**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ**Федеральное государственное бюджетное образовательное учреждениевысшего образования **«Сибирский государственный университет науки и технологий   
имени академика М.Ф. Решетнева»**

|  |  |  |
| --- | --- | --- |
|  | Институт информатики и телекоммуникаций |  |
|  |  |  |
|  | Кафедра информатики и вычислительной техники |  |
|  |  |  |

**КУРСОВАЯ РАБОТА**

|  |  |
| --- | --- |
| Программная реализация поиска пути в лабиринте |  |
|  |  |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Руководитель | | | |  | | | |  | |  | |  | | В.В. Тынченко | |
|  |  |  | |  |  |  | |  | | подпись, дата | |  | | инициалы, фамилия | |
| Обучающийся | | | | БПИ22-02, 221219040 | | |  | |  | |  | | К.В. Трифонов | |
|  |  | | номер группы, зачетной книжки | | | | | | подпись, дата | |  | | инициалы, фамилия | |

**Красноярск 2023**

Институт информатики и телекоммуникаций

Кафедра информатики и вычислительной техники

|  |
| --- |
| **ЗАДАНИЕ**  на курсовую работу по дисциплине «Алгоритмы и структуры данных»  студенту |
| Группа БПИ22-02 Форма обучения очная |  |
| 1. Тема работы: Программная реализация поиска пути в лабиринте |
| 2. Срок сдачи студентом работы: 30.12.23 |
| 3. Перечень вопросов, подлежащих разработке при написании теоретической части:  Прогнозирование временных рядов. Построение нейронных сетей.  Использование нейронных сетей в прогнозировании временных рядов. |
| 4. Перечень вопросов, подлежащих разработке при написании практической части (либо указать номер варианта задания)  Разработать нейронные сети и спрогнозировать временные ряды. Провести  исследование быстродействия и точности прогнозирования. |
| 5. Дата выдачи задания: 08.09.23 |
| Руководитель: Тынченко В.В. |
| Подпись |
|  |
| Задание принял к исполнению (дата): 08.09.23 |
| (подпись студента) |

# СОДЕРЖАНИЕ

[ВВЕДЕНИЕ 4](#_Toc154351290)

[1 Анализ предметной области 5](#_Toc154351291)

[1.1 Алгоритм Дейкстры 5](#_Toc154351292)

[1.2 Алгоритм A\* 5](#_Toc154351293)

[1.3 Поиск в ширину (BFS) 6](#_Toc154351294)

[1.4 Поиск в глубину (DFS) 8](#_Toc154351295)

[1.5 Jump Point Search (JPS) 8](#_Toc154351296)

[1.6 Вывод 11](#_Toc154351297)

[2 Описание программы 12](#_Toc154351298)

[2.1 Общая характеристика программы 12](#_Toc154351299)

[2.2 Структура программы и данных 12](#_Toc154351300)

[2.3 Интерфейс программы 14](#_Toc154351301)

[2.4 Тестирование программы 17](#_Toc154351302)

[2.5 Вывод 19](#_Toc154351303)

## ВВЕДЕНИЕ

В современном мире область исследования алгоритмов поиска пути приобретает особую актуальность, находя применение в широком спектре задач, от робототехники до компьютерных игр. Программная реализация эффективных и точных методов поиска пути в лабиринте имеет важное значение в контексте разработки автономных систем, планирования маршрутов и создания интерактивных игровых сценариев.

Данная курсовая работа посвящена исследованию и программной реализации различных алгоритмов поиска пути в лабиринте. В ходе работы рассмотрены и проанализированы такие важные методы, как алгоритм Дейкстры, A\*, поиск в ширину (BFS), поиск в глубину (DFS) и Jump Point Search (JPS). Каждый из этих алгоритмов рассматривается с точки зрения своей эффективности, точности и применимости в различных сценариях.

Цель работы заключается в разработке программных реализаций выбранных алгоритмов и в последующем тестировании их производительности в условиях лабиринта. Такой подход позволяет выявить особенности работы каждого алгоритма, а также определить, какие из них наиболее эффективны в различных контекстах использования.

Задачи:

* изучить теоретические основы алгоритмов создания лабиринтов
* изучить теоретические основы алгоритмов поиска путей
* разработать программный код алгоритмов
* провести тестирование программы

# 1 Анализ предметной области

*Программная реализация поиска пути в лабиринте* — это интересная и важная задача, которая используется во многих областях, таких как навигация, планирование транспортных маршрутов, сортировка, робототехника и искусственный интеллект. Существует множество алгоритмов, которые можно использовать для решения этой задачи. Наиболее популярные из них приведены ниже.

## 1.1 Алгоритм Дейкстры

Алгоритм назван в честь голландского ученого Эдсгера Дейкстры, который разработал его в 1956 году. За основу алгоритма берется принцип жадного выбора: на каждом шаге выбирается вершина с наименьшим известным расстоянием от источника и проверяются все её соседние вершины. Если расстояние до соседней вершины через текущую вершину оказывается короче, то расстояние обновляется.

Однако этот алгоритм подходит только для графов без отрицательных ребер, так как при наличии отрицательных циклов может возникнуть бесконечный цикл обновления расстояний.

Алгоритм описывается следующим образом:

* в начале алгоритма расстояние для начальной вершины полагается равным нулю, а все остальные расстояния заполняются большим положительным числом (бо́льшим максимального возможного пути в графе);
* массив флагов заполняется нулями. Затем запускается основной цикл;
* на каждом шаге цикла мы ищем вершину v с минимальным расстоянием и флагом равным нулю. Затем мы устанавливаем в ней флаг в 1 и проверяем все соседние с ней вершины u. Если в них (в u) расстояние больше, чем сумма расстояния до текущей вершины и длины ребра, то уменьшаем его;
* цикл завершается, когда флаги всех вершин становятся равны 1, либо когда у всех вершин c флагом 0 Последний случай возможен тогда и только тогда, когда граф G несвязный.

Работа алгоритма показана на блок-схеме (Рисунок 1 ).

## 1.2 Алгоритм A\*

Является модификацией алгоритма Дейкстры, который использует эвристику для ускорения процесса поиска. Он работает путем оценки стоимости каждого возможного шага и выбора того, который, как предполагается, приведет к наиболее эффективному пути.

Основа алгоритма - это эвристическая функция “расстояние + стоимость” (обычно обозначается как f(x)), которая является суммой двух компонентов: стоимости достижения текущей вершины (x) из начальной (обычно обозначается как g(x) и может быть как эвристической, так и нет), и эвристической оценки расстояния от текущей вершины до конечной (обозначается как h(x)). Эта функция определяет порядок обхода вершин.

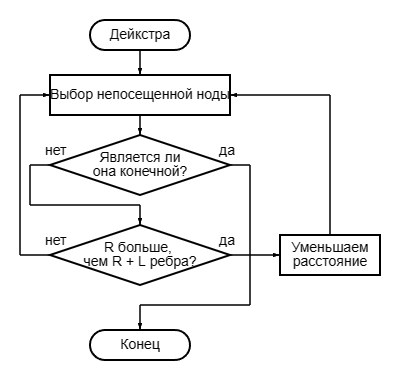


Рисунок 1 – Блок-схема алгоритма Дейкстры

Алгоритм описывается следующим образом:

* начинаем с начальной вершины и устанавливаем ее оценочную стоимость равной эвристической оценке расстояния до цели;
* пока есть не посещенные вершины выбираем вершину с наименьшей оценочной стоимостью и пометьте ее как посещенную. Если это целевая вершина, то путь найден;
* в противном случае для каждой соседней вершины обновляем ее оценочную стоимость, если текущая оценка больше суммы стоимости пути до выбранной вершины, веса ребра между ними и эвристической оценки до цели.

Работа алгоритма показана на блок-схеме (Рисунок 2 ).

## 1.3 Поиск в ширину (BFS)

Начинается с корневого узла и исследует все соседние узлы на данном уровне перед переходом к узлам следующего уровня. Ключевая идея заключается в том, что мы отслеживаем состояние расширяющегося кольца, которое называется границей. В сетке этот процесс иногда называется заливкой (flood fill), но та же техника применима и для карт без сеток.

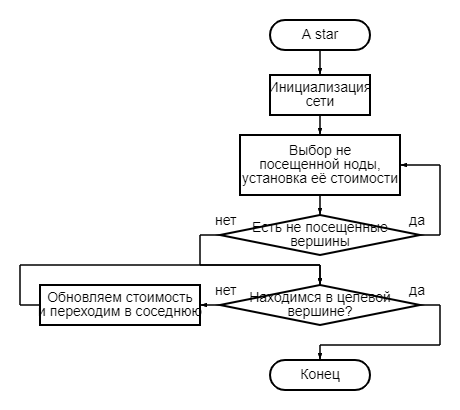


Рисунок 2 – Блок-схема алгоритма A\*

Алгоритм описывается следующим образом:

* берётся первый узел из очереди и помечается как посещенный;
* если он целевой, то завершаем работу программы, иначе проверяем преемников этого узла;
* если все узлы просмотрены, а целевой найден не был, то он недостижим из начального.

работа алгоритма показана на блок-схеме (Рисунок 3).

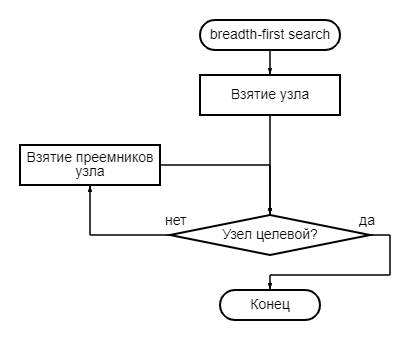


Рисунок 3 – Блок-схема BFS алгоритма

## 1.4 Поиск в глубину (DFS)

Основан на схожем принципе, что и поиск в ширину, однако распространяется в самую глубь графа, пока не достигнет конца, а затем возвращается назад, чтобы исследовать другие ветви. Возврат обратно к высшей вершине происходит только когда в выбранной вершине были рассмотрены все рёбра, в которых не осталось нерассмотренных вершин.

Алгоритм описывается следующим образом:

* начинаем с корневого узла и добавляем его в стек;
* пока стек не пуст извлекаем узел из стека и помечаем его как посещенный;
* добавляем все не посещенные соседние узлы в стек;
* если узел целевой или узлов не осталось (решение невозможно), то цикл заканчивается с соответствующим результатом.

Работа алгоритма показана на блок-схеме (Рисунок 4 ).

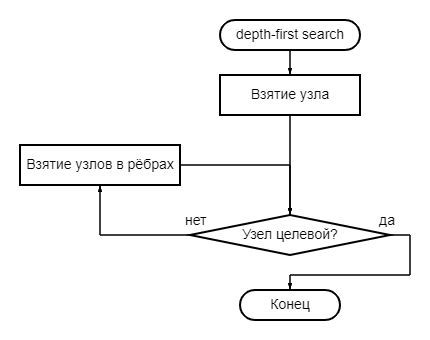


Рисунок 4 – Блок-схема DFS алгоритма

## 1.5 Jump Point Search (JPS)

Является оптимизацией алгоритма поиска A\* для сеток с равномерной стоимостью. Он уменьшает симметрию в процедуре поиска путем обрезки графа, устраняя определенные узлы в сетке на основе предположений, которые можно сделать о соседях текущего узла, при условии, что выполняются определенные условия, относящиеся к сетке. Преимуществом JPS является то, что он не требует предобработки и поэтому тратит меньше памяти, также он является самым современным из перечисленных – он получил широкую огласку в 2012 году, а последние его крупные улучшения были опубликованы в 2014.

Алгоритм описывается следующим образом:

* берём корневой узел;
* если узел не целевой, то выбирается точка (узел графа), в которую совершится прыжок по прямой (Рисунок 5 ) или по диагонали (Рисунок 6 );
* если узел целевой или узлов не осталось (решение невозможно), то цикл заканчивается с соответствующим результатом.

Работа алгоритма показана на блок-схеме (Рисунок 7 ).

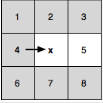


Рисунок 5 – Прямолинейный прыжок JPS алгоритма

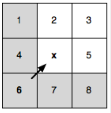


Рисунок 6 – Диагональный прыжок JPS алгоритма

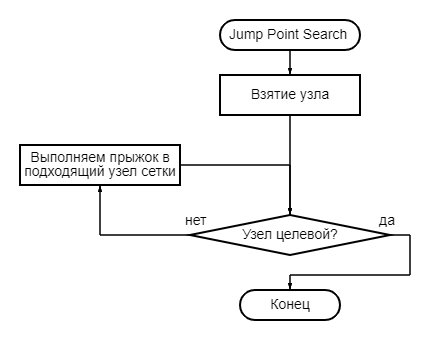


Рисунок 7 – Блок-схема JPS алгоритма

Пусть y является целевой (посадочной) точкой точки х, в направлении d, если y минимизирует значение k по закону: y = x + kd, и будет выполнено любое из условий:

Точка y – точка посадки.

У точки y есть хотя бы один сосед, у которого более 1 соседа.

d –движение по диагонали и существует точка z = y + kidi, которая лежит в ki шагах в направлении di ∈ {d1, d2}, таких что z – точка прыжка из y при условии 1 или 2.

Рассмотрим пример одного из прыжков (Рисунок 8 ). Пусть мы начинаем в точке х и заканчиваем движение по диагонали, пока не наткнёмся на точку у. Из у в точку z можно попасть с ki по горизонтали. Таким образом, z является преемником точки для прыжка x, а это в свою очередь определяет y как преемник для прыжка точки x.

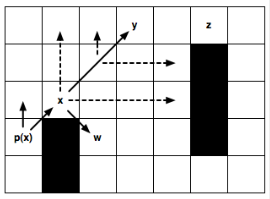


Рисунок 8 – Пример прыжка JPS алгоритма

## 1.6 Вывод

Представляя лабиринт в виде неориентированного графа, можно использовать множество разнообразных алгоритмов, которые могут быть эффективны в конкретном спектре задач. Большинство из них используют схожий принцип последовательного исследования графа с проверкой текущей вершины (точки лабиринта) на то, является ли она целевой (выходом из лабиринта). Однако они кардинально различаются в подходе к порядку выбора рассматриваемых вершин.

# 2 Описание программы

## 2.1 Общая характеристика программы

Название программы - «Изучение быстродействия методов поиска путей в лабиринте».

Программа, написанная в ходе выполнения курсовой работы направлена на изучение быстродействия и особенностей отдельных алгоритмов поиска путей в лабиринтах. Программа способна строить лабиринты тремя методами, находить в них путь и выводить лабиринты на экран с применением форматирования (отрисовкой стен).

Для написания программы был выбран C++, поскольку он имеет более высокое быстродействие по сравнению с python.

Минимальные системные требования:

* процессор: 2 ядра и частотой от 2 ГГц;
* оперативная память: 4 Гб;
* свободное дисковое пространство: 5Мб;
* операционная система: Windows 10 со средой разработки Visual Studio

## 2.2 Структура программы и данных

В разработке программы применялся метод ООП и был разработан класс «Лабиринт» с полями матрицы лабиринта и посещённых клеток и класс «Точка» с полями координат для заполнения списков очереди в алгоритмах поиска. В каждом из классов были реализованы конструкторы (по умолчанию, с параметрами, копий) и деструкторы.

В классе «Лабиринт» также были созданы методы отрисовки лабиринта на матрице с помощью трёх методов (Алгоритмом северо-восточного смещения, комбинированным алгоритмом северо-восточного и северо-западного смещения и алгоритмом «Sidewinder») и три метода поиска путей (BFS, DFS и A star).

Блок схема представлена на рисунке (Рисунок 9).

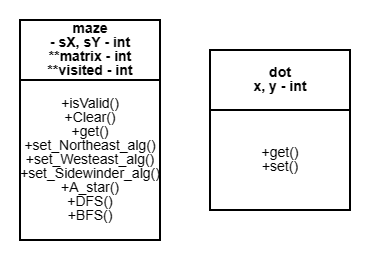


Рисунок 9 – Блок схема классов программы

## 2.3 Интерфейс программы

Использование консоли в качестве взаимодействия с пользователем накладывает некоторые ограничения в отрисовке изображений на экране, однако лабиринты всё равно можно вывести с помощью символов. Для отрисовки стен применять, например, крупные объемные буквы, а для пустых мест ставить пробелы или точки.

Для вывода стен-границ использован символ “H”, для вывода внутренних стен символ “X”, а для проходов - пробелы. Для обозначения стартовой позиции используется символ “S”, а для финиша используется символ “E”. По умолчанию их расположение в левом верхнем и правом нижнем углах соответственно.

Пример вывода лабиринта, заполненного алгоритмом северо-восточного смещения размером 31 x 31 на рисунке (Рисунок 10 ).

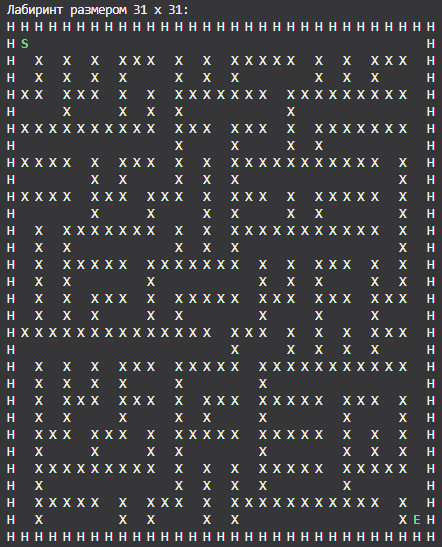


Рисунок 10 – Работа комбинированного алгоритма северо-восточного смещения

Пример вывода лабиринта, заполненного комбинированным алгоритмом северо-восточного смещения размером 31 x 31 на рисунке (Рисунок 11 ).



Рисунок 11 – Работа комбинированного алгоритма северо-восточного смещения

Пример вывода лабиринта, заполненного алгоритмом Sidewinder размером 31 x 31 на рисунке (Рисунок 12 ).

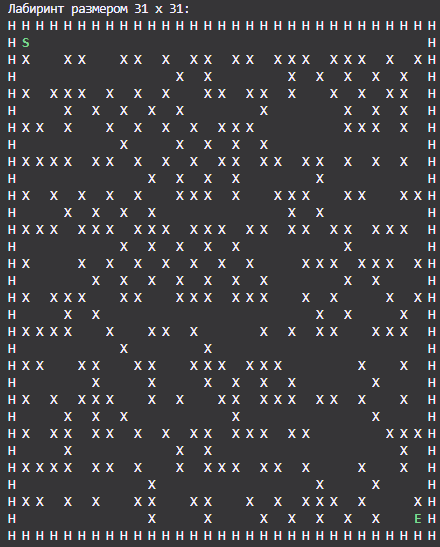


Рисунок 12 – Пример работы алгоритма Sidewinder

Алгоритм северо-восточного смещения получилось реализовать стабильным, он всегда генерирует решаемый лабиринт с одним решением, поэтому дальнейшее тестирование будет проводиться на лабиринтах сгенерированных этим алгоритмом.

**Описание меню программы:**

Визуально, меню отображается в виде пронумерованных пунктов с описанием каждой опции. Это обеспечивает понятность и удобство взаимодействия с пользователем. Каждый пункт меню отображается с новой строки, что улучшает читаемость и структурированность вывода, делая его более интуитивно понятным для пользователя.

Меню представлено на рисунке (Рисунок 12 )

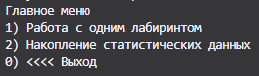


Рисунок 13 – Вид главного меню

Выбор режима: позволяет выбрать режим работы (работа с одним лабиринтом, накопление статистических данных).

*Режим работы с одним лабиринтом*: пользователь вводит с консоли размеры исследуемого лабиринта, далее открывается меню функций по работе с одним лабиринтом: построение (по выбору), решение (с выбором метода) и вывод на экран.

*Режим работы с накоплением статистических данных*: пользователь вводит диапазон тестирования и шаг, далее выбирается метод заполнения и тестируемый метод решения лабиринта. По окончанию решения каждого из N лабиринтов результаты выводятся на экран и файл results.txt.

UML – диаграмма прецедентов (Рисунок 14 ).

UML – диаграмма интерфейса представлена на рисунке (Рисунок 15 ).

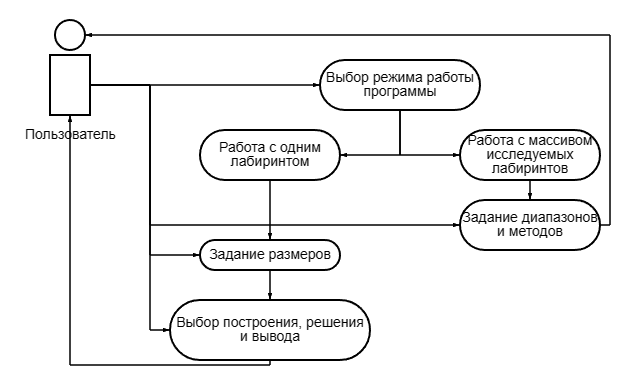


Рисунок 14 – UML диаграмма прецедентов

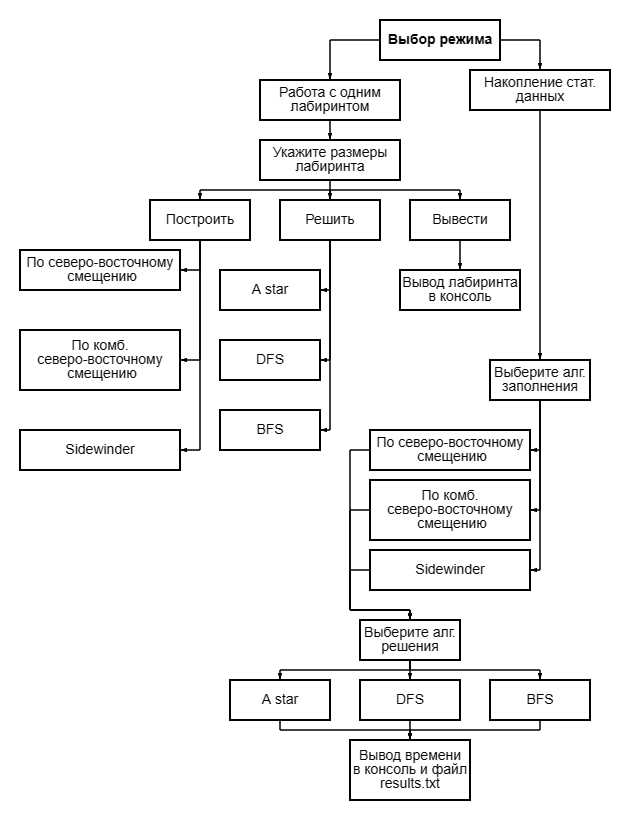


Рисунок 15 – UML диаграмма интерфейса программы

## 2.4 Тестирование программы

Для тестирования работы алгоритмов решения лабиринтов создавался алгоритмом северо-восточного смещения лабиринт размером 15 x 15. Работа алгоритма начиналась с клетки “S” (1,1) левого верхнего угла и заканчивалась, когда финиш “E” становился текущей клеткой.

Для тестирования был сгенерирован лабиринт 15 x 15, он примечателен тем, что у него имеется быстрый путь к финишу только после трёх глубоких перекрёстков.

Тестируемый лабиринт изображен на рисунке (Рисунок 16 ).

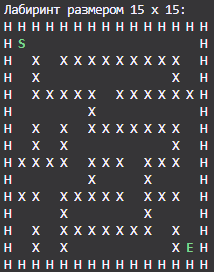


Рисунок 16 – Тестируемый лабиринт

DFS алгоритм (поиск в глубину) по итогам тестирования оказался самым эффективным, т.к. финиш находится в самой отдалённой части лабиринта. Было посещено наименьшее количество точек.

Результат работы DFS алгоритма представлен на рисунке (Рисунок 17 ).

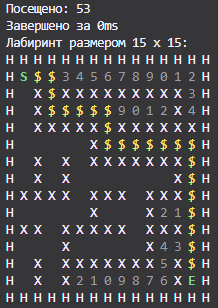


Рисунок 17 – Результат работы DFS алгоритма

BFS алгоритм (поиск в ширину) по итогам тестирования показал себя хуже всех, поскольку целевая точка находится в самом дальнем углу. Этот метод может показать себя эффективнее, чем DSF, если известно, что целевая точка находится не “на глубине”. Было посещено наибольшее количество точек.

Результат работы BFS алгоритма представлен на рисунке (Рисунок 18 ).

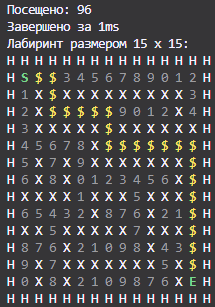


Рисунок 18 – Результат работы BFS алгоритма

Алгоритм A star показывает наиболее оптимальный путь решения, не заходит в заведомо более далёкие от цели пути. Его логика основывается на DSF поиске, за исключением того, что у него присутствует, помимо анализа стоимости длины g(n), ещё и функция h(n), которая вычисляет удалённость точки до цели. Благодаря этому, на развилках, A star выберет дорогу согласно её направлению к цели.

Результат работы A star алгоритма представлен на рисунке (Рисунок 19 ).

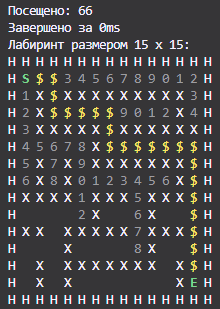


Рисунок 19 – Результат работы A star алгоритма

## 2.5 Вывод

В данном разделе проведен обзор программы, ее структуры, интерфейса и результатов тестирования. Выявлено, что использование алгоритма A\* в программе обеспечивает более эффективный поиск пути в сравнении с методами BFS и DFS, что подтверждает не только теоретические предположения, но и результаты практических испытаний.

# 3 Использование программы

## 3.1 Постановка задачи

Программа должна создавать различные лабиринты, находить в них путь от начальной точки до конечной различными алгоритмами с подсчетом времени и посещенных точек для сравнения различных методов поиска пути.

## 3.2 Реализация программы

Для создания лабиринта были реализованы 3 алгоритма:

* алгоритм северо-восточного смещения
* комбинированный алгоритм северо-восточного смещения
* алгоритм Sidewinder

Алгоритм северо-восточного смещения создаёт простой лабиринт, который имеет два пустых коридора вверху и справа в лабиринте, а также сильное диагональное смещение в направлении от левого нижнего до правого верхнего угла.

Комбинированный алгоритм северо-восточного смещения представляет собой смесь из северо-восточного и юго-западного смещения. Первые 8 клеток лабиринта обрабатываются одним, следующие 8 вторым и т.д. Не имеет пустого коридора справа, а также явное смещение в каком-либо направлении.

Алгоритм Sidewinder создаёт пустые горизонтальные коридоры случайной длины (от 1 до 8) и соединяет их пробелами в случайном месте. Имеет только верхний коридор. Реализовать стабильно его не получилось т.к. иногда возможны ситуации, когда он создаёт замкнутые области, в том числе и вокруг финиша.

Для поиска пути в созданном лабиринте были реализованы 3 алгоритма:

* BFS
* DFS
* A star

Алгоритм BFS создаёт две дополнительные матрицы – булевую **visited** хранящую посещённые точки и целых чисел **distance** хранящую расстояние от старта до этой клетки матрицы (обе имеют ту же размерность, что и чар матрица поля класса maze). Также имеется очередь, в которую в конец записываются новые найденные допустимые точки. Сначала проверяются точки из начала очереди, т.е. старые. Таким образом алгоритм сначала просмотрит точки, которые были обнаружены раньше и пойдёт вширь, а не в глубину новых точек.

Алгоритм DFS использует такие же матрицы, но хранит точки не в очереди, а в стеке, следовательно сначала проверяются точки из конца, т.е. новые. Происходит проход в глубину до тех пор, пока новых точек не появится.

Алгоритм A star

Когда точки в очереди закончились или был найден финиш алгоритм начинает собирать кратчайший путь до финиша, пользуясь матрицей **distance**, сравнивая пути на перекрёстках. Параллельно этому в чар матрицу класса maze записываются путь и последняя цифра расстояния от старта до этой точки для зрительной оценки пути.