**Analysis:**

Contents:

2 - [Project Background](#_Project_Background:)

3-5 - [Specific Objectives](#_Specific_Objectives:)

6-8 - [Research](#_Research:)

9-11 - [Survey Questions and Results](#_Survey_Questions_and)

12-13 - [Problem Model](#_Problem_Model:)

14 - [Sources](#_Sources:)

# **Project** Background**:**

For my project, I intend to create a roguelike game, in the style of games such as *Crypt of the Necrodancer* and the *Pokémon: Mystery Dungeon* series. This has always been my favourite style of game, as I find the fact that they are almost entirely randomly generated each time the game is played makes it infinitely re-playable.

A roguelike game is, generally, one which is a turn-based, grid-based game, wherein the world is entirely randomly generated with every playthrough. The gameplay is almost always based so that the player has an indefinite amount of time to decide to what to do next, as enemies and obstacles will remain stationary until an action is performed by the player. There are exceptions to these basic guidelines, but for the most part, this is the format used and is the format I will be using for my project.

The game will run on PC, and will use the keyboard to control the game, for example using WASD for movement of the player character. It will be a standalone program for the most part, but I may implement a way to compare statistics over a server. The game will be programmed in Python using Pygame, as it is a language that I have confidence in and that I know I can achieve this gameplay using.

The game will have a UI that includes the most important information at that time, including a “minimap” of the current floor, the current floor itself, the player’s health, and the player’s currently equipped items. This will be displayed at all times during the “dungeon exploring” part of the game, which will naturally be the majority.

The target audience for the game is people aged 11+. This is because the mechanics would be relatively simple to understand but may still be too challenging for a young child to comprehend. As well as this, as the setting and style for the game is to be based on that which many roguelike games are, such as a dungeon or dark forest, so enemies would be appropriately designed to fit these areas, which younger children may find unsettling. For example, skeletons, giant spiders, and other fictional enemies. There is no cap on the age that this game would be intended for, as it would be made up mostly of strategy and planning, a skill which can be honed at any age, and a gameplay style which can be enjoyed at any age. The controls would also be relatively simple, so even those with less experience with computers and keyboards would be able to understand them quickly.

# **Specific Objectives:**

When completed, the project should:

1. Have multiple “profiles”.
   1. Each profile should have a “save file” with unique saved statistics, username, and avatar colour to help distinguish between them.
      1. Statistics would include the highest floor reached, total enemies defeated, etc.
      2. Username would be set by the user and would not be able to be the same as another user’s.
      3. Colour of the user would also be unique.
   2. Each profile would have a document saved with that user’s statistics saved into it.
   3. When a profile is created, a difficulty setting can be adjusted.
      1. This would impact the rate at which enemies become more powerful, but not the rate at which items do.
   4. A user may delete their profile.
2. Have a save/load system.
   1. As the game is roguelike, this would only work as a “save and exit” system, wherein the player’s current location, inventory, floor layout, etc would be stored until the next time that user plays the game.
   2. When loaded, the current save file would be deleted, meaning the player cannot use this system to replay sections.
   3. Saving would save the current floor’s grid, which would include the player location, enemy locations, item locations, etc., the player’s currency, the number of floors the player has made it through in this run, etc. to a document, which would be named after the user’s profile.
   4. When loading, the program would open the user’s document and read the grid, currency, etc. from it.
   5. This would be a separate document to the user’s permanent statistic document.
3. Have a menu.
   1. The menu would include “start a new game” and “load last game” options.
      1. If there is no saved game, the “load last game” option would be unusable.
      2. If there is a saved game, and “start a new game” is selected, the saved game would be deleted.
   2. The user would be able to delete their profile here.
4. Have each floor of the dungeon:
   1. Be procedurally generated.
      1. Each floor should be unique, with minimal risk of repetition.
   2. Be made up of rooms with interconnecting corridors.
      1. This would be done by placing rooms first, of random sizes and locations, with randomly placed “doors” on the edges, then placing individual “marker” tiles, then performing a breadth first search to fill in the space between the markers and the doors, making a floor where there are no floating tiles with defined rooms.
   3. Be populated by enemies and items placed randomly.
      1. Both would have a higher chance to be placed in rooms than in corridors, but it would still be possible for them to spawn in corridors.
      2. There would be a set range of each that could spawn, proportional to the size of the floor.
   4. Be larger proportional to the number of previous floors.
   5. Have harder enemies and stronger equipment proportional to the number of previous floors.
      1. These two points would generally increase the difficulty of the game as it progresses to higher floors, due to the enemies being generally more difficult to defeat, having higher attack and health, as well as there being more of them due to the increased size of the floor.
      2. The increase in number and quality of items would help the player keep up with the increasing difficulty.
   6. Have a “staircase” to the next floor.
      1. When stepped on, the current floor will be left, and a new one will be generated.
      2. The player would keep any items they are currently holding.
   7. Have a “boss” enemy that must be defeated to unlock the staircase.
      1. The boss would have more health than other enemies, and would have a unique ability, but would move slower.
      2. Abilities could include being able to turn invisible for a couple of turns, “teleporting” the player to a random empty tile in the room, etc.
5. Have multiple distinct types of enemies.
   1. Each type would have:
      1. A base health.
      2. A base attack.
      3. A set of behaviours, which would be the same for this type of enemy, no matter the floor.
   2. As the player rises through the floors, the base health and attack of enemies would have additions to them.
      1. For example, a base 1 health, base 1 attack enemy could have 3 health, 3 attack on the 5th floor.
      2. This curve could be adjusted depending on the difficulty selected.
   3. When an enemy stands on an item, it will either equip the item or use it instantly.
   4. Enemies would only advance towards the player once they have “seen” the player.
   5. Enemies would take up 1 tile of space, regardless of size.
   6. Enemies would use A\* pathfinding to find the shortest path to the player.
   7. Enemies would be implemented using object-oriented programming.
6. Have multiple distinct types of items.
   1. Different kinds of items would have different attributes.
      1. Weapons would increase damage inflicted and could give alternate effects to attacks.
         1. Added effects could include poison, slowness, etc.
         2. This would be indicated by a different hue for the item.
      2. Armour would decrease damage taken.
      3. Food would heal damage that has been taken and refill hunger.
      4. Scrolls could be used to perform strong, one-time attacks.
   2. As the player rises through the floors, the items found would be progressively more powerful on average.
      1. For example, on the 1st floor, the player could find a wooden sword that adds 1 damage to the player’s base damage, and on the 3rd floor, the player could find a steel sword that adds 2 damage.
      2. New items would also have a higher chance to have an added effect.
   3. Items would be implemented using object-oriented programming.
7. Have a “turn” system.
   1. The gameplay would be split into the player’s “turn” and the enemies’ “turn”.
   2. When it is the player’s turn, the enemies would be stationary and would not perform any actions.
   3. When the player performs any action, it will then be the enemies’ turn.
      1. Actions would include:
         1. Moving a tile.
         2. Attacking an enemy.
         3. Using an item.
         4. Dropping an item from the inventory.
         5. Pressing the “skip a turn” key.
      2. At the start of the player’s turn, their hunger statistic would reduce.
         1. When hunger is equal to 0, the player will begin to slowly lose health over time.
         2. Difficulty would affect how fast the hunger stat goes down by per turn.
   4. When the enemies have all performed an action, it will return to the player’s turn.
      1. Enemies’ actions include:
         1. Moving a tile.
         2. Attacking the player.
      2. If an enemy had not “seen” the player, it would do nothing on the enemy turn.
      3. Enemies would act in ways according to their type and the current situation.
         1. Each enemy type would have set statistics that dictate how likely they are to commit certain actions.
         2. More complex enemies would be able to “observe” the player and make decisions based on their actions.
8. Have a minimap for the player.
   1. The minimap would be filled out as the player moves around the floor.
   2. It would have the location of items and enemies that have been seen by the player highlighted in separate colours.
9. Have a shop/currency system.
   1. Currency would be dropped by enemies when defeated.
   2. As the player collects currency, they would be able to spend it in the shop in order to purchase permanent upgrades.
      1. Upgrades would include extra base health, the ability to start with better items already equipped, etc.
      2. The upgrades would become progressively stronger as they are purchased.
         1. For example, once the weakest armour has been purchased to start each run with, the next strongest would appear in the shop.
   3. This shop would be accessible between “runs”.
   4. Currency not spent would be carried over to the end of the next run.
10. Have some character customisation.
    1. The player would be able to select different characters with different base attributes.

# **Research:**

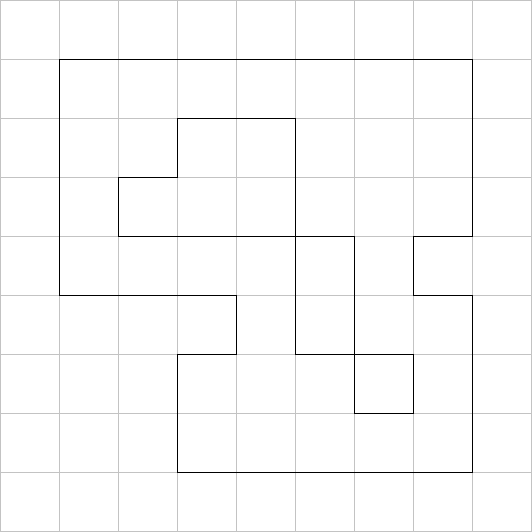
For the most part, I researched my project through the use of websites and blogs run by game developers, particularly those that develop roguelike games. Personally, I found these blogs to be highly interesting to read through, as they tended to be quite informal and “chatty”, while still managing to put across the information. As well as this, I used various guides when researching more challenging individual problems within the overall project, specifically for pathfinding, map generation, and AI.

In researching what it is that makes a game “roguelike”, I found what is known as the Berlin Interpretation, a definition of the genre created in 2008 at the International Roguelike Development Conference in Berlin. This Interpretation outlines the core principles that form a game in this genre into two main groups - high value factors and low value factors - based on the 1980 game *Rogue*.

A high value factor is one that is considered essential for the creation of a roguelike. This includes features such as having a procedurally generated map which is based on a grid, as well as having a permanent loss feature, meaning there is no way to “retry” once you have died or failed an objective.

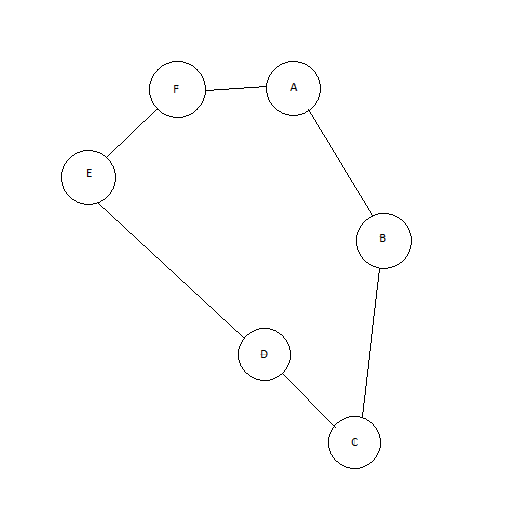
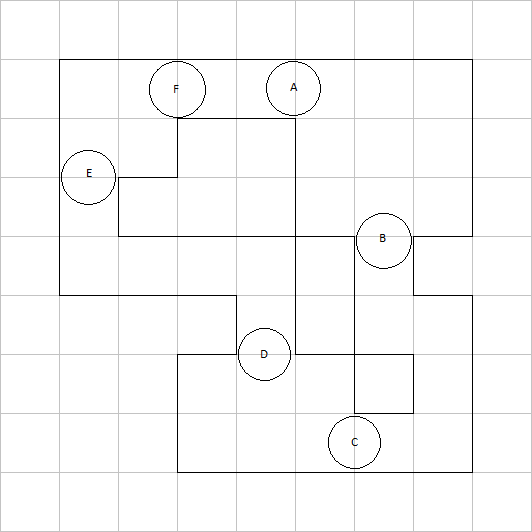
A low value factor is one that is non-essential but is typically in most of the games of the genre. This includes features such as the ability for enemies to interact with items in a similar way to the player, and the fact that the player only controls a single character.

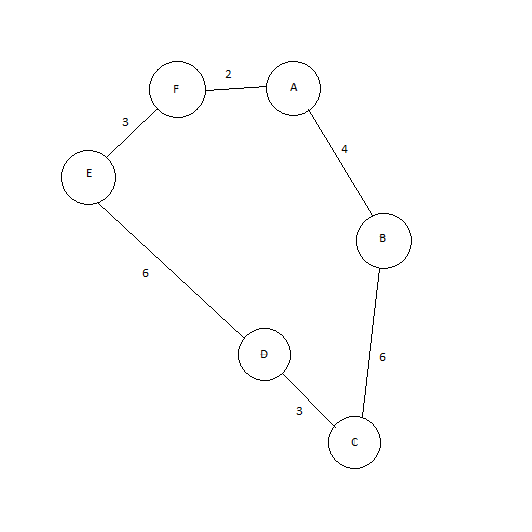
The Berlin Interpretation is fairly loose however, with many games that do not fit all criteria being considered roguelike, such as *Spelunky* and *FTL: Faster Than Light*. It has therefore been criticised for being “outdated” in certain areas, particularly by the developer Darren Grey, such as the low value factor stating that a roguelike should be presented in ASCII, due to that fact that *Rogue* was presented as such.

In order to find the most appropriate search algorithm for each task that requires one (forming the corridors between rooms, enemy pathfinding, etc.) I have done research into the area. I found that for the most part, an A\* search algorithm would be the most appropriate. This is because it is much more efficient than other algorithms, such as breadth first search, and still guarantees a path will be found. As well as this, I found a way to increase the efficiency of the algorithm by adapting the tile-based grid into a weighted graph. This is formed via declaring the nodes to represent the doorways in the rooms, as well as the enemy and the player character, and setting the “weight” between each node to be the number of tiles that must be moved through in order to get between them.   
For example:  
  


This small example dungeon floor is made up of three rooms, 1 22 room, 1 23 room, and 1 33 room, connected with 1 tile wide corridors.

Each doorway (and intersection in corridors if there were any) would become a node in a weighted graph, with the weights between them being the number of tiles between the two points on the grid.





Using this grid, encoded into Python, A\* pathfinding can be used so to find the total distance for an enemy to move from node to node. The player would be a node of their own, so that the enemies can make their way to them using the shortest path possible, once they have seen the player.

This would cause the pathfinding of the enemies to be a lot more efficient, as the distances between the different connected nodes could be found when the floor is first generated. These values would then be stored for the use of pathfinding.

Another aspect that I have researched is AI that would be effective in a roguelike environment. Through this, I found the concept of “stateless AI”. This is a simple form of AI that give each individual variety of enemy its own set of behaviours and how it would interact with the player based on various statistics for that variety. This means that enemy types that would attack in similar way, but would behave very differently, the same code can be used. The only difference would be the stats informing the choices the AI would make.

An example of this from roguebasin.com is the “archer” and the “animal”. In the example, there is a function called “too-far-from-player”, which returns whether the enemy is too far from the player to attack them. In the case of the animal, this function would always return true, as the animal has no ranged attacks, unlike the archer which naturally does. This can be extended to include more complex, situation-based decision making, such as implementing a “morale” statistic, which would basically determine if the enemy is likely to try to attack the player or run away from them. This method of coding interactions helps to give the two a different feeling to encounter without creating redundant code and is a feature I would like to implement into my project.

As well as this, the concept of state machine AIs could be used in my project. State machine AIs are AIs that make decisions based on variable factors. For example, using an “observe” function each turn, where the enemy makes observations pertaining to potential behaviours. This could include an enemy that attacks the player if it has not seen them kill a set number of that same enemy type but will run away if it has. This could help to add a level of complexity and strategy to how the player chooses to deal with enemies.

# Survey Questions and Results:

Due to them being the games of this genre that I have the most experience with, and am therefore more familiar with, my survey asked people what they thought were the most effective features of the *Pokémon: Mystery Dungeon* series and *Crypt of the Necrodancer*, as well as features that they felt would enhance the experience of a roguelike game. In addition, I asked for any other roguelike games that they thought I could take inspiration from.

The survey was a self-selected questionnaire, as I asked for volunteers on my Instagram story. As the survey was more specialised towards the two games, not everyone that volunteered was able to answer the majority of the questions.

***Have you ever played a Pokémon: Mystery Dungeon* game or *Crypt of the Necrodancer*?**

In this question I gave the options of:

* Yes to both.
* Yes to Pokémon: Mystery Dungeon.
* Yes to Crypt of the Necrodancer.
* No to both.

Of the 13 people that responded, 1 person answered that they had played both, 10 people had only played Pokémon: Mystery Dungeon, and 2 had only played Crypt of the Necrodancer.

This was not surprising, due to the fact that Pokémon: Mystery Dungeon is of course part of the popular Pokémon franchise, developed by Spike Chunsoft, and published by The Pokémon Company and Nintendo. It was therefore advertised much more strongly, has a higher level of polish, and was published on the Nintendo DS, making it much more available to children, the target demographic for the series. On the other hand, Crypt of the Necrodancer was originally a standalone game, produced by the independent studio Brace Yourself Games, and published by them as well. Therefore, the game was advertised a lot less, and originally, Crypt of the Necrodancer was released for Windows, OS X, and Linux, making it a lot less noticeable for the most part.

***What are features that you find the most interesting/fun from Pokémon: Mystery Dungeon, in terms of gameplay?***

Of the responses about the aspects people enjoyed about the series, almost all of them referred to the random generation of the levels and how it helps the game to remain fresh. Due to this, this aspect is one of my highest priorities.   
*“Use of an ever changing level design, dungeon levels are always different every time you play”*

Another aspect of the games that was brought up is the hunger mechanic introduced in the series. This is a mechanic wherein the player character has an in-game statistic that is always showed to the player. As the player moves around the dungeon, and performs actions in it, the player’s hunger will reduce from a max of 100. When hunger reaches 0, the player’s health slowly decreases until it is brought above 0 again. In order to increase hunger, food items are present in the game. As well as this, some items that have completely different effects have the side-effect of increasing the hunger stat. This is a feature that I would be interested in including, as it helps to give the player an incentive to explore and find new items, as well as adding another level of depth to the mechanics of the game.  
“*I like the dungeon crawling, and also the hunger meter is pretty unique*”

Another aspect that respondents mentioned that they appreciate is the graphics of the early games in the series. They specified that the vibrant colours and the pixel art style of the games were very appealing. I too find the style incredibly attractive, and I plan to imitate it in the designs for my game.  
“*I haven't played the games in years and yet it still attracted me as the colours and the graphics are so vibrant*”

***What are features that you find the most interesting/fun from Crypt of the Necrodancer, in terms of gameplay?***

A feature that was mentioned in every response to this question is the main gimmick of Crypt of the Necrodancer: the rhythm-based turns, which force the player to play to the beat in order to do well. As the player remains on-beat and defeats enemies without taking damage, a coin multiplier builds up, incentivising the player to do so. Once a beat is skipped, the player takes damage, or the player attempts to perform an action off-beat, the multiplier is set back to 1x, punishing the player. The coins are used mid-run in shops in the dungeon itself in exchange for items, therefore the player is pushed to try to collect as many coins as possible. Coins are dropped by enemies when defeated. I do not think I will include this rhythm-based gameplay, but I would like to include some variety of multiplier for playing well, as well as the mid-run shops.  
“*The multiplier for coins incentivises you to play to the beat*”

Another feature brought up is the difficulty curve of the game, as it starts off difficult and slowly increases in difficulty. This allowed the player to try and fail without making too much progress, meaning that they will not become frustrated due to losing because of inexperience when they are on a deep floor. This kind of difficulty curve is one that I intend to try to capture, as I find that this helps the player to quickly learn all of the concepts of the game, instead of not learning all of them and becoming discouraged due to dying to something unexpected far into the run.  
“*The level design and difficulty arc felt more comfortable than most games, it starts hard and gets slowly harder so you have to learn quickly, and for me dying early in a game so I can master the mechanics beats getting far into a game only to realise I don’t know how to properly play*”

***What features do you think have a place in these games that they don’t have?***

Of the responses given, quite a few referenced ideas pertaining to the story of the games. Examples of these include the idea of a branching storyline or more expansive side-quests. As my game is going to have minimal to no story, these ideas will not have a place in my project.

However, the idea of small side-quests is something that I feel could work, and I may attempt to implement. This could be included by, for example, having a possibility for a sign to be on each floor, perhaps specifying to the player a side objective besides simply reaching the next floor. This could include side tasks like defeating a specific, stronger enemy or finding a specific item or non-hostile NPC. This could be done in the style of “wanted” or “missing person” posters respectively. If the player has found and seen the sign, the specified enemy, item, or NPC would be made to spawn on the next floor, and once the task is complete, the player could receive currency or a useful item.  
“*It should have a random objective*”

Another idea presented referred more to the graphical style of the game, where the idea is that the game should have more different environments, which would make the game more interesting to look at and explore. This could potentially be achieved by the game selecting a random theme for the run as it begins, for example having a snowy dungeon one run, and having one based more on a tropical jungle. There would not be much of a gameplay difference if any, though it would help the game feel less repetitive.  
“*More unique environments*”

Another requested idea that came up was a system of items similar to that of *The Binding of Isaac*, wherein items that are collected by the player give permanent effects to the player’s attacks, movement, etc. As items are collected, their effects combine and stack, which can end up with unique effects. While this is a very interesting idea, I do not think I will include a system like this, as I do not intend to introduce items that would give the player permanent effects mid-run. However, I may do so through the between-run shop, where a basic version of this item system could be implemented.  
“*In games such as The Binding of Isaac, there are synergistic item combinations and “transformations” where certain items give you a set bonus*”

***Which game would you say has the better map generation?***

As most people had only played one or the other of the two games, most people answered that they had no preference on the matter. However, of the 4 people that did have a preference, 3 of them chose the map generation of Pokémon: Mystery Dungeon. The map generation in this series favours rectangular rooms spread out over an area and connected with long, thin corridors, while Crypt of the Necrodancer favours rectangular rooms next to each other, with short, wider corridors, and rooms hidden in the walls due to the game having a mechanic wherein the player can dig through the walls of the dungeon.

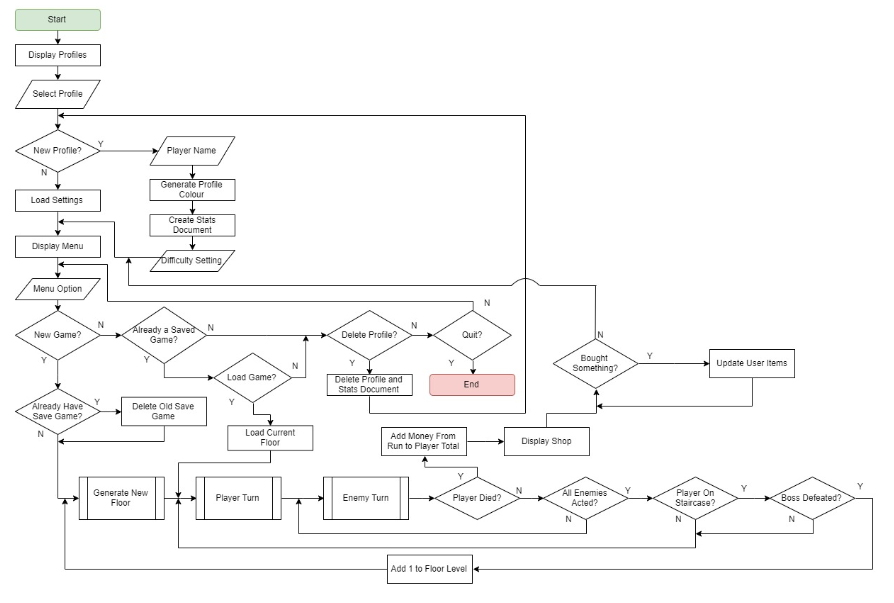
Overall, the generation used in Pokémon: Mystery Dungeon does tend to lend itself more to exploration to completion in order to find every item and enemy on the floor, as the player is only required to visit each room on the floor, as opposed to checking every space in the floor due to areas being hidden behind destructible walls. Therefore, this is the style of map generation that I will use when designing the algorithm for the floors in my game.

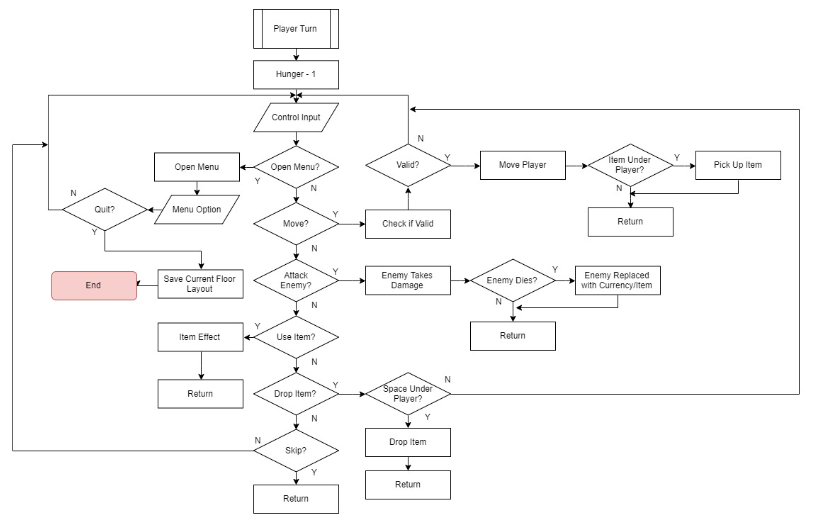
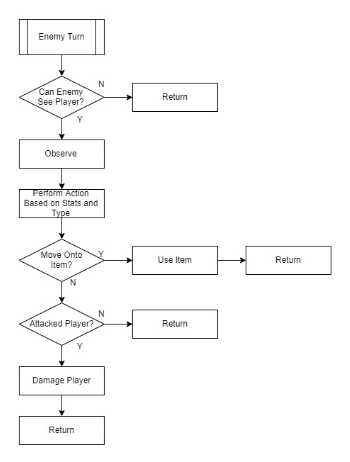
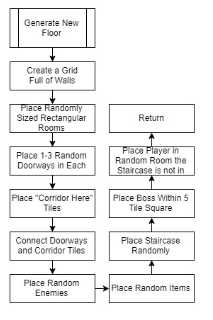
***What other roguelike games would you consider to be a good example?***

An example of a roguelike game that came up often from this question was *Enter the Gungeon*. Almost every person that gave an answer mentioned this game. In essence, this is a game based around bullet-hell gameplay in a randomly generated dungeon, wherein pre-defined rooms are arranged randomly with random enemies and items populating them. Due to the fact that the gameplay is based more around the concept of a bullet-hell roguelike, I do not feel that there is much I can take away from this game to use in my own project, as I am planning to create a turn-based roguelike. However, the concept of pre-designed rooms being populated with random objects and enemies could be used in my project, perhaps with a uniquely shaped room housing the boss and staircase of each floor, as opposed to the randomly sized rectangular ones that the rest of the rooms on each floor will be.

Another example of a suggested roguelike with the same bullet-hell/roguelike gameplay is *The Binding of Isaac*. As well as having a similar gameplay style to *Enter the Gungeon*, the game makes use of pre-defined rooms arranged randomly, populated by random enemies, and giving the player random items. As the gameplay is similar to that of *Enter the Gungeon*, I will likely not take inspiration from it either.

# **Problem Model:**





# **Sources:**

Definition of Berlin Interpretation: <http://www.roguebasin.com/index.php?title=Berlin_Interpretation>

Darren Grey’s Criticism of the Berlin Interpretation: <http://www.gamesofgrey.com/blog/?p=403#targetText=The%20Berlin%20Interpretation%20as%20it,a%20dungeon%20or%20using%20ASCII.>

A\* Algorithm:  
<https://www.redblobgames.com/pathfinding/a-star/introduction.html>

Grid Based A\* Algorithm:  
<https://www.redblobgames.com/pathfinding/grids/algorithms.html>

Stateless AI and Behavioural Example:  
<http://www.roguebasin.com/index.php?title=Roguelike_Intelligence_-_Stateless_AIs>

State Machine AI:

<http://www.roguebasin.com/index.php?title=Roguelike_Intelligence_-_Intrinsic_Information_and_State_Machine_AIs>