# PPA Assignment 2

Author: Yeshuai Cui

Student number: K1924221

King's College London

Due date: 2019/11/26

## Introduction

This game consists of 11 rooms. The player is located in the first room when game starts and player will win if he entered the final room. All the items discarded will dispear on the floor, and there're no limitation on item numbers in rooms contain them .Dwarf and old man are npcs that can walk through rooms, you can trade with dwarf to get a ring, old man will only tell you something. All pickable items have their own weight, player is required to manage the total weight of his backpack. This game also have a combat system and enemies are located in their rooms, player needs to take care of whether or not he has equiped a weapon and his HP.

## Base Tasks

1. All the rooms are created and room features, including items and enemies are added in the **createRooms()** method in **Game** class
2. The player can walk through locations by using **goRoom()** method in Game class, two more direction "up", "down" is added.
3. There're 5 rooms contains 4 distict pickable items and 1 unpickable item. You can pick items by using *pick* or *equip* command (implemented by using **take()** method in **Player** class). All the items discarded will disappear on the floor (inplemented by using **discard()** method in **Game** class) and each room contains pickable items will have infinite number of them, this will avoid player cannot pick items needed anymore and ensures the corrispondence of room discription and items in it. **discard()** method will check if the command have second word by using **hasSecondWord()** method in **Command** class. **take()** method will automatically check which item is in this room and then added to player's backpack.

4. **Backpack** class stores all the items can be recognized by the game and whether each item is in it. Each item has its own weight and backpack has its weight limit(30kg). This class contains methods for items are added or deleted from it.
5. When player enters the final room, he wins and a message is displayed in terminal and the program terminates.
6. **back()** method is in **Player** class. For each move of player, **setLocation()** method in **Player** class is called and results in change in *currentRoom* field and is recorded in an ArrayList *footprint.* When **back()** method is called it removes the latest item in *footprint* and decrements the *index,* set *currentRoom* field to the previous room player was in.
7. New added commands: *"equip"*, "*take"* and "*fill"* are almost same, the difference is the item they added to player's backpack. "*explode"* and "*unlock"* are also similar to each other. They check player's current location and items player has; a new exit is created when all conditions are satisfied. *"look"* is used to display the content on a billboard, billboard will give some hints to player when playing the game. "*fight"* is part of combat system. "*drink"* works like discard and it also sets player's hp to maximum value (3).

## ChallengeTasks

1. There are two characters can move around the maze: dwarf and old man. Their position are represented as **Room** objects in **Game** class. At the end of **goRoom()** method, *dwarfRoom* and *oldmanRoom* are updated to a new location by using **characterMove()** method. This method uses a **getExitList()** method to get all available directions for the room, then uses **Random** object and **.size()** method to generate a random number correspond to all element in *directionList* and return a random **Room** object in the end. If player and npcs are in the same room, a message is displayed on terminal and there's some interaction with dwarf.

2. The parser is extended by adding an inner *if* statement in **getCommand()** method in **Parser** class. **tokenizer.next()** will receive the next word in the input. Three-word command is used when player meets dwarf and trade drink with him.
3. Transporter Room is set both way exit to first room, this is because if only set exits to the random room, npcs may be trapped in that room. This function is accomplished by using **goRandom()** method in **Game** class. *possibleRoom* is a ArrayList that contains all the rooms in the maze except the final room and random room, so player cannot win by simply go to transporter room.
4. Combat system is added to this game. Player's weapon, HP and ring are part of this system. Player can only fight when he has a weapon, drink is used to recover player and ring can halve damage player received when fighting. Player can only use *"back"* to go back when there's an enemy and rooms' description will be upgrated when an enemy is defeated.

## Code Quality

Coupling: **goRandom()** method is in **Game** class instead of in **Player** class. Since all rooms are declared and initialized in **Game** class, if **goRandom** is in **Player** class, it will need access to **Game** class. Any pair of classes in this program diagram is one way or another.
Cohesion: The location and status of player is in a separate **Player** class instead of in **Game** class. Player's location is a *currentRoom* field in **Player** class instead of in Game. **Backpack** and **Item** class are separated.
responsibility-driven design: **Player** class stores data for player's location, so there's a method **setLocation()** changes the field *currentRoom*. **add()** method and **delete()** are in **Backpack** class instead of in **Player** class since **Backpack** stores all the item and its own weight limit.

Maintainability: For **goRandom()** method in **Game** class, elements in ArrayList *possibleRoom* is added by using **createList()** instead of instead of add all elements in **goRandom()**, if list of rooms is need later, programmer can just initialize a list and use the function, this also get rid of code copy-paste. The same as **add()** and **delete()** in **Backpack** class.

## walkthrough of maze

go down; take; back; go west; equip; back; go east; look; go west;go north; fill; unlock; go north; fight; fight; drink; fight; fight; go east; discard slate; take; go west;
go south; discard sword; fill; drink; go south; go west; equip; go east; go east; go east; explode; go north; fight; fight; go north

## Known bugs

For one game object, if use play method twice, the program will continue from last play. If use **play()** method again from a **Game** object that has already winned, if will continue from the last room player was in before win. If player wants to play the game again after winning, he will need to create a new **Game** object.