Solitaire Design Manual

**Model:**
The solitaire game leverages the MVC design pattern in order to effectively implement the code with high cohesion and low coupling. The root of the system comes from the card class, which is the structure used to encapsulate the functionality and image of its respective playing card. A deck class encapsulates the fifty-two cards found in a standard playing card deck. The deck class contains functionality responsible for drawing and shuffling such that the game has the ability to add new cards to the playing screen. Stock, Talon, and Foundations classes all leverage the pile class in order to implement their main functionality. This functionality includes holding cards, and the ability to split the ArrayList of cards allowing for multiple-card movements. The Stock includes the added ability to draw a card into the talon and reset itself to allow for elongated and proper solitaire play. The Talon works in tandem with the stock holding the ability to hold a list of cards and has a first-in-last-out pattern to simulate drawing cards from the top of a pile. The foundations have a more complex functionality allowing for cards to be stored by suit and in increasing order from Ace to King. The primary function of the model class is to complete the backend computations and communicate the results to the controller. This includes functionality such as flipping the top card in a pile, drawing a card, and adding single or multiple cards to a pile. These methods are meant to be called by the controller such that the controller can interpret the results and translate them to the graphical user interface.

**View:**
The view leverages an abundance of containers such that the solitaire view can be compact and functional for the user. The view uses a border pane layered on top of a colored anchor pane. The border pane is anchored such that as the screen increases in size, the game stays in the top left corner. This is done purposefully such that the game is perfectly displayed in split view on all operating systems. The border pane extends downwards such that players can see particularly long chains of cards. In the top portion of the border pane are the information center encapsulating the timer, the number of moves made, a label such that the player knows which game they are playing, and a button dedicated to resetting the stock. On the left of the border pane is a VBox which is used to hold both the stock and talon (added in the controller). On the right side of the border pane is another VBox which is meant to hold the four piles of the foundations (added in the controller). The center of the border pane is dedicated to the tableau. It uses an HBox used to store seven VBoxes which act as the main playing piles.

**Controller:**
The Controller leverages the SetOnClick function to communicate between the VBox elements in the View and the main elements of the Model. The Vbox elements are filled with Imageview objects using the image objects stored by name in the Card class or by new Image objects representing that space is empty. Because it is difficult to remove these ImageView objects from the Vbox, we instead opted to clear out all of the VBoxs and re-add cards based on the Model before it is displayed, which gives the effect of moving the desired cards in the view. We use the SetOnClick function on all children of the Vboxs which makes the cards on the screen have

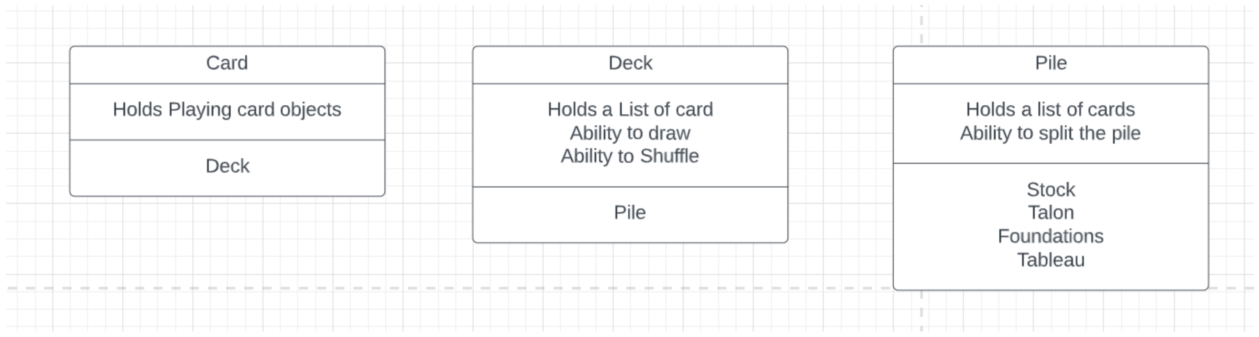
functionality when clicked. We then have some logic within the Controller to differentiate between first and second clicks and how that affects what will be done in the Model.

**User Stories:**

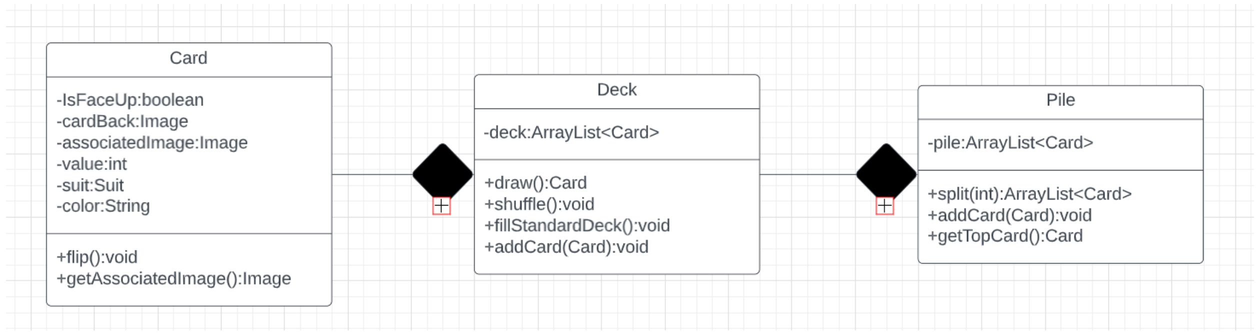
Jack Harris the Solitaire Novice
I want: an instruction menu
So that: I can review the rules when I forget what to do in a game
Quote: “Solitaire seems cool! I might play if I can find an app, but I probably won't if there is no app"
Narrative: I have never played solitaire but of course, I have heard of the game. I like to play little games on my phone or laptop when I am trying to pass the time. I think if there was a solitaire app that I could play, I would learn the rules and play while I am sitting around.

Consuelo Alvarez the Solitaire enthusiast
I want: to restart games if I recognize things are going poorly
So that: I can save time and not waste my energy on games I recognize are going poorly
Quote: "I love solitaire, but who has time to set up a big game anytime they want to play"
Narrative: I am Consuelo Alvarez my mom taught me to play solitaire when I was younger, but I do not really have the time to play anymore. I enjoy the game but have not played in a while. If there was a convenient app that allowed me to play solitaire without having to set up all the cards and play all at once, I think I could play more solitaire.”

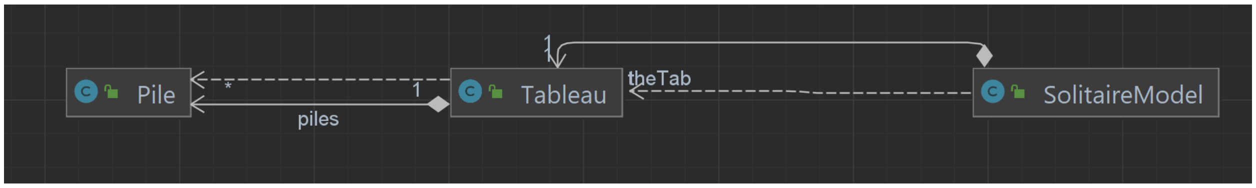
**Card, Deck, and Pile:**
When designing the Card, Deck, and Pile classes, we were quickly able to determine that the classes would build off of each other in functionality to make a complex structure that could be used in the unique classes subsequent to pile Tableau, Stock, Talon, and Foundations (Seen Below).



Encapsulating all of this functionality in the pile class would create a lowly cohesive class that retains a lot of facets of functionality. This includes creating cards, sorting them into a deck structure, and being able to hold a deck/list of cards with the added functionality of splitting the lists of cards. Striving for good design principles, the team settled on a design where a deck holds 52 instances of the card class, and a pile holds a list of cards or a deck depending on the use of the pile (described below). Below is a high-level diagram of the UML for the Card, Deck, and Pile classes.

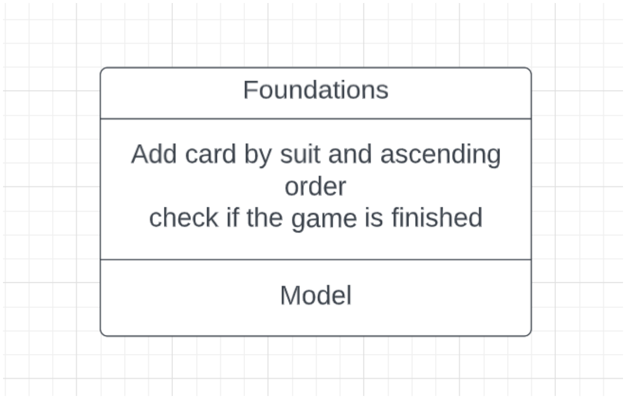


**Tableau, Stock, and Talon**
Tableau directly leverages the pile class in order to create the main game board consisting of seven such piles. Tableau inherits many useful functions from the pile class and uses added functions such as fillable() in order to give the functionality of an actual tableau. This tableau instance is then used in the model class which carries out the main game logic. The UML for these relationships is shown below.

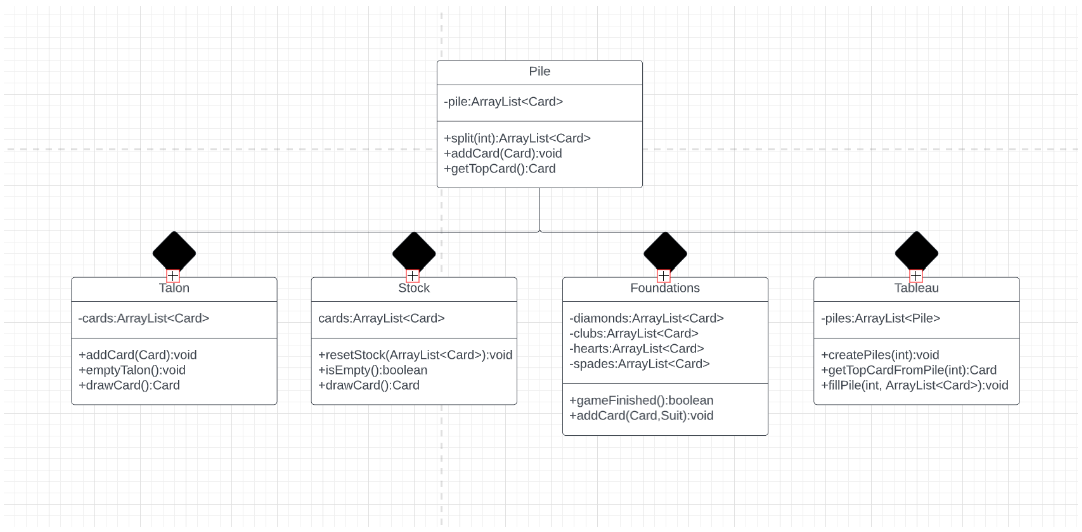


The Stock and Talon classes like Tableau need to use the functionality encapsulated in the pile class. The Stock and Talon classes build off of this platform in order to behave accordingly.

**Foundations:**
Using the Pile class for base Functionality, the foundations add restrictions to which cards can be added in order to add cards by suit and in ascending order. Although placing restrictions on a class inherited is usually seen as a poor design technique. The circumstances of our current design presented made it a logical decision to have our foundations inherit and place restrictions on how cards could be added to the piles (CRC shown below).


Foundations
Add card by suit and ascending order
check if the game is finished
Model

**All Classes that Inherit the Pile Class**



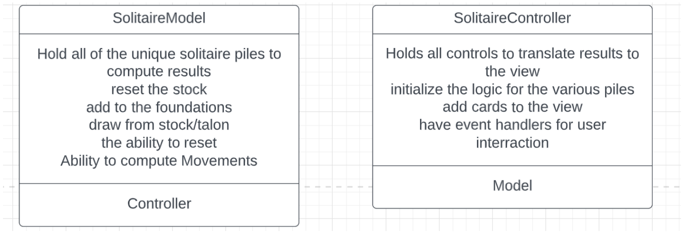
## Model:

The model is the class that brings together the Stock, Talon, Tableau, and Foundations and applies logic for moving cards between them. This class serves as the fundamental base from which the game is played. This makes the movement of cards much simpler and allows to easily implement functionality for resetting the game by simply invoking the constructor in the model class.

## Controller:

The Controller ties click events made on elements of the view with functionality as defined in the Model. This allows us to have seamless movement of cards on the screen tied to the clicks made by the user. Driven by the needs of the user stories to incorporate an instruction menu and undo button; the modular structure of the controller allows the option to add buttons for other functions such as resetting the model or undoing. One of the most useful design features included in our program was the “glow” created when you click a card. Although a small change, this makes all the difference for the user experience. Prior to this functionality, it was hard to keep track of clicks and specifically know which card was currently clicked. This lack of knowledge of the graphical user interface particularly made debugging a nightmare. Unable to tell which card was selected only to find that half of the cards disappeared did not make for intuitive debugging. The glow served as a double-edged sword, making debugging easier while significantly increasing the gameplay of our program.

**Model+Controller CRC**



SolitaireModel
Hold all of the unique solitaire piles to compute results
reset the stock
add to the foundations
draw from stock/talon
the ability to reset
Ability to compute Movements
Controller
SolitaireController
Holds all controls to translate results to the view
initialize the logic for the various piles
add cards to the view
have event handlers for user interraction
Model