Benjamin- Player’s Turn Method

In my experiment, I created a function to simulate two players inputting a number (between 20-30) and seeing who had the bigger number. The one with the bigger number wins. After a player wins 5 times, they win that round. This will parallel the final project where each player will input the card (that matches the suit if they have it, or choose whatever they want) they want for that trick, and see who has the highest number and gets their designated point. After a player receives a certain number of points, they win.

From what I’ve found out, try-except blocks are really important, especially so that a player will select a card, and more importantly, a card with a matching suit if they have one (In my experiment’s case, choosing a number, and more importantly, within that range). One way for each player to have a turn is to have multiple input statements for each player. Since the point threshold won’t be met until after a couple of games, the player's turn method should be in a While True loop until the point has been met, and the loop can be broken.

My script is the BZ_Experiment.py. To run the code, run python3 BZ_Experiment along with the name of the two players. Example below is:

```
Python3 BZ_Experiment Bob Sam
```

Tolu - High Score Method

We all worked on our experimental methods on the python script Final_Project_326.py. In my experiment of the problem, I worked on the high score method which basically reads a txt file of the top 10 players in the different rounds of the game as a list and then updates the list at the end of each round with the name and score of the player with the new high score. If a new high score is found, the player with the lowest score in the list should be removed and the player with the highest score is added to the list. If the list is at its maximum, which is 10, the scores will just replace each other but the length of the list will remain unchanged. If a score is to be added to the middle of the list which accounts for if the player’s score from a round is lower than the highest score but higher than the lowest score, the lowest score should be removed from the list and the new score should be added to the appropriate place in the list while it remains sorted. Then once these changes have been made, the list should be written back into the txt file in order to have an updated txt file.

Things I figured out:

I was able to figure out how to read the content of the text file into a list. For the sake of this experiment I had 2 input statements where the user can input a desired username and score then that information gets added to a specific part in the list depending on what the score was. I was able to figure out how to add the highest score and remove the player with the lowest score.

To run it: The python script “Final_Project_326.py” contains my high score method and to run it, the highscore.txt file is needed or any other txt file that contains two categories which is the player’s name and the player’s score. No module is needed to run what I have so far, the user just has to call the function and pass in the argument and the result will be printed in the console.

Mario - Deal Method

The problem I set out to solve was dealing the actual cards to the players. We start off by using one of the future functions we’ll code called deck() which will return a list of card objects (6-10,J-K, Aces minus an ace of spades). Once we transfer that list of card objects into the deal method, we need to shuffle that list of card objects. We do this by importing the module random so we can use its shuffle function. Once we have shuffled the list of card objects, we’re going to need to give each player 5 cards so I made a counter variable outside a while loop. For every player, the while loop will pop out a card object using shuffledDeck.pop() 5 times (keeping track using a counter variable). This method might have to be called after every turn or we could probably use the turn method. As of now, I’m not sure if I should be using pop() on the shuffled deck or if I have to iterate through it and give each player 5 cards starting from the end of the list though. We possibly would have to make an extra method within the Spur class to be able to return the number of players in the game at any given time as well so I could use it for the for loop that’ll give each player their cards.

Tony- Computer’s Turn Method

For my experiment I have a general idea of how a method for the computer’s turn would work. The method should be able to assess its own hand of cards dealt to it and determine which is the best card to play based on the current card on the top of the play pile. The method will take note of the suits and ranks of the cards. If the computer’s hand contains cards with matching suits it will make a separate list for those cards. If it does not have any matching suits it will simply use the entire given hand. With a list of matching suit cards it will play the card with the highest rank in an attempt to win the trick. With a hand with no matching suits, it will play a card with the lowest rank to save higher ranked cards for future tricks. The method will then return the card the computer wants to play based on those conditions. We figured out that we may need a separate class for cards so that we can possibly make a global list for a deck of cards.

Yanni- Deck Method

In my experiment I have worked with lists before and decided to apply the knowledge to create a function called deck that creates and sets each card with their suits. In this method I first initialize the list of suits and face cards of the deck. Then it will return the card's value corresponding to their suit. For example, the king of hearts. In this method case it will return a total of 35 cards instead of the traditional 52 since the lowest value card is 6. From this I was able to figure out how to append a list of cards with their values.