Juan Carlos Escobar
SE 350 - Final Project

Introduction:

This framework allows quick development of simple grid-based games such as Tic-Tac-Toe, Memory games similar to Simon, and Tile-Matching games similar to Bejeweled, that are interacted by the user with a mouse.

Framework Design:

The framework consists of the following:

- GUI «interface»
- AbstractGUI
- Game «interface»
- AbstractGame
- AI «interface»
- Drawable «interface»
- Board «interface»
- ArrayListBoard

AbstractGUI:

Implements the GUI «interface»
Implements the MouseListener «interface»
The AbstractGUI handles the GUI elements for the game. Any class that extends AbstractGUI will act as the Controller of the MVC design.

AbstractGame:

Implements Game «interface»:
The game rules are implemented here. This is the only class that needs to be created specifically for the type of game that is being created. This class needs to have a way to access the Board in order to retrieve and store data into the board of the game.
Instances of AI classes are instantiated here that can later be swapped in your concrete class on-the-fly by calling a modifier class. Any class that extends AbstractGame will act as the Model of the MVC design.

AI «interface»:

An interface that is implemented by a class that will give your game varying difficulty settings.

Drawable «interface»:

Any class that will contain methods that will paint elements to the GUI needs to implement this interface. Any class that implements Drawable will act as the View of the MVC design.

ArrayListBoard:

Implements the Board «interface»:
A concrete class that provides restricted access to an ArrayList of integer values. This acts as the board that will be translated into the GUI view of the game.

Juan Carlos Escobar
SE 350 - Final Project

How-To:

Simple and fast way to develop grid-based thinking games. Create your concrete game rules, plug them into the framework and your logic is done. Create a concrete GUI design, plug it in and the game has its own GUI that can change to fit the need of the grid-based game's visual design. If you want to have different difficulty levels without having to add them all into your concrete game class logic, create a new class that implements the AI interface, add the specific difficulty logic and plug it into the framework. Varying levels of difficulty are now added.

Design:

When writing down ides for grid-based thinking games I decided that I would need a class that restricted access to an ArrayList by only allowing the setting and modifying of elements, the size, the availability of a certain index, and to allow access to all the entire elements from the board by return them as one string that could be parsed by other classes, if needed. I made this a concrete class since out of the three games that first popped into my head, this was one thing they all had in common.

I decided to create an AbstractGame class that implements a Game interface so that adding the concrete logic of the game would be simple. This way you only need to extend AbstractGame and write your class with all the methods that you need to implement, adding helper methods if necessary, and you can easily plug-in the different logic for any grid-based thinking game you wish to create.

I wanted to create an AbstractAI class that would implement the AI interface, but during the phase when I was coding my design I realized that it would be much more flexible to just have in interface so I could then add different difficulties to the game logic without needing to be too restricted on a specific way.