

```

from card_elements import Card, Deck, Pile
from codecarbon import EmissionsTracker
from itertools import product
import pprint

pp = pprint.PrettyPrinter(indent=4)

with EmissionsTracker() as tracker:

    class Game:

        values = ["A", "2", "3", "4", "5", "6", "7", "8", "9", "10", "J", "Q", "K"]
        signs = {"diamond" : "red", "spades" : "black", "hearts" : "red", "clubs" : "black"}

        print("You have a full deck.")

        #print("The cards in your deck are:")
        #
        #for value, x in product(values, signs.keys()):
        #    colour = signs[x]
        #    print(f"Card: {value}, Colour: {colour}, Value: {x}")

        suits = { #keys are unicode symbols for suits
            u'\u2660': "black",
            u'\u2665': "red",
            u'\u2663': "black",
            u'\u2666': "red",
        }

        numPlayPiles = 7 # Constant

        def __init__(self):
            self.list_of_cards = [
                Card(value, suit) for value in range(1, 14) for suit in [
                    "Diamonds", "Hearts",
                    "Clubs", "Spades"
                ]

            self.deck = Deck(self.values, self.suits)
            self.playPiles = []
            for i in range(self.numPlayPiles):
                thisPile = Pile()
                [thisPile.addCard(self.deck.takeFirstCard(flip=False)) for j in range(i+1)]
                thisPile.flipFirstCard()
                self.playPiles.append(thisPile)
            self.blockPiles = {suit: Pile() for suit in self.suits}
            self.deck.cards[0].flip()

        def getGameElements(self):
            returnObject = {
                "deck": str(self.deck),
                "playPiles": [str(pile) for pile in self.playPiles],
                "blockPiles": {suit: str(pile) for suit, pile in self.blockPiles.items()}
            }
            return returnObject

        def checkCardOrder(self, higherCard, lowerCard):
            suitsDifferent = self.suits[higherCard.suit] != self.suits[lowerCard.suit]

```

```

valueConsecutive = self.values[self.values.index(higherCard.value)-1] == lowerCard.value
return suitsDifferent and valueConsecutive

def checkIfCompleted(self):
    deckEmpty = len(self.deck.cards)==0
    pilesEmpty = all(len(pile.cards)==0 for pile in self.playPiles)
    blocksFull = all(len(pile.cards)==13 for suit,pile in self.blockPiles.items())
    return deckEmpty and pilesEmpty and blocksFull

def addToBlock(self, card):
    if card is None:
        return False
    elif len(self.blockPiles[card.suit].cards)>0:
        highest_value = self.blockPiles[card.suit].cards[0].value
        if self.values[self.values.index(highest_value)+1] == card.value:
            self.blockPiles[card.suit].cards.insert(0,card)
            return True
        else:
            return False
    else:
        if card.value=="A":
            self.blockPiles[card.suit].cards.insert(0,card)
            return True
        else:
            return False

def takeTurn(self, verbose=False):

    #Pre: flip up unflipped pile end cards -> do this automatically
    [
        pile.cards[0].flip()
        for pile in self.playPiles
        if len(pile.cards) > 0
        and not pile.cards[0].flipped
    ]

    #1: check if there are any play pile cards you can play to block piles
    for pile in self.playPiles:
        if len(pile.cards) > 0 and self.addToBlock(pile.cards[0]):
            card_added = pile.cards.pop(0)
            if verbose:
                print("Adding play pile card to block: {0}".format(str(card_added)))
            return True

    #2: check if cards in deck can be added
    if self.addToBlock(self.deck.getFirstCard()):
        card_added = self.deck.takeFirstCard()
        if verbose:
            print("Adding card from deck to block: {0}".format(str(card_added)))
        return True

    #3: move kings to open piles
    for pile in self.playPiles:
        if len(pile.cards)==0: #pile has no cards
            for pile2 in self.playPiles:
                if pile2 is not pile and len(pile2.cards)>1 and pile2.cards[0].value == "K":
                    card_added = pile2.cards.pop(0)
                    pile.addCard(card_added)
                    if verbose:
                        print("Moving {0} from Pile to Empty Pile".format(str(card_added)))

```

```

        return True
    first_card = self.deck.getFirstCard()
    if first_card is not None and first_card.value == "K":
        card_added = self.deck.takeFirstCard()
        pile.addCard(card_added)
        if verbose:
            print("Moving {0} from Deck to Empty Pile".format(str(card_added)))
        return True

#4: add drawn card to playPiles
for pile in self.playPiles:
    if len(pile.cards)>0 and self.deck.getFirstCard() is not None:
        if self.checkCardOrder(pile.cards[0],self.deck.getFirstCard()):
            card_added = self.deck.takeFirstCard()
            pile.addCard(card_added)
            if verbose:
                print("Moving {0} from Deck to Pile".format(str(card_added)))
            return True

#5: move around cards in playPiles
for pile1 in self.playPiles:
    pile1_flipped_cards = pile1.getFlippedCards()
    if not pile1_flipped_cards:
        continue

    pile1_downcard_count = len(pile1.cards) - len(pile1_flipped_cards)
    for pile2 in self.playPiles:
        if pile2 is pile1:
            continue

    pile2_flipped_cards = pile2.getFlippedCards()
    if not pile2_flipped_cards:
        continue

    pile2_downcard_count = len(pile2.cards) - len(pile2_flipped_cards)
    for transfer_cards_size in range(1, len(pile1_flipped_cards) + 1):
        cards_to_transfer = pile1_flipped_cards[:transfer_cards_size]
        if self.checkCardOrder(pile2.cards[0], cards_to_transfer[-1]):
            if pile2_downcard_count < pile1_downcard_count
            or (
                pile1_downcard_count == 0
                and len(cards_to_transfer) == len(pile1.cards)):

                pile2.cards = cards_to_transfer[::-1] + pile2.cards
                pile1.cards = pile1.cards[transfer_cards_size:]
                if verbose:
                    print("Moved {0} cards between piles: {1}".format(
                        transfer_cards_size,
                        ", ".join([str(card) for card in cards_to_transfer])
                    ))
                return True

    return False

def simulate(self, draw = False, verbose=False):

    # clear cache if last turn was not card draw
    if not draw:
        self.deck.cache = []

    turnResult = self.takeTurn(verbose=verbose)

```

```

if turnResult:
    self.simulate(verbose=verbose)

else:
    #End: draw from deck
    if len(self.deck.cards)>0:

        currentCard = self.deck.cards[0]

        if currentCard in self.deck.cache:
            if verbose:
                print("No more moves left!")
            return

        else:
            self.deck.drawCard()
            #if verbose:
                #print("Drawing new card: {}".format(str(currentCard)))
            self.deck.cache.append(currentCard)
            return self.simulate(draw=True, verbose=verbose)

    else:
        if verbose:
            print("No more moves left!")
        return

def merge(self, arr, l, m, r):
    n1 = m - l + 1
    n2 = r - m

    L = [0] * (n1)
    R = [0] * (n2)

    for i in range(0 , n1):
        L[i] = arr[l + i]

    for j in range(0 , n2):
        R[j] = arr[m + 1 + j]

    i = 0
    j = 0
    k = l

    while i < n1 and j < n2:
        if L[i] <= R[j]:
            arr[k] = L[i]
            i += 1
        else:
            arr[k] = R[j]
            j += 1
        k += 1

    while i < n1:
        arr[k] = L[i]
        i += 1
        k += 1

    while j < n2:
        arr[k] = R[j]
        j += 1
        k += 1

```

```

def mergeSort(self, arr, l, r):
    if l < r:
        m = (l+(r-1))//2

        self.mergeSort(arr, l, m)
        self.mergeSort(arr, m+1, r)
        self.merge(arr, l, m, r)

def main():

    thisGame = Game()
    thisGame.simulate(verbose=True)
    print()
    pp.pprint(thisGame.getGameElements())
    print()
    if(thisGame.checkIfCompleted()):
        print("Congrats! You won!")
    else:
        print("Sorry, you did not win")

    sorted_cards = thisGame.
        mergeSort([card.value for card in thisGame.deck.cards], 0, len(thisGame.deck.cards)-1)
    print("Sorted cards:", sorted_cards)
    return

main()

```