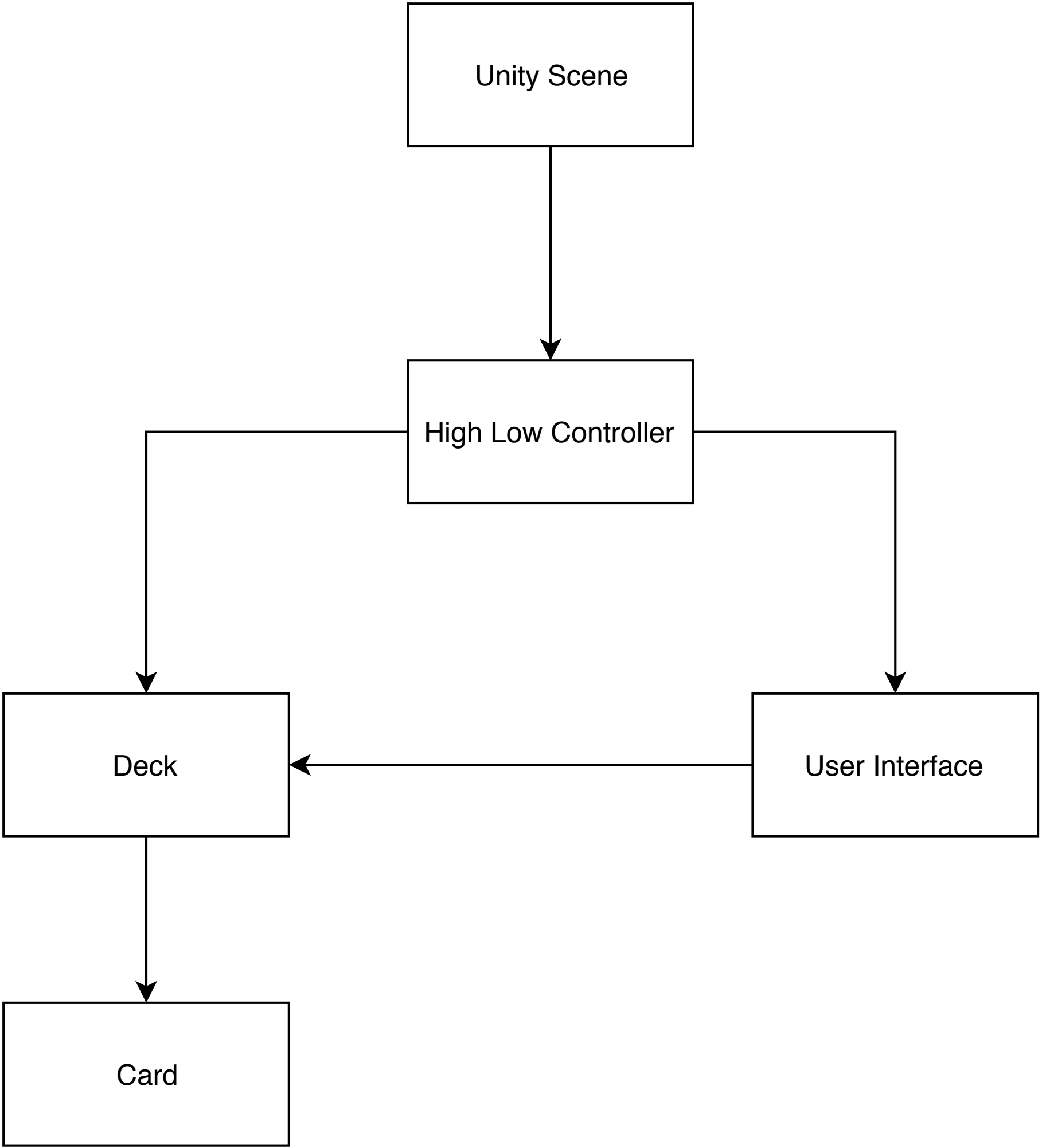

Unity Scene
High Low Controller
Deck
User Interface
Card

Unity Scene

Handles the games display and graphics

- Animations were created in unity
- Graphics such as board design and buttons were personally crafted

High Low Controller

The High Low Controller contains a script called HighLow.cs which handles the core operations of the game. This includes drawing and animating cards, display the game over menu, initializing the deck,

And determining the current state of the game.

- PlayGame() – Starts the game of high low
- UpdateDebugLog() – Updates the in game Debug Log with information regarding cards currently in play.

Deck

Deck.cs handles the construction of the deck of cards that the controller will be drawing from.

Deck Operations

- ResetDeck()
- Shuffle()
- RemainingCard()
- DrawCard()
- AddCard(List<string> deck)

Deck Construction

- GenerateDeckString()
- CreateFinalDeck()
- StringToFaceValue()
- StringToSuitValue()
- StringToTexture()

Shuffling Algorithmns

Below is a list of shuffling algorithms that were developed during the making of the game. The currently used algorithm is the NaivedBiased Shuffle.

- FisherYatesShuffle() – A commonly used algorithm that shuffles a list such that all elements are equally likely to be choosen

- NaivedBiasedShuffle(List<string deck) – A naïve version of the random weight shuffling algorithm that has an algorithm run time of 0(n) the algorithm works by:
  - Calculating the probabilities of each card and normalizing the probabilities so they add up to 1
  - Separating the cards into a stack that matches their probability
  - Shuffle each stack so cards that are popped are completely random
  - Create a loop that will run 52 times
  - Every iteration of the loop generate a random floating number between 0 and 1
  - If a stack probability falls in the range of the random floating point number, pop a card of the stack and it to our new deck of cards

Pros:

- Algorithm is efficient, as we will see in a plot below it had the overall best probability and stay consistent
- Fast run time

Cons

- Hard to read and maintain

- SmartBiasedShuffle(List<string> deck) – Implements the random weighted shuffle algorithm also known as the Walk Alias algorithm
  - https://softwareengineering.stackexchange.com/questions/233541/how-to-implement-a-weighted-shuffle

Pros:

- Easier to maintain and read
- Can be extremely efficient with a binary search

Cons

- Complexity of O(N^2)
- Correctly bugged as I had to quadruple the hearts weight to get proper results.

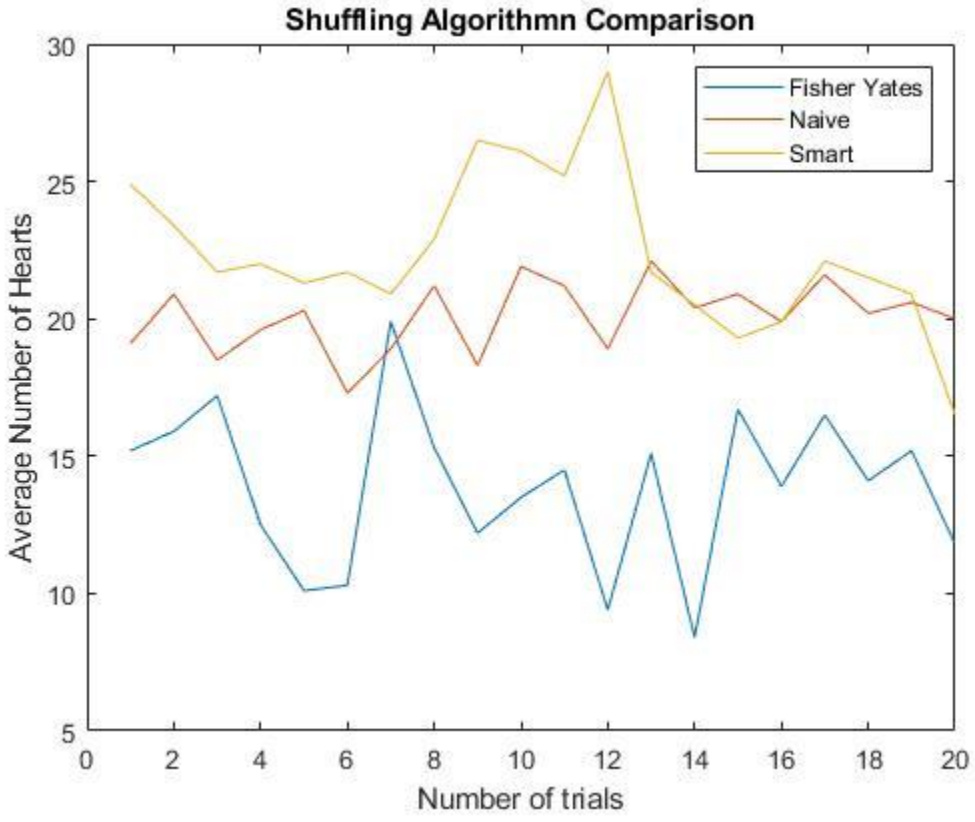
Probability Testing

In order to test the efficiency of my shuffling algorithms I created a test case. The test case involved creating 52 decks and drawing the first card on top of each deck and counting the number of times a heart appears. If my algorithms work correctly I should see double the number of hearts as I would see in a regular fisher yates shuffle. This would be the same when comparing hearts to the other suits as well.

- ExportProbabilityData() - Located on Line 35 in Deck.cs uncomment this line to export a CSV file. Be sure to change the file path in the function.

- TestProbabilities()
- AverageList()
- GenerateProbabilities() – To test different probabilities for other cards go to lines 539-541 and change the string in CardCounter(deck,suit) respectively
- CardCounter()

Results:



Card.cs

Contains a cards value in respect to its suit.