Министерство образования Российской Федерации

Пензенский государственный университет

Кафедра “Вычислительная техника”

**Пояснительная записка**

к курсовому проектированию

по курсу “Логика и основы алгоритмизации в инженерных задачах”

на тему “Разработка игрового агента для игры "Лабиринт”

Выполнил:

Студент группы 19ВВ2

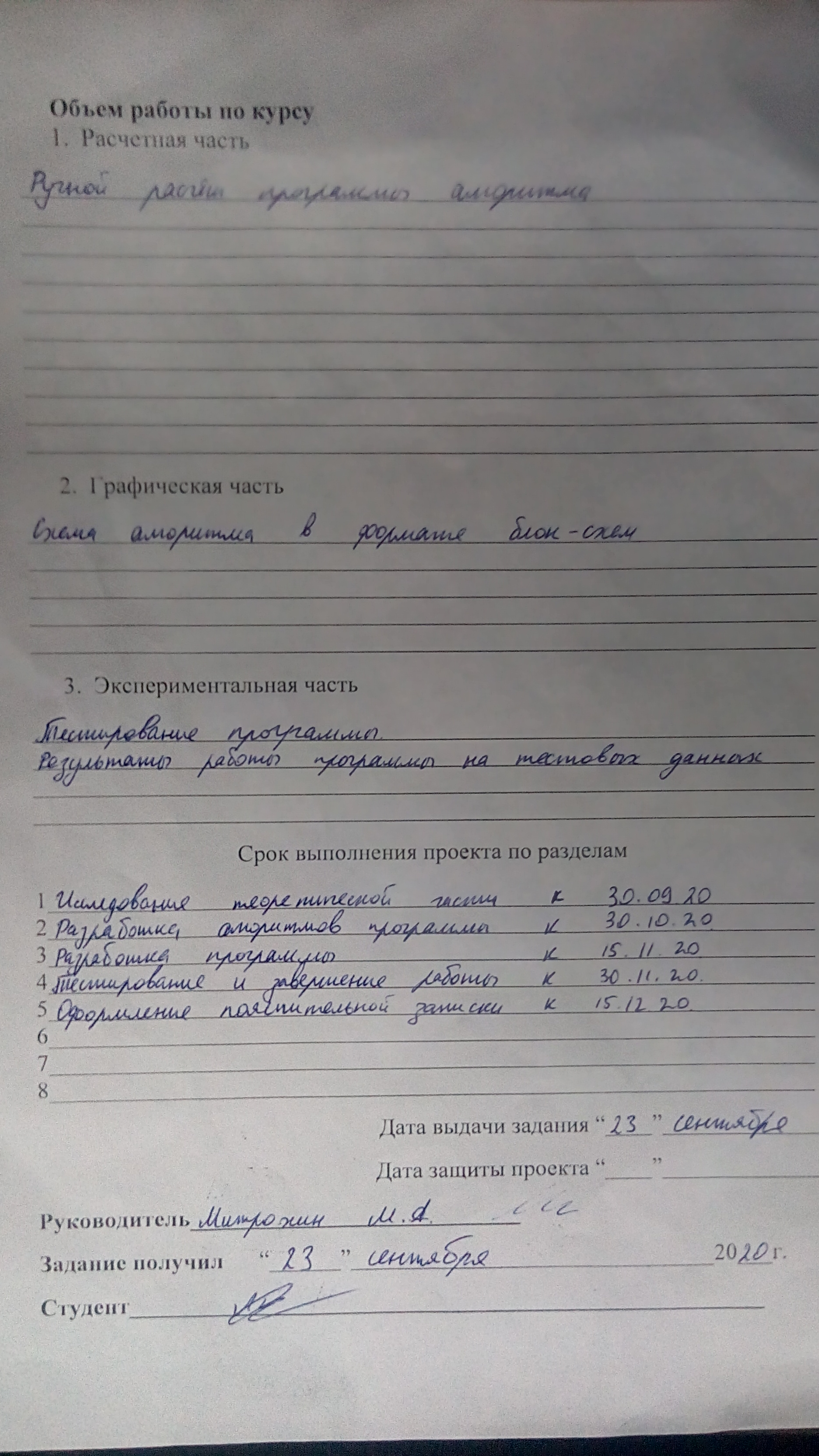
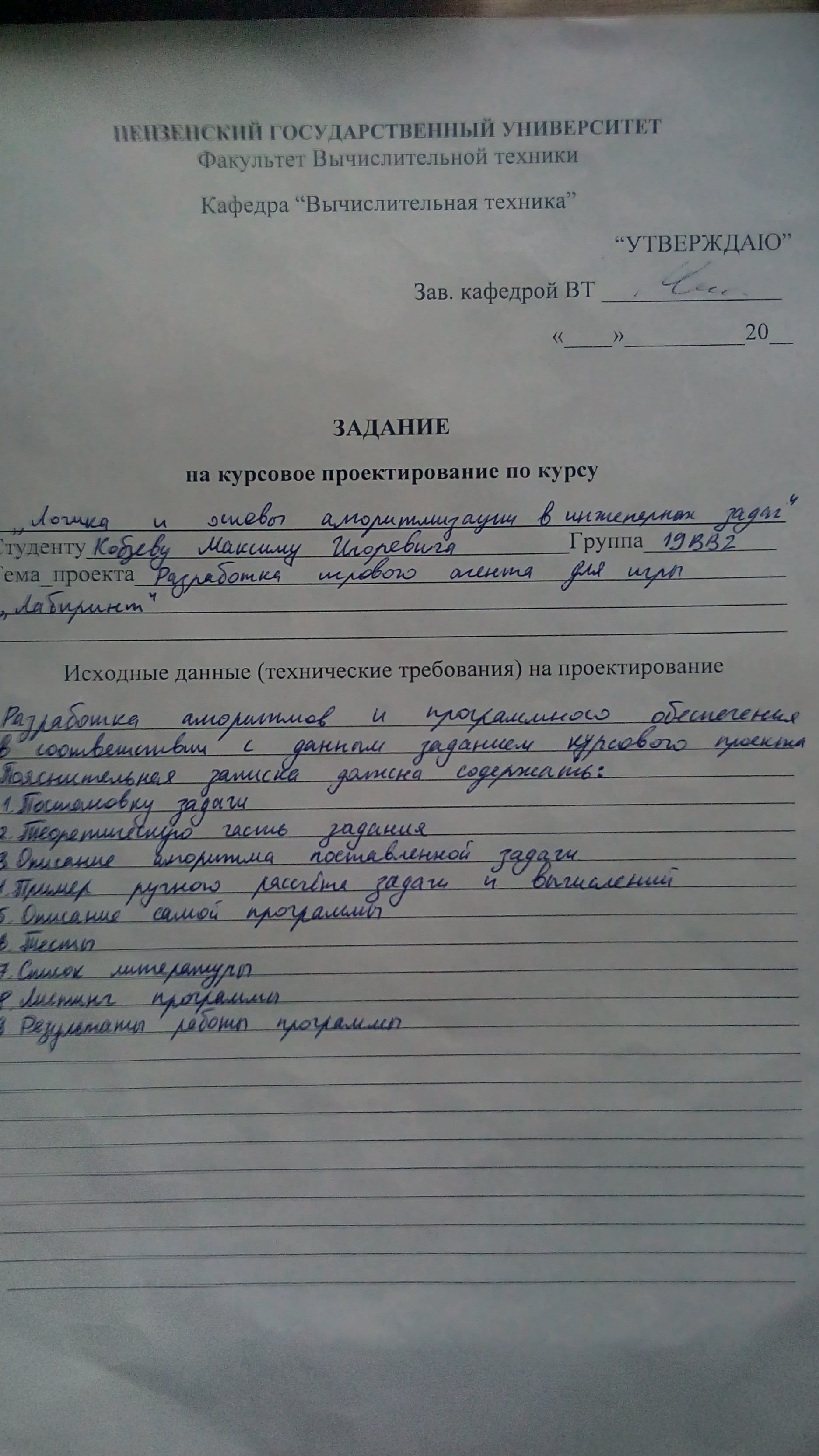
Кобзев Максим

Приняли:

Митрохин М.А.

Юрова О.В.

Пенза 2020.



**Реферат**

Отчет 33 страницы, 35 рисунков.

Цель исследования – разработка программы, реализующей игрового агента для игры “Лабиринт”.

В работе был создан алгоритм для агента, позволяющий ему находить поиск пути в лабиринте.

**Введение**

Данный алгоритм, при заранее сгенерированном лабиринте, позволяет найти выход из него для любого размера лабиринта. При последовательной реализации алгоритма, бот будет следовать инструкции, и проходить лабиринт в реальном времени. Также предусмотрены графические функции, способствующие выводу графики на экран при движении бота. В игре также присутствует игровое меню с заставкой, из которого можно выбрать разные режимы игры.

Операционная система – MS Windows. При разработке использовалась

среда MS Windows c Системой программирования – MS Visual Studio 2017.

Языки программирования – C/C++, а также некоторые функции из WinApi.

1. **Постановка задачи**

Необходимо разработать программу - Игра «Лабиринт», в частности алгоритм игрового агента для прохождения лабиринта. Размеры лабиринта вводить с клавиатуры. Соответствующий режим выбирать из пользовательского меню. Описать правила игры. Обеспечить соревновательный режим между ботом и пользователем, чтобы они состязались на время. Выводить результаты игры в файл. Необходимо также предусмотреть фактор пользователя, чтобы обрабатывались все исключения при вводе информации.

Задания выполняются в соответствии с вариантом №31.

1. **Теоретическая часть задания**

Мой выбранный алгоритм построения лабиринта – алгоритм “раскопки” прохода. Суть его состоит в том, что мы создаем двумерный массив, полностью заполняя его нулями (стенами). Затем помещаем в стартовую клетку “копателя”, который роет дорогу, переписывая стену на путь (единицу). Для создания косметического эффекта, копатель всегда дублирует свой ход, т.е проходит 2 клетки в заданном направлении. Чтобы избежать колец в лабиринте (когда существует несколько подходов в одном направлении), сделана проверка на то, не идёт ли через 2 клетки проход. Если же копатель зашел в тупик, то мы вытаскиваем его, перемещая на свободный проход. Параллельно этому запускается счетчик, который считает итерации цикла.

Для реализации игрового агента был создан алгоритм, который ищет путь в построенном лабиринте. В основу алгоритма легла матрица связности, которая показывает связь клеток или ее отсутствие, на основе которой был построен лабиринт. В ней некоторым числам соответствуют положения объекта, путь, стена, законченная ветка пути, текущая ветка пути. Бот умеет искать путь в соответствии со своим положением в лабиринте (существует несколько возможных вариантов движения бота).

1. **Описание алгоритма программы**

**3.1 Описание основных функций программы**

Для начала, была создана функция mainmech() для реализации многопоточности, чтобы бот и пользователь смогли действовать на одном экране. Соответственно, вызываются 2 потока с передачей функций go2() и runbot(). Далее делаем копию массива maze, с помощью функции copymaze(), для передачи его в поток с пользователем, чтобы не было пересечений объектов. После вызова функции go2(), пользователь может управлять объектом, чтобы пройти игру. После вызова функции runbot(), создаются переменные для графики и дальнейшей работы, а также инициализируется вектор WayBot, в котором хранятся узлы. Вначале запускается функция Find\_way(), необходимая для поиска количества путей из стартовой точки. После ее прохождения возвратится счетчик – число путей. Если он > 1, тогда записываем данную вершину в вектор. Далее запускается цикл, который будет функционировать, пока бот не дойдет до выхода.

Дальше идет псевдорекурсивный алгоритм для прохождения лабиринта. Запускается функция Findway() для поиска кол-ва путей из точки. Она ищет значения, равные ‘1’ – это еще не тронутый проход. Далее, относительно результата работы функции, можно получить 3 различных варианта:

1)Счетчик = 0 – В таком случае, бот ищет клетки со значением ‘4’ – это путь, который привел его в данную точку. Так как мы зашли в тупик, то пока не дойдем до точки с координатами вектора, мы выставляем пройденный назад путь в значение ‘5’ – это конечная (тупиковая) ветка, которую мы потом не берем в счет. После того, как мы дошли до точки с координатами вектора, мы проверяем ее на наличие путей, запуская функцию Findway(). Если счетчик = 0, тогда мы выбрасываем эту точку из вектора, так как мы знаем, что она ведет к тупику, иначе продолжаем цикл.

2)Счетчик = 1 – В этом случае мы знаем, что путь один, и запускается функция Way\_coord(), которая ищет этот путь и продвигает туда бота.

3)Счетчик > 1 – В случае такого результата, мы помещаем координаты точки в вектор, и вызываем функцию Way\_coord(), чтобы пойти в один из путей.

Также есть 2 графические функции: grph\_bot1() и grph\_bot2(), в которых реализована графика движения объекта.

После завершения цикла, снимается время и записывается значение в файл для просмотра результата.

Далее представлены блок-схемы некоторых функций:

**Блок схема для функции Find\_way():**



Рисунок 1 – Блок-схема для функции Find\_way()

**Блок схема для функции Way\_coord():**

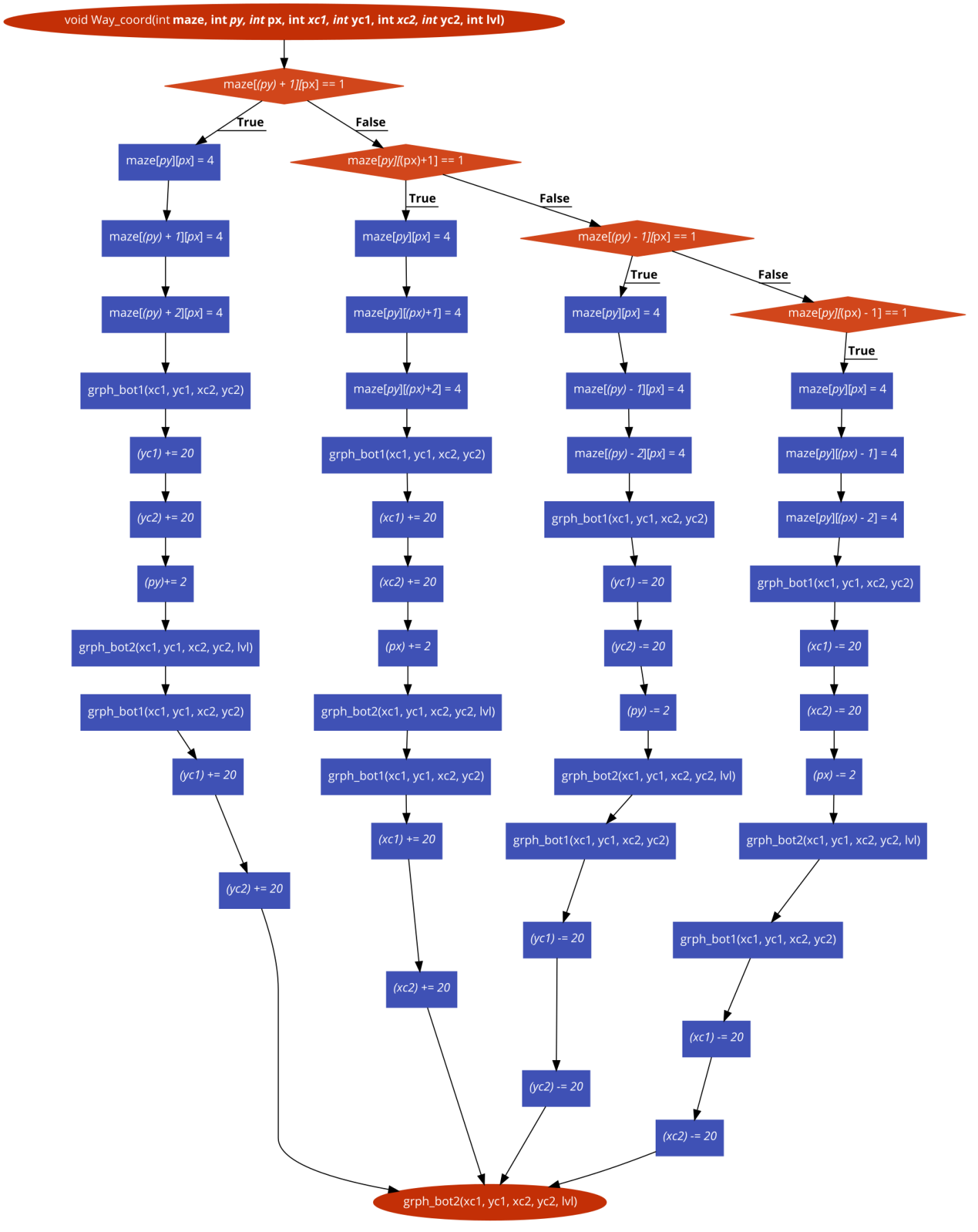
****

Рисунок 2 – Блок-схема для функции Way\_coord()

**Блок схема для функции Copymaze():**

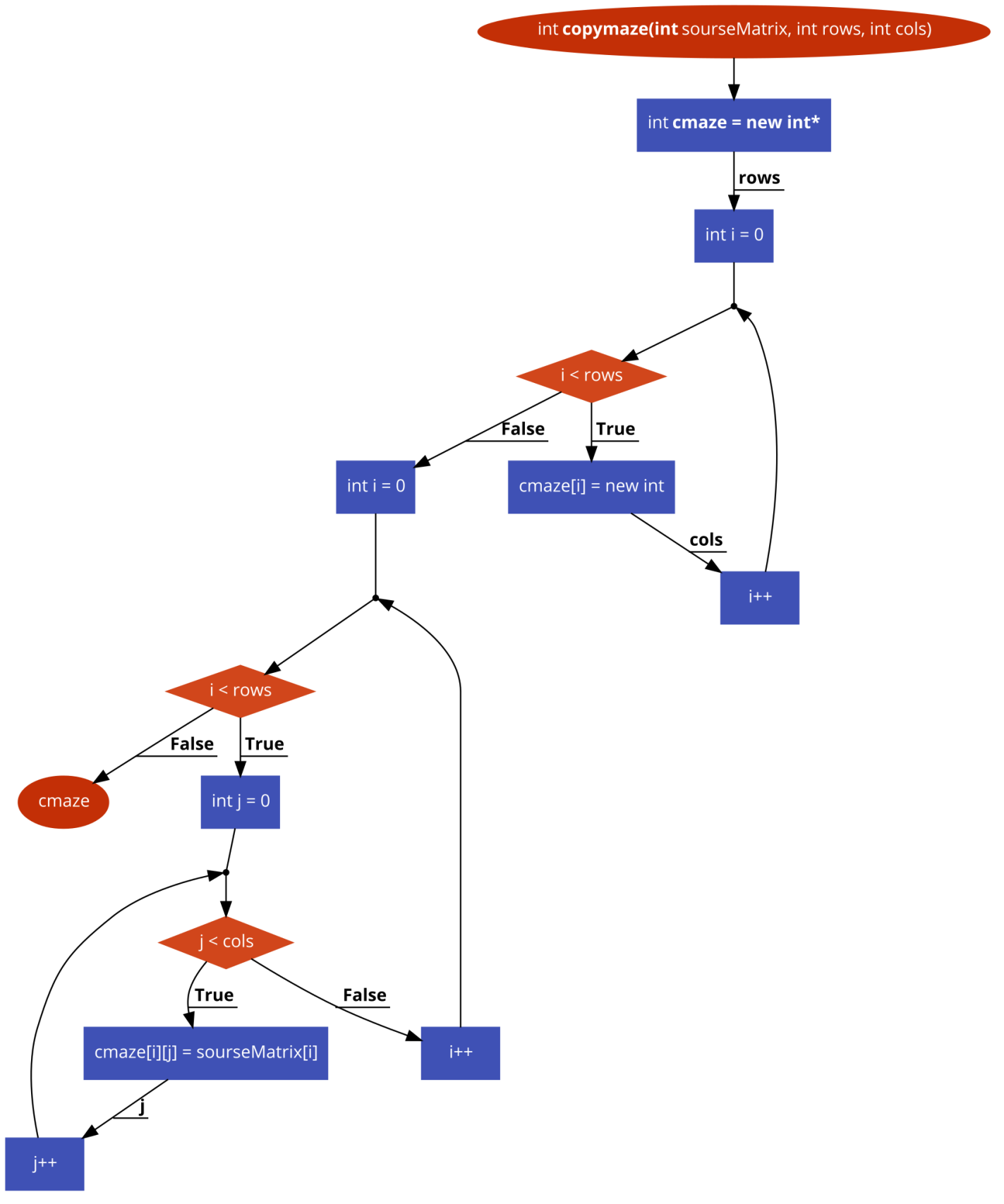
****

Рисунок 3 – Блок-схема для функции copy\_maze()

**Блок схема для функции Grph\_bot1():**

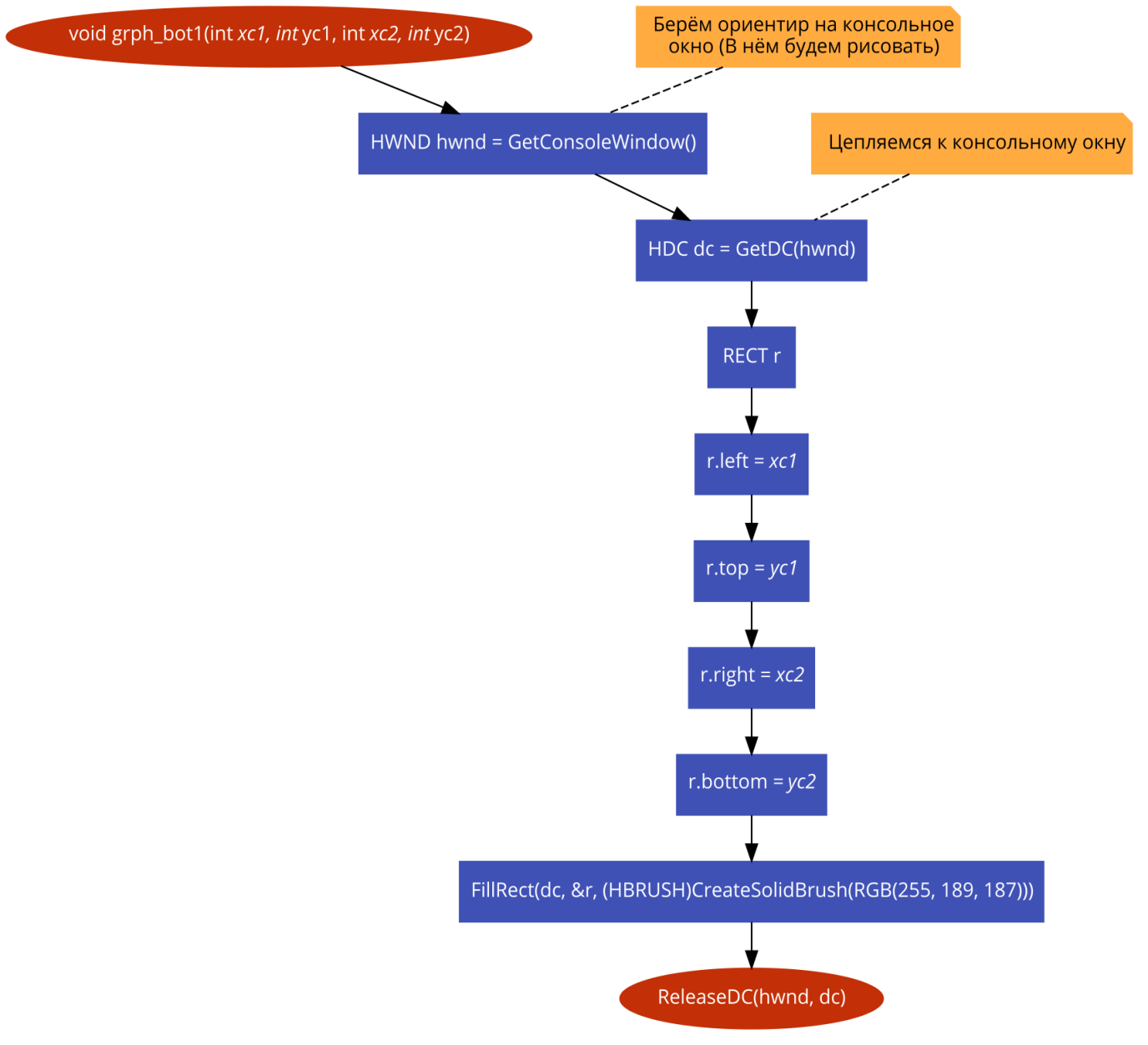
****

Рисунок 4 – Блок-схема для функции grph\_bot1()

**Блок схема для функции Grph\_bot2():**

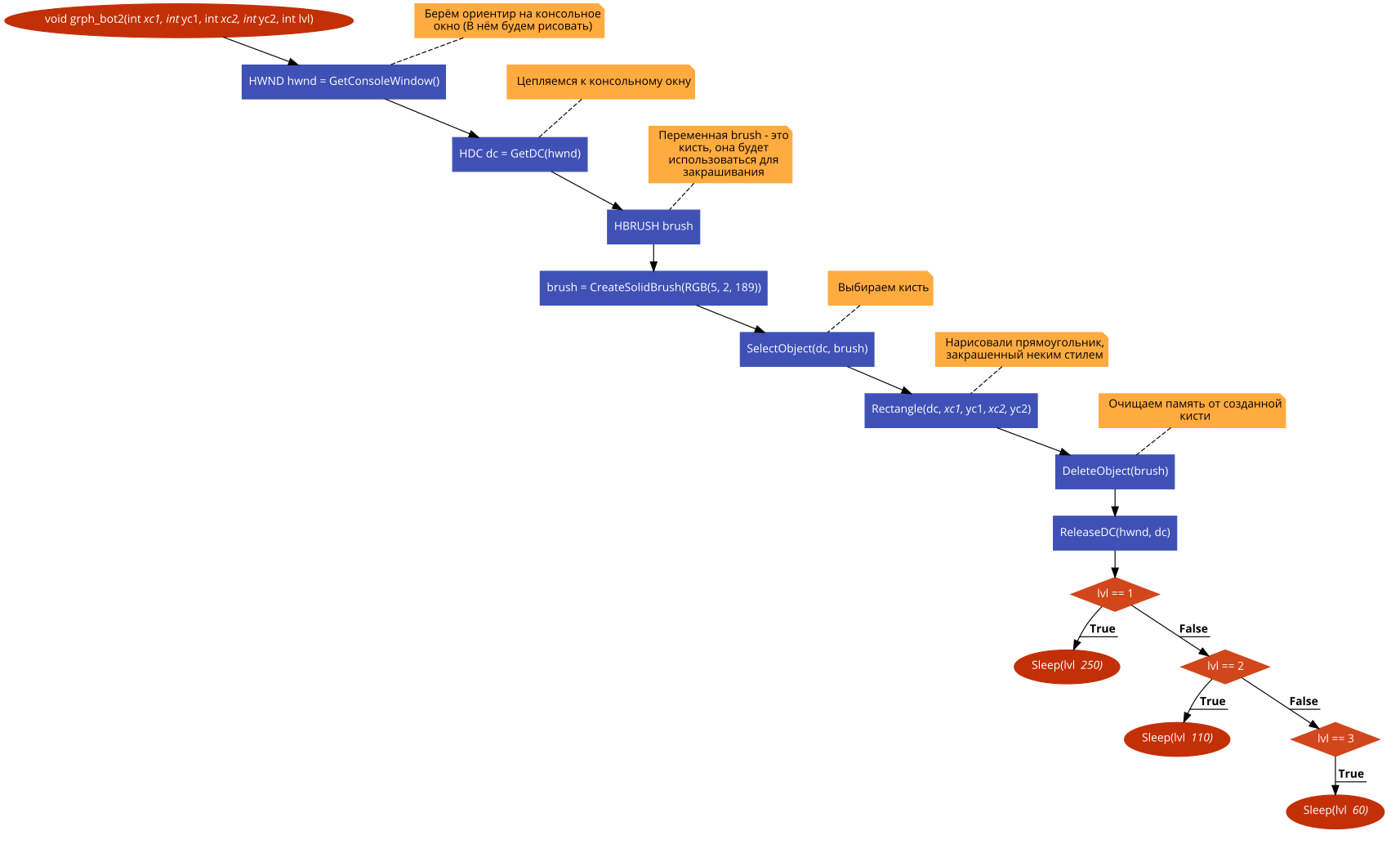
****

Рисунок 5 – Блок-схема для функции grph\_bot2()

**1 часть блок-схемы для функции Runbot():**

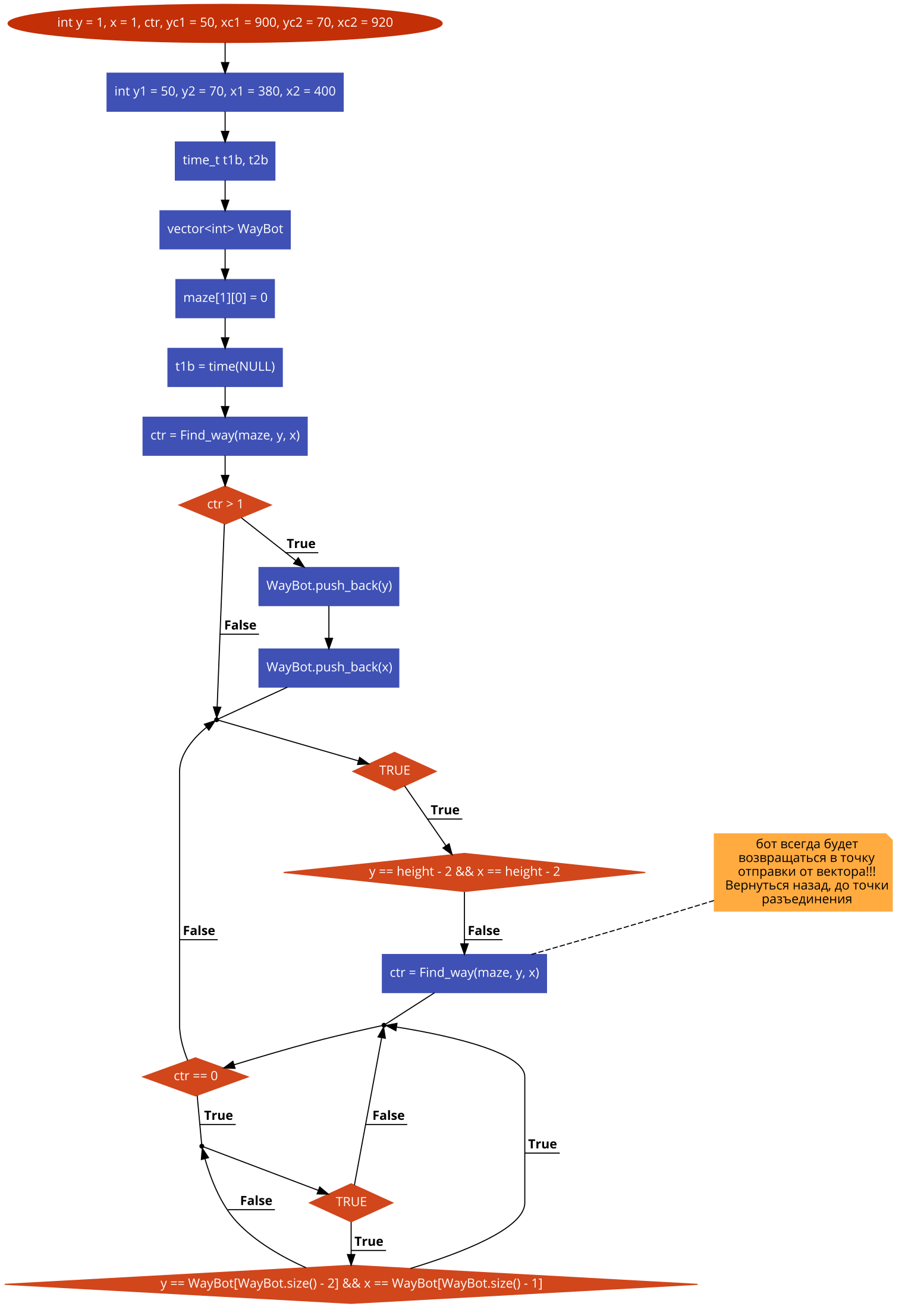
****

Рисунок 6 – Блок-схема для функции runbot() (1)

**2 часть блок-схемы для функции Runbot():**

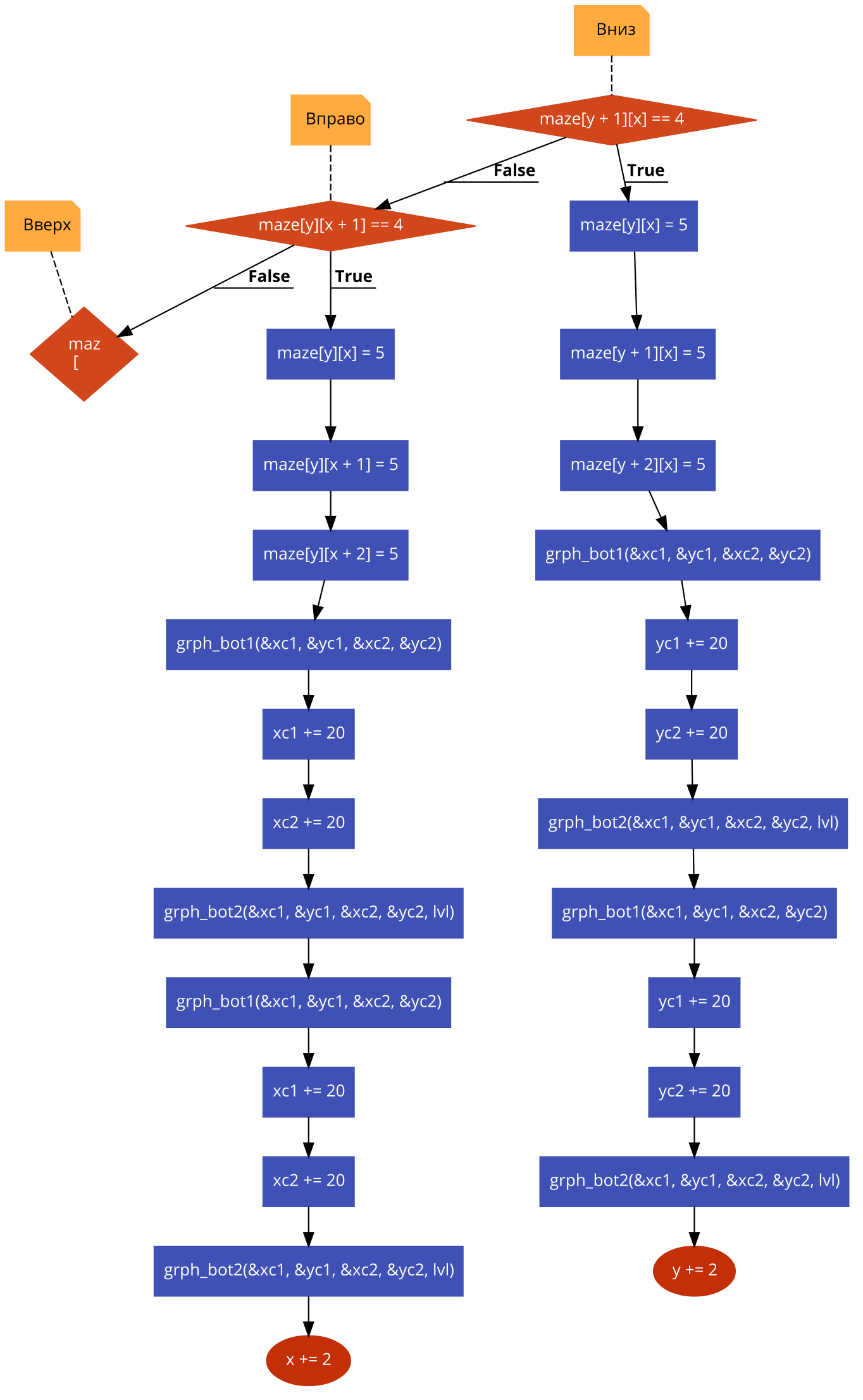
****

Рисунок 7 – Блок-схема для функции runbot() (2)

**3 часть блок-схемы для функции Runbot():**

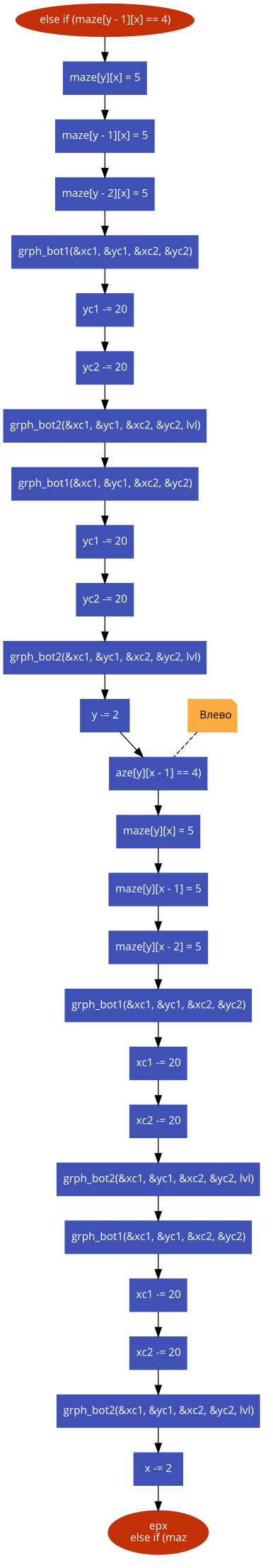
****

Рисунок 8 – Блок-схема для функции runbot() (3)

**4 часть блок-схемы для функции Runbot():**

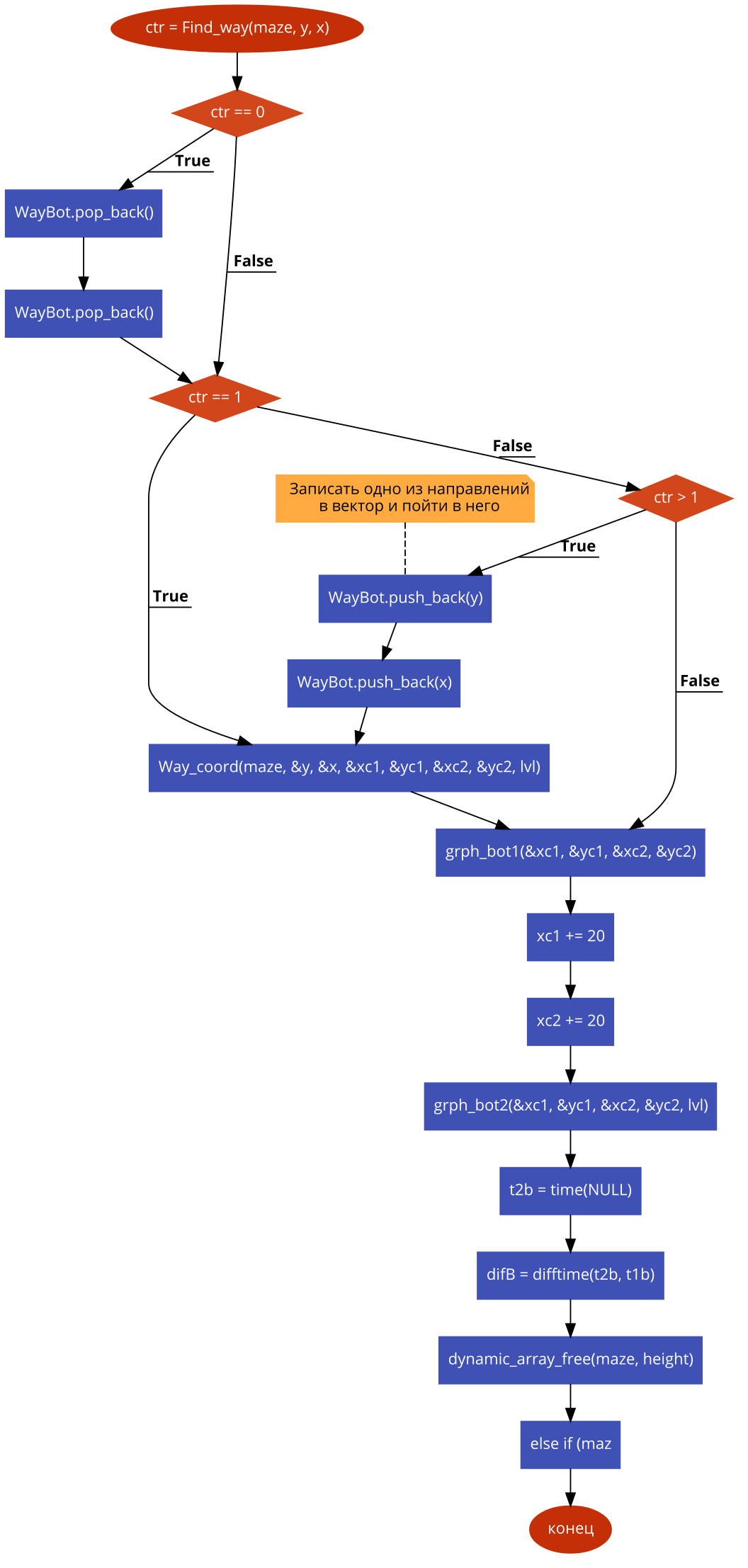
****

Рисунок 9 – Блок-схема для функции runbot() (4)

* 1. **Описание работы вспомогательных функций.**

Главный файл с курсовым проектом – Курсовой проект (лабиринт) содержит в себе главную функцию main, а также некоторые вспомогательные функции для корректной работы игры, функцию Blinking(), убирающую мигание курсора, функцию sizecolor() для реализации цвета фона консоли, цвета символов, а также установления её основного размера. Из этого файла идёт вызов функции пользовательского меню (menu()).

Далее, для демонстрации пользовательского интерфейса, в качестве которого выступают файлы Menu.h, Menu.cpp , реализована механика выбора пунктов меню, а также нажатия на определенный пункт. Перед реализацией самой механики, вызывается функция graphicsformenu(), находящаяся в файлах для графики Graphics.cpp и Graphics.h, которая рисует на поле заставку пользовательского меню.

Непосредственно из самого меню можно переместиться на игровое поле, нажав на соответствующий пункт, где начнут свою реализацию файлы Mechanics.cpp и Mechanics.h. После динамического выделения памяти dynamic\_array() и генерации самого лабиринта mazegame(), вместе с функции deadend(), которая обозначает тупик в дальнейшем прорубании хода, она вытаскивает копателя на свободный путь, вызовется функция dynamic\_array\_print(), выводящая на экран графику игрового поля. После вызова функции go(), пользователь может уже управлять объектом, тем самым проходя лабиринт.

В конце прохождения или в случае принудительного выхода в меню, вызовется функция dynamic\_array\_free(), которая освободит память от созданного динамического массива. Если пользователь выберет режим игры “со временем”, то результаты своего прохождения он может найти в файле Результаты.txt, находящимся в папке с игрой. Также в меню предоставлен пункт – Правила, после нажатия на который, вызовется функция rules() с графическим представлением правил игры. Последним пунктом меню является “Выход”, нажимая на который, пользователь покидает игру.

1. **Руководство пользователя**

Игра лабиринт – развлекательная 2-D игра с элементами

пользовательского интерфейса.

При загрузке программы откроется пользовательское меню, пункты

которого можно выбирать с помощью цифр 1,2,3,4 или, нажимая на стрелочки на клавиатуре.

Вид пользовательского меню показан на рисунке 2.

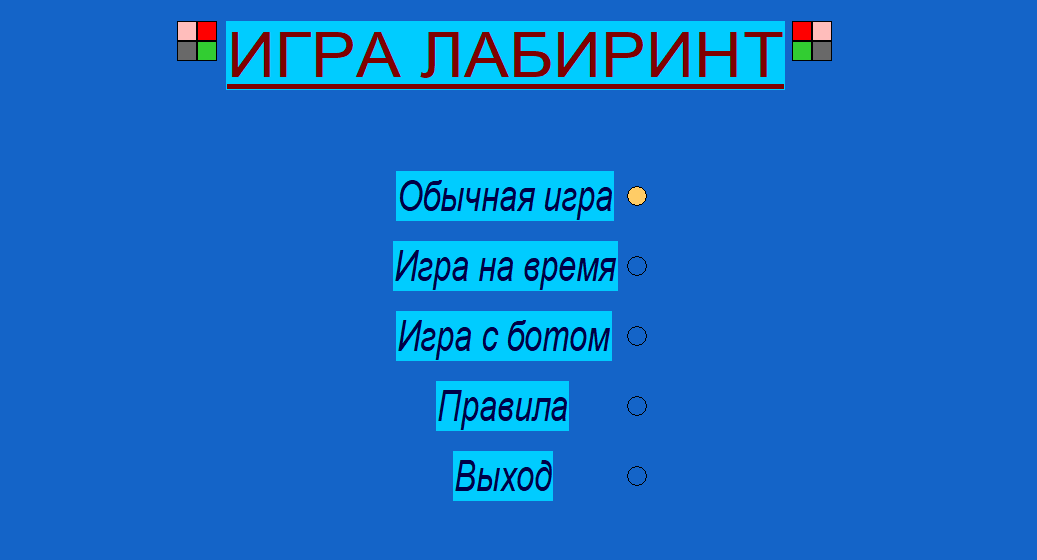


Рисунок 10 - Пользовательское меню

В игре есть 3 режима: 1)Обычная игра – режим, в котором необходимо

ввести размеры проходимого лабиринта и начать прохождение, попутно собирая монетки;

Вид поля при игре в обычный режим показан на рисунке 11.

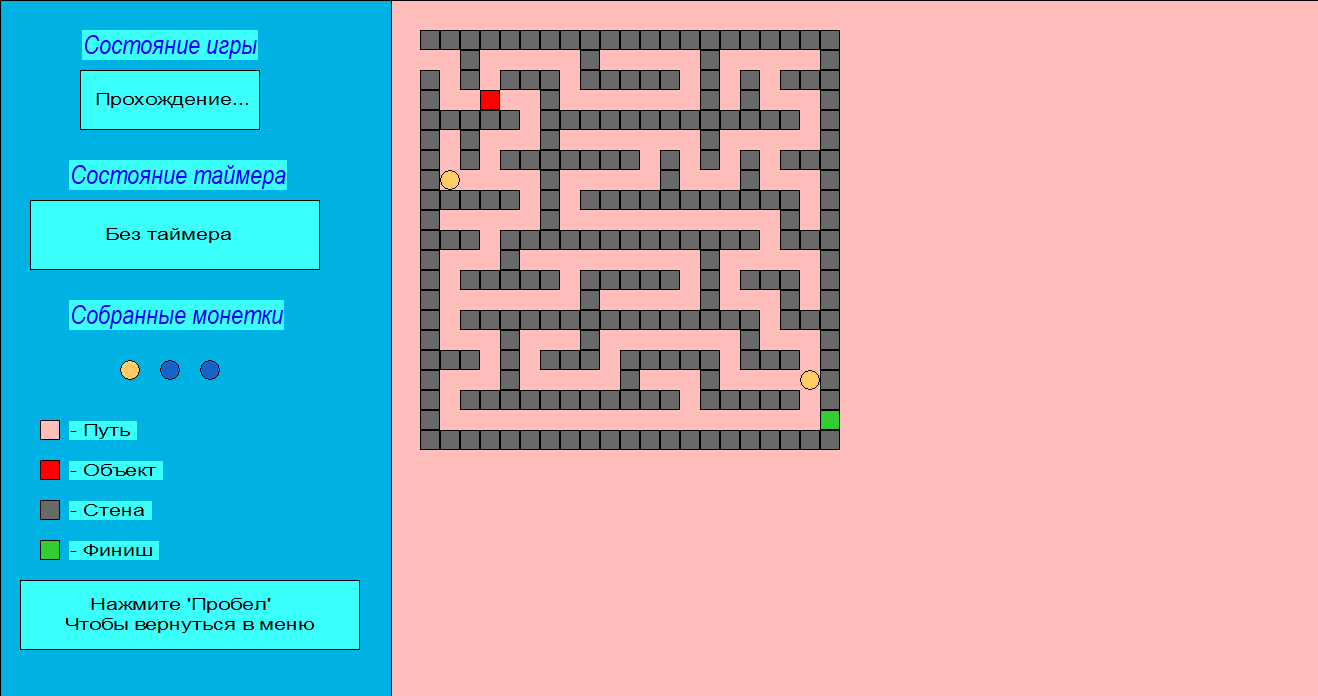


Рисунок 11 – Обычная игра

Если нужно незамедлительно вернуться в меню, то нужно нажать

клавишу ‘Пробел’.

2)Игра на время – режим, в котором уже нужно заботиться о времени,

потраченном на прохождение, ведь результаты уже пойдут в файл Результаты.txt.

Вид поля при игре в обычный режим показан на рисунке 12.

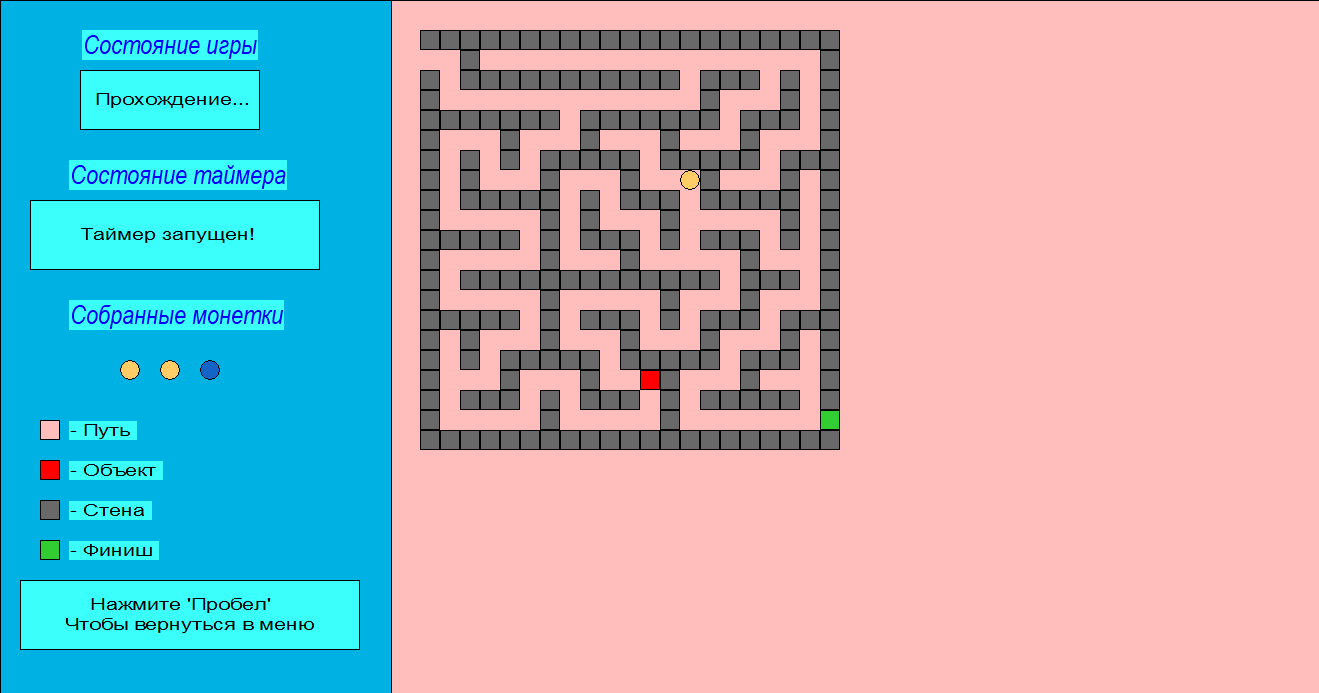
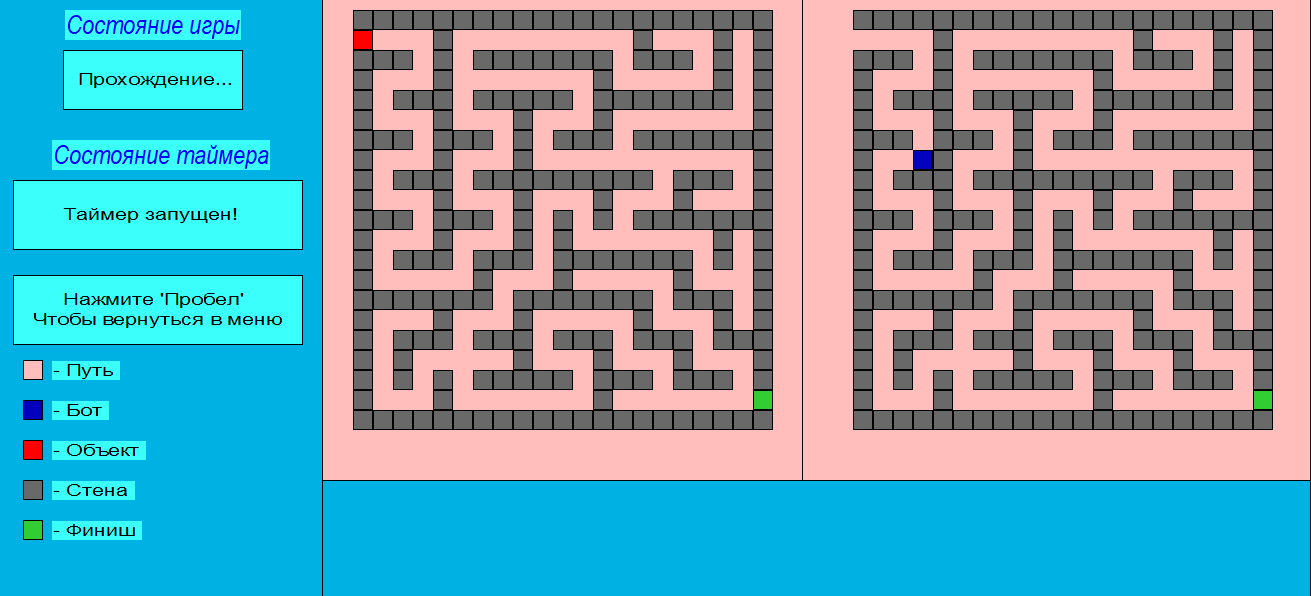


Рисунок 12 - Игра на время

3) Игра против бота – режим, в котором игроку придется соревноваться

с ботом по скорости прохождения лабиринта.

Вид поля при игре в режим игры с ботом показан на рисунке 13.

Рисунок 13 – Игра против бота

Файл с результатами – рисунок 14.



Рисунок 14 - Файл с результатами

Конечно, перед началом освоения игры, нужно ознакомиться с

правилами, выбрав 4 пункт в меню.

Правила игры показаны на рисунке 15.

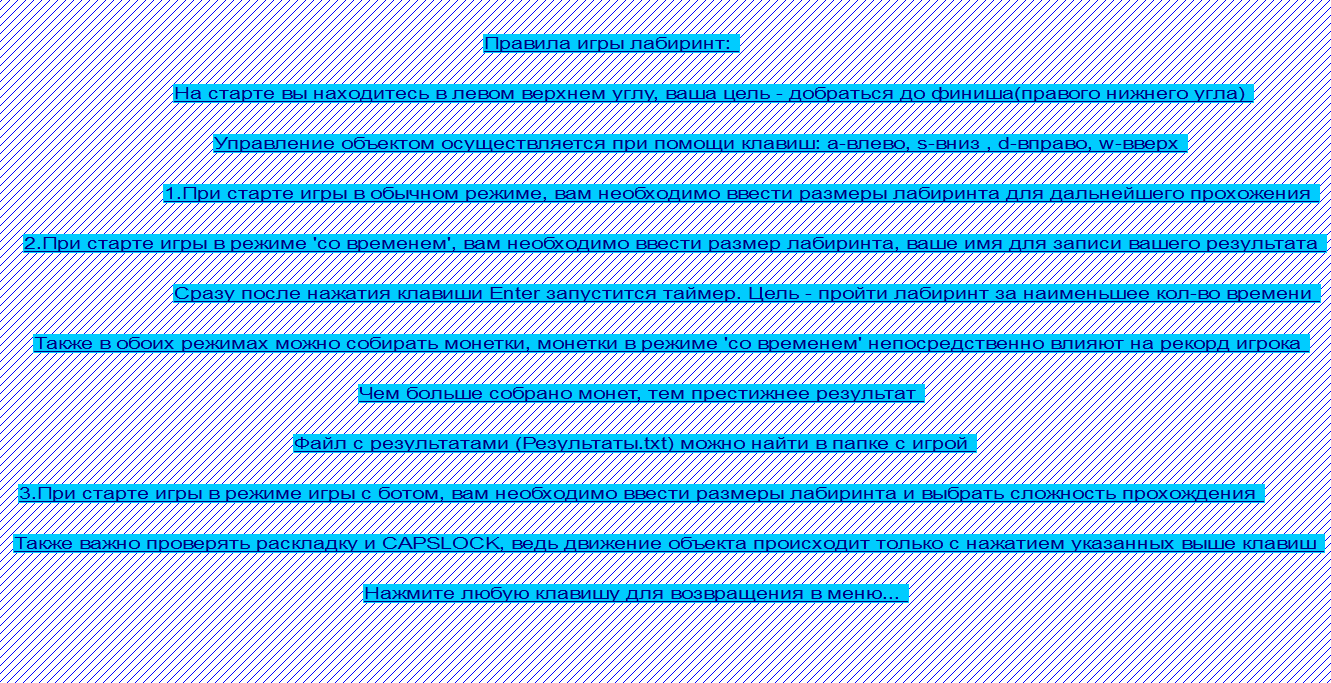


Рисунок 15 - Правила игры

После прочтения, вернуться в меню можно, нажав любую кнопку.

Выйти из меню, а соответственно из игры, можно с помощью пункта “Выход”, или нажав цифру 5.

Поле выход показано на рисунке 16.

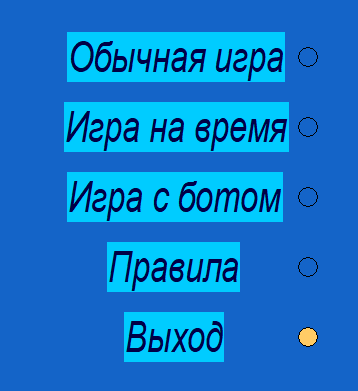


Рисунок 16 - Поле "Выход"

1. **Тестирование**

Таблица 1 – План тестирования (порядок действий)

|  |  |  |  |
| --- | --- | --- | --- |
| Описание теста | Порядок действий | Ожидаемый результат | Полученный результат |
| Запуск программы | Запускаем .exe файл | Автоматическое расширение консоли, с выведенным цветом фона и текста. | Верно |
| Проверка меню | Нажимаем последовательно на клавиши 1,2,3,4 и также выбираем пункты меню | При нажатии на клавиши 1,2,3,4 пользователь переходит на соответствующие позиции, также с выбором пунктов. | Верно |
| Проверка введенных значений | Вводим целенаправленно неверные значения для проверки | Сообщение о вводе будет выводиться снова, пока не введены подходящие значения. | Верно |
| Проверка механики сбора монеток | Дойди до монетки в режиме игры и собрать ее | Монетка должна при собирании исчезать с поля и перемещаться в поле слева. | Верно |
| Окончание игры | Полностью пройти лабиринт в 2 разных режимах. | Должно быть выведено: “Игра пройдена”, если же режим “Со временем”, то еще и “Таймер остановлен”. Также должно появиться окно с возвратом в пользовательское меню. | Верно |
| Проверка введенных значений в режиме игры с ботом | Вводим целенаправленно неверные значения для проверки | Сообщение о вводе будет выводиться снова, пока не введены подходящие значения. | Верно |
| Окончание игры с ботом | Сначала сделать так, чтобы победил бот, затем человек | При победе бота или человека должно вывестись соответствующее окно | Верно |

Тестирование проводилось в рабочем порядке, в процессе разработки,

после завершения написания программы. В ходе тестирования было выявлено и исправлено множество проблем, связанных с вводом данных, изменением дизайна выводимых данных, алгоритмом программы, взаимодействием функций.

1)Тестирование на автоматическое расширение консоли и введение

нового фона и цвета текста.



Рисунок 17 - Расширенная консоль с неким форматом

2)Тестирование пользовательского меню при нажатии кнопок или при

выборе пункта меню.

1. Нажатие на цифру 3 (пример).

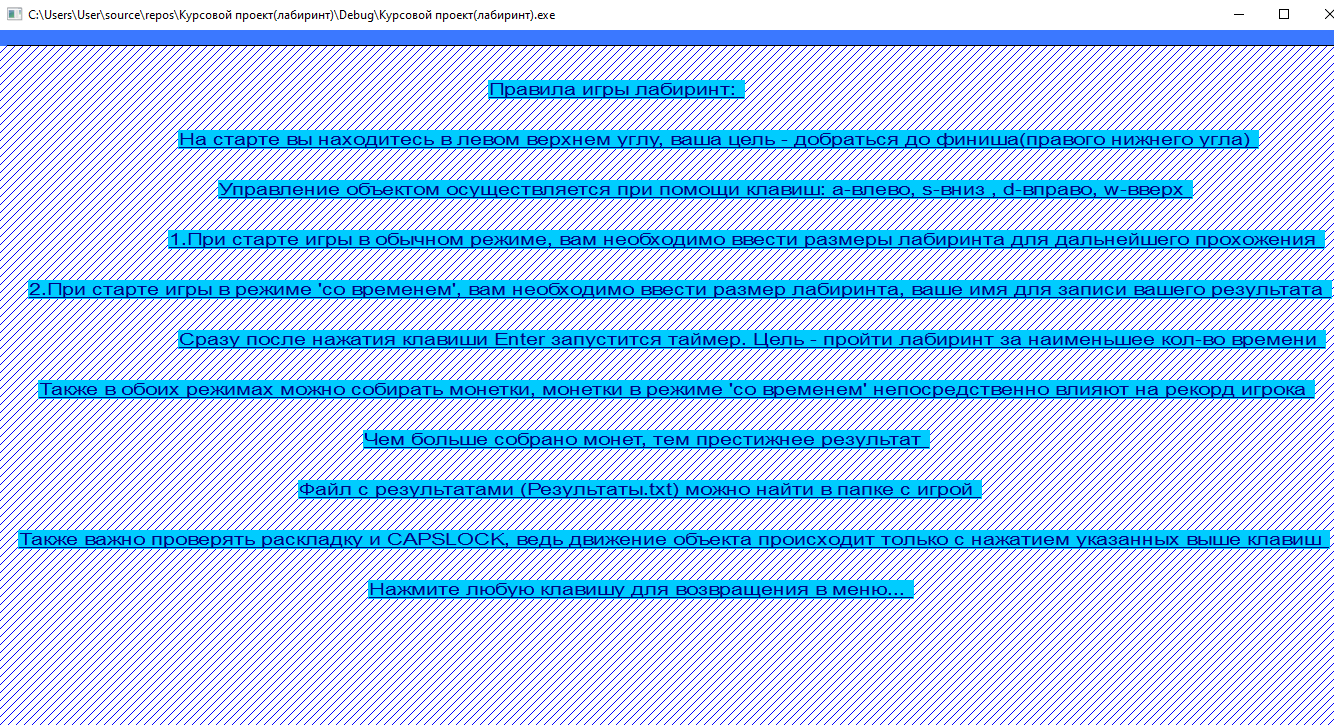


Рисунок 18 - Нажатие на цифру 3, вывод правил игры

2. Выбор пункта с помощью стрелочек в меню + Enter.

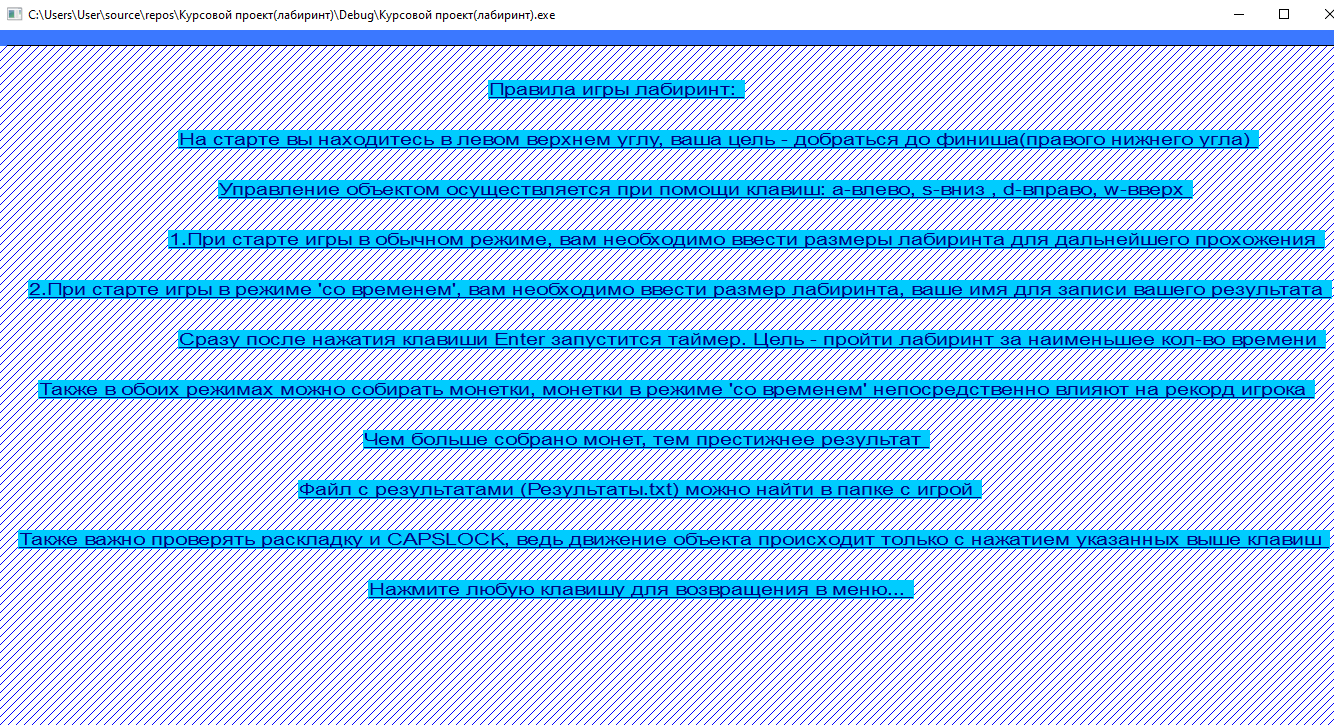


Рисунок 19 - Выбор пункта "Правила" из меню

3)Тестирование на ввод неверного значения для длины/ширины.

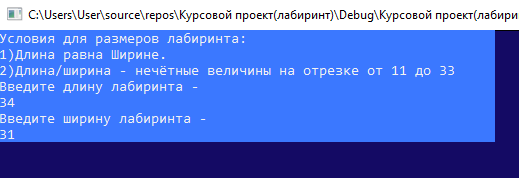


Рисунок 20 - Ввод неверных значений для лабиринта

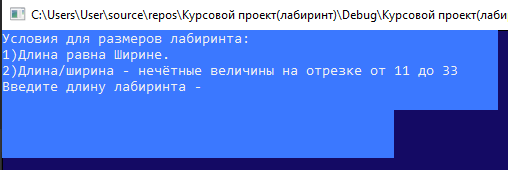


Рисунок 21 - Обнуление введенных данных для ввода новых

4)Тестирование на сбор монеток в игре.

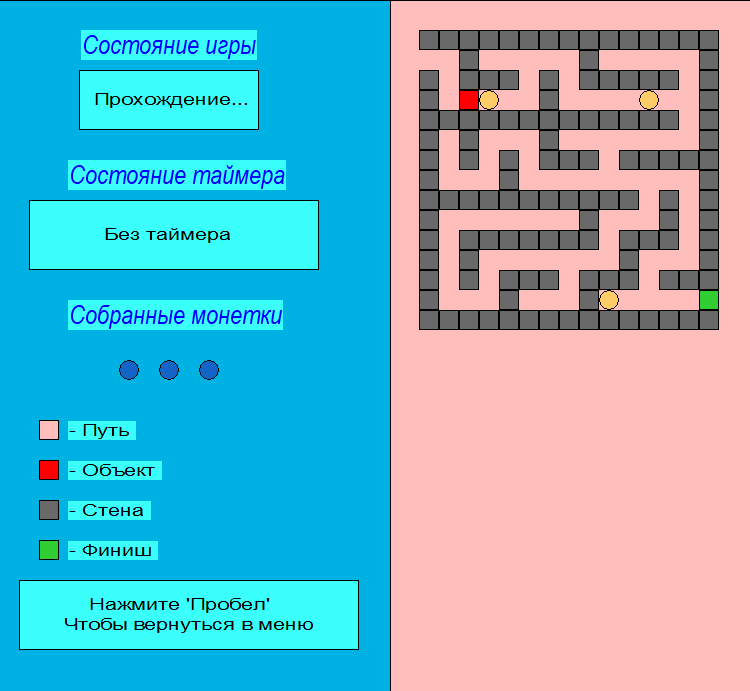


Рисунок 22 - Объект перед монеткой

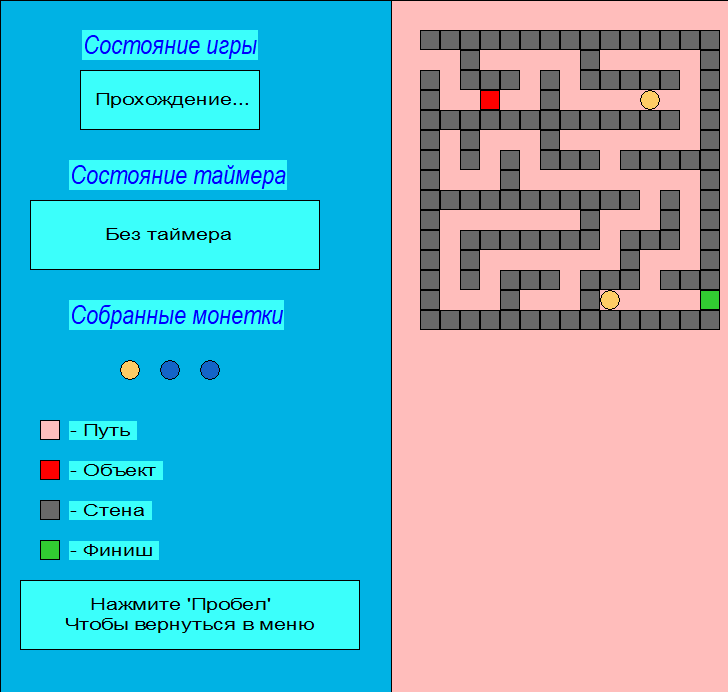


Рисунок 23 - Объект собрал монетку

5)Тестирование на окончание игры и вывод сообщений.



Рисунок 24 - Объект перед финишем

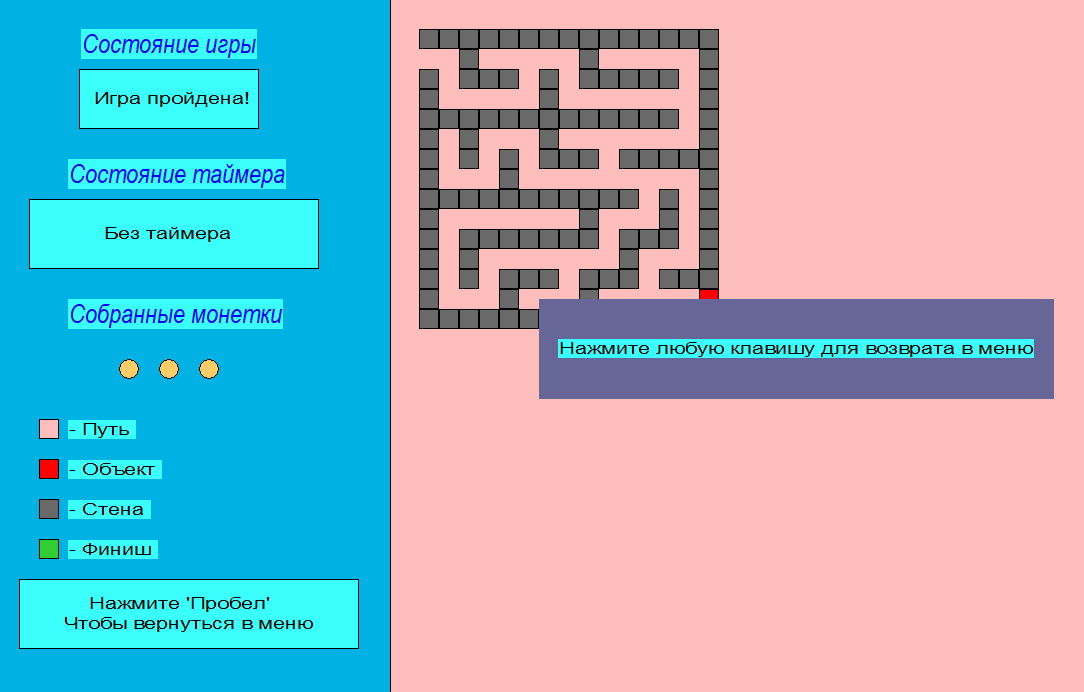


Рисунок 25 - Игра пройдена

6)Тестирование на работоспособность бота.

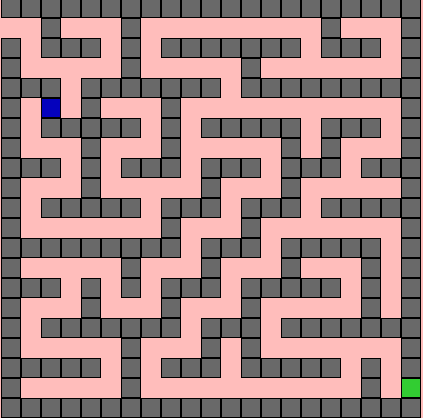


Рисунок 27 – Бот находится в точке

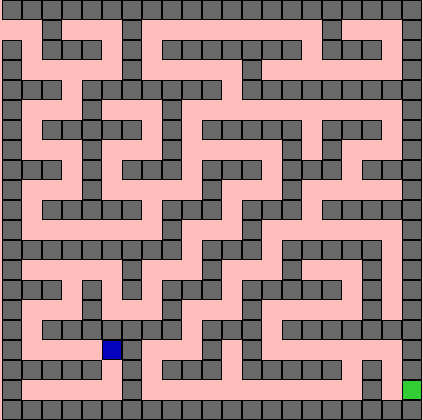


Рисунок 28 – Бот с течением времени поменял свое положение

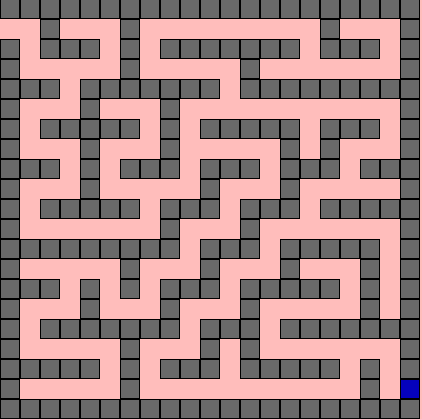


Рисунок 29 – Бот пришел к финишу

7)Тестирование на введенные значения при игре с ботом.

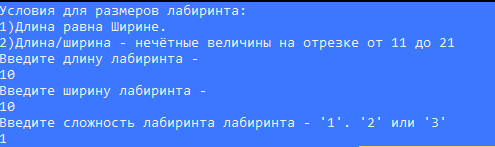


Рисунок 30 – Введение неверных данных

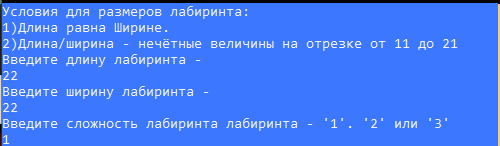


Рисунок 31 – Введение неверных данных

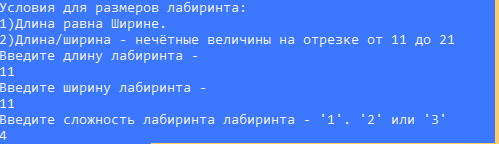


Рисунок 32 – Введение неверных данных

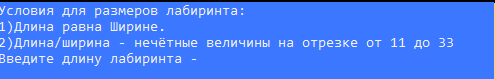


Рисунок 33 – Значения должны быть введены еще раз

7) Проверка на завершение игры в режиме игры с ботом.

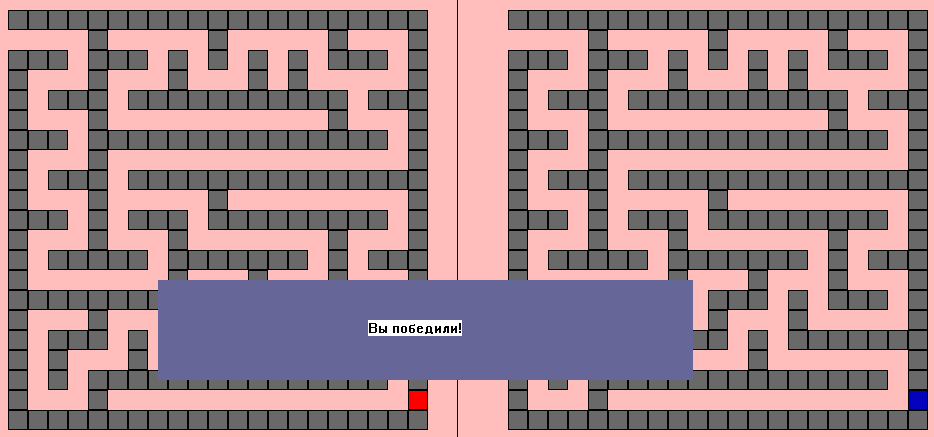


Рисунок 34 – Победа человека

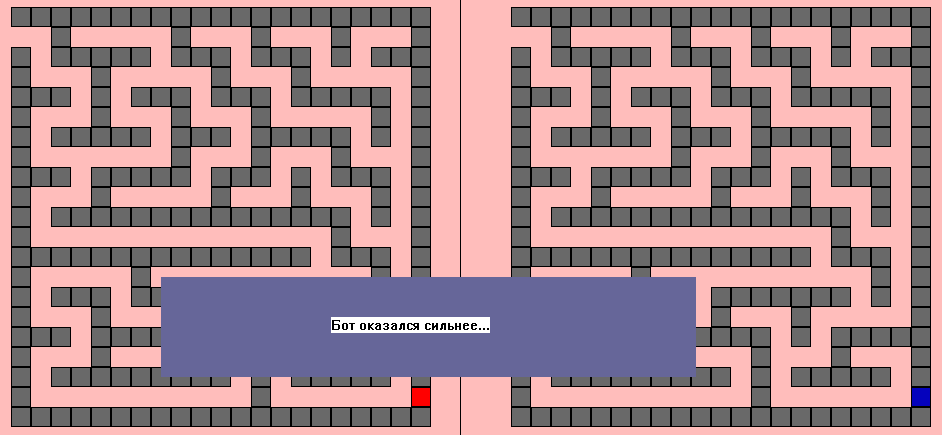


Рисунок 35 – Победа бота

В результате тестирования программы было выявлено, что программа функционирует успешно.

**Заключение**

При выполнении данной курсовой работы были получены навыки разработки многомодульных программ. Были освоено приемы создания графического интерфейса, с помощью WinApi. Изучены функции работы с консолью, способы обработки событий с клавиатуры и мыши. Усвоен механизм реализации пользовательского меню. Изучены функции работы с файлом. Получены базовые навыки программирования на языке Си и Си++. Получены навыки отладки и тестирования программ. Изучены основы алгоритмизации и теории графов.

В рамках данной курсовой была написана игра – “Лабиринт”. Данная программа не требует большого количества системных ресурсов и обеспечивает комфортную игру в нескольких режимах.

**Список используемых источников**

1. Керниган Б. Ритчи Д. Язык программирования С. 1985 г.
2. MSDN.
3. В. Г. Давыдов. Программирование и основы алгоритмизации. 2003 г.

**Приложение A**

**Листинг программы**

Файл: Лабиринт Created by Maksim Kobzev.cpp

#include "Graphics.h"

#include "Mechanics.h"

#include "Menu.h"

//Нейтрализация мигания

void Blinking(int off = 99) {

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

CONSOLE\_CURSOR\_INFO cci;

cci.dwSize = off;

cci.bVisible = false;

SetConsoleCursorInfo(hCon, &cci);

}

//Выставления цвета и размеров консоли

void sizecolor() {

system("color 9F");

HWND hwnd = GetConsoleWindow();

MoveWindow(hwnd, 0, 0, 1370, 740, TRUE);

HMENU hm;

hm = GetSystemMenu(hwnd, FALSE);

RemoveMenu(hm, SC\_SIZE, MF\_BYCOMMAND | MF\_REMOVE);

RemoveMenu(hm, SC\_MAXIMIZE, MF\_BYCOMMAND | MF\_REMOVE);

DrawMenuBar(hwnd);

}

//main функция

int main() {

setlocale(LC\_ALL, "Rus");

srand((unsigned)time(NULL));

sizecolor();

Blinking();

printf("Навигацию по меню осуществляется с помощью стрелок на клавиатуре, нажав Enter, или цифр: 1,2,3,4. Нажмите любую клавишу для перехода в меню...\n");

\_getch();

menu();

return 0;

}

Файл: Graphics.h

#include<stdio.h>

#include<stdlib.h>

#include<windows.h>

#include<iostream>

const int wall = 0, pass = 1, digger = 2, coin = 3;

void menu();

void rules();//Графика правил

void graphicsofmenu();//Графика меню

void dynamic\_array\_print(int\*\* maze, int height, int width, bool stopwatch);//Графика первых двух режимов лабиринта

void dynamic\_array\_bot(int\*\* maze, int height, int width, bool stopwatch);

Файл: Graphics.cpp

#include "Graphics.h"

#include "Mechanics.h"

#include "Menu.h"

void rules() {

HWND hwnd = GetConsoleWindow(); //Берём ориентир на консольное окно (В нём будем рисовать)

HDC dc = GetDC(hwnd); //Цепляемся к консольному окну

RECT window = {}; //Переменная window будет использована для получения ширины и высоты окна

HBRUSH brush; //Переменная brush - это кисть, она будет использоваться для закрашивания

brush = CreateHatchBrush(HS\_BDIAGONAL, RGB(0, 0, 255)); // Создаём кисть определённого стиля и цвета

SelectObject(dc, brush); //Выбираем кисть

RECT WinCoord = {}; //Массив координат окна

GetWindowRect(hwnd, &WinCoord); //Узнаём координаты

Rectangle(dc, 0, 15, WinCoord.right, WinCoord.bottom); //Нарисовали прямоугольник, закрашенный неким стилем

dc = GetDC(hwnd);

static LOGFONT lf;

lf.lfCharSet = DEFAULT\_CHARSET;

lf.lfPitchAndFamily = DEFAULT\_PITCH;

lf.lfHeight = 20;

lf.lfWidth = 10;

lf.lfItalic = 0;

lf.lfUnderline = 1;

lf.lfEscapement = 0;

HFONT hFont = CreateFontIndirect(&lf);

SelectObject(dc, hFont);

SetTextColor(dc, RGB(0, 0, 128));

SetBkColor(dc, RGB(0, 204, 255));

TextOut(dc, 490, 50, L"Правила игры лабиринт:", 23);

TextOut(dc, 180, 100, L"На старте вы находитесь в левом верхнем углу, ваша цель - добраться до финиша(правого нижнего угла)", 100);

TextOut(dc, 220, 150, L"Управление объектом осуществляется при помощи клавиш: a-влево, s-вниз , d-вправо, w-вверх", 90);

TextOut(dc, 170, 200, L"1.При старте игры в обычном режиме, вам необходимо ввести размеры лабиринта для дальнейшего прохожения", 103);

TextOut(dc, 30, 250, L"2.При старте игры в режиме 'со временем', вам необходимо ввести размер лабиринта, ваше имя для записи вашего результата", 120);

TextOut(dc, 180, 300, L"Сразу после нажатия клавиши Enter запустится таймер. Цель - пройти лабиринт за наименьшее кол-во времени", 105);

TextOut(dc, 40, 350, L"Также в обоих режимах можно собирать монетки, монетки в режиме 'со временем' непосредственно влияют на рекорд игрока", 117);

TextOut(dc, 365, 400, L"Чем больше собрано монет, тем престижнее результат", 51);

TextOut(dc, 300, 450, L"Файл с результатами (Результаты.txt) можно найти в папке с игрой", 65);

TextOut(dc, 25, 500, L"3.При старте игры в режиме игры с ботом, вам необходимо ввести размеры лабиринта и выбрать сложность прохождения", 113);

TextOut(dc, 20, 550, L"Также важно проверять раскладку и CAPSLOCK, ведь движение объекта происходит только с нажатием указанных выше клавиш", 117);

TextOut(dc, 370, 600, L"Нажмите любую клавишу для возвращения в меню...", 48);

ReleaseDC(hwnd, dc);

DeleteObject(brush);

ReleaseDC(hwnd, dc);

}

void graphicsofmenu() {

HWND hwnd = GetConsoleWindow(); //Берём ориентир на консольное окно (В нём будем рисовать)

HDC dc = GetDC(hwnd); //Цепляемся к консольному окну

RECT window = {}; //Переменная window будет использована для получения ширины и высоты окна

HBRUSH brush; //Переменная brush - это кисть, она будет использоваться для закрашивания

brush = CreateSolidBrush(RGB(20, 100, 200)); // Создаём кисть определённого стиля и цвета

SelectObject(dc, brush); //Выбираем кисть

RECT WinCoord = {}; //Массив координат окна

GetWindowRect(hwnd, &WinCoord); //Узнаём координаты

SelectObject(dc, brush); //Выбираем кисть

Rectangle(dc, 0, 0, WinCoord.right, WinCoord.bottom); //Рисуем новый прямоугольник, который будет небом

DeleteObject(brush); //Очищаем память от созданной, но уже ненужной кисти

brush = CreateSolidBrush(RGB(255, 204, 102)); // Выбранный элемент

SelectObject(dc, brush); //Выбираем кисть

Ellipse(dc, 760, 215, 780, 235);

DeleteObject(brush); //Очищаем память от созданной кисти

brush = CreateSolidBrush(RGB(20, 100, 200)); // Не выбранный элемент

SelectObject(dc, brush); //Выбираем кисть

Ellipse(dc, 760, 285, 780, 305);

Ellipse(dc, 760, 355, 780, 375);

Ellipse(dc, 760, 425, 780, 445);

Ellipse(dc, 760, 495, 780, 515);

DeleteObject(brush); //Очищаем память от созданной кисти

static LOGFONT lf1;

lf1.lfCharSet = DEFAULT\_CHARSET;

lf1.lfPitchAndFamily = DEFAULT\_PITCH;

lf1.lfHeight = 50;

lf1.lfWidth = 15;

lf1.lfItalic = 1;

lf1.lfUnderline = 0;

lf1.lfEscapement = 0;

HFONT hFont1 = CreateFontIndirect(&lf1);

SelectObject(dc, hFont1);

SetTextColor(dc, RGB(0, 0, 70));

SetBkColor(dc, RGB(0, 204, 255));

TextOut(dc, 530, 200, L"Обычная игра", strlen("Обычная игра"));

TextOut(dc, 527, 270, L"Игра на время", strlen("Игра на время"));

TextOut(dc, 530, 340, L"Игра c ботом", strlen("Игра c ботом"));

TextOut(dc, 570, 410, L"Правила", strlen("Правила"));

TextOut(dc, 587, 480, L"Выход", strlen("Выход"));

DeleteObject(hFont1);

static LOGFONT lf;

lf.lfCharSet = DEFAULT\_CHARSET;

lf.lfPitchAndFamily = DEFAULT\_PITCH;

lf.lfHeight = 70;

lf.lfWidth = 30;

lf.lfItalic = 0;

lf.lfUnderline = 1;

lf.lfEscapement = 0;

HFONT hFont = CreateFontIndirect(&lf);

SelectObject(dc, hFont);

SetTextColor(dc, RGB(128, 0, 0));

SetBkColor(dc, RGB(0, 204, 255));

TextOut(dc, 360, 50, L"ИГРА ЛАБИРИНТ", strlen("Игра ЛАБИРИНТ"));

//Путь

brush = CreateSolidBrush(RGB(255, 189, 187));

SelectObject(dc, brush);

Rectangle(dc, 310, 50, 330, 70);

Rectangle(dc, 945, 50, 965, 70);

DeleteObject(brush);

//Объект

brush = CreateSolidBrush(RGB(255, 0, 0));

SelectObject(dc, brush);

Rectangle(dc, 330, 50, 350, 70);

Rectangle(dc, 925, 50, 945, 70);

DeleteObject(brush);

//Стена

brush = CreateSolidBrush(RGB(105, 105, 105));

SelectObject(dc, brush);

Rectangle(dc, 310, 70, 330, 90);

Rectangle(dc, 945, 70, 965, 90);

DeleteObject(brush);

//Финиш

brush = CreateSolidBrush(RGB(50, 205, 50));

SelectObject(dc, brush);

Rectangle(dc, 330, 70, 350, 90);

Rectangle(dc, 925, 70, 945, 90);

DeleteObject(brush);

DeleteObject(hFont);

ReleaseDC(hwnd, dc);

}

void dynamic\_array\_print(int\*\* maze, int height, int width, bool stopwatch) {

system("cls");

int x1 = 450, y1 = 30, x2 = 470, y2 = 50;

HWND hwnd = GetConsoleWindow(); //Берём ориентир на консольное окно (В нём будем рисовать)

HDC dc = GetDC(hwnd); //Цепляемся к консольному окну

RECT window = {}; //Переменная window будет использована для получения ширины и высоты окна

HBRUSH brush; //Переменная brush - это кисть, она будет использоваться для закрашивания

//Правая граница

brush = CreateSolidBrush(RGB(255, 189, 187)); // Создаём кисть определённого стиля и цвета

SelectObject(dc, brush); //Выбираем кисть

RECT WinCoord = {}; //Массив координат окна

GetWindowRect(hwnd, &WinCoord); //Узнаём координаты

Rectangle(dc, 400, 0, WinCoord.right, WinCoord.bottom); //Нарисовали прямоугольник, закрашенный неким стилем

DeleteObject(brush); //Очищаем память от созданной кисти

//Левая граница

brush = CreateSolidBrush(RGB(0, 178, 228));

SelectObject(dc, brush);

Rectangle(dc, 10, 0, 402, WinCoord.bottom);

DeleteObject(brush);

//Первое поле

brush = CreateSolidBrush(RGB(59, 255, 251));

SelectObject(dc, brush);

Rectangle(dc, 90, 70, 270, 130);

DeleteObject(brush);

//Второе поле

brush = CreateSolidBrush(RGB(59, 255, 251));

SelectObject(dc, brush);

Rectangle(dc, 40, 200, 330, 270);

DeleteObject(brush);

//Третье поле

brush = CreateSolidBrush(RGB(59, 255, 251));

SelectObject(dc, brush);

Rectangle(dc, 30, 580, 370, 650);

DeleteObject(brush);

//Путь

brush = CreateSolidBrush(RGB(255, 189, 187));

SelectObject(dc, brush);

Rectangle(dc, 50, 420, 70, 440);

DeleteObject(brush);

//Объект

brush = CreateSolidBrush(RGB(255, 0, 0));

SelectObject(dc, brush);

Rectangle(dc, 50, 460, 70, 480);

DeleteObject(brush);

//Стена

brush = CreateSolidBrush(RGB(105, 105, 105));

SelectObject(dc, brush);

Rectangle(dc, 50, 500, 70, 520);

DeleteObject(brush);

//Финиш

brush = CreateSolidBrush(RGB(50, 205, 50));

SelectObject(dc, brush);

Rectangle(dc, 50, 540, 70, 560);

DeleteObject(brush);

//Монетки

brush = CreateSolidBrush(RGB(20, 100, 200)); // Не выбранный элемент

SelectObject(dc, brush);

Ellipse(dc, 130, 360, 150, 380);

Ellipse(dc, 170, 360, 190, 380);

Ellipse(dc, 210, 360, 230, 380);

DeleteObject(brush);

static LOGFONT lf;

lf.lfCharSet = DEFAULT\_CHARSET;

lf.lfPitchAndFamily = DEFAULT\_PITCH;

lf.lfHeight = 30;

lf.lfWidth = 10;

lf.lfItalic = 1;

lf.lfUnderline = 0;

lf.lfEscapement = 0;

HFONT hFont = CreateFontIndirect(&lf);

SelectObject(dc, hFont);

SetTextColor(dc, RGB(0, 0, 255));

SetBkColor(dc, RGB(59, 255, 251));

TextOut(dc, 93, 30, L"Состояние игры", strlen("Состояние игры"));

TextOut(dc, 80, 160, L"Состояние таймера", strlen("Состояние таймера"));

TextOut(dc, 80, 300, L"Собранные монетки", strlen("Собранные монетки"));

DeleteObject(hFont); //выгрузим из памяти объект шрифта

static LOGFONT lf1;

lf1.lfCharSet = DEFAULT\_CHARSET;

lf1.lfPitchAndFamily = DEFAULT\_PITCH;

lf1.lfHeight = 20;

lf1.lfWidth = 10;

lf1.lfItalic = 0;

HFONT hFont1 = CreateFontIndirect(&lf1);

SelectObject(dc, hFont1);

SetTextColor(dc, RGB(0, 0, 0));

SetBkColor(dc, RGB(59, 255, 251));

TextOut(dc, 105, 90, L"Прохождение...", strlen("Прохождение..."));

if (stopwatch == 0) {

TextOut(dc, 115, 225, L"Без таймера", strlen("Без таймера"));

}

else TextOut(dc, 90, 225, L"Таймер запущен!", strlen("Таймер запущен!"));

TextOut(dc, 80, 421, L"- Путь ", strlen("- Путь "));

TextOut(dc, 80, 461, L"- Объект ", strlen("- Объект "));

TextOut(dc, 80, 501, L"- Стена ", strlen("- Стена "));

TextOut(dc, 80, 541, L"- Финиш ", strlen("- Финиш "));

TextOut(dc, 100, 595, L"Нажмите 'Пробел'", strlen("Нажмите 'Пробел'"));

TextOut(dc, 75, 615, L"Чтобы вернуться в меню", strlen("Чтобы вернуться в меню"));

DeleteObject(hFont1); //выгрузим из памяти объект шрифта

for (int i = 0; i < height; i++) {

for (int j = 0; j < width; j++) {

switch (maze[i][j]) {

case wall:

brush = CreateSolidBrush(RGB(105, 105, 105));

SelectObject(dc, brush); //Выбираем кисть

Rectangle(dc, x1, y1, x2, y2); //Нарисовали прямоугольник, закрашенный неким стилем

DeleteObject(brush); //Очищаем память от созданной кисти

x1 += 20;

x2 += 20;

break;

case pass:

x1 += 20;

x2 += 20;

break;

case coin:

brush = CreateSolidBrush(RGB(255, 204, 102)); // Выбранный элемент

SelectObject(dc, brush); //Выбираем кисть

Ellipse(dc, x1, y1, x2, y2);

DeleteObject(brush); //Очищаем память от созданной кисти

x1 += 20;

x2 += 20;

}

}

x1 -= (width) \* 20;

x2 -= (width) \* 20;

y1 += 20;

y2 += 20;

}

//Объект

brush = CreateSolidBrush(RGB(255, 0, 0));

SelectObject(dc, brush); //Выбираем кисть

Rectangle(dc, 430, 50, 450, 70); //Нарисовали прямоугольник, закрашенный неким стилем

DeleteObject(brush); //Очищаем память от созданной кисти

//Дорисовка

RECT r;

r.left = 450 + (width - 1) \* 20;

r.top = 30;

r.right = 470 + (width - 1) \* 20;

r.bottom = 50;

FillRect(dc, &r, (HBRUSH)CreateSolidBrush(RGB(255, 189, 187)));

//Дорисовка

brush = CreateSolidBrush(RGB(105, 105, 105));

SelectObject(dc, brush); //Выбираем кисть

Rectangle(dc, 430, 30, 450, 50); //Нарисовали прямоугольник, закрашенный неким стилем

DeleteObject(brush); //Очищаем память от созданной кисти

//Финиш

brush = CreateSolidBrush(RGB(50, 205, 50));

SelectObject(dc, brush); //Выбираем кисть

Rectangle(dc, 430 + (width - 1) \* 20, 30 + (height - 2) \* 20, 450 + (width - 1) \* 20, 50 + (height - 2) \* 20); //Нарисовали прямоугольник, закрашенный неким стилем

DeleteObject(brush); //Очищаем память от созданной кисти

ReleaseDC(hwnd, dc);

}

void dynamic\_array\_bot(int\*\* maze, int height, int width, bool stopwatch){

system("cls");

int x1 = 400, y1 = 30, x2 = 420, y2 = 50;

HWND hwnd = GetConsoleWindow(); //Берём ориентир на консольное окно (В нём будем рисовать)

HDC dc = GetDC(hwnd); //Цепляемся к консольному окну

RECT window = {}; //Переменная window будет использована для получения ширины и высоты окна

HBRUSH brush; //Переменная brush - это кисть, она будет использоваться для закрашивания

//Правая граница

brush = CreateSolidBrush(RGB(255, 189, 187)); // Создаём кисть определённого стиля и цвета

SelectObject(dc, brush); //Выбираем кисть

RECT WinCoord = {}; //Массив координат окна

GetWindowRect(hwnd, &WinCoord); //Узнаём координаты

Rectangle(dc, 828, 0, WinCoord.right, WinCoord.bottom); //Нарисовали прямоугольник, закрашенный неким стилем

//Средняя граница

brush = CreateSolidBrush(RGB(255, 189, 187)); // Создаём кисть определённого стиля и цвета

SelectObject(dc, brush); //Выбираем кисть

Rectangle(dc, 348, 0, 830, WinCoord.bottom);

DeleteObject(brush); //Очищаем память от созданной кисти

//Нижняя граница

brush = CreateSolidBrush(RGB(0, 178, 228));

SelectObject(dc, brush);

Rectangle(dc, 348, 500, WinCoord.right, WinCoord.bottom);

DeleteObject(brush);

//Левая граница

brush = CreateSolidBrush(RGB(0, 178, 228));

SelectObject(dc, brush);

Rectangle(dc, 10, 0, 350, WinCoord.bottom);

DeleteObject(brush);

//Первое поле

brush = CreateSolidBrush(RGB(59, 255, 251));

SelectObject(dc, brush);

Rectangle(dc, 90, 70, 270, 130);

DeleteObject(brush);

//Второе поле

brush = CreateSolidBrush(RGB(59, 255, 251));

SelectObject(dc, brush);

Rectangle(dc, 40, 200, 330, 270);

DeleteObject(brush);

//Третье поле

brush = CreateSolidBrush(RGB(59, 255, 251));

SelectObject(dc, brush);

Rectangle(dc, 40, 295, 330, 365);

DeleteObject(brush);

//Путь

brush = CreateSolidBrush(RGB(255, 189, 187));

SelectObject(dc, brush);

Rectangle(dc, 50, 380, 70, 400);

DeleteObject(brush);

//Бот

brush = CreateSolidBrush(RGB(3, 0, 189));

SelectObject(dc, brush);

Rectangle(dc, 50, 420, 70, 440);

DeleteObject(brush);

//Объект

brush = CreateSolidBrush(RGB(255, 0, 0));

SelectObject(dc, brush);

Rectangle(dc, 50, 460, 70, 480);

DeleteObject(brush);

//Стена

brush = CreateSolidBrush(RGB(105, 105, 105));

SelectObject(dc, brush);

Rectangle(dc, 50, 500, 70, 520);

DeleteObject(brush);

//Финиш

brush = CreateSolidBrush(RGB(50, 205, 50));

SelectObject(dc, brush);

Rectangle(dc, 50, 540, 70, 560);

DeleteObject(brush);

static LOGFONT lf;

lf.lfCharSet = DEFAULT\_CHARSET;

lf.lfPitchAndFamily = DEFAULT\_PITCH;

lf.lfHeight = 30;

lf.lfWidth = 10;

lf.lfItalic = 1;

lf.lfUnderline = 0;

lf.lfEscapement = 0;

HFONT hFont = CreateFontIndirect(&lf);

SelectObject(dc, hFont);

SetTextColor(dc, RGB(0, 0, 255));

SetBkColor(dc, RGB(59, 255, 251));

TextOut(dc, 93, 30, L"Состояние игры", strlen("Состояние игры"));

TextOut(dc, 80, 160, L"Состояние таймера", strlen("Состояние таймера"));

DeleteObject(hFont); //выгрузим из памяти объект шрифта

static LOGFONT lf1;

lf1.lfCharSet = DEFAULT\_CHARSET;

lf1.lfPitchAndFamily = DEFAULT\_PITCH;

lf1.lfHeight = 20;

lf1.lfWidth = 10;

lf1.lfItalic = 0;

HFONT hFont1 = CreateFontIndirect(&lf1);

SelectObject(dc, hFont1);

SetTextColor(dc, RGB(0, 0, 0));

SetBkColor(dc, RGB(59, 255, 251));

TextOut(dc, 105, 90, L"Прохождение...", strlen("Прохождение..."));

if (stopwatch == 0) {

TextOut(dc, 115, 225, L"Без таймера", strlen("Без таймера"));

}

else TextOut(dc, 90, 225, L"Таймер запущен!", strlen("Таймер запущен!"));

TextOut(dc, 80, 381, L"- Путь ", strlen("- Путь "));

TextOut(dc, 80, 421, L"- Бот ", strlen("- Бот "));

TextOut(dc, 80, 461, L"- Объект ", strlen("- Объект "));

TextOut(dc, 80, 501, L"- Стена ", strlen("- Стена "));

TextOut(dc, 80, 541, L"- Финиш ", strlen("- Финиш "));

TextOut(dc, 90, 310, L"Нажмите 'Пробел'", strlen("Нажмите 'Пробел'"));

TextOut(dc, 60, 330, L"Чтобы вернуться в меню", strlen("Чтобы вернуться в меню"));

DeleteObject(hFont1); //выгрузим из памяти объект шрифта

for (int k = 0; k < height; k++) {

for (int i = 0; i < height; i++) {

for (int j = 0; j < width; j++) {

switch (maze[i][j]) {

case wall:

brush = CreateSolidBrush(RGB(105, 105, 105));

SelectObject(dc, brush); //Выбираем кисть

Rectangle(dc, x1, y1, x2, y2); //Нарисовали прямоугольник, закрашенный неким стилем

DeleteObject(brush); //Очищаем память от созданной кисти

x1 += 20;

x2 += 20;

break;

case pass:

x1 += 20;

x2 += 20;

break;

}

}

x1 -= (width) \* 20;

x2 -= (width) \* 20;

y1 += 20;

y2 += 20;

}

x1 = 900;

y1 = 30;

x2 = 920;

y2 = 50;

}

//Объект

brush = CreateSolidBrush(RGB(255, 0, 0));

SelectObject(dc, brush); //Выбираем кисть

Rectangle(dc, 380, 50, 400, 70); //Нарисовали прямоугольник, закрашенный неким стилем

DeleteObject(brush); //Очищаем память от созданной кисти

//Дорисовка

RECT r;

r.left = 400 + (width - 1) \* 20;

r.top = 30;

r.right = 420 + (width - 1) \* 20;

r.bottom = 50;

FillRect(dc, &r, (HBRUSH)CreateSolidBrush(RGB(255, 189, 187)));

RECT r1;

r1.left = 900 + (width - 1) \* 20;

r1.top = 30;

r1.right = 920 + (width - 1) \* 20;

r1.bottom = 50;

FillRect(dc, &r1, (HBRUSH)CreateSolidBrush(RGB(255, 189, 187)));

r1.left = 880;

r1.top = 30;

r1.right = 900;

r1.bottom = 50;

FillRect(dc, &r1, (HBRUSH)CreateSolidBrush(RGB(255, 189, 187)));

//Дорисовка

brush = CreateSolidBrush(RGB(105, 105, 105));

SelectObject(dc, brush); //Выбираем кисть

Rectangle(dc, 380, 30, 400, 50);

Rectangle(dc, 880, 30, 900, 50);//Нарисовали прямоугольник, закрашенный неким стилем

DeleteObject(brush); //Очищаем память от созданной кисти

//Финиш

brush = CreateSolidBrush(RGB(50, 205, 50));

SelectObject(dc, brush); //Выбираем кисть

Rectangle(dc, 380 + (width - 1) \* 20, 30 + (height - 2) \* 20, 400 + (width - 1) \* 20, 50 + (height - 2) \* 20); //Нарисовали прямоугольник, закрашенный неким стилем

Rectangle(dc, 880 + (width - 1) \* 20, 30 + (height - 2) \* 20, 900 + (width - 1) \* 20, 50 + (height - 2) \* 20);

DeleteObject(brush); //Очищаем память от созданной кисти

ReleaseDC(hwnd, dc);

}

Файл: Mechanics.h

#include<math.h>

#include<windows.h>

#include<iostream>

#include<memory.h>

#include<time.h>

#include<malloc.h>

#include<conio.h>

#include<vector>

using namespace std;

#define KEY\_SPACE 32 //определение клавиши "пробел"

int \*\* dynamic\_array(size\_t height, size\_t width);

void dynamic\_array\_free(int \*\*maze, size\_t height);

bool deadend(int x, int y, int\*\* maze, int height, int width);

void go(int\*\* maze, int height, int width, bool stopwatch, bool check\_bot);

void go2(int\*\* maze, int height, int width, double &difP);

void mazegame(int height, int width, bool stopwatch, bool check\_bot, int lvl);

void Way\_coord(int\*\* maze, int \*py, int \*px, int \*xc1, int \*yc1, int \*xc2, int \*yc2, int lvl);

int Find\_way(int\*\* maze, int y, int x);

void runbot(int\*\* maze, int height, int lvl, double &difB);

void mainmech(int\*\* maze, int height, int lvl);

void grph\_bot1(int \*xc1, int \*yc1, int \*xc2, int \*yc2);

void grph\_bot2(int \*xc1, int \*yc1, int \*xc2, int \*yc2, int lvl);

Файл: Mechanics.cpp

#include "Mechanics.h"

#include "Graphics.h"

#include "Menu.h"

#include <thread>

using namespace std;

//Выделение памяти под динамический массив

int \*\* dynamic\_array(size\_t height, size\_t width) {

int \*\*maze;

maze = (int\*\*)malloc(height \* sizeof(int\*));

for (int i = 0; i < width; ++i) {

maze[i] = (int\*)malloc(width \* sizeof(int));

}

return maze;

}

//Освобождение памяти от динамического массива

void dynamic\_array\_free(int \*\*maze, size\_t height) {

for (int i = 0; i < height; i++) {

free(maze[i]);

}

free(maze);

maze = NULL;

}

//Функция для тупика

bool deadend(int x, int y, int\*\* maze, int height, int width) {

int check = 0;

if (x != 1) {

if (maze[y][x - 2] == pass)

check += 1;

}

else check += 1;

if (y != 1) {

if (maze[y - 2][x] == pass)

check += 1;

}

else check += 1;

if (x != width - 2) {

if (maze[y][x + 2] == pass)

check += 1;

}

else check += 1;

if (y != height - 2) {

if (maze[y + 2][x] == pass)

check += 1;

}

else check += 1;

if (check == 4)

return 1;

else

return 0;

}

//Графика движения бота

void grph\_bot1(int \*xc1, int \*yc1, int \*xc2, int \*yc2) {

HWND hwnd = GetConsoleWindow(); //Берём ориентир на консольное окно (В нём будем рисовать)

HDC dc = GetDC(hwnd); //Цепляемся к консольному окну

RECT r;

r.left = \*xc1;

r.top = \*yc1;

r.right = \*xc2;

r.bottom = \*yc2;

FillRect(dc, &r, (HBRUSH)CreateSolidBrush(RGB(255, 189, 187)));

ReleaseDC(hwnd, dc);

}

//Графика движения бота

void grph\_bot2(int \*xc1, int \*yc1, int \*xc2, int \*yc2, int lvl) {

HWND hwnd = GetConsoleWindow(); //Берём ориентир на консольное окно (В нём будем рисовать)

HDC dc = GetDC(hwnd); //Цепляемся к консольному окну

HBRUSH brush; //Переменная brush - это кисть, она будет использоваться для закрашивания

brush = CreateSolidBrush(RGB(5, 2, 189));

SelectObject(dc, brush); //Выбираем кисть

Rectangle(dc, \*xc1, \*yc1, \*xc2, \*yc2); //Нарисовали прямоугольник, закрашенный неким стилем

DeleteObject(brush); //Очищаем память от созданной кисти

ReleaseDC(hwnd, dc);

if (lvl == 1) {

Sleep(lvl \* 250);

}

else if (lvl == 2) {

Sleep(lvl \* 110);

}

else if (lvl == 3) {

Sleep(lvl \* 60);

}

}

//Создание копии динамического массива

int\*\* copymaze(int\*\* sourseMatrix, int rows, int cols)

{

int\*\* cmaze = new int\*[rows];

for (int i = 0; i < rows; i++)

cmaze[i] = new int[cols];

for (int i = 0; i < rows; i++)

{

for (int j = 0; j < cols; j++)

{

cmaze[i][j] = sourseMatrix[i][j];

}

}

return cmaze;

}

//Функция для просмотра количества путей

int Find\_way(int\*\* maze, int y, int x) {

int counter = 0;

if (maze[y + 1][x] == 1) counter++;//Вниз

if (maze[y - 1][x] == 1) counter++;//Вверх

if (maze[y][x + 1] == 1) counter++;//Вправо

if (maze[y][x - 1] == 1) counter++;//Влево

return counter;

}

//Фукнция для просмотра типа пути

void Way\_coord(int\*\* maze, int \*py, int \*px, int \*xc1, int \*yc1, int \*xc2, int \*yc2, int lvl) {

if (maze[\*(py) + 1][\*px] == 1) {

maze[\*py][\*px] = 4;

maze[\*(py) + 1][\*px] = 4;

maze[\*(py) + 2][\*px] = 4;

grph\_bot1(xc1, yc1, xc2, yc2);

\*(yc1) += 20;

\*(yc2) += 20;

\*(py)+= 2;

grph\_bot2(xc1, yc1, xc2, yc2, lvl);

grph\_bot1(xc1, yc1, xc2, yc2);

\*(yc1) += 20;

\*(yc2) += 20;

grph\_bot2(xc1, yc1, xc2, yc2, lvl);

return;

}

if (maze[\*py][\*(px)+1] == 1) {

maze[\*py][\*px] = 4;

maze[\*py][\*(px)+1] = 4;

maze[\*py][\*(px)+2] = 4;

grph\_bot1(xc1, yc1, xc2, yc2);

\*(xc1) += 20;

\*(xc2) += 20;

\*(px) += 2;

grph\_bot2(xc1, yc1, xc2, yc2, lvl);

grph\_bot1(xc1, yc1, xc2, yc2);

\*(xc1) += 20;

\*(xc2) += 20;

grph\_bot2(xc1, yc1, xc2, yc2, lvl);

return;

}

if (maze[\*(py) - 1][\*px] == 1) {

maze[\*py][\*px] = 4;

maze[\*(py) - 1][\*px] = 4;

maze[\*(py) - 2][\*px] = 4;

grph\_bot1(xc1, yc1, xc2, yc2);

\*(yc1) -= 20;

\*(yc2) -= 20;

\*(py) -= 2;

grph\_bot2(xc1, yc1, xc2, yc2, lvl);

grph\_bot1(xc1, yc1, xc2, yc2);

\*(yc1) -= 20;

\*(yc2) -= 20;

grph\_bot2(xc1, yc1, xc2, yc2, lvl);

return;

}

if (maze[\*py][\*(px) - 1] == 1) {

maze[\*py][\*px] = 4;

maze[\*py][\*(px) - 1] = 4;

maze[\*py][\*(px) - 2] = 4;

grph\_bot1(xc1, yc1, xc2, yc2);

\*(xc1) -= 20;

\*(xc2) -= 20;

\*(px) -= 2;

grph\_bot2(xc1, yc1, xc2, yc2, lvl);

grph\_bot1(xc1, yc1, xc2, yc2);

\*(xc1) -= 20;

\*(xc2) -= 20;

grph\_bot2(xc1, yc1, xc2, yc2, lvl);

return;

}

}

//Логика бота

void runbot(int\*\* maze, int height, int lvl, double &difB) {

int y = 1, x = 1, ctr, yc1 = 50, xc1 = 900, yc2 = 70, xc2 = 920;

int y1 = 50, y2 = 70, x1 = 380, x2 = 400;

time\_t t1b, t2b;

vector<int> WayBot;

maze[1][0] = 0;

t1b = time(NULL);

ctr = Find\_way(maze, y, x);

if (ctr > 1) {

WayBot.push\_back(y);

WayBot.push\_back(x);

}

while (TRUE) {

if (y == height - 2 && x == height - 2) break;

//бот всегда будет возвращаться в точку отправки от вектора!!!

//Вернуться назад, до точки разъединения

ctr = Find\_way(maze, y, x);

while (ctr == 0) {

while (TRUE) {

if (y == WayBot[WayBot.size() - 2] && x == WayBot[WayBot.size() - 1]) {

break;

}

//Вниз

if (maze[y + 1][x] == 4) {

maze[y][x] = 5;

maze[y + 1][x] = 5;

maze[y + 2][x] = 5;

grph\_bot1(&xc1, &yc1, &xc2, &yc2);

yc1 += 20;

yc2 += 20;

grph\_bot2(&xc1, &yc1, &xc2, &yc2, lvl);

grph\_bot1(&xc1, &yc1, &xc2, &yc2);

yc1 += 20;

yc2 += 20;

grph\_bot2(&xc1, &yc1, &xc2, &yc2, lvl);

y += 2;

}

//Вправо

else if (maze[y][x + 1] == 4) {

maze[y][x] = 5;

maze[y][x + 1] = 5;

maze[y][x + 2] = 5;

grph\_bot1(&xc1, &yc1, &xc2, &yc2);

xc1 += 20;

xc2 += 20;

grph\_bot2(&xc1, &yc1, &xc2, &yc2, lvl);

grph\_bot1(&xc1, &yc1, &xc2, &yc2);

xc1 += 20;

xc2 += 20;

grph\_bot2(&xc1, &yc1, &xc2, &yc2, lvl);

x += 2;

}

//Вверх

else if (maze[y - 1][x] == 4) {

maze[y][x] = 5;

maze[y - 1][x] = 5;

maze[y - 2][x] = 5;

grph\_bot1(&xc1, &yc1, &xc2, &yc2);

yc1 -= 20;

yc2 -= 20;

grph\_bot2(&xc1, &yc1, &xc2, &yc2, lvl);

grph\_bot1(&xc1, &yc1, &xc2, &yc2);

yc1 -= 20;

yc2 -= 20;

grph\_bot2(&xc1, &yc1, &xc2, &yc2, lvl);

y -= 2;

}

//Влево

else if (maze[y][x - 1] == 4) {

maze[y][x] = 5;

maze[y][x - 1] = 5;

maze[y][x - 2] = 5;

grph\_bot1(&xc1, &yc1, &xc2, &yc2);

xc1 -= 20;

xc2 -= 20;

grph\_bot2(&xc1, &yc1, &xc2, &yc2, lvl);

grph\_bot1(&xc1, &yc1, &xc2, &yc2);

xc1 -= 20;

xc2 -= 20;

grph\_bot2(&xc1, &yc1, &xc2, &yc2, lvl);

x -= 2;

}

}

ctr = Find\_way(maze, y, x);

if (ctr == 0) {

WayBot.pop\_back();

WayBot.pop\_back();

}

}

if (ctr == 1) {

Way\_coord(maze, &y, &x, &xc1, &yc1, &xc2, &yc2, lvl);

}

else if (ctr > 1) {

//Записать одно из направлений в вектор и пойти в него

WayBot.push\_back(y);

WayBot.push\_back(x);

Way\_coord(maze, &y, &x, &xc1, &yc1, &xc2, &yc2, lvl);

}

}

grph\_bot1(&xc1, &yc1, &xc2, &yc2);

xc1 += 20;

xc2 += 20;

grph\_bot2(&xc1, &yc1, &xc2, &yc2, lvl);

t2b = time(NULL);

difB = difftime(t2b, t1b);

dynamic\_array\_free(maze, height);

}

//функция для движения пользователя

void go2(int\*\* maze, int height, int width, double &difP) {

HWND hwnd = GetConsoleWindow(); //Берём ориентир на консольное окно (В нём будем рисовать)

HDC dc = GetDC(hwnd); //Цепляемся к консольному окну

RECT window = {}; //Переменная window будет использована для получения ширины и высоты окна

HBRUSH brush; //Переменная brush - это кисть, она будет использоваться для закрашивания

time\_t t1, t2;

int xG = 0, yG = 1;

int x1g = 380, x2g = 400, y1g = 50, y2g = 70;

t1 = time(NULL);

while (yG != width - 2) {

while (xG != height - 1) {

if (\_kbhit()) //если нажата клавиша

{

switch (\_getch()) {

case 'a'://влево

if (maze[yG][xG - 1] != 0 && maze[1][0] != digger) {

maze[yG][xG] = pass;

maze[yG][xG - 1] = digger;

xG -= 1;

//Зарисовка поля

RECT r;

r.left = x1g;

r.top = y1g;

r.right = x2g;

r.bottom = y2g;

FillRect(dc, &r, (HBRUSH)CreateSolidBrush(RGB(255, 189, 187)));

x1g -= 20; x2g -= 20;

//Объект

brush = CreateSolidBrush(RGB(255, 0, 0));

SelectObject(dc, brush); //Выбираем кисть

Rectangle(dc, x1g, y1g, x2g, y2g); //Нарисовали прямоугольник, закрашенный неким стилем

DeleteObject(brush); //Очищаем память от созданной кисти

}

else break;

break;

case 'd'://вправо

if (maze[yG][xG + 1] != 0) {

maze[yG][xG] = pass;

maze[yG][xG + 1] = digger;

xG += 1;

//Зарисовка поля

RECT r;

r.left = x1g;

r.top = y1g;

r.right = x2g;

r.bottom = y2g;

FillRect(dc, &r, (HBRUSH)CreateSolidBrush(RGB(255, 189, 187)));

x1g += 20; x2g += 20;

//Объект

brush = CreateSolidBrush(RGB(255, 0, 0));

SelectObject(dc, brush); //Выбираем кисть

Rectangle(dc, x1g, y1g, x2g, y2g); //Нарисовали прямоугольник, закрашенный неким стилем

DeleteObject(brush); //Очищаем память от созданной кисти

}

else break;

break;

case 's'://вниз

if (maze[yG + 1][xG] != 0) {

maze[yG][xG] = pass;

maze[yG + 1][xG] = digger;

yG += 1;

//Зарисовка поля

RECT r;

r.left = x1g;

r.top = y1g;

r.right = x2g;

r.bottom = y2g;

FillRect(dc, &r, (HBRUSH)CreateSolidBrush(RGB(255, 189, 187)));

y1g += 20; y2g += 20;

//Объект

brush = CreateSolidBrush(RGB(255, 0, 0));

SelectObject(dc, brush); //Выбираем кисть

Rectangle(dc, x1g, y1g, x2g, y2g); //Нарисовали прямоугольник, закрашенный неким стилем

DeleteObject(brush); //Очищаем память от созданной кисти

}

else break;

break;

case 'w'://вверх

if (maze[yG - 1][xG] != 0) {

maze[yG][xG] = pass;

maze[yG - 1][xG] = digger;

yG -= 1;

//Зарисовка поля

RECT r;

r.left = x1g;

r.top = y1g;

r.right = x2g;

r.bottom = y2g;

FillRect(dc, &r, (HBRUSH)CreateSolidBrush(RGB(255, 189, 187)));

y1g -= 20; y2g -= 20;

//Объект

brush = CreateSolidBrush(RGB(255, 0, 0));

SelectObject(dc, brush); //Выбираем кисть

Rectangle(dc, x1g, y1g, x2g, y2g); //Нарисовали прямоугольник, закрашенный неким стилем

DeleteObject(brush); //Очищаем память от созданной кисти

}

else break;

break;

}

}

}

}

t2 = time(NULL);

difP = difftime(t2, t1);

dynamic\_array\_free(maze, height);

ReleaseDC(hwnd, dc);

}

//Функция для движения объекта по лабиринту

void go(int\*\* maze, int height, int width, bool stopwatch, bool check\_bot) {

int xG = 0, yG = 1, amount = 0;

time\_t t1, t2;

double differ;

FILE \*file;

int x1g = 430, x2g = 450, y1g = 50, y2g = 70;

if (check\_bot) {

x1g = 380, x2g = 400;

}

int xcoin1 = 130, ycoin1 = 360, xcoin2 = 150, ycoin2 = 380;

t1 = time(NULL);

HWND hwnd = GetConsoleWindow(); //Берём ориентир на консольное окно (В нём будем рисовать)

HDC dc = GetDC(hwnd); //Цепляемся к консольному окну

RECT window = {}; //Переменная window будет использована для получения ширины и высоты окна

HBRUSH brush; //Переменная brush - это кисть, она будет использоваться для закрашивания

while (yG != width - 2) {

while (xG != height - 1) {

if (\_kbhit()) //если нажата клавиша

{

switch (\_getch()) {

case 'a'://влево

if (maze[yG][xG - 1] != 0 && maze[1][0] != digger) {

if (maze[yG][xG - 1] == coin) {

brush = CreateSolidBrush(RGB(255, 204, 102)); // Выбранный элемент

SelectObject(dc, brush); //Выбираем кисть

Ellipse(dc, xcoin1, ycoin1, xcoin2, ycoin2);

DeleteObject(brush); //Очищаем память от созданной кисти

xcoin1 += 40;

xcoin2 += 40;

amount++;

}

maze[yG][xG] = pass;

maze[yG][xG - 1] = digger;

xG -= 1;

//Зарисовка поля

RECT r;

r.left = x1g;

r.top = y1g;

r.right = x2g;

r.bottom = y2g;

FillRect(dc, &r, (HBRUSH)CreateSolidBrush(RGB(255, 189, 187)));

x1g -= 20; x2g -= 20;

//Объект

brush = CreateSolidBrush(RGB(255, 0, 0));

SelectObject(dc, brush); //Выбираем кисть

Rectangle(dc, x1g, y1g, x2g, y2g); //Нарисовали прямоугольник, закрашенный неким стилем

DeleteObject(brush); //Очищаем память от созданной кисти

}

else break;

break;

case 'd'://вправо

if (maze[yG][xG + 1] != 0) {

if (maze[yG][xG + 1] == coin) {

brush = CreateSolidBrush(RGB(255, 204, 102)); // Выбранный элемент

SelectObject(dc, brush); //Выбираем кисть

Ellipse(dc, xcoin1, ycoin1, xcoin2, ycoin2);

DeleteObject(brush); //Очищаем память от созданной кисти

xcoin1 += 40;

xcoin2 += 40;

amount++;

}

maze[yG][xG] = pass;

maze[yG][xG + 1] = digger;

xG += 1;

//Зарисовка поля

RECT r;

r.left = x1g;

r.top = y1g;

r.right = x2g;

r.bottom = y2g;

FillRect(dc, &r, (HBRUSH)CreateSolidBrush(RGB(255, 189, 187)));

x1g += 20; x2g += 20;

//Объект

brush = CreateSolidBrush(RGB(255, 0, 0));

SelectObject(dc, brush); //Выбираем кисть

Rectangle(dc, x1g, y1g, x2g, y2g); //Нарисовали прямоугольник, закрашенный неким стилем

DeleteObject(brush); //Очищаем память от созданной кисти

}

else break;

break;

case 's'://вниз

if (maze[yG + 1][xG] != 0) {

if (maze[yG + 1][xG] == coin) {

brush = CreateSolidBrush(RGB(255, 204, 102)); // Выбранный элемент

SelectObject(dc, brush); //Выбираем кисть

Ellipse(dc, xcoin1, ycoin1, xcoin2, ycoin2);

DeleteObject(brush); //Очищаем память от созданной кисти

xcoin1 += 40;

xcoin2 += 40;

amount++;

}

maze[yG][xG] = pass;

maze[yG + 1][xG] = digger;

yG += 1;

//Зарисовка поля

RECT r;

r.left = x1g;

r.top = y1g;

r.right = x2g;

r.bottom = y2g;

FillRect(dc, &r, (HBRUSH)CreateSolidBrush(RGB(255, 189, 187)));

y1g += 20; y2g += 20;

//Объект

brush = CreateSolidBrush(RGB(255, 0, 0));

SelectObject(dc, brush); //Выбираем кисть

Rectangle(dc, x1g, y1g, x2g, y2g); //Нарисовали прямоугольник, закрашенный неким стилем

DeleteObject(brush); //Очищаем память от созданной кисти

}

else break;

break;

case 'w'://вверх

if (maze[yG - 1][xG] != 0) {

if (maze[yG - 1][xG] == coin) {

brush = CreateSolidBrush(RGB(255, 204, 102)); // Выбранный элемент

SelectObject(dc, brush); //Выбираем кисть

Ellipse(dc, xcoin1, ycoin1, xcoin2, ycoin2);

DeleteObject(brush); //Очищаем память от созданной кисти

xcoin1 += 40;

xcoin2 += 40;

amount++;

}

maze[yG][xG] = pass;

maze[yG - 1][xG] = digger;

yG -= 1;

//Зарисовка поля

RECT r;

r.left = x1g;

r.top = y1g;

r.right = x2g;

r.bottom = y2g;

FillRect(dc, &r, (HBRUSH)CreateSolidBrush(RGB(255, 189, 187)));

y1g -= 20; y2g -= 20;

//Объект

brush = CreateSolidBrush(RGB(255, 0, 0));

SelectObject(dc, brush); //Выбираем кисть

Rectangle(dc, x1g, y1g, x2g, y2g); //Нарисовали прямоугольник, закрашенный неким стилем

DeleteObject(brush); //Очищаем память от созданной кисти

}

else break;

break;

case KEY\_SPACE:

file = fopen("Результаты.txt", "a");

fprintf(file, "Лабиринт пройден не полностью\n");

fprintf(file, "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

fclose(file);

stopwatch = 0;

dynamic\_array\_free(maze, height);

system("cls");

menu();

break;

}

}

}

}

static LOGFONT lf1;

lf1.lfCharSet = DEFAULT\_CHARSET;

lf1.lfPitchAndFamily = DEFAULT\_PITCH;

lf1.lfHeight = 20;

lf1.lfWidth = 10;

lf1.lfItalic = 0;

HFONT hFont1 = CreateFontIndirect(&lf1);

SelectObject(dc, hFont1);

SetTextColor(dc, RGB(0, 0, 0));

SetBkColor(dc, RGB(59, 255, 251));

TextOut(dc, 105, 90, L"Игра пройдена!", strlen("Игра пройдена!"));

if (stopwatch == 1) {

TextOut(dc, 80, 225, L"Таймер остановлен", strlen("Таймер остановлен"));

t2 = time(NULL);

differ = difftime(t2, t1);

file = fopen("Результаты.txt", "a");

fprintf(file, "Размеры лабиринта - %d x %d \n", height, width);

fprintf(file, "Кол-во собранных монеток - %d\n", amount);

fprintf(file, "Время прохождения - %.2f сек.\n", differ);

fprintf(file, "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

fclose(file);

}

RECT r;

r.left = 550;

r.top = 300;

r.right = 1065;

r.bottom = 400;

FillRect(dc, &r, (HBRUSH)CreateSolidBrush(RGB(102, 102, 153)));

TextOut(dc, 570, 340, L"Нажмите любую клавишу, чтобы вернуться", strlen("Нажмите любую клавишу, чтобы вернуться"));

DeleteObject(hFont1); //выгрузим из памяти объект шрифта

dynamic\_array\_free(maze, height);

\_getch();

ReleaseDC(hwnd, dc);

menu();

}

//Основная функция

void mazegame(int height, int width, bool stopwatch, bool check\_bot, int lvl) {

int x, y, random, count, coins = 0;

int \*\*maze = dynamic\_array(width, height);

for (int i = 0; i < height; i++) { // Массив заполняется землей-ноликами

for (int j = 0; j < width; j++) {

maze[i][j] = wall;

}

}

x = 1; //Начальные координаты копателя

y = 1;

count = 0; //счетчик

while (count < 1000) {

maze[y][x] = pass; count++;

while (pass) { // Бесконечный цикл, который прерывается только тупиком

random = rand() % 4;

switch (random) {

case 0: if (y != 1) {

if (maze[y - 2][x] == wall) { // Вверх

maze[y - 1][x] = pass;

maze[y - 2][x] = pass;

y -= 2;

}

}

case 1: if (y != height - 2) {

if (maze[y + 2][x] == wall) { // Вниз

maze[y + 1][x] = pass;

maze[y + 2][x] = pass;

y += 2;

}

}

case 2: if (x != 1) {

if (maze[y][x - 2] == wall) { // Влево

maze[y][x - 1] = pass;

maze[y][x - 2] = pass;

x -= 2;

}

}

case 3: if (x != width - 2) {

if (maze[y][x + 2] == wall) { // Вправо

maze[y][x + 1] = pass;

maze[y][x + 2] = pass;

x += 2;

}

}

}

if (deadend(x, y, maze, height, width)) //Проверка на тупик

break;

}

if (deadend(x, y, maze, height, width)) { // Возвращаем копателя из тупика

do {

x = 2 \* (rand() % ((width - 1) / 2)) + 1;

y = 2 \* (rand() % ((height - 1) / 2)) + 1;

} while (maze[y][x] != pass);

}

}

maze[1][0] = digger;

maze[width - 2][height - 1] = pass;

//Графика построенного лабиринта

if (check\_bot) {

//Графика

dynamic\_array\_bot(maze, height, width, stopwatch);

//Потоки

mainmech(maze, height, lvl);

//Движение

go(maze, height, width, stopwatch, 1);

}

else {

while (coins != 3) {

x = 2 \* (rand() % ((width - 1) / 2)) + 1;

y = 2 \* (rand() % ((height - 1) / 2)) + 1;

if (maze[y][x] == pass) {

maze[y][x] = coin;

coins++;

}

}

dynamic\_array\_print(maze, height, width, stopwatch);

//Движение

go(maze, height, width, stopwatch, 0);

}

}

//Функция для вызова потоков

void mainmech(int\*\* maze, int height, int lvl) {

double difP = 0, difB = 0;

FILE \*file;

HWND hwnd = GetConsoleWindow(); //Берём ориентир на консольное окно (В нём будем рисовать)

HDC dc = GetDC(hwnd); //Цепляемся к консольному окну

RECT window = {}; //Переменная window будет использована для получения ширины и высоты окна

HBRUSH brush; //Переменная brush - это кисть, она будет использоваться для закрашивания

RECT r;

r.left = 530;

r.top = 300;

r.right = 1065;

r.bottom = 400;

int \*\*cmaze = copymaze(maze, height, height);

std::thread th(go2, cmaze, height, height, std::ref(difP));

std::thread th1(runbot, maze, height, lvl, std::ref(difB));

th.join();

th1.join();

if (difB < difP) {

FillRect(dc, &r, (HBRUSH)CreateSolidBrush(RGB(102, 102, 153)));

TextOut(dc, 700, 340, L"Бот оказался сильнее...", strlen("Бот оказался сильнее..."));

}

else {

FillRect(dc, &r, (HBRUSH)CreateSolidBrush(RGB(102, 102, 153)));

TextOut(dc, 740, 340, L"Вы победили!", strlen("Вы победили!"));

}

file = fopen("Результаты.txt", "a");

fprintf(file, "Размеры лабиринта - %d x %d \n", height, height);

fprintf(file, "Время бота - %.2f сек.\n", difB);

fprintf(file, "Время игрока - %.2f сек.\n", difP);

fprintf(file, "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

fclose(file);

ReleaseDC(hwnd, dc);

\_getch();

menu();

}

Файл: Menu.h

#include<windows.h>

#include<iostream>

#define STRLEN\_(x) #x

#define STRLEN(x) STRLEN\_(x)

#define LEN 15

#define KEY\_UP 72 //определение клавиши "вверх"

#define KEY\_DOWN 80 //определение клавиши "вниз"

#define KEY\_LEFT 75 //определение клавиши "влево"

#define KEY\_RIGHT 77 //определение клавиши "вправо"

#define KEY\_SPACE 32 //определение клавиши "пробел"

#define KEY\_ESC 27 //определение клавиши "escape"

#define KEY\_ENTER 13 //определение клавиши "enter"

//Пользовательское меню

void menu();

Файл: Menu.cpp

#include "Mechanics.h"

#include "Graphics.h"

#include "Menu.h"

//Функция пользовательского меню

void menu() {

int p = 1, c = 0;

int x11 = 760, x22 = 780, y11 = 215, y22 = 235;

int height, width;

char name[LEN];

system("cls");

FILE \*file;

bool stopwatch = 0;//Проверка на то, нужен ли секундомер и ввод никнейма

graphicsofmenu();

HWND hwnd = GetConsoleWindow(); //Берём ориентир на консольное окно (В нём будем рисовать)

HDC dc = GetDC(hwnd); //Цепляемся к консольному окну

RECT window = {}; //Переменная window будет использована для получения ширины и высоты окна

HBRUSH brush; //Переменная brush - это кисть, она будет использоваться для закрашивания

//Правая граница

brush = CreateSolidBrush(RGB(20, 10, 100)); // Создаём кисть определённого стиля и цвета

SelectObject(dc, brush); //Выбираем кисть

RECT WinCoord = {}; //Массив координат окна

GetWindowRect(hwnd, &WinCoord); //Узнаём координаты

while (TRUE) //навигация по меню

{

int prov = 0;

c = \_getch();

switch (c)

{

case '1':

case '2':

case '3':

case '4':

case '5':

p = c - '0';

case KEY\_ENTER:

switch (p)

{

case 1:

Rectangle(dc, 0, 0, WinCoord.right, WinCoord.bottom); //Нарисовали прямоугольник, закрашенный неким стилем

DeleteObject(brush); //Очищаем память от созданной кисти

while (prov == 0) {

system("cls");

Rectangle(dc, 0, 0, WinCoord.right, WinCoord.bottom); //Нарисовали прямоугольник, закрашенный неким стилем

printf("Условия для размеров лабиринта: \n");

printf("1)Длина равна Ширине. \n");

printf("2)Длина/ширина - нечётные величины на отрезке от 11 до 33 \n");

printf("Введите длину лабиринта - \n");

scanf\_s("%d", &height);

printf("Введите ширину лабиринта - \n");

scanf\_s("%d", &width);

//Проверка значений

\_\_asm {

mov eax, height;

mov ebx, width;

mov edi, 2;

cmp eax, 34;

jge end1;

cmp eax, 10;

jle end1;

mov ecx, eax;

cdq;

idiv edi;

cmp edx, 0;

je end1;

cmp ecx, ebx;

jne end1;

cmp ebx, 34;

jge end1;

cmp ebx, 10;

jle end1;

mov eax, ebx;

cdq;

idiv edi;

cmp edx, 0;

je end1;

jmp endgood;

end1:

mov prov, 0;

jmp finaly;

endgood:

mov prov, 1;

jmp finaly;

finaly:;

}

}

DeleteObject(brush); //Очищаем память от созданной кисти

ReleaseDC(hwnd, dc);

mazegame(height, width, stopwatch, 0, 0);

break;

case 2:

Rectangle(dc, 0, 0, WinCoord.right, WinCoord.bottom); //Нарисовали прямоугольник, закрашенный неким стилем

DeleteObject(brush); //Очищаем память от созданной кисти

char name[LEN];

while (prov == 0) {

system("cls");

Rectangle(dc, 0, 0, WinCoord.right, WinCoord.bottom); //Нарисовали прямоугольник, закрашенный неким стилем

printf("Условия для размеров лабиринта: \n");

printf("1)Длина равна Ширине. \n");

printf("2)Длина/ширина - нечётные величины на отрезке от 11 до 33 \n");

printf("Введите длину лабиринта - \n");

scanf\_s("%d", &height);

printf("Введите ширину лабиринта - \n");

scanf\_s("%d", &width);

\_\_asm {

mov eax, height;

mov ebx, width;

mov edi, 2;

cmp eax, 34;

jge end2;

cmp eax, 10;

jle end2;

mov ecx, eax;

cdq;

idiv edi;

cmp edx, 0;

je end2;

cmp ecx, ebx;

jne end2;

cmp ebx, 34;

jge end2;

cmp ebx, 10;

jle end2;

mov eax, ebx;

cdq;

idiv edi;

cmp edx, 0;

je end2;

jmp endgood2;

end2:

mov prov, 0;

jmp finaly2;

endgood2:

mov prov, 1;

jmp finaly2;

finaly2:;

}

}

printf("Введите ваше имя (< 15 символов) \n");

scanf("%" STRLEN(LEN) "s", name);

file = fopen("Результаты.txt", "a");

fprintf(file, "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

setlocale(LC\_ALL, "Rus");

fprintf(file, "Имя/никнейм - %s \n", name);

fclose(file);

DeleteObject(brush); //Очищаем память от созданной кисти

ReleaseDC(hwnd, dc);

stopwatch = 1;

mazegame(height, width, stopwatch, 0, 0);

break;

case 3:

char name1[LEN];

int lvl;

while (prov == 0) {

system("cls");

Rectangle(dc, 0, 0, WinCoord.right, WinCoord.bottom); //Нарисовали прямоугольник, закрашенный неким стилем

printf("Условия для размеров лабиринта: \n");

printf("1)Длина равна Ширине. \n");

printf("2)Длина/ширина - нечётные величины на отрезке от 11 до 21 \n");

printf("Введите длину лабиринта - \n");

scanf\_s("%d", &height);

printf("Введите ширину лабиринта - \n");

scanf\_s("%d", &width);

printf("Введите сложность лабиринта лабиринта - '1'. '2' или '3' \n");

scanf\_s("%d", &lvl);

\_\_asm {

mov eax, height;

mov ebx, width;

mov edi, 2;

cmp eax, 22;

jge end22;

cmp eax, 10;

jle end22;

mov ecx, eax;

cdq;

idiv edi;

cmp edx, 0;

je end22;

cmp ecx, ebx;

jne end22;

cmp ebx, 22;

jge end22;

cmp ebx, 10;

jle end22;

mov eax, ebx;

cdq;

idiv edi;

cmp edx, 0;

je end22;

jmp egd2;

end22:

mov prov, 0;

jmp fnl2;

egd2:

mov prov, 1;

jmp fnl2;

fnl2:;

}

if (lvl > 3 || lvl < 1) {

prov = 0;

}

}

printf("Введите ваше имя (< 15 символов) \n");

scanf("%" STRLEN(LEN) "s", name1);

file = fopen("Результаты.txt", "a");

fprintf(file, "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

setlocale(LC\_ALL, "Rus");

fprintf(file, "Имя/никнейм - %s \n", name1);

fclose(file);

DeleteObject(brush); //Очищаем память от созданной кисти

ReleaseDC(hwnd, dc);

stopwatch = 1;

mazegame(height, width, stopwatch, 1, lvl);

\_\_asm {

mov eax, y11;

mov ebx, y22;

cmp eax, 215;

jle endwarning;

sub eax, 140;

cmp ebx, 235;

jle endwarning;

sub ebx, 140;

mov y11, eax;

mov y22, ebx;

endwarning:;

}

break;

case 4:

system("cls");

rules();

\_getch();

if (\_kbhit()) \_getch();

\_\_asm {

mov eax, y11;

mov ebx, y22;

cmp eax, 215;

jle endw;

sub eax, 210;

cmp ebx, 235;

jle endw;

sub ebx, 210;

mov y11, eax;

mov y22, ebx;

endw:;

}

break;

case 5:

system("cls");

exit(0);

}

system("cls");

graphicsofmenu();

p = 1;

break;

case KEY\_UP:

if (p > 1) {

brush = CreateSolidBrush(RGB(20, 100, 200)); // Не выбранный элемент

SelectObject(dc, brush); //Выбираем кисть

Ellipse(dc, x11, y11, x22, y22);

DeleteObject(brush);

p--;

y11 -= 70;

y22 -= 70;

brush = CreateSolidBrush(RGB(255, 204, 102)); // Выбранный элемент

SelectObject(dc, brush); //Выбираем кисть

Ellipse(dc, x11, y11, x22, y22);

DeleteObject(brush); //Очищаем память от созданной кисти

}

break;

case KEY\_DOWN:

if (p < 5) {

brush = CreateSolidBrush(RGB(20, 100, 200)); // Не выбранный элемент

SelectObject(dc, brush); //Выбираем кисть

Ellipse(dc, x11, y11, x22, y22);

DeleteObject(brush); //Очищаем память от созданной кисти

p++;

y11 += 70;

y22 += 70;

brush = CreateSolidBrush(RGB(255, 204, 102)); // Выбранный элемент

SelectObject(dc, brush); //Выбираем кисть

Ellipse(dc, x11, y11, x22, y22);

DeleteObject(brush);

}

break;

}

}

}

**Файл с результатами игры**

Файл: Результаты.txt

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Имя/никнейм - Maksim

Размеры лабиринта - 21 x 21

Кол-во собранных монеток - 3

Время прохождения - 28,00 сек.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*