Greedy Best First Search in Automated Snake Game Solvers

Muhammad Daru Darmakusuma - 13518057
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
E-mail (gmail): darudarmakusuma@gmail.com

***Abstract*—Snake game is a classic game where the player maneuvers a line that will grow in length when the line achieve its goal that represented as a food, often as an apple. The line itself is an obstacle for the player to not hit itself. Such game like this can be automated with a route or path planning such as Greedy Best First Search algorithm. The algorithm help the line find its goal with a shortest path, but not promising the safety of the line from itself. The automated solver will be created by using Greedy Best First Search and some forward checking to ensure the line safety from hitting itself.**

Keywords—snake game, greedy best first search, automated

I. Introduction

Snake is a common concept in video game where the player play as a line that moves rapidly and the player need to control the direction where the line moves. The line that the player control grows as the line achieve its goal and the primary obstacle is the line itself to not be hit.

The concept of snake game is originated in the 1976 as an arcade game called *Blockade* and the easy of its making led to many versions of it that many has the word *snake* or *worm* in the title. The snake game concept became popular when Nokia loaded their mobile phone with the game variant in 1998.

The gameplay of snake is quite simple. The player controls a dot, square, object, or line in a plane. The object that player control will moves forward continuously and leave some trail behind like a moving snake, so the snake rapidly gets longer as it achieve its goal. The player loses the game when the snake run itself toward an obstacle like screen border, itself, or other object.

As a classic game it is, the snake game has a lot interest in developing the automated solver to beat the game. The game itself often does not have a winning state because the goal keep being updated as the snake achieve a goal. The game can be called an *endless game*, so that take interest in many programmer to solve the game in the most optimal way. The common objective of the solver is to gain many score as possible without hitting an obstacle, so the snake can fulfill the area or plane of the game.

The approach that can be used to create one of a solver is using a route/path planning. The algorithm that will be used is Greedy Best First Search and a forward checking to see if the snake in a safe state.

II. Theory

A. Graph

First, to know how the Greedy Best First Search works, we need to know the concept of a graph. Graph is a data structure which non-linear and has components such as nodes and edges. Nodes in graph often referred as the vertices and the edges of the graph are the arcs that link two or more nodes in a graph.

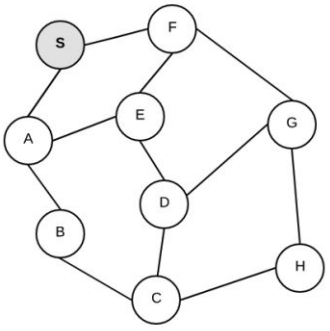


Figure 1. Example of a graph
https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2019-2020/BFS-dan-DFS-(2020).pdf

Graph can be used to solve many problems by representing it as a network which prescribe as link of solutions. Nodes in a graph contains a needed information as what the graph represent. In a solution finding algorithm, a graph often used as a medium to traverse the solution.

B. Graph Traversal

A graph can be used to represent a problem, so to find the solution there is need to be the algorithm to traverse and visit the nodes of the graph systematically.

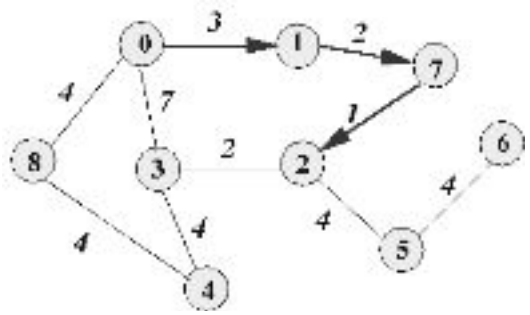


Figure 2. Example of traversing a graph with 0 as the starting node
http://www.mathcs.emory.edu/~cheung/Courses/171/Syllabus/11-Graph/traverse.html

With the assumption that the graph is connected, we can traverse to its nodes with two approach of graph representation:

1. Static Graph

   The graph is already fixed and constructed before the search begin. The graph must completely drawn and the data must be injected, this type often called as *define-and-run*. The graph represented as an structured data.

2. Dynamic Graph

   The graph is being constructed as the search being done. The graph is being defined on the go via an actual forward computation, this type often called as an *define-by-run*.

For traversing the graph there is two type of algorithm that can be used. The algorithms to search the graph:

1. Uninformed Search

   The traversing with uninformed search does not provide any extra information to compute. Example for this search are DFS, BFS, Depth Limited Search, Iterative Deepening Search, and Uniform Cost Search.

2. Informed Search

   The search of graph based on heuristic search. The search able to know non-goal state that more promising than other. Example for this search are Best First Search and A* (A-Star).

C. *Greedy Best First Search*

Each nodes in graph has an evaluation function, that is:

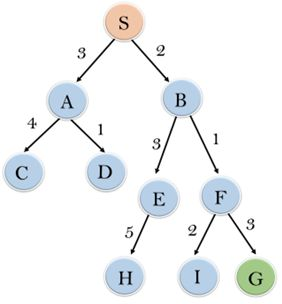
$$f(n) = h(n)$$

f(n) provides the estimated total cost for the nodes and the search will be expanded at the node with smallest *f(n)*. Often, for best-first algorithms, *f* is defined in terms of a heuristic function, *h(n)*. Heuristic functions are the common form way passing the additional knowledge of problem to the search algorithm.

Greedy Best First Search algorithm always pick the path that appears to be the best for the moment which usually called as local minima or *plateu*. This algorithm take both Breadth First Search and Depth First Search as it implementation. Best First Search helped the algorithm to take the advantages from those two algorithm. As the search go with Best First Search, each step the algorithm pick the most promising node. The Best First Search algorithm will expand the node that is closest and has less cost than the others.
Greedy Best First Search algorithm follow these steps:

1. **Step 1**: Place the starting node into the OPEN list.
2. **Step 2:** If the OPEN list is empty, the solution not found, stop, and return failure.
3. **Step 3:** Remove the node n, from the OPEN list which has the lowest value of h(n), and places it in the CLOSED list.
4. **Step 4:** Expand the node n, and generate the successors of node n.
5. **Step 5:** Check each successor of node n, and find whether any node is a goal node or not. If any successor node is goal node, then return success and terminate the search, else proceed to Step 6.
6. **Step 6:** For each successor node, algorithm checks for evaluation function f(n), and then check if the node has been in either OPEN or CLOSED list. If the node has not been in both list, then add it to the OPEN list.
7. **Step 7:** Return to Step 2.

As a graph in "Figure 3" we could traverse it with Greedy Best First Search. At each iteration, each expanded node is evaluated with the evaluation function *f(n)=h(n).*



node	H (n)
A	12
B	4
C	7
D	3
E	8
F	2
H	4
I	9
S	13
G	0

Figure 3. Example of a graph for Greedy Best First Search
https://www.javatpoint.com/ai-informed-search-algorithms

Expand the nodes of S and put in the CLOSED list

Initialization: *Open [A, B], Closed [S]*

Iteration 1: *Open [A], Closed [S, B]*

Iteration 2: *Open [E, F, A], Closed [S, B]*

: *Open [E, A], Closed [S, B, F]*

Iteration 3: *Open [I, G, E, A], Closed [S, B, F]*

: *Open [I, E, A], Closed [S, B, F, G]*

*Hence the final solution path will be****: S----> B----->F----> G***

Greedy Best First Search algorithm had the advantages that is more efficient than BFS or DFS and can do both at the same time having both advantages. The algorithm disadvantages is having an unguided Depth First Search that led to the worst case scenario, it will stuck in a loop. The algorithm is not optimal too to gain the solution.

III. APPLICATION

The variant of the snake game that we use is where the game area is a toroid, the snake can move pass the border of the screen and will reappear in the other side of the border it pass. The player loses when the snake hits its own body and the snake grows longer every time the player reach the goal or which can be represented as food.

By representing the position of the head of snake as the node, each expansion has four successors which is the direction of the head to go. The direction would be the edges to each node.

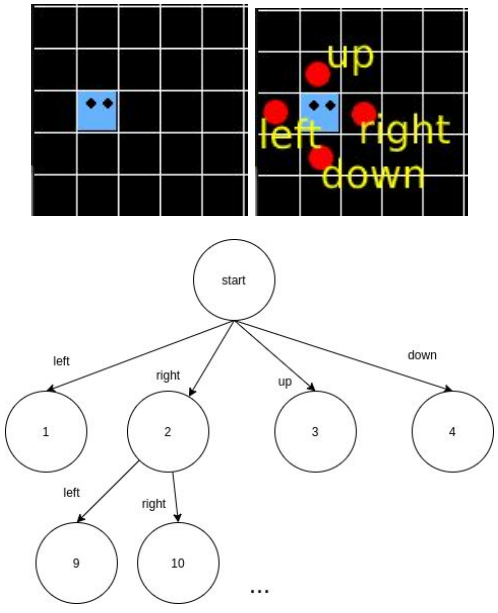


Figure 4. Graph representation for the Snake Game

A. Analysis

Each nodes contains the information of the head position and how far the head to the food or the goal. The heuristic function to find the distance and the lowest cost to a node is by using the Manhattan Distance between the head of the snake and the goal (food) which is:

$$d = |x_1 - x_2| - |y_1 - y_2|$$

where $p_1 = (x_1,y_1)$ and $p_2 = (x_2,y_2)$, p_1 is the position of the food and p_2 is the position of the head of snake. The distance represented by d, so we can safely say $f(n)=h(n)$ where $h(n) = d$.

The heuristic function that we use need to be modified because we use the variant of snake where the game area behave as a toroid. The distance would be different if the snake can move pass the border. We need to know the size of the game area to calculate the true distance. So to calculate the distance in this variant:

$$dx = \min(|x_1 - x_2|, size - |x_1 - x_2|)$$

$$dy = \min(|y_1 - y_2|, size - |y_1 - y_2|)$$

$$d = dx - dy$$

After we defined our heuristic function for the algorithm to work with, we could easily find the minimum route for the snake to get its food.

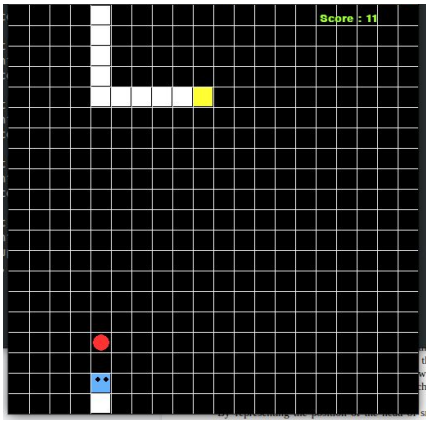


Figure 5. Example of passing the border as nearest

The automated solver can be called finished because now the snake can find its own food, but there is a problem by only using this algorithm. The algorithm did not care if the shortest path has the snake own body on its way, so it will keep hitting itself as it goes and does not return a good amount of score. The Greedy Best First Search do not look up to how the snake will end up. The algorithm just direct the snake to the nearest path to get the food. So we need another algorithm to guard the snake along the way.

The forward checking method that was made is spontaneous and instant by checking few steps further for the snake head and see if there is a body of itself. Every direction

get checked if there is a body of snake a few blocks away, the more it does, the more the snake would avoid those area.

For the record, the snake cannot got to the opposite direction when the length of the snake is greater than two. The snake will hit his own body. So when the nodes of successors at level three of graph, it will only create three successors onwards.

B. Implementation

Implementing the Greedy Best First Search algorithm is quite an ease, the pseudocode for the algorithm:

```
nodes = []
visited = []
for dir in  every direction:
        nodes.append(dir,
        manhattan_dist(snake.head.pos,food.pos,areasize),
        new head position)
sort nodes by dist
for move in nodes:
        if move possible and move not in visited:
                do move
                visited.append(move)
```

We implements the Manhattan Distance as follows:

```
int manhattan_dis(p,q,size):
   dx = min( abs( q.x - p.x ), size - abs( q.x - p.x ) )
   dy = min( abs( q.y - p.y ), size - abs( q.y - p.y ) )
   return dx + dy
```

Visited list is representing the closed list in the Greedy Best First Search and the nodes list represent the opened list in the algorithm. The possible move to do is not countering the current direction and the state is not in visited list so there is no redundancy to expand in the same node.

We know that the Greedy Best First Search does not provide any safety to the snake from any obstacle so we need to modify the algorithm with a checker, a forward checker.

The pseudocode of the Greedy Best First Search algorithm with a simple forward checking can be put like this:

```
nodes = []
visited = []
for dir in  every direction:
        nodes.append(dir,
        manhattan_dist(snake.head.pos,food.pos,areasize),
        new head position, prio = 0)
for move in nodes:
        newPrio = forwardChecker(move)
        move.prio = newPrio
sort nodes by prio, dist
for move in nodes:
        if move possible and move not in visited:
                do move
                visited.append(move)
```

The prio that was added is meant priority to sort what the best move to do by looking few steps ahead. The larger the prio, the worse the move to be done.

Forward checking for the algorithm is made by analyzing what is the common mistake the snake do when it is only use the Greedy Best First Search algorithm. We deconstruct some of the problem for the algorithm. The first one is the snake did not care if there its own body in the way, to solve this problem we need to detect it and add the priority value to the move.

```
if move.newposition is in snake.body.position:
        move.prio += 1
```

The code indicates that the new position of the head is in the body itself. So the priority value got higher which in this case reversed, the higher the worse. By that, we know that the move can be fatal to the snake, so we put it in the end of queue.

If all the move is safe, we need to check the safer move to do by checking some steps ahead if there is a block of snake body or not. We can check it by see three blocks away from the snake head in every direction. The checker will count the amount of snake's body block in each direction 3 blocks away.

```
for 3 block from direction:
        if block.pos is in snake.body.pos:
                move.prio += 1
```

The nearest block in which direction will have the priority added to so the snake stay far away from its own body. To calculate how near the block is, we can use the Manhattan Distance again.

```
for 3 block from every direction:
        if nearest(block,s.body):
                move.prio += 1
```

The common mistake the snake do is to fall on its own trap. The trap was made when the snake go circular and made shape like prison so it can go nowhere. This forward checking was the hardest, so i decide to make a naive checking in every eights block around the snake's head. If there is a diagonal pattern making a door like gesture, the snake must not go in there and find a move to escape. The set of movements is still controlled via priority.

```
for 8 block around snake.head.pos:
        if there is a door like pattern to a direction:
                move.prio += 1
```

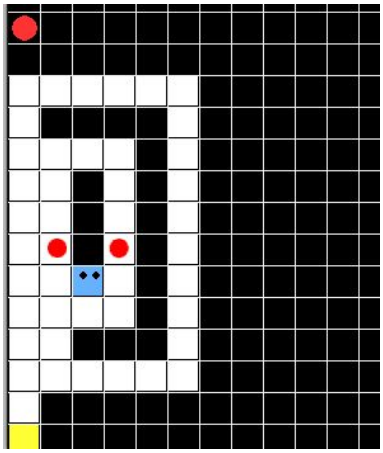


Figure 6. Door-like pattern signed by red dots in snake's body

By the look of "Figure 6", we can see there is a door-like pattern in the top direction of snake. The snake must escape from that direction by looking any best direction to get away. Unfortunately, the algorithm for checking this mistake just work on some few cases.

The success of each run is dependent to the food placement too, if the food placed in the position that is led to a dangerous state, make the snake stuck, it will guarantee it will lose.

C. Case Studies

The Greedy Best First Search algorithm is implemented in Python 3.7 as follows:

```
def best_first_search():
    global s, snack, visited, step
    step += 1
    curr_posx = s.body[0].pos[0]
    curr_posy = s.body[0].pos[1]
    nodes = [] # 0: left, 1: right, 2: up, 3: down
    # Measuring distances
    p = ((curr_posx-1)%20,curr_posy)
    nodes.append(('left',
manhattan_dis((curr_posx-1,curr_posy),snack.pos,size
=rows), p))
    p = ((curr_posx+1)%20,curr_posy)
    nodes.append(('right',
manhattan_dis((curr_posx+1,curr_posy),snack.pos,size
=rows), p))
    p = (curr_posx,(curr_posy-1)%20)
    nodes.append(('up',
manhattan_dis((curr_posx,curr_posy-1),snack.pos,size
=rows), p))
    p = (curr_posx,(curr_posy+1)%20)
    nodes.append(('down',
manhattan_dis((curr_posx,curr_posy+1),snack.pos,size
=rows), p))
    if set(nodes[:][2])<= set(list(map(lambda
z:z.pos,s.body))):
        s.move()
        return
    i = 0
    print()
    forwardChecking()
    best = []
    best = sorted(best,key=lambda t: t[1])
    print(best)
    # Do a best and possible move
    for p in best:
        print(i)
        if p[0] == "left" and s.curr_dir != "right"
and p not in visited:
            print("A")
            s.move(control="left")
            visited.add(p)
            return
        elif p[0] == "right" and s.curr_dir !=
"left" and p not in visited:
            print("B")
            s.move(control="right")
            visited.add(p)
            return
        elif p[0] == "up" and s.curr_dir != "down"
and p not in visited:
            print("C")
            s.move(control="up")
            visited.add(p)
            return
        elif p[0] == "down" and s.curr_dir != "up"
and p not in visited:
            print("D")
            s.move(control="down")
            visited.add(p)
            return
        i+=1
    s.move()
```

Running program:

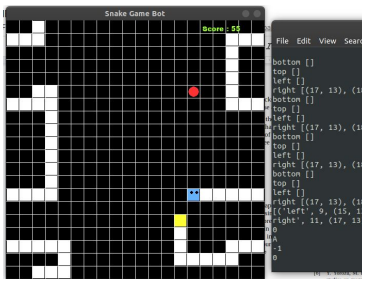


Figure 7. Example 1 of running bot

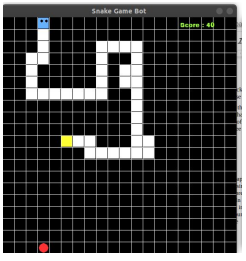
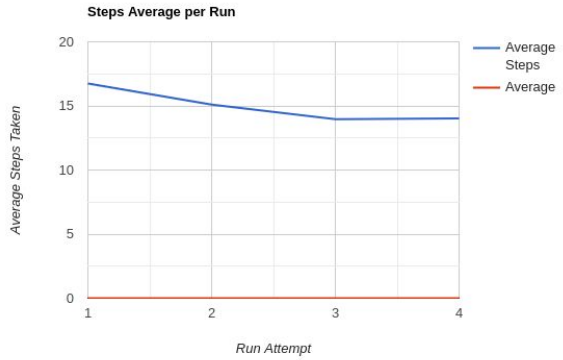
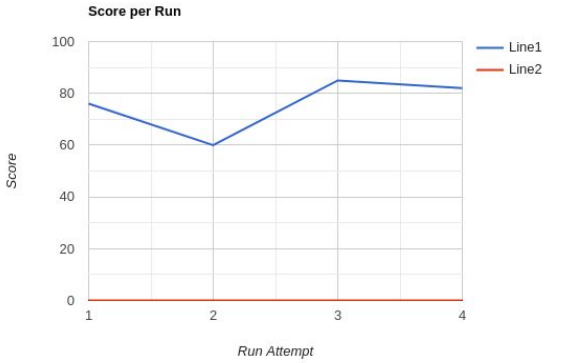


Figure 8. Example 2 of running bot

Statistic of the score and the steps taken each run of the automated snake game solver:





From the statistic of score every run and each steps every run, the automated solver is pretty consistent. Steps taken does not promise a good score.

Program executed in a computer system with specification:

No.	Hardware	Spesifikasi
1	Sistem Operasi	Ubuntu 18.04.4 LTS 64 Bit
2	Prosesor	Intel® Core™ i5-8265U CPU @ 1.60GHz × 8
3	RAM	8 GB
4	Compiler	Python 3.6.9

IV. Conclusion

Greedy Best First Search is a fast algorithm to implement as an automated snake solver and can find the smallest amount of step to find the goal. Unfortunately the algorithm have the disadvantages that can be stuck at an unguided depth-search that will lead to losing. The food placement is a factor to make a run a success. So to complete the algorithm, we need to build a forward checking.

Video Link at Youtube

https://youtu.be/Y-yVeJZXKpI

Source Code Link

https://github.com/mdarud/snake-bot.git

Acknowledgment

First, the author would like to be thankful to God. The author like to express his gratitude and appreciate Mr. Rinaldi Munir for his teachings and lectures in Algorithm Design. The author also be thankful for his family and friends.

References

[1] https://www.javatpoint.com/ai-informed-search-algorithms visited at 3 May 2020, 20.00 WIB.

[2] Munir, Rinaldi. 2018. Slide Kuliah Graf Traversal IF2211 Strategi Algoritma. Bandung: Institut Teknologi Bandung. Diakses pada 2 May 2020, 18.00 WIB.

[3] Munir, Rinaldi. 2018. Slide Kuliah Route/Path Planning IF2211 Strategi Algoritma. Bandung: Institut Teknologi Bandung. Diakses pada 2 May 2020, 18.00 WIB.

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, 4 Mei 2020

Muhammad Daru Darmakusuma, 13518057