# Part 3 go-fish

## game-of-go-fish.py

import go\_fish

from itertools import permutations

import random

class Game:

def \_\_init\_\_(self, number\_of\_players=4):

# get players

self.players = []

self.losers = []

for i in range(number\_of\_players):

new\_player = go\_fish.Player("Player" + str(i))

self.players.append(new\_player)

# randomize starting player

random.shuffle(self.players)

# get deck and shuffle

self.carddeck1 = go\_fish.CardDeck()

self.carddeck1.shuffle()

def get\_card\_from\_deck(self):

try:

card = self.carddeck1.give\_card()

return card

except:

self.game\_over()

def deal\_cards(self):

# deal 5 cards to each player

for player in self.players:

cards = self.carddeck1.give\_n\_cards(5)

player.get\_cards(cards)

def deadly\_duel(self, player1, player2):

suit = player1.random\_suit()

cards\_of\_suit = player2.get\_asked\_for\_cards(suit)

if player2.out\_of\_cards():

self.losers.append(player2)

self.players.remove(player2)

if len(cards\_of\_suit) > 0:

player1.win\_cards(cards\_of\_suit)

return True

# GO FISH

if len(cards\_of\_suit) > 0:

card = self.get\_card\_from\_deck()

player1.get\_card(card)

return False

def play\_game(self):

# start game

game\_on = True

self.deal\_cards()

while game\_on:

for player\_combo in permutations(self.players, 2):

# player1 asks

player1 = player\_combo[0]

player2 = player\_combo[1]

if player1 in self.losers or player2 in self.losers:

continue

win = True

while win:

if player2 in self.players:

win = self.deadly\_duel(player1, player2)

else:

win = False

if len(self.players) < 2:

game\_on = False

self.game\_over()

def game\_over(self):

pass

def get\_winner(self):

winner = go\_fish.Player("zero points")

for player in self.players:

if player.points > winner.points:

winner = player

return winner

if \_\_name\_\_ == "\_\_main\_\_":

# run simulation n times

n = 1000

winners = {}

for i in range(n):

print("new game", i)

game = Game()

game.play\_game()

winner = game.get\_winner()

try:

winners[winner.name] += 1

except:

winners[winner.name] = 1

print("SIMULATION OVER")

print("player name, wins")

for winner in winners:

print(winner, winners[winner])

## go-fish/CardDeck.py

import random

class CardDeck:

def \_\_init\_\_(self):

self.nos = "A 2 3 4 5 6 7 8 9 10 J Q K".split(" ")

self.suits = ['spades', 'diamonds','clubs','hearts']

self.cards = self.get\_card\_deck()

def \_\_len\_\_(self):

return len(self.cards)

def append(self, card):

self.cards.append(card)

def get\_card\_deck(self):

cards = []

for i in self.nos:

for j in self.suits:

cards.append(i + ' ' + j)

return cards

def give\_card(self):

""" gives single card or throws error """

if len(self.cards) > 0:

card = self.cards.pop()

return card

else:

raise Exception("Error. No Cards in Deck.")

def give\_n\_cards(self, n):

""" gives n cards """

cards = []

for i in range(n):

cards.append(self.give\_card())

return cards

def shuffle(self):

random.shuffle(self.cards)

def get\_deck(self, cards):

self.cards = cards

def add\_card(self, card):

self.cards.append(card)

def add\_cards(self, cards):

self.cards = self.cards + cards

def get\_cards\_of\_rank(self, rank):

cards\_of\_rank = []

for card in set(self.cards):

try:

if rank in card:

cards\_of\_rank.append(card)

except:

print("error")

return cards\_of\_rank

def remove\_card(self, card):

if card in self.cards:

self.cards.remove(card)

def remove\_cards(self, cards):

for card in cards:

self.remove\_card(card)

def random\_suit(self):

suits = list(self.suits)

random.shuffle(suits)

random\_suit = suits.pop()

return random\_suit

## go-fish/Player.py

from go\_fish.CardDeck import CardDeck

class Player:

"""

Player of go-fish, can

- get 5 cards

- ask other player for card of given rank

- get asked for card of given rank & give card of given rank (or say go fish)

- get card of given rank

"""

def \_\_init\_\_(self, name):

self.name = name

self.cards = CardDeck()

self.cards.get\_deck([])

self.points = 0

def \_\_str\_\_(self):

return self.name

def get\_card(self, card):

if type(card) is None:

raise Exception("asdf")

self.cards.append(card)

def get\_cards(self, cards):

for card in cards:

self.get\_card(card)

def win\_cards(self, cards):

for card in cards:

# print("player", self, "won cards", cards)

self.points = self.points + 1

def get\_asked\_for\_cards(self, rank):

cards\_of\_rank = self.cards.get\_cards\_of\_rank(rank)

self.cards.remove\_cards(cards\_of\_rank)

return cards\_of\_rank

def random\_suit(self):

random\_suit = self.cards.random\_suit()

return random\_suit

def out\_of\_cards(self):

return not self.has\_cards()

def has\_cards(self):

if len(self.cards) > 0:

return True

return False

returns:

*new game 0*

*new game 1*

*new game 2*

*new game 3*

*…*

*new game 999*

*SIMULATION OVER*

*player name, wins*

*Player3 249*

*Player2 253*

*Player1 242*

*Player0 256*

No, the wins are about the same.