ALGONQUIN COLLEGE

ASSIGNMENT 2.1 – MVC Model (NumPuz[1])

General View

Due Date: prior or on **16th Oct 2022 (Sun, midnight)**

- **2nd Due date** (until Oct 23rd 2022) - 50% off.

Earnings: 4% of your course grade.

Purpose: Define the MVC model for the NumPuz Game.

- This is the third task in JAP. The purpose is to define the complete MVC structure for the game. TIP: In the following sections, check the **TODO** activities in the following sections.
- **PART I:** Considering the GUI defined in the **A11** (and implemented in the **A12**), define the MVC pattern to be used:
  - What is the **controller** (use this part to define behaviors)?
  - What is the **view** (use this part to update the interface)?
  - What is the **model** to be used (consider all data that / properties are relevant to the game?
- **PART II:** Considering your application:
  - Detail the differences between your application and the previous configuration.
  - Evaluate any other pattern that could be used by your application.

Part I – Updating the Project by MVC

1.1. Basic MVC

The new version of **NumPuz** must use **DP** (Design Patterns) and, specifically, the **MVC** (Model-View-Controller). The idea is that **Game** can implement the MVC model with entities

[1] Check the A11 specification for more details.

responsible for managing the activities (**Controller**), define the basic data properties and configuration (**Model**) and the GUI interface itself (**View**).

Note 1: The MVC implementation
The definitions for all methods are free, since you respect the basic principles about the responsibility about each layer. One strategy is to start defining the interface (based on GUI) and identify the model that describes the game (for instance, the String) and finally, define the actions to be developed.

❖ **TODO**:

- Describe the way you can define the MVC components of the game.
  - Ex: Suppose this definition.

Example (from vision "top-down")
Model Class: GameModel – Object: "myModel" (POJO / Plain Java Old Object)
View Element: GameView – Object: "myView" (extends JFrame implements GameController)
Controller Class: GameController – Object: "myController" (responsible for all Actions)
…

Note that, in the future (A22), the implementation to be done is similar to:

```
CalculatorModel myModel = new GameModel();
CalculatorView myView = new GameView();
CalculatorController myController = new GameController(myModel, myView);
myController.start();
```

1.2. Basic Components (View)

1.2.1. Original Proposal

You already planned to define components (A11) and implemented them in the first version of the game. Now, you need to make some adjustments / updates about how they are organized.

TIP
If your game is full functional in the A12, just check if there is something more to be added.

The **Game** must let user to create and play the **NumPuz**. One example of interface has already been provided (Fig.1).

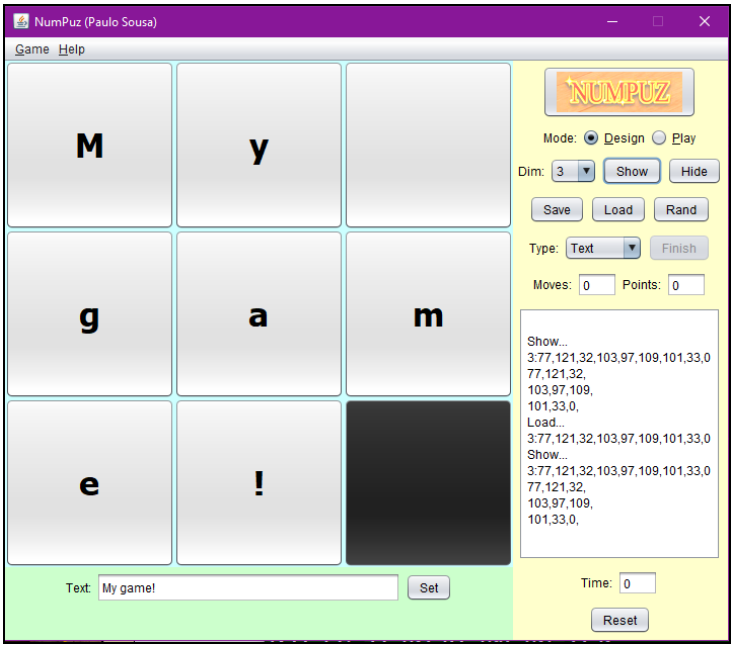

***Fig. 1** – Example of Game interface*

1.2.2. New GUI components

To help the user experience and include additional functionalities, some new elements are supposed to be included in your game.

- New requirements (check if these elements are already included in your original specification):
  - Splash screen: During initialization, one window must be shown during some seconds (ex: Fig.2);

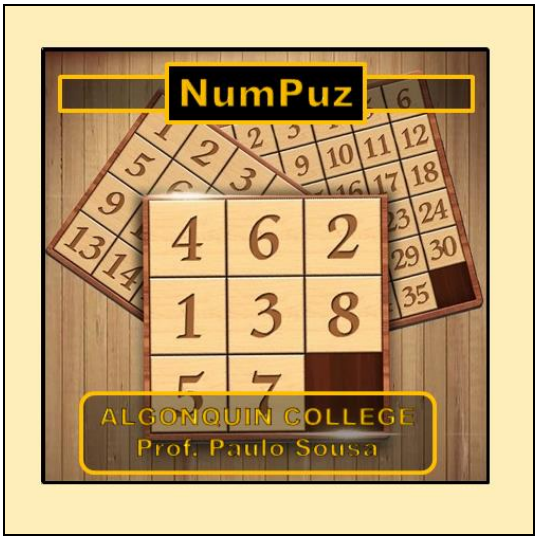

***Fig. 2** – Splash screen*

- Menus: The application must also show some menus (with icon options) to be used as alternative to perform some actions[2] (Fig.3)

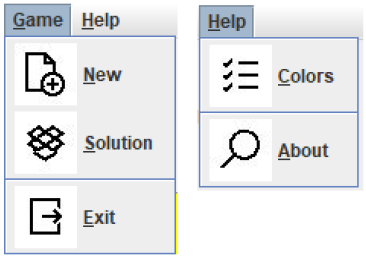

***Fig. 3** – Menu proposed to the game*

- Save / Load: Once the game is created ("Design Mode"), the user can save or, eventually, load a preconfigured game. You are suppose to use some dialogs to help you about this (Fig.4).

[2] The options are similar to those that can also selected in the interface.

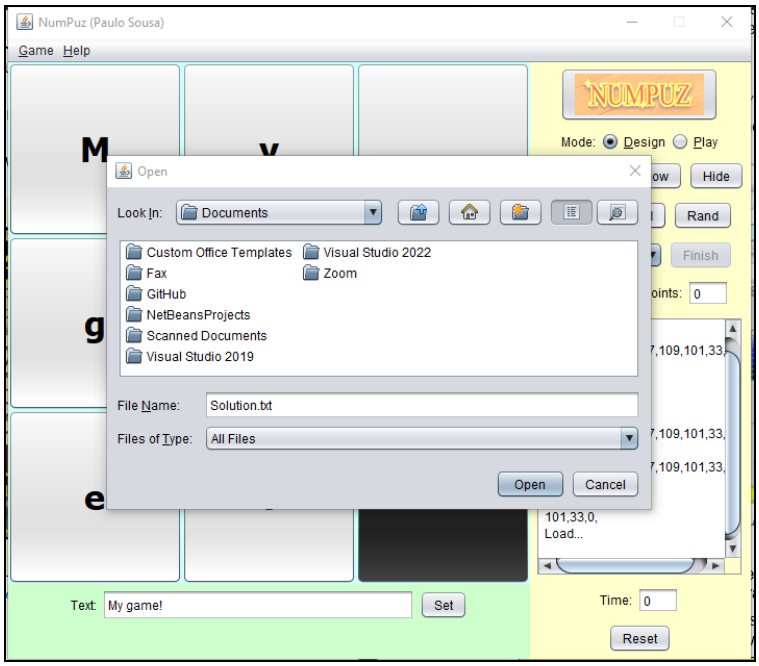

***Fig. 4** – Load / save functionality (file manipulation)*

- Color configuration: During the game ("Play mode"), some preferences can be set (ex: grid colors[3]) – see Fig.5.

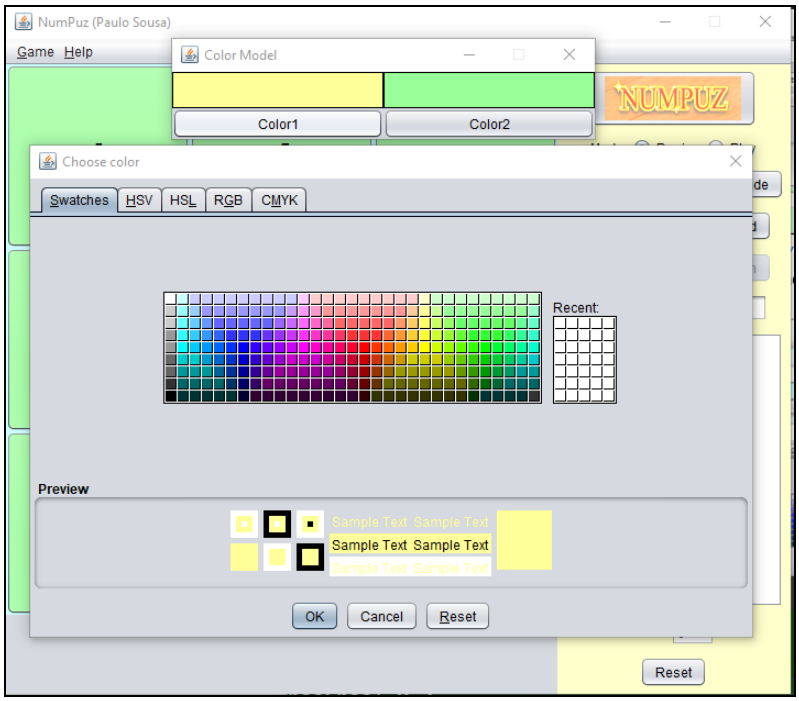

***Fig. 5** – Choicer for user color preferences (color configuration)*

[3] These colors are useful to better describe the spaces to be used in the **NumPuz** game, but depending on the implementation, different colors can be used. For instance, when selecting one specific value, it is possible to help user showing those already used by different colors.

❖ **TODO**:

- Define the list of all components (**Swing / JavaFX**) to be used as classes in your implementation, using the **hierarchy**[4].
  - Ex: if you are using buttons, list them according to the disposal in panels that are also included in the window frame.

Example (from vision "top-down")

Class: JFrame – Object: "GameFrame"

→ Class: JPanel → Object: "GameBoard"

→ Class: JButtons → Objects: "BSave", "BLoad", etc.

→ Class: JLabel → Objects: "LabOperation", "LabName", etc.

…

1.3. Basic Manager (Controller)

1.3.1. General Aspects

The game itself is supposed to be defined by a simple controller, intercepting actions and managing the proper answers to the user.

❖ All components must have a **unique action command**, but the way that you can select the mode as well as the strategy to be used in the game can vary freely.

❖ For example:

- When selecting one component (ex: JButton), the action to be performed should implement one specific method[5].
- The **NumPuz** itself can be composed by buttons, or labels, or text fields.

1.3.2. Additional Functionality

❖ The controller must be able to perform some additional operations such as:

- **Persistence**: You need to be able to save and load the data (see next section), what means that functions for these operations must be created.
  - ***Idea**: When you create a configuration (Design Mode), you can save this configuration for future use in the Play Mode.*

[4] The hierarchy here is given by the composition. Ex: a frame contains a panel that contains labels and buttons, etc. (and not the OO library class.

[5] There is no real obligation to define methods for all actions – and in fact using lambda, simpler notation can be used, however, it is a practice that can be used to better organize the activity.

- **Default game / randomization**: Your game should also be able to create one specific function to define a "default" or randomized game, that can be used since in the beginning.
  - ***Idea**: You can define specific strings obeying the rules for **NumPuz** or, better define a proper method to create a random configuration.*

❖ **TODO**:

- Create a "map" making the link between the GUI component (use the names from objects previously defined in the View) and the corresponding method to be invoked when one specific action is done.

Example
Object: "BSave" → Event: actionPerformed → method: saveGame() Etc.

  - This "map" is interesting to future game implementation and some functions can be invoked by different ways (ex: when using the Menu).

TIP
Some components in the View, maybe do not need to have any action. Ex: JPanels. However, most of time user is interacting with some that must show results / return proper behaviors from actions (ex: JButtons).

1.4. Basic Data (Model)

The model to be used here has multiple objectives:

A. You need to be able to play using the proper data (the configuration).

B. Allow users to collect info from users – ex: points and duration (given by the timer)[6].

1.4.1. The Game Configuration

The configuration is supposed to help the application to start the execution.

❖ Basically, the configuration is composed by:

- **Dimension**. Ex: "3"
- **Sequence of data**. Ex: "3;M,y, ,G,a,m,e,!"

[6] This data will be used by networking programming later.

Example
Data structure used:
→ Values: gridValue → method: updateData()

1.4.2. Generic configuration

The basic data about **NumPuz** is given by the following data:

- **Dimension**: Number that defines the total space for rows, columns and grids (supposed to be dim^2).
- **Board**: The representation of the *grid* (dim x dim) that can be simply a 2-dimensional Character[7] array.
    - ***Note:*** *Basically, even when you have a predefined configuration during the play, since* ***NumPuz*** *has an exponential combination of values, completely different solutions can be found.*

1.4.3. The User Data

For future purposes, the following data will be necessary:

- Game results:
    - **Points**: How much the gamer got during the execution.
    - **Time**: The number of seconds / duration of the match.
- Global info:
    - **Name**: User identification.
    - **Max Score**: Maximum points got by user in different matches.

Part II – Additional Documentation

2.1. GAME EVOLUTION

This second part requires an analysis of your game implementation. So, focus on two aspects: the evolution and the architecture.

- **Evolution**: How your application has been changed and the reasons for additional modification.
- **Architecture**: Focusing on **DP** (*Design Patterns*), what you can imagine could be improved in your game.

[7] Note that we are not focusing exclusively on numbers (what can be used only for dim=2 or dim=3).

- **TODO**:
  - 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.
      - *Example: Eventual modifications in the components previously defined / used (changing JTextfield to JButton, for instance).*
  - Up to now, the only **DP** explicitly mentioned is the MVC. However, both explicitly and implicitly, you can have additional DP to be used in the application.
    - 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.

Part III – Documentation and Submission

Since you are in another document proposal, focus on answer the "**TODO**" activities mentioned in this specification. Remember: right now, you do not need to implement nothing.

- **SUMMARY:** In short, create a document, where you specify the details of your MVC **NumPuz** version.

Note 2: About Teams
Only teams already defined are allowed (and just one member should submit). It means that, if you decided to work alone, you will continue until the end of the course.

Evaluation

- Please read the Assignment **Submission Standard and Marking Guide** (at "Assignments > Standards" section).
- **About Plagiarism**: Your code must observe the configuration required (remember, for instance, the "splash screen" using your name. Similarly, we need to observe the policy against ethic conduct, avoiding problems with the 3-strike policy...

Marking Rubric

Maximum Deduction (%)	Deduction Event
-	**Severe Errors:**
4 pt	Late submission (after 1 week due date)
4 pt	Plagiarism detection (not referenced)
-	**Part I – MVC**
2 pt	Missing architectural elements (View)
1 pt	Missing event handler strategy (Controller)
1 pt	Missing structure / data definition and analysis (Model)
2 pt	Problems with MVC integration
0.5 pt	Other errors
-	**Part II – Additional elements**
0.5 pt	Missing evolution details
0.5 pt	Missing DP aspects
1 pt	GitHub utilization
1 pt	Bonus: interesting ideas, enhancements, discretionary points (1%).
Final Mark	**Formula:** 4*((100- ∑ penalties + bonus)/100), max score 6%**.**

Submission Details

- **Digital Submission: Compress** into a **zip** file with all files (including document).
- Upload the zip file on Brightspace. The file must be submitted prior or on the due date as indicated in the assignment.
- ***IMPORTANT NOTE:*** The name of the file must be **Your Last Name** followed by the last three digits of your student number followed by your lab **section number**. For example: **A21_Sousa123.zip**.
  - If you are working in teams, please, include also your partner. For instance, something like: **A21_Sousa123_Sousa456.zip**.
- ***IMPORTANT NOTE:*** **Assignments will not be marked if there are not source files in the digital submission.** Assignments could be late, but the lateness will affect negatively your mark: see the Course Outline and the Marking Guide. All assignments must be successfully completed to receive credit for the course, even if the assignments are late.

File update: Oct 3rd 2022.

Good luck with A21!