Description

This game is a modified version of the Monopoly Deal card game, whose rules can be found at http://monopolydealrules.com/.

The objective of this game is to complete three sets of coloured properties on the playing field. Each set is only complete when the player has all the properties associated with that colour.

In our version, we adopted various features, in terms of the action cards and rulesets. However, we omitted the monetary features and attributes in our version of the card game.

This game can be played between two to four people, and each player would be distributed a starting hand of five cards, taken from a shuffled deck.

The complete deck has 52 cards – 30 property cards (3 properties per colour, 10 sets in total), 10 Pass Go cards, 2 Dealbreaker cards, 3 Just Say No cards, 4 Forced Deal cards and 3 Sly Deal cards.

When it is the respective player's turn, they are to draw two cards. The player then can choose to either:

1. Play an action card (except for the Just Say No card, as it is a reactive card to other action cards)
2. Play a property card to set down on the playing field
3. Take back one of their property cards on the playing field (if they have any)

Players are able to play action cards to influence the game – Pass Go allows them to draw two cards from the deck, Forced Deal and Sly Deal allows the trading or stealing of properties from other players respectively, Dealbreaker allows the stealing of completed sets of properties from other players and Just Say No is able to cancel out the trading or stealing of properties/sets.

Each player is granted a maximum of three actions per turn, or can end their turn prematurely if they choose to do so. Each player can only have a maximum of 7 cards in their hand, and have to redraw 5 cards from the deck if they run out of cards.

When a player completes a set, all properties associated with that set is regarded as one complete set instead of as their individual properties on the playing field instead. At this point, the completed set can only be taken by other players only via a Dealbreaker card.

The game ends when a player has collected at least 3 completed sets, and the player would be displayed as the winner.

Documentation

It is required for this game to import these various libraries:

1. Random , specifically the shuffle, random integer (or randint), and seed functions (for the random shuffling of the deck)
2. IPython.display, specifically the clear output function (to clear the output between each turn)
3. Time, specifically the sleep function (this is not a required library, but included to slow the code down to be more user-friendly for players)

Variables

- **complete_deck** is a list that contains all the cards within the deck. It is used more as a sample list that would be unchanged, for reference purposes
- **property_cards** is a list that contains all the property cards within the deck
- **action_cards** is a list that contains all the action cards within the deck
- **complete_{insert colour here}** are lists that correspond to the 10 different colours of properties that contain 3 properties for each colours (3 elements in each list), and will be used to define the cards needed to complete each set *(example: complete_brown = ['Brown_1', 'Brown_2', 'Brown 3')*
- **deck** is defined at the beginning of the master/main function *monopoly_deal()* as a shallow copy of the complete_deck list, to be used for the various functions defined
- **discard_pile** is a list that would contain all the cards that are used up/discarded from the players' hands, which would ultimately be reshuffled back into the deck

Functions for start of game and beginning of turn

intro(). This function prompts the user for an input of the number of players, only accepting the string values of 2,3 or 4, and otherwise asks for the user to keep inputting till one of these values are given. The function returns the integer value of the number of players in the game.

This function further sets the global variables for each player, for example as follows:

- **player1_hand** contains a list of the cards in Player 1's hand
- **player1_property** contains a list of cards in Player 1's property playing field

- **player1_set** contains a list of the completed sets in Player 1's playing field
- **player1** contains a dictionary containing the three lists above as values to their corresponding keys "Hand", "Property", and "Set". It also contains "Number", which corresponds to the player number, in this case an integer value of 1.

Global variables are used as we need to access these number of variables that are called, outside this function. This function does not set the global variables that are outside the range of the integer value of number of players given beforehand. For example, if there are only 3 players, only global variables of Players 1,2 and 3 will be set. This is accomplished using nested if statements.

check_shuffle_deck(deck,discard_pile). This function takes in two parameters - the deck list as well as the discard_pile list.

It checks the length of the deck, and if is has less than 5 elements (or cards) left in the deck, it would print a statement to notify and adds the cards from the discard_pile list back into the deck list using the extend method, before clearing the discard_pile of all its elements using the *clear()* function.

A random seed between the integer values of 0 to 1000 (inclusive) is then generated using *seed(randint(0,1000))*, and this seed is used in tandem with the shuffle function on the deck list to randomize the order of the elements/cards. The *seed()* and *random integer (randint(0,1000))* functions are used to lower the pseudo-randomness and increase the true randomness of the *shuffle(deck)* function than if it was used by itself.

draw_card(deck, player_hand). This function takes in two parameters - the deck list as well as the current player's hand list.

It takes the 0^{th} index element from the deck list, appends to the current player's hand list, before removing from the deck list.

discard_card(deck,player_hand,discard_pile). This function takes in three parameters - the deck list, the discard_pile list, as well as the current player's hand list.

It prints the current player's hand, and prompts the player to choose the card they want to discard. If the card does not exist in the player's hand, it would keep prompting the player to input an appropriate card within his/her hand.

It then appends the card chosen as a string value into the discard_pile, whilst removing it from the player's hand.

start_game(total_player_number). This function takes in one parameter - the total number of players that is returned from the *intro()* function beforehand.

It creates two global lists that are to be called in other functions, **player_ls** and **total_players**. *player_ls* (example player_ls = [1,2,3] for 3 players) would be used to choose a player based on their player number, whilst *total_players* would be a list of the appended players' dictionaries.

Depending on the total number of players, the function would prompt each player for their player name, and appends the name given as a new key ("Name") and value (the string input of the name given) into their respective player dictionaries. As of this point, there are 5 keys in each player's dictionary to be called upon ("Hand", "Property", "Set", "Number", and "Name).

Finally, this function passes out cards to each player via the *draw_card(deck,player_hand)* function, until the last player has 5 cards.

start_turn(n). This function takes in the number of the n-th player. It returns the respective player's dictionary, as well as the lists corresponding to their hand, property, set and string value of their name. If n is 2, it will return Player 2's variables, to be used in other functions until the turn ends.

Preliminary functions for action cards

take_property(taker_property,target_property,card_taken). This function takes in three parameters – the lists of current player's properties on the playing field, the target's properties on the playing field, and the string value of the card chosen. It appends the card chosen to the

current player's properties on the playing field, whilst removing the card chosen from the target's properties on the playing field.

take_set(taker_set,target_set,set_taken). Similar to the function above, except that it removes the completed sets from list of the target's playing field, and appends it to the list of the current player's playing field.

trade_property(take_property, target_property, card_taken, card_given). This function takes in four parameters - – the lists of the current player's properties on the playing field, the target's properties on the playing field, the string values of the card chosen by the current player to take, and the card chosen by the current player to trade in exchange for card taken. It appends the card taken to the current player's properties on the playing field, whilst removing the card chosen from the target's properties on the playing field, and does the vice versa for the card chosen by the current player to trade in exchange for the card taken.

choose_target(player_number). This function takes in one parameter – the list of the current player's player number. It returns the lists for the target chosen's hand, property and set.

It creates a list named *player_names* that contains all players' names except for the current player's, and prompts the current player to pick one of the players by name. If the target chosen is not within the *player_names* list, it will continue prompting until a valid answer is chosen.

Once a target is chosen, it assigns the hand, property, and set of the individual chosen as the targeted hand, property, and set for the targeting action card (Forced Deal, Sly Deal, or Dealbreaker) used.

Action card functions

pass_go(player_hand). The function receives one parameter – the list of the current player's hand. It uses the *draw_card(deck,player_hand)* two times and then displays the player's current player's updated hand. It then proceeds to use the

check_shuffle_deck(deck,discard_pile) function to decide whether the deck has to be shuffled again with the discard_pile added back into it.

just_say_no(). This function returns the boolean value *isaidno*, which is used outside of its function in the targeting action cards' functions to decide whether the targeted player is using their Just Say No card in reaction to the current player's targeting action card.

It prompts the user to answer with a "Yes" or "No" to whether they will be using their Just Say No card. If "Yes", it returns a boolean value of True for isaidno, otherwise it is False.

forced_deal(taker_property,target_property,target_hand). This function takes in three parameters – the lists of current player's properties on the playing field, the target's properties on the playing field, and the target's hand.

It checks if the current player and the target has any properties down on the playing field. If there are none on either or both players' property playing fields, the card is discarded and the action ends.

Otherwise, the function displays the current player's property playing field and prompts the current player to choose a property to trade away. If the string value input does not correspond to any element in the list of the player's property playing field, it will keep prompting till a valid answer is chosen.

Afterwards, it displays the target's property playing field. Similarly, it prompts the current player to choose a property to take. If the string value input does not correspond to any element in the list of the target's property playing field, it will keep prompting till a valid answer is chosen.

The target is now given the open to use their Just Say No card, via the *just_say_no()* function. The possible scenarios are as follows:

1. If the target says no, the action continues.
2. If the target says yes, but does not own a Just Say No card, the action continues

3. If the target says yes, and does own a Just Say No card, the action is stopped and both Just Say No from the target hand and Forced Deal card from current player's hand is discarded.

If there is no Just Say No card used, and the action continues, the *trade_property(take_property, target_property, card_taken, card_given)* function is used, and the final properties of the current player and target's respective playing fields are displayed.

sly_deal(taker_property,target_property,target_hand). This function takes in three parameters – the lists of current player's properties on the playing field, the target's properties on the playing field, and the target's hand.

It checks if the target has any properties down on the playing field. If there are none, the card is discarded and the action ends.

Otherwise, the function then displays the target's property playing field, and prompts the current player to choose a property to take. If the string value input does not correspond to any element in the list of the target's property playing field, it will keep prompting till a valid answer is chosen.

The target is now given the open to use their Just Say No card, via the *just_say_no()* function. The possible scenarios are as follows:

1. If the target says no, the action continues.
2. If the target says yes, but does not own a Just Say No card, the action continues
3. If the target says yes, and does own a Just Say No card, the action is stopped and both Just Say No from the target hand and Sly Deal card from current player's hand is discarded.

If there is no Just Say No card used, and the action continues, the *take_property(taker_property,target_property,card_taken)* function is used, and the final properties of the current player and target's respective playing fields are displayed.

deal_breaker(taker_set,target_set,target_hand). This function takes in three parameters – the lists of current player's completed sets on the playing field, the target's completed sets on the playing field, and the target's hand.
It checks if the target has any completed sets down on the playing field. If there are none, the card is discarded and the action ends.

Otherwise, the function then displays the target's completed sets playing field, and prompts the current player to choose a completed set to take. If the string value input does not correspond to any element in the list of the target's completed set playing field, it will keep prompting till a valid answer is chosen.

The target is now given the open to use their Just Say No card, via the *just_say_no()* function. The possible scenarios are as follows:

1. If the target says no, the action continues.
2. If the target says yes, but does not own a Just Say No card, the action continues
3. If the target says yes, and does own a Just Say No card, the action is stopped and both Just Say No from the target hand and Dealbreaker card from current player's hand is discarded.

If there is no Just Say No card used, and the action continues, the *take_set(taker_set,target_set,set_taken)* function is used, and the final completed sets of the current player and target's respective playing fields are displayed.

put_down_property(player_hand,player_property,card_played). This function takes in three parameters – the lists of the current player's hand, their property playing field, and the string value of the card played.

It appends the card chosen by the current player into the list of their property playing field, and removes it from the list of the current player's hand, before displaying the updated lists.

take_back_property(player_hand,player_property, card_played). This function takes in three parameters – the lists of the current player's hand, their property playing field, and the string value of the card played.

It removes the card chosen by the current player from the list of their property playing field, and appends it into the list of their hand, before displaying the updated lists.

action(player_hand,player_property, player_set,player_name). This is the main function for all the action functions. It receives four parameters – the lists of the current player's hand, property playing field, completed sets playing field, and name.

It assigns a counter value, acts, an integer value of 0, and runs a while loop. While acts is less than three:

1. The previous outputs are cleared using the clear_output function imported
2. It displays which player's turn it is, by player name
3. The lists of the current player's hand and properties on their playing field are sorted alphabetically, using the *.sort()* function
4. It displays the lists of every player's properties on the playing field, as well as their completed sets
5. The action number is displayed (for the current player to keep track of number of actions done)
6. *check_shuffle_deck(deck,discard_pile)* is executed to ensure deck has enough cards to be drawn from
7. The lists of the current player's hand and property playing field are displayed
8. The counter, acts has an integer of 1 added to it
9. The function prompts the current player to pick a card to play/take back
    a. If the current player inputs "Done", the turn is ended immediately, regardless of the counter, act's integer value.
    b. If the card chosen is an action card (except for Just Say No), execute the action by its function
    c. If the card chosen is Just Say No, a message is displayed that the card cannot be used proactively, and that the action turn is wasted. Card is kept.
    d. If the card chosen is a property card, execute the action by its function depending on location of the property card chosen (in current player's hand or property playing field)

e. If the card chosen is not within the lists of the current player's hand, playing field, the string value "Done", or is not a valid card name, a message is displayed showing that the action turn is wasted.

10. Once three action turns have been reached, indicated by the counter, act's integer value, the turn has ended

End of turn and change of turn functions

check_hand_no(player_hand). It receives one parameter – the list of the current player's hand.

If the player has no cards left (0 elements in the list), a while loop runs until the player has drawn 5 cards from the deck back into their hand, via the *draw_card(deck,player_hand)* function. Another situation arises if the player has more than 7 cards in their hand (more than 7 elements in the list), a while loop runs until the player has less than 8 cards in their hand, via the *discard_card(deck,player_hand,discard_pile)* function. It then displays the final list of the current player's hand.

check_complete(player_property,player_set). The function receives two parameters – the list of the current player's property playing field, as well as their completed set playing field.

Counters of each individual colour of every property set is set to integer values of 0 *(example brown = 0)*. A for loop is run through the current player's property playing field to check for each corresponding card in the playing field (element in the list), whether it is within the *complete_{insert colour here}* list variables defined at the beginning.

For example, if *'Brown_1'* is within the current player's property playing field, the counter, *brown*, receives an addition of an integer value of 1, as *'Brown_1'* is within the *complete_brown* list, that contains all the brown properties as its elements.

If the counter of any of the coloured property sets reach an integer value of 3, the player's property playing field list has all the properties corresponding to that completed set's colour removed from the list, and has the string value of the completed set's colour *(example 'Brown')* appended to the player's completed set playing field instead.

(Note that though the example focuses on the brown properties, this applies to all 10 property colours too)

check_win(player_set,player_name). The function receives two parameters - the lists of the current player's completed sets playing field, as well as the current player's name. If the current player has at least 3 completed sets (at least 3 elements in the list of the current player's completed sets playing field), the function returns the current player's name. Otherwise, it returns None.

end_turn(player_hand,player_property,player_set,player_name). This is the main function for all the end of turn functions. The function receives four parameters – the lists of the current player's hand, property playing field, completed sets playing field, and name. The function returns either None or the string value for the winner's name.

The code is as follows:

1. It checks for any completed sets that was achieved after the player's overall action turns, using the *check_complete(player_property,player_set)* function
2. It assigns the value winner to the function *check_win(player_set,player_name)*
    a. If the returned value from the function is the current player's name, the function ends and returns the player's name
    b. Otherwise, if returned value is None, the number of cards in the current player's hand is checked and adjusted if necessary, using the *check_hand_no(player_hand)* function, and the function returns None.

change_turn(player_number,player_name). This function receives two parameters – the integer value of the current player's (the player passing the turn on to the next) number, as well as the string value of the current player's name.

The previous outputs during the turn are cleared using the *clear_output()* function. A prompt message displaying the end of the current player's turn is shown and the following occurs:

1. If the integer value of the current player's number is less than the integer value of the length of the list, *player_ls* assigned beforehand, add an integer value of 1 to the player number

2. If the integer value of the current player's number is equal to the integer value of the length of the list, *player_ls*, reassign the player number an integer of 1 (for example when player 4's turn changes to player 1's turn)

The function then returns the updated player's number for the next turn.

Master/Main Function

monopoly_deal() This is the function that runs the game. The pseudocode is as follows:

1. Shuffle the deck using the *seed(randint(0,1000))* and *shuffle(complete_deck)* functions, to shuffle the original copy of the full deck to a corresponding randomised seed value between 0 to 1000 (inclusive)
2. Set global variables for deck as a shallow copy of the complete_deck list, and discard_pile as an empty list, to reference in the rest of the functions
3. Assign *total_player_number* variable to the returned value of the function *intro()*, as well as to set the global variables for the various players' dictionaries and list values ("Hand", "Property", "Set", "Number")
4. Once the total player number has been decided, the *start_game(total_player_number)* function is executed to receive the global variables of the *player_ls* and *total_players*, to be used in the functions later, alongside the addition of player names as key-value pairs to the global player dictionaries assigned in the *intro()* function
5. The *winner* variable is set to None, and a global *player_number* variable is assigned an integer value of 1 (to indicate that Player 1 starts first)
6. While loop is executed from here onwards until a specific player's name is returned as the *winner* variable.
    a. *start_turn(player_number)* to assign the appropriate variables of the current player's dictionary, lists and values.
    b. Current player draw two cards from deck, using *draw_card(deck,player_hand)* function, along with a displayed message to indicate this action
    c. The overall action function is executed (refer to *action(player_hand,player_property, player_set,player_name))*
    d. *winner* variable is assigned to the overall end of turn function (refer to *end_turn(player_hand,player_property,player_set,player_name))*

e. *player_number* is updated using the change_turn(player_number, player_name) function in preparation for the next player's turn

7. Once a player's name is updated in the *winner* variable, the while loop stops and the game ends with a displayed message announcing the winning player's name.