OOP Semester 2, 2021
The University of Adelaide

Project Specification - Major Practical

<u>Introduction</u>

Our project is a Role Playing Based (RPG) game which is called 'Dragon Quest.' It is a text-based game where a player battles against different enemies. The player will have a total of at least 2 party members which will fight against the automated enemy assault. The enemies consist of 3 types which include knights, mages and the final boss, a dragon.
The game will incorporate different attributes such as health and magic power for the player and the enemies along with their own unique abilities. The user's party members will have various skills which can be used to damage the enemy, besides normal attack. For example, a player can use some mana to summon 'fireball' skill which will damage both the health and magic of the enemy. The user's party members and the enemy may also have various skills such as summoning a stun to stop the enemy from attacking for one round or deflecting the other party's attack onto itself. The enemy will also possess various skills such as 'aspiro' which deals both health and mana damage to the player. Moreover, there are items like health potions to increase the player's health.
The game will follow a turn-based system where the player and the enemy take turns attacking each other. The goal of the game for the user is to reach and defeat the final boss without losing all health from the enemy attacks. In order to be victorious, users will need to understand the skills of enemies and their own party members.

<u>Design Description</u>

1. Memory allocation from the stack and the heap
    - Arrays - Spawning different number of enemies after the current one has been defeated (examples include the mage class and the knight class)
    - Strings - Names of the characters as well as their attack skills
    - Objects - Classes such as the dragon, knights, mages, hero and other playable characters will use objects extensively in attack skills and health and mana count
2. User Input and Output

    Input:

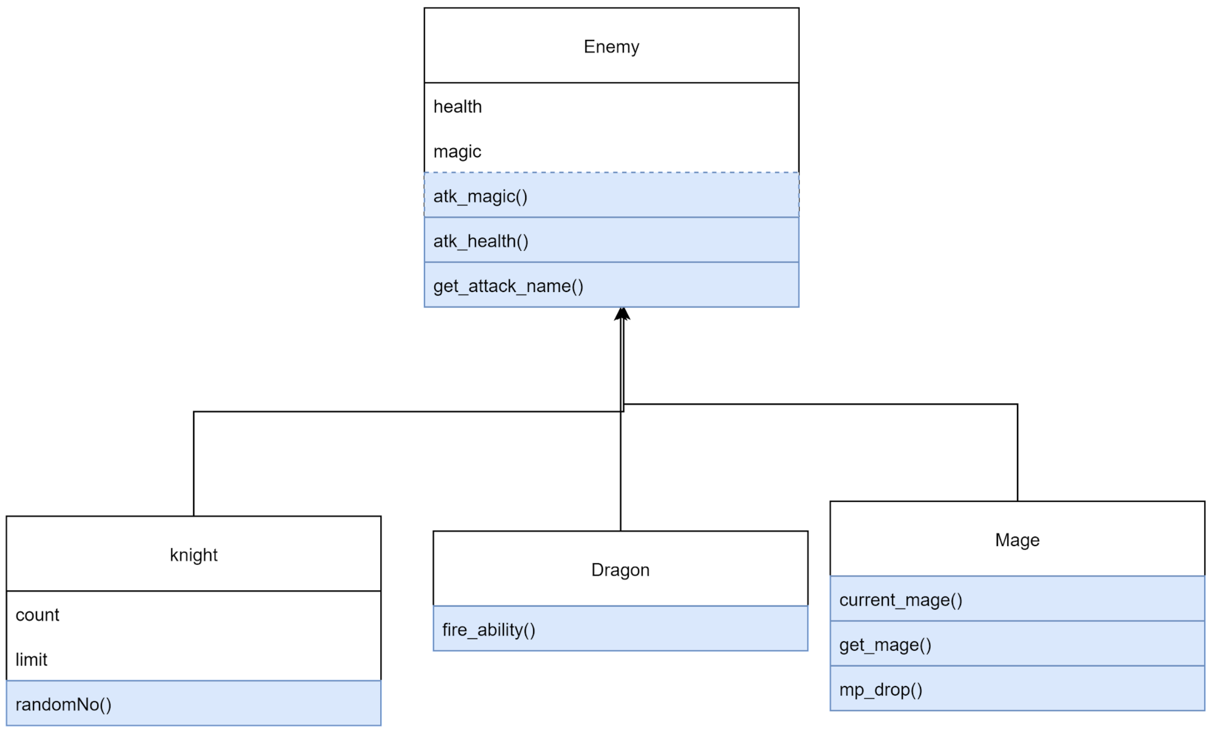
    - Input of different data types - Naming of the characters, skills and inputting the choice of attack will use user input

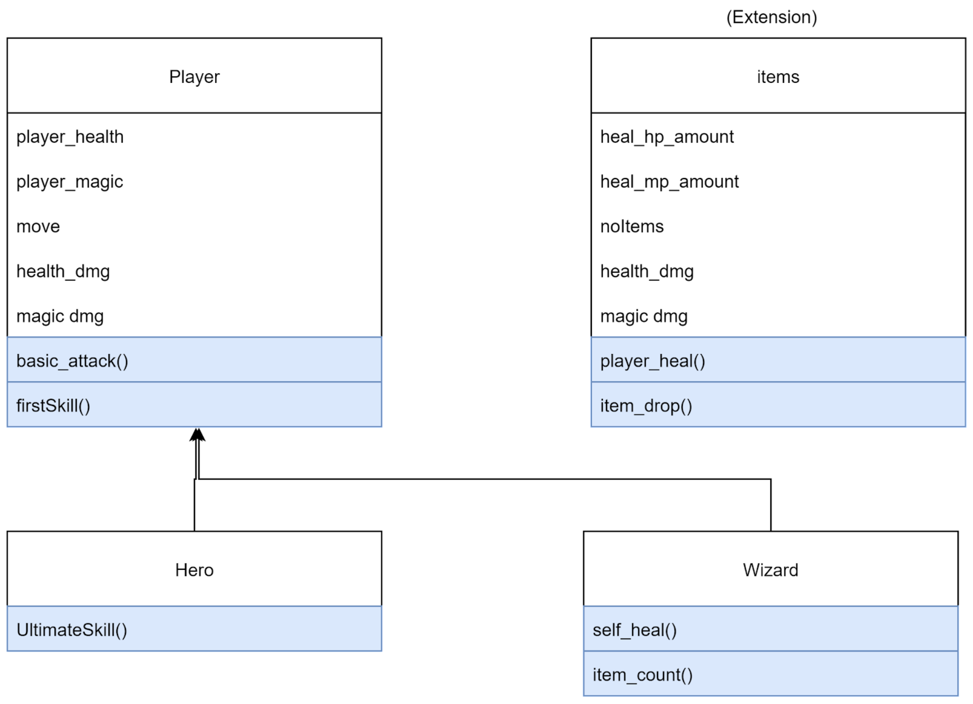
    Output:

    - Output of different types - Plot-setting, display of enemy and player's health and mana stats as well as error messages in user input will be displayed using output
3. Object-oriented programming and design
    - Inheritance - Enemy base class and hero base class

    - Polymorphism - Attack/ability function
    - Abstract Classes - Enemy class (dragon)
    - Virtual functions - Attack damage, mana damage, health and mana stat, names of attack skills
4. Testing
    - Test Inputs / Expected Outputs - Conducted while playing the game with a previously chosen set of input attacks
    - Automated Testing - Includes makefile to automate tests including boundary and extreme values along with invalid inputs such as strings, characters, numbers or symbols in program inputs unsuitable for the case
    - Regression Testing - Conducted through makefiles that will compare the expected output with the obtained output

Class Diagram





Class Description

Enemy class:
The enemy class consists of 3 classes:

- Dragon class (parent class)
- Knight class (child class of dragon)

- Mage class (child class of dragon)

The knight class consists of an array of 3 knights with a similar set stat of health and attack skills. Knights can only be affected by health attacks and they are considered ‘defeated’ when their health reaches 0. Knights can also only damage the playable members with health attacks and cannot affect the player’s magic stats. When all 3 knights are defeated, the mage class is introduced into the game. The mage class consists of an array of 2 mages with an identical set stat of magic and attack skills. Mages can only be affected by magic attacks from the user and are considered ‘defeated’ when their magic stat reaches 0. Mages can only damage the user’s characters’ magic stat. When both the mages are defeated, the dragon class is used in the game as a boss. Dragon can be defeated only when it’s magic and attack skills are both depleted to 0. The dragon can affect the character’s through both magic and health attacks. The enemy class is the parent class of mage, knight and dragon classes.

Player class (Parent, Abstract class):
The player class is a parent class which consists of different party members as the child class such as the hero class.The player class is an abstract class where all the child classes must inherit its properties, functions and methods. All of the child classes will have different health points, magic points and attack damage. The player classes take user as its input and produce an outcome based on that input. If time permits, other player classes will be created like the wizard class, or elf class.

- hero class
  The hero class is the child class of the parent. Common properties will be inherited from the player class like health points, magic points, basic attack and firstSkill. Ultimate skill is unique to the hero class as only the hero can deflect attacks to the enemy.

User Interface:

```
The royal bloodline of knights rush towards the intruder to capture him and his teammates...
Get Ready... Battle begins...
Current knights: 0
Knight 1 entered the battlefield...
Please enter attack: 1.Basic Attack 2.Fireball 3.Ultimate
1

/// basic attack ///
health damage: 50 magic damage: 0

player current hp:100
Knight health: 150
You have been hit with Bao Zakerga, -50 health...
Player hp: 50
Beaten knight 1...
Successfully exterminated 1 knights…

The souls of mortals collected over a millenia plunge towards the hero...
```

```
Get Ready... Battle begins...
Current mages: 0
Mage 1 entered the battlefield...
Please enter attack: 1.Basic Attack 2.Fireball 3.Ultimate
1

/// basic attack ///
health damage: 50 magic damage: 0

You have been hit with Ice Tornado, -100 magic...
Please enter attack: 1.Basic Attack 2.Fireball 3.Ultimate
3

/// Ultimate ///
Reflected Ice Tornado magic damage: -100

Beaten mage 1...
Mage 2 entered the battlefield...
Please enter attack: 1.Basic Attack 2.Fireball 3.Ultimate
2

/// Fireball ///
health damage: 200 magic damage: 200

Beaten mage 2...
Successfully eradicated 2 mages…

The Dragon Prince awakens from its deep slumber...
Dragon King: Rooooaaaaaawwwwrrrrr! Which mortal dares to wake me from my sleep and
cause unrest among my disciples?
You shall pay for your crimes with your life. I shall not let your soul rest in peace and torture it
for a 1000 years in the deepest level of hell!
Get Ready... Battle begins...
Health: 1000
Magic: 1000
Please enter attack: 1.Basic Attack 2.Fireball 3.Ultimate
2

/// basic attack ///
health damage: 200 magic damage: 200

Dragon Prince health: 0
Dragon Prince magic: 1000
Successfully slayed the dragon prince!
```

<u>Code Style</u>

The code for our project 'Dragon Quest' will have all of the file names without any capital alphabets and will use proper indenting equivalent to one tab in functions to ensure proper readability of the code. Each of the functions in a file will consist of comments either within or at the start of the function to explain the functionality of the code, if the coded function is not clear with the given variable names used. Comments will be used to explain if a function is using complex processes or links another function to complete the task it is supposed to process. To ensure that maximum use of comments has been accomplished to explain the complex processes in the code, comments will be written alongside coding. Person 1 will be responsible for the enemy classes (including child classes), while Person 2 will be responsible for the comments in the player classes (including child classes).

<u>Additional Information on Design Specifications</u>

The design for the project will consist of 2 major parent classes, namely the enemy class and the player class. As mentioned before, the player class will consist of the hero class whereas the enemy class will consist of the dragon, mage and knight classes. In addition to this, if the project allows for additional coding an inventory class will be added to the design which will include health and mana potions for the player class to use. All 3 major parent classes will use health and magic variables as the basis for their skills and abilities. The enemy and player classes will also use naming and attack (health and magic) functions for the names of their abilities and their overall damage in terms of health and mana of the enemy. The assessment concepts such as memory allocation, testing, user input and output and object oriented programming will be used in various parts of the code as mentioned right under the 'design description' heading.

<u>Testing Plan</u>

The testing strategy will implement the use of makefiles and automated testing using input files to make sure that the code does not produce an error when faced with unexpected inputs. The input files will include inputs varying from numbers, characters, symbols, string along with invalid inputs. The testing file will read inputs from input files (.txt format) and also compare the output with output (.txt format) files. Testing will be conducted for borderline cases (such that the characters die at exactly 0 health) or close to borderline cases. Furthermore, testing will include invalid input to ensure the code repeatedly asks the user for input.
Unit testing will be conducted for each module/function of the code under the public section under a separate file. Each class will have its corresponding test file such as 'magetest.cpp' for the mage class which will test all of the public functions for the mage class. These files will include a main function which will exhaustively test borderline cases and extreme values for the functions.The inputs and expected outputs will either be monitored manually or using a makefile, depending upon the best possible approach for that specific function.

In addition to this, the makefile will be used to provide a complete set of automated testing using 'input.txt' files and comparing the output with 'output.txt' files. These tests conducted through the makefile will be able to run tests even though the code may be changed. Hence, this concept would cover regression testing as well as possibly using many combined modules to observe changes in the expected output from the obtained output. Hence, integration testing will also be covered through this process. In the end, the overall code will also be tested manually multiple times using various undefined input values including but not limited to fractions, decimals, symbols operators and many more.

Schedule Plan

The goal of the team is to complete the following set of features for the code by the deadline:

- Framework (enemy class and player class class)
- Attacks abilities
- Items (health potion and magic potions) (extension)
- Storyline
- Testing (unit testing and automated testing)
- More characters (extension)
- Music (extension)

If the code is completed before the deadline, the program will be expanded to include different playable characters, items and music.
The entire plan for the project is outlined below along with tasks assigned to different members of the group.

Weeks	Tasks	
	Person 1	Person 2
Week 8 (Planning and Brainstorming)	● Create svn structure ● Discuss possible project structures with group members ● Organise code sharing with group members	● Discuss possible project structures with group members ● Create a shared word document for planning with group members ● Create a class diagram for the project
Week 9 (Basic Coding Planning and Framework)	● Create a basic enemy class framework ● Add enemy skills which will affect the player ● Create a basic main file for integration purposes of enemy and hero class	● Create a basic hero class framework. ● Add player skills which will affect the enemy ● Write a makefile. ● Include various tests in the makefile to ensure early stages of automated testing has been achieved
Week 10 (Extensive Coding and use of polymorphism)	● Add enemy skills which will affect the player (polymorphism) ● Add enemy arrays ● Add items which will assist the player in their battle. ● Determine items dropped by defeating enemies.	● Add player skills which will affect the enemy (polymorphism) ● Add player arrays (if necessary only) ● Automate testing to make testing and debugging a smooth process. ● Integrate the hero class and enemy class together in the main file to

	● Finalise testing plan including possible test cases	ensure the majority of the code framework is complete
Week 11 (Testing and coding extensions)	● Review the code tests and ensure that possible cases including boundary tests and invalid inputs are protected against ● Upload all the files to svn ● Extension: Include music and inventory objects for the player class	● Review the overall code and ensure the code meets criteria requirements while testing the game from an user perspective ● Ensure all design specifications have been met (especially makefile) ● Extension: Add more playable characters to the player class