УДК 621.398

ИССЛЕДОВАНИЕ БЫСТРОДЕЙСТВИЯ АЛГОРИТМОВ ПОИСКА КРАТЧАЙШЕГО ПУТИ В ЛАБИРИНТЕ

А. С. РОМАНЮК
Научный руководитель Н. Н. ГОРБАТЕНКО, канд. техн. наук, доц.
ГУ ВПО «Белорусско-Российский университет»

Существует большое количество алгоритмов, позволяющих найти маршрут между двумя точками. Самым простым и логичным вариантом является полный перебор всех возможных маршрутов из начальной точки в конечную. Первый найденный маршрут и будет являться решением. Такой способ требует использования достаточно больших ресурсов памяти (хранение и анализ всей карты), вследствие чего является малоприменимым в решении практических задач. К более продвинутым вариантам поиска пути можно отнести различные эвристические функции, просматривающие ограниченные участки карты, анализируя вес ячеек, пошагово улучшая текущий результат.

Большое применение технологии поиска пути получили в играх, трассировке печатных плат, создании навигационных карт. Оптимальный (быстрый и логичный в зависимости от необходимой длины) поиск пути возможен только на дискретном пространстве, т. е. на том пространстве, которое было поделено специально для поиска пути. Итог дискретизации пространства – граф.

В математике граф представляет собой структуру, составляющую набор объектов, в которых некоторые пары объекты определенным образом связаны. Они называются связанными вершинами (узлами или точками), каждая пара из которых представляет ребро.

Возможные типы дискретизации пространства:

– навигационная регулярная сетка;

– путевые точки;

– матрица проходимости.

Целью исследования являлось сравнение эффективности работы на реальных сетках с препятствиями широко известного алгоритма поиска в ширину и одного из расширений алгоритма A* под названием Jump Point. Основная идея исследования заключалась в том, чтобы попытаться ускорить поиск пути, пропуская многие точки, которые должны быть просмотрены другими алгоритмами. В данном исследовании дискретизация пространства производилась с помощью матрицы проходимости. Поле поиска представляет набор открытых и закрытых ячеек с максимальным количеством соседей равным восьми.

Для начала опишем, каким образом отсекать множество точек, непосредственно примыкающих к некоторой точке x. Цель заключается в нахождении таких соседей, т. е. *neighbours(x)*, до любых n точек, которых

нельзя достичь цель оптимально. Добиться этого можно путем сравнения двух путей: *p*, который начинается точкой *p(x)*, посещает *x* и заканчивается с n и другим путем *p'*, который так же начинается с *p(x)*, посещает *x* и заканчивается *n*, но не содержат *x*. Кроме того, каждая точка, содержащаяся в *p* или *p'*, должна относиться к *neighbours(x)*.

Существует два случая в зависимости от того, какой переход к *x* происходит из *p(x)*: прямой ход или диагональный. Стоит учесть, что если *x* является началом *p(x)*, то *p(x)* пусто и отсечение не происходит.

Схема алгоритма прямолинейного движения показана на рис. 1. Рекурсивно применяя правило отсечки, получаем *y* в качестве преемника прыжковой точки *x*. Эта точка интересна тем, что есть сосед *z*, в который можно попасть по оптимальному пути только через *y*. Промежуточные точки не генерируются и не рассматриваются.

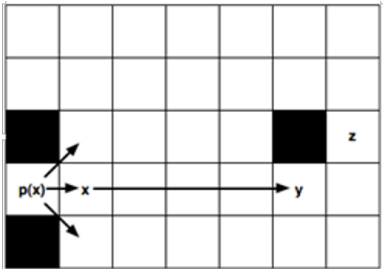

Рис. 1. Прямолинейное движение

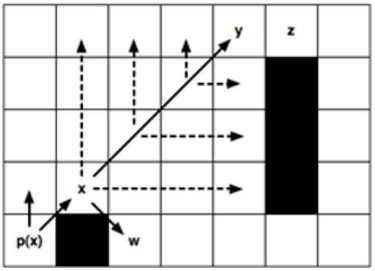

Рис. 2. Движение по диагонали

На рис. 2 приведена схема алгоритма движения по диагонали. При его работе рекурсивно принимаются диагональные правила отсечки. Перед каждым следующим диагональным шагом необходимо рекурсивно пройтись по прямым линиям. Только если обе «прямые» рекурсии не могут определить точку следующего прыжка, то перемещаемся дальше по диагонали. Точка *w* – вынужденный сосед *x*.

Для тестирования алгоритмов была использована карта с одинаковым набором препятствий. Визуальное представление процесса поиска показано на рис. 3 *а*, *б*, *в*.

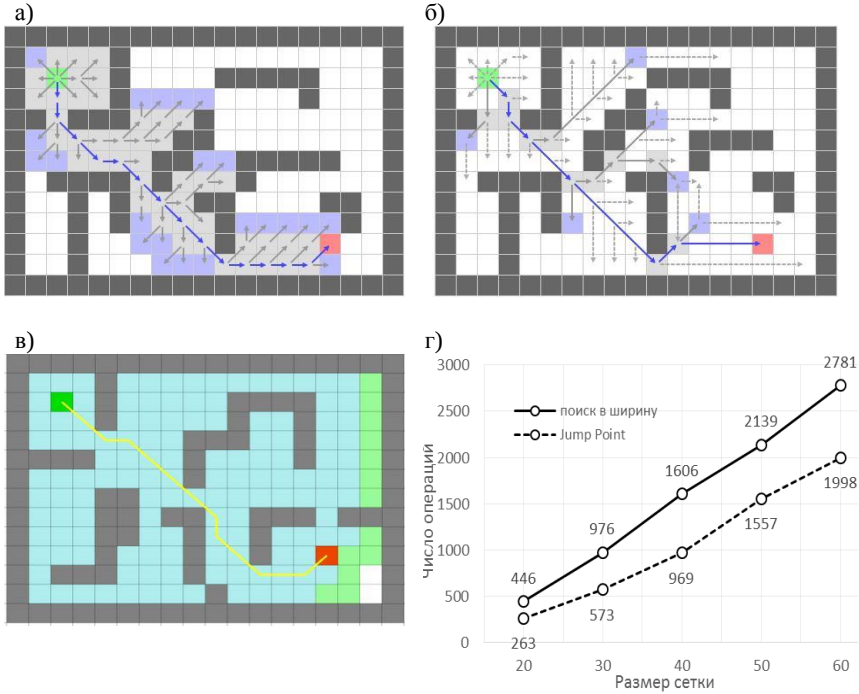

Рис. 3. Результаты работы программы: *а* – алгоритм A*; *б* – Jump Point; *в* – поиск в ширину; *г* – оценка быстродействия алгоритмов

На рис. 3, *г* изображена зависимость количества операций от размера сетки. Видно, что алгоритм Jump Point использует значительно меньшее количество ресурсов, пропуская целое множество точек, которые будут просмотрены поиском в ширину. Однако меняя условия, усложняя карту, добавляя новые препятствия, возможные пути, эти два алгоритма могут деградировать до примерно одинаковых результатов.

Таким образом, алгоритм решения задачи поиска кратчайшего маршрута следует выбирать исходя из конкретного набора входных данных, доступных ресурсов и ожидаемого результата. Так, к примеру, простой перебор всех маршрутов, в отличие от эвристических функций, может выдавать приемлемые результаты на небольших картах и при этом не требовать реализации сложных структур данных.