MONK – Specification

Contents

Specification

The Blesses Dungeons are a procedurally generated dungeons where the monk, traverses the dungeons in search for treasures to support his monastery which is on the edge of financial collapse. But the monk must be wary, for they are not the only things walking within these walls. Many monks have come to the Blessed Dungeons for the same purpose and met grisly ends – only the strongest will make it out alive.

The game has several requirements to be release worthy for the official release – week 22.

These basic requirements include:
A dungeon generation system.

- There are three types of room: Empty, Monster, and Treasure.
    - Empty room will be a super class for any room, with specific requirements being added onto the new subclasses.
- There can only be one treasure room.
    - The room will be created as a singleton.
- A door must pair with another in the next room.
    - A room will check around it for doors leading to it before generating new doors.
- The rooms must fit within a 10X10 grid. (This may change)
    - The rooms will be in a grid with x and y co-ordinates between 0 and 9
- The Monk must enter a room with no monster.
    - Change the state of a monster room to empty if the monk spawns there.
- The treasure room should not have a monster within.
    - Spawn monster method can only be called in a monster room.

An exploration system.

- There can only be one monk.
    - The monk class is a singleton.
- The Monk must be able to get to each room.
    - Doors are generated above and act to move an X and Y value for the monk.
- The Monk can only leave a monster room once the monster is slain.
    - Combat starts on entering a monster room and cannot be left until a character dies.
- The Monk must exit the dungeon from the treasure room.
    - The win method can only be called in the treasure room.

- The Monk must be able to use items within empty rooms.
    - Functions are available if the monk is in a room with no combat active (Monster rooms become empty if the monster is slain).

A combat system.

- The fight should begin upon entering a monster room.
    - If current room = monster then fight().
- Whomever has the higher speed attacks first.
    - If monk.speed > monster.speed…
- Actions have a 50% hit rate.
    - Using a random number generator – 0 miss, 1 hit
- Actions include Attack and Defend.
    - 2 possible actions which call a different method.
- A successful attack will subtract the attacker's attack level from the target's health.
    - Implement hit method (Health – othercharacter.getAttack())
- A successful defend will add 1 to the defender's health.
    - Implement defend method. (Health++)
- If a character's health is or is below 0 they die.
    - If health <= 0 this.destroy
- If them Monk dies, the game is over.
    - If getMonk == null GAME OVER (or something similar)

The initial game will be text-based. Once the basic game nears completion the game will be overhauled to be a top-down based game (either 8-bit style 2D, or a modern 3D with lighting and shadows.)

The I will likely develop the game using CLion as in my opinion, the interface is cleaner, and it offers more versatility and support while developing over Visual Studio. I will use C++ as that is the requirement within the specification and may consider using the CLion support for integration within the Unreal Game Engine. This would allow the front end to be developed using industry standard technology and spend more time developing textures, models, and the code for the game rather than the tools for this functionality.

Structure and Behaviour

Included within this section is a set of diagrams outlining the functionality of several sections of the game.

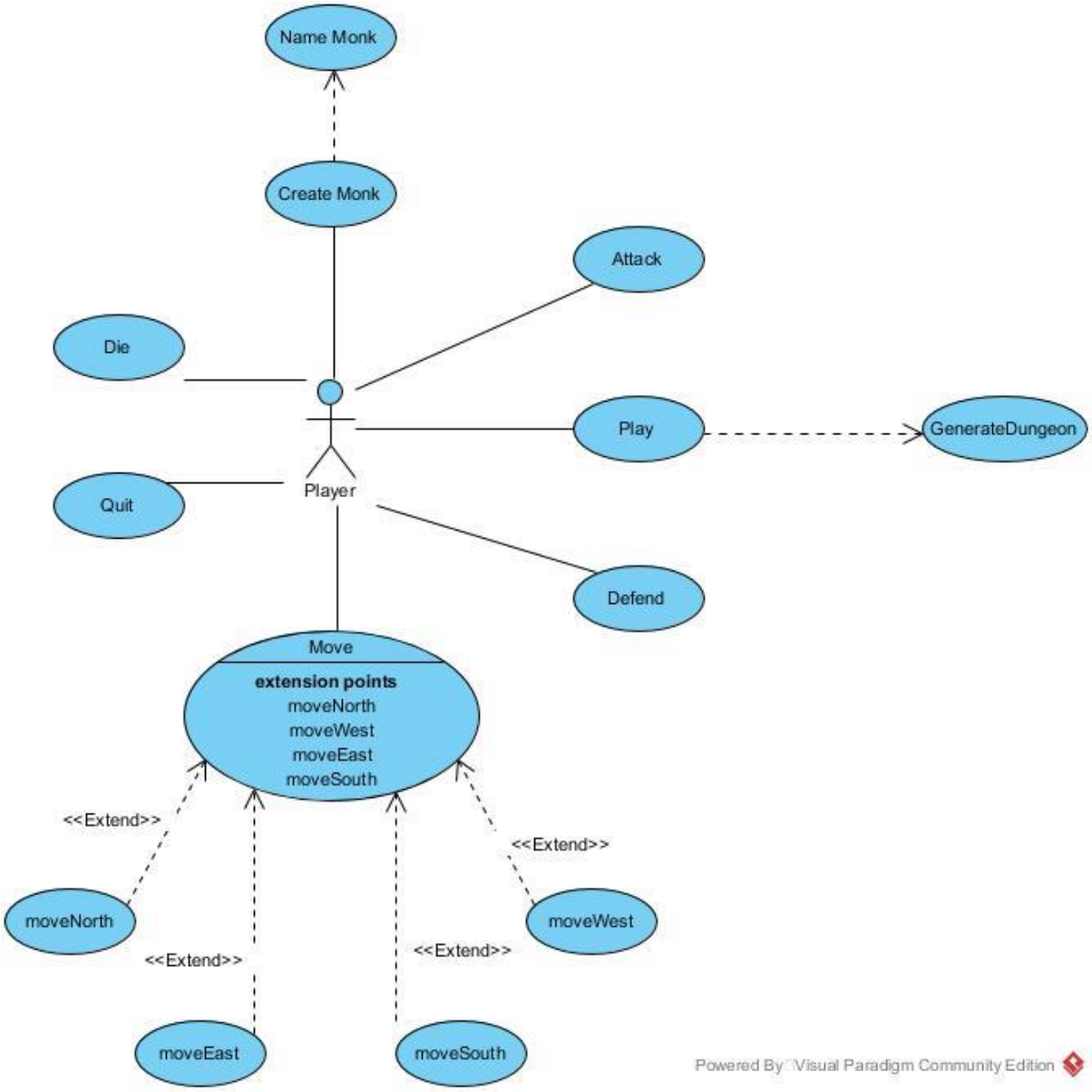

Figure 1

Figure 1 shows the user's basic interaction within the game. The player must be able to create a monk with a name and create a dungeon using the GenerateDungeon method. It must also be possible to win the game and lose.

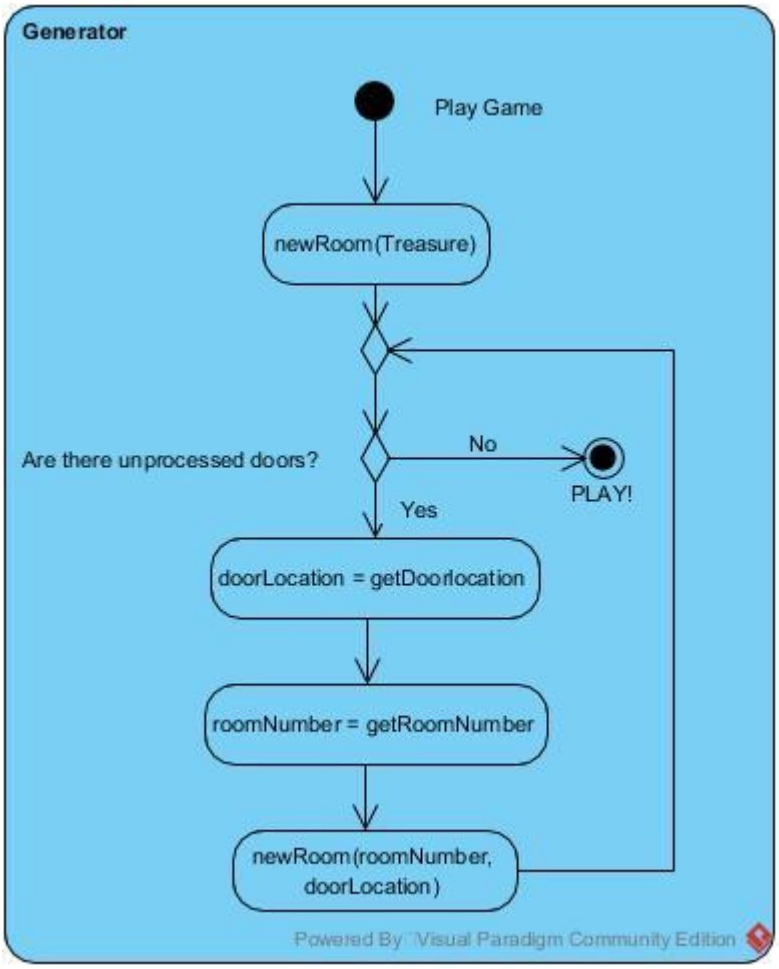

Figure 2

Figure 2 shows the first iteration of the GenerateDungeon method. The first room to be generated is the Treasure room and a cascading recursive function runs until either the dungeon is full or there are no more doors that lead to nowhere. Figure 3 (below) shows a proposed algorithm that runs when a room is created, checking the rooms around it for doors that lead to that room and adds them accordingly. The generator allows rooms to generate no additional doors, but still lead to another room through the door used to generate that room.

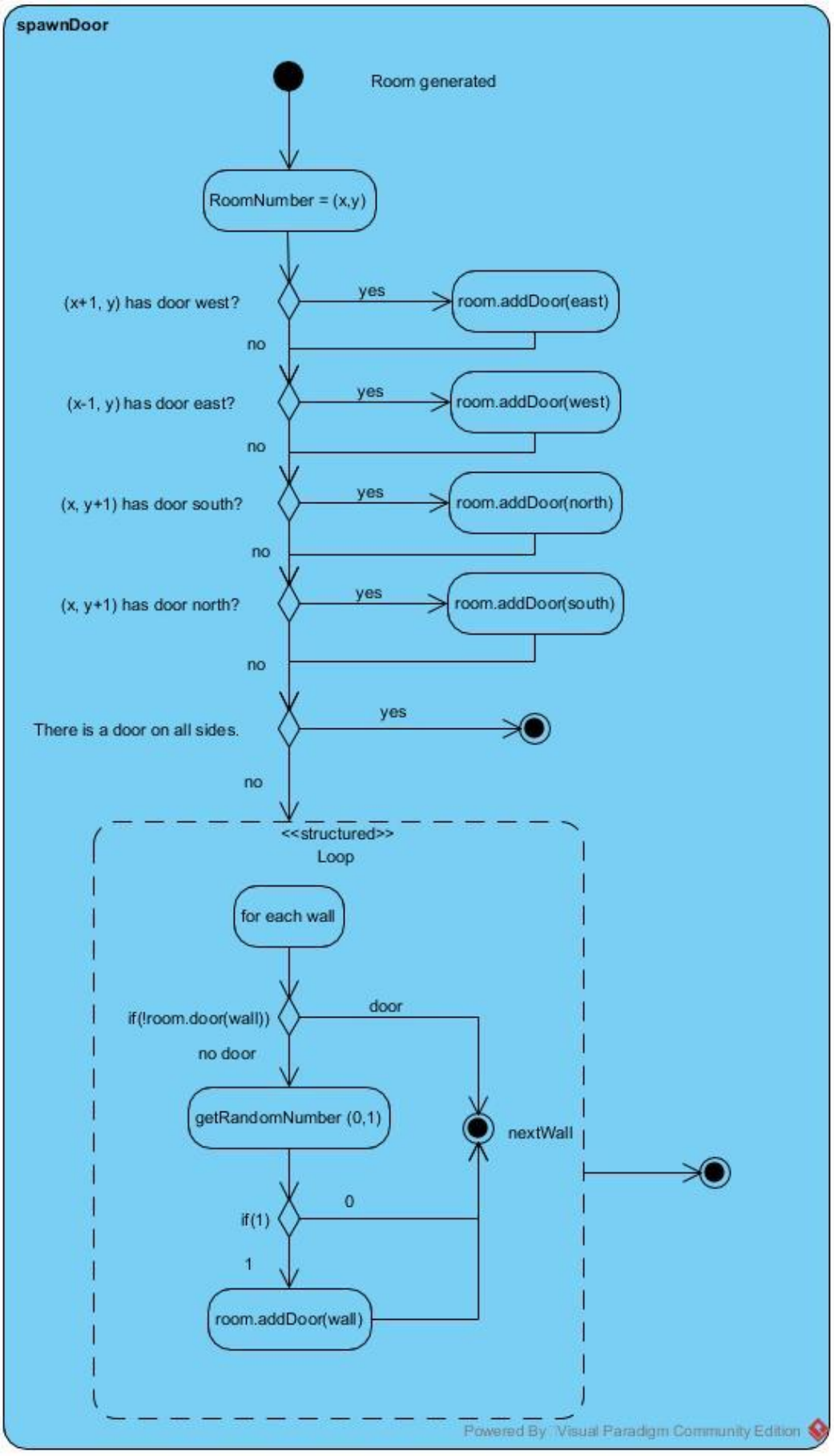
spawnDoor
Room generated
RoomNumber = (x,y)
(x+1, y) has door west?
yes
room.addDoor(east)
no
(x-1, y) has door east?
yes
room.addDoor(west)
no
(x, y+1) has door south?
yes
room.addDoor(north)
no
(x, y+1) has door north?
yes
room.addDoor(south)
no
There is a door on all sides.
yes
no
<<structured>>
Loop
for each wall
if(!room.door(wall))
door
no door
getRandomNumber (0,1)
nextWall
if(1)
0
1
room.addDoor(wall)
Powered By Visual Paradigm Community Edition

Figure 3

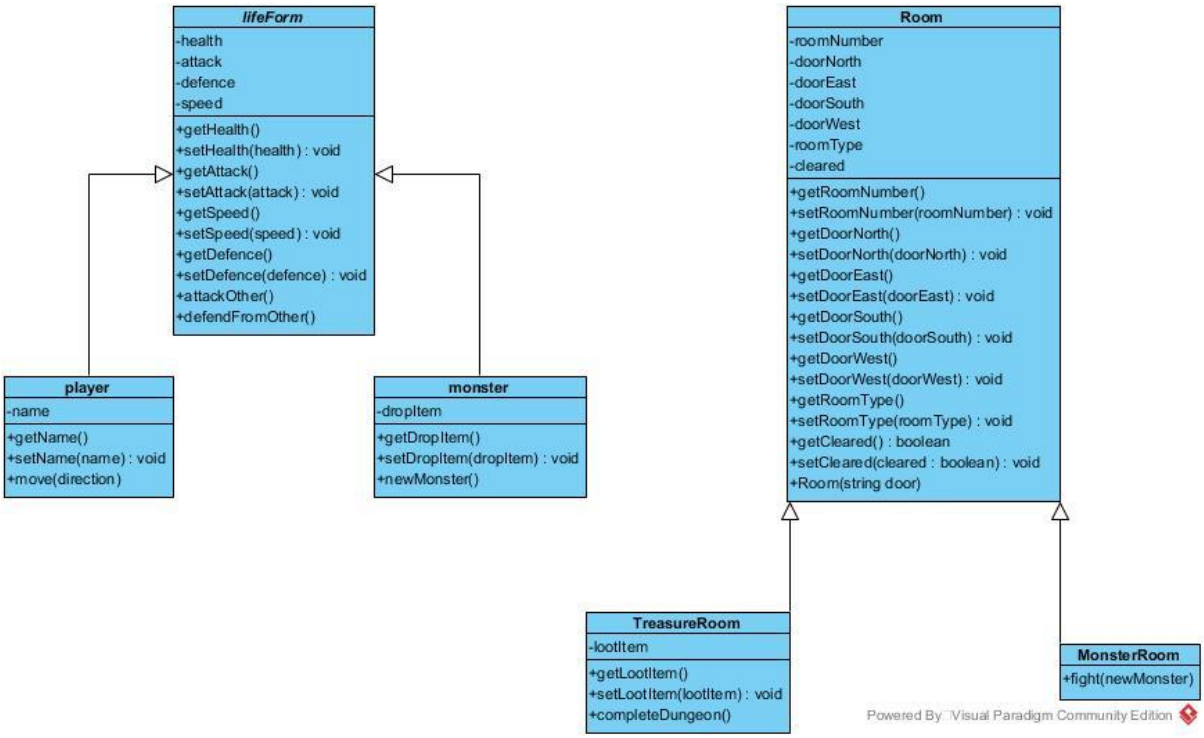

Figure 4

Figure 4 shows the proposed basic classes for both characters and rooms. As lifeform is not the name of a character and is not created it is declared abstract. Monster is the only enemy type currently but can be extended to create different sub types of monsters which implement other types of methods which may include using items or special abilities.

Something not shown in Figure 4 is that a Dungeon is made up of rooms. This is important to clarify as the dungeon is the main play area and if there are no rooms then the whole game will break. A Dungeon will be a singleton but there can be anywhere from 3 to 100 rooms (Must be at least 1 room of each type).

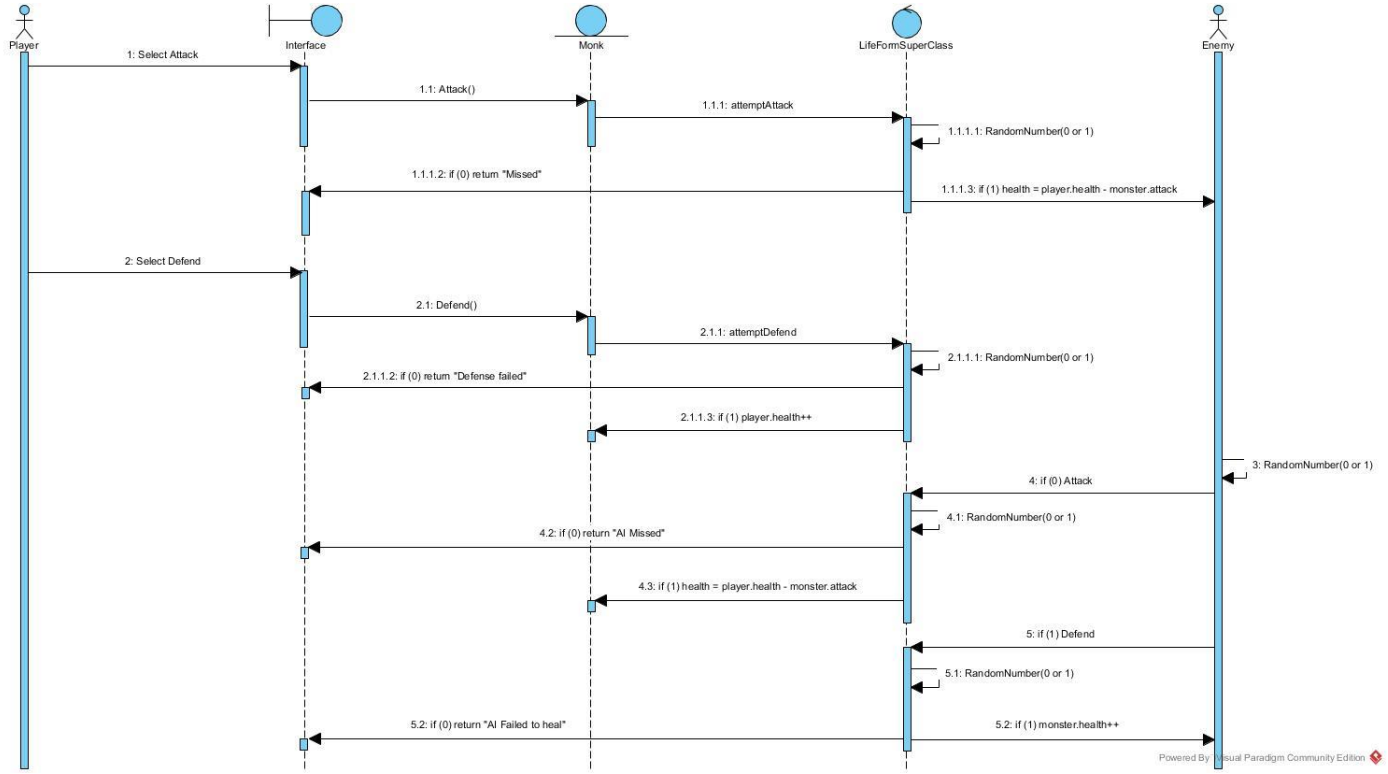

Figure 5

Figure 5 shows the initial concept for the combat mechanics. This doesn't check speed, so the player always moves first. The player can attack and defend as can the AI. Both characters have a 50% chance of success. The AI randomly selects a move, but this may change in the future to use logic based on each character's health.

Use of the Object-Oriented design pattern

There are many examples of good Object-Oriented designs within my suggested solution.

Each element within the game is an object, giving me flexibility within the game to perform many functions within the runtime.

Rooms, the Monk and Monsters are all examples of objects that I will implement. This is due to the nature of each, and there being many different types of each.

Implementing monsters to be an Object allows for quick additions of different types to be made, due to all monsters needing similar attributes. This allows each type of monster (and the Monk) to be subclasses of a super class relating to creatures.

For the main movement within the dungeon, I will utilise the "Observer" Design pattern, as it will change the state (room) based on an interrupt created by the user's input.

The dungeon generation will use a recursive function to create a cascading effect of rooms being generated from the treasure room outwards (If a room cannot be created then end that branch).

The combat will be an "iterator" pattern that runs while both characters are alive.

Interface design and UI components

There will be a list of descriptions with an id for each which can be assigned to a room. This allows a user to interact with the current room and get more information about the setting (Including descriptions of what you can see, the monk's thoughts about the room, events that happen, etc).

For the initial text-based game there will be no interface. When I make the graphical interface for the game the following information will be what I base the design on.

There will be minimal components on the screen. This will fit the theme of the game more as there will be very little to look at, as there would be for the monk within the dungeons.

There will be a health bar. This will be minimal within a corner to allow the player to see their health without it taking up a large section of the screen.

There will also be a map that shows the layout of rooms discovered by the monk. Once the monk has moved through a door the room is discovered and appears on the map.

The cell for the screen will be the room. The room will move, and the monk will remain in the centre of the screen with movement animations to show the character moving within the screen.

If the player presses TAB an inventory will open, allowing the user to use items and access other functions within the game.

Additional material

Music:

While working, Spotify made a music list and one of the pieces of music works well and fits the theme.

The track is Paths by Be'lakor:

https://open.spotify.com/track/5eBJDTLN7lK4mXu5IFs5zC?si=dBCsEeZ_RyinKM-zmLQZ7g

This music is a very mellow track with very little in the way of texture, but the inclusion of subtle reverb and delay create an atmosphere like the track was performed within stone walls. There is consistent use of dissonance within the music which also fits the gloomy feel of the game. The music has a very simple structure, with a melody in the right hand of the piano and basic accompaniment which represents the monk on their own within the dungeons. The progressive nature of the track also references the progressive nature of the game. There are clear themes within the music that can be edited to trigger on certain events, or to loop the music if the player is taking their time (This would be for the graphical game later down the line).

Industry comparison – Runescape Dungeoneering.

Runescape in an MMORPG made by Jagex which has a mini game called Dungeoneering found in the ruins of Daemonheim. Within the game, the player goes from floor to floor exploring rooms to get to the boss room and thus the next floor.

The player is only able to use 3 items and whatever they can find / make within the dungeon.

The floors are procedurally generated based on the floor number, complexity level, number of players, and the value of players' skills – all floors must be possible to complete.

Figure 6

In this 3D environment the camera follows the player as they move through the dungeon. They are able to look into an undiscovered room by clicking on the door, revealing what lies behind it. Figure 6 shows the basic game when it first came out. It has since been updated with better graphics and more features.

There are empty, monster and win (boss in this case) rooms. There is also a dedicated start room and skill / puzzle rooms where the player must complete a task before moving on. This may be a featuire I implement later within development.