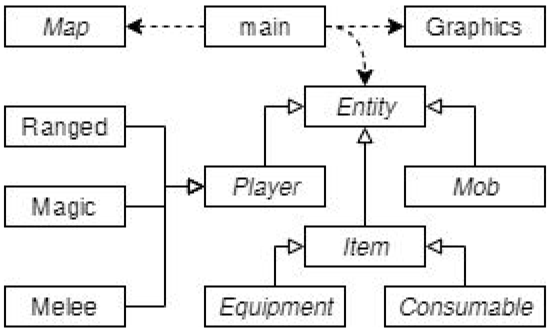
Dungeon Crawler Project Plan

1. Project scope

Our project will draw inspiration heavily from games such as Binding of Isaac and Enter the Gungeon. The game will be a bullet hell roguelike game with real-time combat, randomly generated rooms, WASD+mouse controls, enemies with different fighting styles, items that change the player's fighting style, HP system, and a Pac-man style scoring system with a leaderboard. Besides that, additional core gameplay features we aim to implement so far include: three different player classes (melee, ranged & caster) and a character progression system with level-ups gained by looting and killing monsters. In the spirit of roguelike games as the player progresses more dungeons are generated and the difficulty of the dungeons ramp up endlessly until the player dies.

2. Software structure



A class diagram

Main game loop

The game loop will be divided up to parts to represent the starting menu, the settings menu, a leaderboard, and the game itself. The program will run until it is closed by either closing the window or exiting the game from within. Different states (menus and the main game) are handled as functions inside the main.cpp file. The game loop will keep track of things like total score, number of floors explored by storing all the contents of the game in a Game object.

Map

The hallmark of dungeon crawlers is procedurally generated content; naturally, this also applies to our game in which randomly generated rooms and floors are a core component of the game.

The scope of how vast the final plethora of different floor layouts are yet to be decided, but once the algorithms for generating floors are in place it should be simple to expand the number of floors available. There are many variables that can be tweaked to create widely differing rooms, such as enemies and their placement, different types of obstacles, the geometry of the room itself and different base types for rooms such as shops, item rooms, etc. But excessive randomness is not great from a gameplay standpoint, since if your generated rooms are all widely different the difficulty can also vary wildly. Thus we have to set in place some parameters and design some rooms by hand so that the level of difficulty stays somewhat consistent.

The base of the implementation for maps will be an abstract class, Room that will at least have variables that define its shape and size and containers for enemies and objects, such as obstacles, interactable objects, passages to different rooms. Implementation for travel from one room to another is not decided yet but two possibilities are either to handle passageways as objects that contain pointers to connected rooms, or for rooms to have a container for pointers to its neighbors. Different room types such as boss rooms, shops, etc. will then inherit from this base class and have differing functionalities. Implementation for constructing the room objects is still undecided since it needs to randomly generate varying rooms. One solution would be that room objects are constructed plainly by their size and shape and then the algorithm responsible for the procedural generation fills the containers and setups neighboring rooms and the like. The implementation of map objects is still up in the air, but for example, one idea would be for obstacles, monsters, and even the player to share a base class. In which case an obstacle would be an object that couldn't move, enemies objects that have some AI and the player a controllable object. Since new floors of rooms are created until the player dies it is very important to make sure that memory handling is done properly. After the player completes a floor all the rooms and the contents of those rooms should no longer be stored in memory.

Entity

All objects within the rooms are handled with Entity-type objects, which represent a thing taking up space within a room.

- Player: the main character controllable by the player, has three main classes: melee, ranged, and magic (caster-type character)
- Mobs: AI-controlled mobs found throughout the dungeon. Different mobs will have different attacks and behaviors. Mobs will have a difficulty class, which determines the score received from defeating them, as well as a random chance for an item drop.
- Items: Items are found randomly throughout the dungeon. Items are divided into consumables and equipment. Equipment such as armor and weapons increase the player's capabilities and consumables for example healing potions offer the player more utility.

- Projectiles: All of the attacks in the game are handled as projectiles. Projectiles have different sizes and velocities for example a fireball will have a bigger projectile than an arrow but an arrow will move faster.

Graphics

Two main parts: graphical user interface and character sprites. We will be using simple texture sprites to represent parts of the dungeon. Entities will get their own sprites, with some animation frames to make the game feel more alive. We will possibly add color variations to monster sprites in code, which can be randomly assigned at monster creation. The textures of dungeon walls will adapt to the different shaped rooms, eg., Corners will have their own textures. GUI will consist of sprites and UI elements, such as indicators for player HP, items carried and score.

3. Planned use of external libraries and other used resources & tools

At this stage external libraries will include SFML, further additions might be made as a need arises. SFML seemed like the most suitable of the two suggested multimedia libraries due to its ease of use and better utilisation of the features of C++. SFML's keyboard and mouse reading, graphic and sound modules, transformable object classes and other functions of SFML are going to be used widely in the project. For development and testing we will use the aalto.fi remote access environment to ensure that all testing is done on a consistent platform.

4. Division of work and responsibilities between the group

The rudimentary division of work is as follows, but as the project progresses there's a lot of room for flexing between tasks, such as in the case that a task proves to be more complex than originally thought, multiple group members and be assigned to the task. There's also fluidity in implementing additional features, for example, once the abstract base classes for enemies and items are in place any group member could implement a new type of monster or item if they come up with a great idea for one.

Entities, player - Son

Map gen - Rashid

Item groundwork - Antti

Game loop - Leo

Graphics and sound - Everyone

5. Planned schedule and milestones before the final deadline of the project

The final graphics & sound will be handled at a later stage, but basic graphics will be implemented throughout the project to help with testing features.

We deemed it best to leave at least a week before turning in the project for final debugging and the like.

Stages:

- Research & creating a basic testing setup, first 1 or 2 weeks
- Development of the game: adding features and exploring new concepts, next 2 weeks
- Adding graphics and sound, 1 week
- Testing and debugging, 1 week
- Finalizing the program, a few days
- Project deadline 28.8.

Intended target states for each stage

- Basic graphics, a single room and a controllable player (Basic testing setup)
- Required basic features in place: (Primary development)
  - Simple 2D graphics
  - Moving through doors
  - Combat
  - Collectible power-ups
  - Scoring system
- Additional features in place: (Further development)
  - Random generation of maps, monsters, and collectibles
  - GUI
  - Character progression
  - NPCs (Shopkeepers, etc.)
  - Sound effects, music
- All desired features in place (Testing and debugging/Finalizing the project)