Project's Documentation

Programming the video game version of the board game "SabOOteurs"

MU4RBI01 : Python

Practical Trainer : Philippe GAUTHIER

06/01/2023

Students :
Leo Bellali, 28609955
Yousri Aboudaoud, 28712992
Sofiane Ouali, 3802574

Degree level : M1 ISI | CMI EEA

"*Python is an experiment in how much freedom programmers need. Too much freedom and nobody can read another's code; too little and expressiveness is endangered.*"
(*Guido van Rossum*)

I. INTRODUCTION

The hereby report is aimed to make the code of the project as explicit as possible, for that purpose we will use a color code in which :

- Classes will be written in bold red characters, for instance the class **Player**
- Attributes will be written in bold blue characters, for instance the attribute **role** of **Player**
- Methods will be written in bold green characters, for instance the **printPlayerHand** method that will use **playerHand** of **Player**

II. The code

1) Player

The class **Player** will be used to create the player and provide it with the appropriate size for **playerHand** depending on the total of players, and assign him/here with a randomly chosen role. His/Her **playerHand** will be displayed with **printPlayerHand** and each time the **Player** has to draw a card it will be via **drawAcard.**

2) The elements of the board game

The most important elements are :

- The abstract class **Card** from which all the cards shall inherit the common attributes **cardType, cardName.**
- The **CardGallery** referring to the path card (inheriting from **Card**), its most relevant attributes are the booleans **cN, cE, cW, cS** referring (respectively) to the connection with each direction North, East, West, South, the hereby class has been implemented in this matter with the aim of a much easier management of the connections between the **CardGallery** (which is a central mechanism of the board game). The methods **addToGUI, poseOk** will respectively add the said **CardGallery** and add it to the user interface while taking into consideration the aforementioned directions, as for the second method it shall verify should the chosen position of the card is convenient providing the gallery (ie. The board).
- All the classes inheriting from **CardGallery** are distinguishable by their name starting with **CardGallery** and ending with the direction they connect with, for instance **CardGalleryE** will connect with the east direction and have the attribute **cE** set to **True** whereas the others are set to **False.** The only exception for this is **CardGalleryEnd** which has the additional attribute **isGoal** which will be set to **True** should the card correspond to the winning one.
- The other class inheriting from **Card** is **Card Action** having the attribute **cardType** equal to “Action”. From this class shall inherit the classes related to the breaking or restoration of the tools present in the **toolKit** of **Player** setting (depending on the index of each tool) them to either **True** or **False**, the breaking cards will be distinguishable by the additional “broken” in their names, for instance **CardActionLight** will repair the torch and **CardActionLightBroken** will break it (NB: Each class has a method that will add it to the GUI depending on its action). Among the subclasses of **Card Action** is **Card ActionRockFall** This card shall replace whatsoever **CardGallery** with a **CardGalleryVoid**
- **CardActionSecretPlan** This card will be activated in the main loop of the game

Note Section: Whenever a **Player** is destroyed all his/hers **Card** are going to be destroyed. Hence the composition nexus that relates between them.

3) Game management

The game management is done via the classes:

- **Game**, the latter will set the number of saboteurs, searchers and the number of cards in each hand. It will add the players to a list, and **name2rank** will return the index of a player in the aforementioned list. As for **randomizeRole** it will assign to each **Player** a role in a random way. Finally, **startGame** will set the length of the hand and will draw cards for each player.
- **GameBoard**, will create a **deck** by filling it with cards in a random way and selects a **CardGalleryEnd** as the goal, it will manage the cards positions to ensure that even if a **CardGallery** is put in the external region of the cave it'll then expand the map. It will also check if there's a winner, and finally it will print the cave.

4) User Interface

The user interface is handled in two separate ways, The file "SabOOteur.py" which will use the function welcome of either the :

- "SabOOteurText" that will trigger the main loop (the infinite while(**True**), where the video game functions) as for the loop present in the function play it will be exited as soon as the attribute isGoal is set to **True**. This file will handle the changes in the game.