**(THIS IS THE SUBMISSION DOCUMENT)**

**Overview**

Objective

You will play as Hercule Cage, a determined detective on a perilous mission to rescue his kidnapped son from the clutches of a powerful mafia. Journey through the landscape of a jaded Italy that contain dangerous territories like, dense and mysterious forests to urban mob controlled zones having their own adversaries to deal with

To do that you’ll need to hunt for hidden, recover stolen intelligence and battle your way through numerous deadly and unique enemies. Confront the dark side of the Italian underbelly, and make decisions that will shape your path. Every choice counts and your path is reliant on how the player decides to move the story. Navigate a world where loyalty is rare, danger is looming and the wrong move could end everything.

At the heart of this high stakes journey lies Diavolo Mezzasalma, the elusive mafia kingpin who holds your son’s fate in his hands. To reach him, you’ll need to outsmart and overcome the overwhelming odds and stay one step ahead. The mission is clear: track down Mezzasalma, dismantle his operation and save Kai.

Gameplay Instructions

Move – Type the direction you want to go (e.g. north, south, left, right) into the console to move between locations.

View – Use the command view to look around your current area.

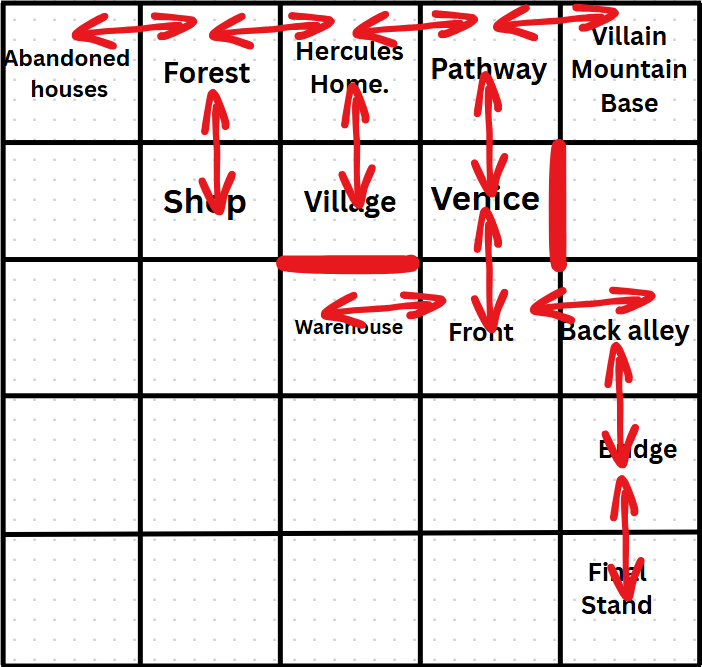
Search – Use search to examine your surroundings for useful items or clues.

Select – Use ab to make a choice or interact with certain objects.

Use – Type use followed by an item name to activate it (e.g. use lockpick).

Map – Follow your map to keep track of where you've been and where to go next.

Map (see next page)



Overview of each class created OR completed Javadoc outlining what each class does

GameItems class acts as the game’s inventory system, where each item in the player inventory is represented by a GameItems object. This class provides the structure needed to create, update and display those items reliably. GameItems is a basic blueprint for creating, accessing and updating in-game items.

GameWeapons class acts to represent the weapons in the game, storing basic information about each weapon such as its name, description and damage value. This allows for players to collect, equip and use various weapons to fight enemies.

The ConsoleFormating class is a utility class that provides simplified, shorthand methods for printing to the console. It wraps standard print functions and line break functionality into more and reusable static methods. Additionally it makes it easier to update or enhance output formatting in the future without having to modify every print statement.

The Enemy class represents enemy characters in the game, holding basic attributes like name, health and two different attack damage values. It allows the program to easily create, store and interact with enemies during battles or encounters.

The purpose of the Npc class is to handle character interactions and story driven dialogues that enhance immersion and narrative progression. It centralizes all the NPC behaviour and dialogue logic, allowing the game to present characters with distinct personalities, offering branching decisions and creating example shops to test a basic gameplay feature.

The Fighting class provides the mechanics between a player and enemies in the games, simulating a turn based battle system where the player can choose to fight, heal or flee. Both the player and enemy take actions that are influenced by random chance which can cause critical hits or misses.

The Player class models the main character of the game in the program, managing the player’s health, weapons, inventory and state. It includes methods for searching items, managing inventory and interacting with systems like combat and navigation. Centralizes all attributes and behaviours of the player.

Navigation class manages the player’s movement on a 5x5 grid which acts as the world map and determines the player;s current location within the games. It handles directional input through inputting letters which represent North, East, South and West and updates accordingly. Whilst also providing context for nearby locations.

Algorithms for each method created OR completed Javadoc outlining what each method

does

Method to system print messages with quicker call method to do so

public static void print(String text) {

System.out.print(text);

}

Method to system print nextline messages with quicker call method to do so

public static void println(String text) {

System.out.println(text);

}

Method to creates a line break

public static void lineBreak() {

System.out.println();

}

Method that creates the name, health and damage the enemy can deal

public Enemy(String name, int health, int AttDam1, int AttDam2) {

this.name = name;

this.health = health;

this.AttDam1 = AttDam1;

this.AttDam2 = AttDam2;

Method that creates the health, 2 attack variables for the enemy

public static void Fighting(Player player, Enemy enemy, Npc npc){

Enemy[] enemies = {

new Enemy("Henchmen", 100, 3, 10),

new Enemy("Claw Enforcer", 180, 5, 25),

new Enemy("Twin Blade Enforcer", 250, 6, 50),

new Enemy("Diavolo Meezasalma", 300, 9, 50)

};

Method that creates variables such as item names, descriptions, values, and types

public GameItems(String ItemName, String Description, int Value, String Type) {

this.ItemName = ItemName;

this.Description = Description;

this.Value = Value;

this.Type = Type;

}

Method that creates weapon names, descriptions and damage

public GameWeapons(String Weaponname, String Description, int Damage) {

this.Weaponname = Weaponname;

this.Description = Description;

this.Damage = Damage;

}

Method that creates the names of all locations on the map

private static final String[] Locations = {

"AbandonedHouses", "Forest", "Hercules Home", "Pathway", "VillainMountainBase",

"Unknown", "Shop", "Village", "Venice", "Unknown",

"Unknown", "Unknown", "Warehouse", "Front", "BackAlley",

"Unknown", "Unknown", "BridgeRoute", "Unknown", "Bridge",

"Unknown", "Unknown", "Unknown", "Unknown", "EndGame"

};

public static int getLocationIndex() {

PX = CurrentLocation % 5;

PY = CurrentLocation / 5;

IndexLocation = PY \* 5 + PX;

return LocationIndex[IndexLocation];

}

private static int[] LocationIndex = {

1, 1, 1, 1, 3,

0, 2, 1, 1, 0,

0, 0, 5, 4, 1,

0, 0, 0, 0, 6,

0, 0, 0, 0, 1

};

private static final boolean[] LocationsVisited = {

true, false, false, false, false,

false, false, false, false, false,

false, false, false, false, false,

false, false, false, false, false,

false, false, false, false, false

};

private static final boolean[] CombatLocations = {

false, true, false, false, true,

false, false, false, false, false,

false, false, false, false, false,

false, false, false, false, true,

false, false, false, false, true

};

public static int getCombatIndex() {

PX = CurrentLocation % 5;

PY = CurrentLocation / 5;

IndexLocation = PY \* 5 + PX;

return CombatIndex[IndexLocation];

}

public static int[] CombatIndex = {

0, 1, 0, 0, 0,

0, 0, 0, 0, 0,

0, 0, 0, 0, 0,

0, 0, 0, 0, 0,

0, 0, 0, 0, 4

};

public static int getSearchIndex() {

PX = CurrentLocation % 5;

PY = CurrentLocation / 5;

IndexLocation = PY \* 5 + PX;

return SearchIndex[IndexLocation];

}

public static int[] SearchIndex = {

2, 1, 1, 3, 4,

0, 0, 1, 1, 0,

0, 0, 0, 5, 7,

0, 0, 0, 0, 8,

0, 0, 0, 0, 9

};

public static int getUseIndex() {

PX = CurrentLocation % 5;

PY = CurrentLocation / 5;

IndexLocation = PY \* 5 + PX;

return UseIndex[IndexLocation];

}

private static int[] UseIndex = {

0, 0, 0, 0, 1,

0, 0, 0, 0, 0,

0, 0, 2, 0, 3,

0, 0, 0, 0, 0,

0, 0, 0, 0, 0

};

A continuous bug report – including how each issue was overcome

Allocation of duties for each team member

Thomas: Coding

Dihain: Coding + Story Building + Dialogue Building

Derin: Coding + Story Building + Dialogue Building