COMP603 Program Design and Construction

Assignment 1 Team6

Andrew Wang (18045290)

Christian Costa Gomes Jorge (21139803)

Week1 – **Both** - Brainstorm of the idea of options, which classes were going to be needed, the direction and approach of the project.

Christian - class communication diagram.

Week2 – **Andrew** – Draft of the source code to have a starting point. Established basic classes such as the *GameMainMenu*, *Function*, *Player*, *Weapon*, *Potions*.

Christian – GameContent(Super Class)**,** *Waves, Mobs* and assisting on already existing classes named prior with methods.

GameContent(Super Class) - The first and super layer of this project that sets the default scope of the game. Containing *name, weapon, consumable*. It has a *toString* method that describes the default contents, compareTo method that compares the current weapon to other weapons in the inventory, Boolean match method that returns true if the input query is contained in the inventory. Implements Comparable to GameMenu.

GameMenu – Source code of the menu feature for the game (the user is able to open when the 'm' key is pressed and a case break of options to select from. Contains the static final SaveFile, takes the Player class, Mob class, and Map class. Displays progress of player, their inventory, progress on the map, player movement (the way we use to test player movement in the program's current state and will be changed when GUI is implemented. The function to save progress & load progress, and to exit the menu interface.

Function - This class invokes methods to change the state of the game, all the functions listed: saveGameState(), and loadGameState(). It also allows us to test these methods to check that they work outside of in-game play.

Player – This class holds all relative attributes for the player, containing; name, hp, weapon and potion. Weapons and Potions are part of the in-built inventory using Array List. This class also communicates with the GameContent and GameMenu. Functions: isAlive() - used for turnbased attacks against mobs. The getEquippedWeapon is used to calculate damage. GameOver() method occurs when the player's HP is below or equal to 0 as taking damage does not always result in a clean 0, as a player could be low HP and receive lethal damage that was more than enough to kill the player. Inventory methods that add potion (potion counter alongside with it so that the player is unable to use a potion infinitely and checks that it above 0) and weapons and display current inventory. Extends GameContent.

Weapon - Communicates with GameContent and GameMenu. Contains the stats and damage calculation of the weapon. Most important would be the toString method which compares the stats of the weapon selected as how much STR (Strength) it would add. Attempt to use a boolean match query for a search and compare, it is a work in progress, hope to see it implemented in the GUI. Extends GameContent.

Potions – Communicates with GameContent and GameMenu. Contains player, potionName, and hpRegeneration method. Takes the needed identifiers from the super class and adds a fixed hp amount for recovery upon using the item. Once a potion is used it decrements the count by 1. Extends GameContent.

Mobs - Communicates with GameContent and GameMenu. Uses Array List to store items, and random library to generate the chance of loot dropping from mob. Class contains the attributes for mobs such as their hp, name, damage and level (used as monster damage calculation in the future). At the moment the mob's damage is set in a random range of 1-5. Implements the isAlive() but with mobHealth. String Array List to add a random drop chance of a weapon and a potion when the player has successfully slain the mob and adds it to the String Array List. Implements the attack from player to mob and decrease mob hp, not sure about the ETA on implementing the turn-based aspect though. Extends GameContent.

Waves – Communicates with Mobs and Player. It has a WaveName, WaveLevel, currentWave, player and mob. Constructor used the Map class to place mobs on the x and y axis. Method to startWave, checkLoot, checkItems, waveDifficulty: easy, medium, hard, and reset function for when the player dies during a wave.

Week3 – **Andrew** – Added *Maps, MobBattle, SaveFile I/O to txt in Menu & GameState class, Drop class methods,* and methods to existing classes.

Christian – Added *Drops*, methods to existing classes.

Map - Communicates with the GameMenu. Wave class uses Map class to spawn mobs on. Attributes are as follows for the construction of the map; 2-dimensional char array, int x and y axis, int player x and y axis. The constructor of the map also initializes the map so that it can make the player position when the map is selected through the menu. The map is also marked with an X after the player successfully slays the mob on that part of the grid. The order follows the player moving to a new part of the uncleared grid, battle initiates, if player win in the turned-based battle, map is marked clear, item drop rate is chance then player moves to next part in the grid.

MobBattle – Communicates with GameContent, GameMenu and utilizes the scanner for player choice. Uses player, potion, weapon and mob variables. The startBattle() method first checks player hp is above 0 to fight, can't fight if you're dead and spawns an alive mob. Then a choice is

entered then the turn-based loop is iterated until either mob is defeated or player is. Takes damage calculation from mob class and player class. However, the mob has a chance to miss their attack. This class extends GameContent and implements Comparable GameMenu.

Drops – This class mainly is used as a median to check when the mob is dead and if the lootdrop method is achieved when the player receives loot a message is printed.

GameState – Class file to save the game and load the game using BufferedReader, BuffererWriter, FileReader, FileWriter, and a couple of IOException, as a text File I/O. Also implements the Function interface, which override the function methods that will be invoked in **GameMenu.**

Week4 – **Both** – Debugging, Optimizing source code.

Christian – Split classes into packages.

Week 5 – **Both** – Debugging, Optimizing source code. *PlayerMovement* feature. *PlayerMoment* feature to communicate with Map to mark location of player that has been cleared, current location of player in the 2-dimensional array.

Week6 – **Christian** – Source code compiles, Optimizing code, source code loophole checking.
Both – Checking features and functionality of classes, Debugging.