**Marima Andrew Mambondiumwe**

**A12-CSC 236 Reflection**

**The dealing pile**

I used a queue to deal 25 cards to both players (the player and the computer), always taking the one at the front. The cards are shuffled each time and the players can take the cards starting the top of the pile. Queues use the FIFO system. The first cards that were placed on the top will be the first to be taken out. The reason I chose this is because it will be easier to handle the dealing pile. Whatever card is on the top will be first to be dealt to the players. I found it easy to work with it because all I am doing is taking each card from the top without having to worry about order.

**a player's playing pile**

I used a stark in this instance because in a stack we remove the item that was most recently added. In the player’s playing pile as soon as that stack is added it is going to quickly be removed and played. Stacks works perfectly in this situation because we remove the cards that were recently added to the player’s pile from the dealing pile. It wouldn’t make sense to use a queue because in a queue we remove the items that were not recently added. In this case we are constantly adding to the player’s pile because the player will be using the cards that are recently added to him. The player will be able to play when the cards have been added so he can remove and play around with them quickly.

**a player's storage pile**

I used a stark for this method so that I could keep order whenever I pile cards in the storage, I would know where to start pulling from the pile. Whenever the playing pile has been filled by completely emptying the storage and cards have been added to storage one by one and we can only take cards from the top we know we have to use data structure stack. I know since I have stacked the cards in the storage, the only way to get them out is to start from the top, taking one by one. It is efficient that way the player can just start from the top of the pile and take one by one as many as necessary.

**the opponent's playing pile**

Similarly, to player’s playing pile, I used a stack to remove the items that were recently added. The first pile of cards added from the dealing pile to the pile of the opponent will be immediately removed or played by the opponent is it will be last in first out (LIFO). The reason I used stacks is because the opponent has a stack of cards and whatever he has now on his hand he can play it and remove it quickly.

**the opponent's storage pile**

I used a stark on this one because, the opponent will keep taking the cards he has in storage till he runs out. We know the cards in the storage are stacked one by one on top of each other. The last card that was stacked will be the first to be taken by the opponent. Let’s say the opponent needs five cards for the next round. Certainly, he won’t take the first ones out of the pile but the last cards on top of the pile. Starks allow the opponent to take the last cards in the storage pile out and use it for the playing pile. The way the cards are arranged in the storage (on top of another) gives a hint to use stack.

**loot pile**

I used a stark for this one because whatever cards the players put down on the pile in the middle of the table will allow them to perhaps take the cards the other opponent left leaving them with an advantage in the game. LIFO is the data structure in this case because two cards are placed on top of the loot pile and the player and computer player won’t know what card to expect when they draw from that pile again, in case of “war.”

**Precisely describe what the data structure should be for the initial pile that needs to be shuffled, and why it may or may not be different from the data structure for the dealing pile.**

Stack in this case would be the best option to shuffle. Stacking in this case if very flexible and allows you to shuffle the cards in a way that does not allow the players to win quickly. This is different from the dealing pile because I used a queue for the dealing pile. Once the cards have been shuffled and are ready to hand out to the players, it is as if the cards are in a line waiting to be taken by one of the players.

When shuffling the cards, you are taking cards that are at the end and placing them on top of the deck. At times you will take some of the cards from the top of the stack and place it somewhere in the middle or end so the cards are shuffled for the fairness of the game. I would use a stack in this case because stacks allow you to always add items to the front and always allow you to remove items from the top/front. It wouldn’t be wise to use a queue here because when using a queue, you would have to add only at the end of the list and remove from the front of the list.

**Most design choices have advantages as well as disadvantages. Describe the primary advantages of using a stack or a queue data structure, which is admittedly restricted in how you would use it, versus a Python list for the last three piles in the list above.**

Stacks and queues both offer a pointer. The pointer is stored for every item in the stack/queue and then whenever you need to find something, each items points to the other item until the wanted item has been found. The pointer is very important in the game we played with cards.

Speed of stack and queues are much higher than linked list based. Also the memory is higher in stack/queue than linked list based.

**Describe at least one disadvantage of using a stack or a queue data structure, versus a Python list for the last three piles in the list above.**

Even though stacks/queues are great to use, stacks/queues are not wise to use when dealing with O(N). We know that stacks and queues are always going to be O(1) and they cannot handle this case with they have to deal with many items.

**Imagine you are in a hackathon where you are restricted to using a single data structure for all of the piles and you have to choose between using all stacks and using all queue. Explain whether you would choose all stacks or all queues if you want to have the smallest impact on how the game functioned? Explain what changes and why you made the choice you did.**

I think in this case I would use stacks that way whenever I would be piling different data in different piles, I would still be able to remove data easily from one pile to another. Stacks allow you to remove the last object from the pile to be the first to come out of the pile (LIFO) that way if I put the data in the wrong pile, I can easily remove it without having to wait for a line (like I would have in a queue). I believe I would be very efficient in playing around with how I would want the game to function. I would define an array and list the changes/modifications I would have to see as its elements and change it around. I would change the number of players, allowing players to group in teams and see which team wins, and also winner and loser price. I would be dealing with three piles in this case and depending on what groups the players would want to be I would arrange the kind of challenge they would have to take on to win. I think stacks on this case would make the game flexible and efficient. Obviously I would not be able to do as much as I would want to do with the game knowing stacks are O(1).