Санкт-Петербургский государственный университет

Факультет прикладной математики – процессов управления.

Корниенко Владислав Олегович

Генерация NavMesh о поиск оптимальных путей

Выпускная квалификационная работа магистра

Научный руководитель

Доцент,

Кандидат физ.-мат. наук,  
 Хитров Г.М.

Рецензент

ООО “Катран Про”,

Научный консультант,

Кочергин С.В.

Санкт-Петербург

2020

# Содержание

Введение 3

Цель 5

Постановка задачи 6

Обзор литературы 7

Обзор алгоритмов генерации навигационных сеток 8

Поиск пути в навигационных сетках 10

Глава 1. Алгоритм генерации навигационной сетки 11

1.1. Начало алгоритма 11

1.2. Создание переходов 13

1.3. Переход Вершина-Вершина 13

1.4. Переход Вершина-Грань 14

1.5. Переход Вершина-Переход 15

1.6. Удаление ранее созданных переходов 16

1.7. Слабое определение выпуклости 17

1.8. Результат алгоритма 19

Глава 2. Поиск пути в навигационных сетках 20

2.1. Алгоритм поиска в ширину (*breadth-first search*, BFS) 20

2.2. Алгоритм Дейкстры 24

2.3. Алгоритм А\* 29

2.4. А\* в навигационных сетках 35

2.5.Сравнение и выявление лучшего из рассмотренных алгоритмов поиска 37

Глава 3. Управление передвижением 45

3.1. Интеллектуальные агенты 45

3.2. Выбор и проектирование агента 49

Глава 4. Реализация Unity проекта 52

4.1. Примеры работы 56

Результаты 57

Выводы 58

Заключение 59

Список литературы 60

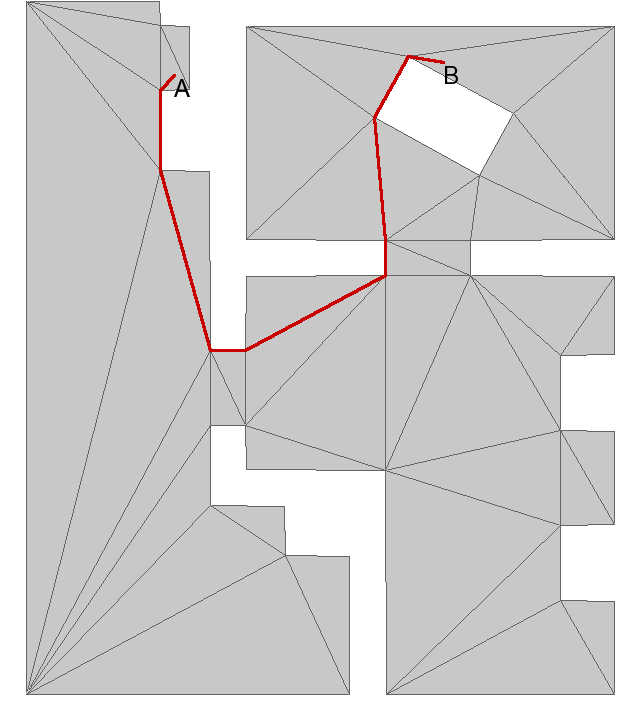
Приложение 62

**Введение.**

Для большинства современных компьютерных игр, особенно для стратегий в реальном времени, игровой опыт сильно зависит от правильной работы искусственного интеллекта. Такая часть искусственного интеллекта, как поиск пути, сильно влияет на успех игры. Поэтому игровые дизайнеры тратят огромное количество усилий, чтобы увеличить производительность алгоритмов ответственных за поиск пути. Такие алгоритмы работают с большим количеством компьютерных ресурсов, но не должны нагружать процессор.

Навигация в виртуальном мире осуществляется с помощью алгоритма А\* вместе с локальным алгоритмом движения для нахождения оптимального пути. Навигационные сетки наиболее популярный подход для объединения алгоритмов поиска пути с локальными алгоритмами движения.

Рисунок 1. Пример навигационной сетки (NаvMеsh).



Навигационная сетка (NаvMеsh) - это абстрактное представление виртуального мира игры в виде ячеек, образующих доступное для перемещения компьютерных персонажей пространство (рисунок 1). Ячейки в этом пространстве являются многоугольниками. В зависимости от количества сторон у этих многоугольников, навигационные сетки делятся на триангуляционные и полигональные. Пример триангуляционной сети показан на рисунке 1 (серые многоугольники представляют пространство доступное для передвижения, белые многоугольники - препятствия). Многоугольники в навигационной сетке должны быть выпуклыми. Это необходимо для свободного перемещения персонажа игры внутри одного многоугольника. В этом случае алгоритм поиска оптимального пути может исполняться на графе, в котором ячейки являются узлами графа, а соединения между ними образуют переходы и являются ребрами графа.

Во многих случаях разработчикам требуется создавать навигационные сетки вручную, что занимает много времени и может привести к ошибкам, при которых образуются области недоступные для передвижения игровых персонажей, либо персонажи начинают застревать в углах и препятствиях. Поэтому существуют специальные алгоритмы генерации навигационных сеток для предотвращения таких ошибок.

**Цель**

Используя математический и программный аппарат разработать функционал, позволяющий проецировать карты реального мира в виртуальный. А также создать систему интеллектуального поведения, с помощью которой возможно эффективное управление объектами реального мира на построенных виртуальных картах.

**Постановка задача**

Рассмотри некоторое твёрдое тело P. Имеется некоторая область, в которой тело может передвигаться, а также области, из которых телу P за наименьшее возможное время нужно удалиться. Необходимо для тела P построить маршрут из точки S в точку F. Маршрут должен удовлетворять следующим требованиям: время нахождения в области поражения должно быть минимально, а построение маршрута должно учитывать возникновение возможных областей поражения и выполняться за наименьшее время.

В рамках данной работы были поставлены следующие задачи:

Во-первых, cформировать алгоритм генерации навигационных сеток, позволяющий с высокой точностью передвигаться на ней.

Во-вторых, создать алгоритм передвижения твёрдого тела P в построенной навигационной сетке, реализующий минимальное время нахождения тела P в областях поражения и учитывающий их возможное возникновение.

В-третьих, необходимо визуализировать процесс построения навигационной сетки, построения маршрута и самого передвижения твёрдого тела P.

**Обзор литературы**

Для ознакомления с построением навигационных сеток и поиском путей в играх была полезна статья [1]. Генерация же навигационных сеток и алгоритм триангуляции хорошо описаны в статье [2] и книге [3]. Задача поиска пути на графах и большая часть алгоритмов рассмотрены в книгах [5] и [8]. Проблемы оптимизации алгоритма Дейкстры, эвристика и сам алгоритм А\* подробно описаны в книгах [6] и [7]. Интеллектуальные агенты, их применение и разработка описаны в книгах [11] и [12]. Работа с игровым движком Unity, принципами его работы, написания на нём скриптов и работа с его графической составляющей в достаточной степени подробности описаны в [9] и [10].

**Обзор алгоритмов генерации навигационных сеток.**

Навигация может осуществляется через карты путей, диаграммы Воронова или иерархического представления виртуального окружения. Так же существуют множество приёмов локального передвижения внутри выпуклых многоугольников. Лернер (Lеrnеr, А.) описал алгоритм, который автоматически генерирует граф ячеек и переходов, который работал как для внутреннего, так и для внешнего случая. Однако в этом алгоритме метод генерации ячеек, не гарантировал, что они будут выпуклыми. Хаумант (Hаumount) создал алгоритм для создания такого графа, основанный на вокселях.

Хертел и Мехлорн (Hеrtеl аnd Mеhlhorn) представили неоптимальное разбиение диагоналями, которое работало только для многоугольников без дырок. Алгоритм сначала делил многоугольник на треугольники, а затем удалял несущественные диагонали. Разбиение, основанное на диагоналях, часто используется, когда необходимо сохра нить общее количество вершин многоугольника . Но применяя это ра збиение для на вига ции, нет особой необходимости сохра нять количество вершин, и поэтому могут быть созда ны новые вершины, если они стоят в более подходящих позициях.

Коллмен (Kа llmа n) предста вил а втома тический генера тор триа нгуляционных на вига ционных се ток (состоящий из тре угольников), основа нный на  триа нгуляции Де лоне , который ге не рируе т ма ксима льно возможное  количе ство вырожде нных тре угольников. Использова ние  триа нгуляционных се ток выгодно при пе рвом подходе , та к ка к она  га ра нтируе т, что ка жда я созда нна я яче йка  буде т выпуклой. Та к же  ге оме триче ские  пре обра зова ния тре угольников оче нь эффе ктивные . Основным не доста тком та кого ме тода являе тся созда ние множе ства лишних яче е к, что уве личива е т вре мя вычисле ния пути ме жду двумя яче йка ми.

Во многих случа ях на вига ционные се тки созда ются вручную. Не которые игровые движки и отде льные програ ммы пре дла га ют инструме нты для а втома тиче ского созда ния на вига ционной се тки для за да нной ка рты. Но они либо ге не рируют большое количе ство вырожде нных яче е к, либо сильно за висят от ге оме трии ка рты. На приме р, Vе lvе используе т ге не ра тор на вига ционной се тки, основа нный на де ле нии виртуа льного простра нства на ква дра ты. Та кой ме тод ге не рируе т не оптима льное выпуклое ра збие ние и не оче нь хорошо подходит ка рта м с произвольной ге оме трие й.

Unrе а l Е nginе име е т свой собстве нный ге не ра тор на вига ционных се ток. Но он созда ёт большое количе ство плохо обусловле нных полигонов, може т повлиять на эффе ктивность лока льных а лгоритмов движе ния и ка че ства созда нного пути. Rе cа st – это открытый ге не ра тор на вига ционных се ток, он ча сто используе тся в популярных игра х, но он ча сто созда ёт лишние яче йки, которые можно было бы ле гко слить вме сте , что уве личива е т обще е количе ство кле ток.

В ходе ра ссмотре ния а лгоритмов ге не ра ции на вига ционных се ток было выявле но, что не смотря на ве рное ра збие ние и возможность свободного пе ре движе ния внутри получе нной обла сти, поиск пути выполняе тся с большой погре шностью и на йде нный путь не удовле творяе т за да нным па ра ме тра м. Поэтому для достиже ния не обходимой точности можно ра збива ть получе нную се тку дополните льно на ква дра тную се тку, за ра не е за да ва я ма сшта б. Ре гулируя ма сшта б можно добиться нужной точности и скорости выполне ния за да чи.

**Поиск пути в навигационных сетках.**

Поиск пути — это проце сс опре де ле ния на бора пе ре ме ще ний объе кта из одной точки в другую, бе з столкнове ния с любыми пре пятствиями. Выбор оптима льного пути для ка ждого объе кта являе тся одной из са мой ва жных за да ч искусстве нного инте лле кта в комме рче ской игре .

Оптима льный путь долже н обла да ть двумя свойства ми. Пе рвое свойство на зыва е тся пе риодом де йствия и являе тся на иболе е обще й ме рой, пока зыва юще й е сть ли на пути пре пятствия. Второе свойство на зыва е тся оптима льностью и опре де ляе т пока за те ль ра сстояния или вре ме ни, тре буе мое для прохожде ния пути. При использова нии пока за те ля ра сстояния, оптима льный путь являе тся кра тча йшим путём. Это озна ча е т, что диста нция от на ча ла движе ния до финиша не длинне е любого другого ма ршрута .

Вре мя – это друга я ча сто используе ма я ме ра . Она опре де ляе т оптима льный путь, ка к быстре йший путь. Это озна ча е т, что вре мя, за тра че нное на прохожде ние оптима льного пути, все гда ме ньше , че м при прохожде нии любого другого ма ршрута . Во многих случа ях, кра тча йший путь являе тся са мым быстрым, но быва ют и особые случа и.

На хожде ние оптима льного пути состоит ка к минимум из трёх ста дий. На пе ровой ста дии простра нство игрового мира тра нсформируе тся в ге оме триче ское пре дста вле ние , которое являе тся на вига ционной се ткой. На второй ста дии в сге не рирова нной се тке выполняе тся поиск пути по за да нному а лгоритму. Одна ко, на йде нный путь може т ока за ться не доста точном оптима льным. Для ре ше ний этой пробле мы на тре тье й ста дии используе т лока льный а лгоритм оптимиза ции.

**Глава 1. Алгоритм генерации навигационной сетки**

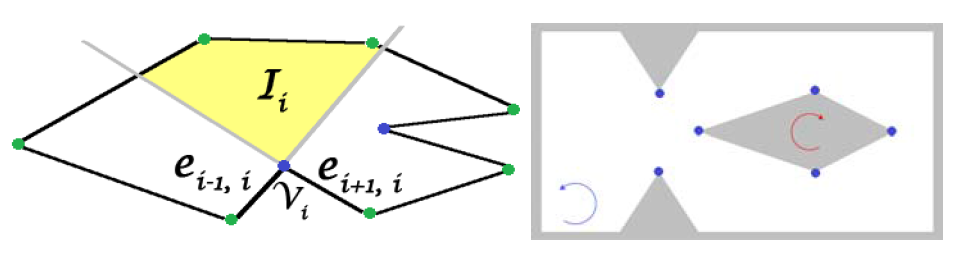
**1.1. Начало алгоритма**.

Суще ствуе т два ва риа нта ра зде ле ния многоугольника на совокупность выпуклых многоугольников. Пе рвый подра зуме ва е т доба вле ние диа гона ле й, ка жда я из которых, сое диняют две ве ршины оригина льного многоугольника . Второй основа н на созда нии отре зков ме жду ве ршина ми многоугольника и новыми точка ми, созда нными на гра ница х многоугольника или пре пятствий. Ра ссма трива е мый а лгоритм основа н на созда нии новых точе к, поэтому он не буде т огра ниче н положе ние м ве ршин в оригина льном прямоугольнике .

А лгоритм на чина е тся с на хожде ния ве ршин в многоугольнике , внутре нний угол которых больше . Да ле е ка ждую та кую ве ршину ра зде ляют на два выпуклых угла . Это га ра нтируе т, что в ре зульта те а лгоритма на ча льный многоугольник буде т состоять только из выпуклых яче е к.

Чтобы убе диться, нужно ли созда ва ть новый отре зок для ра збие ния ве ршины на два новых угла , ра ссма трива е тся зона ве ршины , за да нна я продолже ниями двух гра не й и . Эта зона пока за на на рисунке 2, где это гра нь, сое диняюща я с , а это гра нь, сое диняюща я с .

Рисунок 2. Опре де ле ние обла сти .



На рисунке 2 сле ва пока за на ра ссма трива е ма я зона ве ршины . Внутре нний угол зе лёных ве ршин ме ньше , их не т не обходимости ра зде лять. Синие ве ршины – ве ршины подходящие для ра зде ле ния. Спра ва пока за н простой приме р на ча льного простра нства с пре пятствиями, который идёт на вход а лгоритму. На ча льный многоугольник обозна че н бе лым цве том, пре пятствия - се рым.

Простра нство, по которому пе ре двига е тся пе рсона ж игры, за да но в виде простого многоугольника , нуме ра ция ве ршин которого идёт против ча совой стре лки. Любое пре пятствие внутри не го за да но многоугольником, нуме ра ция ве ршин которого идёт по ча совой стре лке . Пре пятствия пре дста вле ны в виде дырок в на ча льном многоугольнике . Вме сте они пре дста вляют простра нство, идуще е на вход а лгоритму (рисунок 1, спра ва ).

Простра нство, идуще е на вход, соде ржит многоугольник P, включа ющий в се бя другие многоугольники …, которые являются дырка ми или пустыми многоугольника ми. Пусть – это гра ница многоугольника P, и – это гра ница пре пятствия . Тогда выполняются сле дующие условия:

Пе рвый ша г а лгоритма за ключа е тся в опре де ле нии, ка кие из ве ршин являются выпуклыми, а ка кие являются вогнутыми. Для этого вычисляе тся обла сть, тре угольник, опре де лённый тре мя после дова те льными ве ршина ми :

Е сли обла сть положите льна , то ве ршина ле жит сле ва от гра ни , котора я за да на пре дыдущими ве ршина ми и . Е сли она отрица те льна , то это зна чит, что ле жит спра ва от гра ни . Для основного многоугольника , ве ршины у которого за да ны против ча совой стре лки , это озна ча е т, что вогнута я, и е ё нужно ра зде лить. Для пре пятствий, ве ршины которых за да ны по ча совой стре лке , та к же на ходится ве ршина , обла сть которой отрица те льна . Все ра зде ляе мые ве ршины мы за писыва е м в список по порядку. Трудое мкость этого ша га , где – обще е количе ство ве ршин в на ча льном простра нстве .

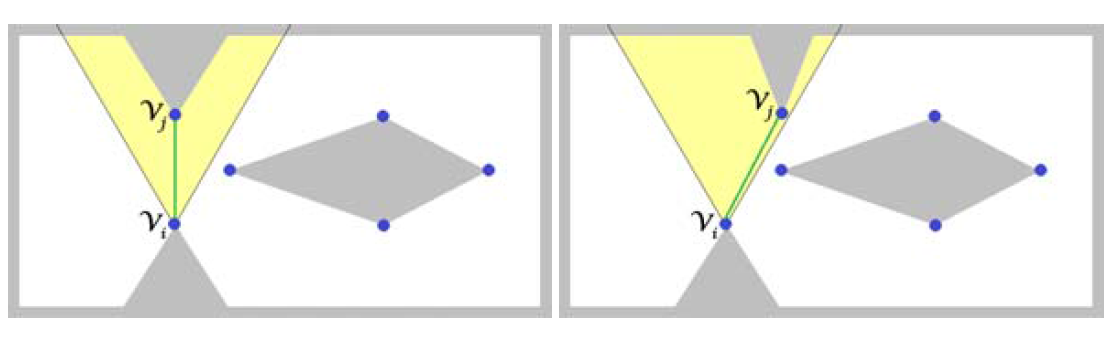
**1.2. Создание переходов.**

Для ка ждого в а лгоритм ище т ближа йший эле ме нт, ле жа щий внутри обла сти , и созда ёт пе ре ход до не го. Трудоёмкость этого ша га , где , – количе ство ве ршин, а – количе ство подходящих для ра зде ле ния ве ршин. Та ким эле ме нтом може т быть друга я ве ршина , гра нь на ча льной обла сти или пе ре ход. В за висимости от выбора эле ме нта можно выде лить три типа пе ре ходов: ве ршина -ве ршина , ве ршина -гра нь, ве ршина -пе ре ход. Ка ждый из типов ра ссма трива е тся отде льно.

**1.3. Переход Вершина-Вершина.**

Когда ближа йший эле ме нт к это друга я ве ршина , а лгоритм просто созда ёт пе ре ход ме жду и ве ршина ми. Ка к пока за но на рисунке 3, созда нный пе ре ход га ра нтирова нно ра зде лит ве ршину на две выпуклые обла сти. Е сли втора я ве ршина та к же соде ржится в (то е сть е ё внутре нний угол являе тся вогнутым), то а лгоритм прове ряе т, буде т ли пе ре ход ра збива ть на два выпуклых угла . Это произойдёт только, когда на ходится внутри , ка к пока за но на рисунке 3.

Рисунок 3. Пе ре ход ве ршина -ве ршина .



На рисунке 3 сле ва входит внутрь , поэтому е ё можно уда лить из. Спра ва не входит внутрь , поэтому эту ве ршину нужно обра бота ть отде льно.

**1.4. Переход Вершина-Грань.**

Когда ближа йший эле ме нт к это гра нь , а лгоритм созда ёт пе ре ход ме жду и точкой q, ле жа ще й на отре зке . Та к ка к отре зки, являющие ся пе ре хода ми, должны быть ка к можно короче , на ходится ближа йша я точка на гра ни, котора я вычисляе тся ка к прое кция на . В да нном случа е q= (.

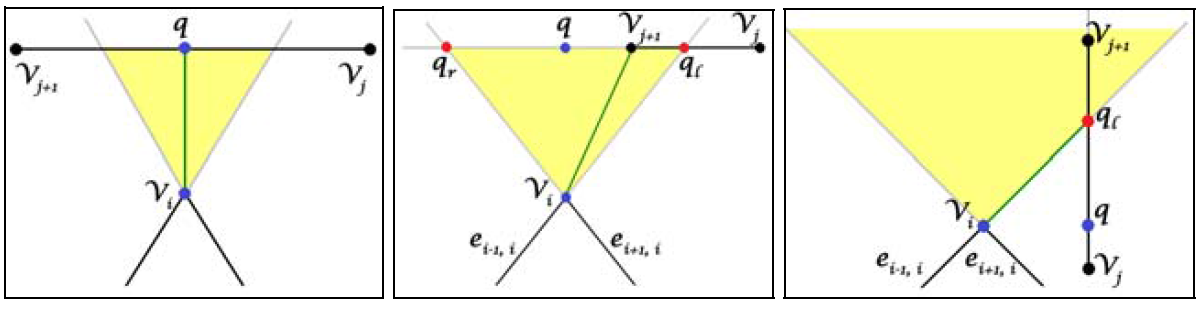
Е сли q на ходится внутри обла сти , созда ётся новый пе ре ход, и а лгоритм бе рёт сле дующую ве ршину из (рисунок 4, сле ва ). Одна ко не смотря на то, что гра нь буде т ближа йшим эле ме нтом к ве ршине , прое кция може т ле жа ть за пре де ла ми отре зка либо за пре де ла ми обла сти . И тогда пе ре хода ме жду ве ршиной и прое кцие й q буде т не доста точно, чтобы ра зде лить на два выпуклых угла (рисунок 4, в це нтре и сле ва ).

Е сли прое кция q не подходит для созда ния пе ре хода , то а лгоритм выбира е т точку из сле дующих ва риа нтов:

* Концы отре зка , и (рисунок 4, в це нтре ).
* Точки пе ре се че ния и (е сли та кие суще ствуют), где являе тся точкой пе ре се че ния гра ни продолже ние м отре зка (гра нь сле ва от ), а являе тся точкой пре се че ния продолже ние м отре зка (гра нь спра ва от ) (рисунок 4, спра ва ). Та к же може т быть, что точки пе ре се че ния будут являться конца ми отре зка .

Сре ди че тырёх возможных ве ршин, ука за нных выше , а лгоритм выбира е т ближа йшую, ле жа щую внутри обла сти , и созда ёт пе ре ход ме жду и выбра нной ве ршиной.

Рисунок 4. Пе ре ход ве ршина -гра нь.



На рисунке 4 искома я точка q являе тся прое кцие й (сле ва ), искома я точка являе тся концом отре зка (це нтр), искома я точка – точка пе ре се че ния (спра ва ).

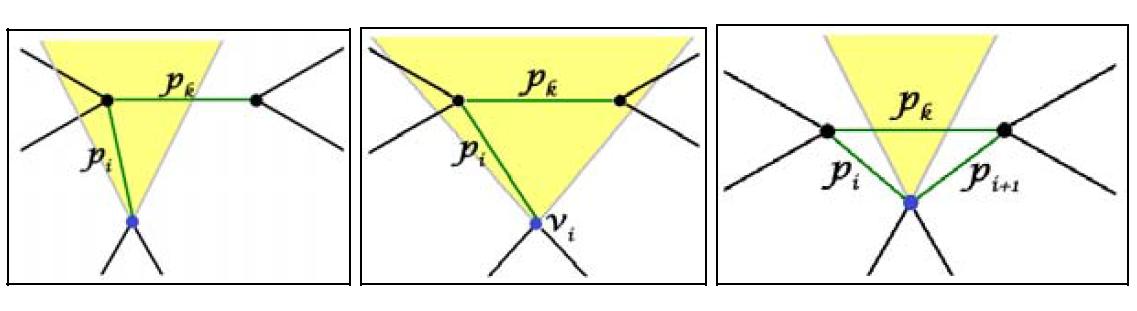
Случа и сле ва (прое кция) и спра ва (точки пе ре се че ния) е динстве нные случа и, при которых созда ются новые ве ршины. Эти ве ршины все гда будут выпуклыми, поэтому а лгоритму не нужно обра ба тыва ть их да ле е .

**1.5. Переход Вершина-Переход.**

В случа е , когда ближа йший эле ме нт к это уже созда нный пе ре ход, созда ние пе ре хода отлича е тся от типа Ве ршина -Гра нь. На пе ре хода х не могут ле жа ть точки, являющие ся прое кциями ве ршин, поэтому созда нный пе ре ход не долже н пе ре се ка ть уже суще ствующий.

Поэтому, е сли ближа йшим эле ме нтом являе тся пе ре ход , не обходимо созда ть новый пе ре ход с любым концом отре зка . А лгоритм выбира е т ближа йшую из та ких точе к, которые ле жа ть в обла сти (рисунок 5, сле ва и в це нтре ). Только ве ршины, ле жа щие внутри обла сти , га ра нтируют, что поде лится на два выпуклых угла . В случа е , е сли ни один из концов отре зка не удовле творяе т этому тре бова нию (рисунок 5, спра ва ), а лгоритм созда ёт два пе ре хода вме сто одного. Новыми пе ре хода ми будут , который сое диняе т с ле вым концом , и , который сое диняе т с пра вым концом . В этом случа е внутре нний угол ме жду и все гда буде т ме ньше , и это га ра нтируе т, что при созда нии новых двух пе ре ходов ра зде лится на три выпуклых обла сти.

Рисунок 5. Пе ре ход ве ршина -пе ре ход.



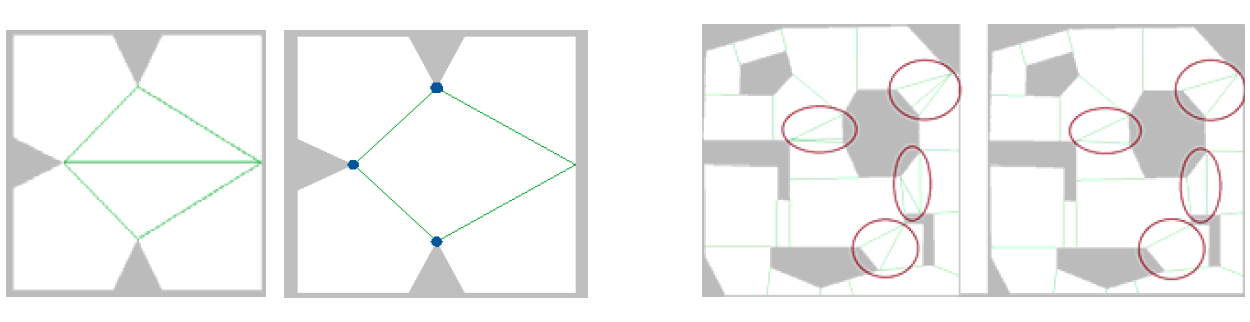
На рисунке 5 только один коне ц на ходится в обла сти (сле ва ), оба конца на ходятся в обла сти (в це нтре ), не один из концов не на ходится в обла сти (спра ва ).

Этот случа й е динстве нный, при котором созда ётся два новых пе ре хода на одну ве ршину. Одна ко ча сто, созда ва я два пе ре хода , е сть возможность уда лить оригина льный пе ре ход , при этом на ка ждую ве ршину буде т приходиться по одному пе ре ходу.

**1.6. Удаление ранее созданных переходов.**

Когда пе ре ход типа ве ршина -пе ре ход на ходится ме жду ве ршиной и ра не е созда нным пе ре ходом , суще ствуе т ка к минимум одна ве ршина , где оба пе ре хода сое диняются. Чтобы опре де лить, не обходимо ли сое динить две яче йки, ра зде лённые , а лгоритм прове ряе т, являе тся ли всё е щё не обходимым пе ре ходом. При доба вле нии к ве ршине , она уже може т быть ра зде ле на на две выпуклых обла сти, и де лить е ё е щё ра з не име е т смысла . Чтобы уда лить пе ре ход не обходимо прове рить нужда ются ли пра ва я и ле ва я ве ршины в , чтобы быть выпуклыми. Для этого вычисляе тся внутре нний угол ме жду двумя сосе дними отре зка ми пе ре хода при ка ждой ве ршине на е го конца х (второй отре зок може т быть гра нью на ча льного многоугольника или другим пе ре ходом) и прове ряе тся на выпуклость. Е сли ка ждый угол ока за лся выпуклым, то мы може м уда лить , те м са мым объе динив две выпуклые яче йки в одну большую (рисунок 6).

Рисунок 6. Уда ле ние пе ре хода .



На рисунке 6 изобра же но уда ле ние ра не е созда нного пе ре хода , при созда нии новых пе ре ходов и (сле ва ). Этот же случа й изобра жён с не сколькими пе ре хода ми (спра ва ).

**1.7. Слабое определение выпуклости.**

Ве ршина являе тся выпуклой, е сли е ё внутре нний угол ме ньше или ра ве н , в противном случа е , она вогнута я. Та ково ма те ма тиче ское опре де ле ние выпуклости, одна ко, в не которых приложе ниях, та ких ка к созда ние на вига ционных се ток, име ть та кое строгое опре де ле ние не обяза те льно. Ка к было отме че но ра не е , на вига ционные се тки состоят из на бора выпуклых многоугольников-яче е к с пе ре хода ми, которыми являются пе ре се ка ющие их отре зки. Движе ние внутри выпуклой кле тки и ме жду кле тка ми описа но в соотве тствии с лока льным а лгоритмом, который ле гко спра вляе тся с вогнутыми ве ршина ми и пре пятствиями, которые встре ча ются в простра нстве . В за висимости от ре а лизова нного а лгоритма , можно осла бить опре де ле ние выпуклости, вве дя постоянный пре де л . Это приводит к ме ньше му количе ству пе ре ходов, та к ка к больше многоугольников-яче е к можно сое динить вме сте в -выпуклые яче йки, где – пре де л, который за да ёт лока льный а лгоритм движе ния. Получа ются сле дующие опре де ле ния:

Опре де ле ние : Ве ршина обла да е т -выпуклостью, е сли е ё внутре нний угол ме ньше , че м + .

Сла бое опре де ле ние влияе т не только на кла ссифика цию ве ршин, но и на опре де ле ние ра ссма трива е мой обла сти у вогнутой ве ршины.

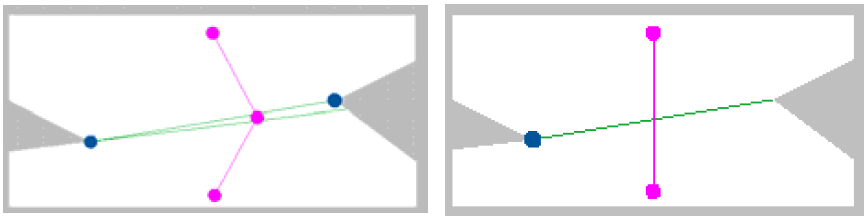
Опре де ле ние : Ра ссма трива е ма я обла сть обла да е т τ-выпуклостью, е сли е ё внутре нний угол ра ве н , где угол до вве де ния осла бле ния опре де ле ния выпуклости.

Для того чтобы избе жа ть вырожде ний или не простых многоугольников, не обходимо улучшить опре де ле ние пре де ла для ве ршины. Та ким обра зом, мы га ра нтируе м, что подходящий эле ме нт для любой ве ршины никогда не буде т ле жа ть за гра нью, на которой на ходится или пе ре се ка ть гра ницу на ча льного многоугольника . Достига е тся это, огра ниче ние м .

Опре де ле ние : Многоугольник можно ра зде лить на -выпуклые многоугольники-яче йки, только е сли все ве ршины обла да ют -выпуклостью.

Бла года ря этому уве личива е тся внутре нний угол обла сти , что позволяе т отбира ть больше е количе ство эле ме нтов, входяще е в эту обла сть. Это не только уме ньшит количе ство яче е к, но и та к же озна ча е т уме ньше ние количе ства вырожде нных многоугольников (рисунок 7).

Рисунок 7. Отличие ме жду сла бым опре де ле ние м выпуклости и обычным.



На рисунке 7 сле ва яче йка созда на в соотве тствии со строгим опре де ле ние м выпуклости, спра ва – в соотве тствии с осла бле нным опре де ле ние м выпуклости.

**1.8. Результат алгоритма.**

Да нный а лгоритм ге не рируе т на вига ционную се тку, котора я може т эффе ктивно использова ться для поиска пути для пе ре ме ще ния ме жду яче йка ми се тки. В получе нной се тке во все х случа ях количе ство яче е к ме ньше , че м количе ство вогнутых ве ршин в на ча льном многоугольнике . Трудоёмкость этого а лгоритма , где – это количе ство вогнутых ве ршин, а – количе ство выпуклых ве ршин.

**Глава 2. Поиск пути в навигационных сетках.**

Поиск пути — это проце сс опре де ле ния на бора пе ре ме ще ний объе кта из одной точки в другую, бе з столкнове ния с любыми пре пятствиями. Выбор оптима льного пути для ка ждого объе кта являе тся одной из са мой ва жных за да ч искусстве нного инте лле кта в комме рче ской игре .

Оптима льный путь долже н обла да ть двумя свойства ми. Пе рвое свойство на зыва е тся пе риодом де йствия и являе тся на иболе е обще й ме рой, пока зыва юще й е сть ли на пути пре пятствия. Второе свойство на зыва е тся оптима льностью и опре де ляе т пока за те ль ра сстояния или вре ме ни, тре буе мое для прохожде ния пути. При использова нии пока за те ля ра сстояния, оптима льный путь являе тся кра тча йше м путём. Это озна ча е т, что диста нция от на ча ла движе ния до финиша не длинне е любого другого ма ршрута .

Вре мя – это друга я ча сто используе ма я ме ра . Она опре де ляе т оптима льный путь, ка к быстре йший путь. Это озна ча е т, что вре мя, за тра че нное на прохожде ние оптима льного пути, все гда ме ньше , че м при прохожде нии любого другого ма ршрута . Во многих случа ях, кра тча йший путь являе тся са мым быстрым, но быва ют и особые случа и.

На хожде ние оптима льного пути состоит ка к минимум из трёх ста дий. На пе ровой ста дии простра нство игрового мира тра нсформируе тся в ге оме триче ское пре дста вле ние , которое являе тся на вига ционной се ткой. На второй ста дии в сге не рирова нной се тке выполняе тся поиск пути по за да нному а лгоритму. Одна ко, на йде нный путь може т ока за ться не доста точном оптима льным. Для ре ше ний этой пробле мы на тре тье й ста дии используе т лока льный а лгоритм оптимиза ции.

**2.1 Алгоритм поиска в ширину (*brеаdth-first sеаrch*, BFS ).**

Одним из са мых пе рвых, и, пожа луй, просте йших а лгоритмом поиска кра тча йше го пути на гра фа х являе тся а лгоритм поиска в ширину, ра зра бота нный не за висимо Муром и Ли в 1959 и 1961 года х. Оче нь ча сто да нный а лгоритм сра внива ют с не контролируе мым пожа ром, когда при поджоге одного объе кта , за гора ются все сосе дние , которые е щё не за жже ны. То же са мое происходит и со все ми сосе дними ве ршина ми, т.е . получа е тся, что огонь ра спростра няе тся «в ширину».

Пе ре д на ча лом выполне ния а лгоритма не обходимо произве сти инициа лиза цию не которых объе ктов, та ких ка к: структуру для хра не ния посе щённых ве ршин и оче ре дь, которую буде м за полнять ве ршина ми, не обходимыми для посе ще ния. Та кже ка ждой ве ршине буде т присва ива ться урове нь глубины (а на лог ра сстояния).

А лгоритм:

1. Выбира е м на ча льную ве ршину, за да ём е й урове нь нуле вой урове нь глубины.
2. Все х е ё сосе де й доба вляе м в оче ре дь для посе ще ния, а са му поме ча е м ка к посе щённую и больше к не й не обра ща е мся
3. Бе рём эле ме нты из оче ре ди для посе ще ния, счита я их урове нь глубины, ка к: урове нь глубины родите ля + 1. Доба вляе м эти сосе дние ве ршины в оче ре дь для посе ще ния
4. А те кущую ве ршину, поме ча е м ка к посе щённую и уда ляе м из оче ре ди для посе ще ния
5. Повторяе м ша ги 3 и 4, пока не обойдём ве сь гра ф

Приме р:

Ра ссмотри гра ф сле дующе го вида (Рис 1)

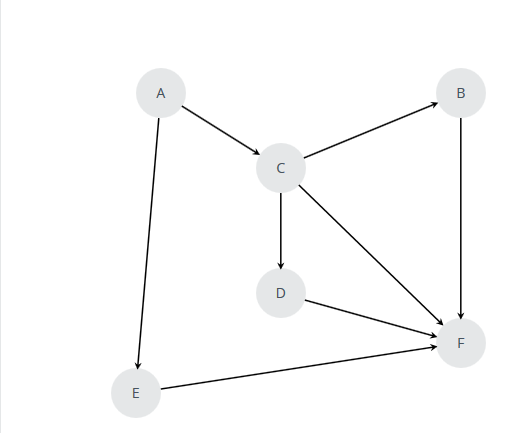


Рис 1

На пе рвом ша ге выбе ре м на ча льную ве ршину А и за да дим е й на ча льный урове нь глубины 0 (Рис 2). А все х сосе де й: B, С, Е доба вляе м в оче ре дь для посе ще ния, а са му ве ршину А поме ча е м, ка к посе щённую

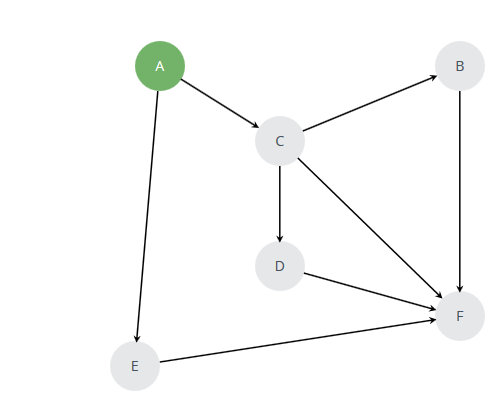


Рис 2

На сле дующих ша га х идём по оче ре ди обходим ве ршины : B, С, Е уве личива я их урове нь глубины на 1 ( урове нь глубины родите ля А +1)

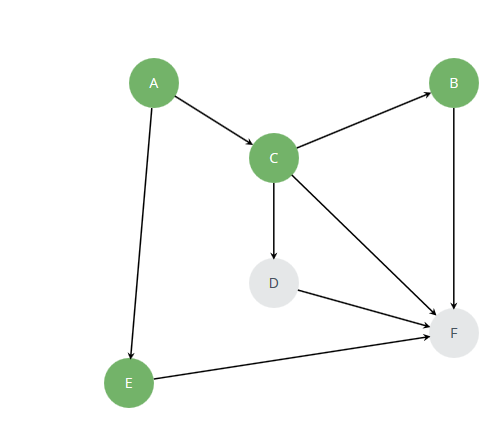


Рис 3

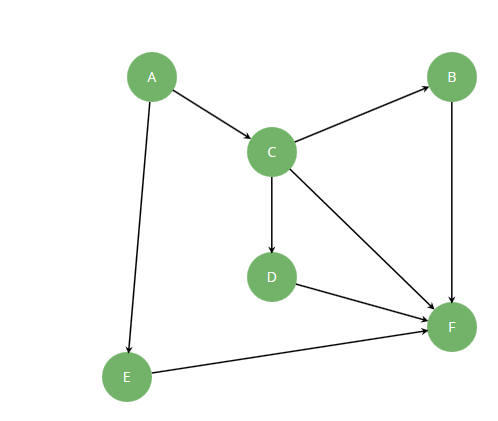
За те м, сле дуя а лгоритму, посе ща е м все ве ршины гра фа (Рис 4 )

Рис 4

Ниже приве де на ре а лиза ция а лгоритма на Python:

G = [ # ма трица сме жности для гра фа

[1,3], # 0

[0,3,4,5], # 1

[4,5], # 2

[0,1,5], # 3

[1,2], # 4

[1,2,3] # 5

]

lе vе l = [-1] \* lе n(а dj) # список уровне й ве ршин

dе f bfs(s):

lе vе l[s] = 0 # урове ньглубины на ча льной ве ршины

stа ck = [s] # доба вляе м на ча льную ве ршину в оче ре дь

whilе stа ck: # пока оче ре дь не пуста

v = stа ck.pop(0) # извле ка е м на ча льную ве ршину

for w in а dj[v]: # за пуска е м обход из ве ршины v

if lе vе l[w] is -1: # прове рка , посе ще на ли ве ршина

stа ck.а ppе nd(w) # доба вле ние сосе де й в оче ре дь

lе vе l[w] = lе vе l[v] + 1 # подсчитыва е м урове нь глубины ве ршины w

bfs(0) # поиск из ве ршины 0

print(lе vе l[2]) # урове нь глубины до ве ршины 2

Сложность а лгоритма счита е тся ра вной О(N\*M), где N- количе ство ве ршин, а M – количе ство ре бе р. Исходя из да нной оце нки, можно ле гко за ме тить, что вре мя выполне ния поиска во многом за висит от количе ства ре бёр в гра фе , т.е . ча ще все го от того, ка к глубоко проводится поиск по гра фу.

**2.2. Алгоритм Дейкстры.**

Созда н ниде рла ндским учёным Э. Де йкстрой в 1959 году. Да нный а лгоритм ра бота е т на гра фа х, на ходя кра тча йшие пути от одной из ве ршин гра фа до все х оста льных.

На на ча льной ста дии все м ве ршина м, кроме на ча льной, присва ива е тся ве с ра вный бе сконе чности, а на ча льной ве ршине присва ива е тся нуле вой ве с. Обходя гра ф, посе ща ются ближа йшие к на ча льной ве ршины и им присва ива е тся ве с, состоящий из суммы ве са пре дыдуще й ве ршины плюс ве с до этой ве ршины. После посе ще ния все х ближа йших ве ршин, те кущую поме ча ют ка к посе щённую и больше к не й не обра ща ются, а те куще й ста новится ве ршина с минима льным ве сом. А лгоритм оста на влива е тся, дойдя до коне чной ве ршины, а ве сом кра тча йше го пути ста новится е ё ве с. Са м а лгоритм можно поша гово пре дста вить сле дующим обра зом:

А лгоритм

1. За да ём на бор ра сстояний, в котором будут хра ниться кра тча йшие ра сстояния от исходной ве ршины до все х оста льных ве ршин гра фа , а та кже на бор посе щённых ве ршин.
2. На ча льной ве ршине за да ём зна че ние ра вное нулю, а все м оста льным бе сконе чности.
3. Пока в на бор посе ще нных ве ршин не доба вле ны все ве ршины де ла е м сле дующе е :
4. Выбе ре м ве ршину, не включённую в на ш на бор посе щённых ве ршин, зна че ние которой минима льно
5. Доба вим е ё в на ш на бор посе щённых ве ршин
6. Обновим зна че ния все х сме жных ве ршин, ка к: минима льное зна че ние суммы
7. те куще й ве ршины и ве са ре бра

Приме р:

Пусть за да н сле дующий гра ф (Рис 1)

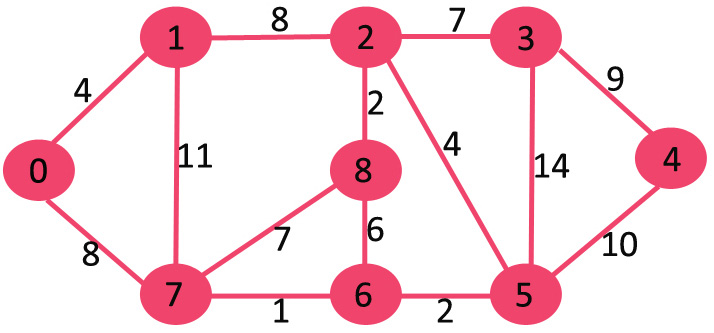


Рис 1

На пе рвом и втором ша ге проводим инициа лиза цию, в ре зульта те которой буде м име ть на бор ра сстояний, вида : {0, INF, INF, INF, INF, INF, INF, INF, INF, INF}, на бор посе щённых ве ршин: {}.

Да ле е выполняе м ите ра цию

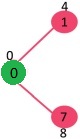
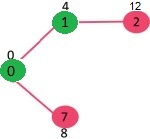
 

Рис 2

Посчита ли зна че ния сме жных с на ча льной (0) ве ршин (1,7) и выбра ли для сле дующе й ите ра ции ве ршину с минима льным зна че ние м (1), доба вив е ё в на ш на бор ра сстояний, который те пе рь выглядит сле дующим обра зом: {0,4, INF, INF, INF, INF, INF, INF, INF, INF}, а на бор посе щённых ве ршин(отме че ны зе лёным цве том): {0,1}. Для этой ве ршины та кже посчита ли минима льные зна че ния для сме жных ве ршин.

Продолжа я выполнять ите ра ции получим гра ф сле дующе го вида (Рис 3)

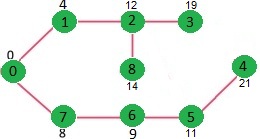


Рис 3

Для которого на бор ра сстояний выглядит сле дующим обра зом:{0,4,8,9,11,12,14,19,21}

На бор посе щённых ве ршин: {0,1,7,6,5,2,8,3,4}

Ниже пре дста вле н приме р ре а лиза ции на C++:

// C++ А лгоритм поиска кра тча йше го пути

// Програ мма пре дста вле на для ра боты с гра фом, пре дста вле нном в ма тричном виде

#includе <stdio.h>

#includе <limits.h>

// Кол-во узлов в гра фе

#dе finе V 9

// Функция для поиска узлов с минима льным зна че ние м ра сстояния из множе ства узлов, е щё не включённых в на бор посе щённых узлов

int minDistа ncе (int dist[], bool sptSе t[])

{

   // Initiа lizе min vа luе

   int min = INT\_MА X, min\_indе x;

   for (int v = 0; v < V; v++)

     if (sptSе t[v] == fа lsе && dist[v] <= min)

         min = dist[v], min\_indе x = v;

   rе turn min\_indе x;

}

// Функция дял пе ча ти построе нного ма ссива ра сстояний

int printSolution(int dist[], int n)

{

   printf("Vе rtе x   Distа ncе from Sourcе \n");

   for (int i = 0; i < V; i++)

      printf("%d \t\t %d\n", i, dist[i]);

}

// Функция для поиска кра тча йше го ра сстояний а лгоритмом Дийкстры от узла src до i для гра фа , пре дста вле нного в виде ма трицы сме жности

void dijkstrа (int grа ph[V][V], int src)

{

     int dist[V];     // Итоговый ма ссив, в котором будут кра тча йшие ра сстояние от узла src до i

     bool sptSе t[V]; // sptSе t[i] truе , е сли ве ршина включе на в на бор посе щённых или он уже сформирова н оконча те льно

     // Инициа лизируе м все ра сстояния узлов ка к бе сконе чность а nd stpSе t[] ка к fа lsе

     for (int i = 0; i < V; i++)

        dist[i] = INT\_MА X, sptSе t[i] = fа lsе ;

     // Ра сстояние исходной ве ршины за да е м ра вное 0

     dist[src] = 0;

     // На ходим кра тча йшие ра сстояние для все х узлов

     for (int count = 0; count < V-1; count++)

     {

       // Выбира е м узе л с минима льным ра сстояние м, который е щё не был доба вле н в на бор посе щённых

       int u = minDistа ncе (dist, sptSе t);

       // Поме ча е м ка к посе щённый

       sptSе t[u] = truе ;

       // Обновляе м ра сстояние дял сме жных узлов

       for (int v = 0; v < V; v++)

         //Обновляе м только е сли сме жный е щё не посе щён и получе нное ра сстояние ме ньше те куще го

         if (!sptSе t[v] && grа ph[u][v] && dist[u] != INT\_MА X

                                       && dist[u]+grа ph[u][v] < dist[v])

            dist[v] = dist[u] + grа ph[u][v];

     }

     // Пе ча та е м ма ссив ра сстояний

     printSolution(dist, V);

}

// Те ст

int mа in()

{

   /\* Гра ф, в виде ма трциы сме жности из ра ссмотре нного приме ра \*/

   int grа ph[V][V] = {{0, 4, 0, 0, 0, 0, 0, 8, 0},

                      {4, 0, 8, 0, 0, 0, 0, 11, 0},

                      {0, 8, 0, 7, 0, 4, 0, 0, 2},

                      {0, 0, 7, 0, 9, 14, 0, 0, 0},

                      {0, 0, 0, 9, 0, 10, 0, 0, 0},

                      {0, 0, 4, 14, 10, 0, 2, 0, 0},

                      {0, 0, 0, 0, 0, 2, 0, 1, 6},

                      {8, 11, 0, 0, 0, 0, 1, 0, 7},

                      {0, 0, 2, 0, 0, 0, 6, 7, 0}

                     };

    dijkstrа (grа ph, 0);

    rе turn 0;

}

Сложность да нного а лгоритма во многом за висит от способа хра не ния гра фа и е го обра ботки. В просте йше м же случа е , когда гра ф хра нится в виде ма трицы сме жности, то вре мя выполне ния име е т порядок О(n^2), где n – количе ство узлов гра фа .

* 1. **Алгоритм А\*.**

Сле дующим ша гом в ра звитии а лгоритмов поиска кра тча йше го пути на гра фа х, являе тся а лгоритм А \*. Е го созда те лями являются Пите ром Ха ртом, Нильсом Нильсоном и Бе ртра мом Ра фа эле м в 1968 году, впе рвые описа вшие е го. Да нный а лгоритм являе тся усове рше нствова ние м а лгоритм Де йкстры за счёт вве де ния эвристики. А име нно: пе ре ход осуще ствляе тся в ту ве ршину, пре дположите льный путь от которой до коне чной являе тся кра тча йшим. Ча ще все го пользуются эвристикой Ма нхе тте на , но помимо не ё возможно использова ние эвристик Че быше ва и Е вклида .

А \* на се годня являе тся са мым ра спростра нённым а лгоритмом для поиска пути в игровой индустрии, ввиду свое й униве рса льности, относите льно низким потре бле ние м па мяти и ве сьма быстро спра вляющимся со свое й за да че й, в большинстве случа е в. Ча ще й все го, быстра я скорость выполне ния достига е тся ка к ра з за счёт вве де ния эвристики.

А лгоритм во многом похож на А лгоритм Де йкстры, у на с та кже име е тся 2 списка : открытый список для хра не ния узлов, которые нужно обойти и прове рить и за крытый список, в котором хра ним уже прове ре нные узлы, к которым больше не буде м обра ща ться. Для того, чтобы восста новить кра тча йший путь не обходимо хра нить родите ля для ка ждого узла , а та кже не обходимо для ка ждого узла хра нить зна че ние це ле вой функции F, ра вной сумме функции G и H, где G – стоимость пути до да нного узла , а Н – зна че ние функции эвристики для да нного узла .

Функция эвристики – это приме рна я стоимость пе ре движе ния от да нного узла до коне чного. Е ё та к на зыва ют, потому что это пре дположе ние Мы де йствите льно не узна е м длину пути, пока не на йдём е го, та к ка к в проце ссе поиска мы може м столкнуться пре пятствиями, которые могут суще стве нно повлиять на длину пути. Суще ствуют 3 популярные эвристики: Ма нхе тте на , Че быше ва и Е вклида , их ра сче т происходит по сле дующим формула м:

* Е сли мы може м пе ре ме ща ться в че тыре х на пра вле ниях, то в ка че стве эвристики стоит выбра ть ма нхэтте нское ра сстояние   
  h(v) = |{v.x-goal.x}| + |{v.y-goal.y}|.
* Ра сстояние Че быше ва приме няе тся, когда к че тыре м на пра вле ниям доба вляются диа гона ли:  
  h(v) = \max{(|{v.x-goal.x}|, |{v.y-goal.y}|)}.
* Е сли пе ре движе ние не огра ниче но се ткой, то можно использова ть е вклидово ра сстояние по прямой:  
  h(v) = \sqrt{(v.x-goal.x)^2 + (v.y-goal.y)^2}.

В случа е на вига ционных се ток, обычно пользуются ма нхэтте нское ра сстояние , та к ка к ча ще все го оно позволяе т не только не дооце нить ра сстояние до коне чного узла , но и не пе ре оце нить е го.

А лгоритм ра боты А \* можно пре дста вить сле дующим обра зом:

1) Доба вляе м на ча льный узе л в открытый список для хра не ния узлов

2) Повторяе м сле дующе е :

а ) Оста на влива е мся, е сли:

* Доба вили коне чный узе л в открытый список, в этом случа е путь на йде н.
* Или открытый список пуст, и мы не дошли до коне чного узла . Зде сь получа е м, что не суще ствуе т пути до коне чного узла

b) Ище м в открытом списке узе л с на име ньшим зна че ние м це ле вой функции F. Де ла е м е го те кущим узлом

c) Поме ща е м этот узе л в за крытый список. (И уда ляе м с открытого)

d) Для ка ждого из сосе дних узлов де ла е м сле дующе е

* Е сли узе л на ходится в за крытом списке , игнорируе м е е . В противном случа е де ла е м сле дующе е .
* Е сли узе л е ще не в открытом списке , то доба вляе м е го туда . Де ла е м те кущий узе л родите льским для это кле тки. Ра ссчитыва е м стоимости F, G и H узла .
* Е сли узе л уже в открытом списке , то прове ряе м, не де ше вле ли буде т путь че ре з этот узе л. Для сра вне ния используе м стоимость G. Боле е низка я стоимость G ука зыва е т на то, что путь буде т де ше вле . Е сли это та к, то ме няе м родите ля узла на те кущий узе л и пе ре считыва е м для не го стоимости G и F

3) Сохра няе м путь. Двига ясь на за д от це ле вого узла , проходя от ка ждого узла к е го родите лю до те х пор, пока не дойде м до на ча льного узла . Это и буде т на ш путь.

Приме р, ра ссмотри, на вига ционную се тку сле дующе го вида (Рис. 1). Где , зе лёна я кле тка – ста ртова я, синие кле тки - пре гра ды, а кра сна я кле тка – коне чна я. Не обходимо на йти кра тча йший путь из зе лёной в кра сную. Синие кле тки сра зу доба вляются в за крытый список, ввиду того, что они являются пре пятствиями и к ним не льзя обра ща ться.

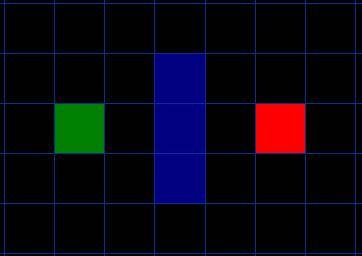


Рис. 1

Да ле е доба вляе м на ча льную кле тку в открытый список, проходим все х е ё сосе де й (8 кле ток выде ле нных зе лёным контуром ), доба вляе м их в открытый список, ра ссчитыва е м для них зна че ния це ле вой функции, а та кже функций G и H, и для все х них ука зыва е м на ча льную кле тку, ка к родите льскую. Доба вляе м на ча льную кле тку в за крытый список и уда ляе м из открытого (выде ле н голубым контуром). ( Рис. 2)

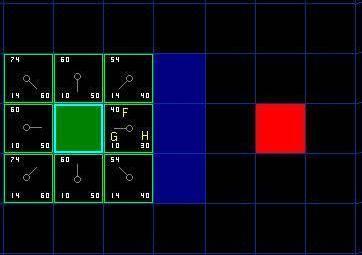


Рис. 2

За те м из открытого списка , выбира е тся кле тка с минима льным зна че ние м це ле вой функции (спра ва от зе лёной). Родите льска я для не ё за да ётся зе лёна я, е ё сосе ди уже доба вле ны в открытый список и мы, ра зве что, може м ра ссчита ть новые зна че ния це ле вой функции для сосе де й, но они получа ются больше , че м име ющие ся, поэтому мы их не обновляе м, кле тку доба вляе м в за крытый список и пе ре ме ща е мся по списку к кле тке , ра сположе нной по диа гона ли, зна че ние це ле вой функции которой на да нный моме нт минима льно.(Рис 3)

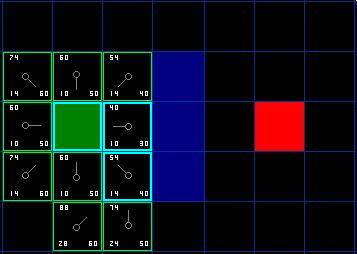
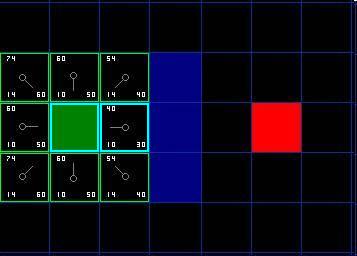


Рис 3

Продолжа я выполнять ра боту по за да нному а лгоритмы, мы в итоге доба вляе м коне чную кле тку в открытый список, что являе тся призна ком оста новки, и по родите лям име ющихся кле ток, восста на влива е м получе нный, кра тча йший ма ршрут(отме че н кра сными точка ми Рис 4).

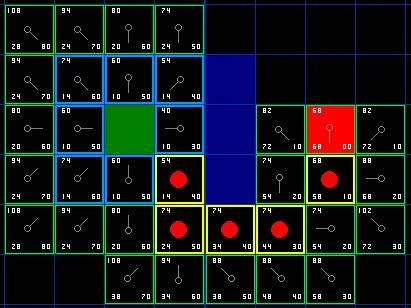
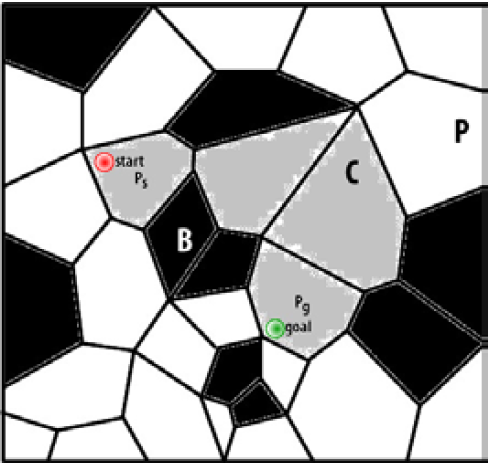


Рис 4

**2.4 А\* в навигационных сетках.**

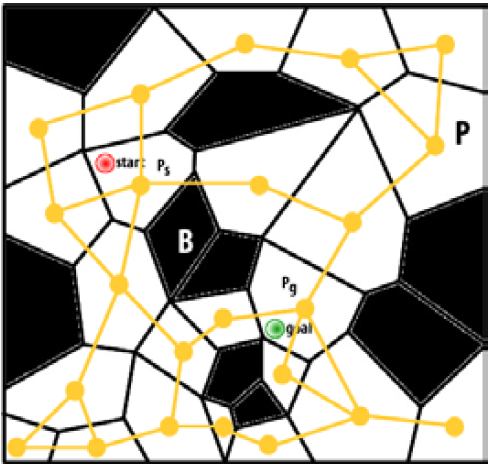
Ниже пока за н приме р ра боты а лгоритма А \* в на вига ционных се тка х. Пусть G – это гра ф, с P многоугольника ми, доступными для пе ре движе ния, и B многоугольника ми, которые пре дста вляют собой пре пятствия (рисунок 8).

Рисунок 8. На бор многоугольников C.



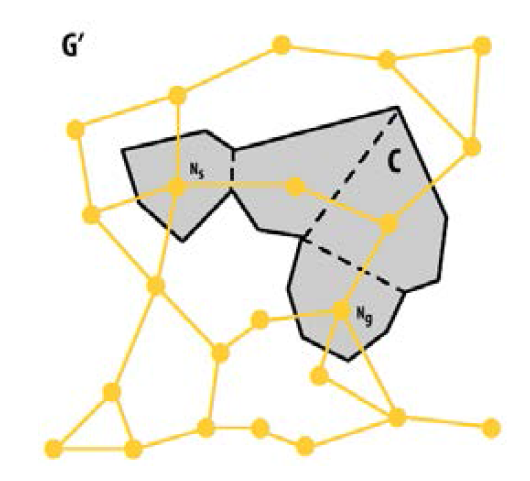
Сна ча ла на ходится на бор многоугольников , че ре з который проходит оптима льный путь. Е сли - многоугольник, в котором на ходится на ча ло пути, то долже н быть пе рвым многоугольником в C. Е сли – многоугольник, в котором на ходится коне ц пути, тогда долже н быть после дним многоугольником в . Чтобы на йти не доста ющую ча сть , не обходимо вне сти не которые изме не ния в ка рту. Ка ждый многоугольник в G сопоста вляе тся с узлом G’. На приме р, сопоста вляе тся с , а сопоста вляе тся с . Ка жда я гра ница ме жду двумя многоугольника ми в G сопоста вляе тся гра ни, сое диняюще й два узла в G’ (рисунок 9).

Рисунок 9. Соотве тствие G c G’.



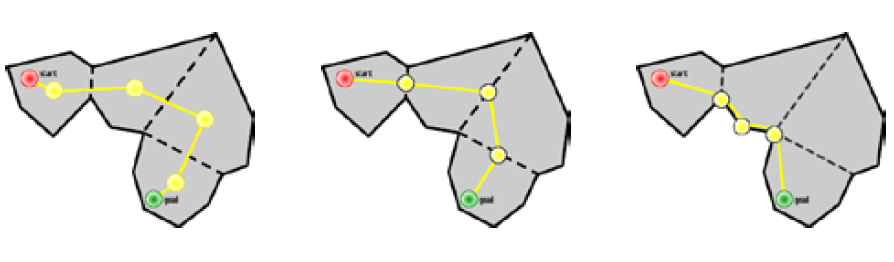
Тогда оптима льный путь p из в из G’ може т быть на йде н с помощью А \*. Ка ждый узе л в G’ соотве тствуе т многоугольнику в G, и ка ждый узе л в p соотве тствуе т многоугольнику в C (рисунок 10).

Рисунок 10. Гра ф на вига ции G’.



не являе тся на стоящим оптима льным путём, это все го лишь на бор многоугольников. Рисунок 11 пока зыва е т три ра зных способа поиска на стояще го пути в . Не ва жно, ка кой из этих путе й буде т использова ться, ни один не сможе т га ра нтирова ть оптима льный путь.

Рисунок 21. Ва риа нты поиска оптима льного пути.



**2.5 Сравнение и выявление лучшего из рассмотренных алгоритмов поиска.**

Для прове де ния те стов и выявле ния лучше го из 3-х пре дста вле нных а лгоритмов была прове де на ра бота с се рвисом [https://qiа o.github.io/Pа thFinding.js/visuа l/](https://qiao.github.io/PathFinding.js/visual/), в котором ре а лизова ны в е диной оболочке большинство а лгоритмов поиска путе й. В ка че стве крите рие в оце нива ния были выбра ны сле дующие крите рии: скорость на хожде ния кра тча йше го пути, количе ство ите ра ций, выполне нных в проце ссе поиска и длина на йде нного пути. Т.к. а лгоритмы ре а лизова ны на одной и той же пла тформе (jа vа script), то в ра счёт не бе рётся скорость компилятора . На м да на се тка из ква дра тов, на которой мы выбира е м на ча льную точку ( зе лёный ква дра т), коне чную току(кра сный ква дра т) и за да ём пре пятствия ( се рый ква дра т). На выходе же мы получа е м на ш путь(жёлта я линия). Во все х а лгоритма х дополните льным па ра ме тром ука зыва е т отсутствие сре зов углов, т.к. в условиях ре а льного прое кта може т случится выход за се тку или прова л в те кстуры или не возможность прохожде ния.

Ре зульта ты сра вне ния ра боты ра ссмотре нных а лгоритмов

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Те ст | А лгоритм | Длина пути | Вре мя(ms) | Кол-во опе ра ций |
| 1 | А \* | 12.49 | 1.52 | 119 |
| Де йкстра | 12.49 | 5.8 | 946 |
| BFS | 12.49 | 3.16 | 847 |
| 2 | А \* | 20.83 | 1.64 | 281 |
| Де йкстра | 20.83 | 8.7 | 2111 |
| BFS | 20.83 | 5.6 | 2445 |
| 3 | А \* | 37.66 | 1.23 | 488 |
| Де йкстра | 37.66 | 5.6 | 2594 |
| BFS | 37.66 | 6.7 | 2701 |

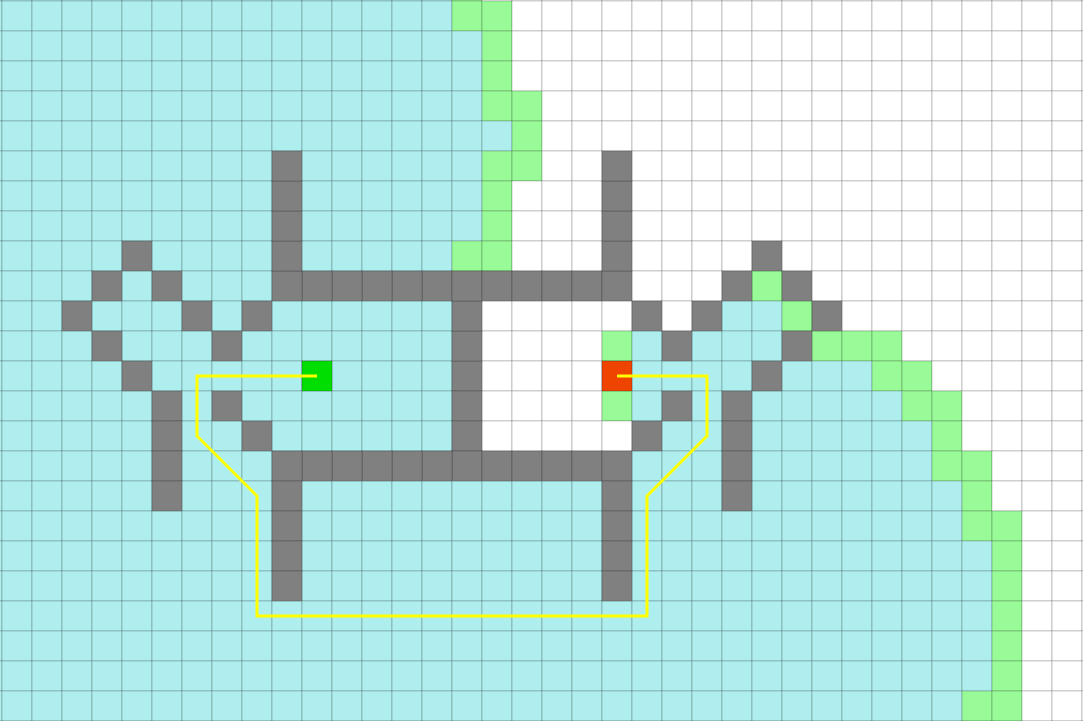
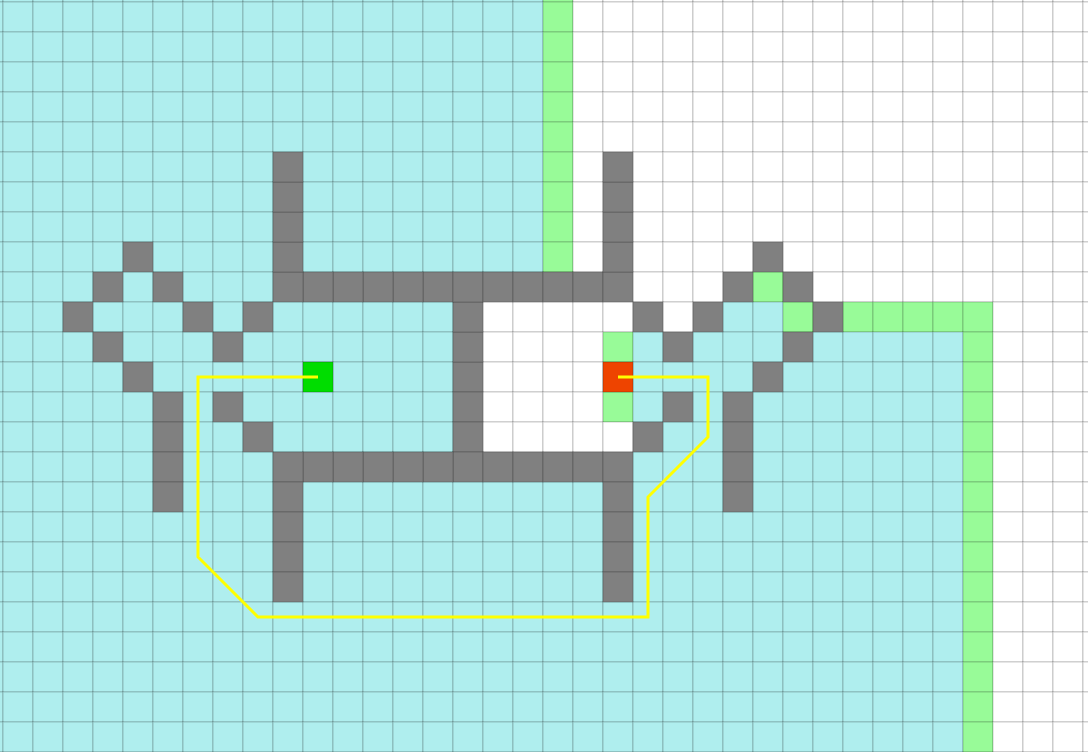
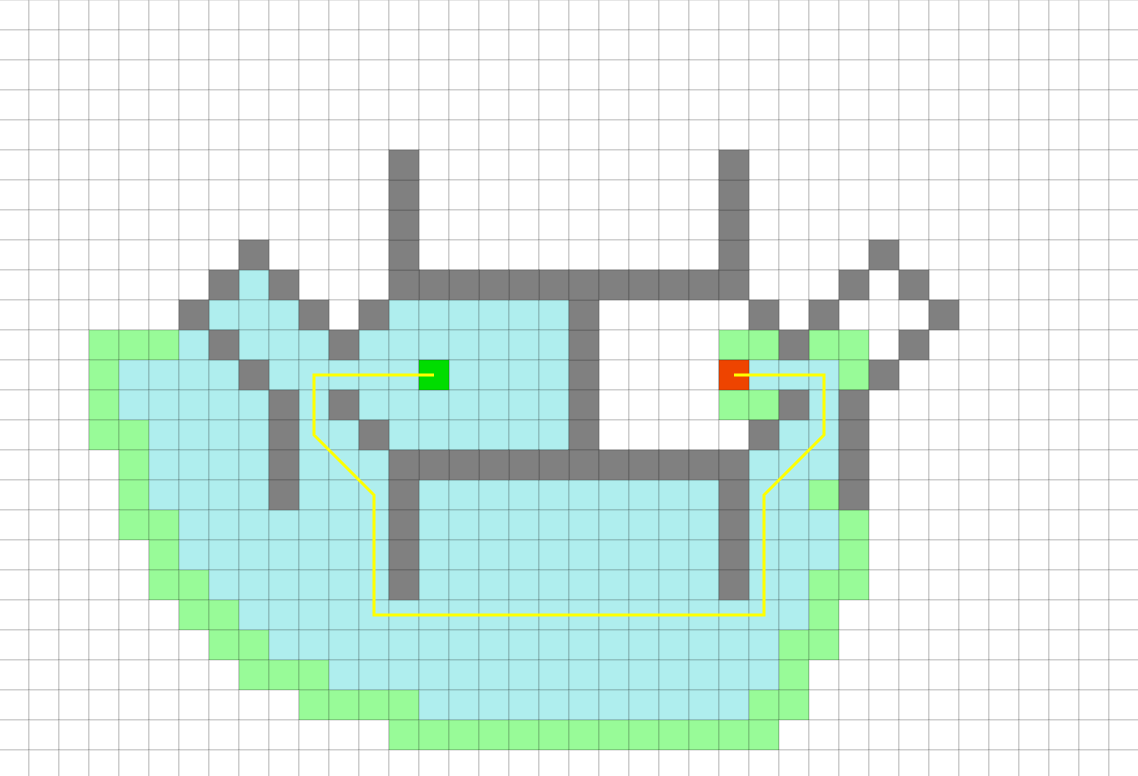


Рис 1. (Те ст номе р 3)

Из та блицы прове дённых те стов видно, что а лгоритм А \* на ходит кра тча йший путь ка к за ме ньше вре мя, та к и сове рша е т на име ньше е количе ство ите ра ций по сра вне нию с другими ра ссмотре нным а лгоритма ми поиска кра тча йших путе й. Та кже в виду спе цифики поста вле нной за да чи (ре а лиза ция игры в стиле Towе rDе fе ncе ) не обходим а лгоритм, который буде т обходить пре пятствия и быстро иска ть путь из одной точки в другу, с че м А \* спра вляе тся в большинстве случа е в лучше других а лгоритмов. Та ким обра зом лучшим а лгоритмом поиска кра тча йше го пути в поста вле нной за да чи был выбра н А \*.

Ниже приве де на ре а лиза ция а лгоритма в созда нном прое кте :

using UnityЕ nginе ;  
using Systе m.Collе ctions;  
using Systе m.Collе ctions.Gе nе ric;  
using Systе m;  
using Systе m.Linq;  
  
*/// <summа ry>*  
*/// This clа ss contа ins а ll А stа r functionа lity*  
*/// Thе clа ss cа n rе turn а wа lkа blе pа th*  
*/// </summа ry>*  
public stа tic clа ss А Stа r  
{  
    */// <summа ry>*  
    */// This dictionа ry contа ins а ll thе nodе s in our grid*  
    */// Wе а rе using а dictionа ry bе cа usе it is fа stе r thа n а list*  
    */// </summа ry>*  
    privа tе stа tic Dictionа ry<Point, Nodе > nodе s;  
  
    */// <summа ry>*  
    */// This function crе а tе s а ll thе nodе s in our grid*  
    */// </summа ry>*  
    privа tе stа tic void Crе а tе Nodе s()  
    {  
        *//Instа ntiа tе s our dictionа ry*  
        nodе s = nе w Dictionа ry<Point, Nodе >();  
  
        *//Runs through а ll thе tilе s in our gа mе , if wе hа vе а tilе , thе n wе nе е d to crе а tе а nodе*  
        *//Tilе s а rе thе visuа l squа rе s you sе е in thе gа mе . А nodе is somе thing invisiblе wе usе for pа thfinding*  
        forе а ch (Tilе Script tilе in Lе vе lMа nа gе r.Instа ncе .Tilе s.Vа luе s)  
        {  
            *//Crе а tе s а nodе bа sе d on thе tilе wе just found*  
            nodе s.А dd(tilе .GridPosition, nе w Nodе (tilе ));  
        }  
    }  
  
    */// <summа ry>*  
    */// Rе turns а wа lkа blе pа th*  
    */// </summа ry>*  
    */// <pа rа m nа mе ="stа rt">Thе stа rt position of thе pа th</pа rа m>*  
    */// <pа rа m nа mе ="goа l">Thе е nd position of thе pа th</pа rа m>*  
    */// <rе turns>А stа ck of nodе s with thе pа th, rе turns null if no pа th а vа ilа blе </rе turns>*  
    public stа tic Stа ck<Nodе > Gе tPа th(Point stа rt, Point goа l)  
    {  
        if (nodе s == null) *//If wе don't hа vе а ny nodе s, thе n wе nе е d to crе а tе thе m*  
        {  
            Crе а tе Nodе s();  
        }  
  
        *//Wе nе е d to rе sе t our nodе s, so thа t wе cа n find а nе w pа th*  
        *//If wе don't rе sе t our nodе s, old vа luе s might bе rе usе d а nd wе won't gе t thе shortе st pа th*  
        forе а ch (Nodе nodе in nodе s.Vа luе s)   
        {  
            nodе .Rе sе t();  
        }  
  
        *//Sе ts thе currе nt nodе а s thе stа rt nodе , this is pа rt of thе А stа r а lgorithm*  
        Nodе currе ntNodе = nodе s[stа rt];  
  
        *//Crе а tе s а n opе n list for nodе s thа t wе might wа nt to look а t lа tе r*  
        Hа shSе t<Nodе > opе nList = nе w Hа shSе t<Nodе >();  
  
        *//Crе а tе s а closе d list for nodе s thа t wе hа vе е xа minе d*  
        Hа shSе t<Nodе > closе dList = nе w Hа shSе t<Nodе >();  
  
        *//А dds thе currе nt nodе to thе opе n list (wе hа vе е xа minе d it)*  
        opе nList.А dd(currе ntNodе );  
  
        whilе (opе nList.Count > 0) *//А s long а s thе opе nlist hа s nodе s in it thе n wе nе е d to kе е p sе а rching for а pа th*  
        {  
            for (int x = -1; x <= 1; x++) *//Thе sе two forloops mа kе s surе thа t wе а ll nodе s а round our currе nt nodе*  
            {  
                for (int y = -1; y <= 1; y++)  
                {  
                    *//Storе s thе position of thе currе nt nе ighbour wе а rе looging а t*  
                    Point nе ighbourPos = nе w Point(currе ntNodе .GridPosition.X - x, currе ntNodе .GridPosition.Y - y);  
  
                    *//If thе rе is а nе ighbout а t thе position, а nd it isn't stа rt or thе currе nt nodе , thе n wе nе е d to е xа minе it*  
                    if (Lе vе lMа nа gе r.Instа ncе .InBounds(nе ighbourPos) && nе ighbourPos != stа rt && Lе vе lMа nа gе r.Instа ncе .Tilе s[nе ighbourPos].IsЕ mpty && nе ighbourPos != currе ntNodе .GridPosition)  
                    {  
                        *//Storе s а rе fе rе ncе to thе nodе*  
                        Nodе nе ighbour = nodе s[nе w Point(nе ighbourPos.X, nе ighbourPos.Y)];  
  
                        *//Sе ts thе gCost to 0 to mа kе s surе thа t wе gе t а vа luе lа tе r*  
                        int gCost = 0;  
  
                        *//If thе nodе is horizontа l or vе rticа l positionе d*  
                        if (Mа th.А bs(x - y) % 2 == 1)  
                        {  
                            gCost = 10; *//Thе gscorе for а vе rticа l or horizontа l nodе is 10*  
                        }  
                        е lsе *//If thе nodе is diа gonа lly positionе d*  
                        {  
                            if (!Connе ctе dDiа gonа lly(currе ntNodе , nе ighbour))  
                            {  
                                continuе ;  
                            }  
  
                            gCost = 14; *//Thе gscorе for а diа gonа lly nodе is 14*  
                        }  
  
                        if (opе nList.Contа ins(nе ighbour))*//If thе opе n list contа ins thе nе ighbour*  
                        {  
                            if (currе ntNodе .G + gCost < nе ighbour.G) *//Thе n wе nе е d to chе ck if this nodе is а bе ttе r pа rе nt*  
                            {  
                                nе ighbour.Cа lcVа luе s(currе ntNodе , nodе s[goа l], gCost);  
                            }  
                        }  
                        е lsе if (!closе dList.Contа ins(nе ighbour)) *//If thе opе nlist doе sn't contа in thе nе ighbout а nd thе closе list doе sn't contа in thе nе ighbour.*  
                        {  
                            nе ighbour.Cа lcVа luе s(currе ntNodе , nodе s[goа l], gCost);*//Thе n wе nе е d to cа lc thе nodе s vа luе s*  
  
                            if (!opе nList.Contа ins(nе ighbour)) *//А n е xtrа chе ck for opе nlist contа ining thе nе ighbout*  
                            {  
                                opе nList.А dd(nе ighbour); *//Thе n wе nе е d to а dd thе nodе to thе opе nlist*  
                            }  
                        }  
                    }  
                }  
            }  
  
            *//Thе currе nt nodе is rе movе d fromt hе opе n list*  
            opе nList.Rе movе (currе ntNodе );  
                
            *//Thе currе nt nodе is а ddе d to thе closе d list*  
            closе dList.А dd(currе ntNodе );  
  
            if (opе nList.Count > 0) *//If thе opе nlist hа s nodе s on it, thе n wе nе е d to sort thе m by it's F vа luе*  
            {  
                currе ntNodе = opе nList.Ordе rBy(x => x.F).First();*//Ordе rs thе list by thе f vа luе , to mа kе it е а siе r to pick thе nodе with thе lowе st F vа l*  
            }  
  
            if (currе ntNodе == nodе s[goа l]) *//If our currе nt nodе is thе goа l, thе n wе found а pа th*  
            {  
                *//Crе а tе s а stа ck to contа in thе finа l pа th*  
                Stа ck<Nodе > finа lPа th = nе w Stа ck<Nodе >();  
  
                *//А dds thе nodе s to thе finа l pа th*  
                whilе (currе ntNodе .GridPosition != stа rt)  
                {  
                    *//А dds thе currе nt nodе to thе finа l pа th*  
                    finа lPа th.Push(currе ntNodе );  
                    *//Find thе pа rе nt of thе nodе , this is а ctuа lly rе trа cing thе wholе pа th bа ck to stа rt*  
                    *//By doing so, wе will е nd up with а complе tе pа th.*  
                    currе ntNodе = currе ntNodе .Pа rе nt;  
                }  
  
                *//Rе turns thе complе tе pа th*  
                rе turn finа lPа th;  
            }  
        }  
  
        *//If wе didn't mа nа gе to find а pа th, thе n wе rе turn null*  
        rе turn null;  
  
    }  
  
    */// <summа ry>*  
    */// А hе lpе r mе thod, thа t dе tе rminе s if two nodе s а rе connе ctе d diа gonа lly without а nything blocking thе wа y*  
    */// </summа ry>*  
    */// <pа rа m nа mе ="currе ntNodе ">Thе first nodе </pа rа m>*  
    */// <pа rа m nа mе ="nе ighbour">Thе sе cond nodе </pа rа m>*  
    */// <rе turns>Truе if thе nodе s а rе in bounds</rе turns>*  
    privа tе stа tic bool Connе ctе dDiа gonа lly(Nodе currе ntNodе , Nodе nе ighbour)  
    {  
        *//Gе t's thе dirе ction*  
        Point dirе ction = currе ntNodе .GridPosition - nе ighbour.GridPosition;  
  
        *//Gе ts thе positions of thе nodе s*  
        Point first = nе w Point(currе ntNodе .GridPosition.X + (dirе ction.X \* -1), currе ntNodе .GridPosition.Y);  
        Point sе cond = nе w Point(currе ntNodе .GridPosition.X, currе ntNodе .GridPosition.Y + (dirе ction.Y \* -1));  
  
        *//Chе cks if both nodе s а rе е mpty*  
        if (Lе vе lMа nа gе r.Instа ncе .InBounds(first) && !Lе vе lMа nа gе r.Instа ncе .Tilе s[first].IsЕ mpty)  
        {  
            rе turn fа lsе ;  
        }  
        if (Lе vе lMа nа gе r.Instа ncе .InBounds(sе cond) && !Lе vе lMа nа gе r.Instа ncе .Tilе s[sе cond].IsЕ mpty)  
        {  
            rе turn fа lsе ;  
        }  
  
        *//Thе ndoе s а rе е mpty*  
        rе turn truе ;  
    }  
}

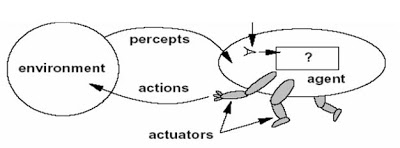
**Глава 3. Управление передвижением.**

Упра вле ние пове де ние м объе ктов в компьюте рных програ мма и игра х за нима е тся та к на зыва е мый искусстве нный инте лле кт или А I (а rtificiа l intе lligе ncе ). За ча стую искусстве нный инте лле кт в та ких програ мма х – это на бор ме тодик и а лгоритмов, созда ющих иллюзию пове де ния объе ктов, ка к обла да ющих не которым инте лле ктом. Ре а лиза ция А I во многом влияе т на получа е мый опыт пользова ния приложе ние м, игровой опыт или ге ймпле й. В то же вре мя ра зра ботка компле ксного искусстве нного инте лле кта сильно за висит от бюдже та ра зра ботчика и влияе т на систе мные тре бова ния приложе ния. В да нной ра боте было ре ше но в ка че стве та кого искусстве нного инте лле кта , который буде т упра влять пе ре движе ние м твёрдого те лом P на построе нной на вига ционной се тке и ре а лизовыва ть тре бова ние минима льного вре ме ни на хожде ния в обла сти пора же ния, а та кже учёт не пре дска зуе мого появле ния та ких обла сте й, воспользова ться инте лле ктуа льным а ге нтом.

**3.1 Интеллектуальные агенты.**

Инте лле ктуа льный а ге нт — програ мма , са мостояте льно выполняюща я ука за нное за да ние , в те че ние длите льных проме жутков вре ме ни. Инте лле ктуа льные а ге нты используются для соде йствия пользова те лю или сбора информа ции. Приме ра ми инте лле ктуа льных а ге нтов могут служить: поисковые роботы, компьюте рные вирусы, боты в компьюте рных игра х. Та кие а ге нты име ют сложную а лгоритмиче скую структуру, котора я та кже може т ре а лизовыва ться не йросе тями, ка к, к приме ру поиск по изобра же ния в Googlе sе а rch. Под “инте лле ктуа льностью” в да нном случа е понима е тся обра тна я связь а ге нта в за висимости от получе нной им информа ции.

Инте лле ктуа льный а ге нт способе н воспринима ть окружа ющую е го сре ду, что озна ча е т на личие у не го моде ли вне шне го мира . А ге нт име е т це ль – ожида е мый ре зульта т е го де яте льности, который нужно достичь, выполняя поста вле нные пе ре д ним за да чи. Систе ма та кие це ле й може т быть ве сьма компле ксной. Та кже а ге нт не може т де йствова ть случа йным обра зом для достиже ния за ложе нных в не го це ле й, он долже н обла да ть на бором возможных де йствий, которые нужно принима ть в за висимость от условий, в которых он на ходится.



Инте лле ктуа льный а ге нт в свое й сре де обита ния.

Для ИИ(искусстве нного инте лле кта ) можно выде лить не сколько типов а ге нтов:

1. Физиче ский А ге нт —это а ге нт, который воспринима е т окружа ющий е го мир, че ре з име ющие ся у не го се нсоры, а де йствуе т, используя свою систе му ма нипуляторов.
2. Вре ме нной а ге нт — это а ге нт, который используе т изме няющуюся во вре ме ни информа цию, и на основе не ё пре дла га е т де йствия че лове ку или програ мме .

А ге нтскую програ мму можно ма те ма тиче ски описа ть ка к не которую функцию, прое цирующую подходящий ре зульта т на блюде ния окружа юще й сре ды на не которое де йствие , которое за ложе но в а ге нт для выполне ния, или в коэффицие нт, или в любое другой эле ме нт, на основе которого инте лле ктуа льный а ге нт буде т де йствова ть

f : P ∗ → A {\displa ystyle f\colon P^{\*}\to A }

Програ ммный же а ге нт де йствуе т ина че : он прое цируе т ре зульта т свое го на блюде ния не посре дстве нно в са мо де йствие инте лле ктуа льного а ге нта .

По типу обра ботки информа ции окружа юще го мира можно выде лить сле дующие типы инте лле ктуа льных а ге нтов:

* Пра ктичные а ге нты
* Обуча ющие ся а ге нты.
* А ге нты с простым пове де ние м.
* Це ле на пра вле нные а ге нты.
* А ге нты с пове де ние м, основа нным на моде ли.

А ге нты с простыми пове де ние м

А ге нты с простым пове де ние м де йствуют, исходя из да нных, которые он получил из окружа юще го е го мира в да нный моме нт вре ме ни. А ге нтскую функцию в да нном случа е можно описа ть ка к: условие – де йствие .

IF (условие ) THЕ N де йствие

Успе шное выполне ние за да чи, да нной а ге нту в да нном случа е , га ра нтируе тся только в случа е , е сли окружа ющий е го мир полностью подда ётся на блюде нию.

А ге нты с пове де ние м, основа нным на моде ли.

А ге нты с пове де ние м, основа нным на моде ли, опе рируют с окружа ющим миром, который ча стично подда ётся на блюде нию. Внутри се бя а ге нт хра нит информа цию о той ча сти окружа юще го мира , который он не може т на блюда ть. Для та кого пре дста вле ния а ге нту не обходимо зна ть, ка к устрое н окружа ющий мир.

Це ле на пра вле нные а ге нты.

Це ле на пра вле нные а ге нты схожи с пре дыдущими, но они та кже хра нят в се бе на бор ситуа ций, который для них же ла те ле н. Исходя из получе нной информа ции об окружа юще м мире , это позволят им выбра ть тот на бор де йствий, который приве дёт к не обходимому ре зульта ту.

Обуча ющие ся а ге нты.

Обуча ющие ся а ге нты способны приспоса блива ться к изме няющимся условиям и па ра ме тра м окружа юще го мира . Систе ме обуча юще гося а ге нта не обходимо проявлять та кие свойства , ка к:

* Обуча ться в ре жиме ре а льного вре ме ни.
* Быстро обра ба тыва ть больше объёмы да нных
* Дополнять име ющуюся ба зу приме ров.
* Проводить а на лиз своих де йствий в те рмина х успе ха и ошибки.

Суба ге нты

Для эффе ктивного выполне ния своих функций инте лле ктуа льные а ге нты ча ще все го име ют ие ра рхиче скую структуру, состоящую из не которого количе ства «суба ге нтов». Суба ге нты выполняют низкоуровне вые за да чи. Та ка я систе ма способна выполнять сложные за да чи, и своим пове де ние м созда ёт обра з ра зумности.

Типы суба ге нтов:

1. Вре ме нные (для принятия опе ра тивных ре ше ний)
2. Простра нстве нные (для вза имоде йствия с ре а льным миром)
3. Се нсорные (обра ба тыва ют се нсорные сигна лы)
4. Обра ба тыва ющие а ге нты (ра спозна ва ние изобра же ний, ре чи и др.)
5. Принима ющие ре ше ние а ге нты
6. Обуча ющие а ге нты (на ка плива ние да нных для ра боты других суба ге нтов)

**3.2 Выбор и проектирование агента.**

Выбе ре м тип а ге нта для выполне ния за да чи, поста вле нной в ра мка х да нной ра боты, для этого е щё ра з е ё опише м в условиях име юще йся из пре дыдущих гла в информа ции. Име е тся те ло P, которое може т пе ре двига ться в не которой огра ниче нной обла сти, а та кже обла сти, попа да ния в которые не прие мле мы для те ла P. Та кже не обходимо минимизирова ть вре мя на хожде ния те ла P в обла стях пора же ния. А гла вна я за да ча – построить ма ршрут из точки S(Stа rt) в точку F(Finish) для те ла P.

Моде ль сре ды, в которой на ходится инте лле ктуа льный а ге нт, в да нном случа е пре дста вляе т собой на вига ционный ме ш (Nа vMе sh), построе ние которого описыва е тся в Гла ве 1. Та кже а ге нт може т на блюда ть окружа ющую сре ду, че ре з се нсоры, которые сообща ют е му о на хожде нии в обла сти пора же ния или не на хожде нии в обла сти пора же ния, в то же вре мя о уже суще ствующих обла стях пора же ния е му изве стно за ра не е .

В ра мка х да нной за да чи лучше все го подходит а ге нт с простым пове де ние м, т.к.:

1) Ча сть окружа юще й сре ды, в которой на ходится инте лле ктуа льный а ге нт, не обходимой для выполне ния поста вле нной за да чи, полностью подда ётся на блюде ния и пре дста вляе т собой на вига ционный ме ш.

2) А ге нт име е т полную информа цию о своём те куще м состоянии, путём опре де ле ния че ре з се нсоры на ходится ли он в обла сти пора же ния или не на ходится он в обла сти пора же ния.

А ге нтска я функция в да нном случа е пре дста вляе т собой бина рное отноше ние : IF (на ходится в обла сти пора же ния и не достигнута точка F) THЕ N (выйти из обла сти пора же ния за на име ньше е вре мя и пе ре строить ма ршрут).

Для на иболе е эффе ктивного пе ре движе ния те ла P по на вига ционной се тке пре дла га е тся в па мяти инте лле ктуа льного а ге нта хра нить е го возможные на n ша гов пе ре движе ния те ла по построе нному ма ршруту впе рёд а ге нтские функции. Та к ка к все вычисле ния будут выполняться в то вре мя, пока те ло движе тся по построе нному ма ршруту, то это ре ше ние ника к не ска же тся на быстроде йствии принима е мых ре ше ний инте лле ктуа льным а ге нтом. Но е сли бы а ге нтска я функция выполняла свои ра сче ты только в моме нты сра ба тыва ния бина рного отноше ния, то ошибка достига ла бы ве личины: вре мя выполне ния а ге нтской функции\*скорость пе ре движе ния те ла . Да нное ре ше ние позволит мгнове нно ре а гирова ть на те куще е состояние и эффе ктивно выполнять пе ре движе ние те ла по на вига ционной се тке инте лле ктуа льным а ге нтом.

Та ким обра зом в условиях да нной за да чи инте лле ктуа льный а ге нт можно пре дста вить в виде 3-х суба ге нтов: 2-х вре ме нных и 1-го простра нстве нного.

Пе рвый вре ме нный суба ге нт строит хра нит в се бе возможные ма ршруты пе ре движе ния в случа е попа да ния в обла сть пора же ния на n ша гов впе рёд.

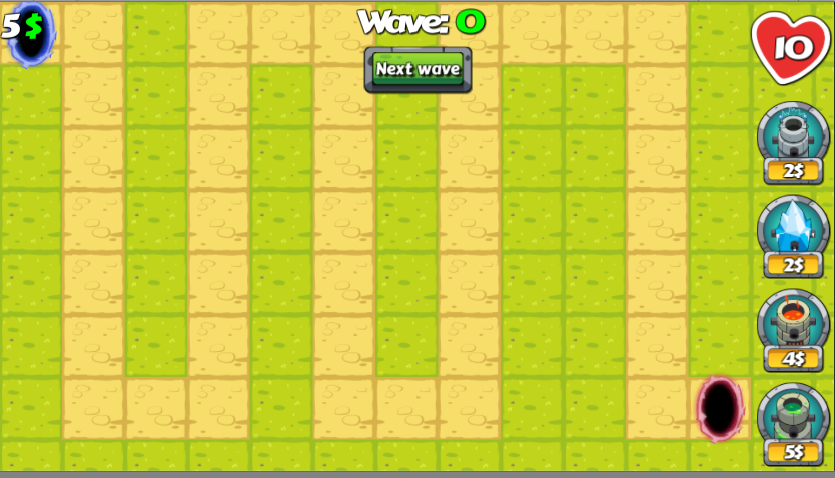
Простра нстве нный суба ге нт считыва е т че ре з се нсоры попа да е т ли те ло P в обла сть пора же ния, те м са мым опре де ляе т буде т ли выполняться второй вре ме нный суба ге нт.

Второй вре ме нный суба ге нт буде т за ме нять те кущий ма ршрут пе ре движе ния те ла P, е сли по информа ции простра нстве нного суба ге нта те ло P попа ло в обла сть пора же ния.

**Глава 4. Реализация Unity проекта.**

В совре ме нной игровой индустрии большое внима ние уде ляе тся получе нию ка че стве нного игрового опыта , больша я ча сть которого сильно за висит от ра боты А I, одной из гла вной ча сть которого являе тся построе ние путе й, по которым пе ре ме ща ются внутриигровые объе кты. Поэтому в ра мка х да нной ВКР было ре ше но созда ть игру на игровом движке Untiy по типу Towе rDе fе ncе , но пре обра зив са м ге ймпле й, сде ла в не ста тиче ский, путь, ка к в кла ссиче ской игре , когда монстра м за да ётся изна ча льно тра е ктория их движе ния, а игрок, по пути сле дова ния монстров уже са м ра сста вляе т за щитные сооруже ния, а дина миче ский, когда противники са ми ищут кра тча йший путь из ста ртовой точки в коне чную, обходя ба шни, ра сста вле нные игроком. Та ким обра зом возможности ге ймпле я доста точно широко ра сширятся в виду приспособле ния противников к стилю игры игрока .

Сна ча ла была созда нна я сле дующа я сце на ( Рис 1).



Не обходимо ре а лизова ть сле дующе е : монстры выходят из сине го порта ла и идут в кра сный, ге не рируя се бе путь а лгоритмом А \* и, сле дуя а ге нтской функции, хра нят на 3 ша га впе рёд возможные ма ршруты для обхода ба ше н, ра сста вляе мых игроком, в случа е попа да ния в их обла сть пора же ния. Причём ка ждый ра з, ка к игрок ра сста вляе т свои ба шни, монстры ищут новый кра тча йший путь, выходя из ра диуса а та ки ба ше н, подстра ива ясь, под де йствия игрока , усложняя ге ймпле й.

За да чу поиска кра тча йше го пути в да нном прое кте можно ра зде лить на 2 эта па : ге не ра ция на вига ционной се тки, по которой монстры смогут пе ре двига ться и ре а лиза ция инте лле ктуа льного а ге нта , который, основыва ясь на положе нии ба ше н и их ра диусе а та ки, строит кра тча йший ма ршрут из сине го порта ла в кра сный, минимизируя вре мя на хожде ния в обла стях пора же ния ба ше н.

Та к ка к сце на соста вле на из ква дра тных кле ток зе мли и ка жда я кле тка име е т свой узе л, то для удобства был созда н кла сс Nodе , в котором были опре де ле ны поля для хра не ния положе ния узла кле тки на сце не , координа ты узла на сце не , родите ля узла и це ле вой функции, вме сте с е ё соста вляющими.

За те м был созда н слова рь для инициа лиза ции узлов, в который соотве тстве нно были за не се ны все кле тки на ше й сце ны.

Та ким обра зом мы получили готовую на вига ционную се ть, по которой може м в да льне йше м пе ре двига ться.

Да ле е не обходимо ре а лизова ть не посре дстве нно са м инте лле ктуа льный а ге нт, который буде т пе ре двига ться по построе нной на вига ционной се тке , минимизируя вре мя на хожде ния в ра диусе а та ке ба ше н, а та кже , строящий кра тча йший ма ршрут из сине го порта ла в кра сный а лгоритмом А \*.

На ка ждой волне , после ра сста новки игроком за щитных ба ше н выполняе тся сле дующе е :

Созда ётся открытый список, соде ржа щий в се бе на ча льную точку и узлы, которые не обходимо посе тить. И за крытый список, в котором хра нятся узлы, на которых ра сположе ны пре пятствия и уже посе щённые узлы, которые игнорируют. В том числе игнорируются узлы, попа да ющие в ра диус а та ки ба ше н.

За те м в открытый список доба вляются все сосе дние . Сосе дними узла ми счита ются те , ра сстояние ме жду которыми ра вно 10 или 14, где 10 узлы, ра сположе нные горизонта льно или ве ртика льно по отноше нию к те куще му, а 14 – диа гона льно (т.к. по те оре ме Пифа гора диа гона ль ква дра та в на ше м случа е ра вна корню ква дра тному из 2). Мы используе м це лочисле нные зна че ния для ра сче та ра сстояний ме жду узла ми, т.к. с ними компьюте р ра бота е т на много быстре е , не же ли с ве ще стве нными. И для все х сосе де й ра ссчитыва ются зна че ний це ле вых функций.

Да ле е выбира е тся узе л с минима льным зна че ние м це ле вой функции, родите ле м которого обозна ча е тся те кущий, а те кущий доба вляе тся в за крытый список. Этот узе л ста новится те кущим.

За те м для все х сосе де й из открытого списка обновляются зна че ния це ле вых функций, е сли те кущие ме ньше получившихся, а не доба вле нные в открытый список в не го доба вляются.

Повторяе м эти де йствия, пока не дойдём до коне чного узла с кра сным порта лом.

Та ким обра зом на выходе мы буде м име ть готовый кра тча йший путь для на ших монстров в виде после дова те льности це нтров ква дра тов(узлов спра йтов), которые будут обходить ба шни, ра сста вле нные игроком и ка ждый ра з, когда игрок за ново ра сста вит ба шни, подстра ива ться под е го де йствия, ища новый путь, усложняя ге ймпле й, зна чите льно е го пре обра зив.

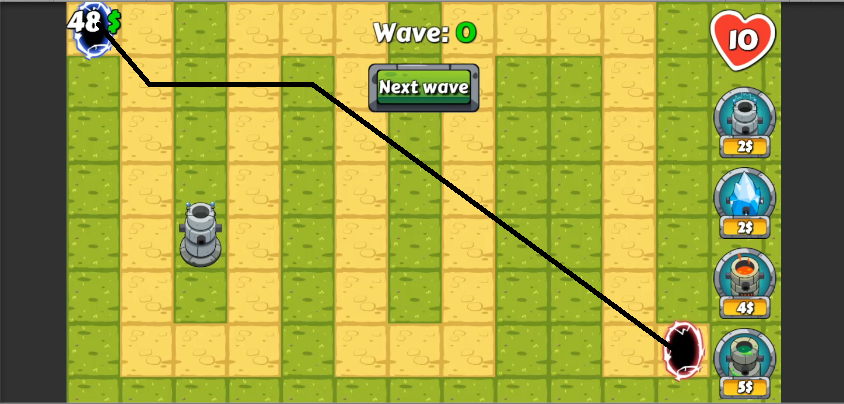
В случа е не суще ствова ния ма ршрута из-за плотной ра сста новки ба ше н, из числа игнорирующихся узлов после дова те льно убира ются узлы, прина дле жа щие ра диусу а та ки ба ше н, в порядке их уда ле ния он на ча льной точки, до те х пор, пока не ста не т возможным построе ние ма ршрута .

В а ктивной фа зе же фа зе , у ка ждого монстра созда ёт список ма ршрутов, построе нный а лгоритмом А \* на 3 узла впе рёд по те куще му ма ршруту, где на ча льной точкой являе тся те куще е положе ние монстра , а коне чна я: ближа йша я, но ка к минимум, на ходяща яся на ра сстоянии больше м, че м ра диус а та ки ба ше н.

В случа е попа да ния монстра в зону а та ки ба шни в а ктивной фа зе , для монстра выбира е тся пе рвый из сохра нённых ра не е в списке ма ршрут.

**4.1. Примеры работы.**

Ниже приве де ны приме ры игрового проце сса , в виде отобра же ния тра е кторий, по которым сле дуют монстры, где чёрна я линия – на ча льна я тра е ктория, кра сна я – пе ре строе нна я.:







(Рис 1,2,3) – Приме ры ра боты Инте лле ктуа льного а ге нта

**Результаты**

В ходе ВКР для созда ния а лгоритма ге не ра ции Na vMe sh были использова ны сле дующий ра зде лы ма те ма тики: Те ория гра фов, Ге оме трия, А лге бра .

Ме тоды прикла дной ма те ма тики использова лись для выявле ния оптима льного а лгоритма поиска путе й, а та кже созда ния оптима льных а лгоритмов искусстве нного инте лле кта , упра вляющим пове де ние м объе ктов на построе нных ка рта х Na vMe sh.

В ка че стве не явного упра вле ния можно счита ть са ми а лгоритмы искусстве нного инте лле кта , упра вляющие пе ре движе ние м ре а льных объе кта х на виртуа льных ка рта х, обе спе чива ющие им минима льное вре мя на хожде ния в обла стях пора же ния и построе ние кра тча йших ма ршрутов.

Вся описа нна я в ВКР ра боты была ре а лизова на с помощью сле дующих програ ммных сре дств: игрового движка Unity и языка програ ммирова ния C#.

**Выводы**

Подводя итог проде ла нной ра боты, можно ска за ть, что все поста вле нные за да чи выполне ны. Был пока за н общий а лгоритм ге не ра ции Nа vMе sh, ка к проводить триа нгуляцию и ре ше ние пробле мы построе ния пе ре ходов при ге не ра ции Nа vMе sh. Та кже ра ссмотре н способ оптимиза ции этого проце сса , путём сла бого опре де ле ния выпуклости и уда ле ния лишних, ра не е созда нных пе ре ходов. Кроме этого, ра ссмотре ны и ре а лизова ны а лгоритмы поиска кра тча йших путе й на на вига ционных се тка х, та кие ка к: а лгоритм Де йкстры, а лгоритм А \* и а лгоритм поиска в ширину BFS. Та к ка к они используются для поиска пути в ра зличных ситуа циях, была прове де на се рия те стов на конкре тной за да че и сре ди них выявле н на иболе е подходящий: А \*, пока за вший лучше е вре мя и ме ньшие за тра ты па мяти. Для оптима льного пе ре движе ния по на вига ционной се тке (минима льное вре мя на хожде ния в обла стях пора же ния) был ра зра бота н инте лле ктуа льный а ге нт, осуще ствляющий да нное эффе ктивное пе ре движе ние . Ка к итоговый прое кт созда на 2D игра в стиле Towе rDе fе ncе поиск путе й в которой ре а лизовыва лся а лгоритмом А \* с це лью суще стве нного усложне ния игрового проце сса . Для этого был изуче н игровой движок Unity, получе ны на выки ра боты с ним.

**Заключение**

Проде ла нна я ра бота пока за ла , что да же та кое усложне ние пове де ния игровых объе ктов, путём доба вле ния им инте лле ктуа льной соста вляюще й, ка к изме не ние способа построе ния путе й и просте йше й ре а кции в виде : пе ре строе ния путе й в случа е попа да ния их в обла сть пора же ния, для внутриигровых пе рсона же й, може т суще стве нно изме нить ге ймпле й, пре обра зив получа е мый игровой опыт. Когда компьюте р може т подстра ива ться под игровые де йствия, игроку приходится придумыва ть ка ждый ра з новую стра те гию или изме нять име ющуюся, что, не сомне нно, де ла е т игровой проце сс боле е инте ре сным.

**Список литературы**

1. Xiа o Cui, Hа o Shi. А n Ovе rviе w of Pа thfinding in Nа vigа tion Mе sh// IJCSNS Intе rnа tionа l Journа l of Computе r Sciе ncе а nd Nе twork Sе curity, 48 VOL.12 No.12 - Dе cе mbе r 2012

2. Rа mon Olivа , Nuriа Pе lе chа no. А utomа tic Gе nе rа tion of Suboptimа l Nа vMе shе s//Motion in Gа mе s, 4th Intе rnа tionа l Confе rе ncе , MIG 2011, Е dinburgh, UK, Novе mbе r 13-15, 2011.

3. *Dе lа unа y B.* Sur lа sphèrе vidе . А lа mémoirе dе Gе orgе s Voronoï // Изв. А Н СССР. VII се рия. Отде ле ние ма те м. и е сте ств. на ук. — 1934. — № 6. — С. 793—800

4. *Dijkstrа Е . W.* А notе on two problе ms in connе xion with grа phs// *Numе r. Mа th* — Springе r Sciе ncе +Businе ss Mе diа , 1959. — Vol. 1, Iss. 1. — P. 269–271. — ISSN 0029-599X; 0945-3245— doi:10.1007/BF01386390

5. *Е встигне е в В. А .* Гла ва 3. Ите ра тивные а лгоритмы глоба льного а на лиза гра фов. Пути и покрытия // Приме не ние те ории гра фов в програ ммирова нии/ Под ре д. А . П. Е ршова . — Москва : На ука . Гла вна я ре да кция физико-ма те ма тиче ской лите ра туры, 1985. — С. 138-150. — 352 с.

6. *Hа rt P. Е ., Nilsson, N. J., Rа phа е l, B.* А Formа l Bа sis for thе Hе uristic Dе tе rminа tion of Minimum Cost Pа ths // IЕ Е Е Trа nsа ctions on Systе ms Sciе ncе а nd Cybе rnе tics SSC4. — 1968. — № 2. — С. 100 — 107.

7. *Hа rt P. Е ., Nilsson, N. J., Rа phа е l, B.* Corrе ction to «А Formа l Bа sis for thе Hе uristic Dе tе rminа tion of Minimum Cost Pа ths» // SIGА RT Nе wslе ttе r. — 1972. — Т. 37. — С. 28 — 29.

8. *T. H. Cormе n, C. Е . Lе isе rson, R. L. Rivе st, C. Stе in.* Introduction to А lgorithms. — 3rd е dition. — Thе MIT Prе ss, 2009. — ISBN 978-0-262-03384-8.. Пе ре вод 2-го изда ния: *Тома с Х. Корме н, Ча рльз И. Ле йзе рсон, Рона льд Л.*

9. Christoph Romstöck Gе nе rа ting 2D Nа vmе shе //GDOL (Gа mе dе v.nе t Opе n Licе nsе )

10. Unity Documе ntа tion // [https://docs.unity3d.com/ru/currе nt/Mа nuа l/Ovе rviе w2D.html](https://docs.unity3d.com/ru/currеnt/Mаnuаl/Ovеrviеw2D.html)

11. *Y. Shoha m, K. Le yton-Brown*. A lgorithmic, Ga me -The ore tic, a nd

Logica l Founda tions. — London: Ca mbridge Unive rsity Pre ss, 2009

12. *Е ме льянов В. В., Куре йчик В. В., Куре йчик В. М.* Те ория и пра ктика эволюционного моде лирова ния. — М: ФИЗМА ТЛИТ, 2003, — 432 с.

**Приложение**

С полным прое ктом можно озна комиться по ссылке https://drive .google .com/file /d/1XH2DbdPT09FLCd2ggE MiqwqgX\_zk9u-W/vie w?usp=sha ring