Algonquin College Logo

# SCHOOL OF ADVANCED TECHNOLOGY

### ICT - Applications & Programming

### Computer Engineering Technology – Computing Science



A21

Game MVC

Team:

Jawad Awada - : 041032551/ Preston Barre - Id: 041032867

Picross Proposal

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

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

*This template is very similar to your A11, but going deeper with the components and methods description and dividing them into the MVC components.*

* 1. **MVC Details**

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

**Our Example** (from vision “top-down”)

**Model Class: A12UI – Object “gameModel” (Main game interface)**

**View Element: GameView – Object : gameView (extends Jframe implments newGameClicked)**

**Controller Class: UX – Object: “userExperience” (responsible for all actions)**

**Example** (from vision “top-down”)

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

**View** Element: GameView – Object: “gameView” (extends JFrame implements GameController)

**Controller** Class: GameController – Object: “gameController” (responsible for all Actions) …

* 1. **View Component**

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

*\** ***TIP****: Review the components defined in the A11 and put them in the proper hierarchy (as you done in the A12).*

**Our Example** (from vision “top-down”)

Class: JFrame – Object: “A12UI”

-> Class: JPanel – Object: “markPanel”

-> Class: JCheckBox – Object: “markCheckbox”

-> Class: JPanel – Object: “topPanel”

-> Class: JLabel – Objects: “label”

-> Class: JPanel – Object: “trPanel”

-> Class: JLabel – Object: “label”

-> Class: JPanel – Object: “northPanel”

-> Class: JPanel – Object: “markPanel”

-> Class: JCheckBox – Object: “markCheckbox”

-> Class: JPanel – Object: “topPanel”

-> Class: JLabel – Objects: “label”

-> Class: JPanel – Object: “trPanel”

-> Class: JLabel – Object: “label”

-> Class: JPanel – Object: “leftPanel”

-> Class: JLabel – Objects: “lLabel”

-> Class: JPanel – Object: “centerPanel”

-> Class: JButton – Objects: “button”

-> Class: JPanel – Object: “rightPanel”

-> Class: JPanel – Object: “innerRightPanel”

-> Class: JPanel – Object: “controPanel”

-> Class: JLabel – Object: “controLabel”

-> Class: JPanel – Object: “bottomRightPanel”

-> Class: JLabel – Object: “restartLabel”

-> JLabel – Object: “timeLabel”

-> JLabel – Object: “newGameLabel”

-> Class: JPanel – Object: “pointsPanel”

-> JLabel – Object: “pointsLabel”

* + - ***Example (Main Frame)****:*

**Example** (from vision “top-down”)

Class: JFrame – Object: “GameFrame”

→ Class: JPanel → Object: “GameuBoard”

→ Class: JButtons → Objects: “BSave”, “BLoad”, etc.

→ Class: JLabel → Objects: “LabOperation”, “LabName”, etc.

…

* ***Note****: The professor interface continues being a proposal. Focus on your ideas using the best user experience.*
  + - ***Example (Splash)****:*

**Example** (from vision “top-down”)

Class: JFrame – Object: “splashWindow”

→ Class: JPanel → Object: “splashPanel”

…

* 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:*

**OUR EXAMPLE**

Object: “New Game”

* Event: newGameClicked -> method: resetGame()

Etc.

// Future implementations - events (in controller)

**class** Controller **implements** ActionListener {

**public** **void** newGameClicked (ActionEvent e) {

Object eventSource = e.getSource();

// Button (New Game)

**if** (eventSource == *newGameClicked*)

resetGame();

// Others…

|

|

**Example**

Object: “aboutButton”

→ Event: actionPerformed → method: showAbout()

Etc.

// Future implementations - events (in controller)

**class** Controller **implements** ActionListener {

**public** **void** actionPerformed(ActionEvent e) {

Object eventSource = e.getSource();

// Button (about)

**if** (eventSource == *aboutButton*)

showAbout();

// Others…

|

|

* 1. **Model Component**

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

**Our Example:**

Data structure used:

Values: game

Method: welcomePlayer()

Properties:

dimension (int)

Methods related:

changeDimension()

getDimension()

Player (class)

Property: name (String)

Methods related:

changeDimension()

getDimension()

Properties:

centerPanel (JPanel)

Layout: GridLayout(x, y)

Buttons:

JButton

Properties:

Preferred (int size) size: Dimension(x, y)

/\*

Using a welcomePlayer() method and the two primary characteristics of dimension and Player, this data structure describes a game. The getDimension() function and the changeDimension() method both let you to modify the integer value of the dimension property. The getName() function and changeName() method are both available for the name property of the Player class, which can be updated.

Moreover, a JPanel with a GridLayout layout of x rows and y columns is represented by the centerPanel attribute. Each button in the grid is created using the GridLayout and has the requested size of Dimension (x, y).

\*/

**Example**

Data structure used:

→ Values: game → method: updateData()

→ Properties: dimension (int) → methods related: changeDimension(), getDimension(), …

→ Properties: Player (class)

→ Property: name (String) → methods related: changeDimension(), getDimension(),

|  |  |
| --- | --- |
| **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).*
    - *If so, explain why you need to do some adjustments.*

Original proposal would have no option for the user to change the grid size, different colours, added a mark panel, changed various positions of certain panels and added a feature so that the player can enter their name and it will show up on the screen welcoming them.

* 1. **Others DP**
     + *Define (at least one) additional DP that you could use in your Game application.*
  + *Explain what is this DP and the reason why it could be recommended.*

The dynamic programming would be utilized to manipulate the grid size based on the user’s chouce of a couple of present options that are interchangeable.

**References**

*[Include eventual references used here]*

Algonquin College

Winter, 2022