Application of Graph Theory in Route Selection for the Game *Slay the Spire*

Jaya Mangalo Soegeng Rahardjo - 13520015[1]
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
[1]author@itb.ac.id

***Abstract*—Slay the Spire is a video game where the player must traverse through a treacherous dungeon filled with enemies and bosses. The player can move forward in the dungeon by picking routes that constantly branch and merge with each other, forming a directed graph. A program can be used for pathfinding utilizing Dijkstra's algorithm to find a route that is optimal for the player to follow. This paper discusses the implementation of a basic pathfinding program and the various ways the program can be upgraded.**

***Keywords*—Dijkstra's Algorithm, Graph Theory, Slay the Spire, Video Game**

I. Introduction

Slay the Spire is a rogue-like, turn-based, deck-building, action video game developed and published by MegaCrit Games. The player takes control of one of 4 characters with unique special abilities and decks and traverse through the dungeon named "The Spire". To win the game, the player must traverse through three 'acts' or sections of the spire. Each section consists of 16-17 floors which can be encounters or events and a boss floor located on the very last floor.

Combat-wise, the game does it in a turn-based system with cards that you have obtained in your deck, you will draw 5 cards from your deck every turn and gain 3 energy which you can spend to use the cards that you drew either to attack the enemy, defend yourself, cast buffs and debuffs, and other actions. During the enemy's turn, they can also do similar things such as attack the player, defend themselves, cast buffs and debuffs and so on. The enemy's next action can be seen during the player's turn so that they can prepare against it. The players have health points (HP) which are expected to be reduced after each encounter and can be increased in various ways, if the player's HP reaches 0, then the game is over.

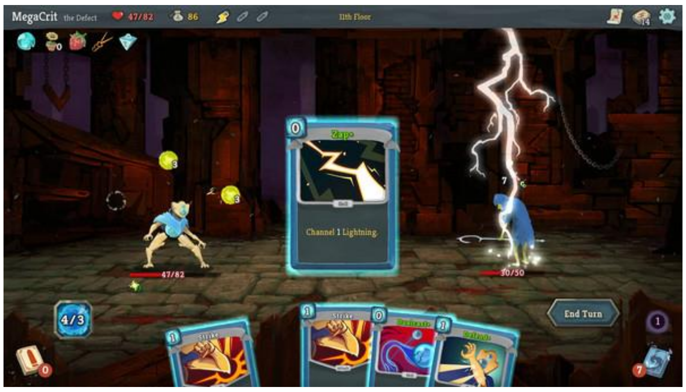

Figure 1, Screenshot of Slay the Spire's combat gameplay (store.steampowered.com/app/646570/Slay_the_Spire/ accessed on December 11, 2021)

Exploration-wise, the player will be given a map filled that shows routes which the player can take to go through the floors. In the map, each floor will be given a symbol stating what you will encounter where you to choose said floor. For example, a monster symbol will signify an enemy encounter, a monster with horns symbol means a mini-boss encounter, a money sack is a shop where you can spend gold to buy various items, and so on.

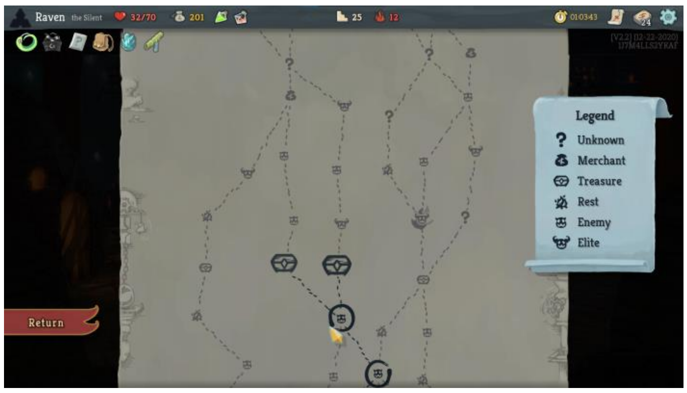

Figure 2. Screenshot of Slay the Spire's navigation map

By defeating enemies and going through the dungeon, you will gain cards that you can add to your deck, upgrade cards, or gain powerful relics and potions which will give you unique and powerful buffs. The game follows a risk-reward mechanic where stronger enemies will give better rewards, you may follow a risk-free route but you may not gain the same rewards you would for following a risky route. However following a too risky route might mean you getting overwhelmed and getting a game over via dying. Therefore, a flexible strategy with a good balanced of risk and safety in choosing which routes to go is a necessary to win the game.

In an actual game, the pathfinding by the player must be dynamic based on said player's current situation, "am I strong enough to take risks?", "can I defeat the enemies encountered in this route?" However, for streamlining purposes, we will apply a static pathfinding algorithm where it will calculate the optimal route at the very start of the game and not during the middle.

II. Theoretical Foundation

A. Graph Theory

Graphs are a type of visualizations often used to represent discrete objects and their relations together. Graphs are structures made of vertices and edges, where a vertices are nodes/objects and edges are the relations that connect these vertices together.

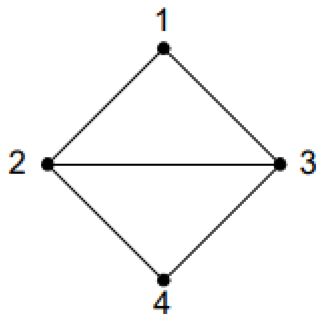

Figure 3. A representation of a graph with 4 vertices and 5 edges (https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2020-2021/Graf-2020-Bagian1.pdf, accessed on December 13 2021)

A formal definition of a graph would be defined as G = (V,E), with V representing the sets of all vertices in G and E representing the sets of all the edges in G.

B. Types of Graphs

Based on their complexity, graphs are divided into two types: Simple graphs and complex graphs. Simple graphs are graphs that does not contain any duplicate edges or pseudo edges, graphs that contain those are called complex graphs with a graph containing duplicate edges called a multigraph and a graph containing pseudo edges called a pseudo-graph.

Based on their directions of their edges, graphs can also be divided into two types: undirected and directed. Undirected graphs means that a graphs edges has no directions while a directed graph's edges does.

There is also another type of graph which is differentiated based on the edges, a weighted graph is a graph where each edge is given a numerical value. An example is a graph of roads between cities, if the edge connecting city A and B is given a weight, it may mean the distance between city A and city B.

C. Dijkstra's Algorithm

Djiksta's Algorithm is an algorithm usually used in finding the shortest path for a weighted graph. Founded by Edsger W. Dijkstra in 1956 to find the shortest paths between nodes in a graph. Dijkstra's Algorithm has then had many variations with the most common ones are to find the shortest path between a singular point and all other points within that graph. Another common variant is an algorithm that will calculate the shortest path from a point in a graph to another point in a graph.

The first variant is commonly visualized with a table containing all the important details such as the name of the node, the shortest distance so far, the current shortest path to get said node and if the node has been checked or not. This table will be filled with important information as the algorithm runs and it will stop as soon as all nodes have been checked.

The second variant is commonly visualized with a priority queue. The algorithm will constantly fill the queue with data on which node to process. The queue is ordered by the weight with a lower weight being more prioritized. The algorithm will constantly add to the queue and update it until we reach or process the end goal.

Example and visualization of Djikstra's Algorithm second variant:

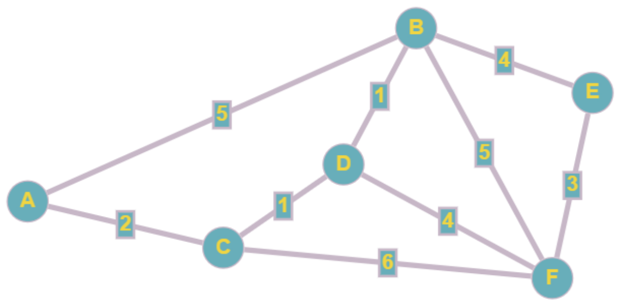

Figure 4. A graph example of a Dijkstra's Algorithm

Say we want to find the shortest path from node A to F, the algorithm would do it step by step.

1. Insert Node A into the priority queue with the weight 0.

Node	Weight	Route
A	0	A

Table I. Priority Queue after Node A is inserted.

2. Process the first node in the queue, since only A is in the queue, Node A will be processed first. Processing Node A would mean inserting all the nodes connected to A which is Node B and Node C with weight 5 and 2 respectively into our queue.

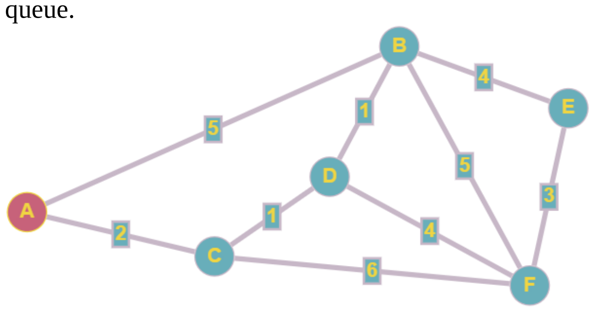

Figure 5. Visualization of a Dijkstra's Algorithm processing node A.

Node	Weight	Route
C	2	A-C
B	5	A-B

Table II. Priority Queue after Node A is processed.

3. We will again process the first node in our queue, this time its C because its weight is lower than B. Processing Node B would mean inserting (or updating if possible) all the nodes connected to B. We will insert Node D and Node F. Node D will have the weight 3 (2+1) and node F will have the weight 8 (2+6).

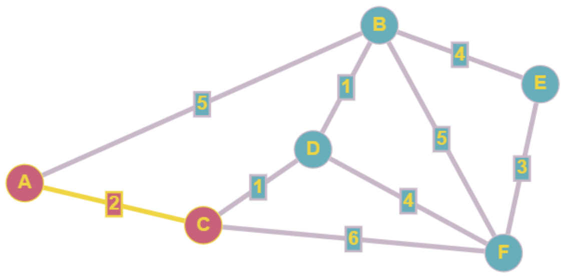

Figure 6. Visualization of a Dijkstra's Algorithm processing node C.

Node	Weight	Route
D	3	A-D
B	5	A-B
F	8	A-F

Table III. Priority Queue after Node C is processed.

4. We will then process Node D. There are two nodes connected to it which is Node B and F. Both of these are already in our queue so we check if we can update it. Using the A-C-D-B path would mean that B will have a weight of 4 and therefore shorter than using the previous A-B path. Because A-C-D-B is shorter, we will replace the Node B in our queue with the new one. We will repeat the same step with Node F causing F to have a weight of 7 (2+1+4) with the route A-C-D-F instead of A-C-F.

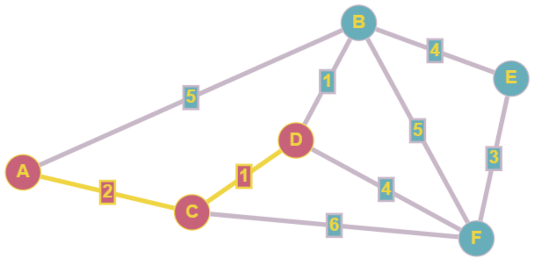

Figure 7. Visualization of a Dijkstra's Algorithm processing node D.

Node	Weight	Route
B	4	A-C-D-B
F	7	A-C-D-F

Table IV. Priority Queue after Node D is processed.

5. The next process will be Node B which will mean Node E with a weight of 8 will be inserted to our queue. The algorithm would then check Node F. The route through B to F (A-C-D-B-F) would have a weight of 9 which is bigger than its current weight (7), therefore we ignore it and not update node F in our queue.

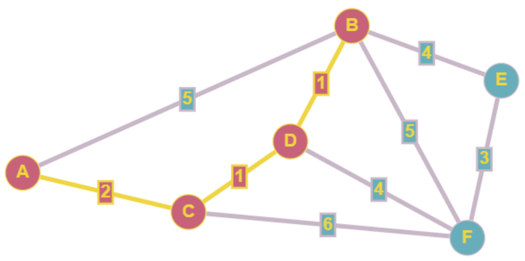

Figure 8. Visualization of a Dijkstra's Algorithm processing node B.

Node	Weight	Route
F	7	A-C-D-F
E	8	A-C-D-B-E

Table V. Priority Queue after Node B is processed.

6. Our next process is Node F, which is our target node. Because of this, the algorithm is finished and we found our shortest path from Node A to F, A-C-D-F with a weight of 7. We can now deallocate our queue as we will not be using it anymore, if you were to use the first variant, we will continue going until we reach Node E, which is unnecessary for our purpose.

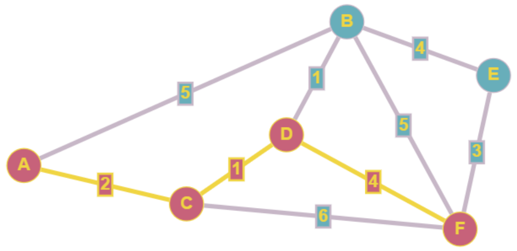

Figure 9. Visualization of a Dijkstra's Algorithm processing node F.

With this, we have successfully found the shortest route from A to F with our algorithm.

III. Slay the Spire

Slay the Spire has 6 main nodes not including the boss node and the buffed elite node, some nodes are much preferable than others due to the rewards and effects it gives by choosing said nodes. So choosing a route that will go through the most of these "rewarding" nodes are preferred.

A. Rest

Nodes that are symbolized by a campfire is a rest site. These rest sites are randomly generated throughout the dungeon and little of them are available per act, other than the one rest site guaranteed to be available right before the boss, every other rest site is random and generally 2-3 rest sites can be reached most of the time.

In a rest site, we can do several actions such as 'Sleep' to rest to heal our character, ‘Forge’ to upgrade our cards, or ‘Recall’ to gain the red key. If we have some certain effects in the game through relics, there are other options available such as Toke: remove a card from you deck, Lift: gain +1 strength, Dig: gain a random relic. Only one of these options can be used at each rest site.

At any point, a rest site is extremely valuable and should be prioritized at almost every scenario, the ability to increase your health points (HP) when low on it or the ability to improve your card’s strength in battle is extremely useful.

B. Merchants

Nodes that are symbolized by a bag with a dollar sign, is a merchant. Merchants are very rare and often only two of them are generated throughout the act.

In a merchant node, you can buy and only buy items and services using gold that you collected. You can either buy a card, buy a relic, buy a potion, or remove 1 card from your deck.

Merchant nodes are very useful for the most part even if there are some exceptions such as running out of gold making a merchant tile useless. Even then, the very few merchant nodes should be prioritized the same way or perhaps more than the rest site.

C. Treasure

Nodes that are symbolized by a chest is a treasure. A treasure node will only show up exactly once each route on the very middle of the dungeon. No matter what route you take, you will only ever be able to get one treasure each route.

A treasure node will give you the option to choose between a free random relic or the blue key and therefore very valuable, more valuable than a rest site or a merchant node. However, even though they are more valuable, as there is exactly only one treasure node accessible each route no matter which route the player chooses, there is no need to put them in our consideration while playing the game or when calculating with our algorithm.

D. Enemy

Nodes that are symbolized by a monster is an enemy encounter. These are for the most part randomly generated and extremely common and are guaranteed to at least be encountered at least a few times.

An enemy encounter will force you in a win or die situation with relatively weak enemies, you are expected to be hit with minor damage every encounter with against basic monsters. After each successful encounter, you will gain money and a card which you can add to your deck. You can rarely obtain potions sometimes.

Though an enemy encounter is essential to build your deck as it is the most common way to get cards so a few enemy encounters are recommended, you are generally guaranteed to reach the recommended encounters anyway so it is safe to assume that even if we try to actively avoid as many enemy encounters as we can, we will still get enough encounters. Therefore we should minimize the amount of basic enemy encounters we go through.

E. Unknown

Nodes that are symbolized by a question mark symbol is an unknown event or encounter. They are randomly generated and are very common throughout the spire second only to basic enemy encounters.

An unknown node can sometimes be a random event, an enemy encounter, a merchant, or a treasure, with that order of rarity. A random event being very common and treasure being very rare.

The random events that you encounter tend to be more beneficial to the player though some of them can be detrimental. Landing an enemy encounter would be unfortunate and the least wanted outcome, while getting a merchant would be fortunate and even more so for getting a treasure.

Though it is random, an unknown event node tends to be beneficial and should be more prioritized more than an enemy encounter tile.

F. Elite

The last type of nodes are elites, symbolized with a monster that has horns. They are randomly generated all over the spire and are slightly more common than a rest site.

These elite enemies are incredibly powerful and dangerous, you are expected to lose a good chunk of health fighting them or you may even die. There are 9 different elites or mini bosses that you can encounter in the spire. With an act having 3 different elites each. Each elites has different strengths and weaknesses and should be carefully considered. As a good matchup towards an elite can be a breeze while a bad one most likely will cause a game over.

Due to the dangers an elite possess, they certainly will give a good reward. After a successful elite encounter, the player are guaranteed to get a random relic, cards, gold and a chance to gain a potion. But as they are dangerous, they should be avoided and the player should only encounter a safe amount. The safe amount of elite encounter will vary greatly based on the player’s strength and nearby nodes such as other elites or a rest site.

G. Boss

A boss node is always located at the end of each act, with it being shown by a big symbol of the boss you are going to fight. Similar to elites, there are 9 possible bosses with 3 an act having 3 different bosses each. Furthermore, each boss will have different strengths and weaknesses similar to elites. Unlike elites however, you can tell exactly which boss you will fight therefore you can prepare a game plan and deck to beat that certain boss right at the beginning of the act.

H. Buffed Elite

A Buffed Elite node is a one of a kind node that can show up only once per act, unless you have already beaten it once on a previous act before. They are symbolized by an elite node but surrounded with fire effects.

These elites are stronger than a normal elites and should be proceeded with even more caution than a normal elite. If you were to win against a buffed elite, you will gain the same rewards as you would with a normal elite but you will also gain the green key.

The prioritization of the buffed elite is a special scenario and may needed to be calculated differently based on the scenario. On one hand, they are dangerous and should be avoided more than a normal elite but they are also necessary to gain the green key.

I. Act 4

Act 4 is an optional act to get the true ending of the game. To access it, you must defeat the act 3 boss and get 3 keys, the red key, the blue key, and the green key. These keys are gained from doing a recall in a rest site, choosing the green key over a relic, and defeating a buffed elite respectively.

Getting these keys are a challenge as the red key will force you to sacrifice a rest site to only recall over doing other useful actions such as resting or smithing, the blue key will force you to sacrifice a relic and the green key will force you to change your route to fight and defeat buffed elite.

If you were to get all these keys and defeat the act 3 boss, you will access act 4. There are only 4 nodes and a straight route in act 4, making it unneeded to be calculated. These nodes are a rest site, a shop, and a special act 4 elite and the final boss of the game, The Corrupt Heart. If you manage to defeat the heart, you will get the true ending of the game.

IV. APPLICATION OF DIJKSTRA'S ALGORITHM

A. Transformation of Slay the Spire's Map into a Graph

While Slay the Spire's map is a graph already, it will be needed to be modified for our program to run it. The first thing we need to do is transform it into a weighted-directed graph.

For the directional aspect, it is very simple. As we cannot go down a floor, all of our nodes will only be connecting towards the floor above it.

An extra thing we can do is connect some of the nodes together. A series of nodes that do not branch off and only lead to a single line can be connected and its values combined. This part is not necessary but it may increase our algorithm's efficiency later on.

For the weight aspect, we can mainly do it by assigning values of based on the desirability of a node. Say that due to the desirability of a rest site, we will assign it a low value such as 3 and we will assign an undesirable node, an elite node for example, a high value such as 7.

We are unlikely to get the optimal number of such values in the first try and we have to modify it as we go, similar how an AI would change the values of its code dynamically. At any point, let us try to set these values.

1. Rest Site
   Rest sites are quite valuable, only slightly behind nodes such as shops and treasure. Therefore we will apply a low number such as 3.
2. Merchants
   Merchants are probably the second most valuable node only behind a treasure node. So will assign an extremely low value such as 2.
3. Treasure
   The most valuable node is a treasure node, but as stated before, it is guaranteed and the changing the value will not affect the program at all. But we will assign it a low value like 1 for consistency.
4. Enemy
   Enemies are quite undesirable so we will assign a medium value such as 5.
5. Unknown
   Unknown tiles are a wild card and can be both good or bad, but as they are slightly better than they are bad, they will be assigned a number lower than an enemy such as 4.
6. Elite
   Elites are dangerous and should be avoided for the most part, therefore we will assign it a high weight like 7 or so.

B. A Run of Our Code

A case study with the full 15-16 floors in a typical Slay the Spire's act may be a bit too much so we will recreate a smaller scale set of floors for our case study.

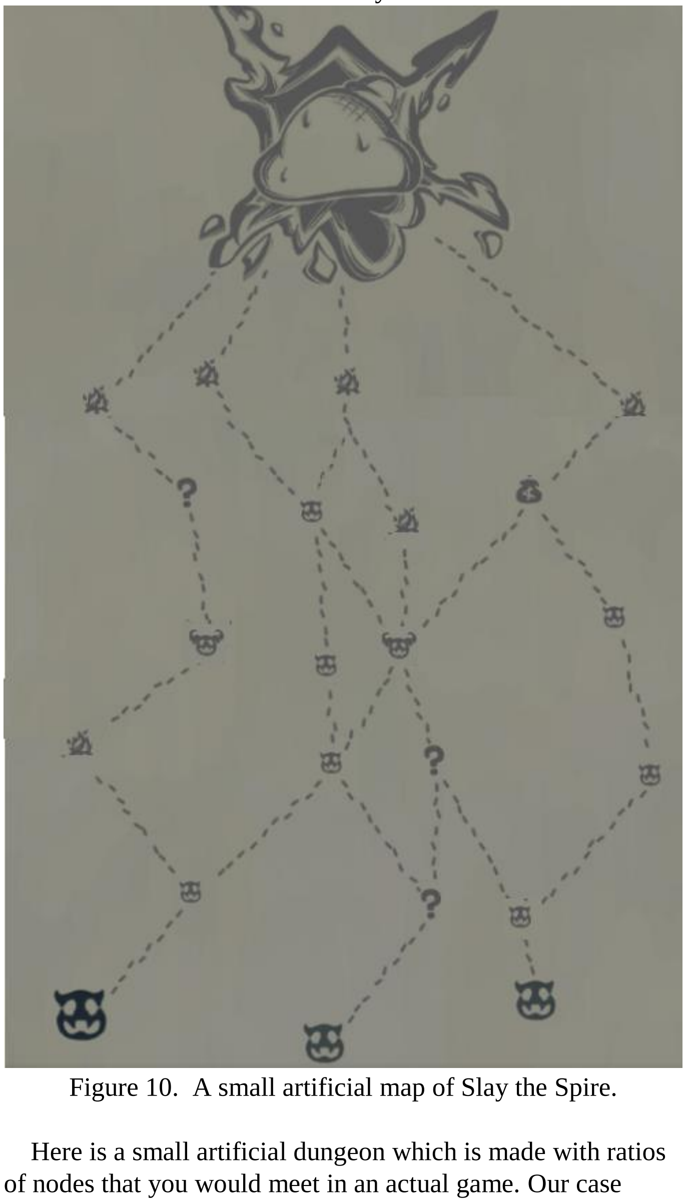
Figure 10. A small artificial map of Slay the Spire.

Here is a small artificial dungeon which is made with ratios of nodes that you would meet in an actual game. Our case study also does not have the guaranteed treasures in the middle as they are not very important to take into consideration.

As you would in an actual game, the game starts with you being able to pick any of the 3 bottom monster nodes, after you pick it, you will move up the chain normally and fight the boss:

The Slime Boss. An important thing to note is that we will consider the 4 rest sites before the boss the end state, not the boss itself.

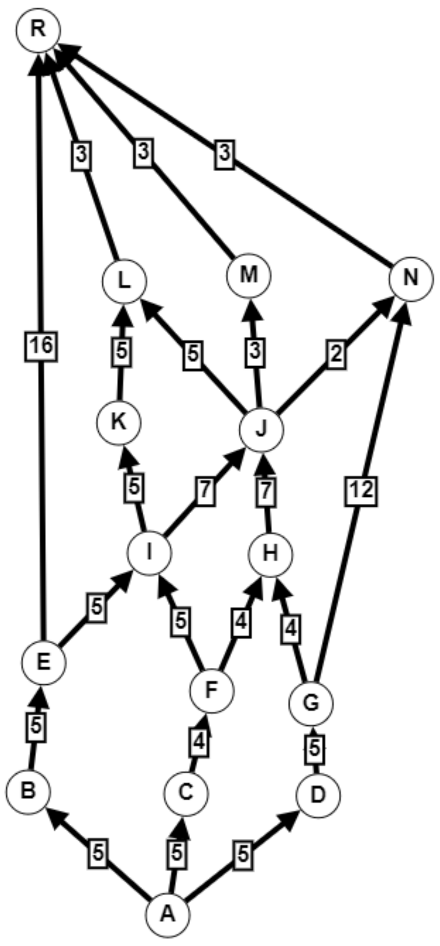
Figure 11. Transformation of Slay the Spire’s map into a graph

Here is our graph after we convert it, we created the start point A which can move to the first three monster nodes to our choosing, and the four rest sites have been made into one node as our final state. We have our graph ready to be run and calculated by our algorithm.

No	Node Processed	Inserted/Updated
1	A	B(5),C(5),D(5)
2	B	E(10)
3	C	F(9)
4	D	G(10)
5	F	H(13), I(14)
6	E	R(26)
7	G	N(22)
8	H	J(20)
9	I	K(19)
10	K	L(24)
11	J	M(23)
12	N	R(26) → R(25)
13	M	-
14	L	-
15	R	End State Reached, Deallocating Queue

Table V. List of processes in the priority queue.

After our algorithm runs, we found the optimal route to from start to finish. It goes through the path A-D-G-N-R with a weight of 25. A thing to note is that the path through A-D-G-H-J-M-R has the same weight of 25, but due to ADGNR being inserted into the queue first, it will be the one returned as the solution.

C. Case Studies

As our “prototype” program is a static algorithm that streamlined a lot of things from Slay the Spire, there certainly are room for improvements.

Listed below are case studies of scenarios that could happen in the game and other things that we can study and add onto our program.

1. Act 4

   To reach act 4, you must reach certain conditions, the main condition to consider is the mandatory buffed elite fight. So our algorithm must find the shortest route to the boss that also goes through the buffed elite node.

   This is fairly simple as all we need to do is to run our program once with the start node as the starting position of the player and the buffed elite as the target node. After we find the optimal route from start to elite. We run our program again, this time with the buffed elite as the start node and the 4 campfires as the target node.
2. Risk and Reward

   One of the main focus of the game is the risk and reward in the game. A route that goes through a risky route might be more rewarding (and necessary) causing the player to become stronger and having an easier time to defeat the boss.

   A major flaw of our program is that it will try to dodge elites as much as possible and choose the safer non-elite options. This is quite opposite of what we want actually. In general, players would want to kill 1 or 2 elites and ignore the rest as more than that might be a bit too risky.

   A way we can do this is too change the value of said elite based on how many elites we have encountered on the path so far. For example, say we have a route that goes through two elites, the program will assign the first elite to a low and desirable value such as 3, while the second elite while be designated a 5, and so on. These numbers are still more desirable or at least the same with your typical monster node. If we encounter a third elite node, the program will designate it a high value and try to avoid it.
3. Static Player Strength

   As the player goes through the spire, they are expected to grow stronger, so that an elite they encountered nearing the end of act would be much easier than an elite you encounter near the start of the spire. The same concepts goes for enemies and weirdly enough, merchants as well. You are more likely to be richer at the end of the spire (unless you spend it) than you are at the beginning of the game.

The way to implement this is a bit straightforward. Each node in the game are split already split into floors and you can assign each node a floor with a higher floor being better. An elite on a high floor will have a lower value than an elite located in the lower floors.

4. A Dynamic Program

Compared to our static program that calculates based on assumptions at the start of the game, a dynamic program that runs alongside that player sounds much better. A dynamic program can calculate the strength of our player, their HP, their gold so that they can make a route according to the current situation of our player.

Obviously, such a program would be challenging to implement, so let us dissect the problem. The things that are easy to consider are HP and gold as they are straightforward. Low on HP? Find a safer route and try to find a rest site to heal. You have lots of gold? Find a merchant.

The more challenging thing to implement is character strength, player strength is rather arbitrary and difficult to be defined. Some cards work better with another card, a “bad” card can become good if it has synergies, and the same for “good” cards becoming bad due to a lack of synergy. The same goes for relics and their synergy with cards and decks.

With a character having around 80 unique cards each and not counting the global cards for all characters, there are thousands of combinations that makes it hard to clearly assign an integer that represents player strength.

A weak solution to this problem is to calculate the strength of a player based on how many relics he has and how many “good” cards he has. A good card can be ranked by its in-game rarity, a rare/gold card tends to be stronger than a common/silver card. Though this gets us closer towards calculating the strength of the player, it isn’t very accurate. Due to the fact that this program only counts rarity and synergy between cards.

5. AI?

An AI, even though very complex can be added into our program, it can learn alongside the player’s decision using the data to manipulate its priorities of nodes and their values. The main problem is that an AI typically needs hours of training and as the game cannot be fully automated, a player must play alongside the AI to learn, and since a player is a human, they will have their limits.

V. Conclusion

Graphs are very useful tools that has many applications, pathfinding is a common use of graph but there are other uses as well. Using graph theory and Dijkstra’s algorithm, our program can find an optimal route in the game Slay the Spire. Therefore this program can be used to help a player find a good route to play and beat the game.

The program is still only a prototype made to be the foundation to the solution to our problem. It can be improved in so many ways such as in the points mentioned previously.

VI. Acknowledgment

This paper is made possible thanks to all the lecturers and professors of IF2120 Discrete Mathematics. The author, an undergraduate of ITB in class 1 IF2120, would like to thank Dr. Ir. Rinaldi Munir, MT as the designated class lecturer for class 1. The author would like to thank the creators of Slay the Spire for creating a wonderful game to both be played and studied.

References

[1] *https://store.steampowered.com/app/646570/Slay_the_Spire/ , accessed on December 14 2021.*

[2] https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2020-2021/Graf-2020-Bagian1.pdf *, accessed on December 14 2021.*

[3] https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-Greedy-(2021)-Bag2.pdf *, accessed on December 14 2021.*

[4] Mehlhorn, Kurt; Sanders, Peter (2008). "Chapter 10. Shortest Paths" (PDF). *Algorithms and Data Structures: The Basic Toolbox*. Springer. doi:10.1007/978-3-540-77978-0. ISBN 978-3-540-77977-3.

[5] https://medium.com/@xuejunwang/slay-the-spire-a-great-combination-of-randomness-and-strategy-3a4134fb502a *, accessed on December 14 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.

Bandung, 14 December 2021

[illegible]

Jaya Mangalo Soegeng Rahardjo 13520015