при любом уровне сложности.



Рисунок 7 – Окно рекордов

**Графический интерфейс**

Программа использует консольный интерфейс для отображения игрового поля и управления игрой. Консольный интерфейс позволяет легко взаимодействовать с пользователем и отображать всю необходимую информацию.

**Плохо работающие моменты**

В некоторых случаях возможны задержки при обработке ввода, что может приводить к незначительным задержкам в управлении змейкой. Эти задержки могут быть вызваны особенностями работы консольного ввода-вывода и могут быть устранены путем оптимизации кода.

Заключение

В ходе выполнения учебной практики была реализована классическая игра «Змейка» на языке программирования C++ без использования сторонних библиотек и графического интерфейса. Были изучены и применены на практике основные концепции программирования, такие как работа с массивами, управление вводом и выводом, обработка событий и реализация алгоритмов. Программа успешно работает и позволяет управлять змейкой, собирая еду и избегая столкновений. Были реализованы возможности выбора любого уровня и смены уровня сложности игры. В процессе выполнения работы были получены ценные навыки и знания в области программирования и алгоритмизации. Процесс реализации игры «Змейка» позволил углубить понимание принципов работы с консольными приложениями и научиться эффективно использовать стандартные библиотеки C++ для решения практических задач.

Репозиторий проекта на GitHub (пользователь: flexyw1be): <https://github.com/flexyw1be/snakeGame> (дата обращения: 18 июля 2024 г.).

Дальнейшее развития проекта

* Добавить различные бонусы, которые дают временные улучшения змейке, например, увеличение скорости, возможность проходить через собственный хвост или получения дополнительных очков.
* Введение режима игры на двоих или даже больше игроков, где они могут соревноваться друг с другом или играть в кооперативном режиме.
* Реализовать объекты на поле, которые движутся или меняют свое местоположение, добавляя элемент сложности и необходимости продумывать свои шаги наперед.
* Введение различных типов еды с разными эффектами: некоторые могут замедлять или ускорять змейку, давать дополнительные жизни или временные преимущества.
* Введение опций для настройки уровня сложности, скорости змейки, размеров поля и других параметров, чтобы игроки могли создавать свой идеальный игровой опыт.

Список источников

1. Руководство по языку программирования С++ // Metanit URL: https://metanit.com/cpp/tutorial/ (дата обращения: 01.07.2024).
2. Описание игры «Змейка» // Wikipedia URL: https://en.wikipedia.org/wiki/Snake\_(video\_game\_genre) (дата обращения: 25.06.2024).
3. Документация по языку программирования C++ // Learn.Microsoft URL: https://learn.microsoft.com/ru-ru/cpp/cpp/?view=msvc-170 (дата обращения: 01.07.2024).
4. Классы - Основы С++ // Яндекс Образование URL: https://education.yandex.ru/handbook/cpp/article/classes (дата обращения: 02.07.2024).

ПРИЛОЖЕНИЕ А

Исходный код игры

#**include** <iostream> //подключение библиотек

#**include** <windows.h>

#**include** <string>

#**include** <conio.h>

#**include** <stdlib.h>

#**include** <fstream>

**using** **namespace** std;

ifstream fin; //подключаем модуль для работы с файлами

//БЛОК ОБЪЯВЛЕНИЯ ГЛОБАЛЬНЫХ ПЕРЕМЕННЫХ

//Объявление констант

const int WIDTH = 23; //ширина игрового поля

const int HEIGHT = 22; //высота игрового поля

const int MENUPOINTS = 6; //кол-во пунктов в меню

const int MAX\_LEN\_SNAKE = (WIDTH - 3) \* (HEIGHT -2); //максимальная длина змейки

//переменные, отвечающие за направление движения

const int UP = 0;

const int DOWN = 1;

const int LEFT = 2;

const int RIGHT = 3;

bool isRunning = true; //переменная для работы игрового цикла

bool exxit = false; //переменная для работы цикла меню

// целочисленные переменные

int coef; //коэффицент для увеличения скорости змейки

int currMenu = 0; //переменная для выбора пункта меню

int ch = 0; //переменная дла запоминания нажатия клавиатуры

int diff = 4; //сложность игры

int easy\_r = 0; //рекорд в легком режиме

int hard\_r= 0; //рекорд в среднем режиме

int insane\_r=0; //рекорд в сложнос режиме

int bestScore; //лучший результат в выбранном режиме

HANDLE hStdOut = GetStdHandle(STD\_OUTPUT\_HANDLE); //номер потока для буфера консоли в системе,

string selectMap = "1 map"; //текущая карта

string Map[] = { //массив строк - карта игрового поля

"######################\n",

"| |\n",

"| |\n",

"| |\n",

"| |\n",

"| |\n",

"| |\n",

"| |\n",

"| |\n",

"| |\n",

"| |\n",

"| |\n",

"| |\n",

"| |\n",

"| |\n",

"| |\n",

"| |\n",

"| |\n",

"| |\n",

"| |\n",

"| |\n",

"######################\n"

};

string Logo = { //лого игры

"\n\n\t ZZZZZZZZZZZ ZZZZ ZZZZ ZZ ZZZZ ZZZZ ZZZZZZZZZZZZ\n"

"\tZZZZZZZZZZZZ ZZZZZ ZZZZ ZZZZ ZZZZ ZZZZ ZZZZZZZZZZZZ\n"

"\tZZZZ ZZZZZ ZZZZ ZZZZZZ ZZZZ ZZZZ ZZZZ \n"

"\tZZZZ ZZZZ Z ZZZZ ZZZ ZZZ ZZZZ ZZZZ ZZZZ \n"

"\tZZZZZZZZZZZ ZZZZ Z ZZZZ ZZZ ZZZ ZZZZZZZZ ZZZZZZZZZZZZ\n"

"\t ZZZZZZZZZZZ ZZZZ Z ZZZZ ZZZ ZZZ ZZZZZZZZ ZZZZZZZZZZZZ\n"

"\t ZZZZ ZZZZ Z ZZZZ ZZZ ZZZ ZZZZ ZZZZ ZZZZ \n"

"\t ZZZZ ZZZZ ZZZZZ ZZZZZZZZZZZZ ZZZZ ZZZZ ZZZZ \n"

"\tZZZZZZZZZZZZ ZZZZ ZZZZZ ZZ ZZ ZZZZ ZZZZ ZZZZZZZZZZZZ\n"

"\tZZZZZZZZZZZ ZZZZ ZZZZ ZZ ZZ ZZZZ ZZZZ ZZZZZZZZZZZZ\n\n\n",

};

string MenuText[] = { //массив строк - пункты меня

"New game\n",

"Choose difficulty\n",

"Choose map\n",

"Records\n",

"Authors\n",

"Exit\n"

};

string LevelsText[] = { //массив строк - меню выбора сложности

"easy\n",

"hard\n",

"insane\n"

};

string MapsText[] = { //массив строк - меню выбора карты

"1 map\n",

"2 map\n",

"3 map\n"

};

//БЛОК ОПИСАНИЯ ФУНКЦИЙ

void **ClearScreen**(){ //функция очищения консоли

DWORD n;

DWORD size;

COORD coord = {0};

CONSOLE\_SCREEN\_BUFFER\_INFO csbi;

HANDLE h = GetStdHandle(STD\_OUTPUT\_HANDLE);

GetConsoleScreenBufferInfo(h, &csbi);

size = csbi.dwSize.X \* csbi.dwSize.Y;

FillConsoleOutputCharacter(h, TEXT ( ' ' ), size, coord, &n);

GetConsoleScreenBufferInfo(h, &csbi);

FillConsoleOutputAttribute(h, csbi.wAttributes, size, coord, &n);

SetConsoleCursorPosition(h, coord);

}

void **loadRecords**(){ //функция загрузки рекордов

fin.open("records.txt"); //открываем текстовый файл с рекордами

string str; //инициализируем переменную типа string

getline(fin, str); //считываем строку с рекордом в легком режиме

easy\_r = stoi(str); //преобразуем в целочисленный вид

getline(fin, str);//считываем строку с рекордом в среднем режиме

hard\_r = stoi(str);//преобразуем в целочисленный вид

getline(fin, str);//считываем строку с рекордом в сложном режиме

insane\_r = stoi(str);//преобразуем в целочисленный вид

fin.close(); //закрываем файл

}

//БЛОК ОПИСАНИЯ КЛАССОВ

**class** **Food**{ //класс для описания объекта еды

**public**:

int x = 1 + (rand() % (WIDTH - 3)); //случайно задается координата x

int y = 1 + (rand() % (HEIGHT - 2)); //случайно задается координата y

char sign = '@'; //символ отрисовки еды на игровом поле

};

**class** **Snake**{//класс для описания объекта змеи

**public**:

int dir = UP; //направление движения змеи

char sign = 'o'; //символ отрисовки змеи на игровом поле

int x[MAX\_LEN\_SNAKE] = {0}; //массив координат по оси x

int y[MAX\_LEN\_SNAKE] = {0}; //массив координат по оси н

int len = 2; //длина змеи

};

Snake **checkKeys**(Snake snake){ //функция для считывания нажатий клавиатуры и управления змейкой

**if** (GetKeyState('W') & 0x8000){ //считываем нажатие клавиши W

**if**(snake.dir != DOWN) { // если это вохможно, то изменяем дивжение

snake.dir = UP;

}

}

**if** (GetKeyState('S') & 0x8000){ //считываем нажатие клавиши S

**if**(snake.dir != UP) { // если это вохможно, то изменяем дивжение

snake.dir = DOWN;

}

}

**if** (GetKeyState('A') & 0x8000){ //считываем нажатие клавиши A

**if**(snake.dir != RIGHT) { // если это вохможно, то изменяем дивжение

snake.dir = LEFT;

}

}

**if** (GetKeyState('D') & 0x8000){ //считываем нажатие клавиши D

**if**(snake.dir != LEFT) { // если это вохможно, то изменяем дивжение

snake.dir = RIGHT;

}

}

**return** snake; // возвращаем объект змеи

}

**class** **Game**{ //класс для описания работы игры

**public**:

Snake snake; //создание объекта змеи

void **gotoxy**(short x, short y){ //функция для переноса курсорв в консоли в точку (x,y)

SetConsoleCursorPosition(hStdOut, {x, y}); //переносим курсор стандартной функцией

}

void **showMenu**(){ //метод отрисовки меню

exxit = false;

**while** (!exxit){ //основной цикл

system("cls"); //очистка консоли

cout<<Logo; //вывод лого

**for** (int i = 0; i < MENUPOINTS; i++){ //цикл для вывода пунктов меню

**if**(currMenu == i){

cout<<"\t\t\t\t>>>\t"<<MenuText[i];

}

**else** {

cout<<"\t\t\t\t\t"<<MenuText[i];

}

}

cout<<"\n\t\t\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n\t\t\t\t\tYour map: "<<selectMap<<endl; //вывод текущих сложности и карты

cout<<"\t\t\t\t\t\Your Difficulty: "<<LevelsText[diff/2 - 2];

gotoxy(0, 13 + currMenu); //перемещение курсора в точку (0,13)

ch = \_getch(); // считываем нажатие клавишт

**if** (ch == 224) {

ch = \_getch();

}

**switch** (ch){

**case** 27: exxit = true; system("cls"); **break**; //обработка нажатие клавиши Escape

**case** 72: currMenu--; **break**; //обработка нажатия стрелки вверх

**case** 80: currMenu++; **break**; //обработка нажатия стрелки вниз

**case** 13: //обработка нажатия клавиши Enter, вызов соответствующих функций

system("cls");

**if** (currMenu == 0){

run();

}

**if** (currMenu == 1){

setDifficulty();

}

**if** (currMenu == 2){

setMap();

}

**if** (currMenu == 3){

getRecords();

}

**if** (currMenu == 4){

authors();

}

**if** (currMenu == 5){

exxit = true;

}

**break**;

} //обработка границ меню, чтобы курсор не вышел за границы

**if** (currMenu < 0) {

currMenu = 0;

}

**if** (currMenu > 5) {

currMenu = 5;

}

}

}

void **endGame**(){ //метод обработки конца игры

ofstream out; //подключаем модуль для записи в файл

exxit = false; //изменяем переменну окончания игры

**if** (diff == 4 && easy\_r < snake.len) { //проверяем, не побил ли игрок рекорд

easy\_r = snake.len;

}

**if** (diff == 6 && hard\_r < snake.len) {

hard\_r = snake.len;

}

**if** (diff == 8 && insane\_r < snake.len) {

insane\_r = snake.len;

}

out.open("records.txt"); //открываем файл records.txt

out<<easy\_r<<endl<<hard\_r<<endl<<insane\_r<<endl; //записываем рекорды в файл

**while** (!exxit){ //цикл для вывода информации

system("cls"); //очистка консоли

cout<<Logo; //вывод лого и основной информации

cout<<"\n\n\n\n\t\t\t\t\t GAME OVER";

cout<<"\n\n\n\t\t\t\t\tYour score: "<<snake.len;

cout<<"\n\t\t\t\t\tBest score: "<<bestScore;

ch = \_getch(); //считываем нажатия с клаивтуры

**if** (ch == 224) {

ch = \_getch();

}

**switch**(ch){ //закрываем окно, если нажаты Space или Escape

**case** 27: exxit = true;

**case** 13: exxit = true;

}

}

exxit = false;

}

void **setMap**(){ //метод выбора игровой карты

exxit = false; //изменяем переменну окончания игры

currMenu = 0; //по умолчанию выбираем 1 карту

**while** (!exxit){ //цикл для вывода информации

system("cls"); //очистка консоли

cout<<Logo; //вывод основной информации

cout<<"\t\t\t\t\tChoose Your Map\n";

**for** (int i = 0; i < size(MapsText); i++){

**if**(currMenu == i) {

cout<<"\t\t\t\t>>>\t"<<MapsText[i];

}

**else** {

cout<<"\t\t\t\t\t"<<MapsText[i];

}

}

cout<<"\t\t\t\t\tYour map: "<<selectMap;

gotoxy(0, currMenu); //перемещаем курсор к выбранному пункту

ch = \_getch(); //считываем нажатия с клаивтуры

**if** (ch == 224) {

ch = \_getch();

}

**switch** (ch){ //обработка нажатий клавиш и выбора соответствующей карты

**case** 27: exxit = true; **break**;

**case** 72: currMenu--; **break**;

**case** 80: currMenu++; **break**;

**case** 13:

exxit = true;

system("cls");

**if** (currMenu == 0){

selectMap = "1 map";

fin.close();

fin.open("map1.txt");

}

**if** (currMenu == 1){

selectMap = "2 map";

fin.close();

fin.open("map2.txt");

}

**if** (currMenu == 2){

selectMap = "3 map";

fin.close();

fin.open("map3.txt");

}

**break**;

} //обработка границ меню, курсор не должен выходить за эти границы

**if** (currMenu < 0) {

currMenu = 0;

}

**if** (currMenu > size(MapsText) - 1) {

currMenu = size(MapsText) - 1;

}

}

exxit = false;

currMenu = 0;

}

void **setDifficulty**(){ //метод выбора сложности

exxit = false; //изменяем переменну окончания игры

currMenu = 0; //по умолчанию выбираем легкую сложность

**while** (!exxit){ //цикл для вывода информации

system("cls"); //очистка консоли

cout<<Logo; //вывод информации

cout<<"\t\t\t\t\tChoose Your Difficulty\n";

**for** (int i = 0; i < size(LevelsText); i++){

**if** (currMenu == i) {

cout<<"\t\t\t\t>>>\t"<<LevelsText[i];

}

**else** {

cout<<"\t\t\t\t\t"<<LevelsText[i];

}

}

cout<<"\t\t\t\t\tYour Difficulty: "<<LevelsText[diff/2 - 2];

gotoxy(0, currMenu); //перемещаем курсор в нужную позицию

ch = \_getch(); //считываем нажатия с клаивтуры

**if** (ch == 224) {

ch = \_getch();

}

**switch** (ch){ //обработка нажатий клавиш клавиатуры, выбираем нужную сложность

**case** 27: exxit = true; **break**;

**case** 72: currMenu--; **break**;

**case** 80: currMenu++; **break**;

**case** 13:

exxit = true;

system("cls");

**if** (currMenu == 0){

diff = 4;

bestScore = easy\_r;

}

**if** (currMenu == 1){

diff = 6;

bestScore = hard\_r;

}

**if** (currMenu == 2){

diff = 8;

bestScore = insane\_r;

}

**break**;

} //обработка границ меню, курсор не должен выходить за эти границы

**if** (currMenu < 0) {

currMenu = 0;

}

**if** (currMenu > 2) {

currMenu = 2;

}

}

exxit = false; //изменяем переменну окончания игры

currMenu = 0; //присваиваем переменной значение по умолчанию

}

void **run**(){ //основной игровой метод

loadMap(); //загружаем игровую карту

snake.x[0] = WIDTH/4; //прасваиваем голове змеи начальное занчение

snake.y[0] = HEIGHT/4;

int time = clock(); //создаем переменную, чтобы отрисовывать картинку с каким-то периодом

coef = (snake.len - 2) /5; // переменная, для увеличивания скорости змеи

Food food; // создаем экзмепляр класса Food

**while** (isRunning){ //основной игровой цикл

gotoxy(0, 0); //переносим курсор в точку (0,0)

snake = checkKeys(snake); //проверяем нажатия для управления змейкой

**if** (snake.x[0] == food.x && snake.y[0] == food.y){ //проверяем, съела ли змея еду

food.x = 1 + (rand()%(WIDTH-3));

food.y = 1 + (rand()%(HEIGHT-2));

snake.len++;

}

**if** ((snake.len - 2) /5 > coef){ //проверяем, нужно ли увеличивать скорость

diff += 1;

coef += 1;

}

**if** (Map[snake.y[0]][snake.x[0]] != ' ') { //проверяем, не наткнулась ли змейка на препятствие

isRunning = false;

}

**if** ((clock() - time)\* diff / CLOCKS\_PER\_SEC >= 1){ //цикл отрисовки картинки

time = clock();

**if** (snake.dir == UP) { //двигаем змейку в соответствующем направлении

--snake.y[0];

}

**if** (snake.dir == DOWN) {

++snake.y[0];

}

**if** (snake.dir == LEFT) {

--snake.x[0];

}

**if** (snake.dir == RIGHT) {

++snake.x[0];

}

**if** (snake.x[0] == WIDTH-1) {//проверка выхода за края карты

snake.x[0] = 0;

}

**if** (snake.x[0] == -1) {

snake.x[0] = WIDTH-2;

}

**if** (snake.y[0] == HEIGHT) {

snake.y[0] = 0;

}

**if** (snake.y[0] == -1) {

snake.y[0] = HEIGHT-1;

}

**for** (int i = 2; i<snake.len; i++){ //проверка, не наткнулась ли змейка сама на себя

**if**(snake.x[0] == snake.x[i] && snake.y[0] == snake.y[i]) {

isRunning = false;

}

}

Map[food.y][food.x] = food.sign; //отрисовываем еду

**if** (Map[snake.y[0]][snake.x[0]] != ' ' && Map[snake.y[0]][snake.x[0]] != food.sign) { //проверяем, не наткнулась ли змейка на препятствие

isRunning = false;

}

**for** (int i = snake.len; i>= 0; i--){ //двигаем змею

snake.x[i+1] = snake.x[i];

snake.y[i+1] = snake.y[i];

}

cout<<Logo; //выводим логотип змеи

**for** (int i = 0; i < snake.len; i++) { //отрисовываем змейку

Map[snake.y[i]][snake.x[i]] = snake.sign;

}

Map[snake.y[0]][snake.x[0]] = '0'; //отрисовка карты

**for** (int i = 0; i < HEIGHT; i++) {

cout<<"\t\t\t\t"<<Map[i];

}

**for** (int i = 0; i < snake.len; i++) {

Map[snake.y[i]][snake.x[i]] = ' ';

}

cout<<"\t\t\t\t\tScore: "<<snake.len - 1<<endl; //отрисовка HUD

cout<<"\t\t\t\t\tBestScore: "<<bestScore<<endl;

}

}

endGame(); //выводим окно конца игры

isRunning = true; //задаем начальное значение основным переменным

fin.close();

fin.open("map1.txt");

snake.len = 2;

}

void **getRecords**(){ //метод для получения рекордов

exxit = false; //изменяем переменну закртытия окна

**while** (!exxit){ //цикл вывода информации

system("cls"); //очистка консоли

cout<<Logo; //вывод информации

cout<<"\t\t\t\t\tRecords";

cout<<"\n\t\t\t\t\tEasy mode: "<<easy\_r;

cout<<"\n\t\t\t\t\tHard mode: "<<hard\_r;

cout<<"\n\t\t\t\t\tInsane mode: "<<insane\_r;

ch = \_getch(); //считываем нажатие клавиатуры

**if** (ch == 224) {

ch = \_getch();

}

**switch** (ch){ //закрываем окно, если нажаты Space или Escape

**case** 27: exxit = true;

**case** 13: exxit = true;

}

}

exxit = false; //изменяем переменную закрытия окна

}

void **loadMap**(){ //метод загрузки карты

string str; //строковая переменная для текущей строчки карты

**for** (int i = 0; i < HEIGHT; i++){ //цикл для закгрузки карты

getline(fin, str); //считываем i-ую строку карты в переменную str

Map[i] = str + '\n'; //записываем строку в массив Map

}

}

void **authors**(){ //метод для отображения авторов игры

exxit = false; //изменяем переменну закртытия окна

**while** (!exxit){ //цикл вывода информации

system("cls"); //очистка консоли

cout<<Logo; //вывод информации

cout<<"\t\t\t\t\tGame Crated By\n\n";

cout<<"\t\t\t\t\t @flexyw1be\n\t\t\t\t\t @vladbily";

cout<<"\n\n\n\t\t\t\t\t 2024";

ch = \_getch(); //считываем нажатие клавиатуры

**if** (ch == 224) {

ch = \_getch();

}

**switch** (ch){ //закрываем окно, если нажаты Space или Escape

**case** 27: exxit = true;

**case** 13: exxit = true;

}

}

exxit = false; //изменяем переменную закрытия окна

}

};

int **main**(){ //основная функция программы

HWND hwnd;

char Title[1024];

GetConsoleTitle(Title, 1024);

hwnd = FindWindow(NULL, Title); //задаем размеры окна консоли

SetConsoleTextAttribute(hwnd,FOREGROUND\_GREEN | FOREGROUND\_INTENSITY );

MoveWindow(hwnd,0,0,800,700,TRUE);

HANDLE console\_color;

srand((unsigned int)time(NULL)); //инициализируем rand()

console\_color = GetStdHandle(STD\_OUTPUT\_HANDLE); //меняем цвет консоли и цвет текста

SetConsoleTextAttribute(console\_color, 6);

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

system("color 8E");

loadRecords(); //загружаем рекорды

fin.open("map1.txt"); //открываем карту по умолчанию

bestScore = easy\_r; //получаем рекорд в соответствующем режиме

Game game; //создаем экземпляр класса Game

game.showMenu(); //используем метод showMenu

**return** 0;

}