

Determining Lowest Risk Path in the game Slay The Spire using A* Algorithm

Ilyasa Salafi Putra Jamal - 13519023
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
E-mail (gmail): 13519023@std.stei.itb.ac.id

Abstract—Slay The Spire is a roguelike deckbuilder game developed by Mega Crit Games. As a roguelike, the game uses random generation to craft its map. The game will randomly generate several interconnected paths for each map, each with randomly selected encounters. Any player must choose which path to go in order to progress the game. A pathfinding algorithm can be used here to help evaluate all possible paths. This paper explores the idea by using A* Algorithm to assist in finding lowest risk path possible.

Keywords—Path; Encounter; Risk; Heuristic; Game

I. INTRODUCTION

Slay The Spire is commonly set as an example of a modern roguelike game. Roguelike games are usually described as games with permadeath mechanic, procedurally or randomly generated play area, and some kind of character progression system, terms that describe Slay The Spire well. The term 'roguelike' originates from a game developed in 1980 called *Rogue*. The game had ASCII interface and uses plenty of randomization in its gameplay mechanics, a notable randomization used was the randomized levels generated for each playthrough. The game then quickly gain popularity in the UNIX community.



Figure 1. *Rogue*(1980) (Source: <https://web.archive.org/web/20160919020229/http://insight.ieeeusa.org/insight/content/views/371703>)

Games similar to *Rogue* then starts to appear, further spreading the popularity of, at the time, a new genre of game, now known as 'roguelike'. An attempt to define the genre is then made. During the International Roguelike Development Conference held in 2008 in Berlin, a definition has been made for the genre, now known as the 'Berlin Interpretation'.

The Berlin Interpretation lists several factors that determines whether or not a game is a roguelike game. Some notable factors are random environment generation, permadeath, high complexity/difficulty/challenge, and resource management.

In the last few years, there has been an increase of interest in roguelike games, or roguelikes. Several newer roguelike games have gained high amount of popularity and success, games such as Hades, Darkest Dungeon, Risk of Rain, Monster Train, and Slay The Spire.

As a roguelike game, Slay The Spire also uses random generation for creating the game's map. During map generation, Slay The Spire randomly generates several interconnected paths that leads to a randomly selected boss. The paths are also populated with an assortment of encounters, each with plenty of variations that are randomly chosen. Any player is required to go through these random paths in order to progress and ultimately beat the game.

Unfortunately, knowing which path is the optimal path is a very difficult task. The interconnected paths mean there can be hundreds of possible paths to go through. This is not helped by the fact that the map is always randomly generated for each run, making learning to spot the most optimal path an even harder task.

Fortunately, paths on the game can still be coldly analyzed by pathfinding algorithms. While in most cases it will not provide the most optimal path, pathfinding algorithms can still assist player in determining the optimal path by highlighting several notable paths, such as the lowest and highest risk paths. This paper explores that particular idea by utilizing the currently best pathfinding algorithm, the A* Algorithm, to find the lowest risk path on a particular map from the game.

II. THEORETICAL FOUNDATIONS

A. Graph

A graph is defined as a tuple of two sets, a non-empty set of vertices and a set of edges. Graphs are used to represent the discrete objects and the relation between said objects. The objects are represented as the vertices, while the edges represent the relation between the objects. Mathematically, it is

usually represented as $G = (V, E)$, where G is the graph itself, V is the non-empty set of vertices and E is the set of edges.

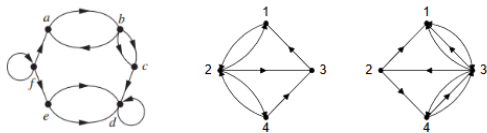


Figure 2. Directed graph (Source: <https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2020-2021/Graf-2020-Bagian1.pdf>)

This paper uses a type of graph called directed graph or digraph. In this type of graph, every edges is given a directional orientation. This paper also uses weighted graph concept. Every edges in a weighted graph is given a numerical value that determines its priority or ‘weight’.

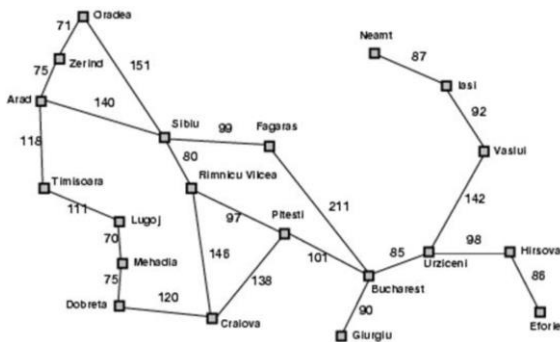


Figure 3. Weighted graph (Source: <https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Route-Planning-Bagian1-2021.pdf>)

B. Breadth First Search

Breadth First Search (from this point on will be referred as BFS) is a type of uninformed or blind search algorithm, meaning no information about the goal is needed for this algorithm to function. BFS searches a path in a graph to the goal by expanding search or visit from a node to its neighboring nodes. It will then continue expanding in the same manner for each neighbor nodes until it reaches the goal node, or all nodes has been searched or visited. It is a relatively simple algorithm that is capable of determining the least steps to reach the goal. However, this algorithm considers that steps = cost and does not consider differing weight or cost for visiting each nodes.

C. Uniform Cost Search / Dijkstra’s Algorithm

Dijkstra’s Algorithm or Uniform Cost Search (from this point on will be referred as UCS) is a type of uninformed search similar to BFS. This algorithm expands BFS by prioritizing which node to search or visit based on their weight or cost, either smallest or biggest first. Other than that, it works similarly to BFS. It is capable of determining the least (or most) weight or cost to reach the goal. UCS, however, still blindly searches every node and thus is prone to wasted searches due to not having information about the goal.

D. Greedy Best First Search

Greedy Best First Search (from this point on will be referred as GBFS) is a type of informed search algorithm, meaning it needed information on the goal and searches paths based on that information. The information provided is usually in the form of heuristics relevant to the search. The value of an admissible heuristic must always conform to this equation:

$$h(n) \leq h^*(n) \quad (1)$$

Where $h(n)$ is the heuristic cost from n to goal and $h^*(n)$ is the true cost from n to goal. This means admissible heuristics never overestimate the cost to reach the goal.

GBFS works by expanding search or visit to neighboring nodes that appears to be closest to the goal based on the heuristics given. Because of that, GBFS is faster and more efficient compared to UCS, as it only searches ‘promising’ nodes. However, GBFS is prone to local plateau and dead-ends due to the nature of heuristics. When that happens, GBFS will need backtrack to other ‘promising’ nodes, thus yielding sub-optimal paths.

E. A* Algorithm

A* Algorithm (read: A star) is a type of informed search algorithm similar to GBFS. This algorithm combines the accuracy of UCS and the heuristics from GBFS to consistently find optimal paths with a more reasonable time and efficiency. A* algorithm uses the following evaluation function to determine which node to expand:

$$f(n) = g(n) + h(n) \quad (2)$$

Where $f(n)$ is the estimated total cost of the path through n to goal, $g(n)$ is the total cost taken so far to reach n , and $h(n)$ is the estimated cost from n to goal.

The $g(n)$ part of the function is similar to how UCS evaluates which node to expand to, while $h(n)$ is similar to how GBFS evaluates its expansion with heuristics. After evaluating, the algorithm then expands its search to the node with the most optimal $f(n)$ value. This usually means the smallest value available.

F. Slay The Spire



Figure 4. Slay The Spire (Source: https://store.steampowered.com/app/646570/Slay_the_Spire/)

Slay The Spire is a roguelike deckbuilder video game developed by MegaCrit. The game entered early access in late

2017 and was officially released in January 2019. As of this writing, Slay The Spire is available for Windows, Linux, macOS, Nintendo Switch, PS4, Xbox One, iOS, and Android. In Slay The Spire, the player plays as one of the four different characters. The player then go through a randomly generated map filled with enemies, resting places, and random events.



Figure 5. Slay The Spire combat (Source: https://store.steampowered.com/app/646570/Slay_the_Spire/)

Slay The Spire offers combat with a turn-based system, similar to that of traditional RPGs, with a key difference. The difference lies in what actions the player can take during his turn. Actions available in a turn is dictated through cards drawn from a shuffled deck. The player can alter the deck through various encounters on the map. Each action also has an energy cost and the player only has a limited amount of energy each turn.

The game is divided into three or four acts. Each act has its own randomly generated map with several possible paths to the boss of the act. The boss of each act is randomly selected from the three possible choices for each act. Each act also contains its own pool of enemies, elites, and random events.



Figure 6. Slay The Spire's randomly generated map (Source: Screenshot taken from the game)

In every act, the player can choose from several randomly generated paths on the map. Each path have a certain encounter associated with it, such as entering combat with regular or elite enemy, opening treasure chests, trading with a merchant, etc. Each encounter has its own risks and rewards. For example, fighting an elite is a very risky move, but is also greatly rewarded. The success of a run often depends on managing risks like this. This paper focuses on this aspect of the game.

III. DETERMINING LOWEST RISK PATHS

Before you begin to format your paper, first write and save the content as a separate text file. Keep your text and graphic files separate until after the text has been formatted and styled. Do not use hard tabs, and limit use of hard returns to only one return at the end of a paragraph. Do not add any kind of pagination anywhere in the paper. Do not number text heads-the template will do that for you.

Finally, complete content and organizational editing before formatting. Please take note of the following items when proofreading spelling and grammar:

A. Types of Encounters and Their Associated Risks



Figure 7. Encounters legend (Source: Screenshot taken from the game)

Including boss encounter, there are seven types of encounters possible, eight if the player has completed the game atleast once with the first three characters. Each encounter has its own risk and reward associated with it.

- Boss



Figure 8. Boss encounters (Source: Screenshot taken from the game)

Boss encounters are always encountered at the end of an act. The symbol shown is determined by the type of boss selected by the game. For example on the left is a symbol for Hexaghost while on the right is a symbol for Slime Boss. Defeating the bosses of each act is necessary to progress through the game. As such, these encounters are **unavoidable risks** and serve as the 'goal' for the algorithm.

- Rest Site



Figure 9. Rest site encounter (Source: Screenshot taken from the game)

In rest site encounter, the player can either rest to regain some lost hp, or smith to upgrade a card from the deck. There can also be several more actions that can be taken, such as recall, dig, toke (or smoke) and lift weights. The recall action is only available after completing the game atleast once with the first three characters, while other additional actions are gained through the rewards gained during the run. Because its purely beneficial nature, this encounter has **little to no risk** associated with it.

- Treasure Room



Figure 10. Treasure room encounter (Source: Screenshot taken from the game)

Treasure room encounters reward player with a randomly selected artifact which grants various passive effects. The artifact can have a varying degree of effect on the run, from having almost no effect to game-breakingly strong. While the reward can vary, this encounter is still purely beneficial to the player and posses **little to no risk**.

- Merchant Shop



Figure 11. Merchant shop encounter (Source: Screenshot taken from the game)

In the merchant shop encounter, the player can trade currency gained from other encounters to gain a select choice of rewards. There is always a risk that the currency the player has is not sufficient to trade with anything meaningful. However, the risk it has is still **very miniscule** and in most cases, the encounter is still beneficial to the player.

- Enemy



Figure 12. Enemy encounter (Source: Screenshot taken from the game)

This encounter is the most common encounter of the game. In this encounter, the player is put into combat against randomly selected regular enemy of the act. If the player won the combat, he is granted a moderate reward. Because of this, there is **a risk** of dying and failing the run. However, the difficulty, and ultimately the risk, is not as high as fighting an elite enemy.

- Elite



Figure 13. Elite encounter (Source: Screenshot taken from the game)

In the elite encounter, the player is put into combat against a randomly selected elite enemy of the act. While granting greater rewards compared to regular enemies, elite enemies are much tougher compared to their regular counterpart. They also have their own quirks, making this encounter significantly more difficult compared to regular enemy encounters, thus making this encounter **highly risky**.

- Unknown Room



Figure 14. Unknown encounter (Source: Screenshot taken from the game)

In unknown room encounters, the player is presented with a randomly selected event from the current act events pool. These random events can have a wide variety of effects, risks, and rewards. Several events are purely beneficial events or events with great reward but little risk. However, there are also events that throws the player in a very risky situation, such as fighting two elite enemies back to back, or even throws the player into a high risk situation with little reward. Because of its high randomness, going into this encounter is a **high risk** gambit with potentially great reward.

- Buffed Elite



Figure 15. Buffed elite encounter (Source: Screenshot taken from the game)

The buffed elite encounter only starts to appear after the player has completed the game atleast once with the first three characters. In this encounter, the player will be put into combat against a buffed elite enemy. The elite enemy will receive several buffs during combat, making this encounter much harder even compared to just elite encounter. However, the reward gained on this encounter is necessary to access the fourth hidden act. Nonetheless this encounter still posses a **very high risk** as it is simply a harder elite encounter.

Based on explanation above, each encounter can be classified as the following:

Encounter	Risk Level
Boss	Unavoidable
Rest	Low
Treasure	Low
Merchant	Low
Enemy	Medium
Elite	High
Unknown	High
Buffed Elite	Very High

Figure 16. Encounter risk levels (Source: writer's repertoire)

B. Graph Representation and Heuristics

The objective of the game is to reach the current act boss and defeating it. The map of each act is generated randomly so that each map can have several possible paths towards the boss. Each paths has different encounters, which offers differing risks and rewards. The player is expected to manage the risk and reward balance by choosing appropriate paths. Too high of a risk, and the player character will die, thus failing the run. This paper focuses on determining the paths with the least risk.

The map of each act can be represented as a directed weighted graph. The edges are directed because the player cannot backtrack to encounters on the floor below the current floor. The weight each edge is determined by the risk associated with the encounter on the target node. Low risk encounter has 1 weight value, medium risk has 2, high risk has 3, and very high risk has 4. Unavoidable risk encounter has 0 weight value as the still paths have to go through it under any circumstances, making its weight irrelevant. Below is an example map from the game followed by its graph representation:

Figure 17. Slay The Spire map example (Source: Screenshot taken from the game)

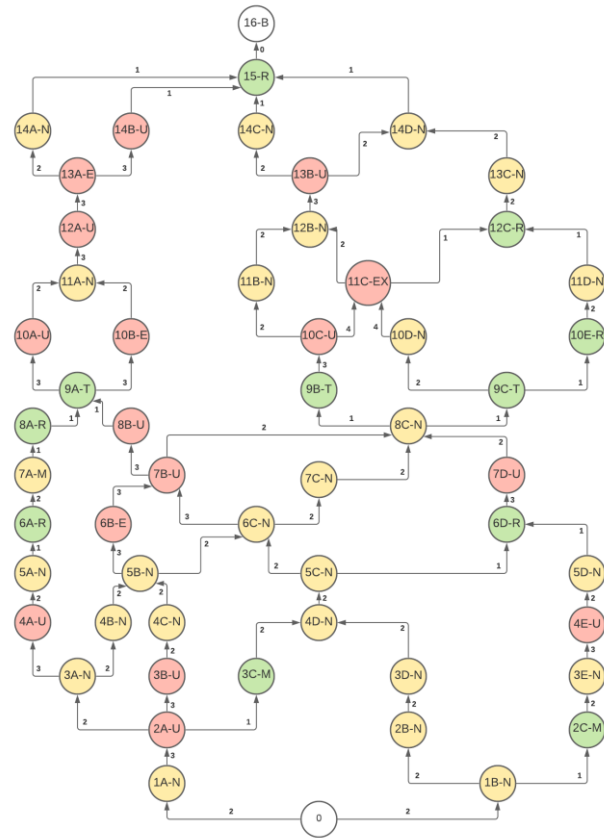
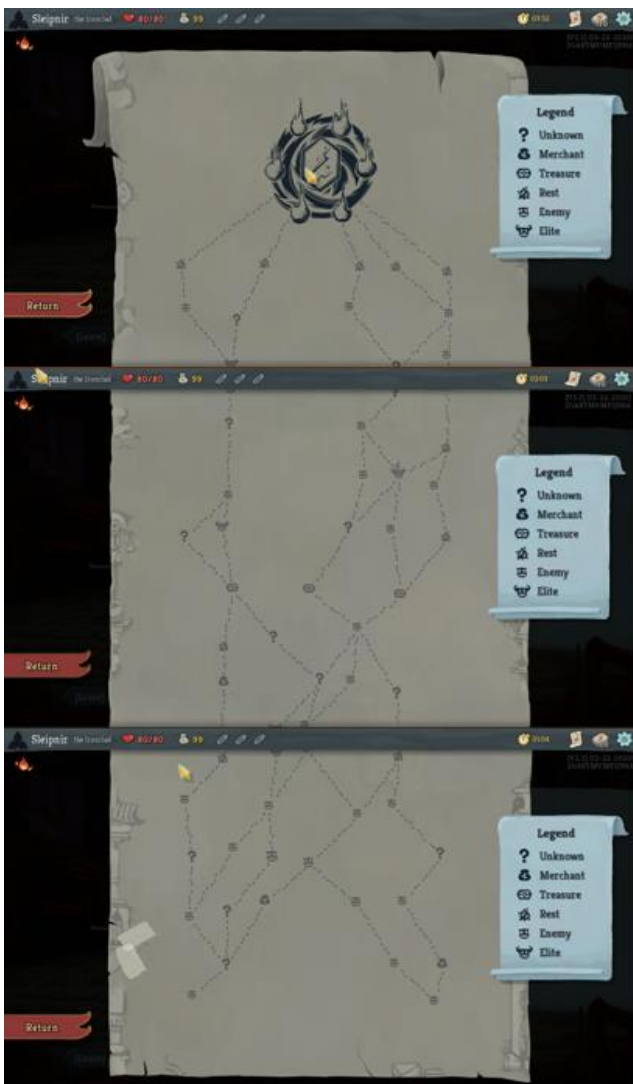


Figure 18. Graph representation of the example map (Source: writer's repertoire)



Legend	Encounter	Risk Level	Weight
B	Boss	Unavoidable	0
R	Rest	Low	1
T	Treasure	Low	1
M	Merchant	Low	1
N	Enemy	Medium	2
E	Elite	High	3
U	Unknown	High	3
EX	Buffed Elite	Very High	4

Figure 19. Graph legend (Source: writer's repertoire)

Node notation used on the graph:

$$XY-Z \quad (3)$$

Where X is the floor number of the encounter, Y is the encounter identifier on each floor, and Z is the encounter type.

Note that all encounters before the boss are always rest encounters and are all linked to the boss encounter, thus it can be simplified as in the graph representation. Also note that encounter 0 is used as a dummy node for coding purposes and the last two floors do not have Y value due to being the only encounter on the floor.

The A* algorithm is type of Informed Search. That means it needs information regarding its goal, usually in the form of an estimation or value called heuristic. The heuristics for this

calculation will be the amount of floor needed to be climbed to reach the boss. Taking from the example above, the map has sixteen floors and thus the heuristic at the first floor encounters will be fifteen, reduced to fourteen on second floor encounters, and so on until it reaches zero on the boss floor. The following is a table of the graph that includes the heuristic values:

Encounters					Floor	Heuristic
0					0	16
1A-N		18-N			1	15
2A-U		2B-N	2C-M		2	14
3A-N	3B-U	3C-M	3D-N	3E-N	3	13
4A-U	4B-N	4C-N	4D-N	4E-U	4	12
5A-N	5B-N		5C-N	5D-N	5	11
6A-R	6B-E	6C-N		6D-R	6	10
7A-M	7B-U		7C-N	7D-U	7	9
8A-R	8B-U	8C-N			8	8
9A-T		9B-T		9C-T	9	7
10A-U	10B-E	10C-U	10D-N	10E-R	10	6
11A-N	11B-N	11C-EX		11D-N	11	5
12A-U		12B-N		12C-R	12	4
13A-E		13B-U		13C-N	13	3
14A-N	14B-U	14C-N	14D-N		14	2
15-R					15	1
16-B					16	0

Figure 20. Table representation of the graph (Source: writer's repertoire)

Note that the data on the spreadsheet is in reversed order, start to goal instead of goal to start as with the previous graph. This is done to ease data input process in the algorithm.

The heuristics here is used to enforce the directed nature of the graph. It also enforces the game's rule as the player cannot backtrack to an encounter on the previous floor.

C. The Algorithm

A* algorithm will be used to determine the paths with the lowest risk. As explained previously, A* search algorithm uses the following evaluation function:

$$f(n) = g(n) + h(n) \quad (4)$$

Using it for the paper's purpose, the 'cost' for the function would be the risk taken on an encounter. That means $g(n)$ would be the sum of risk taken so far to reach encounter n . For $h(n)$, we can use the heuristics determined previously for each encounter on each floor.

The following is a snippet of the algorithm implemented in Python 3 for this particular purpose:

```
neighbors = graph.get(currNode.name)
for key, value in neighbors.items():
    # Create the neighbor node
    neighbor = Node(key, currNode)
    # If already visited, skip current loop
    if(neighbor in visited):
        continue
    # A* evaluation function, f(n) = g(n) + h(n)
    neighbor.g = currNode.g + graph.get(currNode.name, neighbor.name)
    neighbor.h = heuristics.get(neighbor.name)
    neighbor.f = neighbor.g + neighbor.h
    # If neighbor is in expand list and has lower f(n)
    if(expandable(neighbor) == True):
        expand.append(neighbor) # Add neighbor to expand list
```

Figure 21. Snippet of A* implementation in Python 3 (Source: writer's repertoire)

The implementation above follows an example from <https://www.annytab.com/a-star-search-algorithm-in-python/> with several modifications to fit the paper's purposes.

The code above will be ran for every node expansion the algorithm does. The algorithm will expand its search according to the sorted content of expand list. The list itself is sorted from the lowest risk to highest. The expansion will continue until the goal has been reached or the expand list is empty, meaning there is no possible path.

D. Source Code and Testing

The source code used for testing, along with the test files, can be accessed from <https://github.com/SleipnirMk1/Astar-SlayTheSpire>. In the code, Graph class is used to represent the graph data representation from the game map, while the Node class is used to represent encounters.

```
# Directed Graph Representation
class Graph: ...

# Node class for the graph
class Node: ...
```

Figure 22. Graph and Node class (Source: writer's repertoire)

The program receives map data input through a customized txt file, below is the content txt file for the example map above:

```
TestMap1.txt - Notepad
File Edit Format View Help
Floors
16
Encounters
0-ST
1A-N 18-N
2A-U 2B-N 2C-M
3A-N 3B-U 3C-M 3D-N 3E-N
4A-U 4B-N 4C-N 4D-N 4E-U
5A-N 5B-N 5C-N 5D-N
6A-R 6B-E 6C-N 6D-R
7A-M 7B-U 7C-N 7D-U
8A-R 8B-U 8C-N
9A-T 9B-T 9C-T
10A-U 10B-E 10C-U 10D-N 10E-R
11A-N 11B-N 11C-EX 11D-N
12A-U 12B-N 12C-R
13A-E 13B-U 13C-N
14A-N 14B-U 14C-N 14D-N
15-R
16-B
Links
0,1A,18
1A,2A 1B,2B,2C
2A,3A,3B,3C 2B,3D 2C,3E
3A,4A,4B 3B,4C 3C,4D 3D,4D 3E,4E
4A,5A 4B,5B 4C,5B 4D,5C 4E,5D
5A,6A 5B,6B,6C 5C,6C,6D 5D,6D
6A,7A 6B,7B 6C,7B,7C 6D,7D
7A,8A,8B 7B,8C 7C,8C 7D,8C
8A,9A 8B,9A 8C,9B,9C
9A,10A,10B 9B,10C 9C,10D,10E
10A,11A 10B,11A 10C,11B,11C 10D,11C 10E,11D
11A,12A 11B,12B 11C,12B,12C 11D,12C
12A,13A 12B,13B 12C,13C
13A,14A,14B 13B,14C,14D 13C,14D
14A,15 14B,15 14C,15 14D,15
15,16
```

Figure 23. Txt file for example map (Source: writer's repertoire)

The first part of the text file is the floor amount, it is used to determine the heuristics. The next part is the encounters. Here, all encounters from the map are listed per floor with space or tab as separator for encounters on the same floor. 0-ST encounter is used as a dummy node for processing purposes. The last part is the links. Here, all links from certain encounters are listed per floor with space or tab as separator for different encounter source.

The txt file above is processed by readFile function which also assigns the risk levels for each encounter. A graph from

the file is then built to represent the map from the game by the buildGraph function.

```
# Read custom txt file
def readFile(filename): ...

# Link the encounters
def buildGraph(encounterRisk, edges): ...
```

Figure 24. readFile and buildGraph function (Source: writer's repertoire)

The entire program is ran with the main function, where it serves as an interface (Command Line Interface) for inputting files and generating paths from a certain starting node, not just from the bottom floor, based on inputs from the user.

Below is a test run done with the previous example map from the game:

```
TERMINAL PROBLEMS OUTPUT DEBUG CONSOLE 1: Python
Try the new cross-platform PowerShell https://aka.ms/pscore6

PS C:\Users\Lenovo> & C:/Users/Lenovo/AppData/Local/Programs/Python/Python38-32/python
.exe c:/Tugas/Stima/Makalah/Astar.py
Enter custom txt file path:
C:\Tugas\Stima\Makalah\TestMap1.txt
Search lowest risk path from where? (Enter '0' if from the beginning)
0
Path with the lowest risk:
{'0': 0, '1B': 2, '2C': 3, '3E': 5, '4E': 8, '5D': 10, '6D': 11, '7D': 14, '8C': 16,
'9C': 17, '10E': 18, '11D': 20, '12C': 21, '13C': 23, '14D': 25, '15': 26, '16': 26}
Total risk value taken: 26
Enter 'exit' when you are done
Search lowest risk path from where? (Enter '0' if from the beginning)
█
```

Figure 25. Test run 1 (Source: writer's repertoire)

The program yields the following path from the graph above:

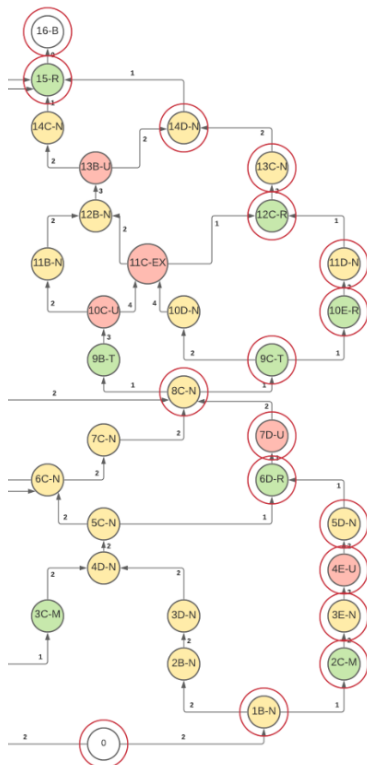


Figure 26. Test first path (Source: writer's repertoire)

As visible above, the program avoids the cluster of elites and unknown rooms on the left of the graph. It also avoids encounter with the buffed elite and yields a path with cumulative risk value of 26.

The program can also receive input where the starting node is not from node 0, as shown below:

```
Search lowest risk path from where? (Enter '0' if from the beginning)
9B
Path with the lowest risk:
{'9B': 0, '10C': 3, '11B': 5, '12B': 7, '13B': 10, '14C': 12, '15': 13, '16': 13}
Total risk value taken: 13
Enter 'exit' when you are done
Search lowest risk path from where? (Enter '0' if from the beginning)
█
```

Figure 27. Test run 2 (Source: writer's repertoire)

The program yields the following path from the graph above:

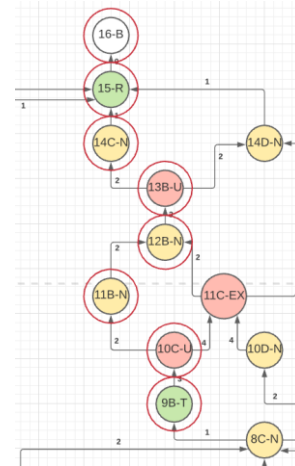


Figure 28. Test second path (Source: writer's repertoire)

As visible above, the algorithm starts its search from node 9B as instructed. The algorithm does not just direct the path to the previous low risk path, it still searches path that is the lowest risk from the new starting node. Because of that, the resulting path is visibly different from the previous path, as returning to the previous path means encountering a buffed elite, a very high risk encounter.

A full test run can be seen in the youtube video linked on the following section. As for the source code, as stated before, it can be accessed, along with the test file from <https://github.com/SleipnirMk1/Astar-SlayTheSpire>.

IV. CONCLUSION

The A* algorithm's performance and accuracy has been proven countless times before, as it can consistently provide the most optimal pathing while maintaining time and processing complexity. This paper is no different, it proved that A* Algorithm is still very reliable when it comes to pathfinding, as it is always capable of finding paths with the lowest possible risk in Slay The Spire. With some tweaking, it can also find paths with highest risk too.

However, winning in Slay The Spire is not just about taking the least, or most, risk. Taking less risky paths can potentially limit the character's power growth from the rewards gained, and thus dooming the run in the long term.

Slay The Spire is about risk and reward management, the player must balance between the two aspects in order to both reach the endgame and build-up enough power to survive the endgame.

One advice the writer can give as an experienced player, is that the paths provided by this algorithm should not be treated as a strict guideline, instead it is advised to mix-up the path by taking risky paths near the safer paths. That way the high rewards from high risk encounters can still be gained while having a safety net to fall into when the risk becomes too much to handle. For example, in figure 25, a possible mix-up would be to pick encounter 10D and 11C and then fall back into encounter 12C instead of following the path suggested by the algorithm.

Perhaps it is possible to create an algorithm where it also considers the rewards, not just the risks, on each paths, and while at it, also considers the location of certain types of encounter. For example, an elite encounter on higher floors is less risky compared to elite encounter on lower floors. Unfortunately, such sophisticated algorithm is not within the current capability of the writer.

In conclusion, the A* algorithm is proven to be accurate when determining the lowest risk path possible during a run in Slay The Spire. Further development for the algorithm may include reward assesment and evaluating certain encounter's location to determine, not the lowest risk path, but the most optimal path.

VIDEO LINK AT YOUTUBE

<https://youtu.be/DVPd3uV949w>

GITHUB REPOSITORY

<https://github.com/SleipnirMk1/Astar-SlayTheSpire>

ACKNOWLEDGEMENT

In this section I would like to express my gratitude towards God, my friends, and my family for supporting me all the way up to this point of my life where I finish this paper. I would also like to express my gratitude to Ir. Rila Mandala, M.Eng.,Ph.D. as my lecturer and also to the entire ITB informatics study lecturers. It is thanks to their efforts that I am able to finish this paper and understand the various algorithm strategies commonly used in programming. These strategies are very fascinating to learn and I am certain it will be very useful for me in the future. Lastly I would like to apologize for any mistakes that occurred in the process of making this paper. I

hope this paper can help and inspire other people to learn, think rationally, and perhaps spark interest in the game I like so much.

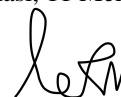
REFERENCES

- [1] Munir, Rinaldi, "Graf (Bag.1)", <https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2020-2021/Graf-2020-Bagian1.pdf>, 2020, Accessed 8 May 2021.
- [2] Munir, Rinaldi, "Penentuan Rute (Route/Path Planning) Bagian 1: BFS, DFS, UCS, Greedy Best First Search", <https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Route-Planning-Bagian1-2021.pdf>, 2021, Accessed 8 May 2021.
- [3] Munir, Rinaldi, "Penentuan Rute (Route/Path Planning) Bagian 2: Algoritma A*", <https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Route-Planning-Bagian2-2021.pdf>, 2021, Accessed 8 May 2021.
- [4] Red Blob Games, "Amit's A* Pages", <http://theory.stanford.edu/~amitp/GameProgramming/>, 1997, Accessed 9 May 2021.
- [5] Administrator, "A* Search Algorithm in Python", <https://www.annytab.com/a-star-search-algorithm-in-python/>, 22 Jan. 2020, Accessed 9 May 2021.
- [6] Red Blob Games, "Introduction to the A* Algorithm", <https://www.redblobgames.com/pathfinding/a-star/introduction.html>, 26 May 2014, Accessed 10 May 2021.
- [7] Mega Crit Games, "Slay the Spire", https://store.steampowered.com/app/646570/Slay_the_Spire/, 23 Jan. 2019, Accessed 10 May 2021.
- [8] Brewer, Nathan. "Going Rogue: A Brief History of the Computerized Dungeon Crawl", 7 Jul. 2016, <https://web.archive.org/web/20160919020229/http://insight.ieeeusa.org/insight/content/views/371703>, Accessed 11 May 2021
- [9] International Roguelike Development Conference, "Berlin Interpretation", 2008, http://www.roguebasin.com/index.php?title=Berlin_Interpretation, Accessed 11 May 2021

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.

Bekasi, 11 Mei 2021



Ilyasa Salafi Putra Jamal dan 13519023