Journal of Computer and Communications, 2016, 4, 15-25
http://www.scirp.org/journal/jcc
ISSN Online: 2327-5227
ISSN Print: 2327-5219

Pathfinding in Strategy Games and Maze Solving Using A* Search Algorithm

Nawaf Hazim Barnouti, Sinan Sameer Mahmood Al-Dabbagh, Mustafa Abdul Sahib Naser

Al-Mansour University College, Baghdad, Iraq
Email: nawafhazim1987@gmail.com, nawafhazim1987@yahoo.com

How to cite this paper: Barnouti, N.H., Al-Dabbagh, S.S.M. and Naser, M.A.S. (2016) Pathfinding in Strategy Games and Maze Solving Using A* Search Algorithm. *Journal of Computer and Communications*, **4**, 15-25.
http://dx.doi.org/10.4236/jcc.2016.411002

Received: July 25, 2016
Accepted: September 4, 2016
Published: September 8, 2016

Abstract

Pathfinding algorithm addresses the problem of finding the shortest path from source to destination and avoiding obstacles. One of the greatest challenges in the design of realistic Artificial Intelligence (AI) in computer games is agent movement. Pathfinding strategies are usually employed as the core of any AI movement system. In this work, A* search algorithm is used to find the shortest path between the source and destination on image that represents a map or a maze. Finding a path through a maze is a basic computer science problem that can take many forms. The A* algorithm is widely used in pathfinding and graph traversal. Different map and maze images are used to test the system performance (100 images for each map and maze). The system overall performance is acceptable and able to find the shortest path between two points on the images. More than 85% images can find the shortest path between the selected two points.

Keywords

Pathfinding, Strategy Games, Map, Maze Solving, Artificial Intelligence, Search Algorithm, A*

1. Introduction

Pathfinding in computer games has become an investigated area for many years. It is just about the most popular but very difficult game Artificial Intelligence (AI) problem in game industry. The problem of pathfinding in commercial computer games has to be solved in real-time, usually under demands of limited memory and CPU resources [1]. The computational work is supposed to find a path by using a search algorithm. For this reason, pathfinding on large maps is capable of doing significant performance bottlenecks.

Pathfinding could be used to give answers to the question "How do I get from source to destination?". In most cases, the path from source (current point) to the destination (next point) could possibly include several different solutions, but if possible, the solution needs to cover the following goals:

1. The way to get from source A to destination B.
2. The way to get around obstacles in the way.
3. The way to find the shortest possible path.
4. The way to find the path quickly.

Some path finding algorithms solve none of these problems, and some others solve all of these problems. In some cases, no algorithm could solve any of these problems. An example of finding the shortest path from source to destination is shown in **Figure 1**. The shortest path with regards to distance is not always the fastest in time [2]. Different costs will be concerned with moving from source A to destination B rather than moving from source A to destination C. Pathfinding algorithm is concerned with the problem of finding the shortest path from source to destination and preventing obstacles [3]. Several search algorithms including A* search algorithm, Bread-First search algorithm and Depth-First search algorithm, were created to solve the problem of finding the shortest path.

Pathfinding has become a popular and frustrating problem in game industry because the importance of game industry increased. Games such as role-playing games and real-time strategy games usually have characters routed on a mission from their current location to a predetermined or player determined destination [1]. Agent movement is amongst the greatest challenges in the design of realistic AI in computer games. Pathfinding strategies are generally utilized as the core of any AI movement system. The most common challenge of pathfinding in video games is the way to avoid obstacles smartly and look for the most beneficial path over different places. Modern computer game industry is becoming much bigger and more complex every year, both with regards to the map size and the number of units existing in the industry [2] [4].

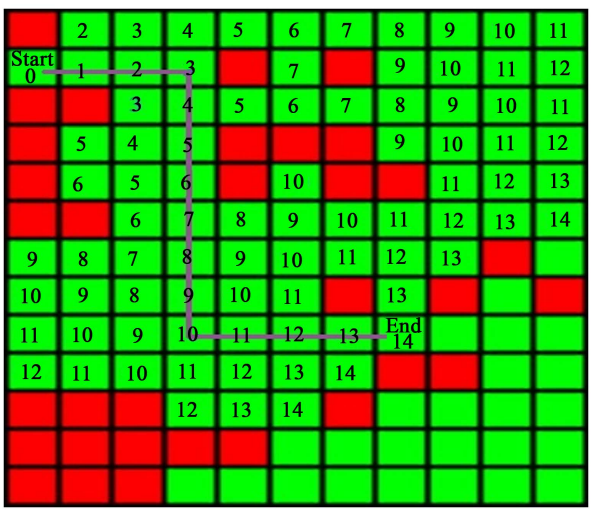

Figure 1. Shortest pathfinding from source to destination.

Pathfinding study comes with importance for different working areas including: logistics, game programming, operation management, system analysis and design, project management, network and production line. The shortest path study developed capability about thinking cause and effect, learning and thinking like human in AI.

Maze is a puzzled technique exactly where the maze will be discovered by the solver with using the most efficient route at the shortest possible time to reach the destination. In fact, the maze is widely known from the labyrinth, as the labyrinth comes with a single through route with twists and turns but without branches, and is not designed to be difficult to understand. In the maze, the pathways and walls are fixed. The majority of the maze solving algorithms is strongly related to graph theory where maze without loops are similar to a tree in graph theory. When the maze has multiple solutions, the solver can find the shortest path from source to destination [5] [6].

1.1. Pathfinding Search Algorithms

Pathfinding is the study of figuring out how to get from source to destination. The shortest path problem is amongst the most well studied in the computer science literature. Given a weighted graph, the problem is to look for the minimum total weight path in the graph between pairs of nodes. There are several algorithms developed for variants of the problem. Variants include directed versus undirected edges. A graph is a number of nodes and arcs that connect them, and a labeled graph has one or more description attached with each node that distinguishes the node from any other node in the graph. Graph search is divided into blind search and heuristic search [7].

Blind search is sometimes called a uniformed search since it has no knowledge about its domain. The only option that a blind search is capable of doing is to distinguish a non-goal state from a goal state. Blind search has no preference as to which state (node) that may be expanded next; while heuristic search is the study of the algorithms and rules of discovery and development. Heuristics is rules of thumb which can solve a given problem, but do not guarantee a solution. Heuristics is knowledge about the domain that can help guide search and reasoning in the domain [7] [8].

1.1.1. Depth-First Search Algorithm

Depth-First search algorithm using Last-In-First-Out stack and are recursive in algorithm. This algorithm goes deeper into the search space at any time when this is possible. It is simple to implement, but major problem with this algorithm is that it requires large computing power for small increase in map size [6] [8].

1.1.2. Depth-Limited Search Algorithm

Depth-Limited search algorithm is a uniformed search and works just like Depth-First search algorithm, but avoids its drawbacks regarding completeness by imposing a maximum limit on the depth of the search. This algorithm will find a solution whenever it is within the depth limit, which guarantees at least completeness on all graphs.

1.1.3. Breadth-First Search Algorithm

Breadth-First search algorithm uses First-In-First-Out queue. This algorithm involves

visiting nodes one at a time. This Breadth-First search algorithm visits nodes in the order of their distance from the source node, where distance is measured as the number of traversed edges [6] [8].

1.1.4. Best-First Search Algorithm

Best-First search algorithm uses two lists (open list and closed list) to limitation states just like Breadth-First search algorithm. At each step, Best-First search sorts the queue according to a heuristic function. This algorithm simply picks the unvisited node with the best heuristic value to visit next.

1.1.5. Hill-Climbing Search Algorithm

Hill-Climbing search algorithm expands the current state in the search and evaluates its children. It is an iterative algorithm that starts with arbitrary solution to a problem, and then makes an effort to find a better solution by incrementally changing a single element of the solution. If the change produces a better solution, an incremental change is made to the new solution, repeating until no additional improvement can be located. This algorithm cannot recover from failures of its strategy.

1.1.6. The Flood-Fill Search Algorithm

In Flood-Fill search algorithm a data structure is maintained that acts for the entire known map. This algorithm is very simple, but it requires a huge number of procedural calls that can cause recursion stack to overflow, with no mechanism to determine whether the visited pixels are actually tested before. Let suppose (x, y) is the center pixel, then (x, y) needs to be filled when one or more of its four adjacent horizontal or vertical neighbors are filled. Diagonal pixels are not considered as connected in this scheme [9].

2. Challenges in Computer Game Artificial Intelligence

In this section, main problems that occur when developing AI for computer games are briefly described. The list is not including all problems, but is made to give an idea of the problems type that real computer games represent to the AI community. The list of challenging problems in computer game is shown in **Table 1**. The conclusion which can be taken through this list of problems is that not only games can benefit from better AI techniques, but AI also can benefit from the challenges that computer games offer [10].

3. The Proposed Methodology

Pathfinding requires significant amount of resources especially in movement intensive games. For that reason, an efficient and inexpensive approach is needed. In this work, images from strategy games that represent a map and maze solving images are used. A* is widely used in pathfinding and graph traversal. This algorithm will be used to find the shortest path between two points on the image. The proposed methodology is shown in **Figure 2**.

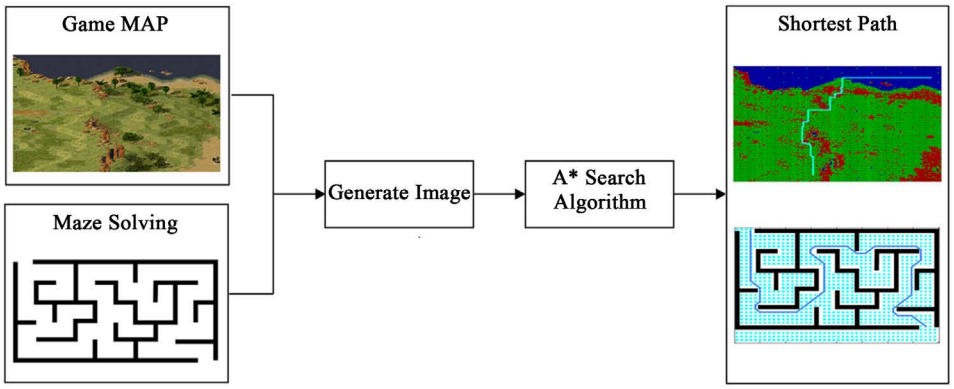

Figure 2. The proposed methodology.

Table 1. Computer game artificial intelligence challenging problems.

#	Problem	Description
1	Complex decision spaces	Most of the state-of-the-art computer games involve complex strategic (real-time strategy games) or believable behavior (interactive dramas). The two kinds of behaviors share the characteristic of obtaining huge decision spaces.
2	Authoring support	Hand crafted behaviors are ultimately software code in a complex programming language, at risk from human errors.
3	Unanticipated situations	It is certainly not feasible to prepare for all possible situations and player strategies which can came across during game play.
4	Knowledge engineering	Even assuming that strategies or behaviors are handcrafted, authoring these kinds of behavior sets in a game requires a huge human engineering effort.
5	Replay ability and variability	Player might get bored of seeing the same strategies and behaviors again and again.
6	Rhetorical objective	It is possible, that human engineered behaviors or strategies do not achieve the game objectives completely.

3.1. Image Acquisition

Images from strategy games are used. The map is converted into three main colors (red, green, and blue). There are three elements that can be move on the map (first element can only move on the blue area that represent the water, second element can only move on the green area that represent the ground, and the third element can move anywhere on the map that represent the amphibious). Maze solving also used in this work, white area represents walkable space while black area represents non-walkable space. With the development of computers, maze solving algorithms are becoming automated, but the execution time required solving the maze still scale unfavorably with maze size and complexity. The input images are shown in **Figure 3**.

3.2. A* Search Algorithm

A* is a generic search algorithm which can be used to find solutions for several problems, pathfinding basically to be one of them. This algorithm brings together feature of uniform-cost search and heuristic search. For pathfinding, A* algorithm again and again examines the most offering unexplored location it has seen. When a location is

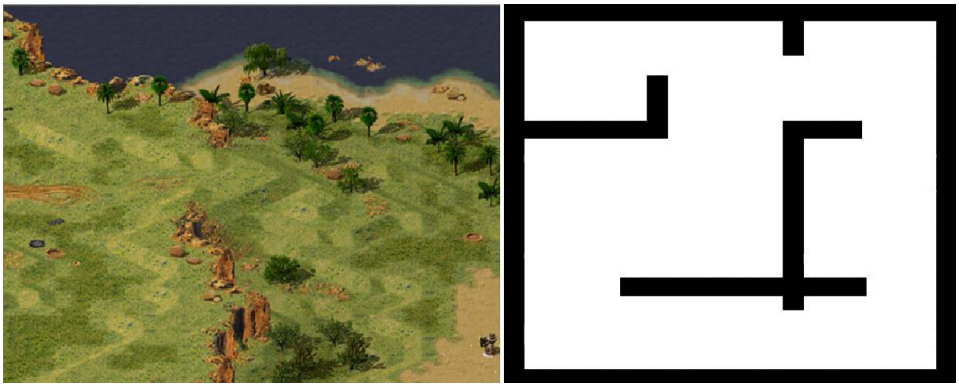

Figure 3. The original input images.

examined, the A* algorithm is completed when that location is the goal. In any other case, it helps make note of all that location neighbors for additional exploration [7]. A* could possibly be the most popular pathfinding algorithm in game AI. The time complexity of this algorithm is depending on heuristic used [8].

3.2.1. How A* Works

The game map needs to be well prepared or pre-processed before the A* algorithm can work. The game environment will be handled as a graph [3]. This involves breaking the map into different points or locations, which are usually called nodes. These nodes are used to record the progress of the search. In addition to holding the map location, each node has three other attributes that are fitness, goal, and heuristic commonly known as *f*, *g*, and *h* respectively [8]. The purpose of *g*, *h*, and *f* is to quantify how promising a path is up to the present node. The attributes *g*, *h*, and *f* are described in **Table 2**. Different values will be assigned to paths between the nodes. Typically, these values would represent the distances between the nodes. The cost between nodes doesn't have to be distance. The cost could be time, if you wanted to find the path that takes the shortest amount of time to traverse. A* using two lists (open list and closed list). The open list contains all the nodes in the map that have not been totally explored yet. The closed list contains of all the nodes that have been totally explored. In addition to the standard open/closed lists, marker arrays can be used to find out whether a state is in the open or closed list [7] [11].

3.2.2. A* Algorithm

1. Let's *P* is the source node.
2. Assign *g*, *h*, and *f* values to *P*.
3. Add the source node to the open list.
4. Repeat the following steps:

A. Look for the node which has the lowest *f* on the open list. Refer to this node as the current node.

B. Switch it to the closed list.

C. For each reachable node from the current node

Table 2. Attributes description.

#	Attribute	Description
1	g	Represent the cost of getting from the source node to the destination node (the summation of all values in the path between the source and destination.
2	h	Represent the estimated cost from the source node to the destination node.
3	f	Represent the summation of g and h and is the best estimate of the cost for the path going through the source node. $f = g + h$.

a) If it is on the closed list, ignore it.
b) If it isn't on the open list, add it to open list. Make the current node the parent of this node. Record the g, h, and f value of this node.
c) If it is on the open list already, check to see if this is a better path. If so, change its parent to the current node, and recalculate the g and f value.
D. Stop when
a) Add the destination node to the closed list.
b) Fail to find the destination node, and the open list is empty.
5. Tracing backwards from the destination node to the source node. That is the path.

4. Experimental Results

The analysis is done using a laptop with Intel(R) Core(TM) i7-5500U CPU 2.40 GHz with 12.0 GB RAM. The programming language Visual Basic is utilized to build the application program. Graphical user interface (GUI) is built to make the system easier to use and speed up the user's work.

The main interface shows the image (map or maze) and the main buttons that can be used to implement the operations for pathfinding on the selected image. Browse Image button is used to load any image with different format from the hard drive. Generate Map button will convert the image into:

1. Convert the map into three main colors (red, green, and blue) as shown in Figure 4.
2. Convert the maze into a grid maze as shown in Figure 5.

A* Find Path button will find the shortest path between source to destination using A* search algorithm. The algorithm shows the ability to find the shortest path between the two points (source and destination) that must be given by the user.

Image compression techniques can be used to minimizing the image size without degrading the quality of the original image to an unacceptable level. The reduction in image size allows more images to be stored in the hard drive and reduce the computational time.

A* search algorithm will find the path between source and destination. Source and destination location must be selected. Then select one of the three elements that can move on the map (amphibious, ground, or water). Maze solving is considering the image as black and white image. There is one element that can only move on the maze white area. According to the information, the algorithm will find the shortest path. Without knowing which element is moving on the map, the system will find the path supposing that the element can move anywhere on the map as shown in Figure 6.

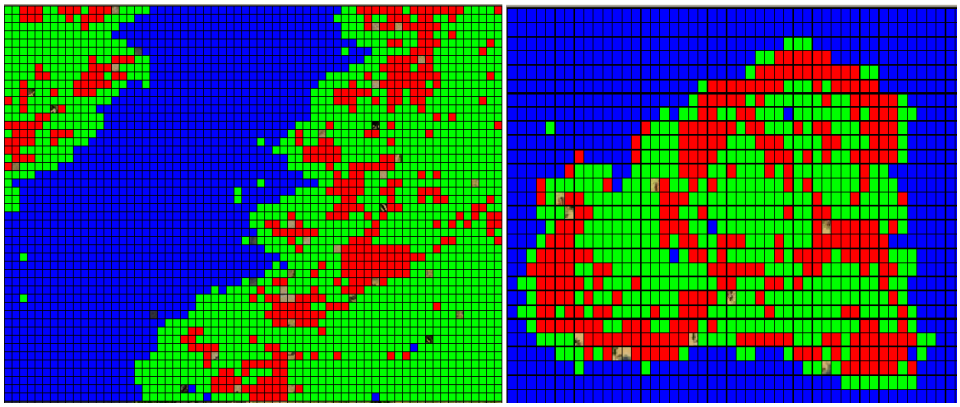

Figure 4. Map converted to red, green, and blue colors.

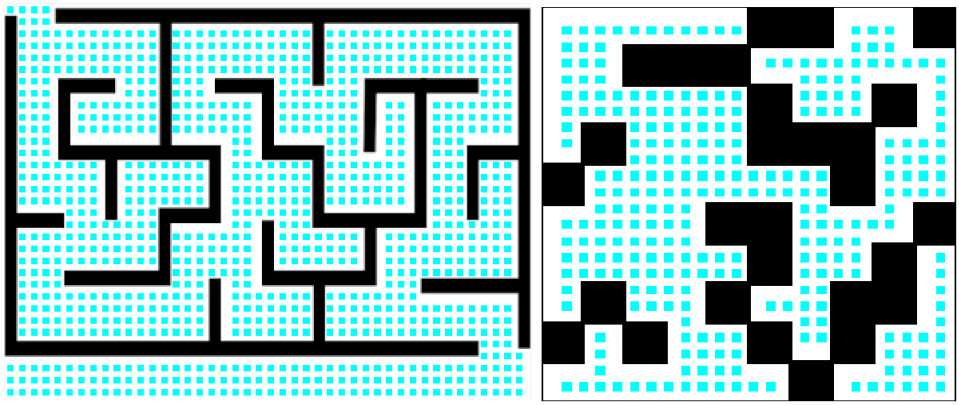

Figure 5. Maze converted to grid maze.

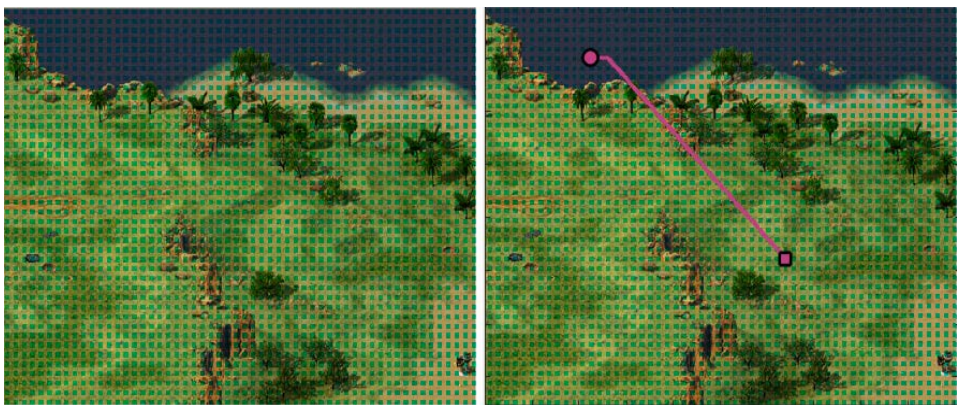

Figure 6. Pathfinding without knowing which element is moving.

A* search algorithm is directed algorithm which means that it does not blindly search for a path but instead it calculates the best direction to move and then it can backtrack the path. A* will not only find a path between source and destination but it will find the shortest path quickly. Pathfinding is so much powerful because it changes how the game is played. For example, the agents in the game "Red Alert" find the shortest path straight into the player base so that they can destroy him, the agents must

be smart enough to look for other path when the bridge in the map is destroyed. The system GUI implementation is shown in Figure 7. The result of finding the shortest path in game map is shown in Figure 8. The result of finding the shortest path in maze solving is shown in Figure 9.

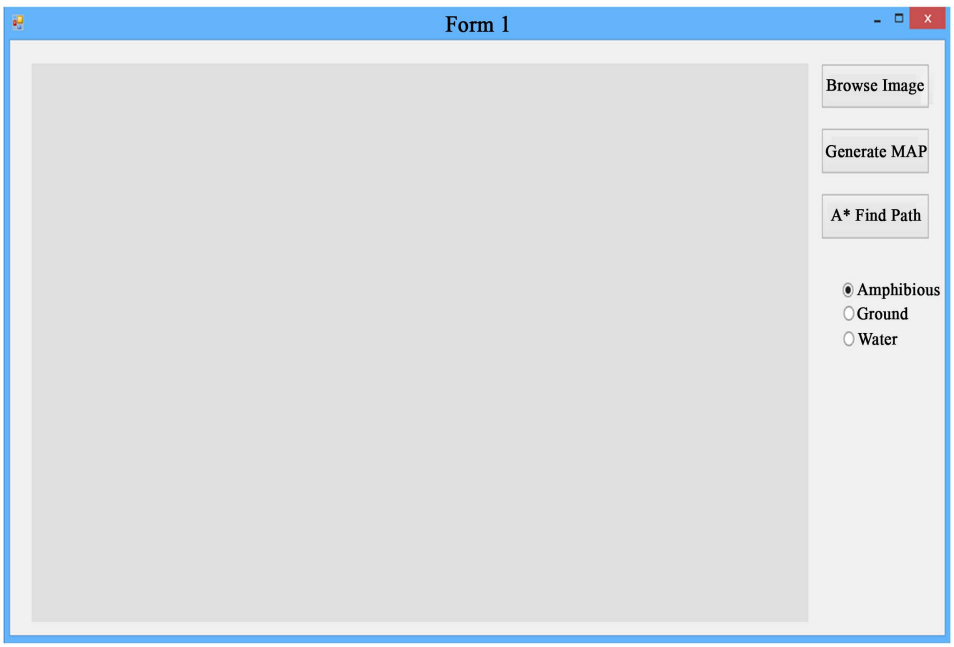

Figure 7. The system GUI implementation.

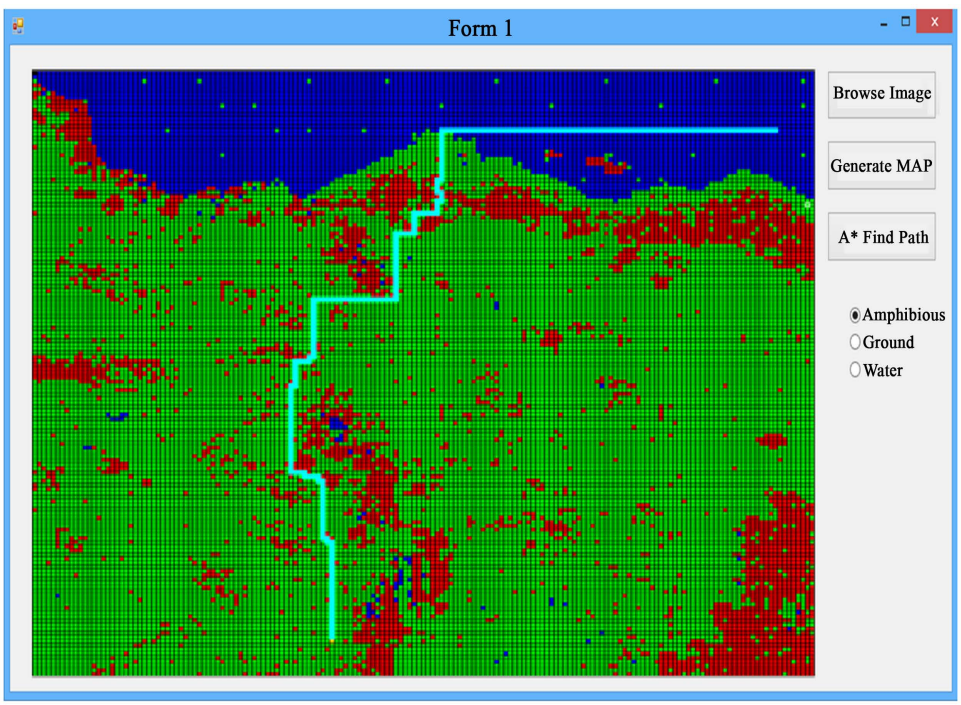

Figure 8. The shortest pathfinding using A* search algorithm.

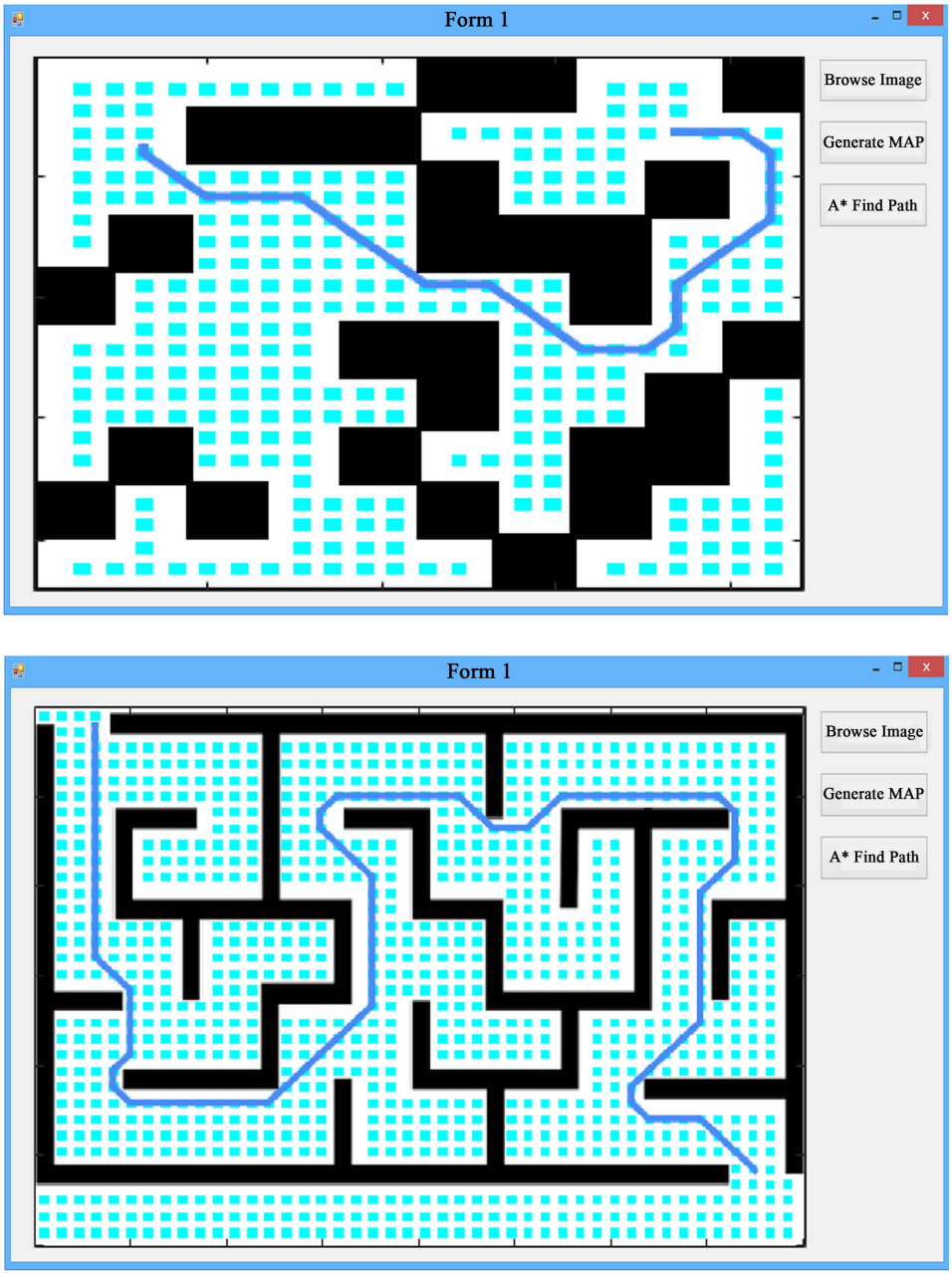

Figure 9. Maze solving using A* search algorithm.

5. Conclusion

The basic core of pathfinding algorithm is just a small piece of the puzzle in AI games. The most concern problem is how to use the algorithm to solve difficult problems. A* algorithm is the most popular algorithm in pathfinding. In this work, A* search algorithm is implemented to find the shortest path between source and destination. The algorithm is tested by using images that represent a map which belongs to strategy games or images that represent a maze. Pathfinding is so important in strategy games and

other applications. The map is converted into three main colors; each map has different correspondence to the three colors, while the maze considers the image as black and white. 100 different images for each map and maze are used. After selecting source and destination points, the system will find the shortest path between the selected points. The system overall performance is acceptable and able to find the shortest path between two points on the image. More than 85% images can find the shortest path between the selected two points. Other search algorithms can be used and each algorithm has different characteristics.

References

[1] Lawrence, R. and Bulitko, V. (2013) Database-Driven Real-Time Heuristic Search in Video-Game Pathfinding. *IEEE Transactions on Computational Intelligence and AI in Games*, **5**, 227-241. http://dx.doi.org/10.1109/TCIAIG.2012.2230632

[2] Algfoor, Z.A., Sunar, M.S. and Kolivand, H. (2015) A Comprehensive Study on Pathfinding Techniques for Robotics and Video Games. *International Journal of Computer Games Technology*, **2015**, Article ID: 736138. http://dx.doi.org/10.1155/2015/736138

[3] Botea, A., Bouzy, B., Buro, M., Bauckhage, C. and Nau, D. (2013) Pathfinding in Games. *Dagstuhl Follow-Ups*, **6**, 21-31.

[4] Graham, R., McCabe, H. and Sheridan, S. (2015) Pathfinding in Computer Games. *The ITB Journal*, **4**, 57-81.

[5] Lagzi, I., Soh, S., Wesson, P.J., Browne, K.P. and Grzybowski, B.A. (2010) Maze Solving by Chemotactic Droplets. *Journal of the American Chemical Society*, **132**, 1198-1199. http://dx.doi.org/10.1021/ja9076793

[6] Gordon, V.S. and Matley, Z. (2004) Evolving Sparse Direction Maps for Maze Pathfinding. *Congress on Evolutionary Computation,* 2004, *CEC* 2004, Vol. 1, 835-838. http://dx.doi.org/10.1109/cec.2004.1330947

[7] Cui, X. and Shi, H. (2011) A*-Based Pathfinding in Modern Computer Games. *International Journal of Computer Science and Network Security*, **11**, 125-130.

[8] Ansari, A., Sayyed, M.A., Ratlamwala, K. and Shaikh, P. (2015) An Optimized Hybrid Approach for Path Finding. https://arxiv.org/ftp/arxiv/papers/1504/1504.02281.pdf

[9] Bond, C. (2011) An Efficient and Versatile Flood Fill Algorithm for Raster Scan Displays.

[10] Ram, A., Ontanón, S. and Mehta, M. (2007) Artificial Intelligence for Adaptive Computer Games. *FLAIRS Conference*, Key West, 7-9 May 2007, 22-29.

[11] Björnsson, Y., Enzenberger, M., Holte, R.C. and Schaeffer, J. (2005) Fringe Search: Beating A* at Pathfinding on Game Maps. *Proceedings of the* 2005 *IEEE Symposium on Computational Intelligence and Games* (*CIG*05), Essex University, Colchester, 4-6 April 2005, 125-132.