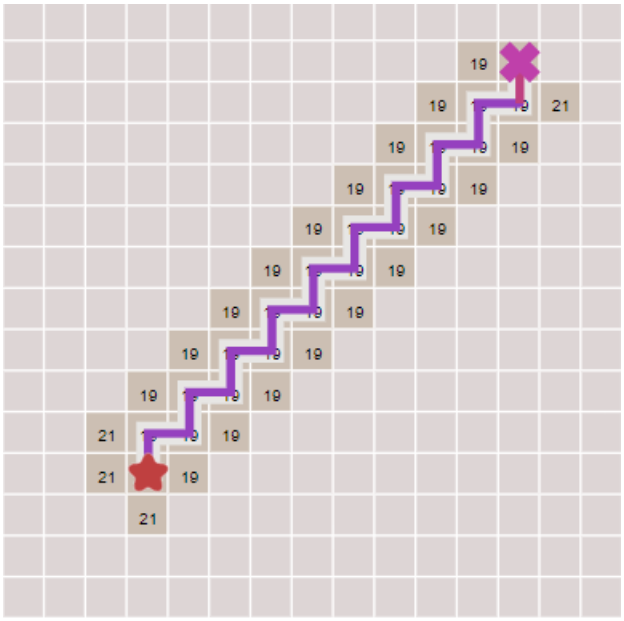
Отчёт по практике

Шибанин Георгий

6 августа 2019 г.

1 Задача поиска маршрута на карте

Задача состоит в том чтобы найти путь из одной точки карты в другую. Звёздочка - старт, а крест - финиш. Выглядит довольно просто, достаточно строить маршрут по направлению к цели и в конце концов маршрут будет готов.

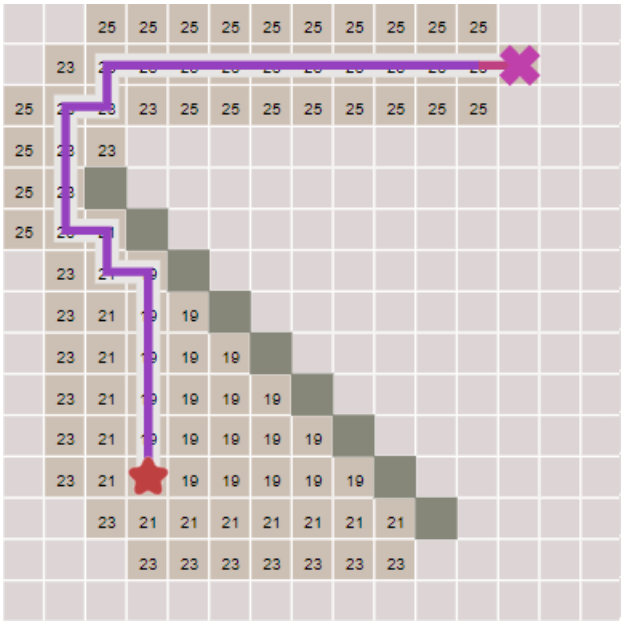


Но что, если на пути будут встречаться препятствия? Если их просто обойти, будет ли наш путь оптимальным? То есть, будет ли он самым быстрым(коротким), в отличие от других. Для того чтобы найти данный путь воспользуюсь алгоритмом A* поиска пути по графу.

Представление карты:

В своей реализации данного алгоритма я буду использовать карты в виде сеток(представление которых можно увидеть на картинках). То есть, на вход подаются размеры сетки, клетки которые недоступны для перемещения, и веса некоторых клеток, под весами подразумевается что движением по данным клеткам будет более затратным, чем по обычным.

Сам же граф будет представлять из себя вершины - центр клетки, и рёбра - соединяющие вершину с краями клетки. В моей реализации будет разрешено ходить в 8 направлениях, вверх, вниз, влево, вправо и по диагонали.



2 Поиска маршрута с наименьшей стоимостью с помощью алгоритма A*

Принцип работы данного алгоритма:

Сам алгоритм является совокупностью двух других алгоритмов, Дейкстры и поиска по первому наилучшему совпадению(жадный поиск). От Дейкстры он берёт способность находить кратчайший путь, а от поиска по первому наилучшему совпадению берёт эвристическую функцию, которая сообщает насколько мы близки к цели, тем самым путь строится по направлению к цели.

Рассмотрим алгоритм Дейкстры:

Алгоритм Дейкстры работает путем посещения вершин графа, начиная с начальной точки. Затем он рассматривает все ближайшие, еще не исследованные вершины, добавляя их к набору вершин, которые нужно исследовать. И так далее расширяет область посещённых точек, пока не достигнет конечной цели.

Поиск по первому наилучшему совпадению:

В отличие от алгоритма Дейкстры при выборе следующей рассматриваемой вершины, данный алгоритм берёт вершину ближайшую к конечной точке, тем самым расширяет свой путь исследования в направлении конечной вершины.

Главной составляющей алгоритма является функция $f(u) = g(u) + h(u)$, где $g(u)-$ стоимость пути от начальной точки до вершины u, $h(u)-$ эвристическая функция показывающая стоимость пути от вершины u до конечной цели, $f(u)-$ длина пути до цели.

Псевдокод A*:

```
a_star_search(graph, start, goal):
    g = {}
    g[start] = 0
```

```
parent = {}
parent[start] = start
open = PriorityQueue()
open.put(start, 0)
while (open is not empty):
    current = open.get()
    if current == goal:
        break
    for next in neigbors(current):
        new_cost = g[current] + graph.cost(current, next)
        if next not in g or new_cost < g[next]:
            g[next] = new_cost
            f = new_cost + h(goal, next)
            open.put(next, f)
            parent[next] = current
```

$g(u)-$ стоимость самого дешёвого пути от start до u

$h(u)-$ показывает эвристическое приближение стоимости пути от u до goal. В моём случает помимо четырёх стандартных направлений я использую движение по диагонали, следовательно для вычисления пути буду использовать метрику Чебышева: $h(u) = max(|u.x - goal.x|, |u.y - goal.y|)$.

parent− для каждой вершины показывает на то из какой вершины мы прибыли.

open− очередь с приоритетами в которой приоритет строится на основе функции f, данная очередь пополняется когда новый путь к точке лучше, чем наилучший предыдущий путь.

new_cost− переменная которая показывает стоимость нового предполагаемого пути. Является суммой пути до вершины *current* и $graph.cost(urrent, next)-$ стоимостью перехода от вершины *current* к следующей *next* вершине.

Сам алгоритм работает достаточно просто. Первым шагом положим в очередь *open* нашу стартовую вершину *start*. Далее, пока очередь не опустеет будем выполнять следующие шаги. Извлекаем первую по приоритету вершину *current* из *open*, если данная вершина является *goal*, то останавливаем работу алгоритма, иначе пройдемся по *next*− соседям вершины. Рассчитываем новую предполагаемую стоимость пути *new_cost* от *start* до *next*. И если нам ещё не известна стоимость пути до *next* вершины, или стоимость нового предполагаемого пути *new_cost* меньше, чем наилучший предыдущий путь, то обновим/добавим стоимость пути до *next* вершины, найдем значение функции f, положим в нашу очередь *open* новую вершину с приоритетом f, и установим для вершины *next*, то что мы прибыли в неё из вершины *current*.