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 вычислительных технологий**

**ОТЧЁТ №3**

**Дисциплина: Многоагентное моделирование**

Работу выполнил: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ А. А. Иванов

Направление подготовки: 02.03.03 Математическое обеспечение и администрирование информационных систем

Преподаватель: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_А. А. Миков

Краснодар

2024

**Цель работы**

Написать программу на любом языке программирования, моделирующую игру между агентом (игроком) и болванчиком в теннис.

**Описание задачи**

Контекст

Агент играет 1 на 1 против болванчика. Алгоритм болванчика достаточно прост: при подаче мяча в радиус действия (окружность радиуса r)

он всегда отбивает его в случайную точку половины корта агента. Агент может отбить мяч, который попадает в его радиус действия

(полукруг перед ним радиуса 2r, агент всегда смотрит в сторону противника). При отбивании агент, как и болванчик, могут перемещаться

не более, чем на l от той точки, в которой они находятся. Цель агента - победить болванчика. Для этого он пользуется следующей

стратегией: он делит половину корта болванчика на квадраты. В момент отбития он выбирает один из квадратов, в который он отправит

мяч (алгоритм выбора квадрата реализующий разрабатывает самостоятельно). Ввиду неидеальности исполнения, мяч может с 5% вероятностью

попасть в один из соседних клеток или в аут, если выбранный квадрат находится у края корта (в квадрате мяч попадает в случайную точку).

Задача

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 графика, одна из

переменных может быть константой, а две другие проецироваться на оси координат).

Ремарки

Пример алгоритма выбора квадрата: агент рассчитывает на две или более итерации 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 соотношение радиусов действия агента и болванчика или их максимальное преодолимое расстояние. Агент теряет очки (их получает

болванчик) если его мяч улетел в аут или он не смог отбить подачу болванчика. Болванчик теряет очки, если не отбивает удары агента

(в аут болванчик попасть не может).

Правила большого тениса

Подаёт всегда агент, вне зависимости от того выиграл или проиграл подачу. Подача происходит с задней линии (область А) в области E

или F (при подаче считается, что агент не может промахнуться и всегда попадает в выбранный квадрат). При подаче агент стоит по центру

задней линии, а болванчик на пересечении областей D, E, F. Аутом считается вся область за задней линией и полосы по краям, т.е. все

области, кроме A, B, C, D, E, F.

Счёт в гейме

Счёт ведётся следующим образом: 0-15-30-40-гейм. Если возникает ситуация, при которой у обоих игроков по 40 очков (ровно), то далее

игра происходит до тех пор, пока разница не будет равняться двум выигранным мячам. Игрок, который выиграл одно очко после счёта

"ровно", получает оценку "больше".

Сет

Очки 0-15-30... и т.д. складываются в геймы, геймы в сеты, сеты в матчи. Самый распространённый формат проведения соревнований

заключается в том, что для выигрыша одного сета нужно выиграть 6 геймов, но с разницей в 2 очка. Т.е. если счёт на табло значится

6-5, то проводится ещё один гейм, чтобы получилось 7-5. Но если счёт становится равный 6-6, то играется "тай-брейк". Это особый гейм,

в котором за каждый выигранный мяч присваивается одно очко, счёт ведётся до семи. Опять же вступает правило: разница в счёте должна

равняться минимум двум очкам (7/5, 7/4 и т.д.). Но если счёт становится 7-6, то игра продолжается до тех пор, пока разница не станет

равной двум.

Счёт в сетах

Игрок, который выиграл сет, получает в свой актив одно очко. Большинство соревнований имеет такой формат, что игрокам нужно сыграть

минимум 2 сета и выиграть их. Но на некоторых турнирах (например, Уимблдон) теннисистам нужно провести 3 сета. Если счёт на 3-сетовых

матчах становится равным 1-1, то проводится третий сет для определения победителя. В рамках задания используется второй вариант.

**Описание решения**

Для реализации данной задачи был выбран Python, а также объектно-ориентированный подход.

Было создано 7 файлов: task.txt (описание задачи); main.py (основная логика симуляции игр); Cort.py (класс теннисного корта); Zone.py (класс зоны теннисного корта); Ball.py (класс мяча); Player.py (класс агента/игрока); Dummy.py (класс болванчика).

Cort.py

Логика решения задачи была следующей: нужно реализовать корт, который сам разобьёт себя на зоны (для более простого управления границами) и который будет отвечать за отрисовку состояний (для отладки).  
В данном классе была написаны следующие методы:  
\_\_init\_\_(): конструктор класса, который инициализирует корт с полями matrix (проецирующая матрица) и all\_zones (словарь всех зон);

division\_into\_zones(): в конструктор класса поступало значение n – кол-во квадратов, на которое разбивается корт, и на основе этого значения вычислялась ширина и длина корта для сохранения пропорций зон (не всякое n позволяло разбить корт на квадраты с сохранением пропорций). Вычисленная длина и ширина поступали в эту функцию для определения пространственных координат зон на основе имени зоны. Этот метод вызывал создание объектов-зон из другого класса, который будет рассмотрен ниже. На основе созданных зон формируется словарь, который возвращается в конструктор корта;

start\_position(): метод, который устанавливает игрока и болванчика на исходные позиции в зонах A и D перед первой подачей игрока (после того, как игрок/болванчик получает очко, снова срабатывает этот метод);

print\_cort\_state(): метод, который отрисовывает корт с символами игрока (P), болванчика (D) и мяча (B) на соответствующих клетках по координатам для отладки.

Zone.py

\_\_init\_\_(): в данном классе присутствует только конструктор, который в зависимости от имени зоны и общей длины и ширины корта задаёт зонам минимальную и максимальную координату по X и Y соответственно (это нужно для того, чтобы при обработки действий игрока и болванчика работать с частями корта, у которых уже «вшиты» координаты).

Ball.py

\_\_init\_\_(): конструктор класса, который создаёт объект-мяч с полями X и Y (координаты мяча на корте);

new\_location(): метод, который вызывают игрок и болванчик у мяча при совершении подачи (таким образом мяч «летает» по корту).

Player.py

\_\_init\_\_(): конструктор игрока, который создаёт объект с полями x, y, r, l, score, game, set, match (последние 4 поля не использовались, потому что было решено работать со счётом в главном цикле (решено это было поздно, когда про эти поля уже забыли));

pitch(): метод, реализующий подачу в случайную клетку корта. В нём также реализована ситуация промаха, обрабатывается промах в соседнюю клетку и в аут;

tactic\_pitch(): метод, который реализует подачу в целенаправленно выбранную клетку корта. Тоже реализован промах в соседнюю клетку и в аут.

zone\_selection(): метод, который выбирает зону подачи. Реализован как случайный выбор, так и специальный алгоритм по вычислению самой дальней клетки от болванчика;

move(): метод для перемещения игрока по полю в сторону переданного ему мяча. Отдельно рассматривается случай, когда мяч прилетел игроку за спину (ball.x < player.x), так как тогда игрок не может воспользоваться полукругом-обзором (обзор радиуса 2r распространяется перед игроком). В случае, если игрок может «дотянуться» до мяча, тогда он выполняет подачу в половину корта болванчика.

Dummy.py

\_\_init\_\_(): конструктор, инициализирует объект-болванчик с полями как у игрока;

pitch(): метод, реализующий подачу болванчика (болванчик всегда случайно выбирает зону и клетку подачи);

move(): метод перемещения болванчика. Тут отдельно не рассматриваются ситуации с мячом за спиной, потому что у болванчика обзор – окружность, так что он всегда пытается «дотянуться» до мяча с использованием обзора.

main.py

В данном файле присутствует только одна функция:

game(): эта функция моделирует игру с поочерёдными подачами от игроку болванчику и наоборот, проверяет попадания в аут и следит за общим счётом (до 40 очков с разницей в 2 мяча – чтобы выиграть гейм).

Остальная логика игры реализована в циклах, в которых поочерёдно перебираются различные варианты разбиения корта на квадраты; для каждого разбиения корта запускается цикл по r; для каждого r запускается цикл по l; для каждого l запускается 100 игр, по результатам которых вычисляется вероятность победы игрока. Для всех этих значений строятся графики, которые будут представлены в блоке «Примеры вывода».

**Код программы**

Cort.py

import math as mt

from Zone import \*

from Ball import \*

from Player import \*

from Dummy import \*

from colorama import Fore, Style

class Cort:

    def \_\_init\_\_(*self*, *number\_of\_squares*: int) -> None:

        """

        Метод, который создаёт корт и исходя из заданного числа квадратов строит проекцию корта в виде матрицы

        Args:

            number\_of\_squares (int): Кол-во квадратов, на которое делится корт

        Returns:

            None

        """

        width = int(mt.sqrt(*number\_of\_squares* // 3))                          *# получение ширины корта исходя из кол-ва квадратов*

*self*.matrix = [[0 for i in range(width \* 3)] for j in range(width)]   *# создание проецирующей матрицы в соотношении 3:1 (длина:ширина)*

*# print("Корт в виде матрицы:")*

*# for i in range(width):*

*# print(self.matrix[i])*

*self*.all\_zones = *self*.division\_into\_zones(width \* 3, width)           *# разделение матрицы на игровые зоны*

*# print("Игровые зоны")*

*# for zone in self.all\_zones.values():                                  # перебор всех зон из словаря*

*# print(f"Зона {zone.name}")*

*# print(f"min\_x: {zone.min\_x}")*

*# print(f"max\_x: {zone.max\_x}")*

*# print(f"min\_y: {zone.min\_y}")*

*# print(f"max\_y: {zone.max\_y}")*

    def division\_into\_zones(*self*, *length*: int, *width*: int) -> dict:

        """

        Метод, который создаёт 6 игровых зон

        Args:

            length (int): Длина корта

            width (int): Ширина корта

        Returns:

            dict: Словарь всех игровых зон

        """

        a = Zone(*length*, *width*, "A")   *# поочерёдное создание всех зон*

        b = Zone(*length*, *width*, "B")

        c = Zone(*length*, *width*, "C")

        d = Zone(*length*, *width*, "D")

        e = Zone(*length*, *width*, "E")

        f = Zone(*length*, *width*, "F")

        return {"A": a, "B": b, "C": c, "D": d, "E": e, "F": f}

    def start\_positions(*self*, *a*: Zone, *d*: Zone, *player*: Player, *dummy*: Dummy) -> None:

        """

        Метод, который перемещает игрока и болванчика на стартовые позиции (центр A и пересечение F, E, D)

        Args:

            a (Zone): Зона A, в которой будет располагаться игрок

            d (Zone): Зона D, в которой будет располагаться болванчик (берётся клетка, максимально близкая к границе F, E относительно D)

            player (Player): Игрок

            dummy (Dummy): Болванчик

        Returns:

            None

        """

        player\_x = round((*a*.min\_x + *a*.max\_x) / 2)   *# середина зоны А по X*

        player\_y = round((*a*.min\_y + *a*.max\_y) / 2)   *# середина зоны А по Y*

*player*.x = player\_x

*player*.y = player\_y

        dummy\_x = *d*.min\_x   *# край зоны D по X*

        dummy\_y = round((*d*.min\_y + *d*.max\_y) / 2)   *# середина зоны D по Y*

*dummy*.x = dummy\_x

*dummy*.y = dummy\_y

*# print("Стартовое положение игрока (P) и болванчика (D)")*

*# for i in range(len(self.matrix)):*

*# for j in range(len(self.matrix[i])):*

*# if i == player.y and j == player.x:   # печатаем символ игрока на корте*

*# self.matrix[i][j] = f"{Fore.BLUE}P{Style.RESET\_ALL}"*

*# elif i == dummy.y and j == dummy.x:   # печатаем символ болванчика на корте*

*# self.matrix[i][j] = f"{Fore.BLUE}D{Style.RESET\_ALL}"*

*# else:*

*# self.matrix[i][j] = f"{Fore.GREEN}O{Style.RESET\_ALL}"   # иначе печатаем ноль (нужно чтобы символы игроков, болванчиков и мячей не дублировались)*

*# for row in self.matrix:*

*# print(" ".join(row))*

    def print\_cort\_state(*self*, *player*: Player, *dummy*: Dummy, *ball*: Ball) -> None:

        """

        Метод, который печатает текущее состояние корта

        Args:

            player (Player): Игрок

            dummy (Dummy): Болванчик

            ball (Ball): Мяч

        Returns:

            None

        """

        print("Текущее состояние корта")

        for i in range(len(*self*.matrix)):

            for j in range(len(*self*.matrix[i])):

                if i == *player*.y and j == *player*.x:   *# печатаем символ игрока на корте*

*self*.matrix[i][j] = f"{Fore.BLUE}P{Style.RESET\_ALL}"

                elif i == *dummy*.y and j == *dummy*.x:   *# печатаем символ болванчика на корте*

*self*.matrix[i][j] = f"{Fore.BLUE}D{Style.RESET\_ALL}"

                elif i == *ball*.y and j == *ball*.x:     *# печатаем символ мяча на корте*

*self*.matrix[i][j] = f"{Fore.BLUE}B{Style.RESET\_ALL}"

                else:

*self*.matrix[i][j] = f"{Fore.GREEN}O{Style.RESET\_ALL}"           *# иначе печатаем ноль (нужно чтобы символы игроков, болванчиков и мячей не дублировались)*

        for row in *self*.matrix:

            print(" ".join(row))

Zone.py

class Zone:

    def \_\_init\_\_(*self*, *length*: int, *width*: int, *name*: str) -> None:

        """

        Метод, который задаёт зоне координаты исходя из её названия (у каждой зоны своё положение на корте)

        Args:

            length (int): Длина корта

            width (int): Ширина корта

            name (str): Название зоны

        Returns:

            None

        """

*self*.name = *name*                   *# задаём зоне имя*

        if *name* == "A":                    *# исходя из имени выделяем зоне конкретные координаты (у каждой зоны своё положение)*

*self*.min\_x = 0

*self*.max\_x = (*length* // 4) - 1

*self*.min\_y = 0

*self*.max\_y = *width* - 1

        elif *name* == "B":

*self*.min\_x = *length* // 4

*self*.max\_x = (*length* // 2) - 1

*self*.min\_y = *width* // 2

*self*.max\_y = *width* - 1

        elif *name* == "C":

*self*.min\_x = *length* // 4

*self*.max\_x = (*length* // 2) - 1

*self*.min\_y = 0

*self*.max\_y = (*width* // 2) - 1

        elif *name* == "D":

*self*.min\_x = (*length* // 4) \* 3

*self*.max\_x = *length* - 1

*self*.min\_y = 0

*self*.max\_y = *width* - 1

        elif *name* == "E":

*self*.min\_x = *length* // 2

*self*.max\_x = ((*length* // 4) \* 3) - 1

*self*.min\_y = 0

*self*.max\_y = (*width* // 2) - 1

        elif *name* == "F":

*self*.min\_x = *length* // 2

*self*.max\_x = ((*length* // 4) \* 3) - 1

*self*.min\_y = *width* // 2

*self*.max\_y = *width* - 1

        else:

            print(f"Получено некорректное имя зоны: {*name*}")   *# если получили неверное имя*

            raise ValueError                                   *# то вызываем ошибку значения (передали неверное значение)*

Ball.py

class Ball:

    def \_\_init\_\_(*self*) -> None:

        """

        Метод, который создаёт объект-мяч с нужными полями

        Args:

            None

        Returns:

            None

        """

*self*.x = None   *# при инициализации мяча у него нет координат*

*self*.y = None

    def new\_location(*self*, *new\_x*: int, *new\_y*: int) -> None:

        """

        Метод, который переносит мяч в новое местоположение

        Args:

            new\_x (int): Новое положение по X

            new\_y (int): Новое положение по Y

        Returns:

            None

        """

*self*.x = *new\_x*

*self*.y = *new\_y*

Player.py

from Zone import \*

from Ball import \*

from Dummy import \*

import random as rnd

class Player:

    def \_\_init\_\_(*self*, *r*: int, *l*: int) -> None:

        """

        Метод, который создаёт объект-игрока с нужными полями

        Args:

            r (int): Радиус обзора игрока (полукруг перед ним)

            l (int): Максимальное расстояние перемещения игрока между атаками

        Returns:

            None

        """

*self*.r = 2 \* *r*

*self*.l = *l*

*self*.x = None   *# при инициализации игрока у него нет координат*

*self*.y = None

*self*.score = 0

*self*.game = 0

*self*.set = 0

*self*.match = 0

    def pitch(*self*, *zone*: Zone, *ball*: Ball, *message*: str) -> bool:

        """

        Метод, который выбирает в зоне случайный квадрат и отправляет туда мяч, а также реализует промах

        Args:

            zone (Zone): Зона корта, куда полетит мяч

            ball (Ball): Мяч

            message (str): Сообщение о промахе (если есть)

        Returns:

            flag (bool): Флаг показывает, попал ли игрок из-за промаха в аут

        """

        square\_x = rnd.randint(*zone*.min\_x, *zone*.max\_x)   *# выбираем случайную координату по X в зоне*

        square\_y = rnd.randint(*zone*.min\_y, *zone*.max\_y)   *# выбираем случайную координату по Y в зоне*

        if *message* == "Не промах":                       *# стандартная ситуация (не промах)*

*ball*.new\_location(square\_x, square\_y)        *# и отправляем туда мяч*

*# print(f"Игрок отправил мяч в клетку {ball.x}, {ball.y} (это зона {zone.name})")*

            return False

        else:   *# нестандартная ситуация - промах*

*# "ленивый рандом" (при рандоме всё равно может выбраться точка == изначально выбранной, так что используем while)*

            square\_of\_miss\_x = rnd.randint(square\_x - 1, square\_x + 1)

            square\_of\_miss\_y = rnd.randint(square\_y - 1, square\_y + 1)

            while square\_of\_miss\_x == square\_x and square\_of\_miss\_y == square\_y:

                square\_of\_miss\_x = rnd.randint(square\_x - 1, square\_x + 1)

                square\_of\_miss\_y = rnd.randint(square\_y - 1, square\_y + 1)

*# проверка попадания в аут*

            if (square\_of\_miss\_y < *zone*.min\_y and (*zone*.name == "E" or *zone*.name == "D")) or (square\_of\_miss\_x > *zone*.max\_x and *zone*.name == "D") or (square\_of\_miss\_y > *zone*.max\_y and (*zone*.name == "D" or *zone*.name == "F")):

*# print(f"Игрок хотел отправить мяч в клетку {square\_x}, {square\_y} (это зона {zone.name}), но промахнулся и попал в аут ({square\_of\_miss\_x}, {square\_of\_miss\_y})")*

                return True

*# а это просто попадание в соседнюю клетку*

            else:

*ball*.new\_location(square\_of\_miss\_x, square\_of\_miss\_y)

*# print(f"Игрок хотел отправить мяч в клетку {square\_x}, {square\_y} (это зона {zone.name}), но промахнулся и попал в клетку {ball.x}, {ball.y}")*

                return False

    def tactic\_pitch(*self*, *max\_x*: int, *max\_y*: int, *ball*: Ball, *message*: str, *dict\_of\_zones*: dict) -> bool:

        """

        Метод, который отправляем мяч в заданный квадрат, а также реализует промах

        Args:

            max\_x (int): Координата X для дальнего квадрата

            max\_y (int): Координата Y для дальнего квадрата

            ball (Ball): Мяч

            message (str): Сообщение о промахе (если есть)

            dict\_of\_zones (dict): Словарь всех зон

        Returns:

            flag (bool): Флаг показывает, попал ли игрок из-за промаха в аут

        """

        if *message* == "Не промах":

*ball*.new\_location(*max\_x*, *max\_y*)

*# print(f"Игрок отправил мяч в клетку {ball.x}, {ball.y}")*

            return False

        elif *message* == "Промах":

            square\_of\_miss\_x = rnd.randint(*max\_x* - 1, *max\_x* + 1)

            square\_of\_miss\_y = rnd.randint(*max\_y* - 1, *max\_y* + 1)

            while square\_of\_miss\_x == *max\_x* and square\_of\_miss\_y == *max\_y*:

                square\_of\_miss\_x = rnd.randint(*max\_x* - 1, *max\_x* + 1)

                square\_of\_miss\_y = rnd.randint(*max\_y* - 1, *max\_y* + 1)

*# проверка попадания в аут*

            zone\_d = *dict\_of\_zones*.get("D")

            zone\_e = *dict\_of\_zones*.get("E")

            if (square\_of\_miss\_y < zone\_e.min\_y) or (square\_of\_miss\_x > zone\_d.max\_x) or (square\_of\_miss\_y > zone\_d.max\_y):

*# print(f"Игрок хотел отправить мяч в клетку {max\_x}, {max\_y}, но промахнулся и попал в аут ({square\_of\_miss\_x}, {square\_of\_miss\_y})")*

                return True

*# а это просто попадание в соседнюю клетку*

            else:

*ball*.new\_location(square\_of\_miss\_x, square\_of\_miss\_y)

*# print(f"Игрок хотел отправить мяч в клетку {max\_x}, {max\_y}, но промахнулся и попал в клетку {ball.x}, {ball.y}")*

                return False

    def zone\_selection(*self*, *dict\_of\_zones*: dict, *ball*: Ball, *flag*: str, *tactic*: str, *dummy*: Dummy) -> None:

        """

        Метод, который случайным образом выбирает зону для подачи в раунде

        Args:

            dict\_of\_zones (dict): Словарь всех зон корта

            ball (Ball): Мяч

            flag (str): Тип подачи (первая или обычная)

            tactic (str): Тип стратегии (рандомная или дальний квадрат)

            dummy (Dummy): Болванчик

        Returns:

            None

        """

        if *tactic* == "random":   *# тактики нет, выбираем случайную точку*

            message = "Не промах"

            if *flag* == "first":   *# первая подача направлена только в E/F*

                rand\_zone = rnd.randint(1, 2)

                zone = *dict\_of\_zones*.get("E") if rand\_zone == 1 else *dict\_of\_zones*.get("F")

                return *self*.pitch(zone, *ball*, message)

            elif *flag* == "default":   *# а обычная подача уже в D, E, F*

                rand\_zone = rnd.randint(1, 3)

                if rand\_zone == 1:

                    zone = *dict\_of\_zones*.get("D")

                elif rand\_zone == 2:

                    zone = *dict\_of\_zones*.get("E")

                else:

                    zone = *dict\_of\_zones*.get("F")

*# проверяем, промахнулся ли игрок*

                probability\_of\_miss = rnd.randint(1, 100)

                if 1 <= probability\_of\_miss <= 5:

                    message = "Промах"

                else:

                    message = "Не промах"

                return *self*.pitch(zone, *ball*, message)

            else:

                print(f"Получен некорректный тип подачи: {*flag*}")

                raise ValueError

        elif *tactic* == "far square":   *# тактика есть, выбираем самую дальнюю точку от болванчика*

            message = "Не промах"

            dummy\_x = *dummy*.x

            dummy\_y = *dummy*.y

            max\_x = 0

            max\_y = 0

*# зону F не используем, потому что максимальные значения по X охватывает и E, а по Y только D*

            zone\_d = *dict\_of\_zones*.get("D")

            zone\_e = *dict\_of\_zones*.get("E")

            if *flag* == "first":

*# выбираем дальнюю координату по X с учётом первой подачи (только в E/F)*

                if abs(dummy\_x - zone\_e.min\_x) >= abs(dummy\_x - zone\_e.max\_x):

                    max\_x = zone\_e.min\_x

                else:

                    max\_x = zone\_e.max\_x

*# выбираем дальнюю координату по Y*

                if abs(dummy\_y - zone\_d.min\_y) >= abs(dummy\_y - zone\_d.max\_y):

                    max\_y = zone\_d.min\_y

                else:

                    max\_y = zone\_d.max\_y

                return *self*.tactic\_pitch(max\_x, max\_y, *ball*, message, *dict\_of\_zones*)

            elif *flag* == "default":

*# выбираем дальнюю координату по X с учётом не первой подачи (D, E, F)*

                if abs(dummy\_x - zone\_e.min\_x) >= abs(dummy\_x - zone\_d.max\_x):

                    max\_x = zone\_e.min\_x

                else:

                    max\_x = zone\_d.max\_x

*# выбираем дальнюю координату по Y*

                if abs(dummy\_y - zone\_d.min\_y) >= abs(dummy\_y - zone\_d.max\_y):

                    max\_y = zone\_d.min\_y

                else:

                    max\_y = zone\_d.max\_y

*# проверяем, промахнулся ли игрок*

                probability\_of\_miss = rnd.randint(1, 100)

                if 1 <= probability\_of\_miss <= 5:

                    message = "Промах"

                else:

                    message = "Не промах"

                return *self*.tactic\_pitch(max\_x, max\_y, *ball*, message, *dict\_of\_zones*)

    def move(*self*, *ball*: Ball) -> bool:

        """

        Метод, который реализует перемещение по корту к мячу и определяет, можно ли отбить мяч

        Отдельно рассматриваются случаи когда мяч за спиной у игрока и когда он спереди

        Ситуация с мячом позади: игрок сначала пытается сравняться с мячом по оси X, чтобы быть в одной вертикали

        Это делается для того, чтобы игрок смог воспользоваться своим обзором (полукруг перед собой + одна с ним полоса по оси Y)

        Когда он сравнялся с мячом, то начинает проверять, достаёт ли до мяча с учётом обзора (self.x + self.r)

        Ситуация, когда мяч спереди игрока: игрок сразу пользуется обзором и пробует поравняться с мячом по условию (self.y + self.r)

        или (self.y - self.r) (когда мяч снизу или сверху соответственно), а затем аналогично по оси X

        Args:

            ball (Ball): Мяч

        Returns:

            flag (bool): Флаг, который отвечает, можно отбить мяч или нет

        """

        ball\_x = *ball*.x

        ball\_y = *ball*.y

        number\_of\_movements = *self*.l

        if ball\_x < *self*.x:

            while *self*.x > ball\_x:

                if number\_of\_movements > 0:

*self*.x -= 1

                    number\_of\_movements -= 1

                else:

*# print("Игрок не смог добежать и отбить мяч!")*

                    return False

            while (*self*.y + *self*.r) < *ball*.y or (*self*.y - *self*.r) > *ball*.y:

                if number\_of\_movements > 0:

                    if (*self*.y + *self*.r) < ball\_y:

*self*.y += 1

                    elif (*self*.y - *self*.r) > ball\_y:

*self*.y -= 1

                    number\_of\_movements -= 1

                else:

*# print("Игрок не смог добежать и отбить мяч!")*

                    return False

        elif ball\_x > *self*.x:

            while (*self*.y + *self*.r) < ball\_y or (*self*.y - *self*.r) > ball\_y:

                if number\_of\_movements > 0:

                    if (*self*.y + *self*.r) < ball\_y:

*self*.y += 1

                    elif (*self*.y - *self*.r) > ball\_y:

*self*.y -= 1

                    number\_of\_movements -= 1

                else:

*# print("Игрок не смог добежать и отбить мяч!")*

                    return False

            while (*self*.x + *self*.r) < ball\_x or (*self*.x - *self*.r) > ball\_x:

                if number\_of\_movements > 0:

                    if (*self*.x + *self*.r) < ball\_x:

*self*.x += 1

                    elif (*self*.x - *self*.r) > ball\_x:

*self*.x -= 1

                    number\_of\_movements -= 1

                else:

*# print("Игрок не смог добежать и отбить мяч!")*

                    return False

*# print("Игрок смог добежать и отбить мяч!")*

        return True

Dummy.py

from Zone import \*

from Ball import \*

import random as rnd

class Dummy:

    def \_\_init\_\_(*self*, *r*: int, *l*: int) -> None:

        """

        Метод, который создаёт объект-болванчика с нужными полями

        Args:

            r (int): Радиус обзора болванчика (полукруг перед ним)

            l (int): Максимальное расстояние перемещения болванчика между атаками

        Returns:

            None

        """

*self*.r = *r*

*self*.l = *l*

*self*.x = None   *# при инициализации болванчика у него нет координат*

*self*.y = None

*self*.score = 0

*self*.game = 0

*self*.set = 0

*self*.match = 0

    def pitch(*self*, *dict\_of\_zones*: dict, *ball*: Ball) -> None:

        """

        Метод, который случайным образом выбирает зону для подачи в раунде, клетку в зоне и отправляет туда мяч

        Args:

            dict\_of\_zones (dict): Словарь всех зон корта

            ball (Ball): Мяч

        Returns:

            None

        """

        rand\_zone = rnd.randint(1, 3)

        if rand\_zone == 1:

            zone = *dict\_of\_zones*.get("A")

        elif rand\_zone == 2:

            zone = *dict\_of\_zones*.get("B")

        else:

            zone = *dict\_of\_zones*.get("C")

        square\_x = rnd.randint(zone.min\_x, zone.max\_x)   *# выбираем случайную координату по X в зоне*

        square\_y = rnd.randint(zone.min\_y, zone.max\_y)   *# выбираем случайную координату по Y в зоне*

*ball*.new\_location(square\_x, square\_y)            *# и отправляем туда мяч*

*# print(f"Болванчик отправил мяч в клетку {ball.x}, {ball.y} (это зона {zone.name})")*

    def move(*self*, *ball*: Ball) -> bool:

        """

        Метод, который реализует перемещение по корту к мячу и определяет, можно ли отбить мяч

        У болванчика тактика простая: сначала сравняться к мячом по Y, а затем по X

        Тут отдельно не рассматриваются случаи когда мяч за спиной или спереди, т.к. у болванчика зона видимости - окружность

        Смысл алгоритма очень прост (рассмотрим на примере оси Y, т.е. вертикаль)

        Если мяч выше болванчика (ball.y < dummy.y), то болванчик проверяет, может ли он "дотянуться" до этой клетки

        с учётом обзора (уменьшает свою координату на self.r); если его координата всё равно больше, то он перемещается на 1 вверх

        (self.y -= 1) и проверяет опять.

        Если же ball.y > (dummy.y + self.r), то болванчик перемещается на 1 вниз (self.y += 1) и проверяет опять.

        С осью X ситуация аналогичная

        Args:

            ball (Ball): Мяч

        Returns:

            flag (bool): Флаг, который отвечает, можно отбить мяч или нет

        """

        ball\_x = *ball*.x

        ball\_y = *ball*.y

        number\_of\_movements = *self*.l

        while (*self*.y + *self*.r) < ball\_y or (*self*.y - *self*.r) > ball\_y:

            if number\_of\_movements > 0:

                if (*self*.y + *self*.r) < ball\_y:

*self*.y += 1

                elif (*self*.y - *self*.r) > ball\_y:

*self*.y -= 1

                number\_of\_movements -= 1

            else:

*# print("Болванчик не смог добежать и отбить мяч!")*

                return False

        while (*self*.x + *self*.r) < ball\_x or (*self*.x - *self*.r) > ball\_x:

            if number\_of\_movements > 0:

                if (*self*.x + *self*.r) < ball\_x:

*self*.x += 1

                elif (*self*.x - *self*.r) > ball\_x:

*self*.x -= 1

                number\_of\_movements -= 1

            else:

*# print("Болванчик не смог добежать и отбить мяч!")*

                return False

*# print("Болванчик смог добежать и отбить мяч!")*

        return True

main.py

from Cort import \*

from Ball import \*

from Player import \*

from Dummy import \*

import time

import matplotlib.pyplot as plt

from mpl\_toolkits.mplot3d import Axes3D

import numpy as np

def game(*n*: int, *r*: int, *l*: int, *player\_pitch*: list, *player\_tactic*: str) -> bool:

    """

    Метод, который моделирует игру в теннис между игроком и болванчиком

    Args:

        n (int): Кол-во квадратов, на которое разбивается корт

        r (int): Радиус обзора игрока и болванчика

        l (int): Максимальное расстояние перемещения игрока и болванчика

        player\_pitch (list): Список возможных подач игрока (первая и обычная)

        player\_tactic (str): Тактика, по которой играет игрок (случайная и дальний квадрат)

    Returns:

        flag (bool): Флаг показывает, выиграл ли игрок в текущей партии

    """

    player\_score = 0   *# очки игрока*

    dummy\_score = 0    *# очки болванчика*

    cort = Cort(*n*)

    all\_zones = cort.all\_zones

    player = Player(*r*, *l*)

    dummy = Dummy(*r*, *l*)

    ball = Ball()

    cort.start\_positions(all\_zones.get("A"), all\_zones.get("D"), player, dummy)

    while True:

*# игрок всегда подает первым после установки на исходные позиции*

        player.zone\_selection(all\_zones, ball, *player\_pitch*[0], *player\_tactic*, dummy)  *# первая подача*

*# cort.print\_cort\_state(player, dummy, ball)*

        while True:   *# цикл для текущего розыгрыша мяча до тех пор, пока кто-то не проиграет*

*# болванчик пытается отбить мяч*

            flag\_dummy = dummy.move(ball)

            if not flag\_dummy:   *# если болванчик не отбил мяч*

                player\_score += 1

*# print(f"Очко игрока! Счёт: Игрок {player\_score} - {dummy\_score} Болванчик")*

                cort.start\_positions(all\_zones.get("A"), all\_zones.get("D"), player, dummy)

                break   *# возвращаемся к подаче игрока*

*# если болванчик отбил, он подает обратно игроку*

            dummy.pitch(all\_zones, ball)

*# cort.print\_cort\_state(player, dummy, ball)*

*# теперь игрок должен отбить мяч*

            flag\_player = player.move(ball)

            if not flag\_player:   *# если игрок не отбил мяч*

                dummy\_score += 1

*# print(f"Очко болванчика! Счёт: Игрок {player\_score} - {dummy\_score} Болванчик")*

                cort.start\_positions(all\_zones.get("A"), all\_zones.get("D"), player, dummy)

                break   *# возвращаемся к подаче игрока*

*# если оба отбили, игрок подает свою обычную подачу*

            miss = player.zone\_selection(all\_zones, ball, *player\_pitch*[1], *player\_tactic*, dummy)   *# последующий подачи игрока имеют шанс промаха*

            if miss == True:

                dummy\_score += 1

*# print(f"Игрок попал в аут! Счёт: Игрок {player\_score} - {dummy\_score} Болванчик")*

                cort.start\_positions(all\_zones.get("A"), all\_zones.get("D"), player, dummy)

                break

            else:

                pass

*# cort.print\_cort\_state(player, dummy, ball)*

*#time.sleep(1)*

*# проверка условий победы*

        if (player\_score >= 10 or dummy\_score >= 10) and abs(player\_score - dummy\_score) >= 2:

            break   *# если один из игроков набрал 40 и разница не менее 2 очков*

*#time.sleep(1)*

*# проверка, кто победил*

    if player\_score > dummy\_score:

*# print("Игрок победил!")*

        return True

    else:

*# print("Болванчик победил!")*

        return False

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

    n\_values = [48, 192, 432, 768]              *# несколько возможных вариантов разбиения корта на квадраты*

    r\_values = range(1, 6)                      *# радиус обзора от 1 до 5*

    l\_values = range(1, 11)                     *# максимальное расстояние перемещения от 1 до 10*

    player\_pitch = ["first", "default"]

    player\_tactics = ["random", "far square"]

*# проводим симуляции для каждого n*

    results = {}  *# результаты для каждого n будут сохраняться здесь*

    for n in n\_values:

*# создаём массив для хранения результатов*

        player\_wins = np.zeros((len(r\_values), len(l\_values)))

        for r\_idx, r in enumerate(r\_values):

            for l\_idx, l in enumerate(l\_values):

                player\_total\_win = 0

                dummy\_total\_win = 0

*# запускаем 100 игр*

                for \_ in range(100):

                    player\_game\_win = 0

                    dummy\_game\_win = 0

                    player\_set\_win = 0

                    dummy\_set\_win = 0

*# пока кто-то не выиграет 2 сета*

                    while player\_set\_win < 2 and dummy\_set\_win < 2:

                        player\_game\_win = 0

                        dummy\_game\_win = 0

*# пока кто-то не выиграет гейм с разницей в 2*

                        while True:

                            if game(n, r, l, player\_pitch, player\_tactics[0]):

                                player\_game\_win += 1

                            else:

                                dummy\_game\_win += 1

                            if (player\_game\_win >= 6 or dummy\_game\_win >= 6) and abs(player\_game\_win - dummy\_game\_win) >= 2:

                                break

*# определение победителя сета*

                        if player\_game\_win > dummy\_game\_win:

                            player\_set\_win += 1

                        else:

                            dummy\_set\_win += 1

*# подсчёт победителя в соревновании*

                    if player\_set\_win > dummy\_set\_win:

                        print("Игрок выиграл в соревнованиях!")

                        player\_total\_win += 1

                    else:

                        print("Болванчик выиграл в соревнованиях!")

                        dummy\_total\_win += 1

*# записываем результат (количество побед игрока) для текущего r и l*

                player\_wins[r\_idx, l\_idx] = player\_total\_win

*# сохраняем результаты для текущего n*

        results[n] = player\_wins

*# строим 3D график для текущего n*

        fig = plt.figure()

        ax = fig.add\_subplot(111, *projection*='3d')

*# создаём сетку для r и l*

        r\_grid, l\_grid = np.meshgrid(r\_values, l\_values)

*# транспонируем player\_wins для корректного отображения на графике*

        ax.plot\_surface(r\_grid, l\_grid, player\_wins.T, *cmap*='viridis')

        ax.set\_xlabel('r (радиус обзора)')

        ax.set\_ylabel('l (максимальное расстояние перемещения)')

        ax.set\_zlabel('Победы игрока')

        ax.set\_title(f'Результаты для n = {n}')

        plt.show()

    print(f"Завершено построение графиков для всех значений n.")

**Примеры вывода**

Ниже представлены графики вероятности победы игрока при использовании алгоритма, реализующего выбор самой дальней клетки относительно болванчика (n – кол-во квадратов, на которые разбивается корт с сохранением заданных пропорций зон).

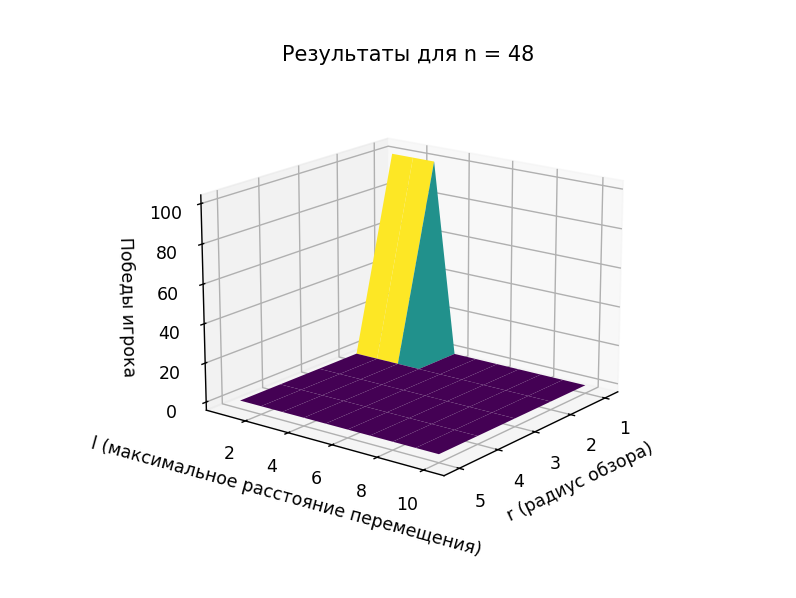


Рисунок 1 – график для n=48 (тактика)

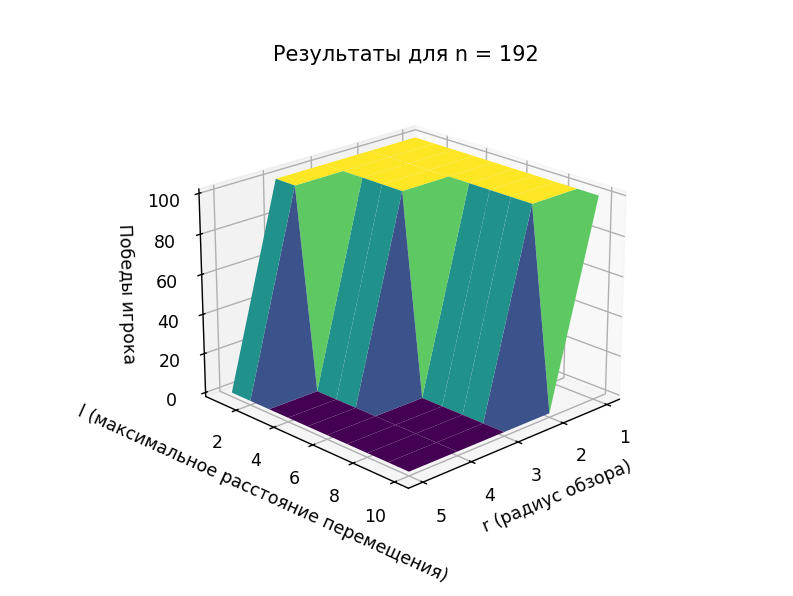


Рисунок 2 – график для n=192 (тактика)

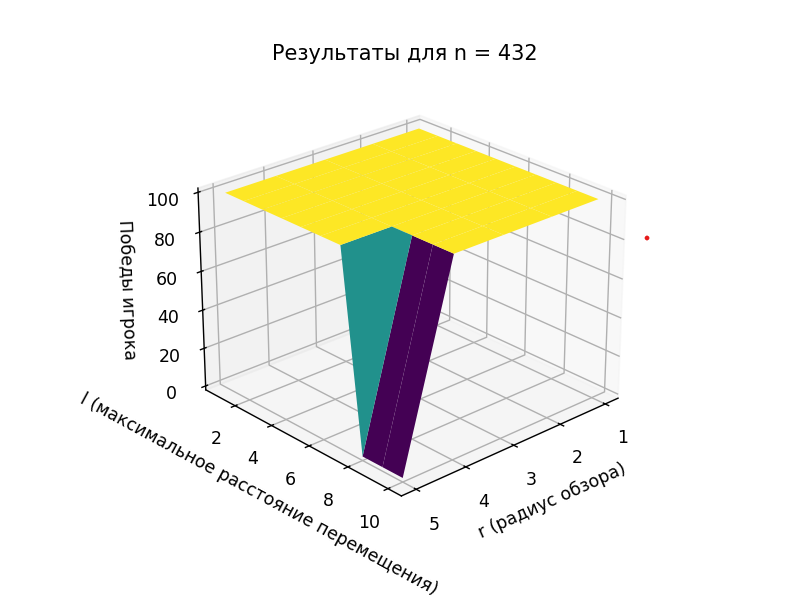


Рисунок 3 – график для n=432 (тактика)

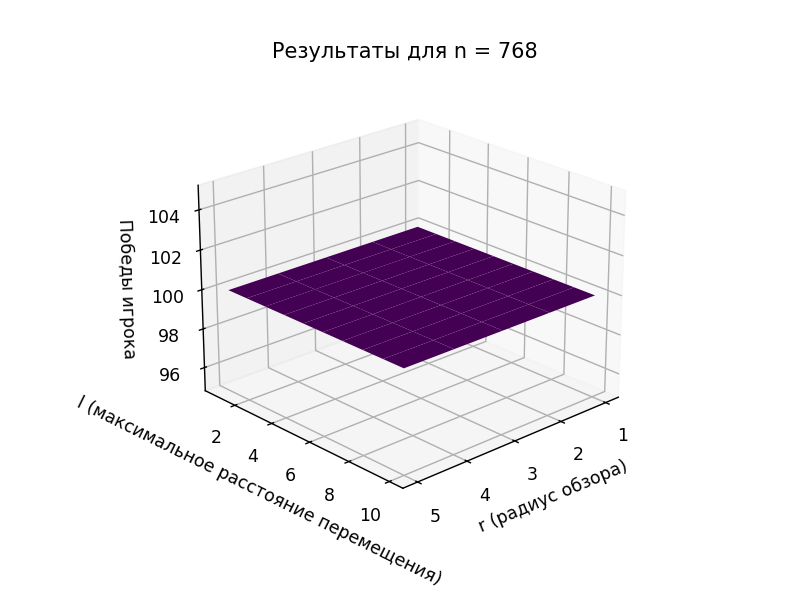


Рисунок 4 – график для n=768 (тактика)

Исходя из вышепоказанных графиков можно сделать вывод, что игрок чаще побеждает при большом разбиении корта, так как тогда болванчик просто не может добежать до угловых зон (эти зоны выбираются как наиболее дальние) и отбить подачу, а при малом разбиении наоборот, болванчик практически всегда побеждает, потому что всегда отбивает подачу (в результате игрок проигрывает, потому что при подаче в угол рискует промахнуться и попасть в аут).

Ниже представлены графики вероятности победы игрока при использовании случайного алгоритма выбора квадрата для подачи.

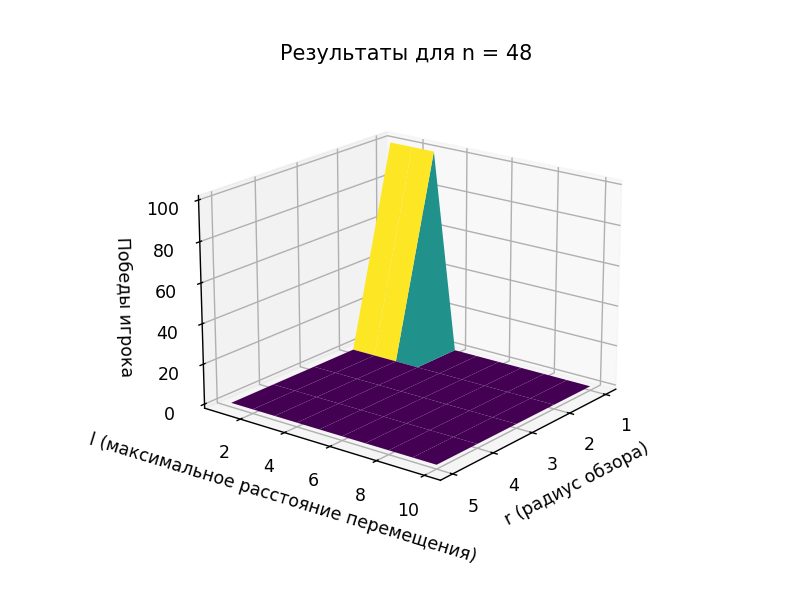


Рисунок 5 – график для n=48 (случайность)

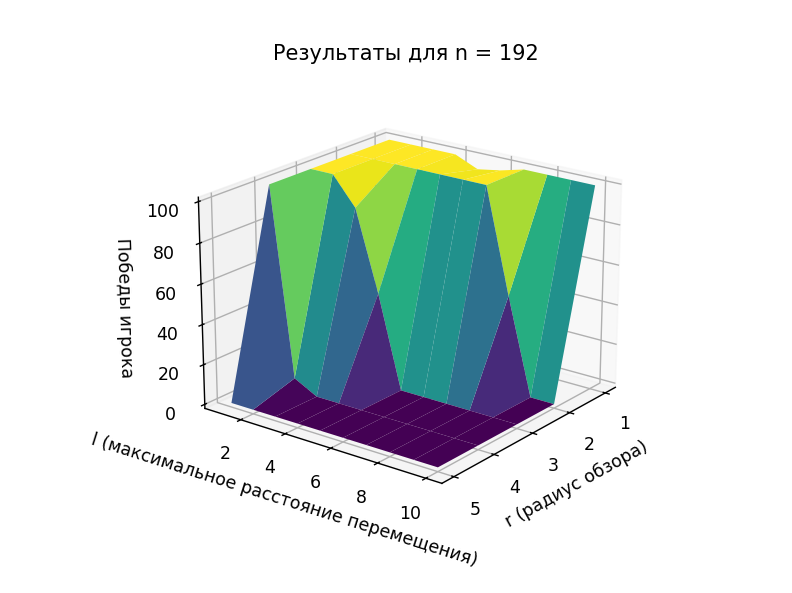


Рисунок 6 – график для n=192 (случайность)

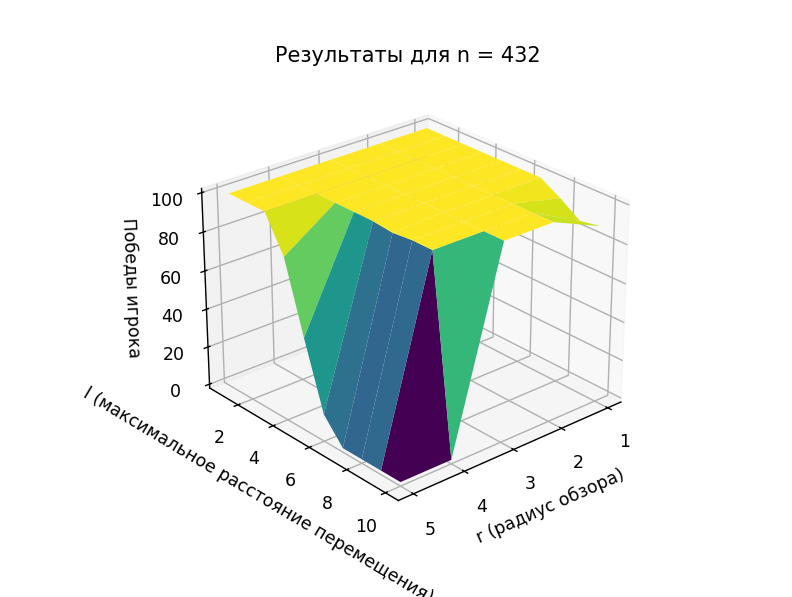


Рисунок 7 – график для n=432 (случайность)

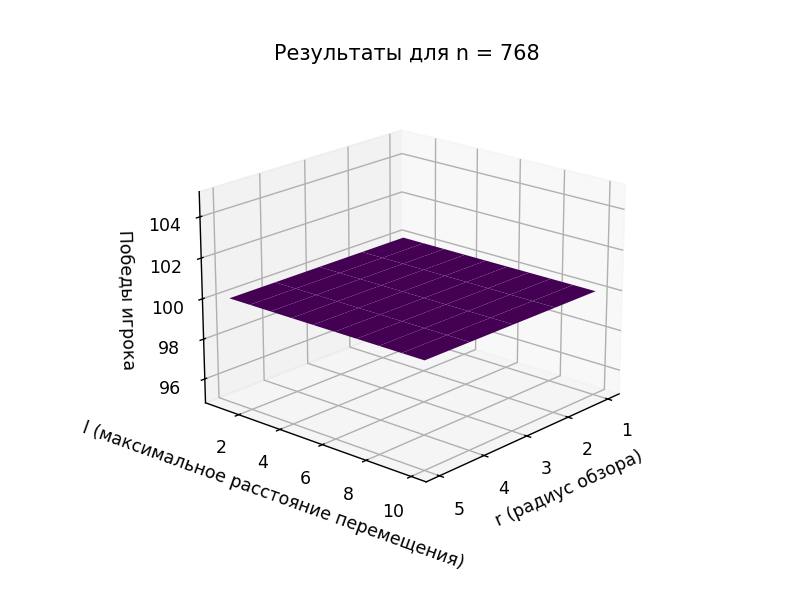


Рисунок 58 – график для n=768 (случайность)

Исходя из вышепоказанных графиков можно сделать вывод, что даже с использованием случайного алгоритма игрок чаще побеждает болванчика на больших разбиениях корта из-за того, что может попасть в ту зону, до которой болванчик не успеет добежать.

Исходя из графиков 2, 3 и 6, 7 можно сделать вывод, что на средних разбиениях алгоритм выбора дальней клетки показывает более стабильные шансы на выигрыш.

Однако, исходя из графиков 1, 4 и 5, 8 можно сказать, что на очень маленьком разбиении практически всегда выигрывает болванчик вне зависимости от стратегии подачи из-за того, что всегда может отбить мяч, а игрок может промазать в аут. Но на очень большом разбиении, наоборот, болванчик всегда проигрывает независимо от алгоритма, потому что игрок, из-за того что всегда подаёт первым, имеет преимущество в выборе зоны.