Yousuf Qari
Final Project Report
12/12/2021

To begin, it is worth noting that I have thoroughly enjoyed creating this game and overall navigating how to determine all the possibilities for either of two players to win this fairly simple game of cards. I relied heavily on assigning variables, importing programs, using conditions, and using booleans to create while loops. To describe my program, I will describe a walkthrough of the main program, main.py, from which one could run the entire game. The first step of the main function is to create a boolean to use in a while loop and assign three zero values to three variables representing the wins for each player and the draws. Then, the code asks for a file name for a CSV file that would hold the record information. After opening the while loop with the boolean, the program calls a function, *write_and_read_csv,* from another program called *game_functions.* I separated most of the functions related to the CSV file, dealing cards, and trading cards into its own program in order to stay organized and allow for more ease in debugging. The function writes a CSV file with the name provided as input, including the wins for each player and the draws. Then the program reads the CSV file so the users know the current record. In addition, I include two lines for exception handling in case the file cannot be found. Once that is done, the main function calls on another function named *create_deck* from the program *card_deck* which is imported in. Using this and the random module, I assign four random cards from a deck of all the cards to four variables. To create the deck, I create an empty list and two lists with all the names of attackers and defenders. Then, using a for loop, I create a dictionary for each card including information about its strength, group and name so I could append it to the initially empty list. After this, I couple each two the card dictionaries and assign them to variables representing the players, so each player has two cards. Then I use the *card_score* function which adds an attacking score, defending score, or both depending on the combination of cards belonging to a player. After the four cards are created, I assign attributes of each card to many variables for easy use. After 'dealing' the cards and their values to the users, the main function calls on a function called *trade_request*, in *game_functions.* This function asks each player whether they want to trade or stick with their cards. If they decide to trade, it calls a function called *trade_card*, which deletes the card they do not want from the player's card

```
def create_deck():
    deck = []
    attacking_cards = ['Cat', 'Eagle', 'Liger', 'Sea serpent',\
        'Gargoyle', 'Hydra','Vampire', 'Giant', 'Werewolf', \
            'Dragon']
    defending_cards = ['Dog', 'Owl', 'Gnome', 'Mermaid', \
        'Fairy', 'Centaur', 'Hippogriff', 'Sphinx', 'Gryphon', \
            'Unicorn']
    for attacker in attacking_cards:
        card = {}
        card['name'] = attacker
        card['group'] = 'Attack'
        index = attacking_cards.index(attacker)
        card['strength'] = index + 1
        deck.append(card)
    for defender in defending_cards:
        card = {}
        card['name'] = defender
        card['group'] = 'Defence'
        index = defending_cards.index(defender)
        card['strength'] = index + 1
        deck.append(card)
    # print (deck)
    return deck
```

```
def trade_card(hand):
    card_1_name = (hand['card 1']['name'])
    card_2_name = (hand['card 2']['name'])
    option_bool = True
    while option_bool:
        card_to_replace = str(input('\nWhat card would you like to trade out?\
            \n1. ' + (card_1_name) +\
            '\n2. ' + (card_2_name) + '\n'))
        deck = card_deck.create_deck()
        new_card = (random.choice(deck))
        card_to_replace = card_to_replace.capitalize()
        option_1 = ('1' in card_to_replace) or (card_1_name in card_to_replace)
        option_2 = ('2' in card_to_replace) or (card_2_name in card_to_replace)
        if option_1:
            del hand['card 1']
            hand ['card 1'] = new_card
            del hand['attacking score']
            del hand['defending score']
            del hand['combination']
            card_1 = hand ['card 1']
            card_2 = hand ['card 2']
            # print (card_1, card_2)
            hand = card_score(card_1, card_2)
            option_bool = False
            return (hand)
        elif option_2:
            del hand['card 2']
            hand ['card 2'] = new_card
            del hand['attacking score']
            del hand['defending score']
            del hand['combination']
            card_1 = hand ['card 1']
            card_2 = hand ['card 2']
            # print (card_1, card_2)
            hand = card_score(card_1, card_2)
            option_bool = False
            return (hand)
        else:
            print('Please choose between card 1 and card 2.')
```

dictionary and uses *card_score* to add a random new card. In order to make the function able to withstand inputs that do not fit, it is made with a while loop and a boolean, and uses a logical operator, "or", to accommodate for the users to either provide the menu option or the card name when choosing the card they wish to discard. After this, the main program prints the new cards if either user decides to trade and then compares the two players' cards in the *matchup_functions* program. In this program, the *matchup* function evaluates the card combination matchups and calls a function the same program to determine the winner for this scenario. For example, a matchup between two attack cards and two attack cards would have the *matchup* function call *both_attack.*

```
elif player_1_cards['combination'] == 'two attack' and player_2_cards['combination'] == 'two attack':
    result = both_attack(player_1_cards,player_2_cards)
    return result
```

This function compares the value associated with the 'attacking score' key in each players' card dictionaries, and returns either 'player 1 wins', 'player 2 wins', or 'draw', depending on which is greater.

```
#determines who will win
def both_attack(player_1_cards,player_2_cards):
    if player_1_cards ['attacking score'] > player_2_cards ['attacking score']:
        print ('player 1 wins')
        return 'player 1 wins'
    elif player_1_cards ['attacking score'] < player_2_cards ['attacking score']:
        print ('player 2 wins')
        return 'player 2 wins'
    elif player_1_cards ['attacking score'] == player_2_cards ['attacking score']:
        print ('draw')
        return 'draw'
```

Once the value is returned to *matchup*, it will be returned back to the main function. Based on the result, the main function uses conditionals to add wins or a draw to the variables assigned at the very beginning of the function, which would be changed in the CSV file as well. After asking the users whether they would like to play again, it either continues the game and restarts the while loop initialized at the beginning of the main function or tells the final record of wins by calling the function *write_and_read_csv* and printing a thank you to the user for playing. For the test file, all the functions in the class *TestCardGame,* follow the same pattern. First, they assign test variables, call the functions they are testing with the test variables, and then use assertEqual to see if the return value of the functions matches what they are supposed to.