Aalto University

ELEC-A7151 - Object oriented programming with C++

Project Plan

Dungeon Crawler

Member: Phi Dang, Mai Vu, Khoa Nguyen, Taige Wang

1. Brief Introduction

The Dungeon Crawler project is a Rogue-lite, turn-based gameplay. The Rogue-lite element will be implemented with perma-death features, random dungeons, and a currency/upgrade system that carries on after the player's death, which helps to reduce the difficulty of the game. The turn-based gameplay element takes inspiration from the famous game franchise Fire Emblem, where each room of the dungeon is a tile-based map.

2. Features and Functionalities

a. Rogue-lite gameplay

The dungeon is essentially a randomized tree of rooms inspired by a rogue-like game Slay the Spire. A room or rooms from the previous level connect(s) with other room(s) in the next one, creating a tree similar to the image below.

(Source: https://comicsverse.com/slay-the-spire-excited/)

There are two types of rooms, Dungeon and Resting rooms. Dungeon rooms are random rooms with, similarly to Slay the Spire, randomized objectives such as mini-boss fight, eliminating all enemies, or getting to the treasure. These random rooms also have randomized terrains as well as enemies to make the gameplay unique for each run. Resting rooms, which are always the same, have a Store where the player can spend his/ her money collected during the run on upgrading or purchasing items and units. It is also a place for healing. The last room of the dungeon is the boss room.

b. Turn-based gameplay

The player has a team of 5 members/units. At the start of the whole game, the player is given 5 basic units. During the runs, the player can spend his/her money to buy new units and upgrade the existing units to make the runs easier.

(Source: Google)

Each playable unit has its own type, such as range, melee, etc., and some special abilities. The unit may enter the combat phase when there is at least one enemy unit in attack range. The attack range of each unit varies depending on its type. For example, an archer can attack in a circle of radius 3. A sniper can shoot in a 6-8 range but only in a straight line. A melee can attack in a circle but with a radius 1. Each unit also has its special attack/ability to make the game more strategic. The special attack needs to be charged for a unit to use it. A mana/rage or a turn-based cooldown system is needed in order to balance the game.

The player's units all share the same inventory that the player can apply consumables and equipment items. The item equipped to the unit temporarily increases the unit's stats. The item and the player's currency are lost if a run ends. The items can be upgraded or purchased at shops. It can also be found as loot from treasures and drops from slain enemies with a randomized drop rate. Items are classified by rarity. Upgrading an item raises its rarity by one. Some items are exclusive to some classes of units. For example, swords can only be equipped to melee class units, while staffs can only be equipped to mages. However, every unit can be equipped with armor.

Units can be upgraded using the money collected at a much higher price or by a different currency system. Unlike items and money, these upgrades are carried on even after a run ends.

Units can be bought in shops. Purchasing new units can add more to the strategic element of the game since the player can swap in and out units and make the team with different types of units. Different combinations of units create different play styles. The melee comp tends to be more aggressive and approach combat head-on, while range comp tends to kite back. Building a team with each unit supporting each other well is up to the player. Also, this feature can make every player’s experience of the game different.

At each turn, each player's unit and enemy's unit get to make a move. A move is complete when a unit moves or uses a consumable item. Equipping an item is not considered a move. After moving, consuming, or equipping an item, a unit gets to attack if it is able to. When all the player's units make their move, the player's turn ends. Then, it is the bot's turn to make a move. The run ends when all player's units are killed, or when the boss is slain, or when the player quits the game.

3. Implementation

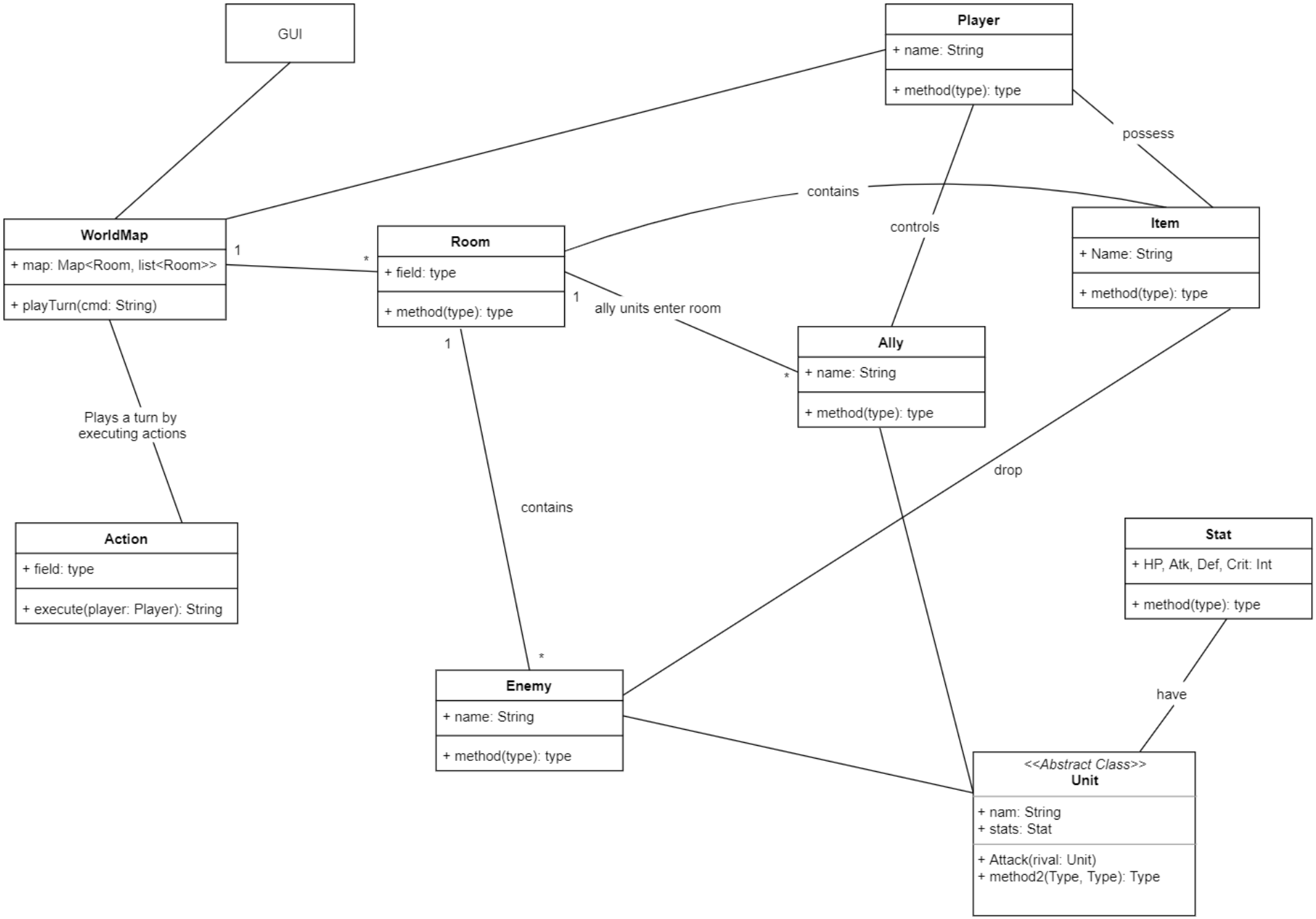
a. UML design

- **WorldMap:** This is basically where the whole progress of a game occurs. Accordingly, WorldMap contains multiple Rooms. Each Room connects to several other ones like a graph. Here, a Player might try to win the game by finding a path from a start Room to an end Room.
- **Room:** A Player needs to complete certain tasks to overcome a Room before reaching other connected Rooms. There are several Room types. One room possibly has various Items for a Player to loot, or Enemy to fight. A player also has a chance to enter a Resting Room where he/she could regenerate his/her health and purchase items.
- **Player:** A class contains all current information about a player. A Player controls Ally units to conquer the WorldMap. He/she could execute certain actions to fight enemies, loot drops, use items, or move from a Room to the other.
- **Unit:** An abstract class defines Ally and Enemy units. A Unit always has certain Stat, including health point, attack damage, defence armors, etc. A Unit might also attack other Units. Ally Units are controlled by a player to fight against Enemy units. Killing an Enemy unit will give a player money and sometimes even drops.
- **Item:** An Item might be collected through fighting, or available in a Room, or bought at a shop. Each Item has a unique ability such as raising certain stats temporarily, regenerating health or mana.
- **GUI:** An object produces a graphical user interface.

The figure on the next page shows more details.

b. External libraries

The project uses Simple and Fast Multimedia Library (SFML) as a 3rd party library for building the game. SFML is free and widely used, especially in developing 2D game worlds. It also supports different types of media, such as sound, font, audio, etc.


GUI
Player
+ name: String
+ method(type): type
possess
contains
controls
WorldMap
+ map: Map<Room, list<Room>>
+ playTurn(cmd: String)
1
*
Room
+ field: type
+ method(type): type
1
ally units enter room
Item
+ Name: String
+ method(type): type
Ally
+ name: String
+ method(type): type
*
Plays a turn by
executing actions
1
drop
contains
Action
+ field: type
+ execute(player: Player): String
Stat
+ HP, Atk, Def, Crit: Int
+ method(type): type
*
Enemy
+ name: String
+ method(type): type
have
<<Abstract Class>>
Unit
+ nam: String
+ stats: Stat
+ Attack(rival: Unit)
+ method2(Type, Type): Type

4. Roles

a. General Role

- Project Manager: In charge of team's Github, checking team's progress and code - Phi
- Members: Code – Mai, Khoa, Taige

b. Implementation Phase Role

- Class World, Room and Item - Khoa
- Class Player and Unit - Taige
- GUI – Mai
- Bot moving algorithm - Phi

5. Planned Schedule

Work package	Task	Deadlines	Total (hours)
1. Planning	1.1 Write project plan document 1.2 Draw UML 1.3 Research for inspirations	30.10.2020	8
2. Rough implementation	2.1 Implement skeletal class structure 2.2 Implement rough classes, functions and attributes	7.11.2020	20

3. Detailed implementation	3.1 Implement functions 3.2 Modify and strengthen the relationship between classes	21.11.2020	40
4. Testing	4.1 Create test files 4.2 Create text GUI	04.12.2020	20
5. GUI	5.1 Implement graphic for the game 5.2 Implement reactive functions	04.12.2020	40-60
6. Demo	6.1 Test runs 6.2 Presentation	11.12.2020	8