Algonquin College Logo

# SCHOOL OF ADVANCED TECHNOLOGY

### ICT - Applications & Programming

### Computer Engineering Technology – Computing Science



A21

Game MVC

Team:

[Thomas Stanley] - Id: [**041-037-136**]

Picross Proposal

***This template is suggested (not mandatory) to answer A21 Specification.***

|  |  |
| --- | --- |
| **Part**  **1** | **GUI Definition** |

* 1. **MVC Details**

**MVC Components:**

**Model** Class: Model – Object: “Model” (contains array and text file objects used by program)

**View** Element: UserInterface– Object: “UserInterface” (Contains all UI features, calls new controller object within)

**Controller** Class: Controller – Object: “Controller” (Contains all button and program functionality)

Main would look as follows.

Model myModel = new Model();

UserInterface UI = new UserInterface(mymodel);

* 1. **View Component**

***Splash Screen***

**Example** (from vision “top-down”)

Class: JFrame – Object: “splashWindow”

→ Class: JPanel → Object: “splashPanel”

Label: Object: “JLabel” (Will contain the splash screen artwork)

***Main Frame***

**Example** (from vision “top-down”

Class: JFrame – Object: “frame”

→ Class: JPanel → Object: “topPanel”, “clockpanel”, “logoPanel”, “topNums”, “leftPanel”, “centerPanel”, “rightPanel”

→ Class: JButtons → Objects: “GridButton”, “Submit”, etc.

→ Class: JLabel → Objects: “LeftLabel”(label 1-5) , “topNums”(label 1-5), “logoLabel”, “clockLabel”

* Class: JMenuBar -> “topMenu” (To be used to implement Colour Picker)
  1. **Controller Component**

*Describe aspects of your controller using, for example, one unique action command. Create the “map” to define functions with actions. This activity is to plan what will happen in your action. Ex:*

*Controller extends JFrame()*

Methods: designButtons(panel, frame) (contains Actionlister for ‘submit’ button in ‘design’ mode.)

Countdown(label) (contains actionlistener for timer. ActionListener ends game at 0:00 secs)

Button: Grid Button 1-25

**Button Details**

Object: “grid button” (method implemented in each grid button to create pattern array)

Event -> actionPerformed

Parameters -> (int x, int y)

Method Content-> grid[x][y]=1;

**Design Submit Button**

Object: “Design Submit button” (Button is used is design mode to pass array to play method in **Model Component,** passes the grid and true value, to indicate the user designed a grid.

Event -> actionPerformed -> method: playMode(designGrid, true);

**Design Save button**

Object: “Design Save Button”

Event -> actionPerformed

Parameters -> grid[][]

**Play mode Submit Button**

Object: “Play Submit Button” (Button is used in play mode to submit your solution, takes in both playerGrid and correctGrid to compare. )

Event -> actionPerformed -> Method: isEqual(playergrid, correctGrid);

**BSave**

Object: BSave (Saves array to text file)

Event -> actionPerformed -> method: gridSave(grid);

**BLoad**

Object: BLoad (Loads array from text file)

Event -> actionPerformed

*Finally, what is your idea to define the model to be used in a “default” (randomized) game.*

1.4 **Model Component**

The ‘default’ will be determined if a user starts ‘Play” mode without designing a grid beforehand in ‘Design’ mode.

This will be done through the Boolean value ‘isDesigned’, which is passed to the method ‘mapGrid’ to determine which 2d array should be returned for use in the game board.

**Variables**

DefaultGrid[][]: (For if the game grid is not designed).

playerGrid[][]: (Grid to be used if the user designs a grid or loads one in from text file).

emptyGrid[][]: (To be used in controller class for playMode() as the active game board, comparing to either playerGrid[][] or

defaultGrid[][]).

**Methods**

readFile(): (reads text file with pattern in it)

saveFile(): (saves playermade grid to a file)

int[][] mapGrid(designGrid, defaultGrid): (this method returns a grid to be played with in play mode. Uses the user designed grid, and if the user has not created a grid, it returns a default grid.)

|  |  |
| --- | --- |
| **Part**  **2** | **Implementation Design** |

* 1. **Game Evolution**

The major difference to the program structure between this assignment and the last, is that my classes were split up in a way that did not implement any design pattern, and so refactoring my classes to better fit the MVC framework is the main difference.

The reason I need to do this adjustment is so that the program is a lot cleaner in its construction, which leads to better security in its variables, and ease of scalability as the program increases in size and scope.

The colour picker is also a new feature to be added to the View element of the MVC framework. This will allow for greater user customisation.

Another difference is the added element of persistence in the form of a few new features such as high score, user identification, time, and the ability to save and load picross grids.

* 1. **Others DP**

I think it would be possible to implement a Factory Method for this assignment, seeing as how we have currently an MVC design pattern, we could take out the Model element, then use the Controller and View elements for use in other ways. We could essentially swap out the Model class for another and use the remaining elements to create a program which functions entirely differently.

**References**

Algonquin College

Winter, 2022