МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение
высшего образования
«Южно-Уральский государственный университет
(национальный исследовательский университет)»
Высшая школа электроники и компьютерных наук
Кафедра системного программирования

ДОПУСТИТЬ К ЗАЩИТЕ

Заведующий кафедрой, д.ф.-м.н.,
профессор

_____ Л.Б. Соколинский

«___»_____ 2024 г.

Разработка компьютерной игры в жанре «Logic» для платформы Windows

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА
ЮУрГУ – 09.03.04.2024.308-567.ВКР

Научный руководитель,
профессор кафедры СП, д.ф.-м.н.,
доцент
_____ Т.А. Макаровских

Автор работы,
студент группы КЭ-403
_____ М.Б. Рысс

Ученый секретарь
(нормоконтролер)
_____ И.Д. Володченко
«___»_____ 2024 г.

Челябинск, 2024 г.

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение
высшего образования
«Южно-Уральский государственный университет
(национальный исследовательский университет)»
Высшая школа электроники и компьютерных наук
Кафедра системного программирования

УТВЕРЖДАЮ
Зав. кафедрой СП
_____ Л.Б. Соколинский
29.01.2024 г.

ЗАДАНИЕ
на выполнение выпускной квалификационной работы бакалавра
студенту группы КЭ-403
Рыссу Марку Борисовичу,
обучающемуся по направлению
09.03.04 «Программная инженерия»

1. **Тема работы** (утверждена приказом ректора от 22.04.2024 г. № 764-13/12)

Разработка компьютерной игры в жанре «Logic» для платформы Windows.

2. **Срок сдачи студентом законченной работы:** 03.06.2024 г.

3. **Исходные данные к работе**

3.1. Свейгарт Эл. Учим Phyton, делая крутые игры / Эл Свейгарт; [пер. с англ. М.А. Райтмана]. – Москва: Эскимо, 2018. – 416 с.

3.2. Свейгарт Эл. Большая книга проектов Python. – СПб.: Питер, 2022. – 432 с.

3.3. Лутц М. Изучаем Python, том 1, 5 – е изд.: Пер. с англ. – СПб.: ООО «Диалектика», 2019. – 832 с.

3.4. Бизли Д., Джонс Б.К. Python. Книга рецептов. / пер. с анг. Б. В. Уварова. – М.: ДМК Пресс, 2019. – 648 с.

4. **Перечень подлежащих разработке вопросов**

4.1. Анализ актуальных игровых решений.

4.2. Формулировка требований к проектируемой игре.

4.3. Функционал разрабатываемой игры.

4.4. Реализация игры.

4.5. Тестирование.

5. Дата выдачи задания: 29.01.2024 г.

Научный руководитель,
профессор кафедры СП, д.ф.-м.н., доцент Т.А. Макаровских

Задание принял к исполнению М.Б. Рысс

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Актуальность

В последние годы наблюдается значительный рост интереса к компьютерным играм, что подтверждается данными исследования, проведенного компанией «Newzoo» [1]. На рисунке 1 отчетливо видно увеличение числа геймеров по всему миру. Это явление связано с несколькими ключевыми факторами.

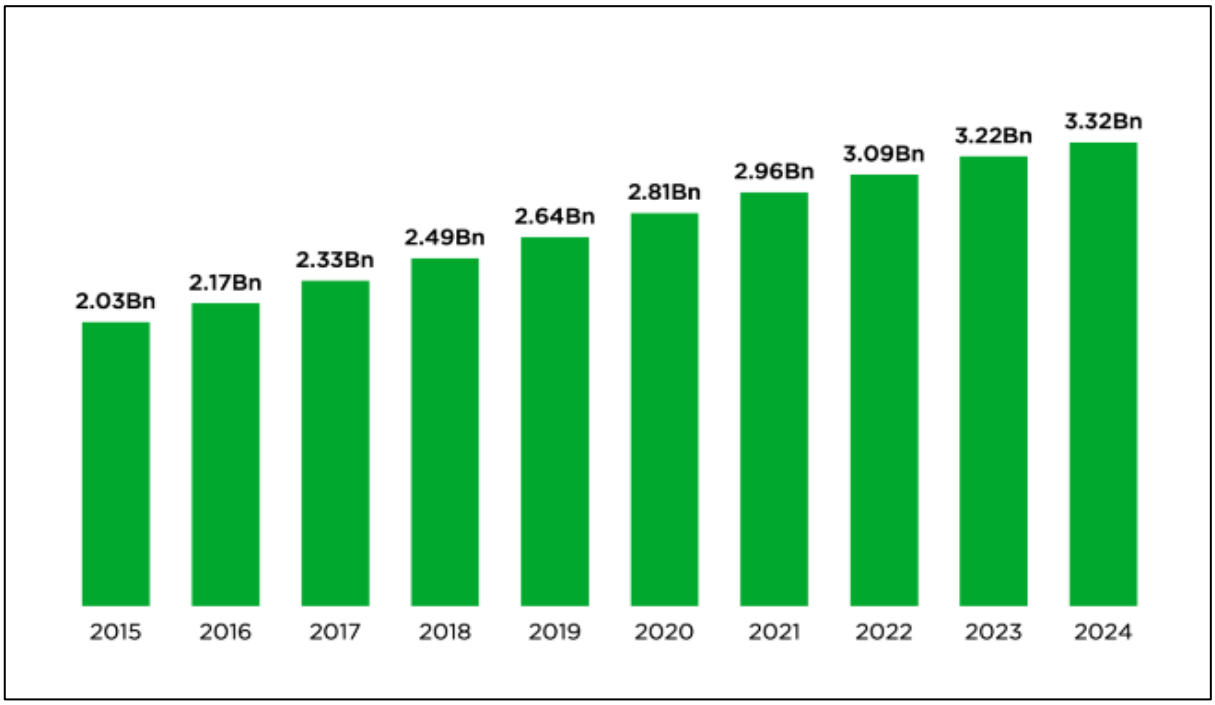

Рисунок 1 – Количество геймеров по годам

Во-первых, развитие технологий и доступность высокоскоростного интернета привели к тому, что все больше людей имеют возможность играть в компьютерные игры. Это особенно актуально для развивающихся стран, где интернет-технологии развиваются быстрыми темпами.

Во-вторых, с каждым годом увеличивается количество и качество игрового контента. Разработчики игр предлагают все более сложные и захватывающие сюжеты, улучшают графику и звуковое сопровождение. Это привлекает новых игроков и удерживает старых.

В-третьих, компьютерные игры стали популярным способом проведения досуга. Они позволяют людям расслабиться, отвлечься от повседневных

забот и получить удовольствие. Это особенно актуально в условиях современного быстрого темпа жизни.

В-четвертых, компьютерные игры стали популярным видом спорта. По всему миру проводятся киберспортивные соревнования, которые привлекают внимание миллионов зрителей. Это стимулирует развитие индустрии и увеличивает количество игроков.

Таким образом, рост интереса к компьютерным играм обусловлен несколькими факторами, включая развитие технологий, улучшение качества игрового контента, изменение образа жизни людей и развитие киберспорта.

Этот феномен обусловлен тем, что большое количество людей выбирают в качестве досуга игры, что предоставлено на рисунке 2.

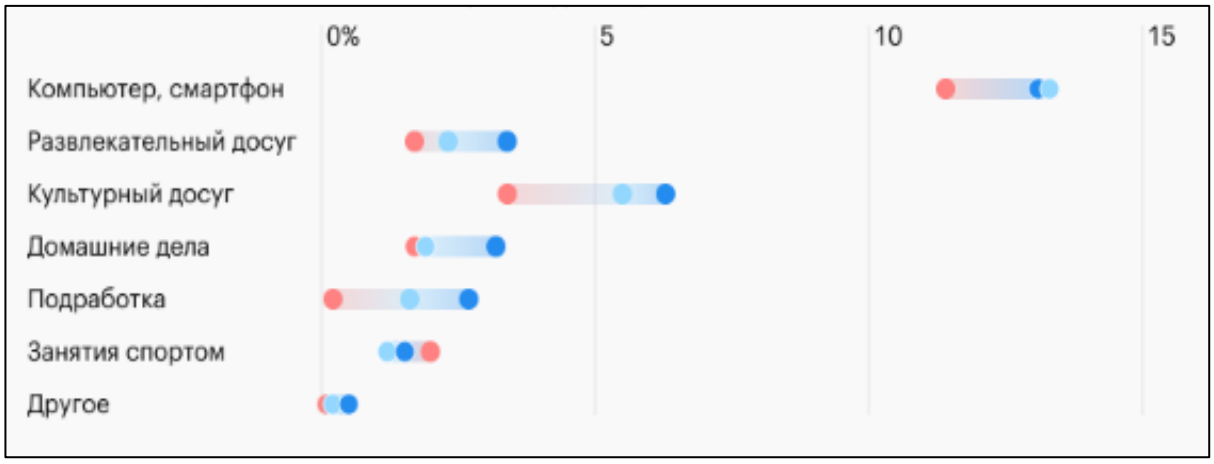

Рисунок 2 – Время провождения досуга граждан России

Данные, показывают, что объем рынка игровой индустрии постоянно увеличивается. Это подтверждает общую тенденцию роста интереса к компьютерным играм и увеличения числа геймеров по всему миру.

Такой рост обусловлен несколькими факторами. Во-первых, развитие технологий и доступность высокоскоростного интернета позволяют все большему количеству людей играть в компьютерные игры. Во-вторых, качество игрового контента постоянно улучшается, разработчики предлагают все более сложные и захватывающие сюжеты, улучшают графику и звуковое сопровождение. В-третьих, компьютерные игры становятся популяр-

ным способом проведения досуга, позволяющим людям расслабиться и получить удовольствие. В-четвертых, развитие киберспорта стимулирует рост индустрии и увеличивает количество игроков.

Все эти факторы в совокупности приводят к постоянному увеличению объема рынка игровой индустрии, что наглядно демонстрируется на рисунке 3.

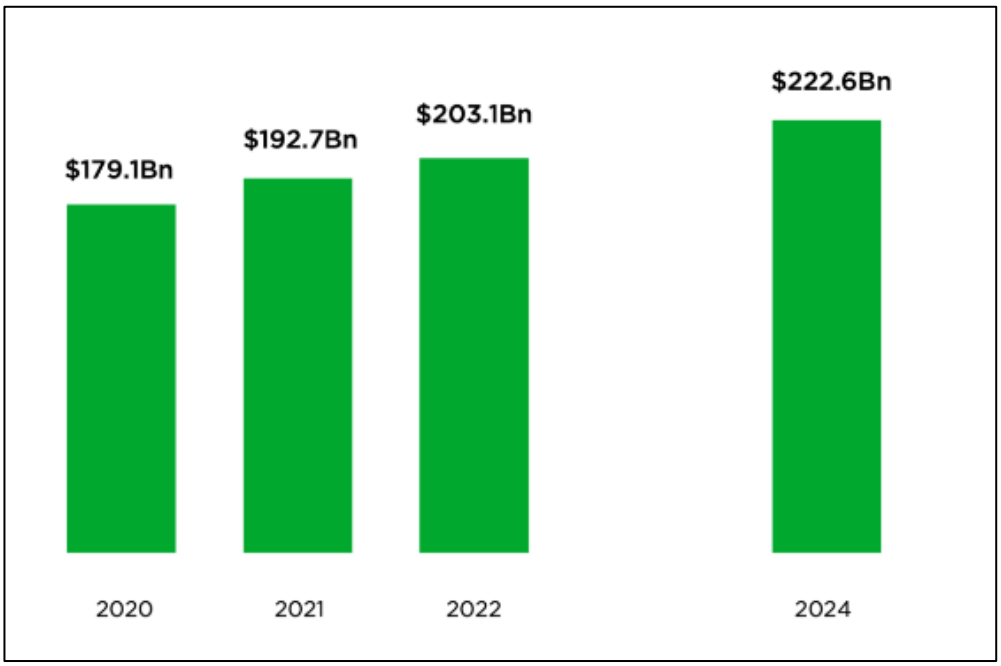

Рисунок 3 – Капитализация рынка игровой индустрии по годам

Постановка задачи

Целью выпускной квалификационной работы является разработка компьютерной игры в жанре «Logic» для платформы Windows. для достижения поставленной цели необходимо выполнить следующие задачи:

1) произвести анализ игровых решений;
2) сформулировать требования к проектируемой игре;
3) разработать функционал игры;
4) реализовать игру;
5) произвести тестирование игрового приложения.

Структура и содержание работы

Работа состоит из введения, пяти глав, заключения и списка литературы. Объем работы составляет 38 страниц, объем списка литературы – 17 источников.

В первой главе проводится детальный анализ игровых решений в жанре «Logic» на платформе Windows. Описывается текущее состояние рынка компьютерных игр, объемы продаж и тенденции развития данного сегмента. Приводятся конкретные примеры наиболее популярных игр в данном жанре, а также дается краткое описание каждой из них. Помимо этого, в главе перечисляются игры, относящиеся к другим подвидам жанра «Логика», таким как головоломки, стратегии, пазлы и т.д.

Во второй главе рассматривается процесс разработки требований к создаваемой игре. Здесь подробно описываются функциональные и нефункциональные требования, которые необходимо учесть при проектировании игры. Кроме того, обсуждаются ключевые аспекты, которые следует учитывать при определении требований к игре, жанр, игровой процесс и технические характеристики. Также описана концепция игры.

В третьей главе содержатся примеры диаграмм вариантов использования в разных игровых меню, таких как выбор размера игрового поля и выбора уровня сложности.

В четвертой главе описывается метод реализации игрового интерфейса, метод проверки, метод отображения цифр, метод создание сложности и метод создание игрового поля игры.

В пятой главе описаны результаты прохождения юзабилити тестирования и функционального тестирования.

В приложении продемонстрирован листинг класса «Sudoku» и модуль расположения цифр на игровом поле.

1. АНАЛИЗ АКТУАЛЬНЫХ ИГРОВЫХ РЕШЕНИЙ

1.1. Объем рынка компьютерных игр на платформе Windows

На рисунке 4 изображено деление рынка игровой индустрии на платформы компании «Newzoo» [1], на гейминг персональных компьютеров приходится 38,2 миллиардов долларов. На рисунке 5, видно, что более 75% всех персональных компьютеров используют операционную систему Windows.

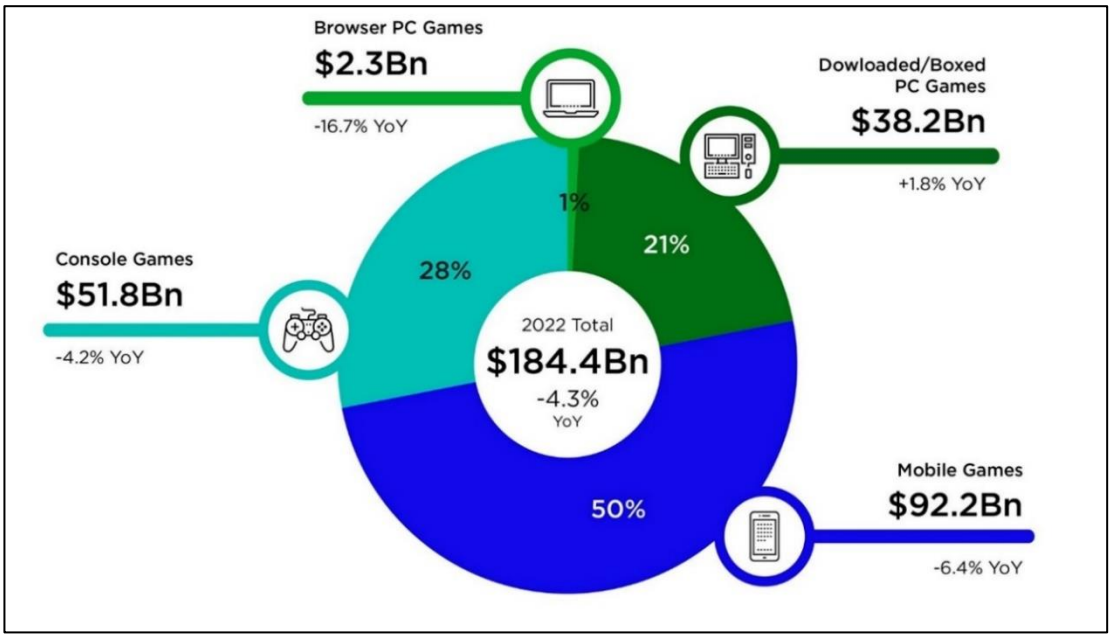

Рисунок 4 – Объем рынка видео игр

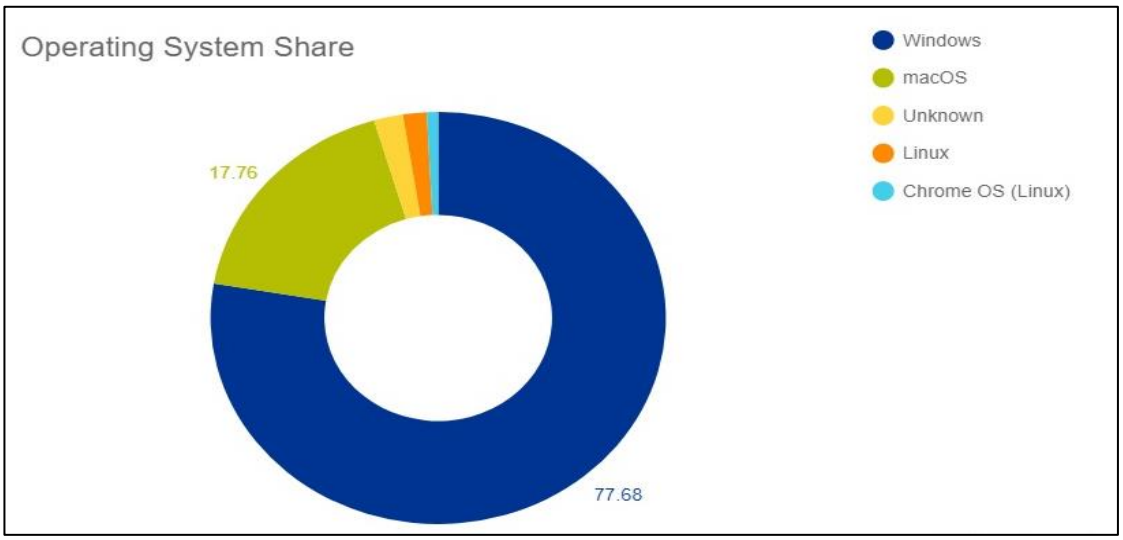

Рисунок 5 – Доля рынка операционных систем

На рисунке 6 изображен график в процентном выражении от всего рынка компьютерных игр на платформе Windows, он дает понять, что самый популярный жанр игр это «Puzzle», с результатом в 21%, который является поджанром «Logic».

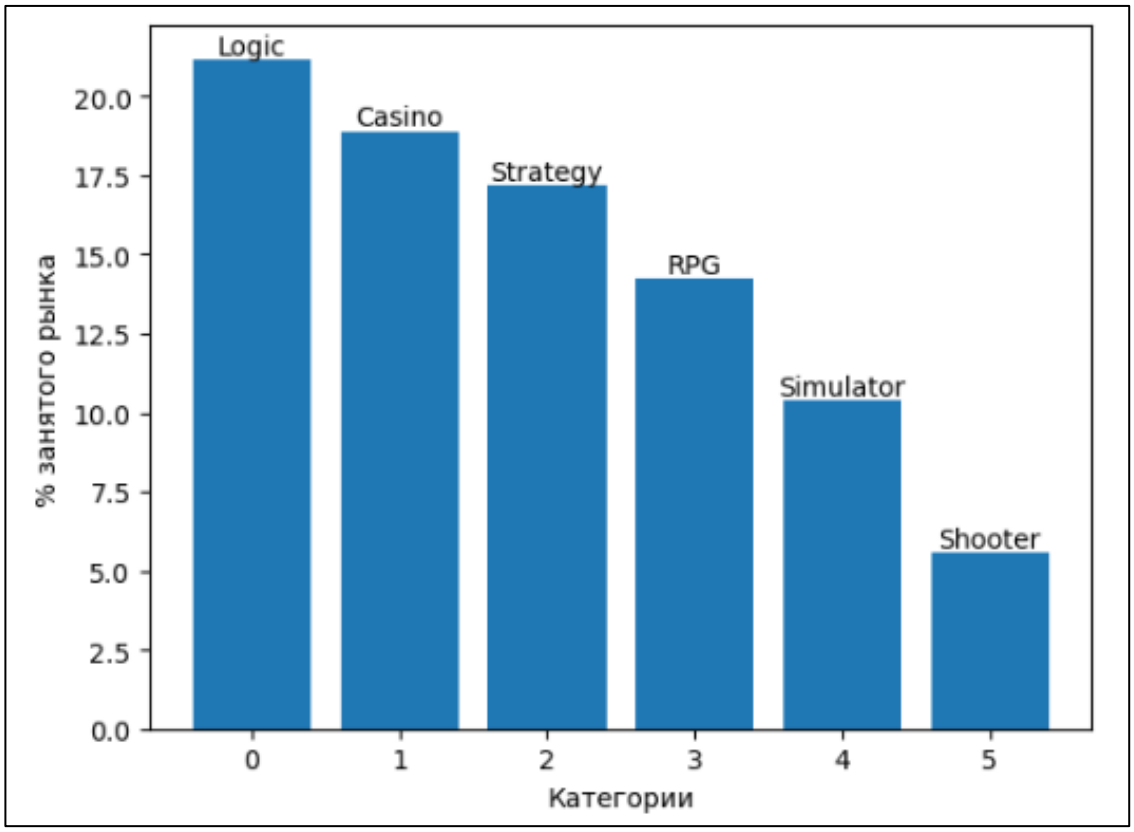

Рисунок 6 – Распределение доли рынка компьютерных игр

1.2. Сравнительный анализ аналогов

Жанр «Logic» представляет собой обширное пространство для творчества разработчиков игр. Он включает в себя множество поджанров и разновидностей игр. Одним из самых известных представителей этого жанра является «Тетрис» [2]. Эта игра, созданная в 1984 году Алексеем Пажитновым, стала настоящим хитом и до сих пор пользуется огромной популярностью среди геймеров всего мира. На рисунке 7 представлен прототип этой игры.

Помимо «Тетриса», в жанре «Logic» существует огромный подраздел «Настольные игры». Этот подраздел включает в себя огромное количество игр с более чем тысячелетней историей. Стоит отметить, что речь идет о

физических настольных играх, таких как го и шахматы. Первые электронные версии этих игр появились в 1985 и 1967 годах соответственно.

Таким образом, жанр «Logic» предлагает широкий спектр возможностей для любителей интеллектуальных развлечений.

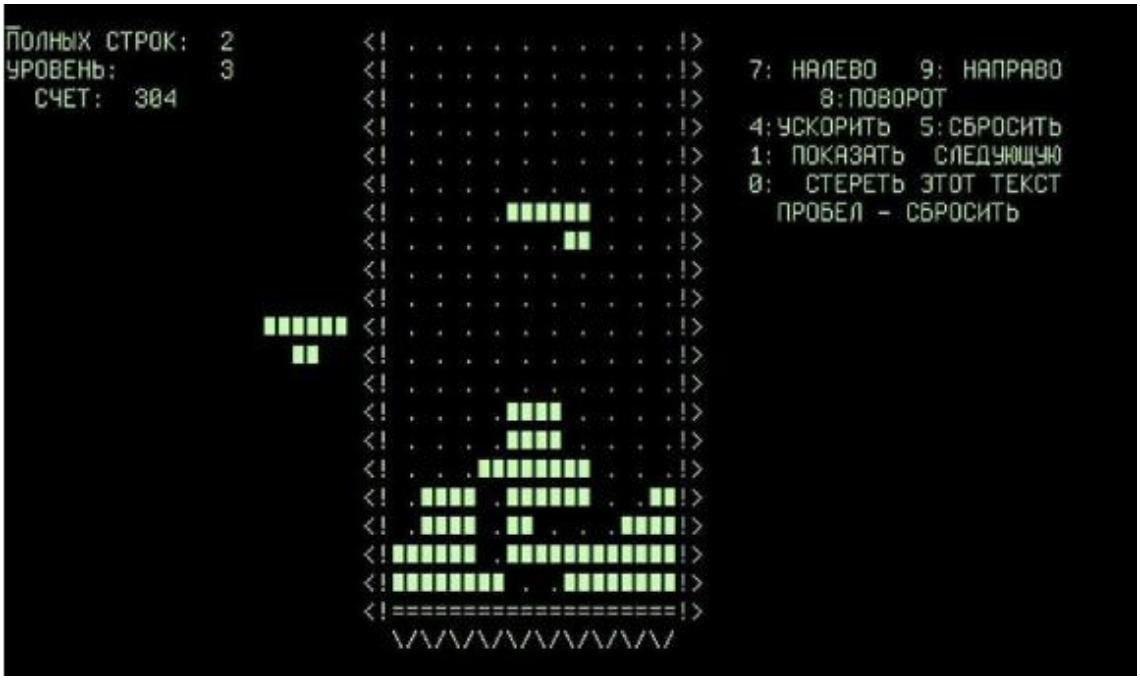

Рисунок 7 – Первый игровой процесс игры тетрис

Современным образцом жанра логических игр выступает игра, созданная Павлом Толкачёвым, под наименованием «Филворды – найди слова из букв». Исходя из информации источника [3], данная игра обладает лидерской позицией в жанре в России, с численностью скачиваний более 10 миллионов. На рисунке 8 отображено игровое поле этой игры.

В центре внимания игрового процесса располагается зона для игры, занимающая 80% экрана, что соответствует общим стандартам для данного жанра. Это обусловлено тем, что ключевой акцент делается на четкость и распознавание игрового процесса.

Рисунок 8 – Игровое поле игры «Филворды – найди слова из букв»

Также среди представителей жанра «Logic», есть еще множесто видов игр.

1. Стратегии. В этих играх игроку предлагается управлять ресурсами, строить здания, развивать армию и сражаться с противниками. Примерами стратегий могут служить цивилизация, Starcraft и Age of Empires.
2. Пазлы. В этих играх игроку предлагается собрать картинку из множества маленьких кусочков. Пазлы могут быть разной сложности и размера, и они также пользуются большой популярностью среди любителей логических игр.
3. Ребусы. В этих играх игроку предлагается расшифровать различные коды и шифры. Примером такой игры может служить Cryptograms.
4. Игры на логическое мышление. Сюда относятся шахматы и другие игры, требующие от игрока глубокого анализа ситуации и принятия правильных решений.
5. Математические игры. В этих играх игроку предлагаются математические задачи различной сложности. Примерами таких игр могут служить 2048 и Математикус.

6. Игры на память. В этих играх игроку предлагается запомнить расположение объектов на экране и воспроизвести его через некоторое время. Примером такой игры может служить Simon Says.

7. Игры на пространственное мышление. В этих играх игроку предлагается решить задачу, связанную с перемещением объектов в пространстве. Примером такой игры может служить Rubik's Cube.

8. Логические викторины. В этих играх игроку предлагаются вопросы на проверку знаний в различных областях. Примером такой игры может служить Trivial Pursuit.

9. Игры на поиск предметов. В этих играх игроку предлагается найти определенные объекты на экране. Примером такой игры может служить Hidden Object Games.

10. Игры на решение загадок. В этих играх игроку предлагается разгадать загадку или головоломку, чтобы продвинуться дальше в игре. Примером такой игры может служить The Room.

В жанре «Logic» можно найти множество уникальных и захватывающих игр, которые бросают вызов вашим умственным способностям. От классических головоломок до сложных стратегий, от красочных пазлов до таинственных ребусов – каждая игра предлагает уникальный опыт и возможность проверить свои навыки логического мышления. Математические игры помогут улучшить ваши навыки счета, а игры на память помогут развить способность запоминать информацию. Криптографические игры станут настоящим вызовом для любителей дешифровки и кодирования. Каждая игра в жанре «Logic» станет отличным способом провести время с пользой и удовольствием.

2. ФОРМУЛИРОВКА ТРЕБОВАНИЙ К ПРОЕКТИРУЕМОЙ ИГРЕ

Функциональные требования

Функциональные требования – это набор правил, которым должна соответствовать система, чтобы выполнять свои функции. Они определяют, какие задачи должна решать система и каким образом она должна это делать. Функциональные требования описывают функциональность системы, то есть ее возможности и способы их реализации. Разрабатываемое приложение должно удовлетворять следующим функциональным требованиям.

1. Игра должна обладать меню, в котором будет предоставлен выбор между началом игры и выходом.
2. Игра должна обладать меню, выбором размера игрового поля.
3. Игра должна предоставлять меню выбора между игроком и автоматическим решения головоломки.
4. Игра должна обладать меню, выбора уровня сложности.
5. Игра должна обладать кнопкой выхода в главное меню.
6. Игра должна работать на платформе Windows. Это означает, что игра должна быть совместима с операционной системой Windows и корректно работать на компьютерах под управлением этой ОС.
7. Графика и игровой процесс должны быть оптимизированы для обеспечения плавного игрового опыта без задержек и зависаний
8. Игра должна быть устойчивой к сбоям и ошибкам, чтобы пользователи могли продолжать играть без прерываний.

Нефункциональные требования

Нефункциональные требования – это требования, которые относятся к характеристикам системы, но не связаны напрямую с ее функциональностью. Они определяют, как система должна работать, но не описывают конкретные функции, которые она должна выполнять. Примерами нефункциональных требований могут быть производительность, надежность, безопас-

ность, масштабируемость и другие характеристики системы. Разрабатываемое игровое приложение должно удовлетворять следующим нефункциональным требованиям:

1) иметь интуитивно понятный интерфейс [4];
2) быть написанным на Python [5, 6, 7];
3) быть написано в оболочке Pycharm [8].

Концепция

Предлагаемым игровым приложением является логическая игра на платформе Windows. Представляющую собой логические задачи на комбинаторику.

Цель игры

Целью игры является развлекательный характер, достигаемый путем прохождения числовой головоломки, также игра предлагает режим создания и решения игровой задачи, созданной пользователем. Задачей каждого игрового поля является успешное решение игровой задачи, характеризующееся расстановкой последовательности чисел в столбцы, строки и блоки.

Игровые уровни

Различие между уровнями игры заключается в размере игрового поля и уровне сложности. Это увеличивает сложность решения головоломки. Самым сложным является уровень с самым большим полем и самой высокой сложностью.

Интерфейс

Интерфейс игрового приложения состоит нескольких игровых меню с разным функционалом. Одна из них появляется перед игроком сразу после загрузки приложения. Она состоит из кнопки «Играть», кнопки «Выход» изображены на рисунке 9.

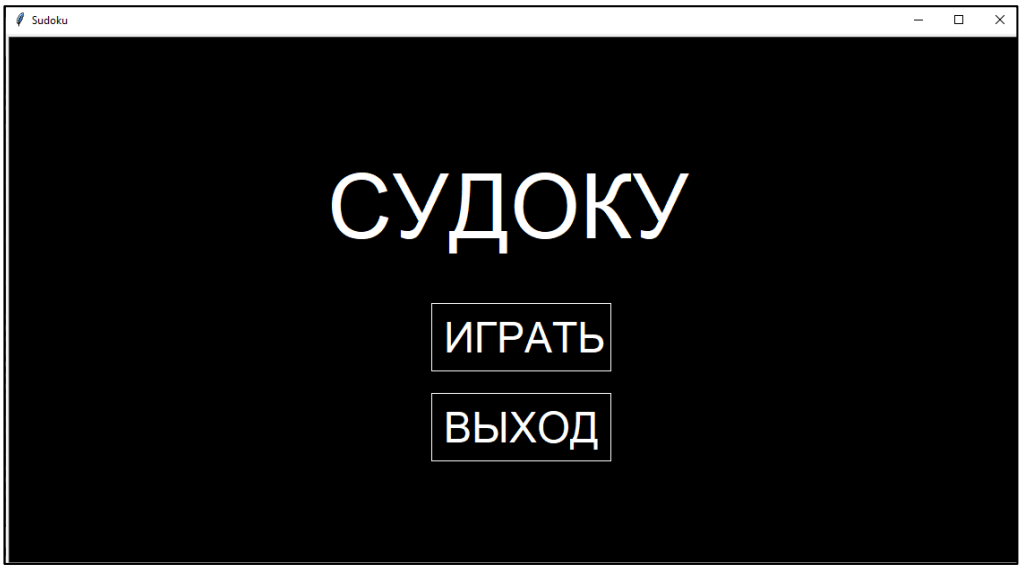

Рисунок 9 – Главное меню

При нажатии кнопки «ИГРАТЬ» показывается панель «Выбора размера поля», показана на рисунке 10, на которой предоставлены кнопки размера поля: «4 x 4», «9 x 9», «16 x 16». При нажатии кнопки «ВЫХОД» происходит выход из игрового приложения.

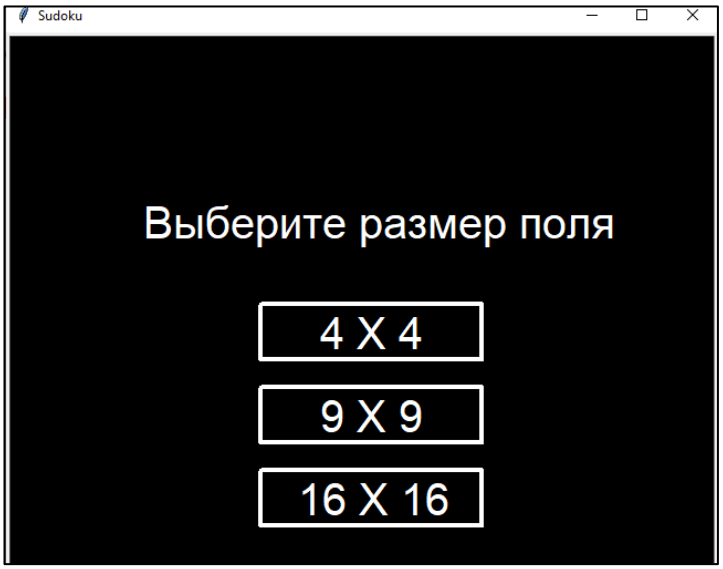

Рисунок 10 – Меню выбора размера игрового поля

После выбора размера игрового поля появляется панель выбора режима (рисунок 11), в котором предоставлены 2 кнопки «Игрок» и «Решить».

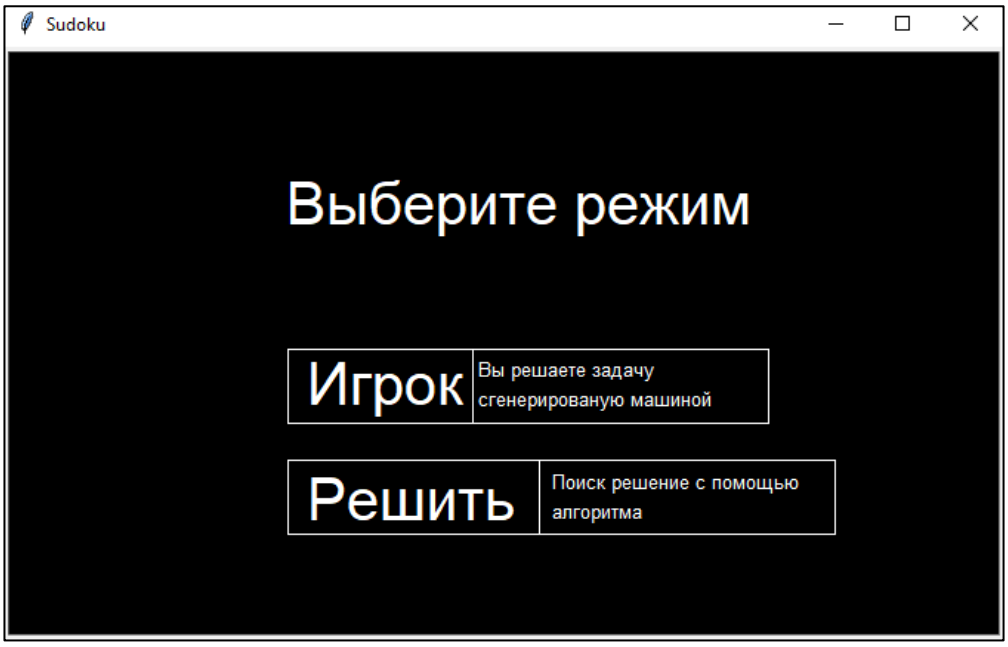

Рисунок 11 – Меню выбора режима игры

Эти две копки непосредственно предлагают начать выбранный игровой режим. В случае нажатии кнопки «Решить», появляется игровое поля с выбранным размером игрового поля. При нажатии кнопки «Игрок» появляется меню выбора сложности (рисунок 12), в котором предоставляется возможность выбора уровень сложности игры.

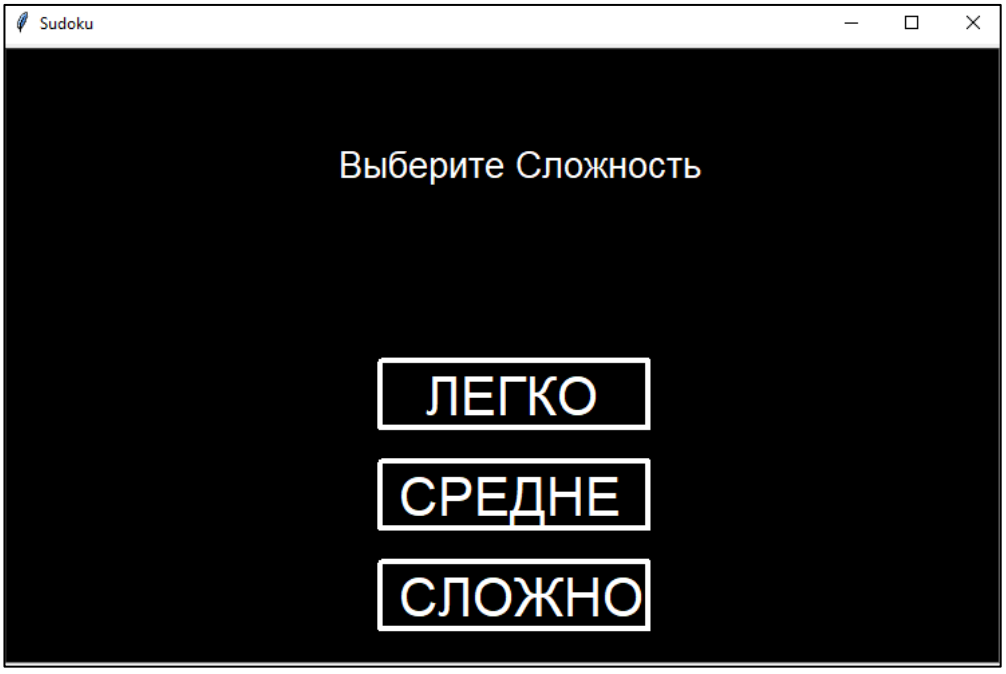

Рисунок 12 – Меню выбора сложности

Игровой процесс

Игровой процесс имеет два режима игры. Режи «Игрок», где алгоритм создает игровое поле по заранее выбранным параметрам и предлагает решить его пользователю (рисунок 13).

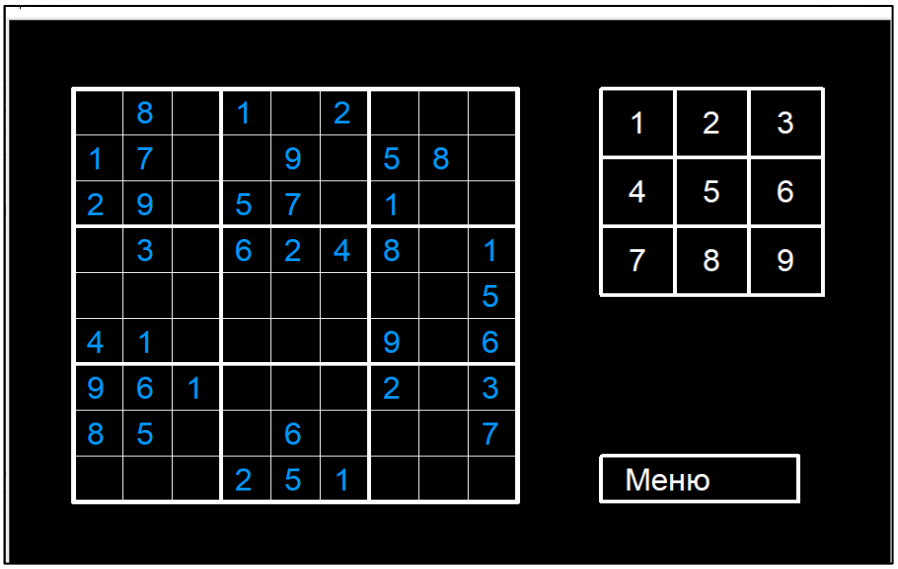

Рисунок 13 – Игровое поле в режиме «Игрок»

Во втором же режиме игроку предоставляется возможность создать головоломку самому и проверить ее на возможное решение. На Рисунок **14** – Игровое поле в режиме «Решить»рисунке 14 показано пустое игровое поле режима «Решить», которое заполняет сам игрок.

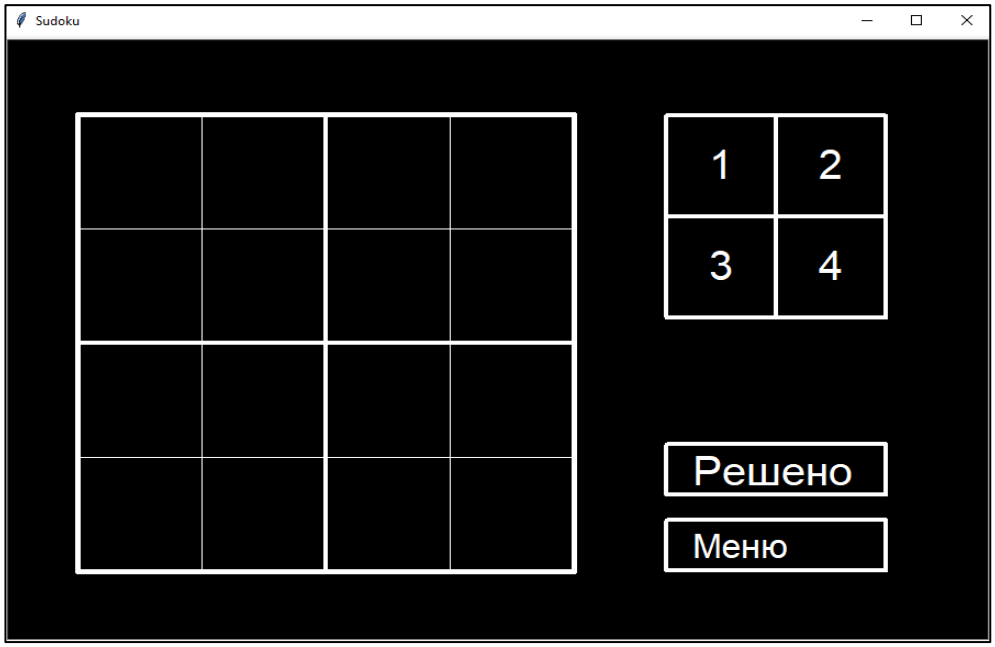

Рисунок 14 – Игровое поле в режиме «Решить»

На рисунке 15 продемонстрировано игровое поле, которое нарушает правило головоломки, игра подсвечивает неправильно поставленную цифру красным цветом, чтобы игрок сразу обратил внимание на нее.

Рисунок 15 – Игровое поле в режиме «Решить» с ошибкой

После решения головоломки в обоих режимах все числа на игровом поле загораются зеленым цветом, что сигнализирует о прохождении головоломки (рисунок 16).

Рисунок 16 – Решенное игровое поле

3. ФУНКЦИОНАЛ РАЗРАБАТЫВАЕМОЙ ИГРЫ

3.1. Диаграмма прецедентов

При помощи языка графического описания объектного ориентирования UML [9], было построено четыре диаграмм прецедентов игровых приложений.

В рамках реализуемого приложения предусмотрен один актер пользователь, который взаимодействует с игровым приложением.

Диаграмма вариантов использования главного меню (рисунок 17). В ней находится два вариантов использования такие как начать игру и закрыть приложение игры.

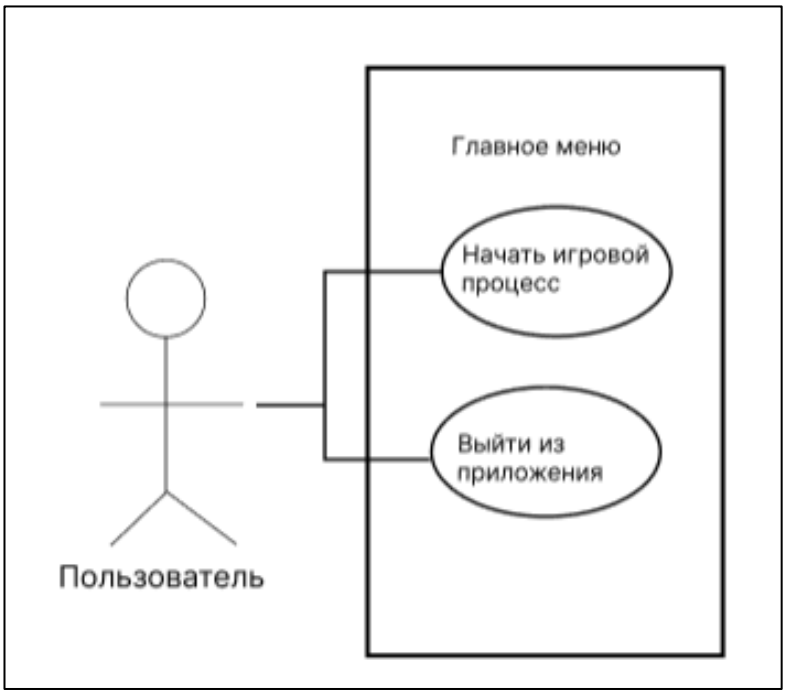

Рисунок 17 – Диаграмма вариантов использования в главном меню

Диаграмма варианта использования меню выбора размера игрового поля представлена на рисунке 18. В ней находится 3 варианта, которые устанавливают выбранный размер игрового поля.

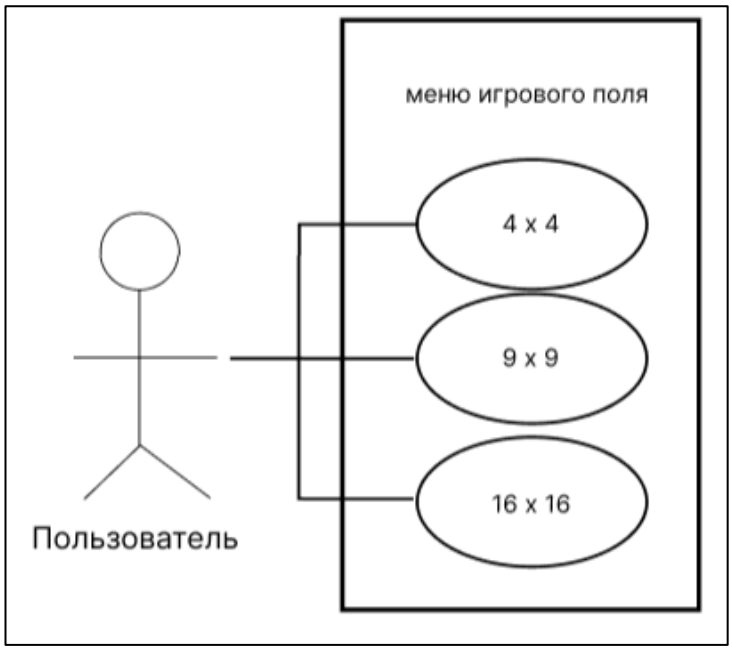

Рисунок 18 – Диаграмма вариантов использования меню выбора размера игрового поля

Диаграмма варианта использования меню выбора режима игрового процесса представлена на рисунке 19.

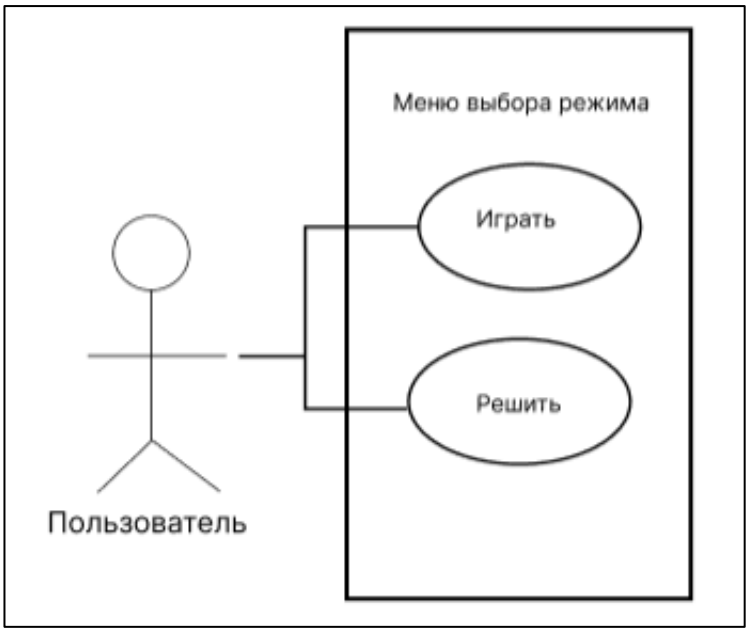

Рисунок 19 – Диаграмма вариантов использования меню выбора игрового режима

Диаграмма вариантов использования игрового процесса пользователем представлена на рисунке 20.

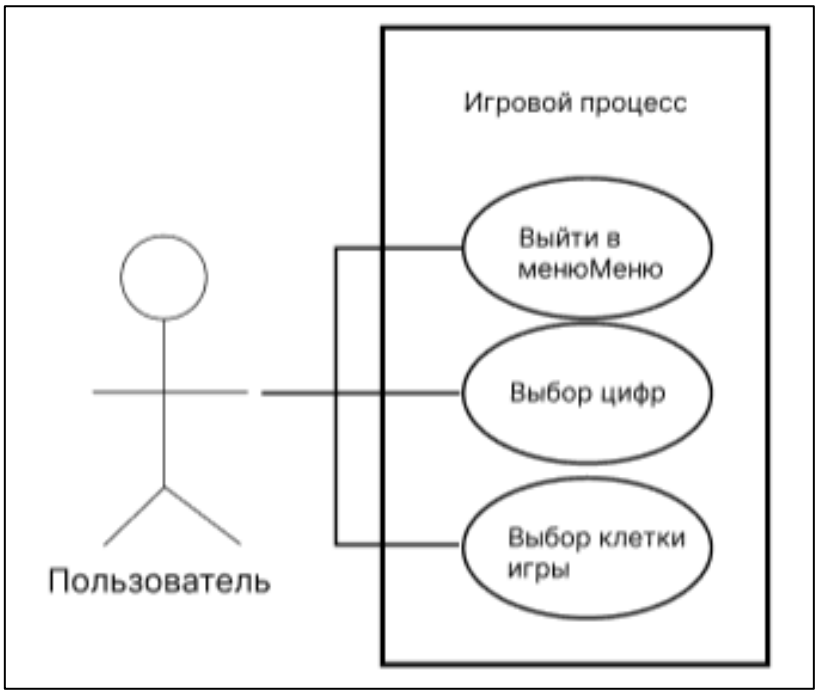

Рисунок 20 – Диаграмма вариантов использования игрового процесса пользователем

Краткое описание вариантов использования системы пользователем:

1) начать игровой процесс – начинает игровой процесс;

2) выйти из игрового приложения – выходит из игры;

3) 4x4 – создает игровое поля размером четыре ячейки по вертикали 5 четыре по горизонтали;

4) 9x9 – создает игровое поля размером девять ячеек по вертикали и девять по горизонтали;

5) 16x16 – создает игровое поля размером шестнадцать ячеек по вертикали и шестнадцать по горизонтали;

6) играть – режим позволяющий решить сгенерированную алгоритмом головоломку, которую решает пользователь;

7) решить – предоставляет пустое поле с возможностью самому создать головоломку и возможностью алгоритма ее решить;

8) меню – выход в главное меню;

9) выбор цифр – выбирает цифру, которую можно будет поставить;

10) выбор клетки игры – вставляет выбранную цифру в клетку.

4. РЕАЛИЗАЦИЯ

Для реализации игрового приложения был выбран язык программирования Python, были также выбраны такие модули.

1. Turtle [10] – этот модуль предоставляет простой интерфейс для рисования двухмерных изображений.

2. Random [11] – этот модуль предоставляет функции для генерации случайных чисел.

3. Math [12] – этот модуль предоставляет математические функции.

4. Time [13] – этот модуль позволяет программе споосб работать с датой и временем.

5. Os [14] – этот модуль предоставляет функции для работы с операционной системой.

4.1. Реализация компонента пользовательского интерфейса

Реализован метод `welcome_screen` для отображения входного меню, он устанавливает размер экрана приложения, рисует две кнопки начала игры и кнопку выхода (рисунок 21).

```
def welcome_screen():
    screen.setup(600, 600)
    pen.clear()
    pen2.clear()
    pen.goto(-210, 50)
    pen.write('СУДОКУ', font = ('Arial', 75, 'normal'))
    pen.goto(-95, -0)
    rect(pen, 200, 75)
    pen.goto(-95, -100)
    rect(pen, 200, 75)
    pen.goto(-80, -65)
    pen.write('ИГРАТЬ', font = ('Arial', 35, 'normal'))
    pen.goto(-80, -165)
    pen.write('ВЫХОД', font = ('Arial', 35, 'normal'))
    while True:
        screen.onclick(welcome_screen_check)
        if is_welcome_screen_over:
            return
        screen.update()
```

Рисунок 21 – Метод `welcome_screen`

Метод `grid_size_screen` благодаря использованию библиотеки turtle, которая позволяет создать графический интерфейс, создает меню выбора размерности игрового поля и ожидает нажатие в определенной области, после нажатия пользователя на необходимую область возвращает значение выбранного игрового поля (рисунок 22).

Также в методе определен шрифт и размер надписи выбора размерности.

```
def grid_size_screen():
    pen.clear()
    pen2.clear()
    screen.setup(600, 600)
    pen.goto(-200, 50)
    pen.write('Выберите размер поля', font = ('Arial', 30, 'normal'))
    pen.goto(-95, -0)
    coor_board_box = [-95, -75, -150, -50]
    size_board_plase = [200, 50]
    for i in range(1, 4):
        rect(pen, size_board_plase[0], size_board_plase[1])
        pen.goto(coor_board_box[0], coor_board_box[i])
    pen.goto(-40, -50)
    pen.write('4 X 4', font = ('Arial', 30, 'normal'))
    pen.goto(-40, -125)
    pen.write('9 X 9', font = ('Arial', 30, 'normal'))
    pen.goto(-60, -200)
    pen.write('16 X 16', font = ('Arial', 30, 'normal'))
    while True:

        screen.onclick(grid_size_check)
        if is_grid_size_over:
            return
        screen.update()
        time.sleep(0.001)
```

Рисунок 22 – Метод `grid_size_screen`

Реализованный метод `mode_selection_screen` (рисунок 23) создает экран выбора режима, в котором устанавливаются будущие правила игрового поля посредством возвращения результата ожидания функции выбора пользователем нужного режима.

```
def mode_selection_screen():
    pen.clear()
    pen2.clear()
    screen.setup(600, 600)
    pen.goto(-150, 75)
    pen.write('Выберите режим', font = ('Arial', 30, 'normal'))
    pen.goto(-150, 0)
    rect(pen, 125, 50)
    pen.goto(-25, 0)
    rect(pen, 200, 50)
    pen.goto(-136, -47)
    pen.write('Игрок', font = ('Arial', 30, 'normal'))
    pen.goto(-150, -75)
    rect(pen, 170, 50)
    pen.goto(20, -75)
    rect(pen, 200, 50)
    pen.goto(-136, -125)
    pen.write('Решить', font = ('Arial', 30, 'normal'))
    pen.goto(-20, -22)
    pen.write('Вы решаете задачу', font = ('Arial', 10, 'normal'))
    pen.goto(-20, -42)
    pen.write('сгенерированую машиной', font = ('Arial', 10, 'normal'))
    pen.goto(30, -98)
    pen.write('Поиск решение с помощью', font = ('Arial', 10, 'normal'))
    pen.goto(30, -118)
    pen.write('алгоритма', font = ('Arial', 10, 'normal'))
    while True:
        screen.onclick(mode_selection_screen_check)
        if is_mode_selection:
            return
        screen.update()
        time.sleep(0.01)
```

Рисунок 23 – Метод `mode_selection_screen`

Следующий метод `difficulty_screen` (рисунок 24), предоставляет возможность игроку выбрать сложность игры от самого легкого до сложного, разница между сложностями заключается в количестве исходных цифр, чем меньше цифр тем сложнее, количество пустых ячеек в на каждом уровне находится в таблице – Количество пустых ячеек на игровом поле, однако если учитывать метод `grid_size_screen`, то даже при самом сложном уровне, но в поле 4x4 игра все равно будет легкой, так же при легком уровне сложности но размерами поля 16x16 игровой поле игра все же будет сложнее.

```
def difficulty_screen():
    pen.clear()
    pen2.clear()
    screen.setup(600, 600)
    pen.goto(-125, 130)
    pen.write('Выберите Сложность', font = ('Arial', 20, 'normal'))
    pen.goto(-95, -0)
    rect(pen, 200, 50)
    pen.goto(-95, -75)
    rect(pen, 200, 50)
    pen.goto(-95, -150)
    rect(pen, 200, 50)
    pen.goto(-60, -50)
    pen.write('ЛЕГКО', font = ('Arial', 30, 'normal'))
    pen.goto(-80, -125)
    pen.write('СРЕДНЕ', font = ('Arial', 30, 'normal'))
    pen.goto(-80, -200)
    pen.write('СЛОЖНО', font = ('Arial', 30, 'normal'))
    while True:
        screen.onclick(difficulty_check)
        if is_difficulty:
            return
        screen.update()
        time.sleep(0.01)
```

Рисунок 24 – Метод `difficulty_screen`

4.2. Реализация метода проверки

В реализованном методе `isvalid` (рисунок 25) проверяется наличие совпадений в строках, столбцах и блоках игрового поля после установления туда необходимой цифры.

```
def isvalid(self, k, i, j):
    # проверка линии
    if not self.is_used_in_row(k, i):
        # проверка колонки
        if not self.is_used_in_column(k, j):

            if not self.is_used_in_box(k, i, j):
                return True
    return False
```

Рисунок 25 – Метод `isvalid`

Метод `solve_board` (рисунок 26) проверяет вся головоломку на ее решение при каждой вставки цифры в ячейку, при обнаружении полностью заполненной игровой доски она считает головоломку решенной. Алгоритмом решения, реализованного в данном методе, является метод «backtracking» [15].

```
def solve_board(self, row, col):
    r = row  # Текущая строка
    c = col  # Текущий столбец
    screen.update()

        for i in range(self.n):
        for j in range(self.n):

            num = self.board[i][j]
            self.board[i][j] = 999
            if not self.isvalid(num, i, j) and num != 0:
                self.board[i][j] = num
                return False

            self.board[i][j] = num
        if r == self.n - 1 and c == self.n:
        return True

        if c == self.n:
        c -= c
        r += 1
        if self.board[r][c] != 0:
        return self.solve_board(r, c + 1)      for i in range(1, self.n
+ 1):
        if self.isvalid(i, r, c):
            self.board[r][c] = i
            if self.demo:
                self.show_board()
            if self.solve_board(r, c + 1):                  return True
            self.board[r][c] = 0      return False
```

Рисунок 26– Метод `solve_board`

4.3. Реализация метода отображения цифр

Метод `draw_numbers_on_grid` (рисунок 27), продемонстрированный в, является центральным в работе программы и благодаря с его помощью игрок узнает, что правильно решил головоломку, потому что все цифры окрасятся в зеленый. При нахождении ошибки, если несколько одинаковых цифр находятся в одной ячейке или же одном столбце, блоке она подсветит все повторяющиеся ячейки красным цветом, ячейки перестанут святиться красным как только пользователь уберет неверно поставленную цифру из ячейки.

```
def draw_numbers_on_grid():
    d = 50
    size = 25
    if n == 4:
        d = 113
        size = 50
    if n == 16:
        d = 30
        size = 14
    x_shift = -160
    for i in range(n):
        for j in range(n):
            if s.board[i][j] != 0:
                if [i, j] in unmodifiable_cells:
                    pen2.setheading(0)
                    pen2.fillcolor('white')
                    pen2.color('#0099FF')
                else:
                    pen2.pencolor('white')
                    num = s.board[i][j]
                    s.board[i][j] = 999
                    if s.isvalid(num, i, j):
                        pen2.pencolor('white')
                    else:
                        pen2.pencolor('#FF0000')
                    s.board[i][j] = num
                if is_solved():
                    pen2.color('#00FF00')
                if n == 4:
                    pen2.goto(-d * n // 2 + x_shift + j * d + 40, d * n
// 2 - i * d - 90)
                if n == 9:
                    pen2.goto(-d * n // 2 + x_shift + j * d + 15, d * n
// 2 - i * d - 45)
                if n == 16:
                    if s.board[i][j] >= 10:
                        pen2.goto(-d * n // 2 + x_shift + j * d + 5, d *
n // 2 - i * d - 27)
                    else:
                        pen2.goto(-d * n // 2 + x_shift + j * d + 12, d
* n // 2 - i * d - 27)
                pen2.write(str(s.board[i][j]), font = ('Arial', size,
'normal'))
```

Рисунок 27 – Метод `draw_numbers_on_grid`

4.4. Сложность головоломки

В игре реализовано 3 уровня сложности: легко, средне, сложно. Уровень сложности определяется количеством пустых ячеек после генерации всего игрового поля. В таблице 1, предоставлено количество пустых ячеек в зависимости от сложности и размера игрового поля

Таблица 1 – Количество пустых ячеек на игровом поле

Размер поля	Легко	Средне	Сложно
4x4	5	8	10
9x9	27	45	63
16x16	48	80	112

4.5. Создания игрового поля

Цикл генерации игрового поля начинается с создания нулевого игрового пол заполненного нолями. После чего запускается цикл создания головоломки, который обнуляет его, потом вызывает метод `diagonal` (рисунок 28), который заполняет диагональные блоки поля.

```
def diagonal(self):
    for i in range(0, self.n, self.srn):
        self.fill_box(i, i)
```

Рисунок 28 – Метод `diagonal`

Потом вызывает срабатывание метода `solve_board` (рисунок 29), который решает головоломку, начиная с верхнего левого угла, при успешном решении головоломки вызывает метод `remove_elements` (рисунок 28).

```
def remove_elements(self):
    already_removed = []
    count = 0
    while count < self.missing:
        r = random.randint(0, self.n - 1)
        c = random.randint(0, self.n - 1)
        if [r, c] not in already_removed:
            self.board[r][c] = 0
            count += 1
        already_removed.append([r, c])
```

Рисунок 29 – Метод `remove_elements`

Который случайным образом удаляет необходимое количество значений, количество которых зависит от выбранного уровня и размера игрового поля. Если метод `solve_board`, не может решить головоломку программа создает новое поля с новым набором случайных чисел.

5. ТЕСТИРОВАНИЕ

5.1. Функциональное тестирование

Функциональное тестирование – это тестирование программного обеспечения в целях проверки реализуемости функциональных требований, то есть способности ПО в определенных условиях решать задачи, нужные пользователям [16]. Итоги функционального тестирования представлены в таблице 2.

Таблица 2 – Функциональное тестирование

№	Название теста	Ожидаемый результат	Полученный результат	Успех?
1	Запуск игры	Отображения главного меню	Отображения главного меню	да
2	Вызов меню выбора размера	Отображения меню выбора размера игрового поля	Отображения меню выбора размера игрового поля	да
3	Выбор режима	Появления меню выбора режима	Появления меню выбора режима	да
4	Сложность игрока	Появление меню выбора сложности	Появление меню выбора сложности	да
5	Создать самому	Появление пустого игрового поля	Появление пустого игрового поля	да
6	Создание игрового поля	Появление игрового поля с созданной головоломкой	Появление игрового поля с созданной головоломкой	да
7	Выход в главное меню	При нажатии кнопки меню, появляется главное меню	При нажатии кнопки меню, появляется главное меню	да
8	Вставка кнопки	При выборе цифры и ее вставки в игровое поле, на игровом поле появляется число	При выборе цифры и ее вставки в игровое поле, на игровом поле появляется число	да
9	Проверка ошибка	При вставки одинаковых чисел в столб, цифры загораются красным	При вставки одинаковых чисел в столб, цифры загораются красным	да
10	Растяжка	При растягивании окна, игра реагирует корректно	При растягивании окна, игра реагирует корректно	да
11	Решение	При нажатии кнопки решить головоломка сама решается	При нажатии кнопки решить головоломка сама решается	да
12	При введении неверного	После введения невозможной головоломки, пишется ошибка	После введения невозможной головоломки, пишется ошибка	да

Окончание таблицы 2

№	Название теста	Ожидаемый результат	Полученный результат	Успех?
13	Верное решение	После решения головоломки все ячейки загораются зеленым	После решения головоломки все ячейки загораются зеленым	да
14	Закрытие	После закрытия головоломки, игра должна закрыться	После закрытия головоломки, игра должна закрыться	да

5.2. Юзабилити тестирование

Юзабилити-тестирование – это метод оценки удобства и эффективности интерфейса. Для оценки привлекаются представители целевой аудитории продукта, которые работают с интерфейсом, выполняя специально подобранные задания. На основе поведения респондентов юзабилити-специалист делает выводы о наличии в интерфейсе юзабилити-проблем и их характере [17].

Участникам юзабилити тестирования были выданы следующие задачи:

1) создать игровое поля размером 4 х 4;
2) создать игровое поля размером 9 х 9;
3) создать игровое поля размером 16 х 16;
4) пройти головоломку;
5) проверить неправильность постановки числа;
6) создать свою головоломку;
7) получить решение вашей головоломки;
8) свернуть приложение;
9) создать новую головоломку;
10) свернуть приложение.

В ходе юзабилити тестирования была выявлена проблема с генерацией головоломки размером 16 х 16 (рисунок 30).

```
hard : 16  * - * - * - * - *  7.411833763122559
hard : 16  * - * - * - * - *  11.214726209640503
hard : 16  * - * - * - * - *  17.382036447525024
hard : 16  * - * - * - * - *  39.58124375343323
hard : 16  * - * - * - * - *  4.263978958129883
hard : 16  * - * - * - * - *  12.121653318405151
hard : 16  * - * - * - * - *  4.568037033081055
hard : 16  * - * - * - * - *  27.310839414596558
hard : 16  * - * - * - * - *  60.229146003723145
hard : 16  * - * - * - * - *  34.20128035545349
hard : 16  * - * - * - * - *  8.580033302307129
```

Рисунок 30 – Логи алгоритма создания уровня сложности «сложно» на игровом поле 16x16

После всестороннего анализа и тщательного тестирования было принято решение усовершенствовать алгоритм создания игрового поля, основанный на методе «backtracking». В рамках этого процесса были внесены ряд изменений, направленных на оптимизацию работы алгоритма. Одним из ключевых аспектов стало снижение количества проверок, необходимых для функционирования алгоритма. Благодаря этим изменениям, время, затрачиваемое на создание игрового поля, существенно сократилось, результаты представлены на рисунке 31.

```
hard : 16  * - * - * - * - *  0.04401731491088867
hard : 16  * - * - * - * - *  1.2181732654571533
hard : 16  * - * - * - * - *  0.40109062194824222
hard : 16  * - * - * - * - *  0.057014226913452152
hard : 16  * - * - * - * - *  0.38838872146606445
hard : 16  * - * - * - * - *  0.39309000968933105
hard : 16  * - * - * - * - *  0.16704654693603516
hard : 16  * - * - * - * - *  0.12302756309509277
hard : 16  * - * - * - * - *  0.07901811599731445
hard : 16  * - * - * - * - *  2.282473087310791
hard : 16  * - * - * - * - *  0.033005714416503906
```

Рисунок 31 – Логи алгоритма «backtracking», создания уровня сложности «сложно» на игровом поле 16x16

ЗАКЛЮЧЕНИЕ

В рамках данной работы была разработана компьютерная игра в жанре «Logic», специально адаптированная для операционной системы Windows. Реализация проекта осуществлялась с использованием языка программирования Python, который обладает широкими возможностями для создания интерактивного контента и удобными инструментами для разработки программного обеспечения.

В процессе работы над проектом были решены следующие задачи.

1. Анализ современных тенденций в разработке игр.
2. Формулировка концепции игры, определение ее основных принципов и особенностей.
3. Разработка функциональной структуры игры, создание прототипа и последующая его доработка.
4. Реализация полноценного игрового приложения, включая графический интерфейс, логику игры и обработку пользовательского ввода.
5. Тестирование игры на различных уровнях сложности, выявление и устранение ошибок.

Для дальнейшего улучшения и расширения игры запланированы несколько шагов.

1. Добавление звукового сопровождения, создание атмосферных звуковых эффектов и музыкального фона.
2. Разработка дополнительных игровых режимов, включая кооперативный и соревновательный режимы, а также режимы с различными уровнями сложности.
3. Внедрение системы достижений и наград, чтобы стимулировать игроков к более глубокому погружению в игровой процесс.

ЛИТЕРАТУРА

1. Официальный сайт newzoo. [Электронный ресурс] URL: https://newzoo.com (дата обращения: 11.02.2024 г.).

2. История тетриса. Как это было на самом деле. [Электронный ресурс] URL: https://www.igromania.ru/article/11612/Istoriya_tetrisa._Kak_yeto _bylo_na _samom_dele.html (дата обращения: 10.05.2024 г.).

3. Филворды – найди слова. [Электронный ресурс] URL: https://play.google.com/store/apps/details?id=com.merigotech.fillwords&hl=ru (дата обращения: 01.05.2024 г.).

4. Круг С. Не заставляйте меня думать. // Москва, издательство «Э», 2017. – 256 с.

5. Свейгар, Эл. Большая книга проектов Python. – СПб.: Питер, 2022. – 432 с.

6. Лутц М. Изучаем Python, том 2, 5-е изд.: Пер. с англ. – СПб.: ООО «Диалектика», 2020. – 720 с.

7. Бизли Д., Джонс Б. К Python. Книга рецептов / пер. с анг. Б. В. Уварова. – М.: ДМК Пресс, 2019. – 648 с.

8. Pycharm. [Электронный ресурс] URL: https://www.jetbrains.com/ru-ru/ (дата обращения: 10.04.2024 г.).

9. Хританков А.С. Проектирование на UML [Текст] / А.С. Хританков, В.А. Полежаев, А.И. Андрианов // Директ–Медиа: сб. статей. – Москва, 2018. – 242 с.

10. Библиотека turtle – Turtle graphics. [Электронный ресурс] URL: https://docs.python.org/3/library/turtle.html (дата обращения: 20.05.2024 г.).

11. Библиотека random – Generate pseudo–random numbers. [Электронный ресурс] URL: https://docs.python.org/3/library/random.html (дата обращения: 22.05.2024 г.).

12. Библиотека math – Mathematical functions. [Электронный ресурс] URL: https://docs.python.org/3/library/math.html (дата обращения: 23.05.2024 г.).

13. Библиотека time – Time access and conversions. [Электронный ресурс] URL: https://docs.python.org/3/library/time.html (дата обращения: 23.05.2024 г.).

14. Библиотека os – Miscellaneous operating system interfaces. [Электронный ресурс] URL: https://docs.python.org/3/library/os.html (дата обращения: 23.05.2024 г.).

15. Рафгарден Т. Совершенный алгоритм. Основы. – СПБ.: Питер, 2019. – 256 с.

16. Функциональное тестирование программного обеспечения. [Электронный ресурс] URL: https://unetway.com/tutorial/funkcionalnoe-testirovanie (дата обращения: 15.05.2024 г.).

17. Удаленное юзабилити тестирование сайта. [Электронный ресурс] URL: https://usabilitylab.ru/usability/yuzabiliti-testirovanie (дата обращения: 15.05.2024 г.).

ПРИЛОЖЕНИЕ. Листинг класса «Sudoku» и модуля расположения цифр на игровом поле

Листинг 1 – Реализация модуля расположения цифр

```
def draw_buttons(n):
    srn = math.sqrt(n)
    active_buttons = [i for i in buttons if buttons.index(i) < n]
    for i in buttons:
        i.is_active = False
    for i in active_buttons:
        i.is_active = True
        size = 25
        if n == 4:
            size = 30
        if n == 9:
            i.width = 75
            i.length = 75
        if n == 4:
            i.width = 100
            i.length = 100
        if n == 16:
            i.width = 60
            i.length = 60

        i.x = 150 + (buttons.index(i) % srn) * i.length
        if n == 4:
            i.y = 125 - (buttons.index(i) // srn) * i.width
        else:
            i.y = 150 - (buttons.index(i) // srn) * i.width
        pen.goto(i.x, i.y)
        pen.setheading(0)
        pen.pendown()
        for j in range(2):
            pen.forward(i.length)
            pen.left(90)
            pen.forward(i.width)
            pen.left(90)
        pen.penup()
        if int(i.message) < 10:
            pen.goto(i.x + i.length // 2 - size * 1 // 3, i.y + i.width //
2 - size * 3 // 4)
        else:
            pen.goto(i.x + i.length // 2 - size * 2 // 3, i.y + i.width //
2 - size * 3 // 4)
        pen.write(i.message, font = ('Arial', size, 'normal'))
        if not i.is_active:
            pen.goto(i.x, i.y + i.width)
            pen.pendown()
            pen.goto(i.x + i.length, i.y)
            pen.penup()
        if i.is_selected:
            pen.goto(i.x + 5, i.y + i.width - 5)
            pen.pendown()
            for k in range(2):
                pen.forward(i.length - 10)
                pen.right(90)
                pen.forward(i.width - 10)
                pen.right(90)
            screen.update()
            pen.penup()
```

Листинг 2 – Реализация класса

```
class Sudoku:
    def __init__(self, n = 9, missing = 0, demo = True):
        # n размер матрицы
        self.n = n
        self.board = [[0 for i in range(n)] for j in range(n)]
        self.unsolved_board = []
        self.srn = int(math.sqrt(n))
        self.missing = missing
        self.demo = demo
        self.timer = time.time()

    def fill_box(self, row, col):
        num = [i for i in range(1, self.n + 1)]
        for i in range(self.srn):
            for j in range(self.srn):
                x = random.choice(num)
                num.remove(x)
                self.board[row + i][col + j] = x

    def diagonal(self):
        for i in range(0, self.n, self.srn):
            self.fill_box(i, i)

    def is_used_in_row(self, n, r):
        for i in range(self.n):
            if self.board[r][i] == n:
                return True
        return False

    def is_used_in_column(self, n, c):
        for i in range(self.n):
            if self.board[i][c] == n:
                return True
        return False

    def is_used_in_box(self, n, r, c):
        box_r = self.srn * (r // self.srn)
        box_c = self.srn * (c // self.srn)
        for i in range(self.srn):
            for j in range(self.srn):
                if self.board[box_r + i][box_c + j] == n:
                    return True
        return False

    def isvalid(self, k, i, j):
        # проверка линии
        if not self.is_used_in_row(k, i):
            # проверка колонки
            if not self.is_used_in_column(k, j):
                if not self.is_used_in_box(k, i, j):
                    return True
        return False
def solve_board(self, row, col):
    r = row  # Текущая строка
    c = col  # Текущий столбец
    screen.update
    for i in range(self.n):
        for j in range(self.n):
            num = self.board[i][j]
```

```
            self.board[i][j] = 999
            if not self.isvalid(num, i, j) and num != 0:
                self.board[i][j] = num                  return False
            self.board[i][j] = num
    if r == self.n - 1 and c == self.n:
        return True

    if c == self.n:
        c -= c
        r += 1
    if self.board[r][c] != 0:
        return self.solve_board(r, c + 1)  # Решаем следующутю ячейку в
солбце
    for i in range(1, self.n + 1):
        if self.isvalid(i, r, c):
            self.board[r][c] = i
            if self.demo:
                self.show_board()
            if self.solve_board(r, c + 1                return True
            self.board[r][c] = 0
    return False
```