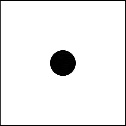
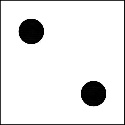
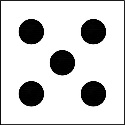
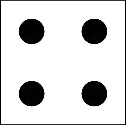
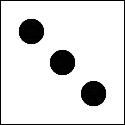
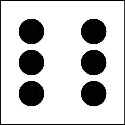
|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Y | A | H | - | D | U | - | O | T |



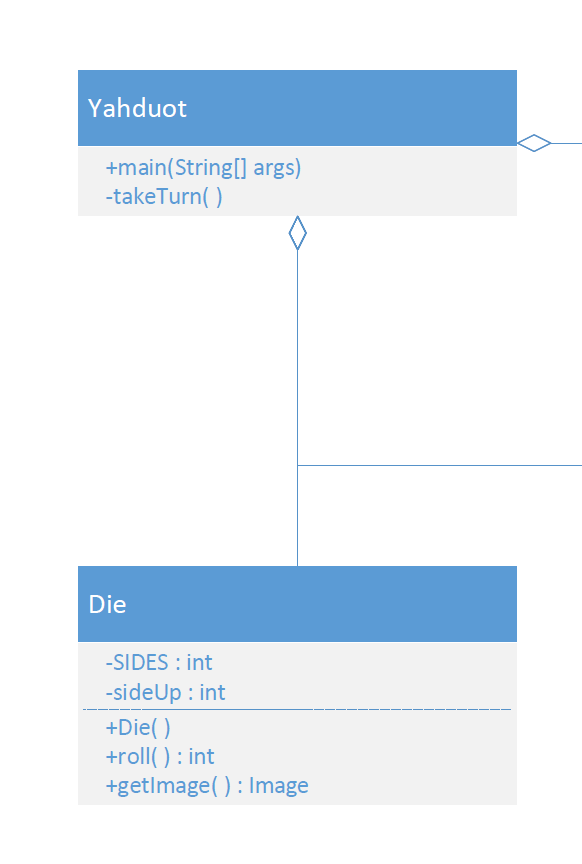
UML Class Design

Erik Fox, Max McKee, Will Fraisl

**Design Pattern**

We are approaching the design of this game using the Model-View-Controller design pattern. This design pattern lends itself to UI/UX projects, and fits Yah-du-ot well. Yah-du-ot is a traditional style board game designed for a computer. Like most traditional style board games, Yah-du-ot t is at its core based around data being manipulated, and having the play-space reflect those changes. Certain combinations of data and user selection determine a winner. We have chosen the Model-View-Controller so that we can separate the data (the model), the play-space (the view), and the user input (the controller) eloquently. Additionally, this will help us pursue alternate design choices for parts of the program without affecting other parts.

**Classes**

**Controller**



This is the main class of the program. This class will handle

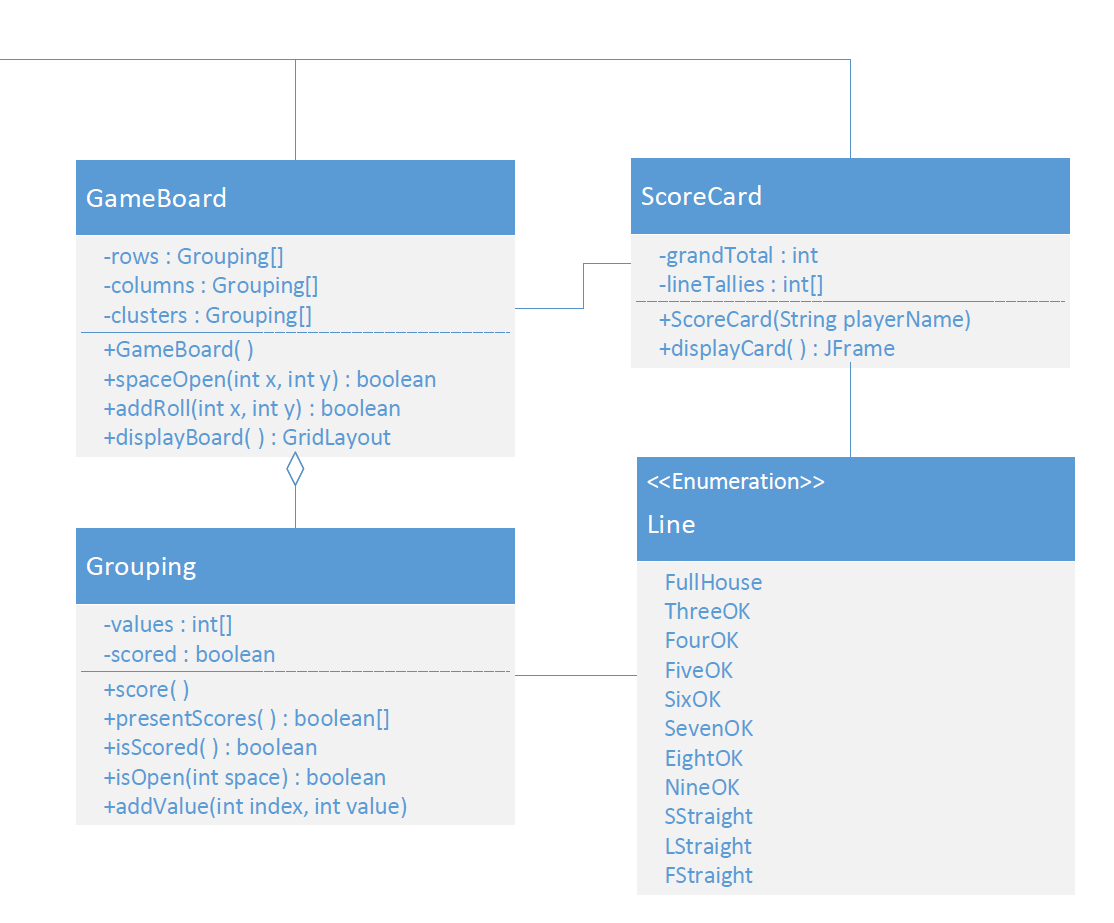
user input and pass data between the model and the view.

This class is the most subject to change, and will likely require  
more helper methods to improve readability. This class will  
contain references to a Die, a GameBoard, a ScoreCard, and   
YahduotUX object. It will then be responsible for passing   
information between the objects during runtime.



The Die class is a tool to be used by the Yahduot class to generate a new roll for the player. The image associated with the current side is stored by the Die to be updated with every roll, and be passed to the view.

**Model**





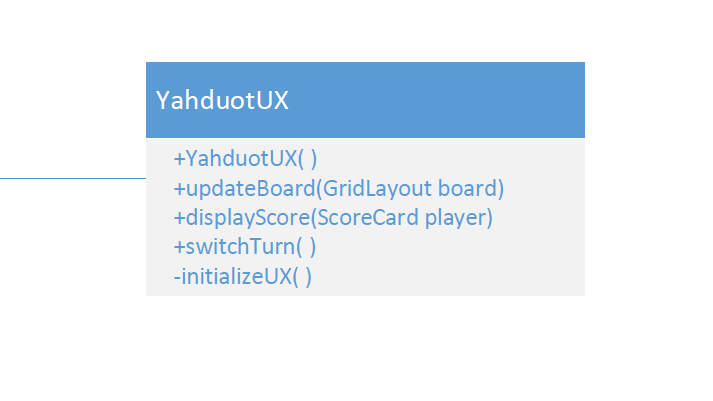
The GameBoard class uses Grouping objects to organize the values played by the players. When a grouping is completed by a new value, control will be passed to the ScoreCard object to score the grouping. The object will not allow a new value to be placed over an old value, and will return false to signify the value could not be added. The object also creates a GridLayout object representation of itself to be passed for display in order to keep data private.



The Grouping class contains an array of values representing a row, column, or cluster on the board. It will accept a value and an index for a new value, and will not overwrite existing values. The object will be able to check whether or not space is available for new values. The object will determine valid scoring options for the grouping when filled. The object then return an array of boolean values to represent the valid scoring options to be presented to the user by the UX.



The ScoreCard class is responsible for maintaining the score of the user at the current point in the game. The object maintains a UX representation of itself to be passed to the UX object for display. The object will also maintain a running total and tally of each line being scored.

**View**



The YahduotUX class handles the various UX elements of

the program and displays the data passed into it by the

Yahduot main class. The object will not modify game

data, but will provide UX elements to enable user input.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Y | A | H | - | D | U | - | O | T |

Complete UML Class Diagram