Using GBFS Algorithm to Determine Pathing for Bots in RTS Games

Aqil Abdul Aziz Syafiq / 13518002
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
E-mail (gmail): 13518002@std.stei.itb.ac.id

***Abstract*—RTS Games are usually games where you can move troops around the map to defeat enemies, this paper will describe an algorithm that can help decide the best moving pattern for the bots when moving from a certain place to another in the game during the game.**

Keywords—GBFS;Graph;Bots;Pathing;RTS Games;Map;Movement

I. INTRODUCTION

Games have been around for centuries, the slightest rules made to have fun are arguably games, and as such, new games have always been found and played by people worldwide, around the clock.

With the advancement of technology, more sophisticated games with stricter game rules, or different ways of playing are immplemented, board games, group games, and finally, computer games.

Figure 1 : Illustration of Video games (google)

Games are fun to play, but there is always one problem, what can you do to have fun, if no one is there to play with you? You might be a loner, or your friends are just away, or maybe you have very different taste in games than your friends, what can you do? Well, unlike other type of games, computer games have the answer, bots.

Bots are Artificially made Intelligence models made to play a specific game, this way, even without friends, you can play your game alongside--be it against or with—bots.

A Bot program will contain everything for a bot to do on all situations, there are a lot of ways to implement bots. The author chose this topic because of their love for games, and also for algorithms.

II. BASE THEORY

A. *Games*

Games are forms of play or sport, which are played by the rules set for each specific game, and which outcome is decided by player's skill, luck, or strength.

The games in this paper refers to video games, games that are played on pc, console, or anything digital, that are playable by multiple entities, unlike sudoku, puzzles, and such.

B. *Bot*

Bots are autonomous program that can interact with problems and solve them by themselves.

In this paper, 'Bots' refer to game bots. Bots that are specifically made for specific games, for specific characters, for specific purposes.

Bots have to be smart, and capable of doing its task, or it might ruin it's purpose overall.

Figure 1 : Dumb bots trying to go through impassable fence (google images)

C. *Pathing*

Pathing is a term, used to define the path taken by a character, when trying to go from one point to another, this means that deciding each step to be taken from a point to another while taking in consideration the obstacles, enemy position, and other if needed. And construct a single pathing to go there.

Figure 2 : Pathing to goal point in green (screenshot)

D. *Graph*

To understand GBFS, understanding of graph will be needed. Graphs are groups of nodes, which are connected by edges.

Graphs are used to represent problems, because most of the times, if a problem can be represented by a graph, solving it will be much easier in its graph forms.

The representation of solution finding can be done by drawing the node space, which records all the moves done by the algorithm to reach the current state.

In a graph, nodes that are connected by an edge are considered neighbours.

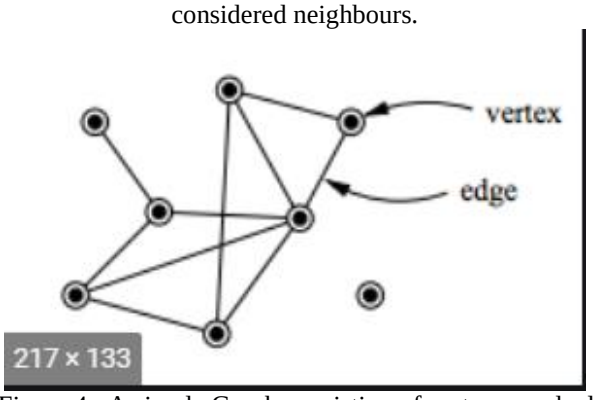


Figure 4 : A simple Graph consisting of vertexes and edges (Google images)

E. *State Space (Ruang Status)*

Keeps track of all checked and will be checked node in the graph, in which each node represents a state of the problem-in-solving, and shows which part got cutoff, and which got continued. Illustrated very easily so it can be understood by most.

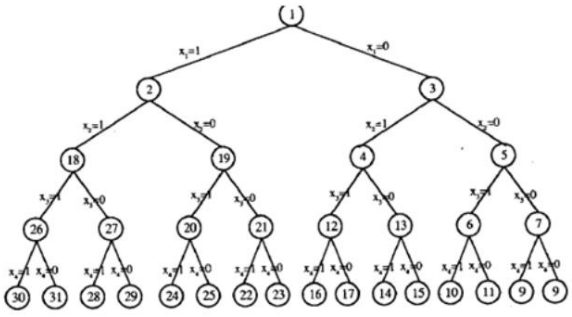


Figure 3 : State Space of a Graph Traversal Sequence (google images)

F. *Graph Traversal*

Graph traversal is something done, to find a certain node in the graph, or a set of edge, or anything that can be considered a solution to a problem by going through the graphs.

Usually in GBFS, a solution is a set of edges, which represents the step taken from source node, to destination node.

There are many other methods to traverse through graphs and each have their own uniqueness and good point. Each of them have their own best case scenario and worst case scenario, a good programmer would know when to use what algorithm and why.

G. *Greedy Best First Search (GBFS)*

Greedy best first search or GBFS is one of the many graph traversal algorithms that takes whatever cost is needed to achieve solution node, and choose moves with least cost possible.

Unlike BFS, and DFS, GBFS will, most of the time, give better results in term of cost, this is because it will always take routes where it is cheapest.

The implementation goes as follow, first make a frontier, a frontier is a space, where all the possible nodes that will be checked are kept, frontier starts off with only 1 node inside it. The starting state.

After that, for every move, every change, every other nodes that are connected by an edge to the starting state, if it is a legal move, it is pushed to the frontier if the same state resulting from the action is not already in the set, and after that, a representation of the state is added to the set, to avoid the same state being checked over and over again.

Next, while the frontier is not empty, it will traverse the frontier to check, which of it is the best node in terms of cost. And pop it out of the frontier, to check if it is the goal state or not, if not, it will be expanded in similar manner as in the paragraph before this one.

This shall continue on and on until 2 possible outcome, either the frontier is empty, or the goal state is found, the former of course means, that there is no goal state found by the algorithm. It does not necessarily means that no goal state exist, maybe it just means that the algorithm, or chosen manner of cost computing does not work for the case tested.

III. Research, Testing, and Analysis

A. Problem Representation

To solve the said problem, that is figuring out a path amidst the map to find the best possible way for a character to move from it's current position to a certain target position. The problem must have a graph representation or at least, a representation that can be solved by graph traversal algorithm.

Now this is easy, because path finding is one of the most common usage of graphs, therefore, due to the presence of start position, obstacle, and goal position, author has chosen the shape of a Maze, to represent the problem.

Like author stated earlier, this is because simple pathfinding is used for this, and a maze can easily represent the obstacle, starting position and goal position of a bot trying to go somewhere.

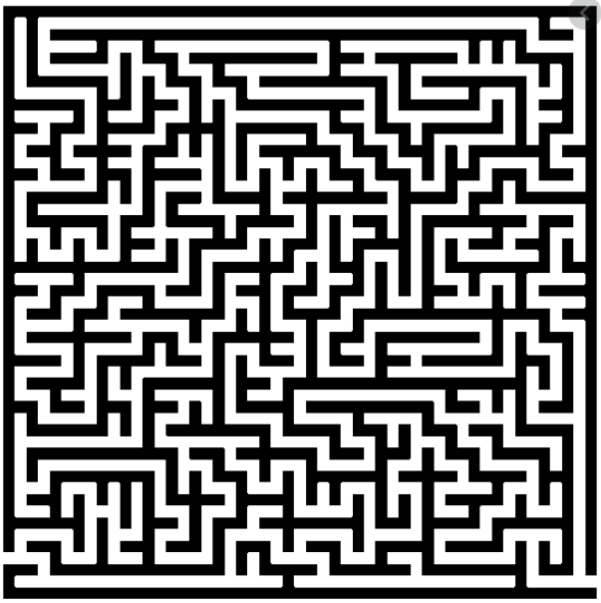


Figure 5 : Literal Maze (Google images)

Despite the picture above, the representation of a maze we are going to use looks more like this :

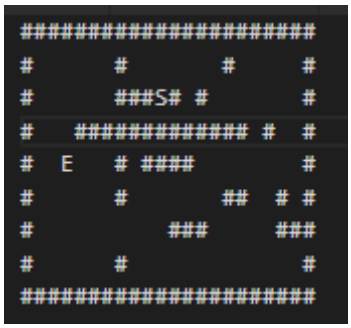


Figure 6 : Maze representation (screenshot)

With this representation, it is easier to implement, as it can be represented by a matrix of character in program, and since it is a matrix, moving between indexes and stuff are easier, and manipulation are easier to be done too, and also how easy it is to understand and comprehend, and that it does not require complex understanding. Author have concluded that this is the best representation to use for this problem, and thus decides to use this representation.

B. GBFS Program

The base algorithm for GBFS was already explained by author in the previous section, in this section author will explain it using pseudocode.

```
frontier = empty array
visited = empty set
frontier.append( initialstate )

while frontier not empty:
    node = frontier.pop()
    check(node)
    if check then
        solution
    else
        for all neighboring nodes do
            check if already visited or not
            if not then
                visit neighboring node
                put in frontier with the cost
                add neighboring node to set
            else
                do nothing with this neighboring node
```

As it was explained before, the frontier will keep all the candidate nodes, which will be checked, and everytime a node is put in the frontier, it is ordered according to the cost of the new node descendingly. Thus, everytime a node is popped, the smallest cost will be at the end of an array and requires no further traversing to find minimum cost.

The check part of the algorithm depends on what kind of solution that is checked, for instance, in this problem we are checking the position of the current node, is it, or is it not, in the same coordinate as the goal position. If it is, then the program will stop and returns the result, if it is not the goal, it will continue on and on until it finds the goal.

What is put in the frontier differs according to need, the Node itself goes without saying, but a node can be represented by other stuff instead of a Node object, for faster execution purposes, in author's case, the cost, position, and path is pushed to the frontier, the cost is used to keep the frontier ordered descendingly by cost, the position is needed as it is the representation of the current node, it is enough, as no matter how a node is reached, other path to it doesn't matter, as GBFS will always find one most efficient path to a node, the path itself is the current step taken from the start, this is used for result showing purposes in the program.

This concludes author's pseudocode of the program, and author believe that most of the source code is not necessary, but it will be put in a video for those with interests.

C. Cases

It is not very hard to find cases, as I am a gamer myself, I find myself in a lot of situation in-game that are representable for this problem. In this case, author will only use 1 game to do cases, for simplicity and consistency purposes, also to keep it short. Author will use the game DotA 2.

i. DotA 2

DotA 2 is a strategy MOBA (Multi player Online Battle Arena) game, while the game says multi player, it is possible to play it with bots, and of course, bots move around the map normally to work with each other or with the player to win the game for their team.

Figure 7 : DotA 2 (google images)

ii. Movement in DotA 2

In DotA 2, movements are done by clicking right click on the map, of course, in this way players can decide themselves, which pathing they want to take when going from 1 point to another.

But for bots, when they want to go from point A to point B, they must decide which path is best, for the path may or may not be blocked by another character, a tree, ledges, and other various obstaces.

Now author will go ingame and put their character in a situation befitting of a test case for our problem. The author came out with this position:

Figure 8 : The Case (screenshot)

As you can see in the picture, the scary stone giant is trapped behind trees and the map border. Here, we will try to help mr Tiny(stone giant guy) to go to his lovely pet pige, the flying pig on the left side of the trees. As you can see, the flying pig is lonely and is in need of a friend, therefore mr Tiny needs to find the shortest path to come to his pet's rescue.

Author shall help mr Tiny with his small problem. Like stated earlier in this section, author needs to convert said case to a representation solvable by graph traversal algorithms.

Author has decided on maze representation for it, and ideally, author would have made a converter of a screenshot to maze representation, but for now, author will make the maze representation manually.

Here is the maze author made in translation of the problem case :

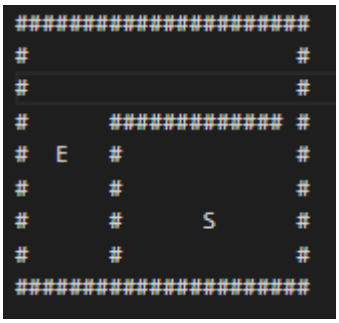


Figure 9 : Totally accurate maze representation (google images)

Ignoring everything else not important to the problem at hand, there are 4 entities in the maze, first off is S, the starting position represents mr Tiny trapped behind the trees, second we have E, representing mr Tiny's pet Pige, the flying pig, next we have # representing the trees and maze border, the maze border are put in position in which author believes, position beyond them are not even worth checking as it being called in the frontier will have 0 chance of happening. The # not in the border represent the evil trees blocking mr Tiny from Pige.

And so, we have the case, now we just have to solve it, onward to the next section !

D. Case Testing

For case testing obviously I have made a program to solve the path finding problems. The problem works by reading the maze shape from a txt file.

Author will show and explain some parts of the program, first of all, here is the implementation of a maze :

```
class Maze:
    def __init__(self):
    def read_string(self,maze):
    def read_file(self,filename):
    def write(self):
    def start_position(self):
    def end(self):
    def get(self,xy):
    def put(self,posi,a):
    def cost(self,pos):
    def move(self,pos,direction):
    def result(self,path):
    def show(self):
```

Figure 9 : Maze class and its methods (ss)

The methods are not shown so the whole thing can fit, notable methods are read_file, to read from certain filename, cost (pos), to calculate the cost of a given coordinate, and show, which is used to print out the maze.

Next we have the solve function:

```
def solve(maze):
    frontier = pq()
    visited = set([])
    move = (0,1),(1,0),(0,-1),(-1,0)
    movedir = '>','v','<','^'

    initial = (maze.cost(maze.start_position()),maze.start_position(),'')
    frontier.put(initial)
            #cost, pos, path
    while len(frontier.arr) > 0:
        _,pos,path = frontier.get()
        # print(pos)
        # print(path)

        for i,direction in enumerate(move):
            # print(pos + direction)
            if maze.get(add(pos,direction)) == 'E':
                # print(path+movedir[i])
                return path + movedir[i]
            if maze.get(add(pos,direction)) != '#' and str(add(pos,direction)) not in visited:
                frontier.put((maze.cost(add(pos,direction)),add(pos,direction),path+movedir[i]))
                visited.add(str(add(pos,direction)))
    return path
```

Figure 10 : Solve function(screenshot)

Note that the Greedy part of the algorithm lies within the frontier put and pop method, they are kept at orderly fashion descendingly, so that whatever popped value comes out, it is assured to have least cost out of all.

So here is the Main program :

```
if __name__ == "__main__":
    INS = Maze()
    x = input("Masukkan nama file : ")
    INS.read_file(x)
    print("Initial State :")
    INS.show()
    path = solve(INS)
    print("Solution : ")
    INS.result(path)
```

Figure 11 : Main program (screenshot)

Author believe that the picture itself should be self-explanatory, so first author will try to run something that is not the meant test case, lets just say that this, is a randomly generated maze for program testing purpose:

```
######################
#      #       #     #
#      ###S# #       #
#   ############## #  #
#  E   # ####        #
#      #       ##  # #
#          ###     ###
#      #             #
######################
```

Figure 12 : Totally randomly generated maze

Here is the solution given by the program :

```
F:\FOLDERKULIAH\INFORMATIKA\STIMARESEARCH>py MazeSolving.py
Masukkan nama file : input.txt
Initial State :
######################
#      #       #     #
#      ###S# #       #
#   ############## #  #
#  E   # ####        #
#      #       ##  # #
#          ###     ###
#      #             #
######################
Solution :
######################
#      #  >>>>v#     #
#      ###S# #>>>>>v #
#   ############## #v #
#  E   # ####     v< #
#  ^   #       ## v# #
#  ^<<<<<  ###    v###
#      #^<<<<<<<<<<  #
######################
```

Figure 13 : Program output (screenshot)

As reader and author can see, the program will print out the maze, but marks the way for it to go to the goal position, and because of GBFS algorithm, this is most likely the optimal solution.

Now we can move forward for the earlier prepared test case, here is the output for such a case :

```
F:\FOLDERKULIAH\INFORMATIKA\STIMARESEARCH>py MazeSolving.py
Masukkan nama file : tes.txt
Initial State :
######################
#                    #
#                    #
#      ############# #
#  E   #             #
#      #             #
#      #     S       #
#      #             #
######################
Solution :
######################
#                    #
#     v<<<<<<<<<<<<<<#
#     v#############^#
#  E<<<#            ^#
#      #            ^#
#      #      S>>>>>^#
#      #             #
######################
```

Figure 14 : Output with ‘THE’ test case (screenshot)

Based on earlier solution and this one, we can assume that this is the optimal path to the solution that exists.

Now, as we have the solution for the problem in maze representation, we mush convert it back. That means that, in-game, it will look like this:

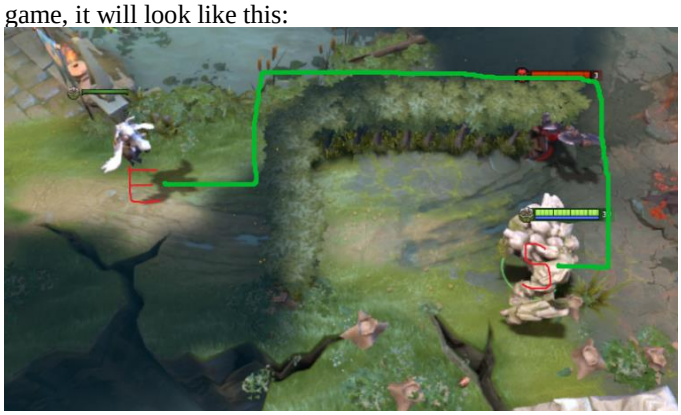


Figure 15 : Solution (Screenshot)

Now, while this may seem obvious for humans, for bots, there is no such thing as obvious and such, even for the simplest solution to a problem, a bot only knows the algorithm and the problem, and the solution for the problem using the algorithm, no matter the complexity and stuff.

Of course, Author wants to state that this is not at all a complete program and can decide bot's move perfectly, as there should be even more factors to be considered in different games, like enemies, fog of war, mission purpose, and stuff like that.

IV. Conclusion

After all that, author must conclude, that although GBFS, and graph traversal techniques in general, are very excellent ways of path finding, there are many other techniques for bots to decide movements, GBFS is only one of many, author is not the first, and surely won't be the last to suggest such a thing.

From the program output we can conclude of how cost efficient, time efficient, and easily programmable this might be.

Author is happy to say that author's target of this research has been fulfilled and author wishes that this paper be useful to anyone reading.

Video Link at Youtube

https://youtu.be/-ko9eBJOFsY

Acknowledgement *(Heading 5)*

Author thanks god, for the ability to work on such a headache on a weekend, also while fasting. Author also wants to thank the much honorable instructors of Algorithm Strategy, that is Mr. Rinaldi Munir, Mrs. Ulfa, and Mrs. Masayu for the chance to work on this paper, and the chance to learn so many from them. Author wishes that they can meet these instructors again in future classes. Author also thanks everyone else that has helped directly or not in this paper.

References

[1] https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2017-2018/A-Star-Best-FS-dan-UCS-(2018).pdf. Diakses 2 Maret 2020.

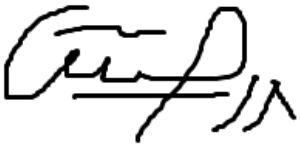
[2] https://www.google.co.id/imghp. Diakses 2 Maret 2020.

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 29 April 2020

Ttd



Aqil Abdul Aziz Syafiq

13518002