Algonquin College Logo

# SCHOOL OF ADVANCED TECHNOLOGY

### ICT - Applications & Programming

### Computer Engineering Technology – Computing Science



A21

Game MVC

Team:

[Ngoc Phuong Khanh Le] - Id: [041004318] / [Dan McCue] - Id: [040774626]

NumPuz Proposal

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

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

* 1. **MVC Details**

*Describe the way you can define the MVC components in your game.*

**Example** (from vision “top-down”)

Model Class: GameModel – Object: “model” (POJO/ Plain Java Old Object)

View Element: GameView– Object: “view” (extends JFrame)

Controller Class: GameController– Object: “controller” (Plain Java Old Object responsible for all Actions)

Main Class: GameApp – Object: x (Declare objects of model, view, controller)

…

* 1. **View Component**

*Describe how your interface should be organized using new components. Show the idea about your “top-down” organization.*

* + - ***Example****:*

**Example** (from vision “top-down”)

Class: JFrame – Object: “frame”

→ Class: JPanel → Object: “playMenu”, “designMenu” “game”, “grids”, “textInput”,

→ Class: JButtons → Objects: “save”, “load”, “start”, “reset”, “show”, “hide”, “setText”

→ Class: GameModel → Objects: “model”

→ Class: JComoBox → Objects: “dim”, “format”

→ Class: JRadioButton → Objects: “design”, “play”

→ Class: JTextField → Objects: “point”, “timer”, “textField”

→ Class: JTextArea → Objects: “detail”

→ Class: JButton[][] → Objects: “matrix”

→ Class: GameModel → Objects: “model”

→ Class: Integer[] → Objects: “shuffle”

→ Class: int → Objects: “oldDim”

…

* ***Note****: The professor interface continues being a proposal. Focus on your ideas using the best user experience.*
  1. **Controller Component**

*Describe aspects of your controller using, for example, one unique action command. Create the “map” to define functions with actions.*

**Example**

Object: “view” (POJO for GameView class)

view.getStartBtn() → Event: actionPerformed → method: actionPlayMode()

view.getDesignBtn() → Event: actionPerformed → method: actionDesignedMode()

view.getSetTextBtn() → Event: actionPerformed → method: actionSetButton()

view.isTypeNum() → Event: actionPerformed → method: actionNumSelected()

view.showSolBtn() → Event: actionPerformed → method: actionShowSol()

view.getLoadBtn() → Event: actionPerformed → method: actionLoadConfig()

view.getSaveBtn() → Event: actionPerformed → method: actionSaveConfig()

view.getColorBtn() → Event: actionPerformed → method: actionChooseColor()

* 1. **Model Component**

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

**Example**

Data structure used:

→ Values: solution → method: updateSolutions()

→ Values: shuffleNum → method: updateShuffleNum()

→ Values: shuffleText → method: updateShuffleText()

→ Values: timer → method: updateTimer()

→ Values: points → method: updatePoint()

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

* 1. **Game Evolution**
  + *Considering this new model, explain:*
    - *What are the differences between the original proposal (A11) and the current project to be developed (A21).*
* Differences between our A11 and A21 is that in A11, we put everything in the same class (GUI, logic, etc.) while in this A21, we separated classes depending on MVC Design Pattern (GameView, GameModel, GameController, GameApp)
* Differences between our A11 and A21 is that in A11, we put everything in the same class (GUI, logic, etc.) while in this A21, we separated classes depending on MVC Design Pattern (GameView, GameModel, GameController, GameApp)
* We changed from menu tabs (JTabbedPane) to drop down menus (JMenuBar)
* Added a splash screen at the start of the game for some seconds to display the Game image/icon
* Adding a file browser to load and save configurations.
* Changed the function panel to implement different JPanel for play mode and design mode
  + - *If so, explain why you need to do some adjustments.*
* We need to make some adjustments because it will bring clarification to our codes, easy to maintain and reuse in the future.
* I need to include menu because I want to provide different access to different functionalities
* I need to use Color because I need accessibility that I can provide for the users

We changed the design and added classes to make a more object-oriented approach to the code

* We changed from tab panes to JMenuBar to have access to a drop-down menu to provide different access to different functionalities. (Ex: Game menu -> exit, new game, settings; Help menu -> about, colors)
* We added the splash screen because it is part of the design implementation.
* To load and save configurations set by the user or timer data, points data.
* We changed the view based on the mode to have functions related to the modes Design and Play.
  1. **Others DP**
     + *Define (at least one) additional DP that you could use in your Game application.*

Observer Pattern, Mediator Pattern, Composite, Flyweight, Façade, etc. I will choose Observer as an additional DP.

* + *Explain what this DP is and the reason why it could be recommended.*
* “Observe Pattern is one of the behavioral design patterns. Observer design pattern is useful when you are interested in the state of an object and want to get notified whenever there is any change.”
* In my game panel which contains my menu for functionalities and grids for buttons, both communicate with each other to change accordingly, specifically, grids will change depending on the state of function panel. In this case, grids panel is interested in the state that function panels might change. Observer DP can help grids panel to register its components to Interface of its Subject (which is function panels) in case if any components in function panels change, grid can change as well. This DP could be recommended because this game functionalities surrounds these two panels and their communication. Creating interfaces that can communicate between the two without calling the other immediately can help the process to be more effective.

**References**

[Observer Design Pattern in Java | DigitalOcean](https://www.digitalocean.com/community/tutorials/observer-design-pattern-in-java)

Algonquin College

Fall, 2022