Rogue Like Planning

## Are your player, items and enemies the same class, different classes in the same family, or completely different classes?

Player and enemy class will be a child class to the parent class called Sprite which will contain common fields and methods for all the characters in the game. All items in the game will also inherit common fields and methods from the parent class called Item. Items includes health potions and gold.

## What logic will you put into your Form class? What logic will you put into your Game Manager class?

All the game logic will be done in the form since there is not going to be a game manager class.

## What class(es) do you need to implement the dungeon? Briefly explain the job of each class, list the data members it must hold, and the methods it must expose. How do the Dungeon and the TileMap communicate?

There will be four classes to implement the dungeon. The Tile, TileList, TileMap and the MapGenerator class. The Tile class will hold the image of each tiles in the game which will be draw onto the form. The TileList class will hold an array of the all the Tile objects in the game. The TileMap class holds a two-dimensional array which will house the map that is generated in the MapGenerator class. This two-dimensional array will hold a list of integers correlating to the index of the TileList array to get the correct tile that is require forming the map. The MapGenerator class will house the algorithm to procedurally generate the map by creating ten rectangular rooms that does not overlap and join them together with a thinner rectangle representing a corridor. The map will then be sent to the TileMap class.

## What data structure(s) do you need to hold collections of enemies and items?

Dynamic list for all the enemies and items in the game. Since there is going to be different types of items in the game, all the items in the game will be controlled by one dynamic list.

## Does the dungeon need pointers to its sprites? Why or why not?

No, however the sprite needs a pointer to the map dungeon to check whether the characters can move according to the tile type that the character is about to move onto to.

## Does the sprite class need a pointer to its dungeon? Why or why not?

Yes, the sprite needs to know what map position it is on in order to make calculation about whether the tile it is on is walkable or not. If the tile is walkable, the sprite can then move its position. If the tile is not walkable nothing will happen to the position of the sprite.

## What enum types (if any) do you need?

Direction enum which consist of north, east, south and west in order to select he correct sprite animation to display and move the sprite in the correct direction. Each direction that the sprite move in will require a different sprint sheet which will make the character face in the direction selected using the enum.

## Does the player sprite need access to the collection(s) of enemy sprites?

No, however all the enemy will have a pointer to the player sprite class as this will be used to check for collision with the player which will then trigger the game’s combat system.

## What class is responsible for creating the collections of enemies and items?

There will be two dynamic list classes which are EnemyList and ItemList. The EnemyList class will control all the behaviours of all the enemy in the game and the ItemList will control all the behaviour that all types of items in the game will exhibit

## If you are using an FSM, what class calls the FSM methods of the sprites?

## At each game cycle, you need to perform collision detection between the player character and each enemy and item in the dungeon. What class or classes hold a method to compare the areas of two entities to check for collision? What is the function header of this method? What other classes are involved in the collision detection logic?

All the items and enemy in the game holds a reference to the player’s position. This is then used to check if the player has collided with an item or an enemy base on its own position. The method for checking the collision will be held in both the items and enemy class. The class that will call these methods are the dynamic list class for each class. For example, the EnemyList class will call the enemy player’s collision with an enemy method while the ItemList class will call the player’s collision with the item method.

## Describe the AI you are going to include. Describe the behaviour and describe the implementation logic

The enemy character will have a chance of pursuing the player character by using randomly generating number as chances. For example, if the chance is greater number is greater than 5 then pursue the player. If the chance says to pursue the player character, the enemy will change direction and start moving in the direction that the player is at. For example, if the player’s x position is smaller than the enemy’s x position, the enemy will start moving left towards the direction of the player.

## Describe the trigonometry you are going to include (if not already contained in the AI from item 12).

If the players x position is smaller than the enemy x position, the enemy will move left. If the player’s x position is more than the enemy’s x position, then the enemy will move right. If the player’s y position is more than the enemy’s y position, then the enemy will move down. Lastly if the player’s y position is smaller than the enemy’s y position, the enemy will move up.

## Describe, in detail, the logic of your battle algorithm and computations.

## Sketch the screen layout -- with -- controls that you will use to provide feedback during battle.

Combat will be turn base similar to the game Pokémon. The player will be able to choose from two different types of attacks by clicking the two different buttons. Both attacks have a chance of missing base on randomly generated number. For example, if chance number is greater than 5, the attack will miss. The chance of a big attack missing is greater than the chance of a small attack missing as the randomly generated chance number will be implemented differently to increase and decrease the chance of an attack missing base on what type of attack was selected. The enemy character will have the same type of attacks with the same chance of missing. The two party will take turn attacking each other and the combat will only end when the either character’s health reaches zero. The sketch of the combat screen can be seen below.

Enemy combat sprite animation

Player combat sprite animation

Heavy attack

Light attack