АНАЛИЗ АЛГОРИТМОВ ПОИСКА ПУТИ В НАВИГАЦИОННЫХ СЕТКАХ

Хирьянов И. Д.
Факультет инфомационных технологий,
Полоцкий государственный университет имени Ефросинии Полоцкой
Полоцк, Республика Беларусь
E-mail: i.khiryanau@psu.by

Проанализированы алгоритмы поиска пути в навигационных сетках, взяты одни из самых распространенных: поиск в ширину, Дейкстры и A. Для тестирования использован сервис PathFinding и выделены ключевые критерии: длина пути, время и количество итераций. В ходе иследования выявлено что A* находит кратчайший путь как за меньшее время, так и совершает минимальное количество операций по сравнению с другими рассмотренными алгоритмами.*

Введение

Выбор оптимального пути для каждого объекта является одной из самой важных задач искусственного интеллекта в игровой индустрии. Он описывается как процесс определения набора перемещений объекта из одной точки в другую, без столкновения с любыми препятствиями. Оптимальный путь должен обладать тремя свойствами: длинной пути, временем и количеством итераций для поиска.

Первое свойство определяет расстояние, требуемое для прохождения пути. При использовании этого показателя, оптимальный маршрут является кратчайшим. Это означает, что дистанция от начала движения до финиша не длиннее любого другого пути.

Второе свойство определяет оптимальный путь, как самый быстрый. Это означает, что время, затраченное на прохождение маршрута, всегда меньше, чем при прохождении любого другого.

Третье свойство определяет сколько ресурсов вычислительной системы потребуется для нахождения оптимального маршрута. Соответственно наименьшее количество итераций является лучшим показателем среди сравнимых алгоритмов.

Требуется проанализировать самые распространенные алгоритмы для нахождения пути и выбрать самый эффективный из них.

I. Алгоритм поиска в ширину

Одним из самых первых алгоритмов поиска кратчайшего пути на графах является алгоритм поиска в ширину [1-2].

Алгоритм:

1. Выбираем начальную вершину, задаём ей нулевой уровень глубины.
2. Всех её соседей добавляем в очередь для посещения, а саму помечаем как посещённую и больше к ней не обращаемся.
3. Берём элементы из очереди для посещения, считая их уровень глубины, как: уровень глубины родителя + 1. Добавляем эти соседние вершины в очередь для посещения.
4. Текущую вершину, помечаем как посещённую и удаляем из очереди для посещения.
5. Повторяем шаги 3 и 4, пока не обойдём весь граф.

II. Алгоритм Дейкстры

Алгоритм Дейкстры работает только для графов без рёбер отрицательного веса [3-4].

Алгоритм:

1. Задаём набор расстояний, в котором будут храниться кратчайшие расстояния от исходной вершины до всех остальных вершин графа, а также набор посещённых вершин.
2. Начальной вершине задаём значение равное нулю, а всем остальным бесконечности.
3. Пока в набор посещенных вершин не добавлены все вершины делаем следующее: выберем вершину, не включённую в наш набор посещённых вершин, значение которой минимально; добавим её в наш набор посещённых вершин; обновим значения всех смежных вершин, как: минимальное значение суммы текущей вершины и веса ребра.

III. Алгоритм A*

Алгоритм A* является усовершенствованием алгоритма Дейкстры за счёт введения эвристики. Переход осуществляется в ту вершину, предположительный путь от которой до конечной является кратчайшим. Чаще всего пользуются эвристикой Манхеттена, но помимо неё возможно использование эвристик Чебышева и Евклида [5].

Алгоритм:

1. Добавляем начальный узел в открытый список для хранения узлов
2. Повторяем следующее: Останавливаемся, если добавили конечный узел в открытый список, в этом случае путь найден. Или открытый список пуст, и мы не дошли до конечного узла. Здесь получаем, что не существует пути до конечного узла. Ищем в открытом списке узел с наименьшим значением целевой функции F. Делаем его текущим узлом.

Помещаем этот узел в закрытый список. (И удаляем с открытого) Для каждого из соседних узлов делаем следующее: если узел находится в закрытом списке, игнорируем ее. В противном случае если узел еще не в открытом списке, то добавляем его туда. Делаем текущий узел родительским для это клетки. Рассчитываем стоимости F, G и H узла. Если узел уже в открытом списке, то проверяем, не дешевле ли будет путь через этот узел. Для сравнения используем стоимость G. Более низкая стоимость G указывает на то, что путь будет дешевле. Если это так, то меняем родителя узла на текущий узел и пересчитываем для него стоимости G и F.

3. Сохраняем путь. Двигаясь назад от целевого узла, проходя от каждого узла к его родителю до тех пор, пока не дойдем до начального узла. Это и будет наш путь.

IV. Сравнение алгоритмов поиска

Для проведения тестов и выявления лучшего из 3-х представленных алгоритмов была проведена работа с сервисом `https://qiao.github.io/PathFinding.js/visual/`, в котором реализованы в единой оболочке большинство алгоритмов поиска путей. В качестве критериев оценивания были выбраны: скорость нахождения кратчайшего пути, количество итераций, выполненных в процессе поиска, и длина найденного пути. Т.к. алгоритмы реализованы на одной и той же платформе (java script), то в расчёт не берётся скорость компилятора.

На рисунке 1 продемонстрирован первый тест. В этом и последующих зеленый квадрат - начало пути, красный - конец, черный - непроходимое припятствие, а все остальное это поле по которому можно передвигаться. Результаты тестирования описаны в таблице 1.

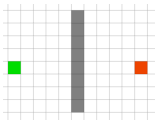

Рис. 1 – Первый тест

Таблица 1 – Результаты первого теста

Алгоритм	Длина пути	Время(ms)	Кол-во операций
A*	13,31	1,1	136
Дейкстра	13,31	1,8	1059
BFS	13,31	1,4	843

Исходя из первого тестирования видно, что оптимальный маршрут по длине находят все. Дейкстра тратит на это больше всего времени и количество операций. Алгоритм A* показывает наилучший результат по всем критериям.

На рисунке 2 показан второй тест, а его результаты - в таблице 2.

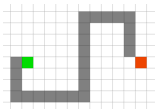

Рис. 2 – Второй тест

Таблица 2 – Результаты второго теста

Алгоритм	Длина пути	Время(ms)	Кол-во операций
A*	16,49	0,2	133
Дейкстра	16,49	2,0	1267
BFS	16,49	1,0	1420

Второе тестирование показало, что A* все еще превосходит другие алгоритмы. Дейкстра в этот раз показал себя лучше BFS по количеству операций, хотя все еще уступает во времени нахождения.

На рисунке 3 продемонстрирован третий тест. В таблице 3 можно ознакомиться с его результатами.

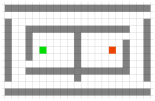

Рис. 3 – Третий тест

Таблица 3 – Результаты третьего теста

Алгоритм	Длина пути	Время(ms)	Кол. операций
A*	29,31	0,3	320
Дейкстра	29,31	9,3	2458
BFS	29,31	9,0	2542

Третье тестирование показало те же результаты, что и второе. Алгоритм A* остается лучшим.

Из таблицы проведённых тестов видно, что A* находит кратчайший путь как за меньшее время, так и совершает минимальное количество операций по сравнению с другими рассмотренными алгоритмами.

Список литературы

1. Л.А. Павлов, Н.В. Первова Глава 6 Структуры и алгоритмы обработки данных // Алгоритмы на графах / Издательство «Лань», 2020г. – 199 – 218 с.
2. Томас Кормен, Чарльз Лейзерсон, Рональд Ривест, Клиффорд Штайн Глава 22.2 Поиск в ширину // Алгоритмы. Построение и анализ второе издание/ Издательский дом "Вильямс 2011г. - 613-622 с.
3. Ананий В. Левитин Глава 9. Жадные методы: Алгоритм Дейкстры // Алгоритмы: введение в разработку и анализ = Introduction to The Design and Analysis of Aigorithms. - Издательский дом "Вильямс 2006г. - 189-195 с.
4. Адитья Бхаргаева Глава 7 Алгоритм Дейкстры // Грокаем алгоритмы. Иллюстрированное пособие для программистов и любопытствующих / Издательство "Питер 2017г. - 151-181с
5. С.Рассел, П. Норвиг Глава 3 Решение задач по средствам поиска // Искуственный интелект. Современный подход. Четвертое издание. Том I. Решение проблем: знания и рассуждения / Издательство "Диалектика 2021г. - 125 -204 с.