CS207 Cryptogram Assignment Status Report

Introduction

The aim of this report is to concisely and accurately describe the functionality of the submitted program, alongside presenting all the unit testing done which accompanies it.

Implemented Functionality and Model

To begin with, the functionality which is implemented in our program is based on the Model, View, Controller Design Pattern. The classes which implement the Model, in our case are the **Player, Players, Cryptogram, CryptogramFactory, LetterCryptogram and NumberCryptogram.** The **Player** class holds all the essential fields and methods enabling it to function in unison with the other classes. In addition, it has 2 constructors – one for loading an already existing player and one for creating a new one. The ladder of which calls a method that creates a new save file, holding the new information. The class **Players** uses an *Array List* of type *Player,* which holds all the instances of the **Player.** The class also has methods which allow the user to *Add, Remove,* and *Find* players, alongside returning lists of the players' accuracies, times, cryptograms played and completed. Our Cryptograms utilize the Factory Model, using a **CryptogramFactory.** The class **CryptogramFactory** in our project has the task of reading strings from a file, in addition to having a method that allows the user to choose in which way they would want the aforementioned string to be encrypted – with letters or numbers. That will subsequently create an instance of either the **LetterCryptogram** or **NumberCryptogram** class, which both inherit from the **Cryptogram** class.

Implemented Functionality and Controller

Secondly, in the presented project, our **Game** class takes the role of the controller. It has methods for saving the game, putting both the encrypted phrase, as well as the non-encrypted one in a *.txt* file. Furthermore, it also uses the method **playGame()**, to instantiate a GUI. Lastly, it also provides the method used for loading the game back into the program, once it has been saved.

Implemented Functionality and View

The class to which the *View* part of the design pattern corresponds to is our **CryptogramGUI**. The **GUI** part works exceptionally well with all errors and exception being handled accordingly. It starts with a little overview stating what the product is about and then we move to the first critical part of the **GUI** which is authentication. Authentication requires the user to either log into an account which has previously been used and saved in a file or just create a new one which is automatically saved in a file after creation. Next is the log in part which is simple it only asks for a username, due to the fact that the game is unplayable without a user. After logging in the player finds himself in the main menu where he could choose from the following: **Scoreboard, New Game, Load Game, Credits and Exit Game.** He could choose to view Statistics play, load a game or exit the game. If he creates a new game or loads a game, he finds himself on the playing field. There the player can play the cryptogram which can be of type letter or number. The player can also get hints, view frequencies and enter letters. Also he could undo the last letter he entered and reset or save the current cryptogram if he chooses to.

Unit Testing

Last but not least, our program has 3 classes devoted to unit testing. The first one of which is the **PlayerTest**, having 7 JUnit tests, this verifying that **Players** works as intended. The second test class we have is the **CryptogramTest** class which has 5 tests, checking if the mapping works correctly, as well as that the frequencies hold correct values. Finally, the **CryptogramFactoryTest,** tests the creation of different cryptograms, as well as the reading of files from the list of phrases.