Лабораторная работа №3: Алгоритм A* («A star»)

Если вы когда-нибудь играли в какую-либо игру на компьютере на основе карт, то вы, вероятно, сталкивались с органами компьютерного управления, которые умеют самостоятельно рассчитывать путь из пункта А в пункт Б. На самом деле это обычная распространенная проблема как в играх, так и в других видах программного обеспечения - поиск пути от начального местоположения до пункта назначения с успешным преодолением препятствий.

Один очень широко используемый алгоритм для такого рода проблемы называют A* (произносится "A-star"). Он является наиболее эффективным алгоритмом для поиска пути в компьютерной программе. Концепция алгоритма довольно проста, начиная с исходного местоположения, алгоритм постепенно строит путь от исходной точки до места назначения, используя наикратчайший путь, чтобы сделать следующий шаг. Это гарантирует, что полный путь будет также оптимальным.

Вам не придется реализовывать алгоритм A*; это было уже сделано за вас. На самом деле существует даже пользовательский интерфейс для того, чтобы экспериментировать с этим алгоритмом:

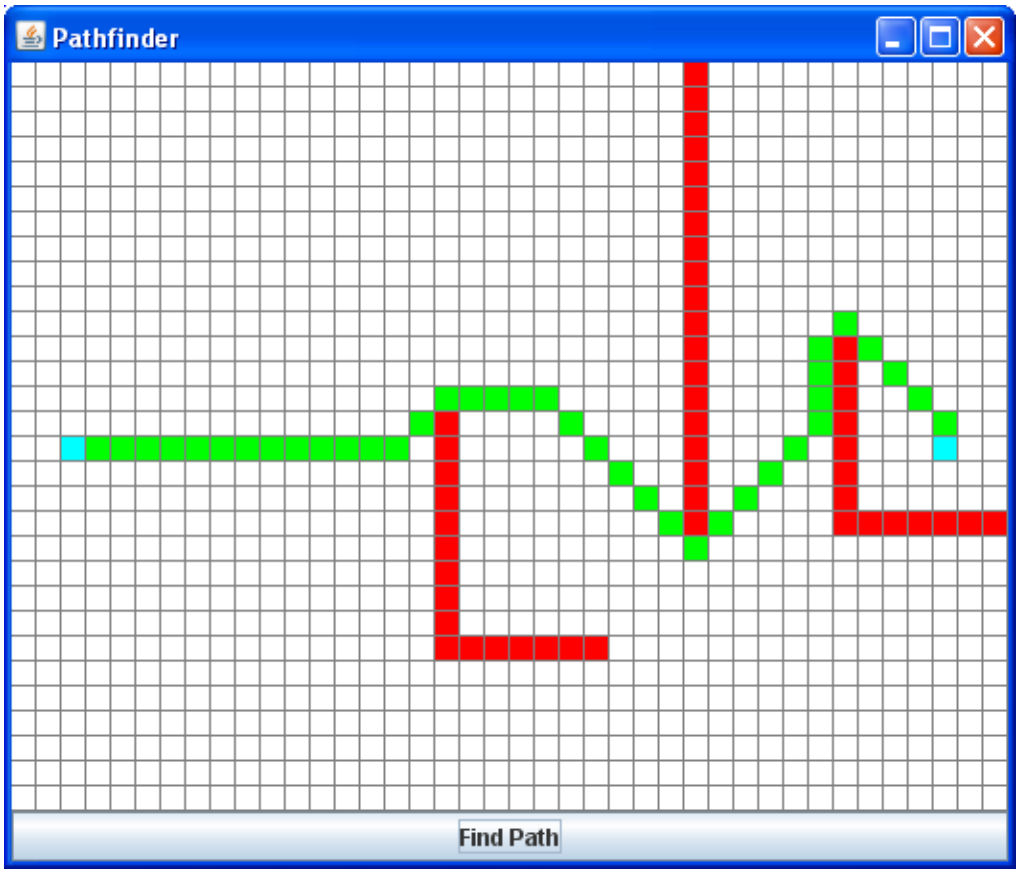

Рисунок 5.1. Пользовательский интерфейс для работы с алгоритмом A*.

Вы можете щелкнуть по различным квадратам, чтобы сделать из них в барьеры (красные) или проходимые клетки (белые). Синие клетки обозначают начало и конец пути. Нажав на кнопку "Find Path", программа вычислит путь, используя алгоритм A*, и затем отобразит его зеленым цветом. В случае если путь не будет найден, то программа просто не отобразит путь.

Алгоритм A* содержит много информации для отслеживания, и классы Java идеально подходят для такого рода задач. Существует два основных вида информации для организации управления алгоритмом A*:

- **Локации** – это наборы координат конкретных клеток на двумерной карте. Алгоритм A* должен иметь возможность ссылаться на определенные места на карте.
- **Вершины** - это отдельные шаги на пути, которые генерирует алгоритм A*. Например, продемонстрированные выше зеленые ячейки - это последовательность вершин на карте.

Каждая путевая точка владеет следующей информацией:

- Расположение ячейки для вершины.
- Ссылка на предыдущую вершину маршрута. Конечный путь - это последовательность вершин от пункта назначения до исходной точки.
- Фактическая стоимость пути от начального местоположения до текущей вершины по определенному пути.
- Эвристическая оценка (приблизительная оценка) остаточной стоимости пути от текущей вершины до конечной цели.

Так как алгоритм A* строит свой путь, он должен иметь два основных набора вершин:

- Первый набор хранит "открытые вершины" или вершины, которые все еще должны учитываться алгоритмом A*.
- Второй набор хранит "закрытые вершины" или вершины, которые уже были учтены алгоритмом A* и их не нужно будет больше рассматривать.

Каждая итерация алгоритма A* довольно проста: найти вершину с наименьшей стоимостью пути из набора открытых вершин, сделать шаг в каждом направлении от этой вершины для создания новых открытых вершин, а затем переместить вершины из открытого набора в закрытый. Это повторяется до тех пор, пока не будет достигнута конечная вершина! Если во время этого процесса заканчиваются открытые вершины, то пути от начальной вершины до конечной вершины нет.

Данная обработка в первую очередь зависит от расположения вершин, поэтому очень полезно сохранять путевые точки как отображение местоположений до соответствующих вершин. Таким образом, вы будете использовать хранилище java.util.HashMap для каждого из этих наборов с объектами Location в качестве ключей, и объектами Waypoint в качестве значений.

Прежде чем начать

Прежде чем начать, вам необходимо скачать исходные файлы для данной лабораторной работы:

- Map2D.java - представляет карту, по которой перемещается алгоритм A*, включая в себя информацию о проходимости ячеек
- Location.java - этот класс представляет координаты конкретной ячейки на карте
- Waypoint.java - представляет отдельные вершины в сгенерированном пути
- AStarPathfinder.java - этот класс реализует алгоритм поиска пути A* в виде статического метода.
- AStarState.java - этот класс хранит набор открытых и закрытых вершин, и предоставляет основные операции, необходимые для функционирования алгоритма поиска A*.
- AStarApp.java - простое Swing-приложение, которое обеспечивает редактируемый вид 2D-карты, и запускает поиск пути по запросу
- JMapCell.java - это Swing -компонент, который используется для отображения состояния ячеек на карте

Обратите внимание, что приложение будет успешно компилироваться в том виде, какое оно есть, но функция поиска пути не будет работать, пока вы не завершите задание. Единственные классы, которые вы должны изменить это Location и AStarState. Все остальное - это код платформы, которая позволяет редактировать карту и показывать путь, который генерирует алгоритм. (Не рекомендуется редактировать исходный код других файлов)

Локации

Для начала необходимо подготовить класс Location для совместного использования с классами коллекции Java. Поскольку вы будете использовать контейнеры для хеширования для выполнения данного задания, то для этого необходимо:

- Обеспечить реализацию метода equals ().
- Обеспечить реализацию метода hashcode().

Добавьте реализацию каждого из этих методов в класс Location, следуя шаблонам в классе. После этого вы можете использовать класс Location в качестве ключевого типа в контейнерах хеширования, таких как HashSet и HashMap.

Состояния A*

После того, как класс Location готов к использованию, вы можете завершить реализацию класса AStarState. Это класс, который поддерживает наборы открытых и закрытых вершин, поэтому он действительно обеспечивает основную функциональность для реализации алгоритма A*.

Как упоминалось ранее, состояние A* состоит из двух наборов вершин, один из открытых вершин и другой из закрытых. Чтобы упростить алгоритм, вершины будут храниться в хэш-карте, где местоположение вершин является ключом, а сами вершины являются значениями. Таким образом, у вас будет такой тип:

HashMap<Location, Waypoint>

(Очевидный вывод из всего этого заключается в том, что с каждым местоположением на карте может быть связана только одна вершина.)

Добавьте два (нестатических) поля в класс AStarState с таким типом, одно для "открытых вершин" и другой для "закрытых вершин". Кроме того, не забудьте инициализировать каждое из этих полей для ссылки на новую пустую коллекцию.

После создания и инициализации полей, вы должны реализовать следующие методы в классе AStarState:

1) public int numOpenWaypoints()

Этот метод возвращает количество точек в наборе открытых вершин.

2) public Waypoint getMinOpenWaypoint()

Эта функция должна проверить все вершины в наборе открытых вершин, и после этого она должна вернуть ссылку на вершину с наименьшей общей стоимостью. Если в "открытом" наборе нет вершин, функция возвращает NULL.

Не удаляйте вершину из набора после того, как вы вернули ее; просто верните ссылку на точку с наименьшей общей стоимостью.

3) public boolean addOpenWaypoint(Waypoint newWP)

Это самый сложный метод в классе состояний A*. Данный метод усложняет то, что он должен добавлять указанную вершину только в том случае, если существующая вершина хуже новой. Вот что должен делать этот метод:

- Если в наборе «открытых вершин» в настоящее время нет вершины для данного местоположения, то необходимо просто добавить новую вершину.
- Если в наборе «открытых вершин» уже есть вершина для этой локации, добавьте новую вершину только в том случае, если стоимость пути до новой вершины меньше стоимости пути до текущей. (Убедитесь, что используете не общую стоимость.) Другими словами, если путь через новую вершину короче, чем путь через текущую вершину, замените текущую вершину на новую

Как вы могли заметить, что в таком случае вам потребуется извлечь существующую вершину из «открытого набора», если таковая имеется. Данный шаг довольно прост - замените предыдущую точку на новую, используя метод HashMap.put(), который заменит старое значение на новое. Пусть данный метод вернет значение true, если новая вершина была успешно добавлена в набор, и false в противном случае.

4) public boolean isLocationClosed(Location loc)

Эта функция должна возвращать значение true, если указанное местоположение встречается в наборе закрытых вершин, и false в противном

случае. Так как закрытые вершины хранятся в хэш-карте с расположениями в качестве ключевых значений, данный метод достаточно просто в реализации.

5) public void closeWaypoint(Location loc)

Эта функция перемещает вершину из набора «открытых вершин» в набор «закрытых вершин». Так как вершины обозначены местоположением, метод принимает местоположение вершины.

Процесс должен быть простым:

- Удалите вершину, соответствующую указанному местоположению из набора «открытых вершин».
- Добавьте вершину, которую вы удалили, в набор закрытых вершин. Ключом должно являться местоположение точки.

Компиляция и тестирование

Как только вы реализуете вышеуказанную функциональность, запустите программу поиска пути, чтобы проверит правильность ее выполнения. Если вы реализовали все правильно, то у вас не должно возникнуть проблем при создании препятствий и последующим поиском путей вокруг них.

Скомпилируйте и запустите программу также, как и всегда:

javac *.java

java AStarApp