Algonquin College Logo

# SCHOOL OF ADVANCED TECHNOLOGY

### ICT - Applications & Programming

### Computer Engineering Technology – Computing Science



A21

Game MVC (Class Diagram)

Team:

[Boyu Li] - Id: [041003345] / [Golden Zhang] - Id: [041051971]

Battleship 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 descriptin and dividing them into the MVC components.*

* 1. **Classes specification**

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

*Describe all classes that you imagine to use in your game.*

**My Class Specifications:**

**NOTE: Please also check with sub class for each class definition**

1. **Class GameView**

**Name**

**GameView** (Class to be used to visualize the user interface)

**Attributes**

JComboBox languageBox (Game language switching component)

JComboBox dimensionBox (Game dimension switching component)

JButton designModeButton (Component to switch game mode to design mod)

JButton randModeButton (Component to switch game mode to random mode)

JTextArea historyArea (Displaying game history component)

JTextField timeTextField (Displaying game time component)

JButton resetButton (Component to reset the game)

JButton playButton (Component to start to play the game)

JProgressBar userLifeProgressBar (Component to display the life of the user player)

JProgressBar adversaryLifeProgressBar (Component to display the life of the machine player)

JButton[][] userGridButtons[][] (Manage all grid buttons in user panel)

JButton[][] adversaryGridButtons[][] (Manage all grid buttons in adversary panel)

**Methods**

initializeMainWindow(): Initialize and visualize the base window of the game

initializeControlPanel(): initialize and visualize the Control Panel with its component of the game.

initializeUserPanel(): initialize and visualize the User Panel with its component of the game.

initializeAdversaryPanel(): initialize and visualize the Adversary Panel with its component of the game.

addListenerForComboBox(ItemListener itemListener): setup the listener for all combo box component

addListenerForButton(ActionListner actionListener): setup the listener for all button component

launchInterface(): Launch the game interface

updateUserProgressBar(): Update the user progress bar

updateAdversaryProgressBar(): Update the adversary progress bar

intialCoordinatesPanel(): Initial all the coordinates of user and adversary panel grid

displaySecondTime(): Set the game of time

displayHistory(): Display the history of the game log

updateButtonColor(): update the color of the user/adversary button

1. **Class GameController**

**Name**

**GameController** (class to be used to implement the functionalities of each component)

**Attributes**

GameView gameView (The object of GameView)

GameModel gameModel (The object of GameModel)

**Methods**

launchGame(): launch the game

refresh(): refresh the game interface

startTiming(): enable the timing component

**2.1 Subclass ButtonListener**

**Name**

**ButtonListener** (class to be used to implement the functionalities of the button components)

**Attributes**

**NULL**

**Methods**

actionPerformed (ActionEvent e): Implementing the listener for all button functionalities

**2.2 Subclass ComboBoxListener**

**Name**

**ComboBoxListener** (class to be used to implement the functionalities of the combo box components)

**Attributes**

**NULL**

**Methods**

ItemStateChanged (ItemEvent e): Implementing the listener for all combo box functionalities

**3.0 class GameModel**

**Name**

**GameModel** (class to be used to store the game data)

**Attributes**

int userLife (The life data of user player)

int adversaryLife (The life data of machine player)

int dimension (The game dimension data)

typeOfState userPanelButtonState[][] (The state data of each button on the user panel)

typeOfState adversaryPanelButtonState[][] (The state data of each button on the adversary panel)

boolean designModeData[][] (The user generated data on design mode)

Ship usership[] (The ship data of user player)

Ship adversaryShip[] (The ship data of machine player)

**Methods**

getUserLife(): getter method for user life

setUserLife(): setter method for user life

getAdversaryLife(): getter method for adversary life

setAdversaryLife(): setter method for adversary life

getDimension():getter method for game dimension

setDimension(): setter method for game dimension

getUserPanelButtonState (): getter method for user panel button state

setUserPanelButtonState (): setter method for user panel button state

getAdversaryPanelButtonState (): getter method for adversary panel button state

setAdversaryPanelButtonState (): setter method for adversary panel button state

getDesignModeData (): getter method for design mode data

setDesignModeData (): setter method for design mode data

getUserShip(): getter method for user ship data

setUserShip(): setter method for user ship data

getAdversaryShip(): getter method for adversary ship data

setAdversaryShip(): setter method for adversary ship data

isHit(): To check if the player successfully hit the ship

updateLife(): update the life of user or machine player

updateState(): update the state data of the user/adversary button

**3.1 subClass Ship**

**Name**

**Ship** (class to be used to play the game)

**Attributes**

Int size (from 1 to dim)[[1]](#footnote-1)

int positionX (row)

int positionY (col)

boolean direction (horizontal/vertical)

typeOfState status (safe / compromised / destroyed)

boolean isVisible (showed or hidden)

**Methods**

setVisibility(): make it visible or not

changeStatus(): modifies the ship status

getSize(): Getter method for size

setSize(): Setter method for size

setPostitionX(): Setter method for positionX

getPositionX(): Getter method for positionX

setPostitionX(): Setter method for positionY

getPositionX(): Getter method for positionY

getDirection(): Getter method for Direction

setDirection(): Setter method for Direction

getStatus(): Getter method for status

setStatus(): Setter method for status

getIsVisible(): Getter method for isVisible

setIsVisible(): Setter method for isvVisible

**4.0 Class GameLauncher**

**Name**

**GameLauncher** (class to be used as the start point of the game)

**Attributes**

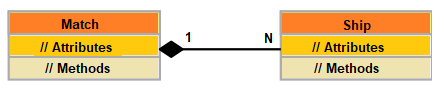
**Methods**

main(): the starting point of the whole program

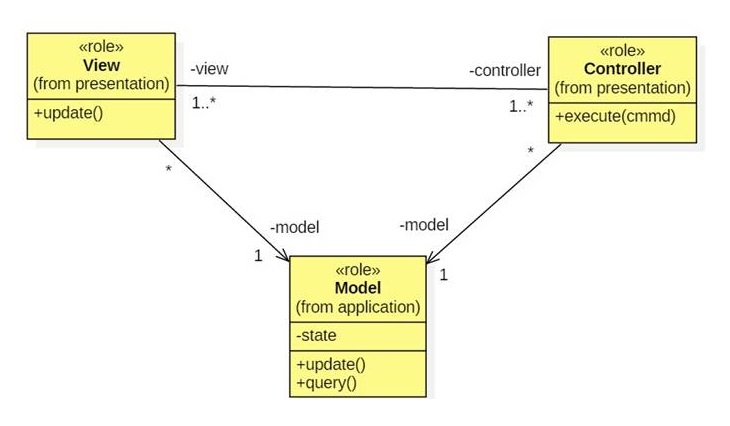
* 1. **Class diagram**

*Describe all classes relationships that you need to define to your game.*

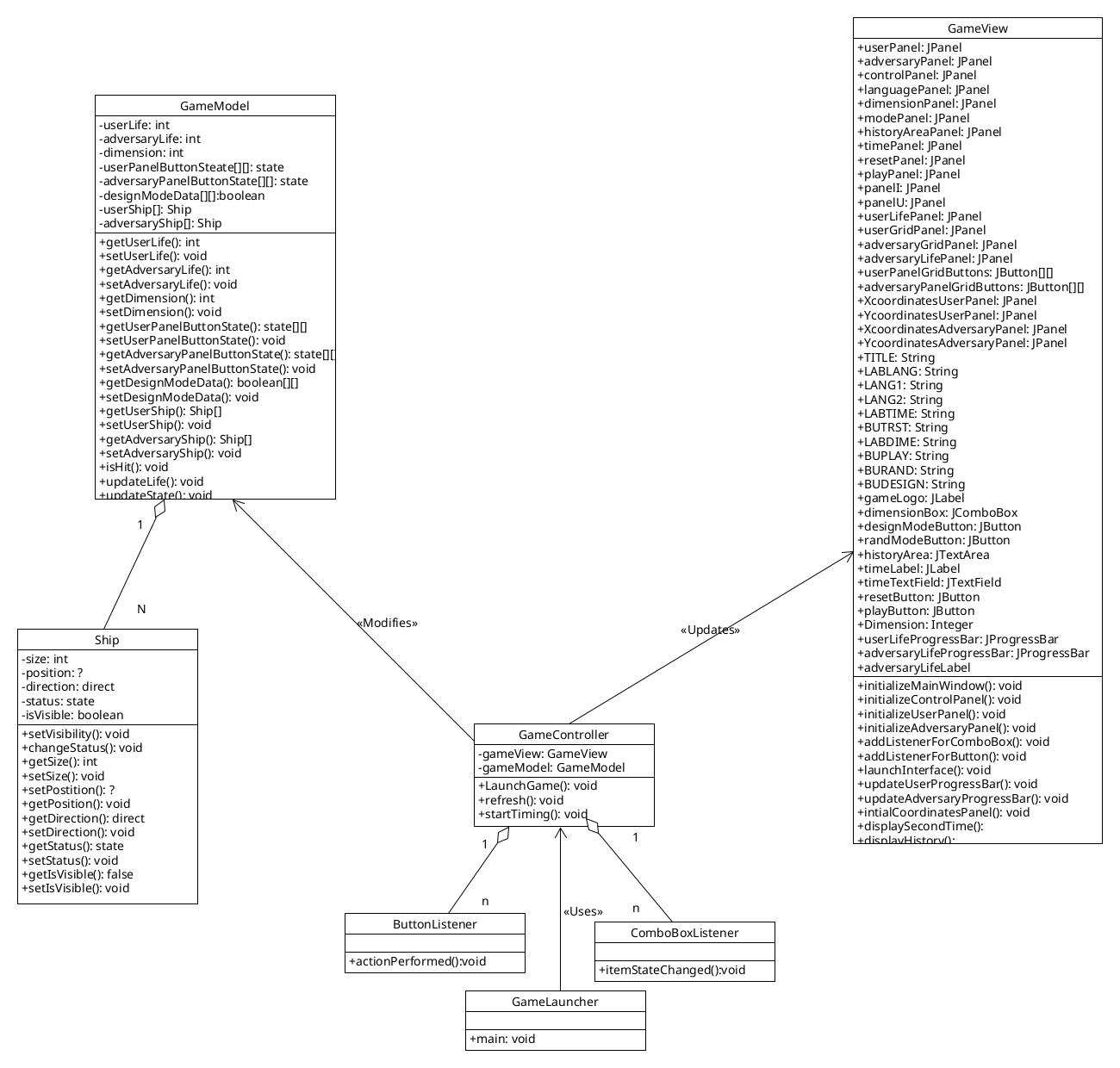
* + - **Example**: Basic relationship between ships and match (Design mode).



* + - **Example**: Basic relationship between MVC components.



* + - In the Model, detail the **entities / data** used to represent the game.
      * For instance, consider important fields such as solution, points, timer, etc.
    - In the View, emphasize all the **visual elements** that you have and needs to include.
      * For instance, what are your containers (panels / frames), and components (ex: buttons, labels, etc.).
    - In the Controller, what are the **methods / behaviors** that must happen?
      * Remember that your controller should unify all actions treated by listeners by calling proper methods[[2]](#footnote-2).



**References**

*[Include eventual references used here]*

Algonquin College

Spring, 2023

1. As mentioned in the previous assignment “dim” is the “dimension” of the game. [↑](#footnote-ref-1)
2. Remember that isolation of methods and encapsulation of entities are basic OO principles. Additionally, it is a good practice to isolate behaviors in proper functions that can be called by different actions (ex: button clicks / menu items, etc.). [↑](#footnote-ref-2)