**Controller**

The controller is the class that represents the functionality of a Controller in the MVC paradigm. It is responsible for starting the game, and instantiating the two central view and model objects (those being the GameView and Board respectively). We choose to make most of the members of this class static. Because the controller isn’t an actual thing in the game, but rather a meta structure for communication between the model and the view, it didn’t make sense to instantiate it. Also, since every game should only ever have one controller, everything inside a controller can just belong to the class itself while maintaining full functionality.

The main responsibility of the Controller is the communication of events and state information between the View and the Model. In order to fulfil this responsibility the board contains a reference to the GameView and the Board. The stack of method calls for when a card is clicked is the following:

“CardStackListener.ActionPerformed() -> Controller.onCardChosen() -> Board.playTurn() -> TurtleMaster.onCardPlayed()”

Because there are a number of ways that a move can be deemed invalid, and the responsibility of deeming a move invalid falls across multiple classes, this stack of boolean function calls serves as an elegant way to signal back to the previous calling class that a move was not valid (which eventually will fall back to the view). If at any point, one of these methods returns false, the previously called method will also return false. This will then eventually fallback to the view which notifies it to prompt the player to pick a different card. Doing so will then start the call stack over again.

**Controller Enumerations**

The Controller package contains two different enumerations which represent card types and tile types. These are used as a common way to pass information between the View and the Model. We choose to use enumerations because of their flexibility and extensibility. For example, if a new TileType or CardType needs to be added, adding it to the enumeration will account for it in every section of the code. For example, the BoardPanel (responsible for drawing the board layout) has an array of tile sprites which is sized based on the number of elements in the TileType enumeration. This array is also indexed using the ordinal value of an element in the TileType enumeration. This was a design pattern we used for the Tile and CardTypes.

**View.BoardPanel**

The BoardPanel is responsible for drawing and updating the board layout. In order to keep this as separate and independent of the model as possible, it is able to track and update player position and direction based solely on the player number and new position. This update occurs every time a valid move is made. If an invalid move is played, the aforementioned method call stack terminates before the Controller instructs the view to update.

**View.CardStack**

The CardStack is an extension of a JButton. When designing this project, we knew we needed an easy way to translate a button click to a CardType without having to include model types in the view. This is originally what inspired the use of enumerations