☐(1) Lab 7

☐(1.1) Problem

☐(1.1.1) Vegas, Solitaire-like Card Game Rules

- ☐ *(1.1.1.1) k cards in the deck*
- ☐ *(1.1.1.2) The cards in the deck are 1, 2, 3 … k*
- ☐ *(1.1.1.3) The cards must be placed in the victory pile in order from 1*
- ☐ *(1.1.1.4) Two discard piles can be used to collect the cards not immediately put on victory pile*
- ☐ *(1.1.1.5) Only the top card from both discard pile is able to be moved*
- ☐ *(1.1.1.6) Cannot move cards between the two discard piles*
- ☐ *(1.1.1.7) Game over when the entire deck is completely discarded*

☐(1.1.2) Prizes

- ☐ *(1.1.2.1) $5 to start playing*
- ☐ *(1.1.2.2) $1 for each card on the victory pile*
  - ☐ (1.1.2.2.1) Check to see if, by error, the discard piles have the next card for the victory pile
- ☐ *1.1.2.3)*

☐ 1.1.3. Flow

- ☐ *1.1.3.1) Every card discarded from the shuffled deck has 3 fates:*
  - ☐ (1.1.3.1.1) Discard Pile 1
  - ☐ (1.1.3.1.2) Discard Pile 2
  - ☐ (1.1.3.1.3) Victory Pile

☐ (1.2) Problem-Solving Session

☐ **(1.2.1) Use whatever strategy comes to your mind and play out the game for the case where k=8, and the cards from the deck are flipped over in the following order: [5,4,6,7,2,8,3,1] (5 first to be discarded from deck)**

- ☐ *(1.2.1.1) Deck Discard 1 - if not #1 > Discard Pile 1*
- ☐ *(1.2.1.2) Deck Discard 2 - if not #1 > Discard Pile 2*
- ☐ *(1.2.1.3) Subsequent Deck Discards:*
  - ☐ (1.2.1.3.1) If not card for Victory Pile > decide Discard Pile 1 or 2
    - ☐ 1.2.1.3.1.1) if (card < D1 & D2) OR (card > D1 & D2), place on discard pile with smallest difference
    - ☐ 1.2.1.3.1.2) If card > Dx & <Dx, put on pile with larger value
- ☐ *1.2.1.4)*  *Pasted_Graphic_2*

☐ **(1.2.2) We will implement "shuffling" of the deck in the following manner: All of the cards will be put sequentially into a data structure to hold them. When we want the next card , we will cycle through the cards, repeatedly taking one off of the top and replacing it on the bottom. We will do this a random number of time. When we stop cycling through the cards, the one on the top at that point is removed and returned as the next 'shuffled' card.**

What data structure will be useful to allow you the functionality you need to implement this version of shuffling?

- ☐ *(1.2.2.1) Shuffled Deck = Stack (Linked List)*
- ☐ *(1.2.2.2) Unshuffled Deck = Queue (Linked List)*
- ☐ *(1.2.2.3) Shuffling Procedure:*
  - ☐ (1.2.2.3.1) Get a random number
  - ☐ (1.2.2.3.2) Take -random number- cards (one at a time) from the top of the unshuffled deck to the bottom of it (enqueue(dequeue)).
  - ☐ (1.2.2.3.3) Return the top card to the shuffled deck (return dequeue)

☐ **(1.2.3. Write code for a function that takes an instance of this data structure, and returns the next 'shuffled' card (modifying the data structure n the process to remove this card). You may assume that any relevant modules have already been imported. make sure each card has an equal chance of being selected.**

- ☐ *(1.2.3.1) While counter in range(unshuffled.size):*

```
n=randint()
For I in range(n):
    enqueue(dequeue)
shuffledeck.push(dequeue)
```

☐ **1.2.4. What data structure will be useful represent the discard piles?**

- ☐ *1.2.4.1) Unshuffled Deck = queue*
- ☐ *1.2.4.2) Shuffled Deck = stack*

- [ ] *1.2.4.3) Discard piles = stacks*
- [ ] *1.2.4.4) Victory pile = stack*

- [ ] **1.2.5. Describe your strategy for utilizing your discard piles.**
  - [ ] *1.2.5.1) Optimize the difference between the cards and keep in descending order*

☐(1.3) Implementation

- ☐ **(1.3.1) Solution in vegas.py**
- ☐ **(1.3.2) Import these modules:**
  - ☐ *(1.3.2.1) rit_lib.py*
  - ☐ *(1.3.2.2) myQueue.py*
  - ☐ *(1.3.2.3) myStack.py*
  - ☐ *(1.3.2.4) myNode.py*

☐(1.4) Requirements

- ☐ **(1.4.1) Python 3.5**
- ☐ **(1.4.2) main() function which:**
  - ☐ *(1.4.2.1) Prompts for input corresponding to number of cards to use in the deck. (Don't validate)*
  - ☐ *(1.4.2.2) Prompt for input corresponding to the number of games to play in the simulation (don't validate)*
- ☐ **(1.4.3. Call functions to setup and play the game the requested number of times**
- ☐ **(1.4.4. At simulation conclusion, output the following:**
  - ☐ *(1.4.4.1) Average number of cards on the victory pile (floating point)*
  - ☐ *(1.4.4.2) Maximum number of cards ever achieved on the victory pile*
  - ☐ *(1.4.4.3) Minimum number of cards ever achieved on the victory pile*
- ☐ **(1.4.5. Use the above listed modules to create queues and stacks**
- ☐ **(1.4.6. May not use lists for this assignment - 0 points for implementation**
- ☐ **(1.4.7. For full credit, you must employ a strategy for utilizing the discard piles that results in an average victory pile size of >=4 when tested with a card deck size of 10. If that number reaches >=5. 5% bonus**

☐(1.5) Additional Analysis

- ☐(1.5.1) **Vegas lets you decide how many cards you want in your deck. What should you choose? What deck size results in the maximum expected size of the victory pile? Is this reasonable?**
- ☐(1.5.2) **Should you always be able to get the entire deck onto the victory pile when the deck is size 2?. What about 3? What about 4? Does your solutions achieve your expectations?**
- ☐(1.5.3) **For a deck of size k, how man discard piles do you need in order to guarantee that all cards can be placed onto the victory pile? What is the order that the cards are flipped over that requires this number of discard piles?**

☐(1.6) Grading & Submission

- ☐(1.6.1) **20% - use all 4 modules**
- ☐(1.6.2) **25% - simulates the game play correctly**
- ☐(1.6.3) **25% - employs a strategy that has an average victory pile of 4 or more when deck = 10 cards**
- ☐(1.6.4) **10% - correct input, simulates the game the requested number of times, and displays the requested output information**
- ☐(1.6.5) **5% - correctly documented and follows style guidelines**
- ☐(1.6.6) **5% bonus if can achieve a average victory pile size of >=5 with deck size of 10**
- ☐(1.6.7) **Submit Vegas.py to dropbox**