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:

Муромский Д. А.

Принял:

Митрохин М. А.

Пенза 2020

# курсовая-1

# курсовая-2

# Реферат

Отчёт 60 стр, 15 рисунков.

ГРАФ, ТЕОРИЯ ГРАФОВ, ПОИСК В ГЛУБИНУ И ШИРИНУ.

Цель исследования - разработка игрового клиента и агента для игры «Лабиринт», используя алгоритмы поиска путей в графе.

В работе рассмотрены правила построения матрицы лабиринта алгоритм генерации лабиринта, создания логики агента с помощью алгоритмов поиска в глубину.

# Введение

Алгоритм поиска (или обхода) в глубину (англ. depth-first search, DFS) позволяет построить обход ориентированного или неориентированного графа, при котором посещаются все вершины, доступные из начальной вершины.

Отличие поиска в глубину от поиска в ширину заключается в том, что (в случае неориентированного графа) результатом алгоритма поиска в глубину является некоторый маршрут, следуя которому можно обойти последовательно все вершины графа, доступные из начальной вершины. Этим он принципиально отличается от поиска в ширину, где одновременно обрабатывается множество вершин, в поиске в глубину в каждый момент исполнения алгоритма обрабатывается только одна вершина.

С другой стороны, поиск в глубину не находит кратчайших путей, зато он применим в ситуациях, когда граф неизвестен целиком, а исследуется каким-то автоматизированным устройством.

Если же граф ориентированный, то поиск в глубину строит дерево путей из начальной вершины во все доступные из неё.

### В качестве среды разработки мною была выбрана среда Qt Creator 4.13.1, язык программирования – Си/С++, язык разметки веб-страниц HTML/CSS.

Целью данной курсовой работы является разработка программы на языке Си/C++, который является широко используемым. Именно с его помощью в данном курсовом проекте реализуется алгоритм поиска в глубину, осуществляющий поиск пути для агента.

# 1 Постановка задачи

Требуется разработать игровой клиента и агента для игры «Лабиринт», используя алгоритмы поиска путей в графе.

Исходные данные в программе представлены в виде двоичный матрицы, где «1» это проход, а «0» это стена. Пользователь может выбирать, путём нажатия на кнопки, размер лабиринта и точность (для пользователя сложность) работы игрового агента. Далее лабиринт должен генерироваться и обрабатываться игровым агентом, а после отображаться на экране. Также необходимо предусмотреть различные исходы генерации и поиска пути, чтобы программа не выдавала ошибок и работала правильно. Устройство ввода – клавиатура и мышь.

# 2 Теоретическая часть задания

В программе реализованы алгоритмы обхода в глубину и в ширину.

**2.1 Обход в глубину**

Идея такого обхода состоит в том, чтобы начав обход из какой-либо вершины всегда переходить по первой встречающейся в процессе обхода связи в следующую вершину, пока существует такая возможность. Как только в процессе обхода исчерпаются возможности прохода, необходимо вернуться на один шаг назад и найти следующий вариант продвижения. Таким образом, итерационно выполняя описанные операции, будут пройдены все доступные для прохождения вершины. Чтобы не заходить повторно в уже пройденные вершины, необходимо их пометить как пройденные.

Таким образом, можно предложить следующую рекурсивную реализацию алгоритма обхода в глубину.

**Вход**: G – матрица смежности графа.

**Выход:** номера вершин в порядке их прохождения на экране.

**Алгоритм ПОГ**

1.1. для всех i положим NUM[i] = False пометим как "не посещённую";

1.2. **ПОКА** существует "новая" вершина v

1.3. **ВЫПОЛНЯТЬ** DFS (v).

**Алгоритм** DFS(v):

2.1. пометить v как "посещённую" NUM[v] = True;

2.2. вывести на экран v;

2.3. **ДЛЯ** i = 1 **ДО** size\_G **ВЫПОЛНЯТЬ**

2.4. **ЕСЛИ** G(v,i) = = 1**И** NUM[i] = = False

2.5. **ТО**

2.6. {

2.7. DFS(i);

2.8. }

Реализация состоит из подготовительной части, в которой все вершины помечаются как не помеченные (п.1.1) и осуществляется запуск процедуры обхода для вершин графа (п.1.2, 1.3). И непосредственно процедуры обхода, которая помечает текущую (т.е. ту, в которой на текущей итерации находится алгоритм) вершину как посещённую (п. 2.1). Затем выводит номер текущей вершины на экран (п.2.2) и в цикле просматривает **v**-ю строку матрицы смежности графа G(v,i). Как только алгоритм встречает смежную с **v** не посещённую вершину (п.2.4), то для этой вершины вызывается процедура обхода (п.2.7).

**2.2 Обход в ширину**

Основная идея такого обхода состоит в том, чтобы посещать вершины по уровням удаленности от исходной вершины. Удалённость в данном случае понимается как количество ребер, по которым необходимо прейти до достижения вершины. Например, если для графа на рисунке 1 начать обход из первой вершины, то вершины 3, 6 и 2 будут находиться на уровне удаленности в 1 ребро, а вершины 5 и 4 на уровне удаленности в 2 ребра.



Рисунок 1 - Граф

Тогда при обходе этого графа в ширину, мы сначала посетим вершины первого уровня удалённости (с номерами 2, 3 и 6), и только после того, как закончатся не посещённые вершины на этом уровне, мы перейдём к следующему. На втором уровне мы посетим все вершины, которые удалены от исходной на 2 ребра (вершины 4 и 5).

Так, алгоритм обхода в ширину продолжает осматривать уровень за уровнем, пока не пройдет все доступные вершины.

Чтобы не заходить повторно в уже пройденные вершины, они помечаются, как и в алгоритме обхода в глубину.

Для того, чтобы проход осуществлялся по уровням необходимо хранить информацию о требуемом порядке посещения вершин. Вершины, которые являются ближайшими соседями исходной вершины (из которой начат обход) должны быть посещены раньше, чем соседи соседей и т.д. Такой порядок позволяет задать структура данных «очередь». Просматривая строку матрицы смежности (или список смежности) для текущей вершины мы помещаем всех её ещё не посещённых соседей в очередь. На следующей итерации текущей вершиной становится та, которая стоит в очереди первой и уже её не посещённые соседи будут помещены в очередь. Но место в очереди они займут после тех вершин, которые были помещены туда на предыдущих итерациях.

Таким образом, можно предложить следующую реализацию алгоритма обхода в ширину.

**Вход**: G – матрица смежности графа.

**Выход**: номера вершин в порядке их прохождения на экране.

**Алгоритм ПОШ**

1.1. для всех i положим NUM[i] = False пометим как "не посещённую";

1.2. **ПОКА** существует "новая" вершина v

1.3. **ВЫПОЛНЯТЬ** BFS (v).

**Алгоритм** BFS(v):

2.1. Создать пустую очередь Q = {};

2.2. Поместить v в очередь Q.push(v);

2.3. пометить v как "посещённую" NUM[v] = True;

2.4. **ПОКА**  Q != ∅ очередь не пуста **ВЫПОЛНЯТЬ**

2.5. v = Q.front() установить текущую вершину;

2.6. Удалить первый элемент из очереди Q.pop();

2.7. вывести на экран v;

2.8. **ДЛЯ** i = 1 **ДО** size\_G **ВЫПОЛНЯТЬ**

2.9. **ЕСЛИ** G(v,i) = = 1**И** NUM[i] = = False

2.10. **ТО**

2.11. Поместить i в очередь Q.push(i);

2.12. пометить v как "посещённую" NUM[v] = True;

Реализация состоит из подготовительной части, в которой все вершины помечаются как не посещённые (п.1.1) и осуществляется запуск процедуры обхода для вершин графа (п.1.2, 1.3).

В самой процедуре обхода сначала создаётся пустая очередь (п. 2.1), в которую помещается исходная вершина, из которой начат обход (п.2.2).

Далее итерационно, пока очередь не опустеет, из неё извлекается первый элемент, который становится текущей вершиной (п. 2.5, 2.6). Затем в цикле просматривается **v**-я строка матрицы смежности графа G(v,i). Как только алгоритм встречает смежную с **v** не посещённую вершину (п.2.9), эта вершина помещается в очередь (п.2.11) и помечается как посещённая (п.2.12). После просмотра строки матрицы смежности алгоритм делает следующую итерацию цикла 2.4 или заканчивает работу, если очередь пуста.

# 3 Описание алгоритма программы

Алгоритм программы показан на рисунке 2.



Рисунок 2 - Алгоритм работы программы

**3.1 Генерация пустой матрицы**

Генерируются 2 матрицы, labirint с самим лабиринтом и visible для хранения пройденных вершин. labirint заполняется стенками (0) , а visible заполняется false и крайние стенки лабиринта true.



Рисунок 3 - Объявление переменных

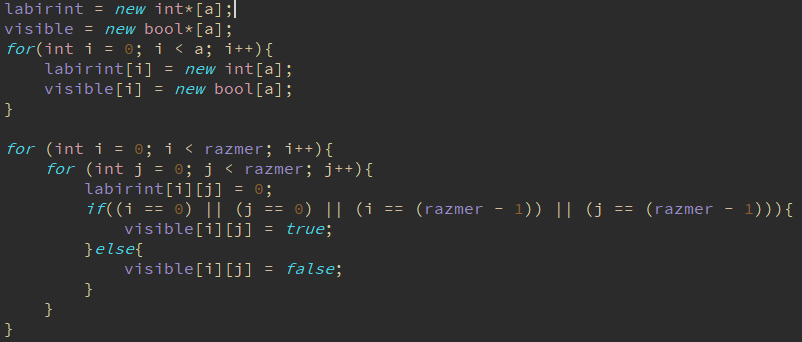


Рисунок 4 - Создания и заполнения матриц

**3.2 Генерация лабиринта (Крот)**

В программе используется алгоритм генерации лабиринта основанный на алгоритме Олдоса-Бродера. Алгоритм Олдоса-Бродера избавлен от какого-либо смещения. Все лабиринты, получаемые с его помощью, абсолютно случайны и не похожи друг на друга. Он не имеет предпочтений по направленности, запутанности или ещё каким-либо характеристикам. Результирующие лабиринты случайны и равновероятны. Его создали два независимых исследователя, изучавших равновероятные варианты остовных деревьев: Дэвид Олдос, профессор Калифорнийского университета Беркли и Андрей Бродер, учёный, ныне работающий в Google. Кроме лабиринтов, алгоритм часто всплывает в работах о математической вероятности, что неудивительно, учитывая принцип его работы.

**Формальный алгоритм:**

1. Выбрать случайную вершину (клетку). Абсолютно случайную;
2. Выбрать случайную соседнюю вершину (клетку) и перейти в неё. Если она не была посещена, добавить её в дерево (соединить с предыдущей, убрать между ними стену);
3. Повторять шаг 2, пока все клетки не будут посещены.

**Плюсы:**

* Отсутствует какое-либо смещение;
* Лабиринты абсолютно случайны, поэтому невозможно создать определённый алгоритм их решения;
* Сложность решения для человека;
* Простая реализация;

**Минусы:**

* Скорость генерации лабиринта, может генерироваться очень долго, потому как алгоритм основан на вероятности;
* Не позволяет генерировать бесконечные лабиринты;
* Сильное падение эффективности под конец генерации;

Алгоритм немного модифицирован под конкретную задачу, так как изначально он рассчитан под генерацию лабиринта с тонкими стенками (стенка находится между клетками прохода, отдельная клетка). Основные изменения:

1. Алгоритм «копает» на 2 клетки, а не на 1.
2. Так как из-за пунка 1) алгоритм «копает» только по нечётным клеткам (1,1),(1,3)(3,1)(3,3), то и нет смысла проверять весь лабиринт на посещёность, достаточно проверить только нечётные клетки.

**3.3 Создание матрицы весов**

С помощью алгоритм обхода графа в ширину заполняется все клетки с проходом определённым весом. Например, выход будет иметь вес «0», соседняя клетка к выходу вес «1», а соседние к ней клетки «2» и тд. Данная матрицы весов поможет боту найти выход из лабиринта.

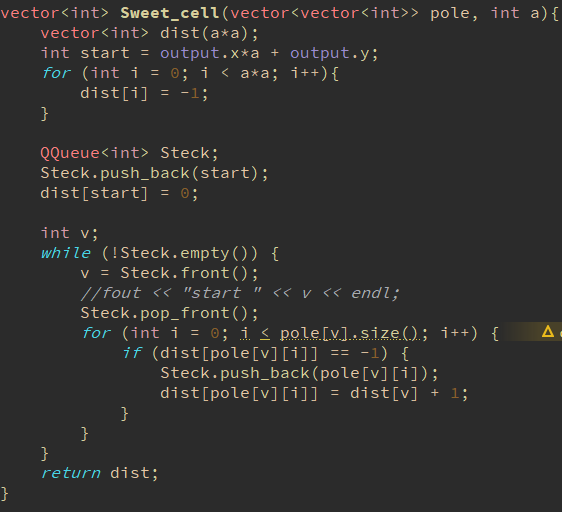


Рисунок 5 - Код функции создания матрицы весов

**3.4 Создание списка смежности**

****Список смежности****представляет граф в виде массива связанного списка. Индекс массива представляет вершину и каждый элемент в его связанном списке, а также представляет другие вершины, которые образуют ребро с вершиной.

## Представление списка смежности

Граф и его эквивалентное представление списка смежности показаны ниже.

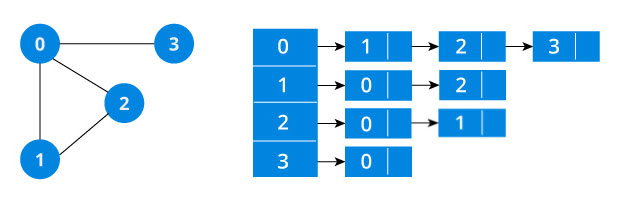
[](https://evileg.com/users/mafulechka/albums/photo/681/)

Рисунок 6 - представление графа в виде списка смежности

Список смежности эффективен с точки зрения хранения, потому что нам нужно хранить только значения для рёбер. Для разреженного графа с миллионами вершин и рёбер это может означать много сэкономленного пространства.

В программе в качестве списка смежности используется конструкция vector<vector<int>>.

**3.5 Алгоритм работы бота**

Бот использует алгоритм, который помогает ему казаться более «живым», он не знает где выход, лишь предполагает и это помогает ему ошибаться, чтобы пользователю не казалось, что бот знает где выход. Достигается это с помощью матрицы весов и списка смежности.

В зависимости от выбранной сложности бот имеет разный радиус обзора, чем он меньше, тем робот точнее идёт к выходу. Но есть некоторые ограничения: если поставить слишком маленькое значение или слишком большое, то бот будет идти прямо к выходу, без всяких ошибок.

Алгоритм:

1. Бот ищет в заданном радиусе клетку с самым маленьким весом (если их несколько, то выбирается случайная).

Блок-схема функции Seach\_point на рисунке 7.

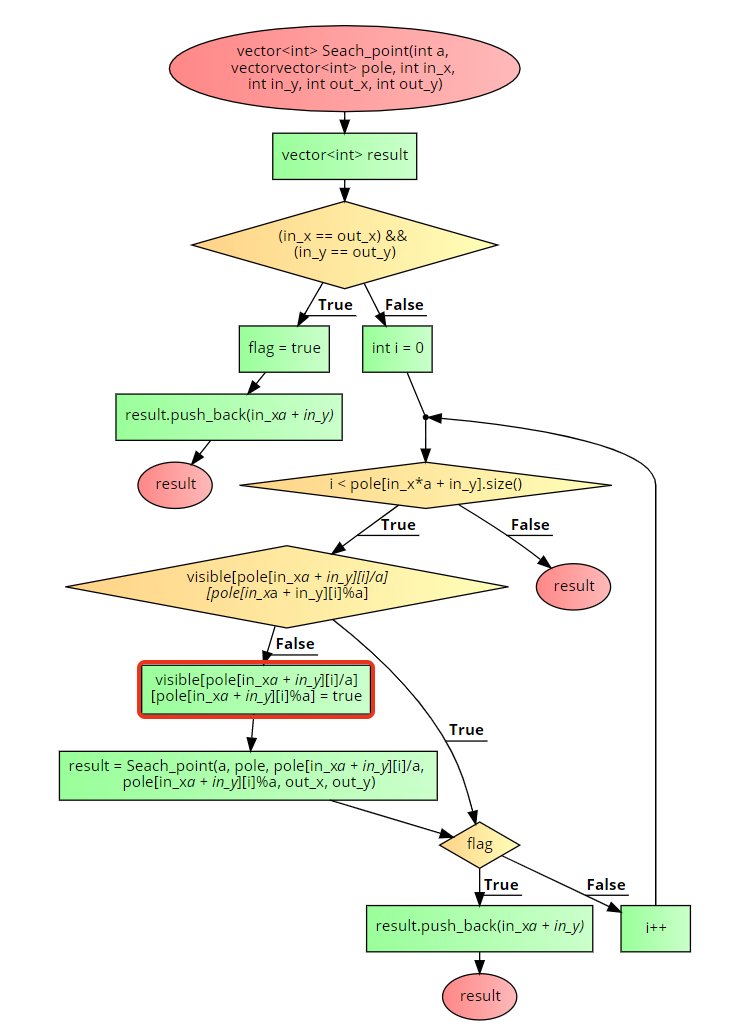


Рисунок 7 - Блок-схема функции Seach\_point

Код функции Seach\_point приведён ниже:

bool flag(false);

vector<int> Seach\_point(int a, vector<vector<int>> pole, int in\_x,int in\_y,int out\_x, int out\_y){

vector<int> result;

if ((in\_x == out\_x) && (in\_y == out\_y)){

flag = true;

result.push\_back(in\_x\*a + in\_y);

return result;

}

for (int i = 0; i < pole[in\_x\*a + in\_y].size(); i++){

if (!visible[pole[in\_x\*a + in\_y][i]/a][pole[in\_x\*a + in\_y][i]%a]){

visible[pole[in\_x\*a + in\_y][i]/a][pole[in\_x\*a + in\_y][i]%a] = true;

result = Seach\_point(a, pole, pole[in\_x\*a + in\_y][i]/a, pole[in\_x\*a + in\_y][i]%a, out\_x, out\_y);

}

if (flag){

result.push\_back(in\_x\*a + in\_y);

return result;

}

}

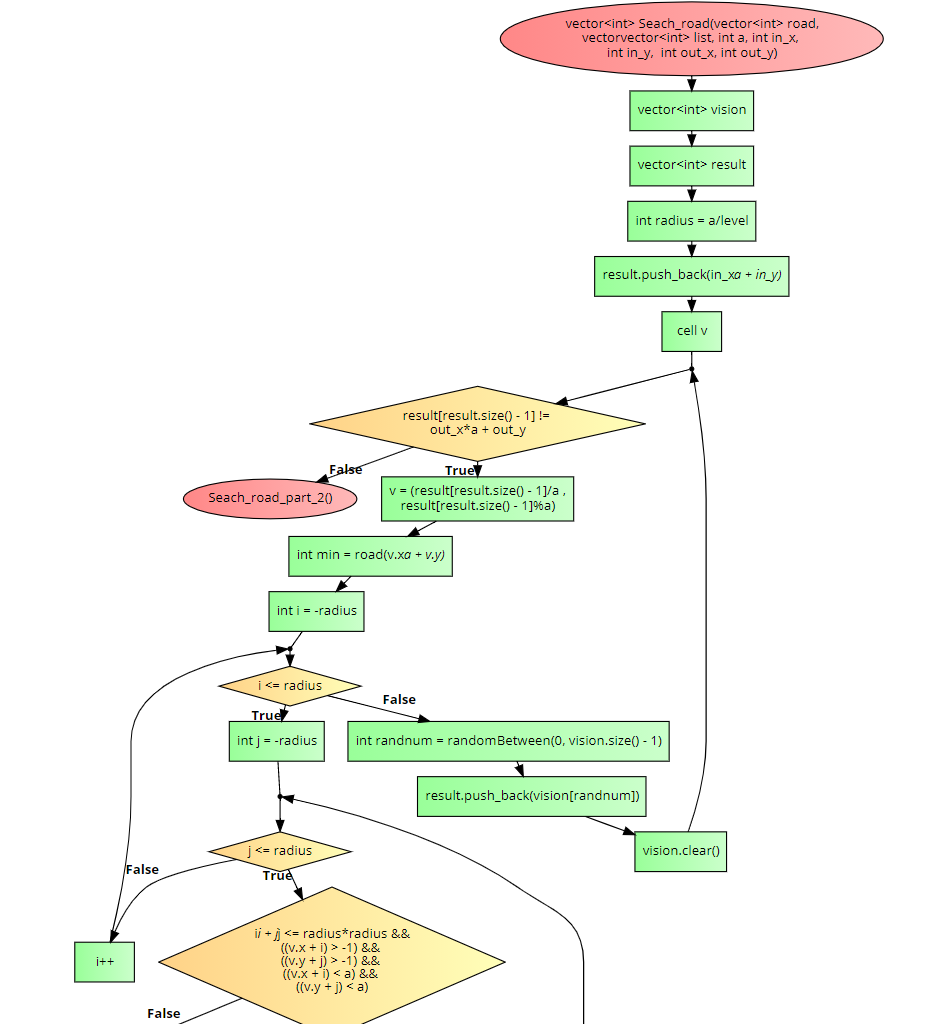
return result;

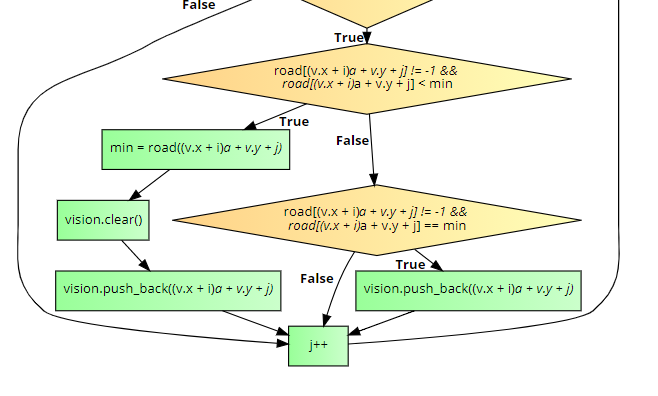
}

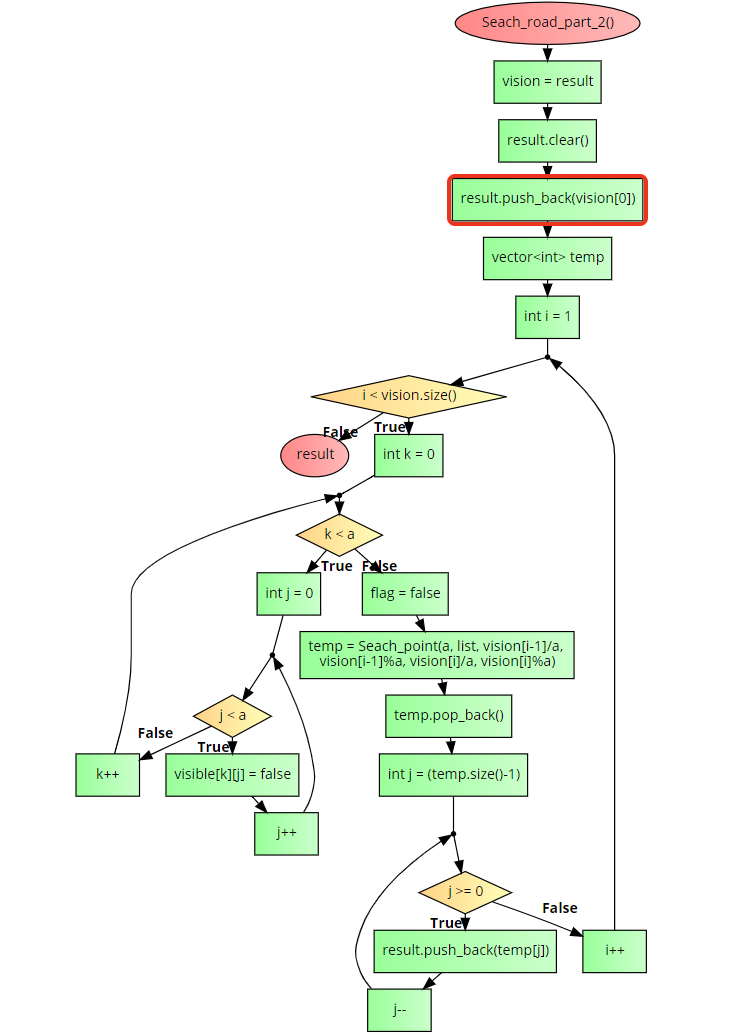
1. Пункт 1) выполняется до тех пор пока не будет найден выход, на выходе данного алгоритма список клеток в которые должен попасть бот.
2. Запускается алгоритм поиска пути методом обхода графа в глубину по спискам смежности.

После выполнения данного алгоритма на выходе получается цепочка точек, по которым пойдёт бот.

Блок-схема:







Код:

vector<int> **Seach\_road**(vector<int> road, vector<vector<int>> list, int a, int in\_x, int in\_y, int out\_x, int out\_y){

vector<int> vision;

vector<int> result;

int radius = a/level;

result.push\_back(in\_x\*a + in\_y);

cell v;

*while*(result[result.size() - 1] != (out\_x\*a + out\_y)){

v = {result[result.size() - 1]/a , result[result.size() - 1]%a};

int min = road[v.x\*a + v.y];

*for* (int i = -radius; i <= radius; i++){

*for*(int j = -radius; j <= radius; j++){

*if*(((i\*i + j\*j) <= (radius\*radius)) && ((v.x + i) > -1) && ((v.y + j) > -1) && ((v.x + i) < a) && ((v.y + j) < a)){

*if* ((road[(v.x + i)\*a + v.y + j] != -1) && (road[(v.x + i)\*a + v.y + j] < min)){

min = road[(v.x + i)\*a + v.y + j];

vision.clear();

vision.push\_back((v.x + i)\*a + v.y + j);

}*else*{

*if* ((road[(v.x + i)\*a + v.y + j] != -1) && (road[(v.x + i)\*a + v.y + j] == min)){

vision.push\_back((v.x + i)\*a + v.y + j);

}

}

}

}

}

int randnum = randomBetween(0, vision.size() - 1);

result.push\_back(vision[randnum]);

vision.clear();

}

vision = result;

result.clear();

result.push\_back(vision[0]);

result.push\_back(vision[0]);

result.push\_back(vision[0]);

result.push\_back(vision[0]);

result.push\_back(vision[0]);

result.push\_back(vision[0]);

result.push\_back(vision[0]);

result.push\_back(vision[0]);

vector<int> temp;

*for* (int i = 1; i < vision.size(); i++){

*for* (int k = 0; k < a; k++){

*for* (int j = 0; j < a; j++){

visible[k][j] = *false*;

}

}

flag = *false*;

temp = Seach\_point(a, list, vision[i-1]/a, vision[i-1]%a, vision[i]/a, vision[i]%a);

temp.pop\_back();

*for* (int j = (temp.size()-1); j >= 0; j--){

result.push\_back(temp[j]);

}

}

*return* result;

}

Скриншоты на рисунках ..... демонстрируют работу алгоритма.

Пример работы бота: <https://youtu.be/6WwZX2VvpBI>

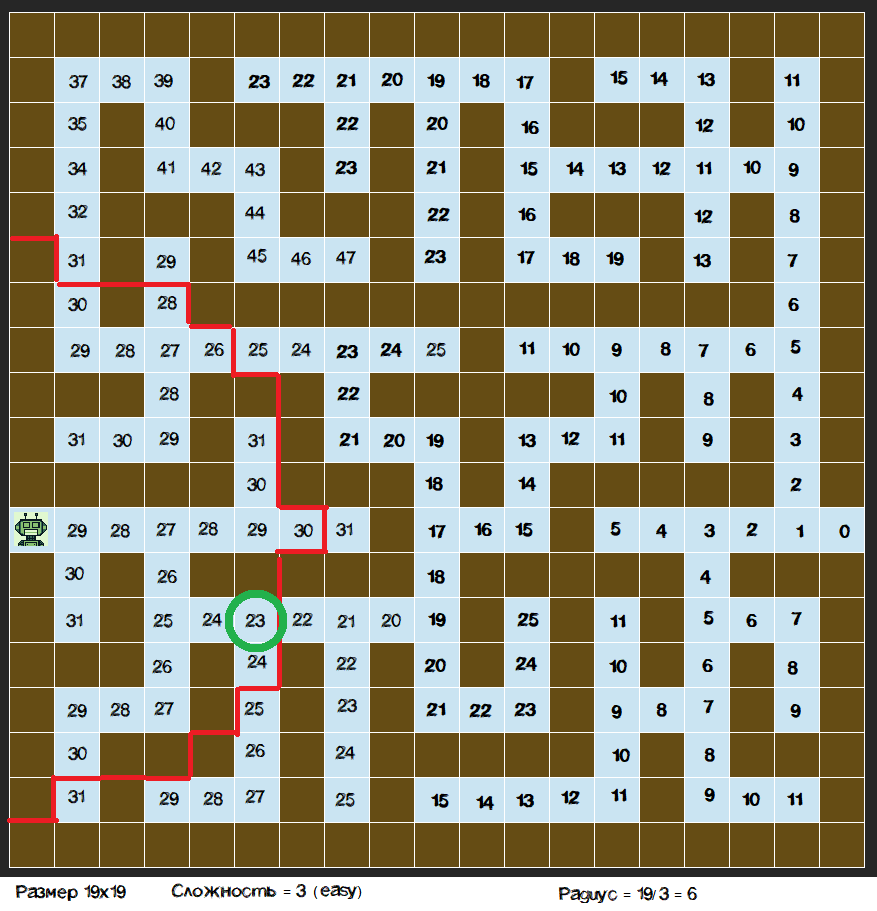


Рисунок 8 - Радиус обзора бота и зелёная клетка с минимальным весом



Рисунок 9 - Радиус обзора из зелёной клетки и синяя клетка с минимальным весом



Рисунок 10 - Радиус обзора из синей клетки и оранжевая клетка с минимальным весом



Рисунок 11 - Радиус обзора из оранжевой клетки и красная клетка с минимальным весом (выход)

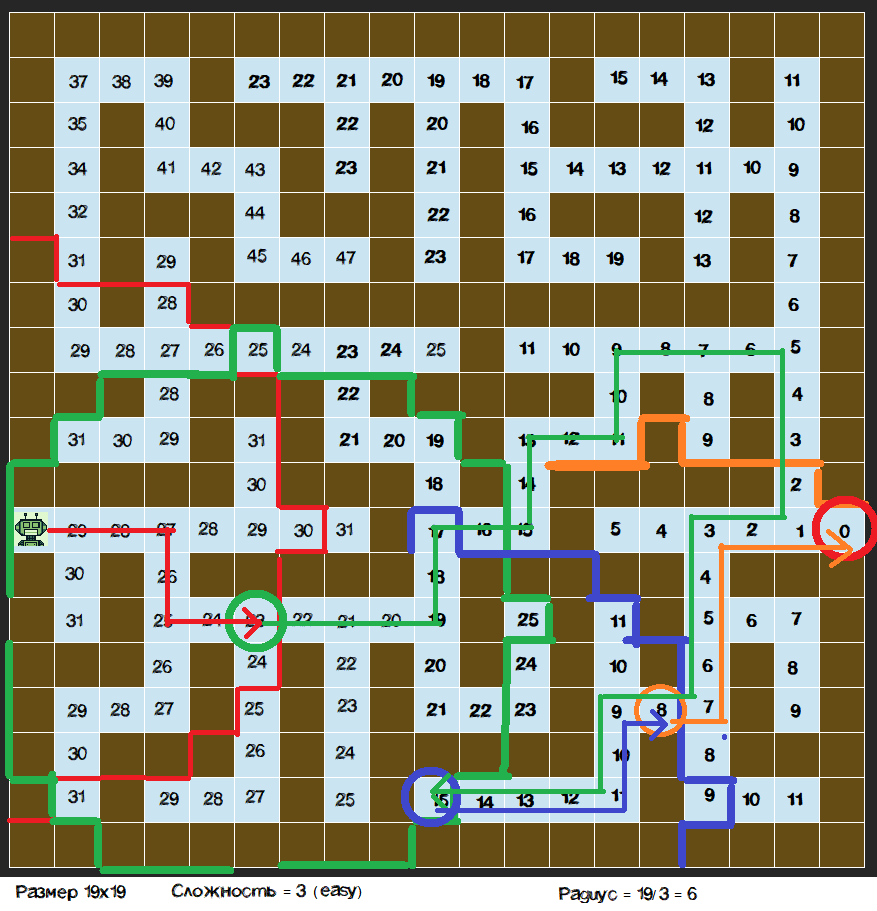


Рисунок 12 - Отображение построение пути бота

На выходе получится путь (строка,столбец):

(12,1) -> (12,2) -> (12,3) -> (12,4) ->

(13,4) -> (14,4) ->

(14,5) -> (14,6) -> (14,7) -> (14,8) -> (14,9) -> (14,10) ->

(13,10) -> (12,10) ->

(12,11) -> (12,12) ->

(11,12) -> (10,12) ->

(10,13) -> (10,14) ->

(9,14) -> (8,14) ->

(8,15) -> (8,16) -> (8,17) -> (8,18) ->

(9,18) -> (10,18) -> (11,18) -> (12,18) ->

(12,17) -> (12,16) ->

(13,16) -> (14,16) -> (15,16) -> (16,16) ->

(16,15) -> (16,14) ->

(17,14) -> (18,14) ->

(18,13) -> (18,12) -> (18,11) -> (18,10) ->

(18,11) -> (18,12) -> (18,13) -> (18,14) ->

(17,14) -> (16,14) ->

(16,15) -> (16,16) ->

(15,16) -> (14,16) -> (13,16) -> (12,16) ->

(12,17) -> (12,18) -> (12,19)

# 4 Описание программы

Для написания данной программы использован язык программирования С/С++, язык разметки веб-страниц HTML/CSS. Язык программирования С/С++ - универсальный язык программирования, который завоевал особую популярность у программистов, благодаря сочетанию возможностей языков программирования высокого и низкого уровней. HTML — стандартизированный язык разметки веб-страниц во [Всемирной паутине](https://ru.wikipedia.org/wiki/%D0%92%D1%81%D0%B5%D0%BC%D0%B8%D1%80%D0%BD%D0%B0%D1%8F_%D0%BF%D0%B0%D1%83%D1%82%D0%B8%D0%BD%D0%B0). Код HTML интерпретируется [браузерами](https://ru.wikipedia.org/wiki/%D0%91%D1%80%D0%B0%D1%83%D0%B7%D0%B5%D1%80); полученная в результате интерпретации страница отображается на экране монитора компьютера или мобильного устройства.

Проект был создан в виде графического приложения Win64 на языке C++/Qt в кроссплатформенной IDE для разработки программного обеспечения Qt Creator 4.13.1. Qt позволяет запускать написанное с его помощью [программное обеспечение](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D0%BE%D0%B5_%D0%BE%D0%B1%D0%B5%D1%81%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%B8%D0%B5) в большинстве современных [операционных систем](https://ru.wikipedia.org/wiki/%D0%9E%D0%BF%D0%B5%D1%80%D0%B0%D1%86%D0%B8%D0%BE%D0%BD%D0%BD%D0%B0%D1%8F_%D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0) путём простой компиляции программы для каждой системы без изменения [исходного кода](https://ru.wikipedia.org/wiki/%D0%98%D1%81%D1%85%D0%BE%D0%B4%D0%BD%D1%8B%D0%B9_%D0%BA%D0%BE%D0%B4). Включает в себя все основные [классы](https://ru.wikipedia.org/wiki/%D0%9A%D0%BB%D0%B0%D1%81%D1%81_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)), которые могут потребоваться при разработке [прикладного программного обеспечения](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%B8%D0%BA%D0%BB%D0%B0%D0%B4%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D0%BE%D0%B5_%D0%BE%D0%B1%D0%B5%D1%81%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%B8%D0%B5), начиная от элементов [графического интерфейса](https://ru.wikipedia.org/wiki/%D0%93%D1%80%D0%B0%D1%84%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B8%D0%B9_%D0%B8%D0%BD%D1%82%D0%B5%D1%80%D1%84%D0%B5%D0%B9%D1%81_%D0%BF%D0%BE%D0%BB%D1%8C%D0%B7%D0%BE%D0%B2%D0%B0%D1%82%D0%B5%D0%BB%D1%8F) и заканчивая классами для работы с [сетью](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%BF%D1%8C%D1%8E%D1%82%D0%B5%D1%80%D0%BD%D0%B0%D1%8F_%D1%81%D0%B5%D1%82%D1%8C), [базами данных](https://ru.wikipedia.org/wiki/%D0%91%D0%B0%D0%B7%D0%B0_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85) и [XML](https://ru.wikipedia.org/wiki/XML). Является полностью объектно-ориентированным, расширяемым и поддерживающим технику компонентного программирования.

Работа программы начинается с запуска лаунчера и запроса от пользователя выбрать размер и сложность лабиринта. Для ввода значений нужно нажать на соответствующие кнопки, для запуска же игры нужно нажать «Начать играть».

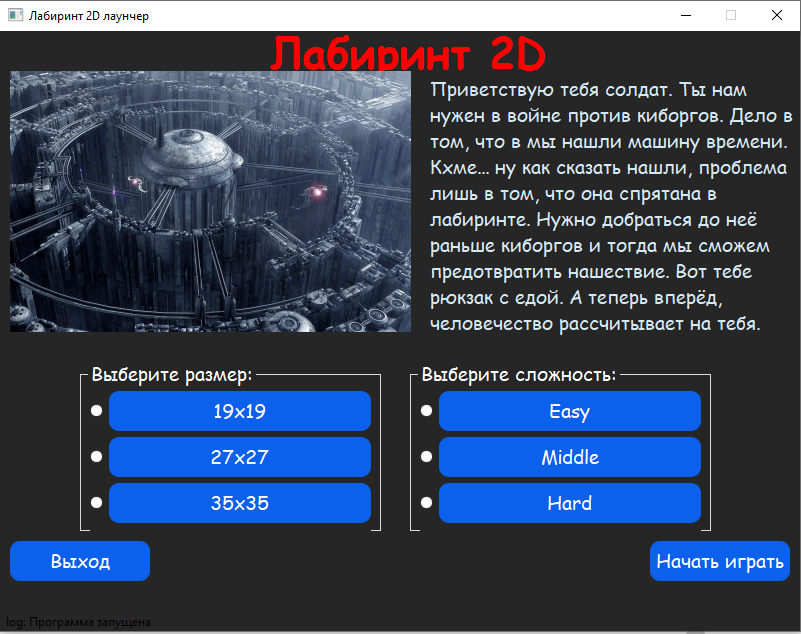


Рисунок 13 - Лаунчер игры

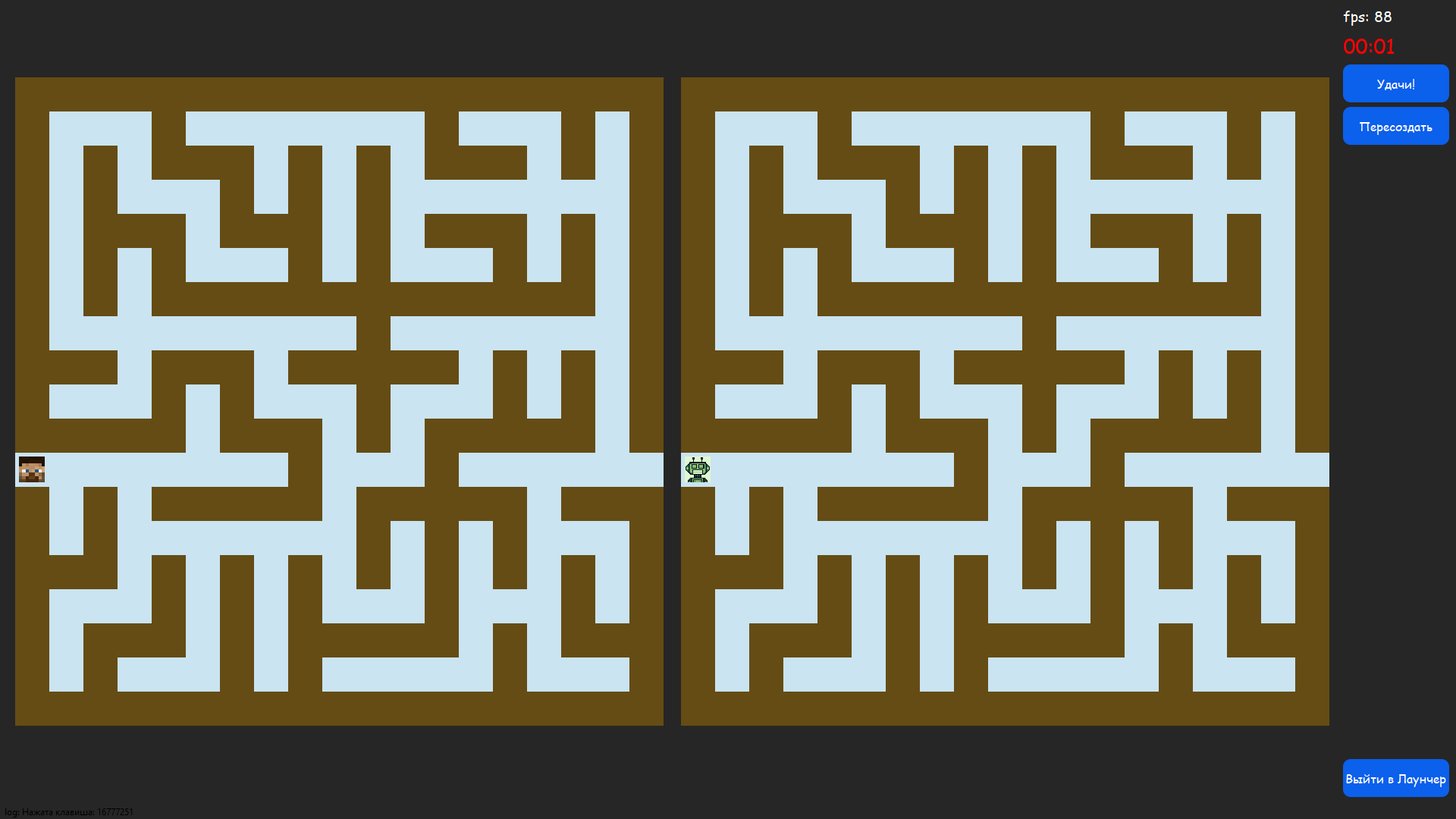


Рисунок 14 - Окно игры при запуске

В открывшемся окне можно увидеть 3 зоны: слева расположено поле с лабиринтом и иконкой игрока, которым может управлять пользователь; посередине находится поле с лабиринтом и иконкой робота, которым управляет бот; справа расположены кнопки для управления «Пересоздать» и «Выйти в лаунчер». Кнопка «Пересоздать» отвечает за пересоздание лабиринта для игрока и пользователя, «Выйти в лаунчер» предназначена для выхода из игры и возврат к лаунчеру. Также в правой зоне можно увидеть таймер и счётчик fps («****F****rame-****P****er-****S****econd» — кадры-в-секунду), показывающий кол-во отрисовок за 1 секунду.

Цель игры: дойти до выхода быстрее, чем бот.

В игре можно управлять аватаром (игровой персонаж под контролем игрока) путём нажатия на соответствующие клавиши:

1. «W» , «Ц» , «Key\_Up» - для движения вверх.
2. «A» , «Ф» , «Key\_Left» - для движения влево.
3. «D» , «В» , «Key\_Rigth» - для движения вправо.
4. «S» , «Ы» , «Key\_Down» - для движения вниз.
5. «Key\_Space» - для пересоздания лабиринта.

Соревнуясь с ботом может быть 2 исхода: победа или поражение, которым соответствуют выводы сообщений в игре:

Пример игрового геймплея: <https://youtu.be/zeztxCumkiU>

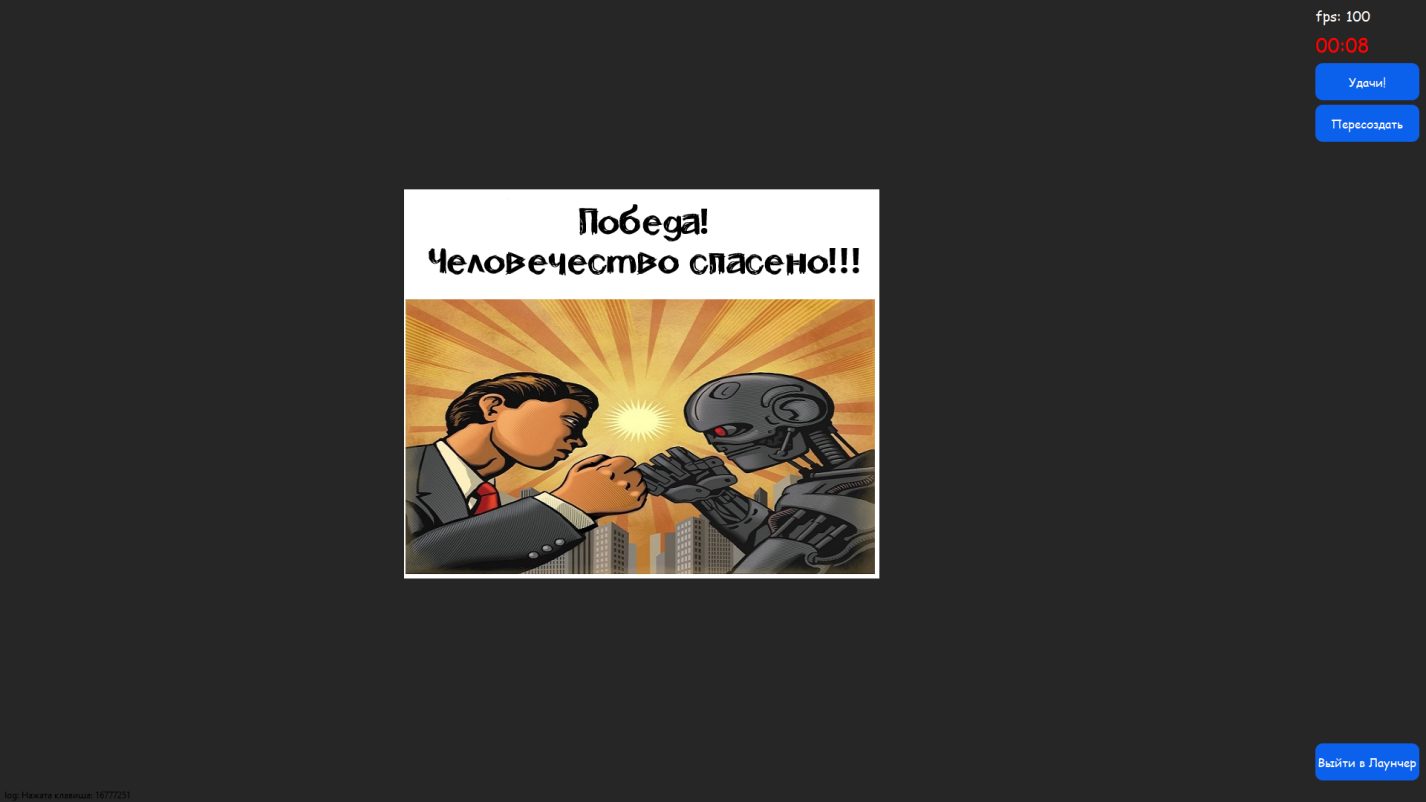


Рисунок 15 - Сообщение при победе



Рисунок 16 - Сообщение при поражении

# 5 Тестирование

Среда разработки Qt Creator 4.13.1 предоставляет все средства, необходимые при разработке и отладке многомодульной программы.

Тестирование проводилось в рабочем порядке, в процессе разработки, после завершения написания программы. В ходе тестирования было выявлено и исправлено множество проблем, в основном связанных с «утечкой» памяти из-за неопытности при работе с ООП, также были проблемы с отрисовкой, из-за чего пришлось полностью переписывать функцию отрисовки разделив её на 2 различные.

Видео с тестированием программы: <https://youtu.be/H-Ndrgjoy-U>

Таблица 1 - План и результаты тестирования

|  |  |  |  |
| --- | --- | --- | --- |
| **Описание теста** | **Действия** | **Ожидаемый результат** | **Полученный результат** |
| Запуск программы | Запуск Файла test\_v01.exe средствами операционной системы | Открытие формы | Верно |
| Старт без выбранных параметров | Нажатие кнопки «Начать играть» , без выбранных параметров «Размер» и «Сложность» | Игра не запустится | Верно |
| Старт с выбранными параметрами | Нажатие кнопки «Начать играть» , с выбранными параметрами «Размер» и «Сложность» | Игра запустится и отобразится игровое поле с 2 лабиринтами | Верно |
| Нажатие клавиш перемещения | Нажимать клавиши перемещения, описанные в пункте 4 | Персонаж должен перемещаться по лабиринту в направлении, которое задают клавиши, и не двигаться, если на пути стена | Верно |
| Пересоздание лабиринта | Пересоздание лабиринта с помощью клавиши «Пересоздать» или нажатием на клавишу «Space» | Лабиринт должен пересоздаваться и персонажи (робот и аватар) вернуться на исходные позиции | Верно |
| Пробовать победить/проиграть | Сначала побеждаем у бота, а затем проигрываем | Должны выводиться соответствующие сообщения на победу и поражение | Верно |
| Выход из программы | Выходим из игры в лаунчер путём нажатия на кнопки «Выйти в лаунчер» и также выходим из лаучера нажатием на кнопки «Выход» | Корректное закрытие программы | Верно |

Для примера показан результат обработки случаю пересоздания лабиринта. Аналогично проведено тестирование согласно всем пунктам плана.

# Заключение

Таким образом, в процессе создания данного проекта разработана игра «Лабиринт 2D» и бот для её прохождения в Qt Creator 4.13.1.

При выполнении данной курсовой работы были получены навыки разработки программ в Qt и освоены приёмы создания списков смежностей, основанных на теории графов. Приобретены навыки по осуществлению алгоритма поиска в глубину. Углублены знания языка программирования Cи/С++, язык разметки веб-страниц HTML/CSS.

Недостатком разработанной программы является время для генерации и обработки лабиринта ботом. Лабиринт 41х41 - 3 сек , 61x61 - 7 сек и при 101х101 программа даже зависает на 20-25 секунд, а Windows выдаёт сообщение о том, что программа не отвечает, но через пару секунд всё восстанавливается.

Программа имеет небольшой, но достаточный для использования функционал возможностей.

# СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ

1 Различные алгоритмы генерации лабиринтов:

URL: <https://habr.com/ru/post/320140/>

URL: <https://habr.com/ru/post/321210/>

2 Используемый алгоритм с объяснением работы:

URL: <https://youtu.be/64f6DE5tED0>

3 Сайт с описание классов и их методов:

URL: <https://doc.qt.io/qt-5/>

4 Форум программистов и сисадминов Киберфорум [Электронный ресурс]: cyberforum.ru. - Режим доступа: <https://www.cyberforum.ru/qt/thread1697756.html> - Загл. с экрана.

# ПРИЛОЖЕНИЕ

# main.cpp

#include "mainwindow.h"

#include <QApplication>

#include <QGroupBox>

int main(int argc, char \*argv[])

{

QApplication a(argc, argv);

MainWindow w;

w.show();

*return* a.exec();

}

# mainwindow.h

#ifndef MAINWINDOW\_H

#define MAINWINDOW\_H

#include <QMainWindow>

#include "game.h"

QT\_BEGIN\_NAMESPACE

*namespace* Ui { *class* MainWindow; }

QT\_END\_NAMESPACE

*class* MainWindow : *public* QMainWindow

{

Q\_OBJECT

*public*:

MainWindow(QWidget \*parent = *nullptr*);

~MainWindow();

*private* *slots*:

void on\_pushButton\_clicked();

void on\_pushButton\_2\_clicked();

void on\_pushButton\_3\_clicked();

void on\_pushButton\_4\_clicked();

void on\_pushButton\_5\_clicked();

void on\_pushButton\_6\_clicked();

void on\_pushButton\_7\_clicked();

void on\_radioButton\_clicked();

void on\_radioButton\_2\_clicked();

void on\_radioButton\_3\_clicked();

void on\_radioButton\_6\_clicked();

void on\_radioButton\_5\_clicked();

void on\_radioButton\_4\_clicked();

void on\_pushButton\_8\_clicked();

*private*:

Ui::MainWindow \*ui;

game \*gamewindow;

};

#endif *//* *MAINWINDOW\_H*

# mainwindow.cpp

#include "mainwindow.h"

#include "ui\_mainwindow.h"

#include "game.h"

#include <QLabel>

#include <QGLWidget>

#include <QtOpenGL>

#include <QTimer>

MainWindow::MainWindow(QWidget \*parent):

QMainWindow(parent),

ui(*new* Ui::MainWindow)

{

ui->setupUi(*this*);

ui->statusbar->showMessage("log: Программа запущена");

}

MainWindow::~MainWindow()

{

*delete* ui;

}

int size\_lab = 0;

int level\_hard = 0;

void MainWindow::on\_pushButton\_clicked()

{

ui->radioButton->click();

}

void MainWindow::on\_pushButton\_2\_clicked()

{

ui->radioButton\_2->click();

}

void MainWindow::on\_pushButton\_3\_clicked()

{

ui->radioButton\_3->click();

}

void MainWindow::on\_pushButton\_4\_clicked()

{

ui->radioButton\_4->click();

}

void MainWindow::on\_pushButton\_5\_clicked()

{

ui->radioButton\_5->click();

}

void MainWindow::on\_pushButton\_6\_clicked()

{

ui->radioButton\_6->click();

}

void MainWindow::on\_pushButton\_7\_clicked()

{

*if* (size\_lab && level\_hard){

ui->statusbar->showMessage("log: Игра запущена...");

hide();

game \*gamewindow = *new* game();

gamewindow->showFullScreen();

gamewindow->Paint3D(size\_lab, level\_hard);

connect(gamewindow, &game::firstWindow, *this*, &MainWindow::show);

}*else*{

ui->statusbar->showMessage("log: Не все параметры введены.");

}

}

void MainWindow::on\_radioButton\_clicked()

{

size\_lab = 19;

}

void MainWindow::on\_radioButton\_2\_clicked()

{

size\_lab = 27;

}

void MainWindow::on\_radioButton\_3\_clicked()

{

size\_lab = 35;

}

void MainWindow::on\_radioButton\_6\_clicked()

{

level\_hard = 5;

}

void MainWindow::on\_radioButton\_5\_clicked()

{

level\_hard = 4;

}

void MainWindow::on\_radioButton\_4\_clicked()

{

level\_hard = 3;

}

void MainWindow::on\_pushButton\_8\_clicked()

{

qApp->quit();

}

# game.h

#ifndef GAME\_H

#define GAME\_H

#include <QMainWindow>

#include <QFont>

#include <QTimer>

#include <QTime>

#include <QGraphicsObject>

#include <QKeyEvent>

#include <QLayout>

#include <QLabel>

#include <QString>

#include <vector>

#include <QBrush>

#include <iostream>

#include <cstdlib>

#include <ctime>

#include <stack>

#include <fstream>

#include <QtMath>

#include "time.h"

#include <QByteArray>

#include <QFile>

#include <windows.h>

#include <QMessageBox>

#include <QInputDialog>

#include <QApplication>

#include <QFont>

#include <QSize>

#include <QPixmap>

#include <QQueue>

*namespace* Ui {

*class* game;

}

*class* game : *public* QMainWindow

{

Q\_OBJECT

*public*:

*explicit* game(QWidget \*parent = *nullptr*);

void Paint3D(int a, int b);

QPixmap \*pic\_hero = *new* QPixmap(":/images/face.png");

QPixmap \*pic\_win\_man = *new* QPixmap(":/images/Robowin.png");

QPixmap \*pic\_win\_robot = *new* QPixmap(":/images/Peoplewin.png");

QPixmap \*pic\_robot = *new* QPixmap(":/images/robot.gif");

QPixmap \*pic\_gen = *new* QPixmap(":/images/gen.png");

~game();

*signals*:

void firstWindow();

*private* *slots*:

void on\_pushButton\_clicked();

void on\_pushButton\_2\_clicked();

void on\_pushButton\_3\_clicked();

void slotAlarmTimer();

void slotAlartFps();

void Reset\_Scene();

void win\_message\_robot();

void win\_message\_man();

void move\_hero(int a, int move\_x, int move\_y);

void move\_robot(int a, int move\_x, int move\_y);

void navigator();

*private*:

Ui::game \*ui;

QGraphicsScene \*scene;

QTimer \*timer\_paint;

QTimer \*timer\_fps;

QTimer \*robot\_time;

*//* *QTimer* *\*timer\_*

void keyPressEvent(QKeyEvent \*event);

};

#endif *//* *GAME\_H*

# game.cpp

#include "game.h"

#include "ui\_game.h"

*using* *namespace* std;

int razmer;

int level;

int\*\* labirint; *//* *1* *-* *проход* *;* *0* *-* *стена;*

bool\*\* visible;

bool button\_3\_ready(*true*);

*//ofstream* *fout("log.txt");*

int fps = 0;

int time\_timer = 0;

QPen \*redpen = *new* QPen(Qt::red);

QPen \*whitepen = *new* QPen(Qt::white);

QPen \*transparentpen = *new* QPen(Qt::transparent);

QBrush \*fonBrush = *new* QBrush(QColor(38,38,38), Qt::SolidPattern);

QBrush \*greenBrush = *new* QBrush(QColor(66,165,23), Qt::SolidPattern);

QBrush \*blackBrush = *new* QBrush(Qt::black, Qt::SolidPattern); *//Dense3Pattern*

QBrush \*brownBrush = *new* QBrush(QColor(101,76,20), Qt::SolidPattern);

QBrush \*whiteBrush = *new* QBrush(Qt::white, Qt::SolidPattern);

QBrush \*blueBrush = *new* QBrush(QColor(7,80,243), Qt::SolidPattern);

QBrush \*grayBrush = *new* QBrush(QColor(202,228,242), Qt::SolidPattern);

QBrush \*redBrush = *new* QBrush(Qt::red, Qt::SolidPattern);

QGraphicsPixmapItem\* item\_hero;

QGraphicsPixmapItem\* item\_robot;

*struct* cell{

int x;

int y;

};

cell input, output;

cell hero = {0, 0};

cell robot = {0, 0};

int razmer\_h;

vector<int> road;

game::game(QWidget \*parent) :QMainWindow(parent), ui(*new* Ui::game){

ui->setupUi(*this*);

ui->statusbar->showMessage("log: Игра запущена...");

ui->graphicsView->setFrameShape(QFrame::NoFrame);

scene = *new* QGraphicsScene(ui->graphicsView);

ui->graphicsView->setScene(scene);

*this*->setFocus();

}

game::~game()

{

*delete* item\_hero;

*delete* item\_robot;

*delete* redpen;

*delete* whitepen;

*delete* greenBrush;

*delete* blackBrush;

*delete* brownBrush;

*delete* whiteBrush;

*delete* blueBrush;

*delete* grayBrush;

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

*delete* [] labirint[i];

}

*delete* [] labirint;

*delete* ui;

}

void delay()

{

QTime dieTime= QTime::currentTime().addSecs(1);

*while* (QTime::currentTime() < dieTime)

QCoreApplication::processEvents(QEventLoop::AllEvents, 100);

}

int randomBetween(int low, int high)

{

*return* (qrand() % ((high + 1) - low) + low);

}

void seach\_exit(int a){

vector<int> cell\_true;

*for* (int i = 1; i < (a - 1); i++){

*if* (labirint[i][a - 2]){

cell\_true.push\_back(i);

}

}

int ran = randomBetween(0 , cell\_true.size() - 1);

output.x = cell\_true[ran];

output.y = a - 1;

labirint[output.x][output.y] = 1;

}

bool labirint\_ready(int a){

*for* (int i = 1; i < a; i += 2){

*for* (int j = 1; j < a; j += 2){

*if* (visible[i][j] == *false*){

*return* *false*;

}

}

}

*return* *true*;

}

void mole(int a, int start\_x, int start\_y){

stack<int> Steck\_x;

stack<int> Steck\_y;

Steck\_x.push(start\_x);

Steck\_y.push(start\_y);

int count = 0;

button\_3\_ready = *false*;

*while* (!labirint\_ready(a)){

count++;

start\_x = Steck\_x.top();

start\_y = Steck\_y.top();

int rand\_move = randomBetween(0,3);

*switch* (rand\_move){

*case* 0:

*if* ((start\_x + 2) < (a - 1)){

*if* (labirint[start\_x + 2][start\_y] == 0){

labirint[start\_x + 2][start\_y] = 1;

labirint[start\_x + 1][start\_y] = 1;

}

visible[start\_x + 2][start\_y] = *true*;

visible[start\_x + 1][start\_y] = *true*;

Steck\_x.pop();

Steck\_x.push(start\_x + 2);

}

*break*;

*case* 1:

*if* ((start\_x - 2) > 0){

*if* (labirint[start\_x - 2][start\_y] == 0){

labirint[start\_x - 2][start\_y] = 1;

labirint[start\_x - 1][start\_y] = 1;

}

visible[start\_x - 2][start\_y] = *true*;

visible[start\_x - 1][start\_y] = *true*;

Steck\_x.pop();

Steck\_x.push(start\_x - 2);

}

*break*;

*case* 2:

*if* ((start\_y + 2) < (a - 1)){

*if* (labirint[start\_x][start\_y + 2] == 0){

labirint[start\_x][start\_y + 2] = 1;

labirint[start\_x][start\_y + 1] = 1;

}

visible[start\_x][start\_y + 2] = *true*;

visible[start\_x][start\_y + 1] = *true*;

Steck\_y.pop();

Steck\_y.push(start\_y + 2);

}

*break*;

*case* 3:

*if* ((start\_y - 2) > 0){

*if* (labirint[start\_x][start\_y - 2] == 0){

labirint[start\_x][start\_y - 2] = 1;

labirint[start\_x][start\_y - 1] = 1;

}

visible[start\_x][start\_y - 2] = *true*;

visible[start\_x][start\_y - 1] = *true*;

Steck\_y.pop();

Steck\_y.push(start\_y - 2);

}

*break*;

}

}

button\_3\_ready = *true*;

}

vector<vector<int>> Convert(int a){

vector<vector<int>> list;

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

*for* (int j = 0; j < a; j++){

list.push\_back(vector<int>());

}

}

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

*for* (int j = 0; j < a; j++){

*if* (labirint[i][j]){

*if* ((i+1) < a){

*if* (labirint[i+1][j]){

list[i\*a + j].push\_back((i+1)\*a + j);

}

}

*if* ((i-1) >= 0){

*if* (labirint[i-1][j]){

list[i\*a + j].push\_back((i-1)\*a + j);

}

}

*if* ((j+1) < a){

*if* (labirint[i][j+1]){

list[i\*a + j].push\_back(i\*a + j + 1);

}

}

*if* ((j-1) >= 0){

*if* (labirint[i][j-1]){

list[i\*a + j].push\_back(i\*a + j - 1);

}

}

}

}

}

*return* list;

}

vector<int> Sweet\_cell(vector<vector<int>> pole, int a){

vector<int> dist(a\*a);

int start = output.x\*a + output.y;

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

dist[i] = -1;

}

QQueue<int> Steck;

Steck.push\_back(start);

dist[start] = 0;

int v;

*while* (!Steck.empty()) {

v = Steck.front();

*//fout* *<<* *"start* *"* *<<* *v* *<<* *endl;*

Steck.pop\_front();

*for* (int i = 0; i < pole[v].size(); i++) {

*if* (dist[pole[v][i]] == -1) {

Steck.push\_back(pole[v][i]);

dist[pole[v][i]] = dist[v] + 1;

}

}

}

*return* dist;

}

bool flag(*false*);

vector<int> Seach\_point(int a, vector<vector<int>> pole, int in\_x,int in\_y,int out\_x, int out\_y){

vector<int> result;

*if* ((in\_x == out\_x) && (in\_y == out\_y)){

flag = *true*;

result.push\_back(in\_x\*a + in\_y);

*return* result;

}

*for* (int i = 0; i < pole[in\_x\*a + in\_y].size(); i++){

*if* (!visible[pole[in\_x\*a + in\_y][i]/a][pole[in\_x\*a + in\_y][i]%a]){

visible[pole[in\_x\*a + in\_y][i]/a][pole[in\_x\*a + in\_y][i]%a] = *true*;

result = Seach\_point(a, pole, pole[in\_x\*a + in\_y][i]/a, pole[in\_x\*a + in\_y][i]%a, out\_x, out\_y);

}

*if* (flag){

result.push\_back(in\_x\*a + in\_y);

*return* result;

}

}

*return* result;

}

vector<int> Seach\_road(vector<int> road, vector<vector<int>> list, int a, int in\_x, int in\_y, int out\_x, int out\_y){

vector<int> vision;

vector<int> result;

int radius = a/level;

result.push\_back(in\_x\*a + in\_y);

cell v;

*while*(result[result.size() - 1] != (out\_x\*a + out\_y)){

v = {result[result.size() - 1]/a , result[result.size() - 1]%a};

int min = road[v.x\*a + v.y];

*for* (int i = -radius; i <= radius; i++){

*for*(int j = -radius; j <= radius; j++){

*if*(((i\*i + j\*j) <= (radius\*radius)) && ((v.x + i) > -1) && ((v.y + j) > -1) && ((v.x + i) < a) && ((v.y + j) < a)){

*if* ((road[(v.x + i)\*a + v.y + j] != -1) && (road[(v.x + i)\*a + v.y + j] < min)){

min = road[(v.x + i)\*a + v.y + j];

vision.clear();

vision.push\_back((v.x + i)\*a + v.y + j);

}*else*{

*if* ((road[(v.x + i)\*a + v.y + j] != -1) && (road[(v.x + i)\*a + v.y + j] == min)){

vision.push\_back((v.x + i)\*a + v.y + j);

}

}

}

}

}

int randnum = randomBetween(0, vision.size() - 1);

result.push\_back(vision[randnum]);

vision.clear();

}

vision = result;

result.clear();

result.push\_back(vision[0]);

result.push\_back(vision[0]);

result.push\_back(vision[0]);

result.push\_back(vision[0]);

result.push\_back(vision[0]);

result.push\_back(vision[0]);

result.push\_back(vision[0]);

result.push\_back(vision[0]);

vector<int> temp;

*for* (int i = 1; i < vision.size(); i++){

*for* (int k = 0; k < a; k++){

*for* (int j = 0; j < a; j++){

visible[k][j] = *false*;

}

}

flag = *false*;

temp = Seach\_point(a, list, vision[i-1]/a, vision[i-1]%a, vision[i]/a, vision[i]%a);

temp.pop\_back();

*for* (int j = (temp.size()-1); j >= 0; j--){

result.push\_back(temp[j]);

}

}

*return* result;

}

int counter\_robot = 0;

void game::navigator(){

*if* ((robot.x == output.x) && (robot.y == output.y)){

*return*;

}

move\_robot(razmer, (road[counter\_robot + 1]/razmer - road[counter\_robot]/razmer), (road[counter\_robot + 1]%razmer - road[counter\_robot]%razmer));

counter\_robot++;

}

void game::win\_message\_man(){

timer\_paint->stop();

timer\_fps->stop();

robot\_time->stop();

scene->clear();

item\_hero = scene->addPixmap(\*pic\_win\_man);

item\_hero->setPos(-320,-252);

}

void game::win\_message\_robot(){

timer\_paint->stop();

timer\_fps->stop();

robot\_time->stop();

scene->clear();

item\_hero = scene->addPixmap(\*pic\_win\_robot);

item\_hero->setPos(-320,-252);

}

void game::move\_hero(int a, int move\_x, int move\_y){

*if* (hero.x == output.x && hero.y == output.y){

*return*;

}

cell temp = hero;

temp.x += move\_x;

temp.y += move\_y;

*if* ((temp.x > -1) && (temp.x < (a)) && (temp.y > -1) && (temp.y < (a)) && (labirint[temp.x][temp.y])){

hero = temp;

*if* (hero.x == output.x && hero.y == output.y){

ui->statusbar->showMessage("log: Победа");

win\_message\_man();

}

}

}

void game::move\_robot(int a, int move\_x, int move\_y){

*if* (robot.x == output.x && robot.y == output.y){

*return*;

}

cell temp = robot;

temp.x += move\_x;

temp.y += move\_y;

*if* ((temp.x > -1) && (temp.x < (a)) && (temp.y > -1) && (temp.y < (a)) && (labirint[temp.x][temp.y])){

robot = temp;

*if* (robot.x == output.x && robot.y == output.y){

ui->statusbar->showMessage("log: Победа");

win\_message\_robot();

}

}

}

void game::slotAlartFps(){

time\_timer++;

ui->label\_2->setText("fps: " + QString::number(fps));

fps = 0;

QString str = "";

*if* (time\_timer/60 > 9){

str = QString::number(time\_timer/60) + ":";

}*else*{

str = "0" + QString::number(time\_timer/60) + ":";

}

*if* (time\_timer%60 > 9){

str += QString::number(time\_timer%60);

}*else*{

str += "0" + QString::number(time\_timer%60);

}

ui->label->setText(str);

}

void game::Paint3D(int a, int b){

razmer = a;

level = b;

ui->statusbar->showMessage("log: Paint3D был запущен.");

labirint = *new* int\*[a];

visible = *new* bool\*[a];

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

labirint[i] = *new* int[a];

visible[i] = *new* bool[a];

}

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

*for* (int j = 0; j < razmer; j++){

labirint[i][j] = 0;

*if*((i == 0) || (j == 0) || (i == (razmer - 1)) || (j == (razmer - 1))){

visible[i][j] = *true*;

}*else*{

visible[i][j] = *false*;

}

}

}

hero.x = 0;

hero.y = 0;

item\_hero = scene->addPixmap(\*pic\_hero);

scene->addItem(item\_hero);

item\_robot = scene->addPixmap(\*pic\_robot);

scene->addItem(item\_robot);

timer\_paint = *new* QTimer();

connect(timer\_paint, *SIGNAL*(timeout()), *this*, *SLOT*(slotAlarmTimer()));

timer\_fps = *new* QTimer();

connect(timer\_fps, *SIGNAL*(timeout()), *this*, *SLOT*(slotAlartFps()));

robot\_time = *new* QTimer();

connect(robot\_time, *SIGNAL*(timeout()), *this*, *SLOT*(navigator()));

ui->pushButton\_3->click();

}

void game::on\_pushButton\_clicked()

{

*this*->setFocus();

*delete* timer\_paint;

*delete* timer\_fps;

*delete* robot\_time;

*emit* firstWindow();

*this*->close();

}

void game::on\_pushButton\_2\_clicked()

{

*this*->setFocus();

}

void game::Reset\_Scene(){

razmer\_h = ui->graphicsView->width();

razmer\_h /= 2;

int razmer\_w = ui->graphicsView->height();

*if* (razmer\_w < razmer\_h){

razmer\_h = razmer\_w;

}

int razmer\_used = razmer;

int h\_used = razmer\_h - razmer\_h/50; *//* *уменьшаем* *размер* *лабиринта* *на* *2%* *для* *удобства* *отображения*

int line = h\_used/razmer\_used;

scene->clear();

*for* (int i = 0; i < razmer\_used; i ++){

*for* (int j = 0; j < razmer\_used; j++){

*if* (labirint[i][j]){

scene->addRect(j\*line - (razmer\_used/2\*line) - razmer\_h/2, i\*line - (razmer\_used/2\*line), line, line, \*transparentpen, \*grayBrush);

scene->addRect(j\*line - (razmer\_used/2\*line) + razmer\_h/2, i\*line - (razmer\_used/2\*line), line, line, \*transparentpen, \*grayBrush);

}*else*{

scene->addRect(j\*line - (razmer\_used/2\*line) - razmer\_h/2, i\*line - (razmer\_used/2\*line), line, line, \*transparentpen, \*brownBrush);

scene->addRect(j\*line - (razmer\_used/2\*line) + razmer\_h/2, i\*line - (razmer\_used/2\*line), line, line, \*transparentpen, \*brownBrush);

}

}

}

\*pic\_robot = pic\_robot->scaled(QSize(line - (line/4), line - (line/4)),Qt::KeepAspectRatio);

item\_robot = scene->addPixmap(\*pic\_robot);

\*pic\_hero = pic\_hero->scaled(QSize(line - (line/4), line - (line/4)),Qt::KeepAspectRatio);

item\_hero = scene->addPixmap(\*pic\_hero);

}

void game::slotAlarmTimer(){

fps++;

int razmer\_used = razmer;

int h\_used = razmer\_h - razmer\_h/50; *//* *уменьшаем* *размер* *лабиринта* *на* *2%* *для* *удобства* *отображения*

int line = h\_used/razmer\_used;

item\_hero->setPos(hero.y\*line - (razmer\_used/2\*line) + (line/8) - razmer\_h/2, hero.x\*line - (razmer\_used/2\*line) + (line/8));

item\_robot->setPos(robot.y\*line - (razmer\_used/2\*line) + (line/8) + razmer\_h/2, robot.x\*line - (razmer\_used/2\*line) + (line/8));

}

void game::keyPressEvent(QKeyEvent \*event)

{

ui->statusbar->showMessage("log: Нажата клавиша: " + QString::number(event->key()));

*if* ((event->key() == Qt::Key\_W) || (event->key() == Qt::Key\_Up) || (event->key() == 1062)){

ui->statusbar->showMessage("log: up");

move\_hero(razmer, -1, 0);

}

*if* ((event->key() == Qt::Key\_S) || (event->key() == Qt::Key\_Down) || (event->key() == 1067)){

ui->statusbar->showMessage("log: down");

move\_hero(razmer, 1, 0);

}

*if* ((event->key() == Qt::Key\_A) || (event->key() == Qt::Key\_Left) || (event->key() == 1060)){

ui->statusbar->showMessage("log: left");

move\_hero(razmer, 0, -1);

}

*if* ((event->key() == Qt::Key\_D) || (event->key() == Qt::Key\_Right) || (event->key() == 1042)){

ui->statusbar->showMessage("log: right");

move\_hero(razmer, 0, 1);

}

*if* (event->key() == 16777220){

*//Reset\_Scene();*

ui->pushButton\_2->click();

}

*if* (event->key() == 32){

*//Reset\_Scene();*

ui->pushButton\_3->click();

}

}

void game::on\_pushButton\_3\_clicked()

{

*this*->setFocus();

ui->pushButton\_3->setEnabled(*false*);

time\_timer = 0;

ui->pushButton\_2->setEnabled(*true*);

timer\_paint->stop();

timer\_fps->stop();

robot\_time->stop();

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

*for* (int j = 0; j < razmer; j++){

labirint[i][j] = 0;

*if*((i == 0) || (j == 0) || (i == (razmer - 1)) || (j == (razmer - 1))){

visible[i][j] = *true*;

}*else*{

visible[i][j] = *false*;

}

}

}

int rand\_start = randomBetween(1 , (razmer/2 )) \* 2 - 1;

labirint[rand\_start][1] = 1;

visible[rand\_start][1] = *true*;

clock\_t start, end;

start = clock();

mole(razmer, rand\_start, 1);

end = clock();

input.x = rand\_start;

input.y = 0;

labirint[input.x][input.y] = 1;

hero = input;

robot = input;

counter\_robot = 0;

seach\_exit(razmer);

Reset\_Scene();

timer\_paint->start(10);

vector<vector<int>> list = Convert(razmer);

vector<int> sweets = Sweet\_cell(list, razmer);

road = Seach\_road(sweets, list, razmer, robot.x, robot.y, output.x, output.y);

timer\_fps->start(1000);

robot\_time->start(300);

ui->pushButton\_3->setEnabled(*true*);

}