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 № 3**

**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**: придумать и описать алгоритм выбора квадрата. Провести моделирование с изменяемыми параметрами (радиус *r*, максимальное расстояние передвижения *l* и количество квадратов *n*).

Построить графики вероятности победы агента в матче (для построения трафика, одна из переменных может быть константой, а две другие проецироваться на оси координат).

**Ремарки**

Пример алгоритма выбора квадрата: агент рассчитывает на две или более итераций свои удары так, чтобы болванчик не успел добраться до точки попадания следующего удара.

Если результаты моделирования не изменяются от параметров, указанных в задаче, можно попробовать изменить соотношения радиусов действия агента и болванчика или их максимальное преодолимое расстояние.

Агент теряет очки (их соответственно получает болванчик) если его мяч улетел в аут или он не смог отбить мяч болванчика. Болванчик теряет очки, если пе отбивает удары агента (в аут, по условию, он попасть не может).

**Решение.**

Для решения задачи были реализованы классы Agent и Bot, содержащие параметры игры (*n, l, r*) и методы *hit* для совершения удара, *take\_ball* для попытки принятия мяча. Стратегия агента заключается в совершении ударов, максимально отдаленных от позиции болванчика. Для этого рассчитывается расстояние от позиции болванчика до 4 углов его части теннисного корта и выбирается наиболее дальний.

Для сбора статистики по выигрышам были выбраны значения *n* от 2 до 10, *l* и *r* от 300 до 500 с шагом 20. Для каждого набора параметров выбирается среднее из 10 игр.

Текст программы на языке Python:

*import* numpy *as* np

*import* matplotlib.pyplot *as* plt

*from* numpy.random *import* rand

width = 823

length = 1190

center = 640

class Agent:

def \_\_init\_\_(*self*, *n*, *l*, *r*):

*# количество квадратов (с каждой стороны)*

*self*.n = *n*

*self*.step\_w = width / *n*

*self*.step\_l = length / *n*

*self*.l = *l*

*self*.r = 2\**r* *## это важно !!!*

*self*.x = width/2

*self*.y = 0

def on\_position(*self*):

*self*.x = width/2

*self*.y = 0

def coords(*self*):

*return* *self*.x, *self*.y

def hit(*self*, *x*, *y*, *f*):

*if* *f*:

*# подаем всегда в левый ближний угол (от себя)*

*return* (rand()\**self*.step\_w + width-*self*.step\_w, rand()\**self*.step\_l + length-*self*.step\_l)

*# отбиваем в дальний от бота квадрат*

*# углы - 0: ЛН, 1: ПН; 2: ПВ; 3: ЛВ*

t = np.argmax([

np.linalg.norm([*x*,*y*]),

np.linalg.norm([width-*x*,*y*]),

np.linalg.norm([width-*x*,length-*y*]),

np.linalg.norm([*x*,length-*y*])])

x\_new = rand() \* *self*.step\_w + (width-*self*.step\_w)\*int(t *in* [1,2])

y\_new = rand() \* *self*.step\_l + (length-*self*.step\_l)\*int(t *in* [2,3])

*# 5% шанс попасть в другой квадрат или аут*

*if* rand() < 0.05:

fl = [-1, 1][int(rand() > 0.5)]

*if* rand() > 0.5:

x\_new += fl \* *self*.step\_w

*else*:

y\_new += fl \* *self*.step\_l

*# print(t)*

*# проверяем на попадание в аут и возвращаем координаты мяча*

*return* ((0 <= x\_new <= width) and (0 <= y\_new <= length), x\_new, y\_new)

def take\_ball(*self*, *x*, *y*):

*# т.к. отбиваем по полуокружности, надо отбежать*

*if* *y* < *self*.y:

*if* *x* > *self*.x:

*x* = max(*x*-*self*.r, *self*.x)

*else*:

*x* = min(*x*+*self*.r, *self*.x)

*if* ((*self*.y - *y*)\*\*2 + (*self*.x - *x*)\*\*2)\*\*0.5 > *self*.l:

*return* False

*else*:

*self*.y = *y*

*self*.x = *x*

*return* True

*else*:

v = np.array([*x*-*self*.x, *y*-*self*.y])

norm = np.linalg.norm(v)

*# print("Norm:", norm)*

*if* norm > *self*.l+*self*.r:

*return* False

*else*:

*# двигаемся к ближайшей точке для отбития*

*if* norm > *self*.r:

v = (norm-*self*.r)\*(v / norm)

*self*.x += v[0]

*self*.y += v[1]

*return* True

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

*return* f"Agent ({*self*.x}, {*self*.y})"

class Bot:

*# его тактика - добегать до мяча "вплотную"*

def \_\_init\_\_(*self*, *l*, *r*):

*self*.l = *l*

*self*.r = *r*

*self*.x = width/2

*self*.y = length-center

def on\_position(*self*):

*self*.x = width/2

*self*.y = length-center

def coords(*self*):

*return* *self*.x, *self*.y

def hit(*self*):

*return* (rand()\*width, rand()\*length)

def take\_ball(*self*, *x*, *y*):

v = np.array([*x*-*self*.x, *y*-*self*.y])

norm = np.linalg.norm(v)

*if* norm > *self*.l+*self*.r:

*return* False

*# двигаемся ближе к мячу для отбития*

*if* norm > *self*.r:

v = *self*.l\*(v / norm)

*self*.x += v[0]

*self*.y += v[1]

*return* True

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

*return* f"Bot ({*self*.x}, {*self*.y})"

def game(*agent*, *bot*):

c1, c2 = 0, 0

f = True

t = None

*while* (c1 < 4 and c2 < 4) or abs(c1-c2) < 2:

*if* f:

*agent*.on\_position()

*bot*.on\_position()

t = *agent*.hit(\**bot*.coords(), f)

*if* not *bot*.take\_ball(\*t):

c1 += 1

f = True

*continue*

t = *bot*.hit()

f = False

*if* not *agent*.take\_ball(\*t):

c2 += 1

f = True

*continue*

t = *agent*.hit(\**bot*.coords(), f)

*if* not t[0]:

c2 += 1

f = True

*continue*

*if* not *bot*.take\_ball(t[1], t[2]):

c1 += 1

f = True

*continue*

t = *bot*.hit()

*return* c1, c2

def play\_match(*agent*, *bot*):

s1, s2 = 0, 0

*for* s *in* range(1, 4):

*# print(f"Set {s}")*

g1, g2 = 0, 0

j = 0

*while* (g1 < 6 and g2 < 6) or abs(g1-g2) < 2:

j += 1

c1, c2 = game(*agent*, *bot*)

*# print(f" Game {j}: {c1} - {c2} ")*

*if* c1 > c2:

g1 += 1

*else*:

g2 += 1

s1 += g1 > g2

s2 += g1 < g2

*return* s1 > s2

d = []

*for* n *in* range(2, 10):

*for* l *in* range(300, 501, 20):

*for* r *in* range(300, 501, 20):

agent = Agent(n,l,r)

bot = Bot(l, r)

s = 0

k = 10

*for* j *in* range(k):

s += play\_match(agent, bot)

d.append((n, l, r, s/k))

*import* pandas *as* pd

res = pd.DataFrame(d, *columns*=['n', 'l', 'r', 'res'])

plt.plot(res.groupby('n')['res'].agg(np.average))

plt.xlabel('n')

plt.ylabel('probability')

plt.show()

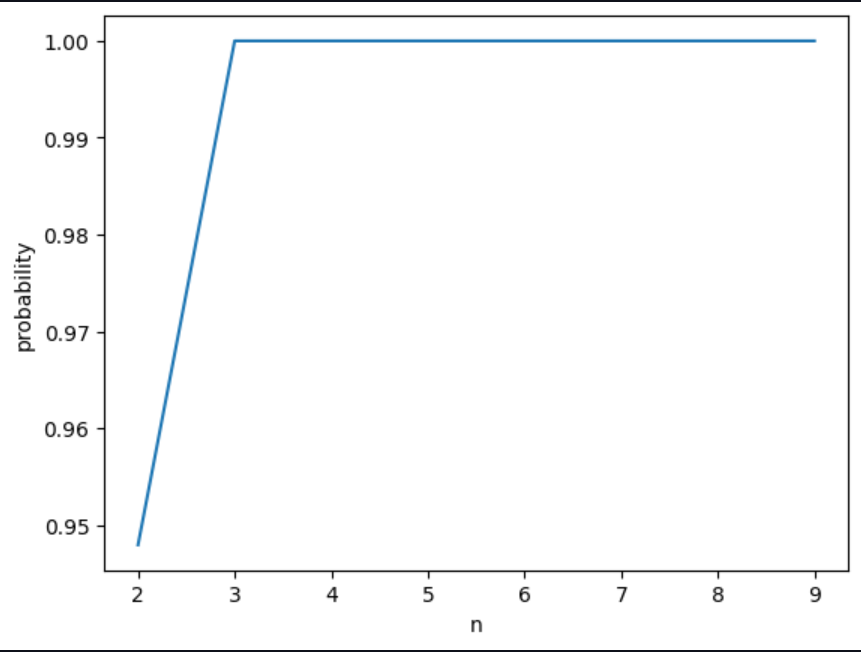


Рисунок 1 - Зависимость вероятности выигрыша от *n*

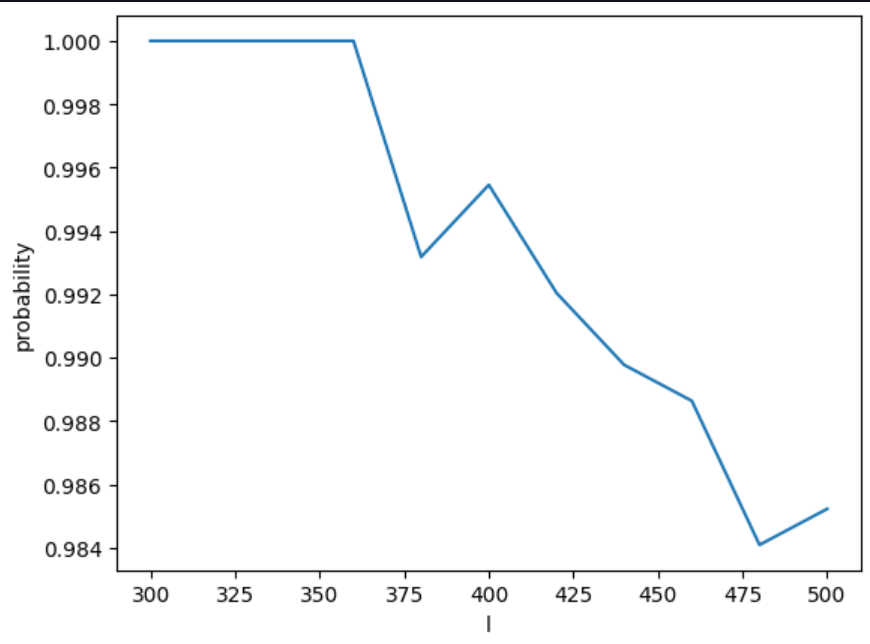


Рисунок 2 - Зависимость вероятности выигрыша от *l*

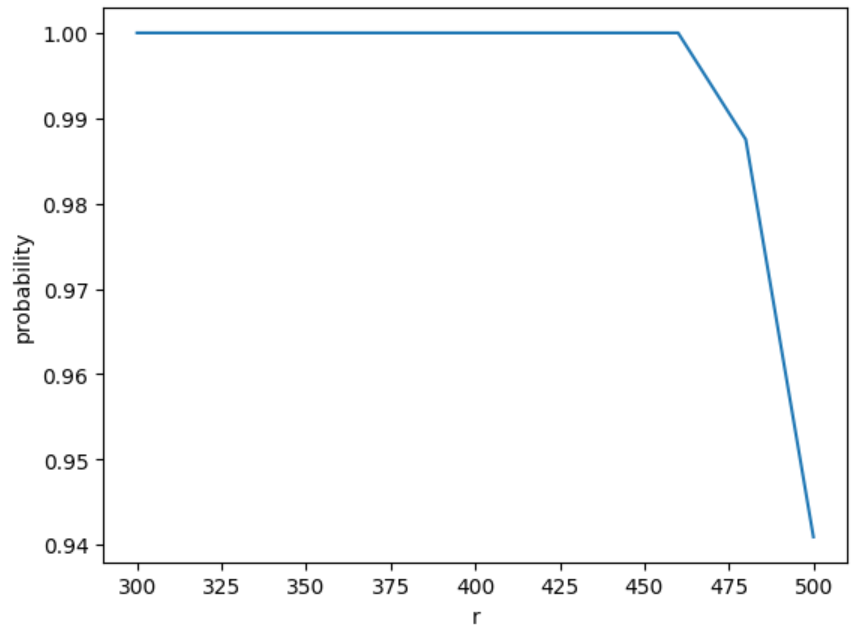


Рисунок 3 - Зависимость вероятности выигрыша от *r*

Вывод: по результатам запусков видно, что, агент побеждает практически в любых случаях, так как его стратегия основана слабости болванчика. Вероятность победить понижается лишь в случаях, когда агент не может ударить вне зоны болванчика (n = 2 или достаточно больших l и r).