МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное бюджетное образовательное учреждение высшего образования
«КУБАНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»
(ФГБОУ ВО «КубГУ»)

Факультет компьютерных технологий и прикладной математики

Кафедра информационных технологий

ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ № 7
по дисциплине
«МНОГОАГЕНТНОЕ МОДЕЛИРОВАНИЕ»

Выполнил студент группы 45/2 _____ Т. Э. Айрапетов

Направление подготовки 02.03.03 Математическое обеспечение и администрирование информационных систем
Курс 4

Отчет принял доктор физико-математических наук,
профессор _____ А.И. Миков

Краснодар
2024 г.

Задание:

Есть 4 агента, поделённые на соперничающие пары и взаимодействующие со средой - карточным столом. Среда содержит 52 карты, делённые на 7 подмножеств, которые определяют её состояние:

1. У агента 1 своё подмножество карт,

2. У агента 2 своё подмножество карт,

3. У агента 3 своё подмножество карт,

4. У агента 4 своё подмножество карт,

5. Открытые карты раунда на столе,

6. Колода бито,

7. Общая колода, из которой добираются карты, с нижней картой козырем.

Агент видит своё подмножество карт, открытые карты на столе и нижнюю (козырную) карту общей колоды. Процесс игры выглядит так:

1. Инициализация (перемешивание, выдача карт, определение порядка ходов, выполняется средой),

2. Игра по правилам ”подкидного и переводного дурака” по раундам,

3. Учёт результата.

В каждом раунде у пары определяется ведущий игрок. Ведущие игроки играют друг против друга, напарники же помогают тем, что подкидывают карты. Ведущим и заходящим игроком первого раунда становится игрок, у которого наименьший номинал карты козырной масти. В соперничающей паре ведущий игрок выбирается случайно.

В ходе игры заходящий игрок кладёт на стол любую из имеющихся у него карт, а отбивающийся игрок (игрок, под которого сделан заход) должен либо побить её, либо взять. Чтобы побить (синоним — покрыть) карту, нужно из имеющихся на руках карт положить на неё старшую карту той же масти, либо козыря, если битая карта — не козырь. Если битая карта - козырь, то побить её можно только старшим козырем. После того, если у игрока остаётся одна карта, то никто не имеет права атаковать его.

Заходящий игрок или его напарник могут положить (подкинуть, подбросить) ещё одну или несколько карт любой масти,-достоинство которых совпадает с достоинством любой из карт, уже участвовавших в данном заходе.

Каждый раунд выглядит так:

1. Заходящий игрок выбирает карту,

2. Если итерация первая, то отбивающийся игрок может перевести ход, если у него есть карта того же достоинства, но другой масти, и добавить её на стол, в таком случае игра продолжается со следующего пункта, но роли изменены,

3. Отбивающий игрок выбирает карту, которой побьёт карту заходящего, если такой карты нет, он обязан взять карту,

4. Если отбивающий находит подходящую карту, он бьёт карту заходящего, и тот может подкинуть карту, если он решает, что подкидывать не будет, он предлагает подкинуть карту своему напарнику,

5. Пункт аналогичен пункту 3, однако если отбивающему нечем крыть новую карту, он забирает все, что лежат на столе (в открытом множестве карт 5), и раунд заканчивается

6. Если подкидывать нечего и все карты биты, то раунд заканчивается,

7. Следующий раунд начинает пара, выигравшая предыдущий раунд.

Ротация ведущих игроков в каждой паре остаётся на усмотрение разработчиков. Если в паре у одного игрока закончились карты, то он выходит, победой считается ситуация, когда у обоих игроков закончились карты.

Задача

Разработать и описать два алгоритма игры (для каждой пары свой). Провести 1000 экспериментов и сравнить количество побед для разных алгоритмов.

Решение.

Для моделирования игр в карты 2 на 2 были реализованы классы Agent, Pair, Pair1, Pair2 и Env(среда). Класс агента содержит поле *ind* (индекс) и поле *hand* (множество карт в руке). Также агент реализует некоторые вспомогательные и общие для всех агентов методы:

- reset - переназначает руку агента новым множеством карт;
- need_cards - возвращает кол-во карт, которых не хватает агенту (от 0 до 6);
- *min_trump* - возвращает минимальный козырь агента;
- *drop* - возвращает список карт, которые агент может подкинуть на стол;
- *pick_up_cards* - метод, собирающий все карты со стола в руку агенту.

Класс пары агентов Pair содержит поля *trump* - козырь текущей игры, *agents* - список агентов (2 или 1) из данной пары. Также в классе пары агентов реализованы вспомогательные и общие для обеих пар методы:

- *swap* - смена агентов местами (в конце раунда или при переводе);
- *cards_count* - возвращает кол-во карт у ведущего игрока в паре;
- *first_hit* - возвращает карту на первом ходе в раунде;
- *beat* - попытка отбить карты на столе (возвращает True, если агент отбил карты, иначе False);
- *transfer* - попытка перевести карты на первой итерации (возвращает True, если агент перевел карты, иначе False);
- *toss_cards* - подбрасывание карт на стол (возвращает кол-во подкинутых карт);

Классы Pair1 и Pair2 наследуются от класса Pair и реализуют перегрузки некоторых методов.

Класс Env реализует все события игры (инициализация, раздача карт, определение порядка ходов, смена ролей и т.д.).

На рисунке 1 можно увидеть пример раздачи и определения порядка ходов.

```
[5, 15, 7, 4]
Min trump Agent: 3
Trump: ♠

Attack:
Agent3 [(9, '♣'), (10, '♣'), (9, '♥'), (13, '♠'), (7, '♥'), (4, '♠')]
Agent2 [(4, '♥'), (7, '♠'), (8, '♦'), (6, '♣'), (11, '♦'), (4, '♣')]

Defend:
Agent0 [(10, '♠'), (5, '♠'), (13, '♦'), (11, '♥'), (10, '♥'), (8, '♠')]
Agent1 [(5, '♦'), (4, '♦'), (3, '♦'), (12, '♥'), (3, '♥'), (8, '♥')]
```

Рисунок 1 - Выбор агента с наименьшим козырем

На рисунке 2 можно увидеть пример событий одного сыгранного раунда. Общий алгоритм моделирования таков: на первой итерации ведущий атакующей пары кладёт карту на стол, после чего ведущий обороняющейся пары должен отбить эту карту, попытаться перевести карту или объявить, что он берёт. В случае, когда он перевел карту, у агентов сменяются роли соответствующим образом. В случае, если агент отбивает карты, то должна быть проверка, можно ли подкидывать (кол-во карт > 1), и, если можно, атакующая пара подкидывает карты по мере возможности. После раунда карты уходят в множество «бито» или к агенту, который их взял, а затем у всех агентов восполняются их руки. На этом этапе проверяется вышел ли какой-либо агент. Если в какой-либо из пар вышли оба агента, то эта пара объявляется победителем и происходит выход из рекурсии. Иначе запускается смена ролей и ещё одна итерация игры.

```
Козырь: ♥
 Атака: Pair2
Agent2 {(10, '♦'), (12, '♥'), (7, '♣'), (3, '♥'), (9, '♦'), (4, '♦')}
Agent3 {(12, '♦'), (8, '♣'), (9, '♥'), (5, '♦'), (11, '♥'), (4, '♠')}

 Защита: Pair1
Agent0 {(12, '♠'), (8, '♠'), (3, '♠'), (14, '♦'), (6, '♠'), (6, '♥')}
Agent1 {(2, '♦'), (8, '♥'), (14, '♣'), (3, '♣'), (13, '♣'), (10, '♣')}

 Ход: (4, '♦')
Стол: {(4, '♦'): (14, '♦'), (4, '♠'): (6, '♠')}
Agent0 отбился
bito: {(4, '♠'), (4, '♦'), (6, '♠'), (14, '♦')}

Агенты восполняют карты из колоды

итог игры: победитель пока не определен

После смены ролей:

 Атака: Pair1
Agent0 {(12, '♠'), (2, '♠'), (8, '♠'), (3, '♠'), (6, '♥'), (11, '♠')}
Agent1 {(2, '♦'), (8, '♥'), (14, '♣'), (3, '♣'), (13, '♣'), (10, '♣')}

 Защита: Pair2
Agent3 {(12, '♦'), (8, '♣'), (9, '♥'), (5, '♦'), (11, '♥'), (2, '♥')}
Agent2 {(6, '♣'), (10, '♦'), (12, '♥'), (7, '♣'), (3, '♥'), (9, '♦')}
```

Рисунок 2 - Пример одного сыгранного раунда

В самой обычной реализации агенты отбиваются минимальными картами и по мере возможности подкидывают случайные некозырные карты. Предположим, что обе пары попытались играть более агрессивно: 1-ая пара будет первым ходом выбирать наугад карту (возможно, козырь), а 2-ая пара будет подкидывать не только обычные карты, но и козыри. После 1000 игр был получен следующий результат

```
d = {1:0, 2:0}
for i in range(1000):
    env.reset()
    d[env.play()] += 1
print(d)
```

✓ 0.9s

```
{1: 236, 2: 764}
```

Рисунок 3 - Результат запусков 1000 игр

Можно сделать вывод, что выбрасывание козырей на первом шаге более опрометчивый поступок, чем подкидывание козыря, так как при подкидывании увеличивается вероятность, что оппонент возьмет карты, а на первом ходе оппонент может просто увеличить количество своих козырей.

Для улучшения стратегий также возможен учёт вышедших из игры карт, а также карт, которые каждый агент «показал» (не смог отбить и взял). Также можно учитывать на каком моменте оппонент взял карты (если подкинутая карта была относительно небольшой по достоинству, то карты этой масти скорее всего «слабость» для оппонента). Более продвинутые стратегии также предполагают взаимодействие между напарниками в паре - если ведущий агент знает, что у напарника есть более подходящие карты для подбрасывания, то лучше уступить место.

Текст программы на языке Python:

```
import numpy as np

cards = [(i,j) for j in '♣♦♥♠' for i in range(2, 15)]

class Agent:
    def __init__(self, ind):
        self.reset(None)
        self.ind = ind

    def reset(self, hand):
        self.hand = hand
        # карты, которые видели другие игроки
        self.public = set()

    # возвращает кол-во карт до полной руки
    def need_cards(self):
        return max(0, 6-len(self.hand))

    def min_trump(self, trump):
        return min([i[0] for i in self.hand if i[1]==trump] + [15]) # 15 - нет козыря

    def drop(self, table, trump):
        """
        возвращает набор карт и козырей, которые можно подкинуть
        """
        s = set()
        for k,v in table.items():
            s.add(k[0])
            if v:
                s.add(v[0])
        t, t1 = [], []
        for i in self.hand:
            if i[0] in s:
                if i[1] == trump:
```

```
                t1.append(i)
            else:
                t.append(i)
        return (sorted(t), sorted(t1))


    def pick_up_cards(self, table):
        for k,v in table.items():
            self.hand.add(k)
            if v:
                self.hand.add(v)

    def __repr__(self):
        return f'Agent{self.ind} {self.hand}'


class Pair:
    ind = 0
    def __init__(self, agents, trump):
        self.agents = agents
        self.trump = trump

    def swap(self):
        self.agents = self.agents[::-1]

    def cards_count(self):
        return len(self.agents[0].hand)

    def pick_up_cards(self, table):
        self.agents[0].pick_up_cards(table)

    def __repr__(self):
        return f'Pair{self.ind}\n' + '\n'.join(list(map(str, self.agents)))

    # выбор наименьшей карты для первого хода
    def first_hit(self):
        a = self.agents[0]
        t = [i for i in a.hand if i[1]!=self.trump]
        if not t:
            t = a.hand

        card = min(t)
        a.hand.remove(card)
        return card

    # попытка отбить карты
    def beat(self, table):
        ag = self.agents[0]
        hand = []
        trumps = []

        for i in ag.hand:
            if i[1] == self.trump:
                trumps.append(i)
            else:
                hand.append(i)
        hand.sort() # список обычных карт
        trumps.sort() # список козырей

        # вспомогательная функция
        def min_to_beat(hand, card):
            for i in hand:
```

```
            if i[1] == card[1] and i[0] > card[0]:
                return i
        return None

    cards = []
    for k,v in table.items():
        if not v:
            t = None
            if k[1] == self.trump:
                t = min_to_beat(trumps, k)
                if not t:
                    return False
                trumps.remove(t)
            else:
                t = min_to_beat(hand, k)
                if not t:
                    if trumps:
                        t = trumps[0]
                        trumps.pop(0)
                    else:
                        return False
                else:
                    hand.remove(t)
            table[k] = t
            cards.append(t)

    ag.hand -= set(cards)
    return True

# попытка перевести карты
def transfer(self, table, opp):
    if len(opp.hand) < len(table) + 1:
        # print('Перевод невозможен')
        return False
    a = self.agents[0]
    t0, t1 = a.drop(table, self.trump)
    if t0:
        table[t0[0]] =  None
        a.hand.remove(t0[0])
        # print('Перевод')
        return True
    return False

# подкинуть карты
def toss_cards(self, table, opp, verbose=0):
    n = len(opp.hand)-1
    m = n

    # заходящий игрок
    a = self.agents[0]
    t0, t1 = a.drop(table, self.trump)
    k = min(n, len(t0))
    indices = np.random.choice(len(t0), k, replace=False)
    for i in indices:
        table[t0[i]] = None
        n-=1
        a.hand.remove(t0[i])

    if len(self.agents)==1:
        return bool(m-n)
    # напарник
    a = self.agents[1]
    t0, t1 = a.drop(table, self.trump)
```

```
        k = min(n, len(t0))
        indices = np.random.choice(len(t0), k, replace=False)
        for i in indices:
            table[t0[i]] = None
            n-=1
            a.hand.remove(t0[i])

        return bool(m-n) # подкинули карты или нет



class Pair1(Pair):
    ind = 1

    def first_hit(self):
        a = self.agents[0]
        t = a.hand
        i = np.random.choice(len(t))
        card = (list(t)[i])
        a.hand.remove(card)
        return card


class Pair2(Pair):
    ind = 2



    def toss_cards(self, table, opp, verbose=0):
        n = len(opp.hand)-1
        m = n

        # заходящий игрок
        a = self.agents[0]
        t0, t1 = a.drop(table, self.trump)
        t0 += t1
        k = min(n, len(t0))
        indices = np.random.choice(len(t0), k, replace=False)
        for i in indices:
            table[t0[i]] = None
            n-=1
            a.hand.remove(t0[i])

        if len(self.agents)==1:
            return bool(m-n)
        # напарник
        a = self.agents[1]
        t0, t1 = a.drop(table, self.trump)
        t0 += t1
        k = min(n, len(t0))
        indices = np.random.choice(len(t0), k, replace=False)
        for i in indices:
            table[t0[i]] = None
            n-=1
            a.hand.remove(t0[i])

        return bool(m-n)

class Env:
    def __init__(self):
        self.bito = set()
        self.deck = []
```

```
        self.trump = ''
        self.table = {}

        self.agents = [Agent(i) for i in range(4)]
        self.attack = None
        self.defend = None

    # начало новой игры
    def reset(self):
        self.bito = set()
        self.table = {}

        np.random.shuffle(cards)

        for i in range(4): # раздача
            self.agents[i].reset(set(cards[i*6:(i+1)*6]))

        self.deck = cards[24:]
        self.trump = cards[-1][1]

        a = Pair1(self.agents[:2], self.trump)
        b = Pair2(self.agents[2:], self.trump)

        t = np.argmin([i.min_trump(self.trump) for i in self.agents])
        if t < 2 and t == 1:
            a.swap()
        elif t > 1:
            a, b = b, a
            if t == 3:
                a.swap()

        self.attack = a # атака
        self.defend = b # защита

    # смена ролей при переводе или новом раунде
    def rotate(self, flag=0):
        """
        flag = 1 - отбивающий взял
        """
        self.attack.swap()
        if flag:
            self.defend.swap()
        else:
            self.attack, self.defend = self.defend, self.attack

    # восполнение карт
    def fill_hands(self):
        """
        - первым берет атакующий и его помощник
        - далее берет помощник защитника, т.к. мог быть перевод
        - в конце берет защитник

        если кто-то победит, то возвращаем номер команды (иначе 0)
        """

        i = 0
        while i < len(self.attack.agents):
            ag = self.attack.agents[i]
            t = ag.need_cards()
            if t == 6 and not self.deck:
                self.attack.agents.pop(i)
                i-=1
            elif t > 0:
```

```
            t = min(t, len(self.deck))
            ag.hand |= set(self.deck[:t])
            self.deck = self.deck[t:]
        i+=1

    if not self.attack.agents:
        return self.attack.ind


    i = 0
    self.defend.swap() # для простоты два раза их свапнем
    while i < len(self.defend.agents):
        ag = self.defend.agents[i]
        t = ag.need_cards()
        if t == 6 and not self.deck:
            self.defend.agents.pop(i)
            i-=1
        elif t > 0:
            t = min(t, len(self.deck))
            ag.hand |= set(self.deck[:t])
            self.deck = self.deck[t:]
        i+=1
    self.defend.swap() # обратный свап

    if not self.defend.agents:
        return self.defend.ind

    return 0 # пока нет победителя


def play(self, verbose=0):
    # в первый ход нужно положить карту на стол вручную
    # должна быть проверка, можно ли подкидывать (кол-во карт > 1) и отбил ли карты отбивающийся
    # после раунда сначала берут карты, а потом смена ролей
    if not self.table:
        if verbose:
            print(' Атака:', self.attack)
        card = self.attack.first_hit()
        if verbose:
            print('\n Защита:', self.defend)
            print('\n Ход:', card)
        self.table[card] = None

    # print()
    first_hit = 1
    t = self.defend.beat(self.table)
    if t:
        first_hit = 0
    while t and self.defend.cards_count() > 1:
        f = self.attack.toss_cards(self.table, self.defend.agents[0])
        if not f: # если не подкинули то отбой
            break
        t = self.defend.beat(self.table)

    # не смог отбить на первой итерации
    if not t and first_hit:
        t = self.defend.transfer(self.table, self.attack.agents[-1])
        if t:
            self.rotate()
            return self.play(verbose)

    if verbose:
```

```
            print('Стол:', self.table)
        if not t:
            if verbose:
                print(f'Agent{self.defend.agents[0].ind} берет карты')
            self.defend.pick_up_cards(self.table)
        else:
            if verbose:
                print(f'Agent{self.defend.agents[0].ind} отбился')
            for k,v in self.table.items():
                self.bito.add(k)
                self.bito.add(v)
        if verbose:
            print('bito:', self.bito)
        self.table = {}

        if verbose:
            print('\nАгенты восполняют карты из колоды\n')
        res = self.fill_hands()
        if verbose:
            print(f'итог игры: {'победитель - команда ' + str(res) if res else 'победитель пока не
определен'}')

        if res:
            return res

        self.rotate(not t)
        if verbose:
            print('\nПосле смены ролей:\n')
            print(' Атака:', self.attack)
            print()
            print(' Защита:', self.defend)

        return self.play(verbose)
        # print(self.attack)


env = Env()


d = {1:0, 2:0}
for i in range(1000):
    env.reset()
    d[env.play()] += 1
print(d)

# для отображения логов env.play(verbose=1)
```