

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

федеральное государственное автономное образовательное учреждение
высшего образования «Самарский национальный исследовательский
университет имени академика С.П. Королева (Самарский университет)»

Институт _____ информатики и кибернетики _____

Кафедра _____ программных систем _____

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

_____ к курсовой работе по дисциплине «Программная инженерия»
_____ по теме «Автоматизированная система генерирования
_____ структуры лабиринта и нахождения выхода из него»

Обучающийся _____ Е.А. Балашова

Обучающийся _____ В.А. Гриднева

Руководитель _____ Л.С. Зеленко

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

федеральное государственное автономное образовательное учреждение
высшего образования «Самарский национальный исследовательский
университет имени академика С.П. Королева (Самарский университет)»

Институт _____ информатики и кибернетики _____

Кафедра _____ программных систем _____

ЗАДАНИЕ

на курсовую работу по дисциплине
«Программная инженерия»
обучающимся в группе № 6402-020302D
Е.А. Балашовой
В.А. Гридневой

Тема проекта: «Автоматизированная система генерирования структуры
лабиринта и нахождения выхода из него» _____

- 1 Исходные данные к проекту: см. приложение к заданию _____
- 2 Перечень вопросов, подлежащих разработке:
 - 2.1 Произвести анализ предметной области: изучить основные принципы составления лабиринтов, изучить алгоритмы генерации лабиринтов и их решения _____
 - 2.2 Выполнить обзор существующих систем-аналогов _____
 - 2.3 Разработать информационно-логический проект системы по методологии UML _____
 - 2.4 Разработать и реализовать программное и информационное обеспечение, провести его тестирование и отладку _____.
 - 2.5 Оформить документацию курсовой работы _____
 - 2.6 Подготовить презентацию по разработанной системе _____
- 3 Перечень графических разработок:
 - 3.1 Структурная схема системы _____
 - 3.2 Канонические диаграммы UML _____
 - 3.3 Схемы основных алгоритмов _____

4 Календарный план выполнения работ

№ п/ п	Содержание работы по этапам	Объем этапа в % к общему объему проекта	Срок окончания	Фактическое выполнение
1	Оформление технического задания и его утверждение	5	20.09.2024	
2	Описание и анализ предметной области	10	27.09.2024	
3	Проектирование системы	40	13.12.2024	
3.1	Разработка структурной схемы системы	5	11.10.2024	
3.2	Разработка функциональной спецификации системы и прототипа интерфейса пользователя	10	25.10.2024	
3.3	Разработка информационно-логического проекта системы и его предъявление руководителю	25	13.12.2024	
4	Реализация проекта, разработка контрольных примеров. Предъявление реализации руководителю	40	13.12.2024	
5	Корректировка проекта и оформление документации проекта. Защита проекта с представлением презентации.	5	27.12.2024	

Задание принял
к исполнению

_____ Е.А. Балашова
_____ В.А. Гриднева

ПРИЛОЖЕНИЕ

к заданию на курсовую работу
обучающимся в группе № 6402-020302D

Е.А. Балашовой

В.А. Гридневой

Тема проекта: «Автоматизированная система генерирования структуры
лабиринта и нахождения выхода из него»

Исходные данные к проекту:

1 Характеристика объекта автоматизации:

- 1) объект автоматизации: лабиринт;
- 2) виды автоматизируемой деятельности:
 - процесс авторизации и регистрации пользователей;
 - процесс генерирования лабиринта в соответствии с алгоритмами;
 - процесс расстановки входа и выхода;
 - процесс автоматического прохождения лабиринта;
 - процесс визуализации нахождения выхода из лабиринта;
- 3) количество ролей пользователей – 2;
- 4) минимальный размер лабиринта по вертикали – 7;
- 5) максимальный размер лабиринта по вертикали – 21;
- 6) минимальный размер лабиринта по горизонтали – 7;
- 7) максимальный размер лабиринта по горизонтали – 21;
- 8) количество входов – 1;
- 9) количество выходов – 1;
- 10) количество способов расстановки входа и выхода – 2;
- 11) количество алгоритмов создания лабиринта – 2;
- 12) количество тем оформления лабиринта – 4;
- 13) количество алгоритмов прохождения лабиринта – 2;
- 14) количество скоростных режимов перемещения персонажа – 4;
- 15) минимальная длина логина – 6 символов;
- 16) максимальная длина логина – 16 символов;

- 17) минимальная длина пароля – 6 символов;
- 18) максимальная длина пароля – 20 символов;
- 2 Требования к информационному обеспечению:
 - 1) информационное обеспечение разрабатывается на основе следующих источников:
 - лабиринты: классификация, генерирование, поиск решений [Электронный ресурс]. URL: <https://habr.com/ru/articles/445378/> (дата обращения: 16.09.2024);
 - алгоритмы генерации лабиринтов и нахождения пути [Электронный ресурс]. URL: <https://tproger.ru/articles/maze-generators> (дата обращения: 16.09.2024);
 - генерация лабиринтов: алгоритм Эллера [Электронный ресурс]. URL: <https://habr.com/ru/articles/667576/> (дата обращения: 16.09.2024);
 - 2) требования к построению лабиринта:
 - лабиринт имеет формат 2D;
 - по периметру лабиринта – стена;
 - толщина стен равна толщине прохода;
 - вход и выход находится на периметре лабиринта, но не в углах;
 - вход и выход не совпадают;
 - стены не должны образовывать изолированные части;
 - наличие тупиков;
 - наличие пути;
 - 3) лабиринты хранятся в файлах, структура файла определяется в процессе проектирования.
- 3 Требования к техническому обеспечению:
 - 3.1 Требования к техническому обеспечению серверной части:
 - 1) тип ЭВМ – IBM PC совместимый;
 - 2) объем ОЗУ – не менее 2 Гб;
 - 3) объем свободного пространства на внешнем диске – не менее 50 Гб;

- 4) наличие подключения к сети Интернет;
- 5) манипулятор – мышь;
- 6) технические характеристики определяются в процессе выполнения проекта;

3.2 Требования к техническому обеспечению клиентской части:

- 1) тип ЭВМ – IBM PC совместимый;
- 1) монитор с разрешающей способностью не ниже 800 x 600;
- 2) манипулятор – мышь;
- 3) технические характеристики определяются в процессе выполнения проекта.

4 Требования к программному обеспечению:

4.1 Требования к программному обеспечению серверной части:

- 1) тип операционной системы – Windows 10 и выше.

4.2 Требования к программному обеспечению клиентской части:

- 1) тип операционной системы – Windows 10 и выше;
- 2) браузер – Google Chrome 86.0.4240.183 (64-битный) и выше.

4.3 Требования к программному обеспечению рабочего места разработчика:

- 1) тип операционной системы – Windows 10 и выше;
- 2) язык программирования – C#;
- 3) среда программирования – Visual Studio 2022;
- 4) среда проектирования – StarUML 5.2.0.

5 Общие требования к проектируемой системе:

5.1 Функции, реализуемые системой:

- 1) функции системы:
 - аутентификация пользователя в системе, настройка интерфейса пользователя на заданную роль;
 - генерирование шаблона лабиринта по заданному размеру;
 - случайная расстановка входа и выхода;
 - контроль расстановки входа и выхода;

- создание структуры лабиринта по заданному алгоритму;
 - проверка структуры лабиринта при загрузке;
 - визуализация генерации лабиринта;
 - визуализация прохождения лабиринта (видимая дорожка);
 - выдача справочной информации;
- 2) функции администратора:
- авторизация пользователя в системе (ввод логина и пароля);
 - настройка параметров лабиринта при создании:
 - 1) задание размера по горизонтали;
 - 2) задание размера по вертикали;
 - 3) выбор способа расстановки входа и выхода;
 - 4) выбор алгоритма генерации лабиринта;
 - 5) настройки темы оформления;
 - ручная расстановка входа и выхода;
 - сохранение лабиринта в файл заданной структуры;
 - просмотр лабиринта;
 - просмотр справочной информации;
- 3) функции игрока:
- регистрация пользователя в системе (ввод логина и пароля);
 - авторизация пользователя в системе (ввод логина и пароля);
 - выбор лабиринта;
 - загрузка лабиринта из файла;
 - изменение темы оформления лабиринта;
 - выбор алгоритма прохождения лабиринта;
 - ручное прохождение лабиринта;
 - выбор алгоритма прохождения лабиринта;
 - выбор режима прохождения лабиринта;
 - выбор задержки перемещения персонажа (для автоматического режима);
 - запуск алгоритма

– просмотр справочной информации.

5.2 Технические требования к системе:

- 1) режим работы – диалоговый;
- 2) время автоматической генерации лабиринта – не более 5 с;
- 3) система должна удовлетворять санитарным правилам и нормам СанПин 2.2.2./2.4.2198-07;
- 4) условия работы средств вычислительной техники (содержание вредных веществ, пыли и подвижность воздуха) должны соответствовать ГОСТ 12.1.005, 12.01.007;
- 5) температура окружающего воздуха – 15-35°C;
- 6) влажность воздуха – 45-75%.

Руководитель

проекта _____ Л.С. Зеленко

Задание принял

к исполнению _____ 13.09.2024 В.А. Гриднева

_____ 13.09.2024 Е.А. Балашова

РЕФЕРАТ

Пояснительная записка 104 с, 61 рисунков, 5 таблиц, 40 источников, 2 приложения.

Графическая часть: 17 слайдов презентации PowerPoint.

ГЕНЕРАТОР ЛАБИРИНТОВ, ГОЛОВОЛОМКА, ЛАБИРИНТ, АЛГОРИТМЫ ПРОХОЖДЕНИЯ ЛАБИРИНТОВ, ТУПИКИ, ПРОХОДЫ, ПУТЬ, НАХОЖДЕНИЕ ПУТИ, РУЧНОЕ ПРОХОЖДЕНИЕ ЛАБИРИНТА.

Объектом автоматизации является лабиринт.

Во время курсового проектирования разработаны алгоритмы и соответствующая им программа, позволяющая выполнять автоматическую генерацию лабиринта и нахождения выхода из него. Лабиринт может быть размером от 7 до 21, может быть сгенерирован с помощью двух алгоритмов генерации, можно задать вход и выход в ручном режиме и в автоматическом. Программа позволяет сохранить лабиринт в файл заданной структуры, а также загрузить его. Лабиринт может быть оформлен в 4 темах, пройти можно как в ручном режиме, так и в автоматическом с помощью двух алгоритмов нахождения пути.

Программа написана на языке C# в среде Visual Studio 2022 и функционирует под управлением операционной системы Windows 10 и выше. Доступ к данным осуществляется с помощью текстового файла.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ	12
1 Описание и анализ предметной области	14
1.1 Описание предметной области.....	14
1.1.1 Описание лабиринта.....	14
1.1.2 Классификация лабиринтов.....	14
1.1.3 Описание алгоритмов генерации лабиринта	21
1.1.4 Описание алгоритмов нахождения пути	23
1.2 Описание систем-аналогов	24
1.2.1 Gotcha.....	25
1.2.2 Онлайн генератор лабиринтов «Plottersvg»	26
1.3 Диаграмма объектов предметной области	27
1.4 Постановка задачи.....	30
1.4.1 Режим администратора	30
1.4.2 Режим игрока	31
2 Проектирование системы.....	33
2.1 Выбор и обоснование архитектуры системы.....	33
2.2 Структурная схема системы	35
2.3 Разработка спецификации требований.....	37
2.3.1 Функциональная спецификация.....	38
2.3.2 Перечень исключительных ситуаций	39
2.4 Разработка прототипа интерфейса пользователя системы.....	42
2.5 Разработка информационно-логического проекта системы	47
2.5.1 Язык UML	47
2.5.2 Диаграмма вариантов использования.....	48
2.5.3 Сценарии	51
2.5.4 Диаграмма классов	57
2.5.5 Диаграмма состояний.....	59
2.5.6 Диаграмма деятельности	61
2.5.7 Диаграмма последовательности.....	62

2.6	Выбор и обоснование алгоритмов обработки данных.....	63
2.7	Выбор и обоснование комплекса программных средств.....	68
2.7.1	Выбор языка программирования	69
2.7.2	Выбор среды программирования	69
2.7.3	Выбор операционной системы	70
3	Реализация системы.....	71
3.1	Разработка и описание интерфейса пользователя.....	71
3.2	Диаграммы реализации.....	76
3.2.1	Диаграмма компонентов	77
3.2.2	Диаграмма развертывания	78
3.2.3	Диаграмма классов	79
3.3	Выбор и обоснование комплекса технических средств.....	79
3.3.1	Расчет объема занимаемой памяти	79
3.3.2	Минимальные требования, предъявляемые к системе	81
	ЗАКЛЮЧЕНИЕ.....	82
	СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ	83
	ПРИЛОЖЕНИЕ А Руководство пользователя	87
A.1	Назначение системы.....	87
A.2	Условия работы системы	87
A.3	Установка системы.....	88
A.4	Работа с системой.....	88
A.4.1	Авторизация и регистрация в системе.....	88
A.4.1	Работа с системой в режиме администратора.....	90
A.4.2	Работа с системой в режиме пользователя.....	93
	ПРИЛОЖЕНИЕ Б Листинг модулей программы	95

ВВЕДЕНИЕ

Слово «лабиринт» пришло в русский язык из немецкого и является его прямым переводом. Первоисточник этого слова – греческий язык, в котором под лабиринтом понималось обширное пространство, состоящее из многочисленных комнат, коридоров и переходов, расположенных по запутанному плану, составленному с целью запутать незнающего человека. Другим значением этого слова считалось подземелье – природные подземные ходы, в которых можно было легко запутаться и потеряться [1].

Лабиринты являются частью культуры разных стран и эпох. Например, при разговоре об античной эпохе слово «лабиринт» ассоциируется с островом Крит и Минотавром. В Древнем Египте лабиринты строились в центре городов и использовались фараонами для управления страной и в религиозных целях. В Индии и Китае считалось, что лабиринты защищают от злых духов, поэтому входы в дома и города обязательно строили в виде лабиринтов.

В современном мире лабиринты имеют более развлекательных характер. Их используют в декоративных целях в ландшафтном дизайне, для тренировки мозга в компьютерных и настольных играх. Исследования показывают, что прохождение лабиринтов учит детей ориентироваться в пространстве и развивает память [2].

Во многих компьютерных играх встречаются лабиринты. Они могут быть как отдельным уровнем в логических играх, так и частью мира в приключенческих. Получается, что пользователя компьютера в работе с лабиринтами ожидают две основные задачи: генерация лабиринта и его прохождение.

Во время курсового проектирования необходимо разработать автоматизированную систему, с помощью которой можно генерировать лабиринт в автоматическом режиме с помощью специальных алгоритмов, а также находить выход из лабиринта вручную или используя алгоритмы нахождения пути.

Разработка системы будет производиться по технологии быстрой разработки приложений RAD (Rapid Application Development), которая поддерживается методологией структурного проектирования и включает элементы объектно-ориентированного проектирования и анализа предметной области [3].

При проектировании системы будут использоваться методология ООАП (Object-Oriented Analysis/Design), в основу которой положена объектно-ориентированная методология представления предметной области в виде объектов, являющихся экземплярами соответствующих классов, и язык моделирования UML (Unified Modeling Language), который является стандартным инструментом для разработки «чертежей» программного обеспечения [4,5].

1 Описание и анализ предметной области

Предметная область – часть реального мира, которая имеет существенное значение или непосредственное отношение к процессу функционирования программы. Другими словами, предметная область включает в себя только те объекты и взаимосвязи между ними, которые необходимы для описания требований и условий решения конкретной задачи [6].

Она включает объекты, изучаемые теорией, а также свойства, отношения и функции, которые принимаются во внимание в теории.

1.1 Описание предметной области

1.1.1 Описание лабиринта

Лабиринт – структура, состоящая из запутанных путей, ведущих к выходу или в тупик [7].

С математической точки зрения лабиринт представляет собой топологическую задачу. Если у лабиринта имеется только один вход и необходимо найти дорогу к единственному выходу, то такую задачу всегда можно решить. Для этого достаточно, идя по лабиринту, все время одной рукой касаться стенки. Таким образом можно всегда найти выход из лабиринта, но путь не будет кратчайшим.

Тот же метод пригоден и в более традиционном случае, когда цель находится внутри лабиринта. Но только если в нем нет путей, по которым можно кружить вокруг цели и возвращаться в исходную точку. Иначе попасть внутрь "островка", вокруг которого проходит замкнутый маршрут, не удастся.

1.1.2 Классификация лабиринтов

Лабиринты можно разбить по пяти различным классификациям: размерность, происхождение, топология, тесселяция и маршрутизация:

1) размерность определяет количество измерений, в которых лабиринт находится [8]. Двумерный лабиринт расположен в двух измерениях и может быть легко отображен на бумаге, на рисунке 1 приведен пример такого лабиринта.

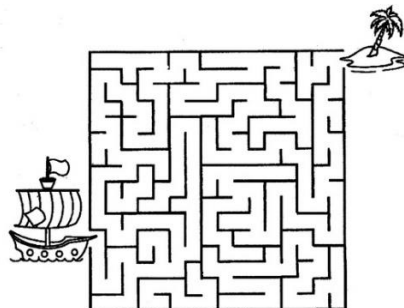


Рисунок 1 – Двумерный лабиринт

Трехмерный лабиринт имеет несколько уровней, проходы могут подниматься вверх и опускаться вниз. Пример трехмерного лабиринта приведен на рисунке 2.

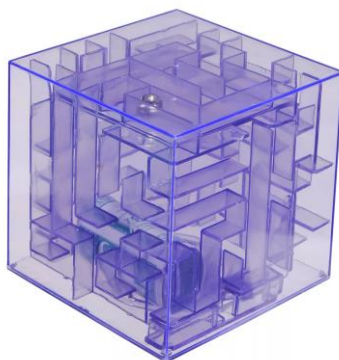


Рисунок 2 – Трехмерный лабиринт

Более высокие размерности лабиринтов трудно смоделировать в реальном мире, но они могут быть представлены математической абстракцией. Например, четырехмерный лабиринт может иметь порталы во времени, что будет интересным элементом компьютерной игры;

2) по происхождению лабиринты могут быть естественными и искусственными. Естественный лабиринт – это природный элемент, созданный либо стихией, либо животными, использующими лабиринт в качестве своего дома. Например, грызуны являются подземными жителями, использующими лабиринты. Они роют себе домики с несколькими выходами, чтобы в случае приближения хищника быстро скрыться из

вида [9]. В роли тупиков в таких лабиринтах выступают «кладовые» комнаты и «детские». На рисунке 3 представлен план нор крота. Примерно так же выглядят норы хомяков, мышей, сурков и других мелких грызунов.

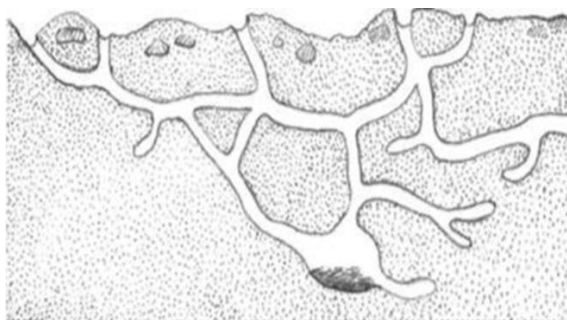


Рисунок 3 – План норы крота

Другим примером естественного лабиринта могут служить муравейники. У них тоже обычно несколько выходов наружу, тупиками являются хранилища ресурсов и комнаты с потомством. План муравейника в разрезе представлен на рисунке 4.

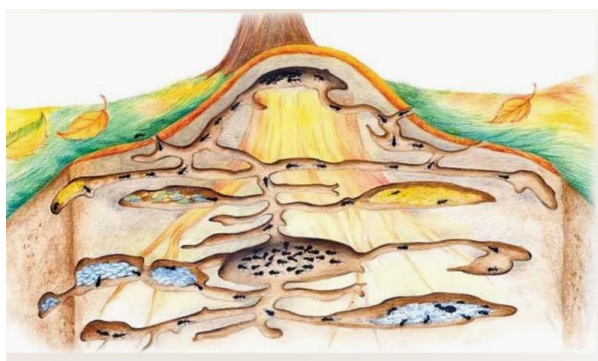


Рисунок 4 – План муравейника

Искусственные лабиринты – лабиринты, созданные человеком в личных целях [9]. К ним можно отнести пещеры, созданные для добычи ресурсов. На рисунке 5 представлен план пещеры в деревне Борцово, которая в XIX веке была одним из основных источников кварца в Санкт-Петербургской губернии.

Другим примером искусственных лабиринтов можно считать садовые лабиринты, созданные для украшения приусадебных участков, парков и площадей. На рисунке 6 показан лабиринт, построенный в 2016 году, для принца Уэльского в Дамфрис-хаусе;

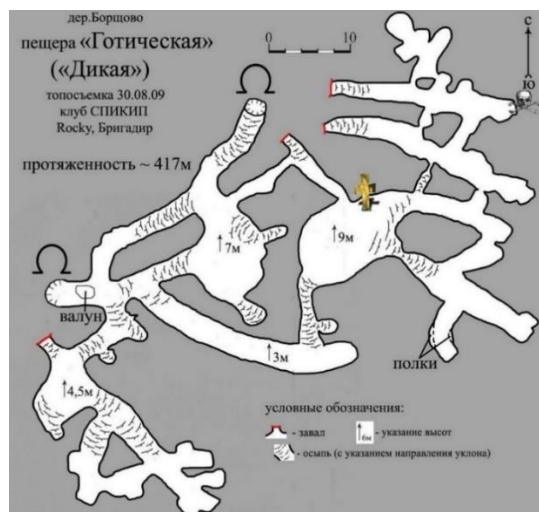


Рисунок 5 – План пещеры

3) класс топологии описывает геометрию пространства, в котором расположен лабиринт. Бывают обычные лабиринты – лабиринты в евклидовом пространстве, и необычные.



Рисунок 6 – Лабиринт в Дамфрис-хаусе

Примерами лабиринтов с необычной топологией могут быть лабиринты на поверхности куба, лабиринты на поверхности ленты Мёбиуса и лабиринты, эквивалентные находящимся на торе, где попарно соединены левая и правая, верхняя и нижняя стороны [8]. Пример лабиринта с необычной топологией представлен на рисунке 7;

4) тесселяция – классификация геометрии отдельных ячеек лабиринта. Существует огромное множество различных форм ячеек лабиринта, среди которых есть как стандартные формы: прямоугольные, треугольные, шестиугольные, так и необычные [8].

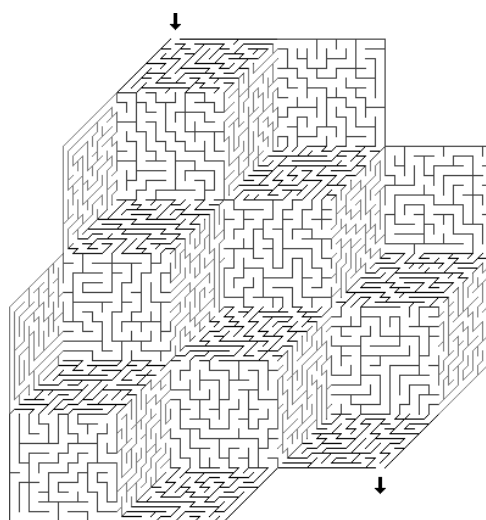


Рисунок 7 – Пример лабиринта с необычной топологией

Дельта-лабиринты состоят из соединенных треугольников, при этом у каждой ячейки может быть до трех соединенных с ней проходов. Пример такого лабиринта представлен на рисунке 8.

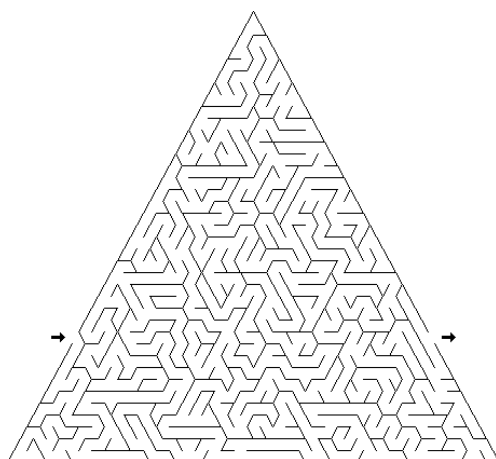


Рисунок 8 – Пример дельта-лабиринта

Гексагональные лабиринты составлены из шестиугольников, и у каждой ячейки может быть не более пяти проходов. Пример такого лабиринта представлен на рисунке 9.



Рисунок 9 – Пример гексагонального лабиринта

Тета-лабиринты состоят из концентрических окружностей проходов, в которых начало или конец находится в центре, а другой - на внешнем крае. Пример такого лабиринта представлен на рисунке 10.



Рисунок 10 – Пример тета-лабиринта

К необычной тесселяции можно отнести дзета-лабиринт, расположенный на прямоугольной сетке, но у которого помимо прямоугольных соединений ячеек, есть диагональные проходы под углом 45 градусов. Пример такого лабиринта представлен на рисунке 11.

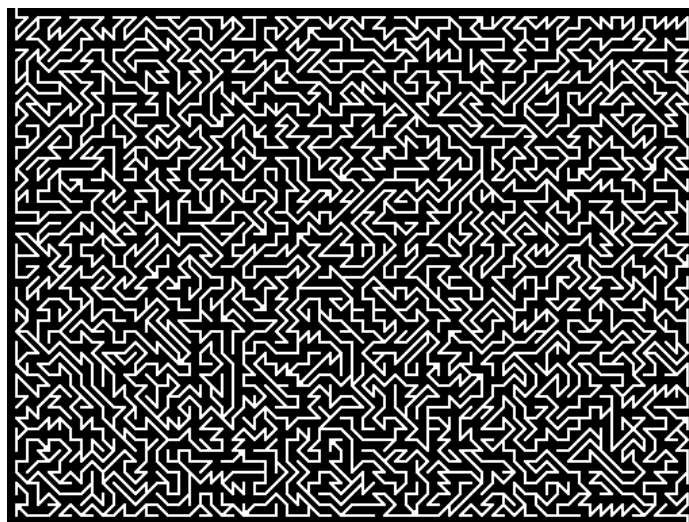


Рисунок 11 – Пример дзета-лабиринта

Фрактальный лабиринт – лабиринт, составленный из лабиринтов, в каждой его ячейке расположен лабиринт меньшего размера. Содержимое копирует себя, создавая бесконечно большой лабиринт. Пример такого лабиринта представлен на рисунке 12;

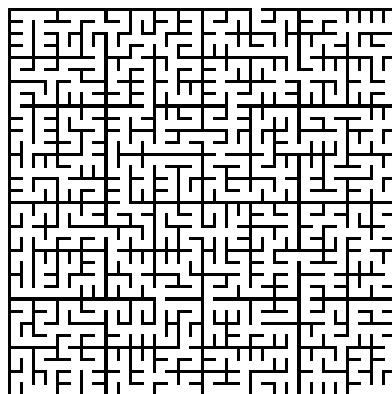


Рисунок 12 – Пример фрактального лабиринта

5) классификация по маршрутизации связана с типами проходов лабиринта в пределах его геометрии. Лабиринт с одиночным соединением, он же «идеальный» лабиринт – это лабиринт, в котором ровно один путь к любой другой точке. В нем нет петель, замкнутых цепей, недостижимых областей. Существует только одно решение такого лабиринта [8]. Пример «идеального» лабиринта приведен на рисунке 13.

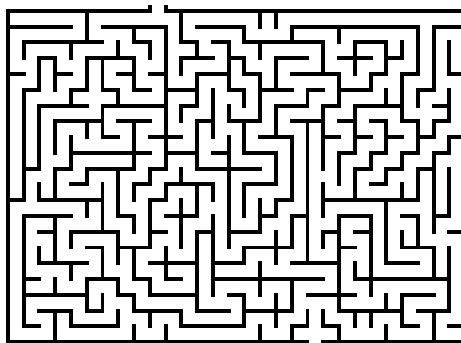


Рисунок 13 – Пример «идеального» лабиринта

Лабиринт с многократными соединениями или «плетеный» лабиринт – это лабиринт, в котором не тупиков. Он состоит из проходов, замыкающих и возвращающихся друг к другу, создавая петли вместо конечных точек. Пример приведен на рисунке 14.

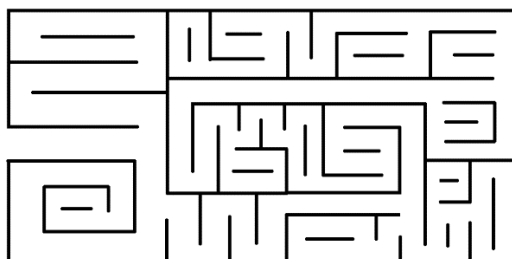


Рисунок 14 – Пример «плетеного» лабиринта

Одномаршрутный лабиринт – лабиринт без развилок, в котором есть только один длинный извивающийся проход, меняющий направление. Потеряться в таком лабиринте можно только от невнимательности или усталости. Пример одномаршрутного лабиринта приведен на рисунке 15.

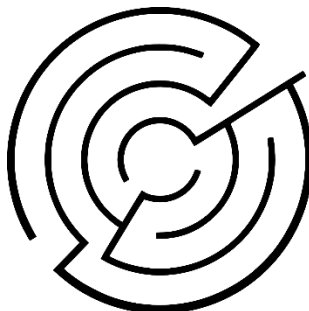


Рисунок 15 – Одномаршрутный лабиринт

1.1.3 Описание алгоритмов генерации лабиринта

При создании лабиринтов требуется применять общие правила: не должно быть замкнутых контуров или петель, изолированных от других частей лабиринта областей, количество входов и выходов для неидеальных лабиринтов должно быть определено. Для генерации лабиринтов существуют множество алгоритмов [10]. Рассмотрим самые популярные из них:

- алгоритм Эллера основан на построчной генерации, где между каждыми двумя клетками строки при определенных условиях случайным образом возникала стенка. Данный метод гарантирует отсутствие циклов и недоступных клеток посредством выполнения условий. В конце все клетки окажутся «в одном множестве», что будет означать, что между каждыми двумя клетками существует путь;

- алгоритм на основе двоичных деревьев. Для каждой ячейки существует проход вверх или влево, но никогда не в обоих направлениях. В версии с добавлением стен для каждой вершины добавляется сегмент стены, ведущий вниз или вправо, но не в обоих направлениях. Каждая ячейка независима от всех других ячеек, за исключением нужды при её создании проверять состояние каких-то других ячеек;

- алгоритм рекурсивного деления начинается с установления случайной горизонтальной или вертикальной стены, пересекающей доступную область в случайной строке или столбце. Потом вдоль нее случайно размещаются пустые места. Этот процесс рекурсивно повторяется для двух подобластей, сгенерированных разделяющей стеной. Для наилучших результатов нужно добавить отклонение в выборе горизонтали или вертикали на основе пропорций области. Например, область, ширина которой вдвое больше высоты, должна более часто делиться вертикальными стенами;

- алгоритм выращивания леса отсортировывает множество ячеек случайным образом в список «новых». Одна или несколько ячеек перемещаются из списка «новых» в список «активных». Далее выбирается ячейка из «активного» списка и вырезается проход в соседнюю несозданную ячейку из «нового» списка, добавляя новую ячейку в список «активных» и объединяя множества двух ячеек. Лабиринт завершён, когда список «активных» становится пустым;

- алгоритм Краскала основан на том, что каждые ячейки помечаются уникальным идентификатором, а затем производится обход всех рёбер в случайном порядке. Если ячейки с обеих сторон от каждого ребра имеют разные идентификаторы, то удаляем стену и задаём всем ячейкам с одной стороны тот же идентификатор, что и ячейкам с другой. Если ячейки на обеих сторонах стены уже имеют одинаковый идентификатор, то между ними уже существует какой-то путь, поэтому стену можно оставить, чтобы не создавать петлю.

В разрабатываемой системе будут использованы следующие алгоритмы:

- алгоритм Олдос-Бродера основан на том, что в сетке выбирается точка и случайно перемещаемся в соседнюю ячейку. При условии, что осуществилось попадание в не вырезанную ячейку, в неё вырезается проход из предыдущей ячейки. Таким образом продолжается движение в соседние

ячейки, пока во все ячейки не появится хотя бы один проход. Этот алгоритм создаёт лабиринты с низким показателем текучести;

- алгоритм Эйлера основан на построчной генерации. Между каждыми двумя клетками строки при определенных условиях, чтобы не было циклов и недоступных клеток, случайным образом возникала стенка. При этом в конце все клетки окажутся «в одном множестве», что будет означать, что между каждыми двумя клетками существует путь.

1.1.4 Описание алгоритмов нахождения пути

Существует множество способов решения лабиринтов, и каждый из них имеет собственные характеристики [10]. Рассмотрим некоторые из них.

- следование по одной руке основано на том, чтобы идти по проходам и при достижении развилки всегда поворачивать направо (или всегда налево). Чтобы применить такое решение лабиринта в реальном мире, нужно положить руку на правую (или левую) стену и постоянно держать её на стене в процессе прохождения лабиринта. При желании можно помечать уже посещённые ячейки и ячейки, посещённые дважды. В конце можно вернуться назад по решению, следуя только по ячейкам, посещённым один раз. Этот метод обязательно найдёт кратчайшее решение, и он совершенно не работает, если цель находится в центре лабиринта и его окружает замкнутая цепь;

- алгоритм цепей начинается с указания нужных мест начала и конца, и тогда алгоритм всегда путь от начала до конца, если он существует. При этом решение склонно быть разумно коротким, если даже не кратчайшим. Это означает, что таким способом нельзя решать лабиринты, в которых неизвестно точное расположение конца;

- волновой алгоритм основан на идее распространения волны от исходной точки в разные стороны. Начальное значение волны – ноль. То есть ближайшие точки, в которые можно пойти, например, верх, низ, левая и правая, и которые еще не затронуты волной, получают значение волны, а

также некоторый модификатор проходимости этой точки. Чем он больше – тем медленнее преодоление данного участка. Значение волны увеличивается на один. Аналогично обрабатываются клетки, отходя от тех, на которой значение волны – 2. При этом на клетках с худшей проходимостью волна задержится. И так дальше все обрабатывается, пока не достигнута конечная точка маршрута. Сам путь в получившемся массиве значений волны вычисляется по наименьшим клеткам. Пример работы алгоритма приведен на рисунке 16.

1.2 Описание систем-аналогов

Сравнительный анализ аналогичных систем – важный шаг в процессе разработки нового продукта.



Рисунок 16 - Пример работы волнового алгоритма

Он позволяет не только понять сильные и слабые стороны конкурентов, но и определить направления для улучшений и внедрения новых идей. Такой анализ включает несколько ключевых аспектов:

- изучение механик и структуры аналогичных лабиринтов для выявления интересных решений;
- оценка визуальных элементов для анализа стилистических подходов, которые привлекают пользователей;
- сравнение уровней взаимодействия с игроком и степени сложности в разных играх, что помогает определить оптимальные настройки для удержания интереса пользователей.

1.2.1 Gotcha

На рынке видеоигр представлено колоссальное количество игр в жанре аркада для любителей интенсивного игрового процесса. Видеоигра «Gotcha» была разработана Алланом Олкорном, дизайнером Pong, а прототип был сконструирован Cyan Engineering, полунезависимой исследовательской дочерней компанией Atari. Игра внесла значительный вклад в жанр, будучи одной из первых игроков на рынке [11].

«Gotcha» – это игра в лабиринт для двух игроков, в которой один игрок пытается поймать другого. Лабиринт состоит из повторяющегося набора элементов, расположенных в несколько столбцов на экране. "Преследователь" представлен квадратом, а "Преследуемый" обозначен знаком плюс. По мере приближения Преследователя к преследуемому электронный звуковой сигнал воспроизводится с возрастающей частотой, пока Преследователь не достигнет Преследуемого.

На рисунке 17 приведена главная экранная форма программы «Gotcha», на которой продемонстрирован интерфейс игры.



Рисунок 17 – Игровой процесс Gotcha

К достоинствам данной системы можно отнести постоянную генерацию нового лабиринта с целью усложнения игрового процесса для игроков.

К недостаткам системы относятся:

- некоторые случайно сгенерированные лабиринты могут быть неравномерно сложными, что может привести к неравному опыту игры для

разных игроков;

- на этапе настроек не предусмотрено регулирование толщины стенок;
- скромное оформление, которое со временем начинает приедаться глазу и надоедать уже во время игрового процесса.

1.2.2 Онлайн генератор лабиринтов «Plottersvg»

В сети Интернет можно найти огромное количество реализаций генерации лабиринтов, которые могут значительно отличаться по функциональности и сложности. Эти реализации варьируются от простых веб-инструментов для создания лабиринтов до сложных программных решений и библиотек, используемых для научных исследований и разработки игр.

Некоторые системы предназначены для демонстрации работы алгоритмов генерации и решения лабиринтов, в то время как другие фокусируются на предоставлении пользователям гибких инструментов для создания уникальных лабиринтов для различных целей.

Рассмотрим один из таких инструментов на сайте [12]. Интерфейс программы, набор ее функций и основные настройки представлены на рисунке 18.

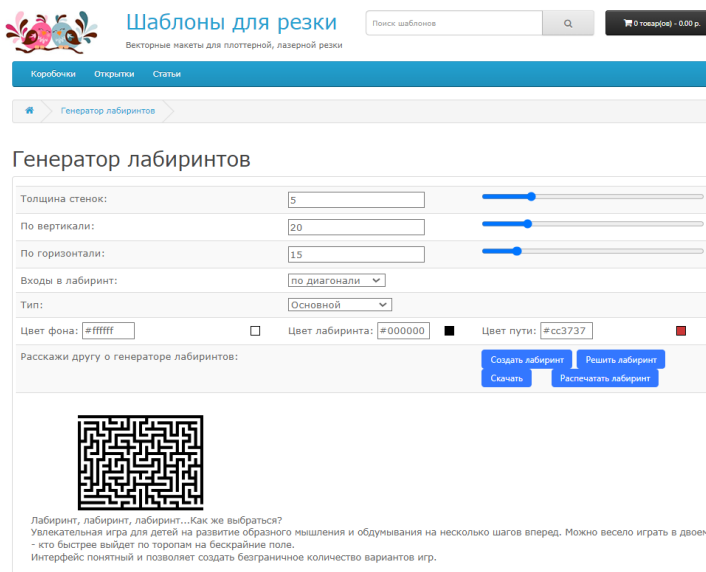


Рисунок 18 – Генератор лабиринтов «Plottersvg»

К достоинствам данной системы относятся:

- регулирование толщины стенок и размера лабиринта по вертикали и горизонтали;
- три режима расположения входов в лабиринт: по диагонали, слева и справа, сверху и снизу;
- возможность изменения цвета фона, пути и стенок лабиринта;
- возможность сохранения и печати лабиринта.

К недостаткам системы относятся:

- на сайте не представлена информация об алгоритмах, используемых при генерации лабиринта и нахождении пути;
- нет возможности ручной расстановки входов и выходов;
- не предусмотрено ручное прохождение и составление лабиринта.

На основании анализа возможностей систем-аналогов были сформулированы требования к разрабатываемой системе (см. таблицу 1).

1.3 Диаграмма объектов предметной области

Объектно-ориентированный анализ и проектирование (ООАП, Object-Oriented Analysis/Design) – технология разработки программных систем. В основу положена объектно-ориентированная методология для представления предметной области в виде объектов, которые являются экземплярами соответствующих классов.

Общие принципы моделирования сложных систем и особенности процесса объектно-ориентированного анализа и проектирования ложатся в основы конструктивного использования языка UML.

Выбор выразительных средств для построения моделей сложных систем определяет и перечисляет те задачи, которые могут быть решены с использованием данных моделей.

Принцип абстрагирования является одним из основных принципов построения моделей сложных систем.

Таблица 1 – Сравнительные характеристики систем-аналогов

Название системы Название показателя	«Gotcha»	Онлайн генератор лабиринтов «Plottersvg»	Разрабатываемая система
Возможность автоматической генерации лабиринта	+	+	+
Автоматическое прохождение лабиринта	+	+	+
Ручное прохождение лабиринта	–	–	+
Оформление по тематике	–	–	+
Выбор размера лабиринта	–	+	+
Визуализация прохождения лабиринта	+	–	+
Возможность выбора алгоритма генерации лабиринта	–	–	+
Возможность выбора алгоритма нахождения пути из лабиринта	–	–	+

Он предписывает включать в модель только те аспекты проектируемой системы, которые имеют свое целевое предназначение к выполнению системой своих функций. При этом все второстепенные детали опускаются, чтобы чрезмерно не усложнять процесс анализа и исследования полученной модели. На финальном этапе получается модель, состоящая из разных объектов с более простым поведением, что упрощает разработку [8].

Объект – элемент системы, в котором хранится перечень всех данных о себе и причисленных операций, производимых над ним. Обладает также основными свойствами объектно-ориентированного программирования.

Диаграмма объектов является статической составляющей взаимодействующих между собой объектов, она должна включить в себя только те объекты предметной области, которые потом преобразуются в диаграмму классов.

Связи между объектами показывают отношения между ними, при необходимости в диаграмме можно перечислить свойства объектов. Диаграмма способна отразить нам пример того, как структуры данных будут выглядеть в определенный момент времени [9].

Лабиринт представляет собой совокупность следующих объектов: клетки, которые могут быть стеной или проходом, персонаж, который движется по проходам.

На рисунке 19 приведена диаграмма объектов предметной области.

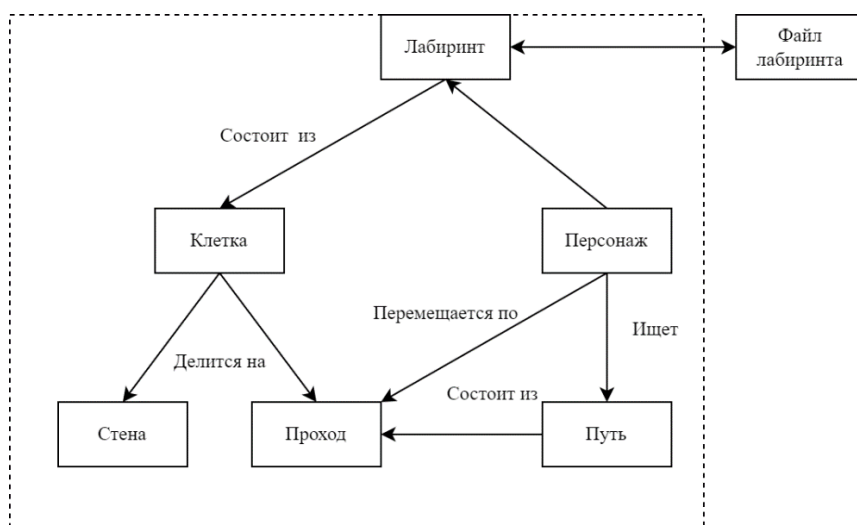


Рисунок 19 – Диаграмма объектов предметной области

1.4 Постановка задачи

Во время курсового проектирования необходимо разработать автоматизированную систему генерирования структуры лабиринта различными способами, а также нахождение выхода из него в автоматическом режиме в соответствии с заданными алгоритмами. Система должна быть реализована в виде настольного приложения.

В системе должно быть реализовано две роли пользователей: администратор и игрок. Каждый пользователь должен авторизоваться в системе: ввести логин и пароль (длина логина должна быть от 6 до 16 символов, длина пароля – от 6 до 20 символов). Система должна проверить учетные записи и, в случае успеха, настроить интерфейс пользователя на заданную роль, в противном случае выдать сообщение об ошибке.

1.4.1 Режим администратора

Основная задача администратора – создание лабиринта. Администратор должен предварительно задать параметры лабиринта: определить его размеры по горизонтали и вертикали (не менее 7 и не более 21), выбрать способ расстановки входа и выхода (автоматический или ручной), выбрать алгоритм генерации лабиринта (Олдоса-Бродера или Эйлера) и тему оформления. В системе должно быть реализовано 4 темы оформления (весна, лето, осень, зима). Лабиринт будет создаваться в автоматическом или ручном режиме и сохраняться в файл заданной структуры. Лабиринт должен быть толстостенным, вход и выход находятся на периметре лабиринта, но не в углах, а также вход и выход не совпадают друг с другом, стены должны образовывать тупики и пути, не должны образовывать изолированные части.

При этом система должна провести проверку корректности введенных параметров, и, в случае ошибки, выдать предупредительное сообщение с возможностью повторного ввода значений. В системе должен осуществляться контроль типов и диапазонов значений параметров. Система

также должна обеспечивать контроль целостности лабиринта, предотвращая создание замкнутых или непроходимых участков.

1.4.2 Режим игрока

Перед началом игры пользователь может загрузить один из предложенных лабиринтов из файла, а также выбрать тему оформления лабиринта. Пользователю с ролью игрок должен быть доступен выбор способа прохождения лабиринта (ручной и автоматический).

Если будет выбран автоматический режим прохождения лабиринта игроку будет предоставлен выбор алгоритма, с помощью которого нужно пройти лабиринт (одной руки или волновой). Помимо этого, будет дана возможность выбрать режим задержки перемещения персонажа, а также при автоматическом прохождении лабиринта будет присутствовать визуализация нахождения выхода из лабиринта.

Если будет выбран ручной режим прохождения лабиринта, тогда у пользователя должна быть возможность перемещения при помощи стандартных действий для нахождения выхода из лабиринта (вперёд, назад, вправо, влево). В программе предусмотрена возможность выдачи и просмотра справочной информации.

Таким образом, система должна решать следующие задачи:

1) функции системы:

- аутентификация пользователя в системе, настройка интерфейса пользователя на заданную роль;
- генерирование шаблона лабиринта по заданному размеру;
- случайная расстановка входа и выхода;
- контроль расстановки входа и выхода;
- создание структуры лабиринта по заданному алгоритму;
- проверка структуры лабиринта при загрузке;
- визуализация генерации лабиринта;
- визуализация прохождения лабиринта (видимая дорожка);

- выдача справочной информации;

2) функции администратора:

- авторизация пользователя в системе (ввод логина и пароля);
- настройка параметров лабиринта при создании:
 - 1) задание размера по горизонтали;
 - 2) задание размера по вертикали;
 - 3) выбор способа расстановки входа и выхода;
 - 4) выбор алгоритма генерации лабиринта;
 - 5) настройки темы оформления;
- ручная расстановка входа и выхода;
- сохранение лабиринта в файл заданной структуры;
- просмотр лабиринта;
- просмотр справочной информации;

3) функции игрока:

- регистрация пользователя в системе (ввод логина и пароля);
- авторизация пользователя в системе (ввод логина и пароля);
- выбор лабиринта;
- загрузка лабиринта из файла;
- изменение темы оформления лабиринта;
- выбор алгоритма прохождения лабиринта;
- ручное прохождение лабиринта;
- выбор алгоритма прохождения лабиринта;
- выбор режима прохождения лабиринта;
- выбор задержки перемещения персонажа (для автоматического режима);
- запуск алгоритма;
- просмотр справочной информации.

2 Проектирование системы

Система – множество элементов, находящихся в отношениях или связях друг с другом, образующее целостность или органическое единство [15].

Проектирование – процесс определения архитектуры, компонентов, интерфейсов и других характеристик системы или её части. Результатом проектирования является проект – целостная совокупность моделей, свойств или характеристик, описанных в форме, пригодной для реализации системы [16].

Проектирование является частью стадии жизненного цикла системы, называемой определением системы. Результаты этой стадии являются входной информацией для стадии реализации системы.

Проектирование системы направлено на представление системы, соответствующее предусмотренной цели, принципам и замыслам; оно включает оценку и принятие решений по выбору таких компонентов системы, которые отвечают её архитектуре и укладываются в предписанные ограничения [16].

На этом этапе решаются задачи определения архитектуры приложения, его функций, требований к функционалу и интерфейсу, составляется структурная схема системы. Разрабатывается спецификация требований программного обеспечения (ПО), прототип интерфейса пользователя системы и информационно-логический проект системы, разрабатываются алгоритмы обработки данных.

2.1 Выбор и обоснование архитектуры системы

Архитектура – это базовая организация системы, которая описывает связи между компонентами этой системы и внешней средой, а также определяет принципы её проектирования и развития [17].

Существует множество архитектурных подходов к разработке ПО, рассмотрим наиболее распространенные из них:

1) монолитная архитектура. Это классический подход к разработке программного обеспечения. Все компоненты, модули и функции находятся в одном целом и работают в тесном взаимодействии друг с другом. Такой подход обеспечивает простоту и легкость в разработке, но может стать проблемой при масштабировании и поддержке продукта [18];

2) клиент-серверная архитектура. Этот подход разделяет программу на две основные части: клиентскую и серверную. Клиентская часть отвечает за интерфейс пользователя и взаимодействие с сервером, а серверная часть выполняет основные вычисления, отвечает за обработку и хранение данных. Такой подход позволяет эффективно использовать ресурсы и легко масштабировать систему [18];

3) распределенная архитектура. Такая архитектура предполагает, что компоненты программного обеспечения размещены на различных узлах сети, которые взаимодействуют друг с другом. Этот подход обеспечивает высокую отказоустойчивость, масштабируемость и производительность [18];

4) сервис-ориентированная архитектура. Это подход, при котором программное обеспечение разрабатывается в виде набора служб, которые могут быть использованы другими приложениями. Это позволяет создавать гибкие и масштабируемые системы, где каждая служба выполняет конкретную функцию и может быть использована повторно [18];

5) микросервисная архитектура. Это подход к разработке системы, при котором обеспечивается разделение ее компонентов на автономные составляющие – микросервисы. Они могут иметь собственную кодовую базу, обособленные базы данных, но при этом логически связаны друг с другом. Такая архитектура позволяет обновлять отдельные микросервисы без влияния на другие службы, что увеличивает гибкость и масштабируемость системы, но тем самым усложняет ее [19].

Разрабатываемая система будет обладать архитектурой монолитного настольного приложения по следующим причинам:

- популярность и распространенность подхода;

- высокая производительность, благодаря доступу к ресурсам машины, на которой установлено ПО, и архитектурному подходу;
- моментальная отзывчивость интерфейса программы к действиям пользователя;
- возможность работать в режиме оффлайн, что позволяет пользователям продолжать использовать приложение даже при отсутствии соединения с интернетом;
- повышенная безопасность данных, так как все необходимые файлы находятся исключительно на компьютере с ПО.

2.2 Структурная схема системы

На этапе построения структурной схемы система разделяется по функциональному признаку на основные подсистемы, между которыми указываются информационные связи и связи по управлению, описывается основное назначение подсистем.

При разработке структурной схемы используется методология структурного проектирования, в основе которой лежит алгоритмическая декомпозиция и иерархия вида «часть-целое», учитывающая, что внутренние связи элементов внутри подсистем сильнее, чем связь между подсистемами. Декомпозиция системы может повторяться многократно, вплоть до уровня конкретных процедур, при этом должна быть обеспечена целостность системы, а все составляющие компоненты взаимоувязаны.

Структурная схема необходима для того, чтобы наглядно увидеть, как спроектирована система. На структурной схеме указывается связь между подсистемами.

На рисунке 20 приведена структурная схема разрабатываемой системы, в ее состав входят следующие подсистемы:

- 1) подсистема аутентификации, которая отвечает за регистрацию нового пользователя и проверку подлинности введенных данных существующим пользователем;

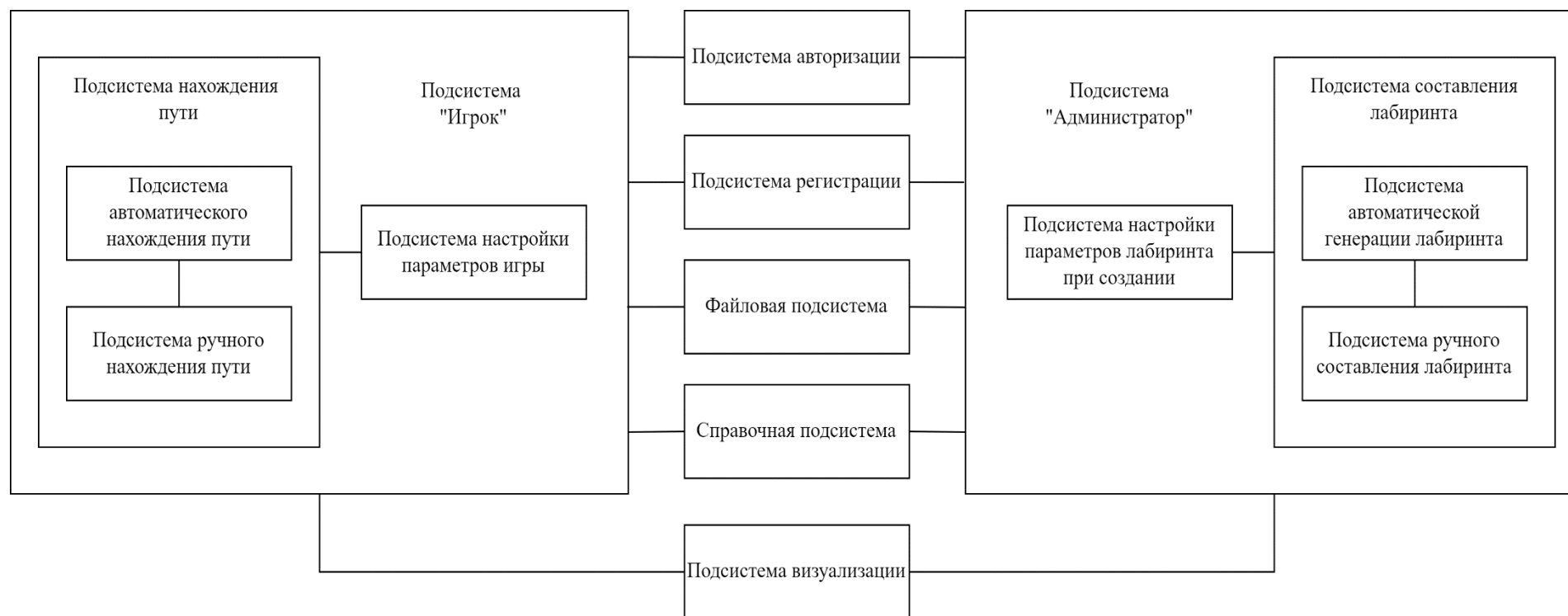


Рисунок 20 – Структурная схема системы

2) подсистема авторизации, которая отвечает за ввод учетных данных пользователя;

3) файловая система, которая отправляет запросы к файлам записи и получает ответы;

4) справочная подсистема, которая отвечает за выдачу сведений о системе и ее разработчиках;

5) подсистема визуализации, которая отвечает за графическое представление лабиринта и отображение выхода из него;

6) подсистема «Администратор», которая включает в себя следующие подсистемы:

- подсистему настройки параметров лабиринта при создании;

- подсистему создания лабиринта, которая включает в себя подсистему автоматической генерации лабиринта и подсистему ручной генерации лабиринта;

7) подсистема «Игрок», которая включает в себя следующие подсистемы:

- подсистему настроек параметров игры, которая отвечает за выбор способа прохождения и создания лабиринта, выбора темы оформления;

- подсистему нахождения пути, которая включает в себя подсистему автоматического нахождения пути и подсистему ручного нахождения пути.

2.3 Разработка спецификации требований

Спецификация требований к программному обеспечению (Software Requirements Specification, SRS) – это описание программной системы, которую необходимо разработать. Она моделируется по образцу спецификации бизнес-требований. Спецификация требований к программному обеспечению устанавливает функциональные и нефункциональные требования, может включать набор вариантов использования пользователем. Правильная спецификация помогает

предотвратить сбои программного обеспечения, получить четкое представление о разрабатываемом продукте [20].

Документ по программному обеспечению создает основу для соглашения между заказчиками и подрядчиками или поставщиками о том, как должен функционировать программный продукт (в рыночно ориентированном проекте эти роли могут играть отделы маркетинга и разработки) [20].

Спецификация требований к программному обеспечению – это строгая оценка требований перед более конкретными шагами проектирования системы. Ее цель – уменьшить количество последующих итераций, обеспечение реалистичной основы для оценки стоимости продукта, рисков и графиков. При правильном использовании спецификации требований к программному обеспечению могут помочь предотвратить провал программного проекта [20].

Документ спецификации требований к программному обеспечению содержит перечень требований, достаточных для разработки проекта. Чтобы вывести требования, разработчик должен иметь четкое и полное понимание разрабатываемых продуктов. Это достигается путем детального и непрерывного взаимодействия с проектной командой и заказчиком на протяжении всего процесса разработки программного обеспечения.

2.3.1 Функциональная спецификация

Функциональные требования – это документ, который понятно и точно описывает существенные технические требования для объектов, материалов или операций. Спецификации помогают устранить дублирование и несоответствия, позволяют точно оценить необходимые действия и ресурсы, выступают в качестве согласующего и справочного документов о внесённых изменениях и предоставляют документацию с конфигурацией. Они дают точное представление о решении проблемы, повышая эффективность разработки системы и оценивая стоимость альтернативных путей

проектирования, служат указанием для испытателей для верификации (качественной оценки) каждого технического требования [21].

Функциональная спецификация не определяет операции, происходящие внутри данной системы и каким образом будет реализована её функция. Вместо этого, она рассматривает взаимодействие с внешними агентами (например, персонал, использующий программное обеспечение; периферийные устройства компьютера или другие компьютеры), которые могут «следить», взаимодействуя с системой.

Функциональная спецификация системы приведена в таблице 2.

2.3.2 Перечень исключительных ситуаций

Исключительная ситуация – это ситуация, при которой система не может выполнить возложенных на нее функций или которая может привести к денормализации работы системы [22].

В таблице 3 приведен перечень исключительных ситуаций для разрабатываемой системы и описаны реакции системы на их возникновение.

Таблица 2 – Перечень исключительных ситуаций

Название подсистемы	Название исключительной ситуации	Реакция системы
1 Справочная	1.1 Невозможно открыть файл справки	Выдача сообщения «Файл справки поврежден»
	1.2 Невозможно найти файл справки	Выдача сообщения «Отсутствует файл справки»
2 Файловая	2.1 Попытка открытия файла с несобственным форматом	Выдача сообщения «Файл поврежден или недопустимого формата»
	2.2 Файл с заданным именем не существует	Выдача сообщения «Файл с заданным именем не существует»

Таблица 3 – Перечень функций, выполняемых системой

Название подсистемы	Название функции	Информационная среда			
		Входные данные		Выходные данные	
		Назначение (наименование)	Тип, ограничения	Назначение (наименование)	Тип, ограничения
1	2	3	4	5	6
Настройки параметров игры	Выбрать режим прохождения	Список режимов прохождения	Ручной, автоматический	Текущий режим	Перечисление
	Выбрать тему лабиринта	Список тем лабиринта	Зима, весна, лето, осень	Текущая тема	Перечисление
	Выбрать алгоритм прохождения	Список алгоритмов	Волновой, одной руки	Текущий алгоритм прохождения	Перечисление
	Выбрать скорость прохождения	Режимы скорости	Целое 0..4	Текущая скорость	Целое
Настройки параметров лабиринта	Задать высоту лабиринта	Допустимый диапазон значений	Целое 7..21	Текущая высота лабиринта	Целое, нечетное
				Код ошибки	Целое
	Задать ширину лабиринта	Допустимый диапазон значений	Целое 7..21	Текущая ширина лабиринта	Целое, нечетное
				Код ошибки	Целое

Продолжение таблицы 2

1	2	3	4	5	6
Настройки параметров лабиринта	Создать шаблон лабиринта	Текущая высота	Целые	Лабиринт	Объект «Лабиринт»
		Текущая ширина	Целые		
	Выбрать способ расстановки входа и выхода	Список вариантов расстановки	Ручной, автоматический	Текущий алгоритм расстановки	Целое
	Поставить вход	Лабиринт	Объект «Лабиринт»	Координаты входа	Целые, на периметре, не в углах
	Поставить выход	Лабиринт	Объект «Лабиринт»	Координаты выхода	Целые, на периметре, не в углах
	Выбрать алгоритм генерации лабиринта	Список алгоритмов генерации	Эйлера, Олдоса-Бродера	Текущий алгоритм генерации	Перечисление
	Выбрать тему лабиринта	Список тем лабиринта	Зима, весна, лето, осень	Текущая тема	Перечисление

2.4 Разработка прототипа интерфейса пользователя системы

Интерфейс – это набор инструментов, который позволяет пользователю взаимодействовать с программой. В более широком смысле термин обозначает любые инструменты для соприкосновения между разными системами и сущностями. Часто говорят, что графический интерфейс – это внешний вид сайта, программы или приложения [23].

Интерфейс пользователя является одним из важнейших элементов программы, это та часть программы, которая находится у всех на виду. Недочеты в пользовательском интерфейсе могут серьезно испортить впечатление даже о самых многофункциональных программах.

Прототипирование интерфейсов представляет собой создание ранней, упрощенной версии сайта или приложения с целью тестирования интерфейса на предмет его комфортности и простоты использования для конечного пользователя до процесса разработки. Создание макетов позволяет не только понять, какие проблемы со стороны пользователя могут возникнуть в процессе использования продукта, но и предотвратить их еще на этапе планирования, уменьшить затраты на исправление ошибок еще до начала разработки [24].

Рассмотрим преимущества использования прототипирования:

- проверка концепции: прототипирование может помочь увидеть, как система будет выглядеть и работать в реальности;
- исследование пользовательских потребностей: создание прототипа поможет понять, что пользователи хотят от системы и как эти потребности могут быть удовлетворены;
- тестирование: прототипирование может использоваться для тестирования продукта или системы на ранней стадии разработки, что может помочь выявить проблемы и недостатки до реализации системы;

На рисунке 21 приведен прототип экранной формы для авторизации пользователя.

Рисунок 21 – Прототип экранной формы для авторизации

Пользователю необходимо будет ввести логин и пароль в соответствующие поля, если он уже был ранее зарегистрирован. Далее пользователь необходимо нажать кнопку «Войти» для перехода к основной форме. Если пользователь еще не зарегистрирован, то он сможет нажать на кнопку «Регистрация» для перехода к соответствующей форме

На рисунке 22 приведен прототип экранной формы для регистрации пользователей.

Рисунок 22 – Прототип экранной формы для регистрации

Пользователю необходимо ввести логин и пароль в соответствующие поля. Также пользователю будет необходимо ввести пароль второй раз в специальное поле ввода для проверки совпадения паролей. Затем

пользователь должен будет нажать кнопку «Подтвердить» для перехода к следующему экрану (форме «Авторизация»).

На рисунке 23 приведен прототип основной экранной формы для администратора.

Рисунок 23 – Прототип основной экранной формы для администратора

«О системе» и «О разработчиках» – кнопки, благодаря которым администратор сможет получить соответствующую информацию.

На основном экране администратору предоставляется возможность настройки параметров лабиринта. В специально предназначенных полях администратор должен ввести высоту и ширину лабиринта и нажать на кнопку «ОК». Система должна сгенерировать шаблон лабиринта с установленными размерами, администратор должен выбирать способ расстановки входа и выхода (ручной и автоматический), алгоритм генерации лабиринта (Эйлера и Олдоса-Бродера), а также выбрать одну из предложенных на выбор тем (зима, весна, лето, осень).

При выборе ручной расстановки входа и выхода, на сгенерированном шаблоне можно будет нажать на клетки, расположенные по периметру лабиринта, для выбора их в качестве входа и выхода в соответствующем порядке. После выбора алгоритма станет доступной кнопка «Сгенерировать». После нажатия на данную кнопку в правой верхней части экрана будет отображен лабиринт с учетом всех введенных администратором параметров.

Кнопка «Сохранить» дает возможность сохранить сгенерированный лабиринт в файл.

На рисунке 24 приведен прототип основной экранной формы для игрока.

о системе о разработчиках [X]

Выбор режима прохождения:

☐ Ручной

☐ Автоматический

Тема лабиринта:

Выбор алгоритма прохождения:

☐ Волновой

☐ Одной руки

Выбор скорости перемещения:

0 1 2 3 4

лабиринт

Загрузить

Пройти

Рисунок 24 – Прототип основной экранной формы для игрока

«О системе» и «О разработчиках» – кнопки, благодаря которым игрок сможет получить соответствующую информацию. Кнопка «Загрузить» предназначена для загрузки лабиринта из файла.

На основной экранной форме у игрока будет возможность настроить параметры игры: выбрать режим прохождения лабиринта (ручной и автоматический) и тему лабиринта.

При выборе автоматического прохождения у игрока станет доступно кнопки выбора алгоритма прохождения лабиринта (волновой и одной руки). При автоматическом прохождении игрок сможет выбрать скорость перемещения (от 0 до 4). При нажатии на кнопку «Пройти» игрок сможет начать прохождение. Если выбрана скорость 0, то вместо кнопки «Пройти» появится кнопка «Сделать шаг» и алгоритм будет работать в пошаговом режиме.

На рисунке 25 приведена навигационная модель разрабатываемого приложения.

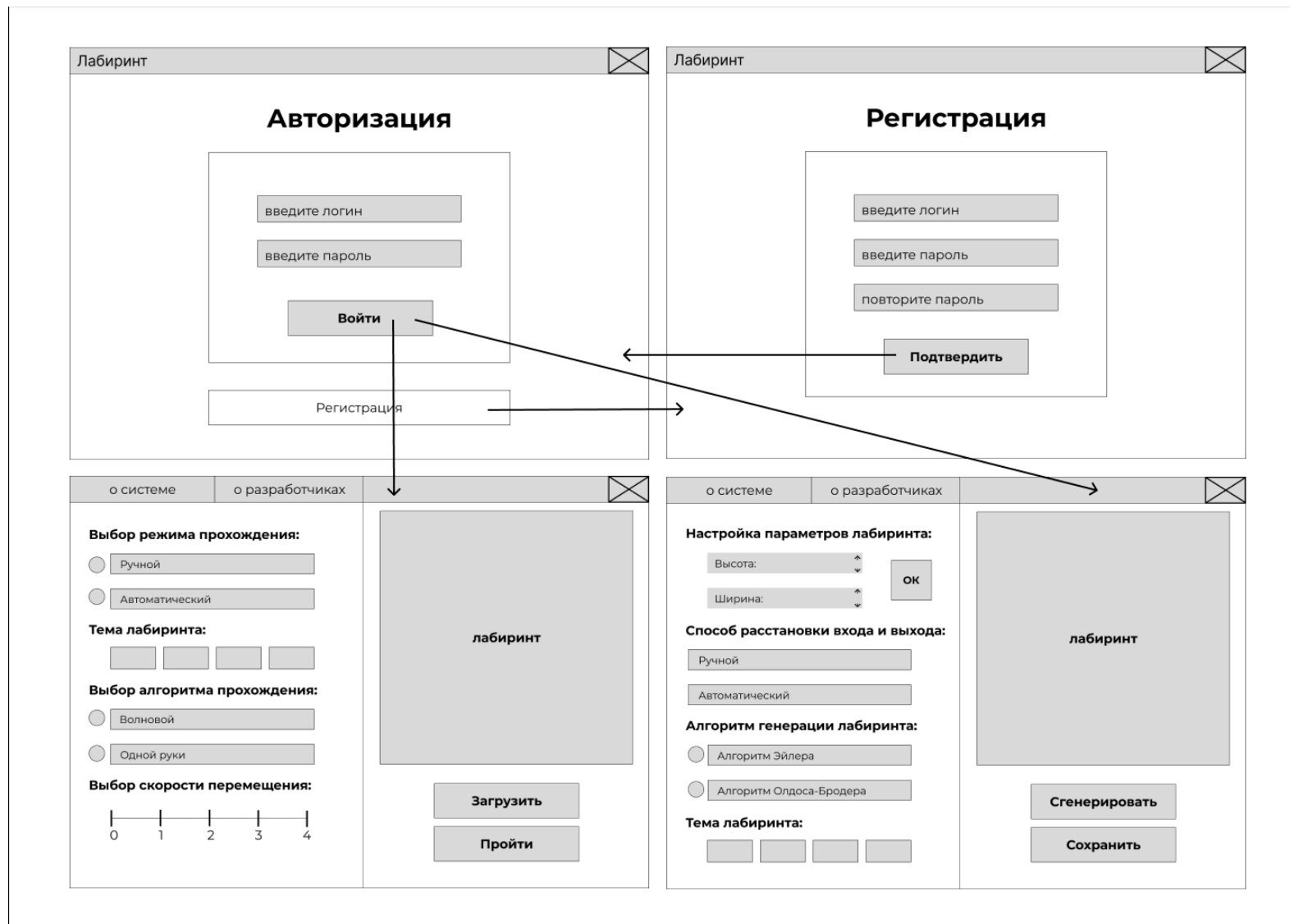


Рисунок 25 – Навигационная модель приложения

2.5 Разработка информационно-логического проекта системы

Информационно-логическая модель – это модель предметной области, которая определяет совокупность информационных объектов, их атрибутов и отношений между объектами, динамику изменений предметной области. Она является схемой, описывающей взаимосвязи функциональных задач, решаемых на предприятии на всех этапах планирования, учёта и управления [26].

Эти модели используют графические представления, показывающие решения как исходной задачи, так и разрабатываемой системы. Как правило, графическое представление более понятно, чем описание требований на естественном языке [27].

Моделирование – это устоявшаяся и повсеместно принятая инженерная методика. Модели являются связующим звеном между процессом анализа и процессом проектирования системы [28].

2.5.3 Язык UML

Для специфицирования (построения точных, недвусмысленных и полных моделей) системы и ее документирования используется унифицированный язык моделирования UML.

Унифицированный язык моделирования (Unified Modeling Language – UML) – это стандартный инструмент для разработки «чертежей» программного обеспечения. Язык UML – это графический язык, описывающий с помощью диаграмм и схем разнообразные процессы и структуры. UML не является языком программирования, но на основании UML-моделей возможна генерация кода [29].

В нотации языка UML определены следующие виды канонических диаграмм:

- диаграмма вариантов использования (use-case diagram);
- диаграмма классов (class diagram);

- диаграмма кооперации (collaboration diagram);
- диаграмма последовательности (sequence diagram);
- диаграмма состояний (statechart diagram);
- диаграмма деятельности (activity diagram);
- диаграмма компонентов (component diagram);
- диаграмма развертывания (deployment diagram).

Рассмотрим для чего при разработке программных систем используют UML:

- для создания «чертежей» программы, схем, которые показывают, как будет устроено программное обеспечение изнутри [30];
- для визуализации уже имеющейся программной структуры. Ряд инструментов позволяет создать UML на основе существующего кода, в таком случае диаграмма сгенерируется автоматически;
- для автоматической генерации кода или технической документации по нему, так как UML поддерживает возможность создания продукта на основе диаграмм;
- для внутренней и внешней коммуникации между сотрудниками, заказчиками и другими: картинки и диаграммы UML понятнее людям, чем текстовые описания.

2.5.4 Диаграмма вариантов использования

Диаграмма вариантов использования представляет собой наиболее общую концептуальную модель сложной системы, которая является исходной для построения всех остальных диаграмм. На ней изображаются отношения между актерами и вариантами использования.

Диаграмма вариантов использования создают, чтобы:

- определить общие границы и контекст моделируемой предметной области на начальных этапах проектирования системы [31];
- сформулировать общие требования к функциональному поведению проектируемой системы [31];

- разработать исходную концептуальную модель системы для её последующей детализации в форме логических и физических моделей [31];
- подготовить исходную документацию для взаимодействия разработчиков системы с её заказчиками и пользователями [31].

Основные части диаграммы вариантов использования:

- актер – это пользователь, который взаимодействует с системой. Это может быть человек, организация или даже сама система;
- варианты использования – овалы горизонтальной формы, обозначающие различные варианты использования, которые может иметь пользователь;
- ассоциация – линия между актёрами и вариантами использования;
- граничные рамки системы – блок, который устанавливает область действия системы для вариантов использования;
- отношения: включение, расширение и обобщение.

На рисунке 26 приведена диаграмма вариантов использования.

В область возможностей актёра «Пользователь» входят следующие функции:

- вход в систему. Такие процедуры, как регистрация и авторизация, являются обязательными для любого пользователя. Главными условиями для регистрации являются ввод логина, ввод пароля и повторение ввода пароля. Для входа в систему пользователю требуется ввести в соответствующие поля логин и пароль;
- просмотр сведений о системе;
- просмотр сведений о разработчиках.

После авторизации в зависимости от введённых данных система предоставляет доступ к функциям и выборам либо для актёра «Администратор», либо для актёра «Игрок», которые в свою очередь являются частными случаями актёра «Пользователь».

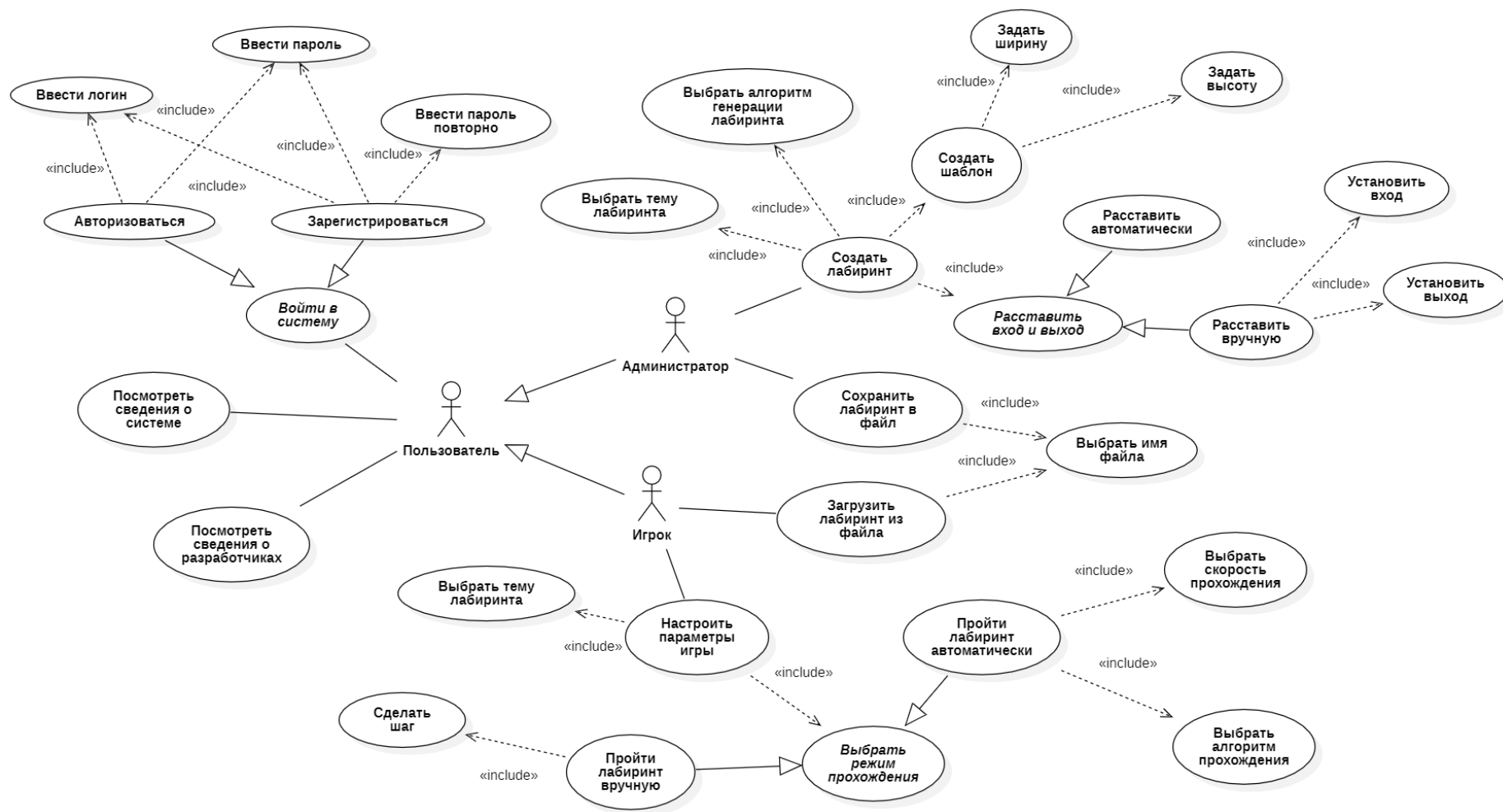


Рисунок 26 – Диаграмма вариантов использования

Актёру «Администратор» предоставляется доступ к:

- сохранению лабиринта в файле. Перед сохранением требуется задать имя файлу;

- созданию лабиринта. Для создания лабиринта есть перечень обязательных условий, которые необходимо учесть для достижения финального результата, а именно:

- 1) расставление входа и выхода. Актёру «Администратор» предоставляется выбор для расставления входов и выходов в лабиринте в режимах вручную или в режиме автоматически;

- 2) настройки параметров лабиринта. Для успешного создания шаблона от актёра «Администратора» требуется задать ширину и высоту в специализированные поля;

- 3) выбор темы лабиринта;

- 4) выбор алгоритма генерации лабиринта.

В возможности «Игрока» входят:

- загрузить лабиринт из файла. Чтобы загрузить лабиринт из файла, нужно выбрать имя файла из существующих;

- настроить параметры игры. Для успешной настройки параметров игры, нужно выбрать тему оформления. Также актёру «Игрок» даётся выбор режима прохождения лабиринта вручную или автоматически. Чтобы пройти лабиринт вручную, будет предоставлена соответствующая кнопка «Шаг». Чтобы пройти лабиринт автоматически, нужно выбрать алгоритм прохождения и указать скорость.

2.5.5 Сценарии

Сценарий (scenario) - определенная последовательность действий, которая описывает действия актеров и поведение моделируемой системы в форме обычного текста [32].

В контексте языка UML сценарий используется для дополнительной иллюстрации взаимодействия актеров и вариантов использования.

Рассмотрим несколько сценариев.


Вариант использования: «Задать ширину»

Краткое описание. Позволяет администратору создать лабиринт и настроить его параметры. Включается в вариант использования «Создать шаблон».

Актант. Администратор.

Предусловия. Компьютер включен, вход в систему выполнен успешно. Пользователь прошел авторизацию как администратор. На экране показан интерфейс создания нового лабиринта, выбран раздел настроек параметров лабиринта.

Основной поток событий.

1) система выводит на экран форму создания нового лабиринта для администратора (см. рисунок 23). На форме отображается раздел «Настроек параметров лабиринта» с полями «Ширина» и «Высота» с предустановленными по умолчанию значениями, равными 15, и кнопкой «ОК». Также на форме отображаются кнопки вариантов способа расстановки входа и выхода: «Ручной» и «Автоматический», выбора алгоритма генерации лабиринта: «Алгоритм Эйлера», «Алгоритм Олдоса-Бродера» и выбор темы лабиринта: «Зима», «Весна», «Лето», «Осень». Эти кнопки временно недоступны администратору. На форме также отображаются кнопки: «О системе», «О разработчиках», «Сгенерировать», «Сохранить», справа вверху имеются кнопка  ;

2) администратор щёлкает левой кнопкой мыши по полю «Ширина»;

A1: Администратор щёлкает по кнопке «О системе».

A2: Администратор щёлкает по кнопке «О разработчиках».

A3: Администратор щёлкает по кнопке ▲ для установления ширины.

A4: Администратор щёлкает по кнопке ▼ для установления ширины.

A5: Администратор щёлкает по полю «Высота».

A6: Администратор щёлкает по кнопке ▲ для установления высоты.

A7: Администратор щёлкает по кнопке ▼ для установления высоты.

A8: Администратор щёлкает по кнопке «ОК».

A9: Администратор щёлкает по кнопке ✕.

3) система делает поле активным;

4) администратор вводит новое значение ширины;

5) система проверяет введённое значение на корректность по допустимому диапазону значений;

A10: Введённое значение ширины не корректно.

6) администратор нажимает кнопку «ОК»;

7) система создаёт шаблон заданного размера, он отображается в правой части формы в специальном поле. Вариант использования завершается успешно.

Альтернативы.

A1: Администратор щёлкает по кнопке «О системе».

A1.1 Переход к варианту использования «Посмотреть сведения о системе».

A2: Администратор щёлкает по кнопке «О разработчиках».

A2.1 Переход к варианту использования «Посмотреть сведения о разработчиках».

A3: Администратор щёлкает по кнопке ▲ для установления ширины.

A3.1 Значение ширины увеличивается на единицу.

A3.1.A1 Значение ширины достигло максимального.

Значение не увеличивается. Кнопка ▲ становится неактивной.

A4: Администратор щёлкает по кнопке ▼ для установления ширины.

A4.1 Значение ширины уменьшается на единицу.

A4.1.A1 Значение ширины достигло минимального. Значение

не уменьшается. Кнопка ▼ становится неактивной.

A5: Администратор щёлкает по полю «Высота».

A5.1 Переход к варианту использования «Задать высоту».

A6: Администратор щёлкает по кнопке ▲ для установления высоты.

A6.1 Переход к варианту использования «Задать высоту».

A7: Администратор щёлкает по кнопке ▼ для установления высоты.

A7.1 Переход к варианту использования «Задать высоту».

A8: Администратор щёлкает по кнопке «ОК».

A8.1 Переход к варианту использования «Создать шаблон».

A9: Администратор щёлкает по кнопке ✕ .

A9.1 Система закрывает текущую форму и завершает работу приложения.

A9.2 Вариант использования завершается.

A10: Введённое значение ширины не корректно.

A10.1 Значение вышло за рамки допустимого диапазона.

A10.1.A1 Значение вышло за рамки допустимого минимального значения. Система автоматически присваивает значение ширины равное минимальному.

A10.1.A2 Значение вышло за рамки допустимого максимального значения. Система автоматически присваивает значение ширины равное максимальному.

A10.2 Переход к п.5 основного потока.

Постусловия. При успешном завершении на экране появляется шаблон заданного размера, он отображается в правой части формы в специальном поле. После создания шаблона кнопки для выбора расстановки входа и выхода, алгоритма генерации и темы лабиринта становятся активными.


Вариант использования: «Выбрать скорость перемещения»

Краткое описание. Позволяет игроку настроить параметры игры и выбрать режим прохождения. Включается в вариант использования «Пройти лабиринт автоматически».

Актант. Игрок.

Предусловия. Компьютер включён, вход в систему выполнен успешно. Пользователь прошёл авторизацию как игрок. На экране отображён интерфейс настроек параметров игры, выбран раздел выбора скорости перемещения.

Основной поток событий.

1) система выводит на экран форму настроек параметров игры для игрока (см. рисунок 24). На форме отображается блок «Выбор режима прохождения» с предложенными выборами: «Ручной» и «Автоматический». Также на форме ниже располагается выбор темы оформления лабиринта с кнопками вариантов выбора «Лето», «Осень», «Зима», «Весна»; блок выбора алгоритма прохождения «Волновой» и «Одной руки», а также «Выбор скорости перемещения» с ползунком и дифференциацией от 0 до 4. При автоматическом режиме, его значение по умолчанию равно 0, а кнопка «Пройти» становится кнопкой «Шаг». На форме также отображаются следующие кнопки: «О системе», «О разработчиках», «Загрузить», «Пройти», а сверху имеется кнопка ;

2) игрок кнопкой мыши перемещает ползунок для выбора скорости вправо на 1 позицию;

A1: Игрок щёлкает кнопкой мыши по кнопке «О системе».

A2: Игрок щёлкает кнопкой мыши по кнопке «О разработчиках».

A3: Игрок щёлкает по кнопке .

A4: Игрок загружает новый лабиринт из файла.

A5: Игрок меняет режим прохождения.

A6: Игрок меняет тему оформления.

A7: Игрок меняет алгоритм прохождения.

A8: Игрок перемещает ползунок на 1 позицию влево.

A9: Игрок не перемещает ползунок выбора скорости.

3) система увеличивает значение на 1 и устанавливает соответствующее значение скорости перемещения;

4) игрок повторяет пункты 2), 3) до тех пор, пока не будет достигнута нужная ему скорость;

A10: Достигнуто максимальное значение скорости.

5) игрок щёлкает кнопкой мыши по кнопке «Пройти». Вариант использования завершается успешно.

Альтернативы.

A1: Игрок щёлкает кнопкой мыши по кнопке «О системе».

A1.1 Переход к варианту использования «Посмотреть сведения о системе».

A2: Игрок щёлкает кнопкой мыши по кнопке «О разработчиках».

A2.1 Переход к варианту использования «Посмотреть сведения о разработчиках».

A3: Игрок щёлкает кнопкой мыши по кнопке .

A3.1 Система закрывает текущую форму и завершает работу приложения, после чего завершается работа приложения.

A4: Игрок загружает новый лабиринт из файла.

A4.1 Переход к варианту использования «Загрузить лабиринт из файла».

A5: Игрок меняет режим прохождения.

A5.1 Переход к варианту использования «Выбрать режим прохождения».

A6: Игрок меняет тему оформления.

A6.1 Переход к варианту использования «Выбрать тему лабиринта».

A7: Игрок меняет алгоритм прохождения.

A7.1 Переход к варианту использования «Выбрать алгоритм прохождения».

A8: Игрок перемещает ползунок на 1 позицию влево.

A8.1 Система уменьшает значение на 1 и устанавливает соответствующее значение скорости перемещения.

A8.2 Игрок изменяет значение скорости ещё раз.

A8.2.A1 Переход к п.2 основного потока.

A8.2.A2 Переход к A8.

A8.3 Значение скорости достигло минимума, перемещение ползунка блокируется в сторону минимума.

A8.3.A1 Переход к п.2 основного потока.

A8.3.A2 Система принимает минимальное значение, устанавливает текущее значение скорости перемещения, кнопка «Пройти» заменяется на кнопку «Шаг».

A9: Игрок не перемещает ползунок выбора скорости.

A9.1 Скорость перемещения объекта по лабиринту остаётся заданной по умолчанию, то есть минимальной, кнопка «Пройти» заменяется на кнопку «Шаг».

A9.2 Вариант использования завершается.

A10: Достигнуто максимальное значение скорости.

A10.1 Значение скорости достигло максимума, перемещение ползунка блокируется в сторону максимума.

A10.2 Система принимает максимальное значение, устанавливает текущее значение скорости перемещения, вариант использования завершается.

Постусловия. При успешном выполнении объект в лабиринте автоматически с заданной скоростью находит выход из лабиринта. Прохождение отображается в правой части формы в специальном поле.

2.5.6 Диаграмма классов

Диаграммы классов – это наиболее часто используемый тип диаграмм, которые создаются при моделировании объектно-ориентированных систем, они показывают набор классов, интерфейсов и коопераций, а также их связи. На практике диаграммы классов применяют для моделирования статического представления системы, они служат основой для целой группы

взаимосвязанных диаграмм – диаграмм компонентов и диаграмм размещения [33].

На рисунке 27 приведена диаграмма классов системы (этап проектирования). В таблице 4 приведено описание классов.

Таблица 4 – Описание классов системы

Название класса	Назначение
Авторизация	Авторизация пользователя в системе
Регистрация	Регистрация пользователя в системе
Администратор	Создание лабиринта и настройки его параметров
Игрок	Настройки параметров игры и прохождение лабиринта
Система	Запуск приложения
Алгоритм Эйлера	Алгоритм генерации лабиринта
Алгоритм Олдоса-Бродера	Алгоритм генерации лабиринта
Волновой алгоритм	Алгоритм прохождения лабиринта
Алгоритм одной руки	Алгоритм прохождения лабиринта

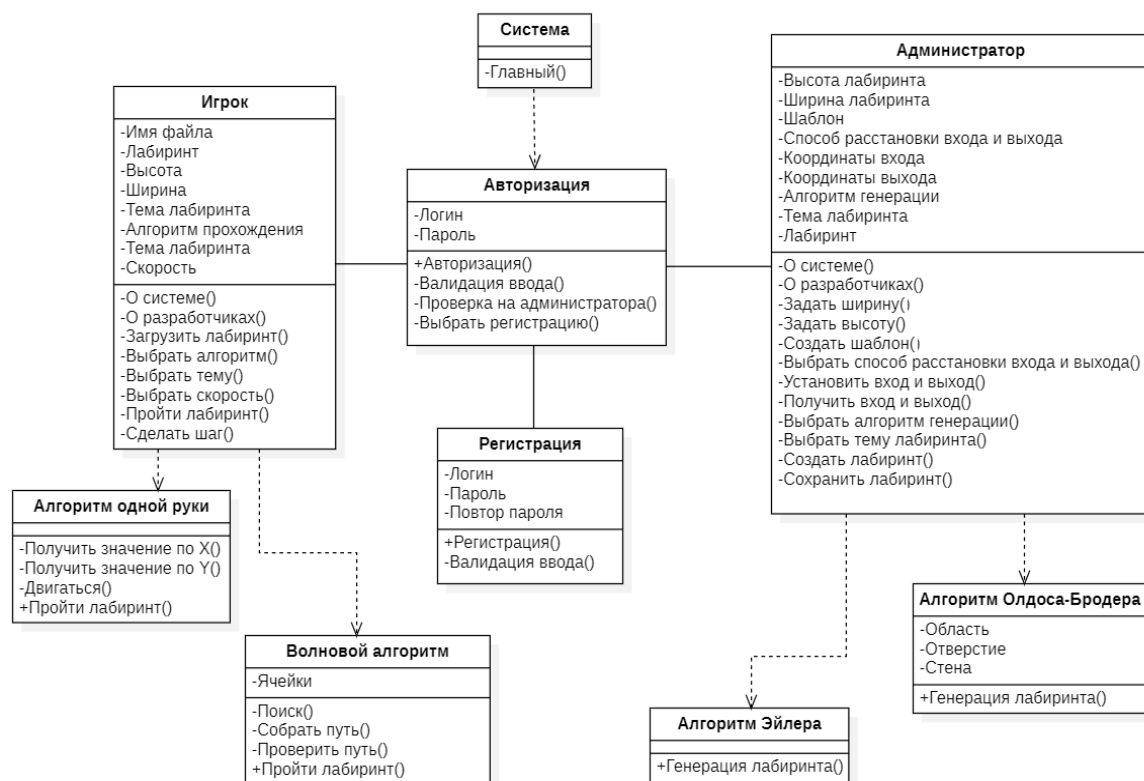


Рисунок 27 – Диаграмма классов системы

2.5.7 Диаграмма состояний

Диаграмма состояний – это тип диаграммы, используемый в UML для описания всех состояний системы и переходов между ними. Она отображает разрешенные состояния и переходы, а также события, которые влияют на эти переходы. Кроме этого, помогает визуализировать весь жизненный цикл объектов и, таким образом, помогает лучше понять системы, основанные на состояниях [34].

На рисунке 28 приведена диаграмма состояний системы. При запуске приложения открывается начальная форма авторизации. В форме пользователю требуется ввести логин и пароль, если данные введены верно, то при нажатии кнопки «Войти» пользователь переходит к работе с приложением в роли администратора или в роли игрока. Если данные неверны, то программа потребует их ввести повторно.

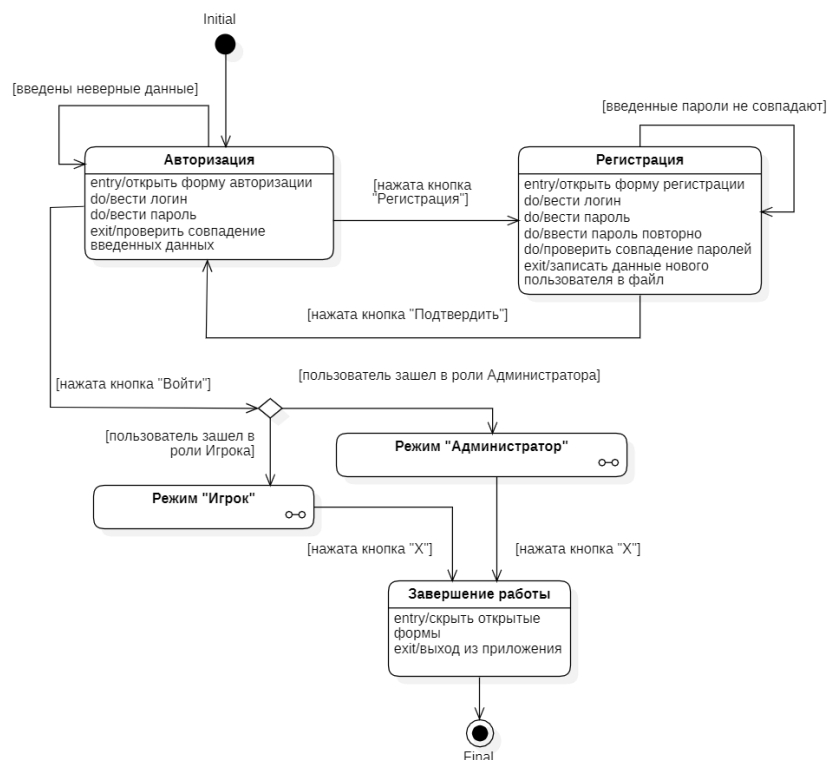


Рисунок 28 – Диаграмма состояний системы

Если пользователь впервые использует систему, требуется его регистрация в ней. Для этого нужно ввести логин и пароль, повторить введенный пароль. Если пароли совпадают, то произойдет запись нового

аккаунта. После этого пользователь, с учетом заданной роли, имеет возможность работы с системой, после чего происходит завершение работы.

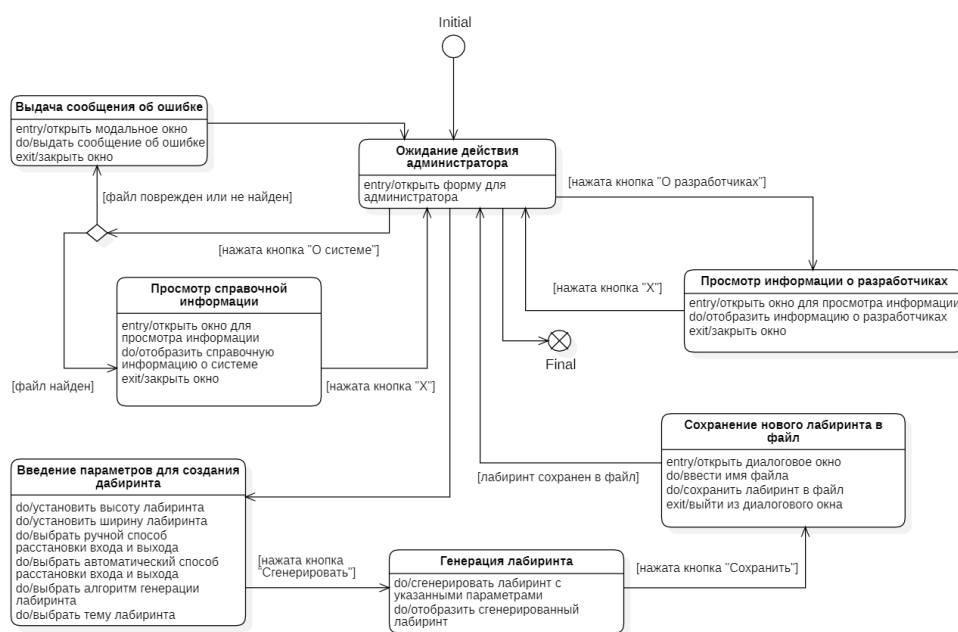


Рисунок 29 – Декомпозиция состояния «Настройка параметров лабиринта в роли администратора»

Администратору предоставляется возможность изменить несколько параметров для дальнейшей генерации лабиринта. По нажатию на кнопку «Сгенерировать» происходит генерация лабиринта, после чего в правом верхнем углу экрана отображается готовый лабиринт. По нажатию на кнопку «Сохранить» осуществляется переход в состояние сохранения файла, который принимает сгенерированный лабиринт, и сохраняет его в файл.

пользователя в роли игрока. Если нажать на кнопку «О системе», осуществляется переход в состояние для просмотра справочной информации. Если нажать на кнопку «О разработчиках» осуществляется переход в состояние для просмотра информации о разработчиках.

Загрузка лабиринта происходит после нажатия на кнопку «Загрузить». Далее игроку предоставляется возможность изменить настройки игры. В состояниях ручного и автоматического режима прохождения пользователю требуется либо выбрать клетки для перемещения, либо настроить параметры прохождения. После завершения прохождения наступает состояние ожидания действий игрока.

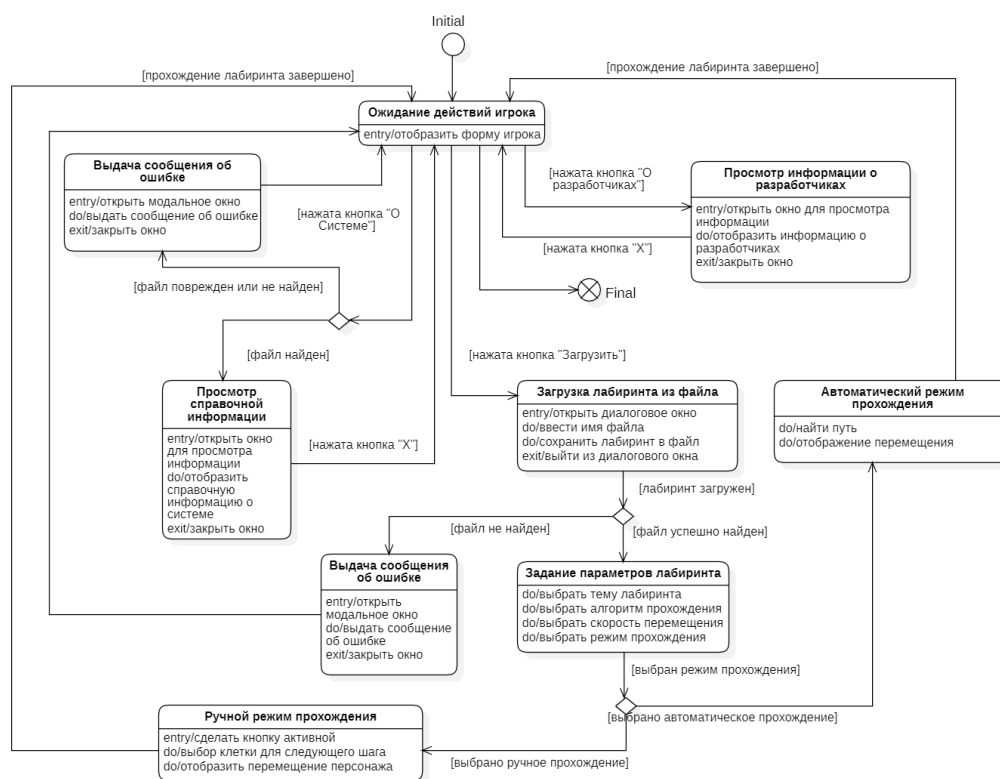


Рисунок 30 – Декомпозиция состояния «Настройка параметров лабиринта в роли игрока»

2.5.8 Диаграмма деятельности

Диаграмма деятельности – это UML-диаграмма, на которой показаны действия. Она представляет собой графическое представление рабочих процессов поэтапных действий и действий с поддержкой выбора, итерации и

параллелизма [35].

Графически диаграмма деятельности представляется в форме графа деятельности, вершинами которого являются состояния действия или состояния деятельности, а дугами – переходы от одного состояния действия к другому [36].

Диаграммы деятельности полезны при параллельном программировании, поскольку можно графически изобразить все ветви и определить, когда их необходимо синхронизировать [35].

На рисунке 31 приведена диаграмма деятельности «Расстановка входа и выхода». После регистрации в роли администратора система открывает форму для создания лабиринта. После этого пользователю нужно выбрать каким способом будут расставлены вход и выход: автоматическим или ручным. При выборе первого варианта система расставляет вход с выходом, соблюдая наложенные ограничения. При выборе второго варианта администратору предлагается возможность выбрать место входа с последующей проверкой системы на корректность расстановки. Если выполнение данной задачи завершилось успешно, пользователю предлагается выбрать место выхода с дальнейшей проверкой. По окончании процесса окошко отображает, в каких местах локализованы вход и выход.

2.5.9 Диаграмма последовательности

Диаграмма последовательности – это UML-диаграмма, на которой для некоторого набора объектов на единой временной оси показан жизненный цикл объекта (создание-деятельность-уничтожение некой сущности) и взаимодействие актёров информационной системы в рамках прецедента [37].

Она позволяет описать взаимодействие между объектами в системе в виде последовательности сообщений, действий и операций, отображая порядок выполнения действий и обмена информацией между объектами во времени [38].

Основными элементами диаграммы последовательности являются обозначения объектов (прямоугольники с названиями объектов), вертикальные «линии жизни», отображающие течение времени, прямоугольники, отражающие деятельность объекта или исполнение им определённой функции, и стрелки, показывающие обмен сигналами или сообщениями между объектами [37].

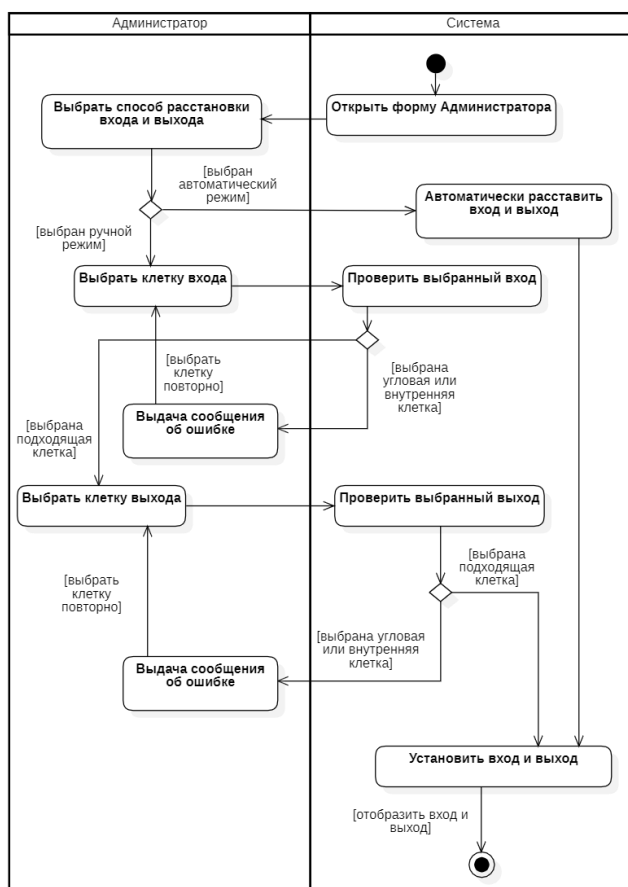


Рисунок 31 – Диаграмма деятельности «Расстановка входа и выхода»

На рисунках 32-33 приведены диаграммы последовательности системы для сценариев, приведенных в п.2.5.3.

2.6 Выбор и обоснование алгоритмов обработки данных

Создание эффективных алгоритмов для генерации лабиринта играет ключевую роль при разработке системы.

Алгоритмы генерации формируют структуру лабиринта, создавая сеть проходных и непроходных маршрутов, комнаты и преграды.

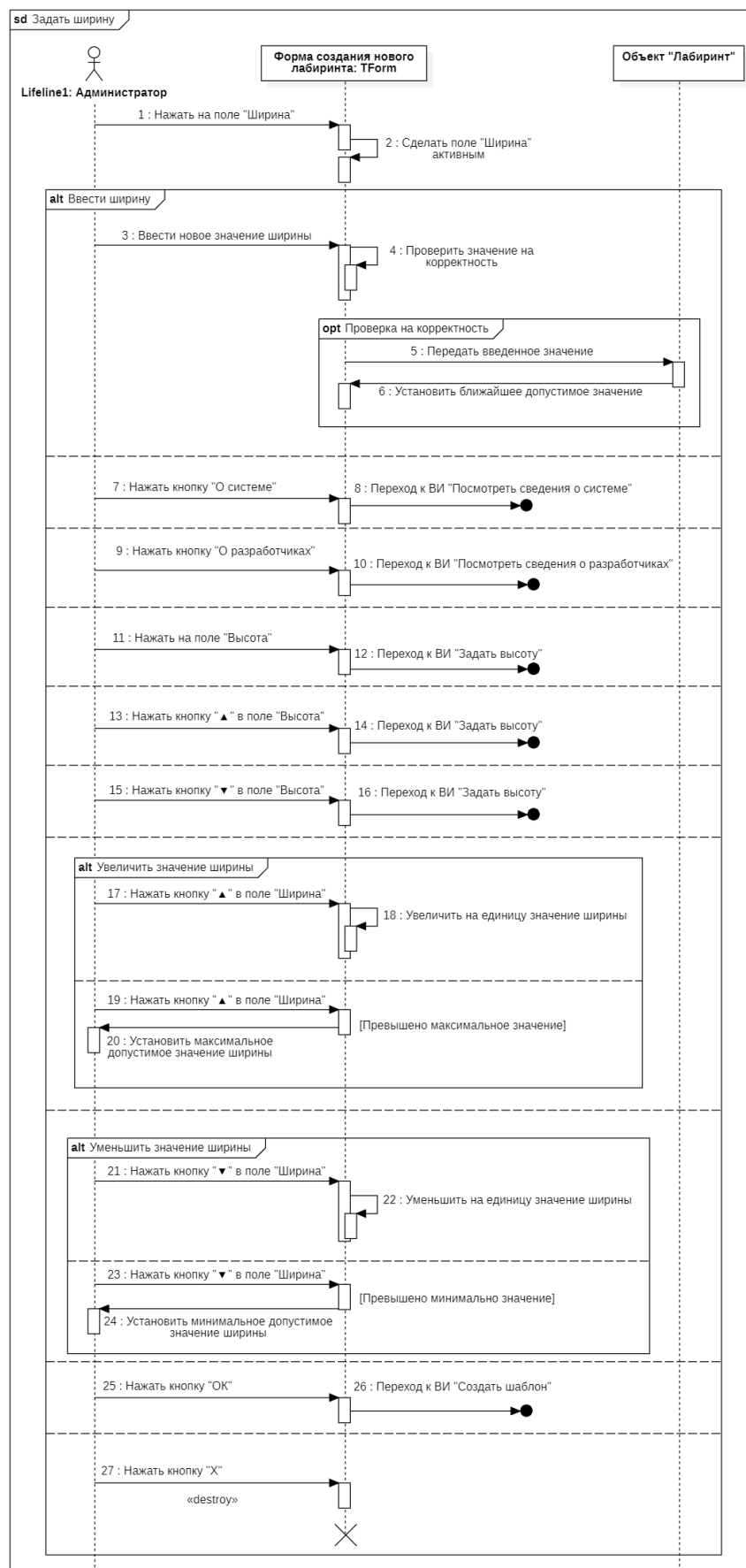


Рисунок 32 – Диаграмма последовательности для варианта использования
«Задать ширину»

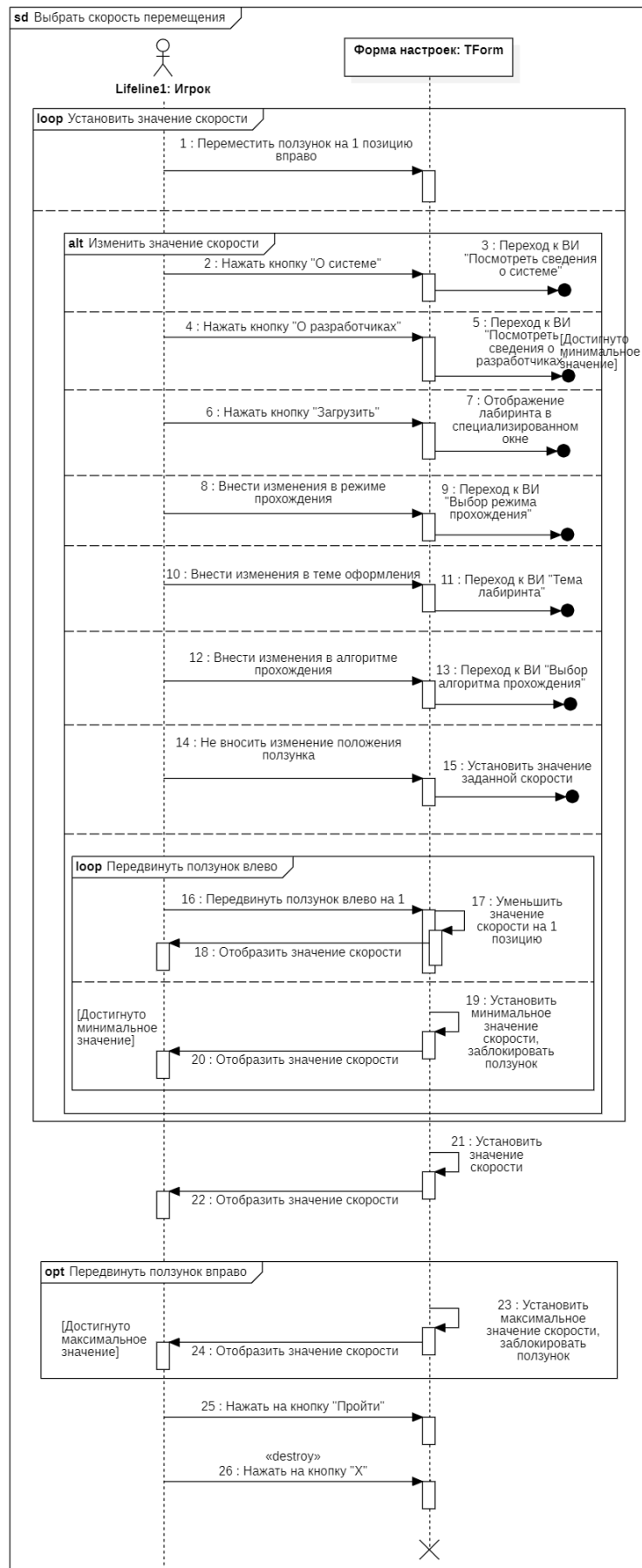


Рисунок 33 – Диаграмма последовательности для варианта использования «Выбрать скорость»

На рисунке 34 приведена схема алгоритма Олдоса-Бродера для генерации лабиринта.

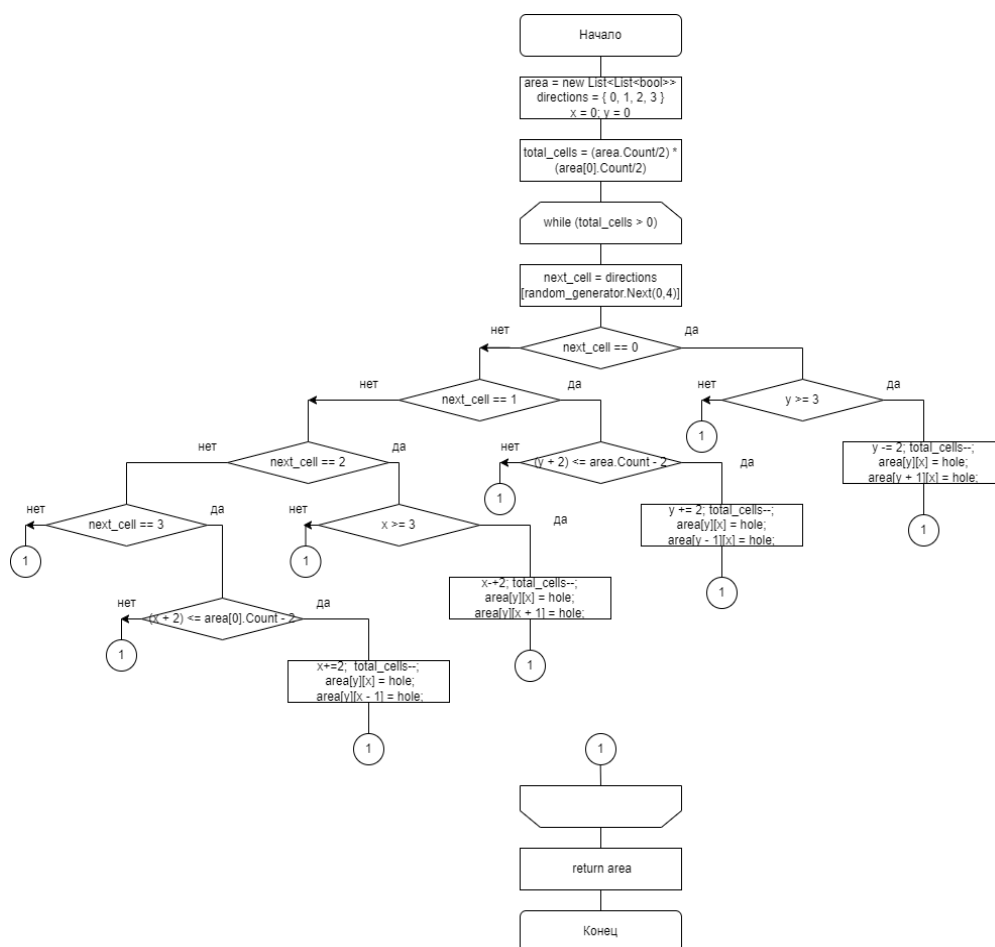


Рисунок 34 – Схема алгоритма Олдоса-Бродера для генерации лабиринта

В дополнение к генерации лабиринтов, важно использовать алгоритмы поиска пути, позволяющие системе автоматически находить выход. Такие алгоритмы рассчитывают оптимальный путь от старта до финиша, помогая персонажу избегать ловушек и препятствий внутри лабиринта.

В алгоритме сначала создается двумерный список `area`, представляющий собой лабиринт, и заполняется значениями `true` (стена) для всех клеток. Затем выбирается случайная стартовая точка внутри лабиринта. Далее устанавливается начальное количество ячеек `total_cells`, которое нужно посетить в процессе генерации лабиринта. Потом создается генератор случайных чисел и выбирается случайное направление из 0, 1, 2, 3 (север, юг, запад, восток).

Пока остаются непосещенные ячейки ($total_cells > 0$), проверяется возможность перемещения в выбранном направлении. Если возможно, перемещение осуществляется, и текущая и следующая ячейки становятся проходами (значение $hole$). Затем уменьшается количество оставшихся ячеек $total_cells$. В конце алгоритма возвращается сгенерированный лабиринт.

На рисунке 35 приведена схема алгоритма «Одной руки» для нахождения пути.

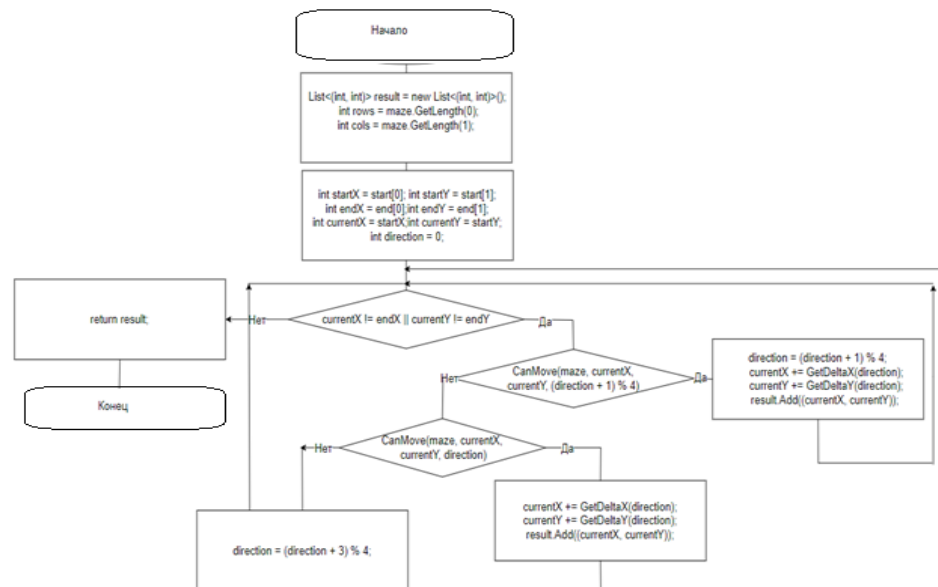


Рисунок 35 – Схема алгоритма «Одной руки» для нахождения пути

Этот алгоритм использует принцип конечного автомата для поиска пути от стартовой точки до конечной точки в лабиринте. Сначала создается список $result$ для хранения координат точек пути, затем получают количество строк и столбцов в лабиринте. Далее устанавливаются начальные координаты $startX$ и $startY$, а также конечные координаты $endX$ и $endY$.

Затем инициализируются переменные $currentX$ и $currentY$ для отслеживания текущего положения в лабиринте, а также переменная $direction$ для отслеживания направления движения.

Далее запускается цикл $while$, который будет выполняться до тех пор, пока текущие координаты не совпадут с конечными координатами. Внутри цикла происходит проверка возможности движения в различных направлениях. Если возможно двигаться вправо, то происходит изменение

направления и обновление текущих координат. Если движение вправо невозможно, но возможно двигаться в текущем направлении, то также происходит обновление текущих координат. Если невозможно двигаться вправо и в текущем направлении, то происходит изменение направления налево.

Функция CanMove проверяет, можно ли двигаться в определенном направлении, основываясь на наличии препятствий в лабиринте и границах лабиринта. Функции GetDeltaX и GetDeltaY возвращают изменение координаты X и Y соответственно, в зависимости от текущего направления.

По завершении цикла возвращается список result с координатами точек пути от начальной до конечной точки в лабиринте.

2.7 Выбор и обоснование комплекса программных средств

При разработке системы правильный подбор программного обеспечения имеет решающее значение. Неправильный выбор может привести к снижению производительности, нестабильной работе и ухудшению пользовательского опыта [39].

Рассмотрим несколько факторов при выборе комплекса программных средств:

- инструменты должны предоставлять возможности для реализации всех необходимых функций, таких как создание и прохождение лабиринтов;
- выбранные инструменты должны поддерживать платформы, на которых планируется запуск системы;
- программные средства должны быть надежными и иметь активную поддержку со стороны сообщества разработчиков. Это гарантирует наличие актуальной документации, ресурсов и готовых решений, что значительно упрощает процесс разработки.

Выбор программного обеспечения охватывает различные аспекты, включая язык программирования, среду разработки и операционную систему.

2.7.1 Выбор языка программирования

В настоящее время существует обширный спектр языков программирования, каждый из которых был создан для решения определенного круга задач. Выбор конкретного языка программирования является важным этапом при проектировании и разработке информационной системы, поскольку от него зависят многие факторы, такие как:

- временные затраты на создание программного обеспечения;
- временные затраты на проведение тестирования;
- возможность переноса разработанного продукта на другие программно-аппаратные платформы;
- возможность оперативного внесения изменений в код программы.

Самыми популярными и востребованными являются следующие языки программирования: C#, Java, Python, JavaScript.

C# – современный объектно-ориентированный язык программирования. C# позволяет разработчикам создавать разные безопасные и надежные приложения. Имеет множество различных библиотек, которые упрощают и ускоряют процесс создания приложения [39]. Для разрабатываемой системы крайне важным требованием являлось наличие встроенного графического конструктора. В качестве языка программирования для разработки данного программного решения был выбран C#, поддерживающий создание приложений под Windows Forms.

2.7.2 Выбор среды программирования

Среда разработки представляет собой совокупность различных инструментов, облегчающих процесс создания программного обеспечения. В ее состав обычно входят текстовый редактор, компилятор или интерпретатор языка программирования, средства автоматизации сборки проекта и отладчик. Кроме того, среда разработки может включать в себя утилиты для интеграции с системами контроля версий и другие вспомогательные

программы. Существуют среды, ориентированные на работу с одним конкретным языком программирования, однако большинство современных сред позволяют разрабатывать приложения сразу на нескольких языках [39].

Например, существуют такие бесплатные среды разработки, как PyCharm, IntelliJ IDEA, Eclipse, Microsoft Visual Studio.

Visual Studio представляет собой мощную интегрированную среду разработки (IDE), обладающую обширным функционалом для создания, редактирования, отладки и компиляции программного кода, а также развертывания готовых приложений. Помимо инструментов, непосредственно связанных с работой над исходным кодом, Visual Studio включает в себя систему контроля версий, различные расширения и множество других полезных возможностей [39].

Таким образом, для реализации поставленной задачи была отобрана интегрированная среда разработки Microsoft Visual Studio 2022.

2.7.3 Выбор операционной системы

Операционные системы (MS DOS, UNIX, OS/2, Windows, Linux и др.) – обязательное дополнение информационной системы (ИС), они предназначены для планирования и управления вычислительными ресурсами ВС, организуют выполнение пользовательских программ и взаимодействие пользователя с компьютером. Операционные системы (ОС) выступают в роли посредника, то есть обеспечивают интерфейс между аппаратурой компьютера и пользователем с его задачами [39].

В качестве платформы для создания и функционирования информационной системы было решено использовать операционную систему Windows 10, разработанную корпорацией Microsoft. Ее популярность значительно возросла, особенно благодаря популярности Windows 10. Windows 10, в основном из-за ее совместимости. Также основными достоинствами являются производительность, безопасность и оптимизация почти для каждого пользователя.

3 Реализация системы

3.1 Разработка и описание интерфейса пользователя

Разработка интерфейса необходима для того, чтобы повысить продуктивность пользователей, обеспечить простоту использования программы, а также для получения конкурентного преимущества, поскольку некачественный интерфейс может негативно сказаться на успехе всего проекта.

Основные особенности разработки интерфейса:

- простота: упрощение интерфейса до основных элементов поможет пользователю легко и быстро освоить продукт [24];
- интуитивность: интерфейс должен быть интуитивно понятным, чтобы пользователи могли взаимодействовать с ним без обучения [24];
- эстетика: интерфейс должен быть привлекательным, потому что хороший визуальный дизайн повышает доверие пользователей к продукту и стимулирует их возвращаться к нему снова [23].
- скорость: система должна быстро реагировать на действия пользователя, оптимизация производительности интерфейса помогает снизить время ожидания и сделать взаимодействие плавным [23].

Во время выполнения курсовой работы было разработано настольное приложение генерирования структуры лабиринта и нахождения выхода из него.

При запуске приложения открывается форма авторизации, чтобы пользователь смог войти в систему под соответствующей ему ролью, она изображена на рисунке 36. Пользователю необходимо заполнить поля для ввода логина и пароля и нажать кнопку «Войти» для входа в систему или кнопку «Регистрация», если пользователь ранее не создавал аккаунт.

Если пользователь не был ранее зарегистрирован, то при нажатии на кнопку «Регистрация» осуществляется переход на форму регистрации, она изображена на рисунке 37.

Авторизация

Введите логин:

Введите пароль:

Войти

Регистрация

Рисунок 36 – Экранная форма авторизации

Регистрация

Введите логин:

Введите пароль:

Повторите пароль:

Подтвердить

Регистрация

Рисунок 37 – Экранная форма регистрации

При регистрации пользователю нужно заполнить поля логина, пароля и повтора пароля, затем нажать кнопку «Подтвердить» для перехода на форму авторизации. По умолчанию создается аккаунт для игрока.

Если при авторизации ввести данные от аккаунта администратора, то при нажатии на кнопку «Войти» произойдет переход на форму «Admin», в которой происходит создание лабиринта.

Для создания лабиринта администратору необходимо ввести ширину, высоту и нажать кнопку «ОК», после чего в специальном поле слева появляется сетка заданного размера. На рисунке 38 приведен пример создания лабиринта с высотой и шириной равными 15.

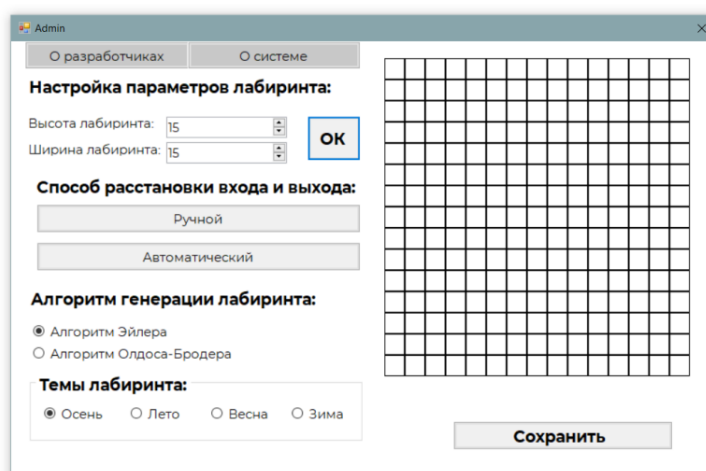


Рисунок 38 – Экранная форма администратора

Далее администратору необходимо выбрать способ расстановки входа и выхода: ручной или автоматический, нажав соответствующую кнопку. При этом важно учитывать правила: вход и выход располагаются строго по периметру лабиринта и не могут находиться в угловых клетках.

На рисунке 39 приведен пример ручной расстановки входа и выхода, где вход выделяется зеленым цветом, а выход – красным.

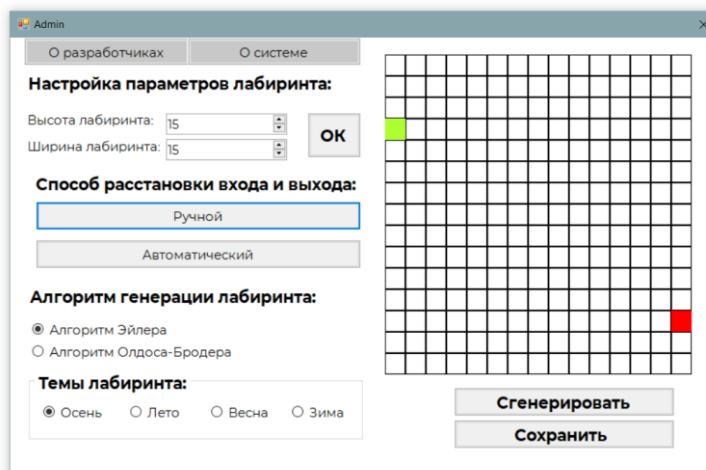


Рисунок 39 – Пример ручной расстановки входа и выхода

Администратор также может выбрать алгоритм генерации лабиринта и тему оформления. При нажатии на кнопку «Сгенерировать» в специальном поле появится лабиринт выбранного размера и заданной темы оформления. Пример сгенерированного лабиринта изображен на рисунке 40.

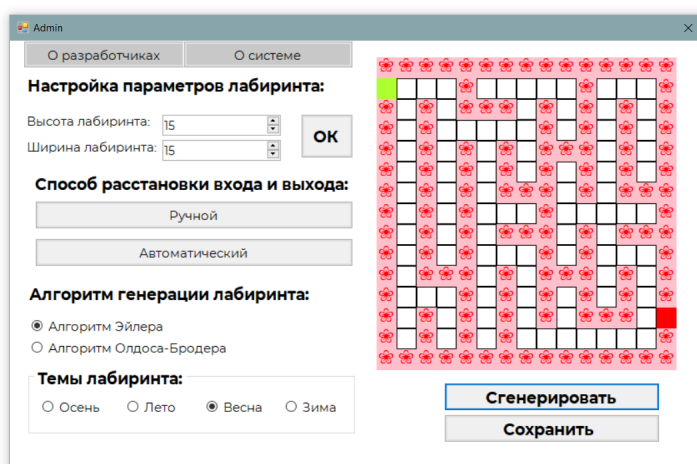


Рисунок 40 – Пример генерации лабиринта

Администратор может повторно нажать кнопку «Сгенерировать» для получения другого результата или нажать кнопку «Сохранить» для того, что оставить этот вариант лабиринта у себя в системе в файле формата *.xml, задав имя и адрес сохранения.

Если при авторизации ввести логин и пароль от аккаунта игрока, то откроется форма «Gamer», изображенная на рисунке 41.



Рисунок 41 – Форма игрока

Для начала работы необходимо загрузить лабиринт из файла, для этого необходимо нажать кнопку «Загрузить» и выбрать подходящий файл. Если будет выбран файл неверного формата или с неверной структурой, система выведет сообщение об ошибке.

При успешной загрузке лабиринта из файла, в специальном окне появится сам лабиринт, пример которого приведен на рисунке 42.

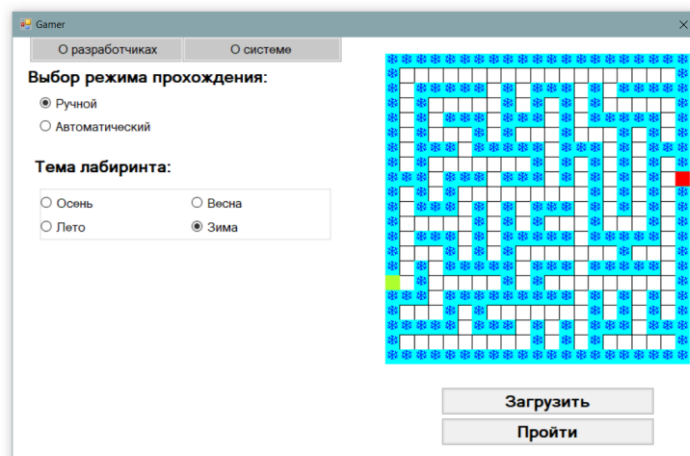


Рисунок 42 – Пример успешной загрузки лабиринта из файла

Пользователь может выбрать тему оформления лабиринта и способ прохождения: ручной и автоматический. Если игрок выберет ручной режим, то нужно нажать кнопку «Пройти» для начала игры. Управление персонажем будет происходить с помощью стрелок на клавиатуре пользователя.

Если игрок выберет автоматический режим прохождения, то появятся дополнительные настройки параметров игры: выбор алгоритма и скорости прохождения, это показано на рисунке 43.

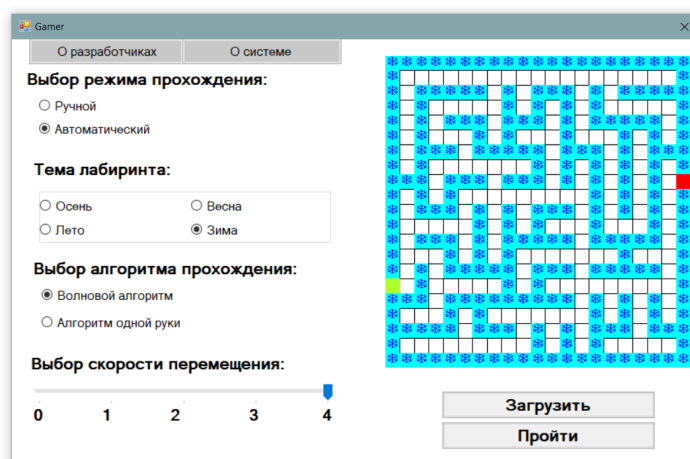


Рисунок 43 – Настройка параметров игры

Пользователь в любой момент может повторно нажать кнопку «Загрузить» и выбрать другой файл.

Для подтверждения выбора всех настроек и запуска игры необходимо нажать кнопку «Пройти». При прохождении лабиринта настройки

параметров игры скрываются от игрока. Пример прохождения лабиринта показан на рисунке 44.

Любой пользователь системы может нажать кнопки «О системе» и «О разработчиках», в таком случае откроется дополнительное окно с соответствующей информацией.

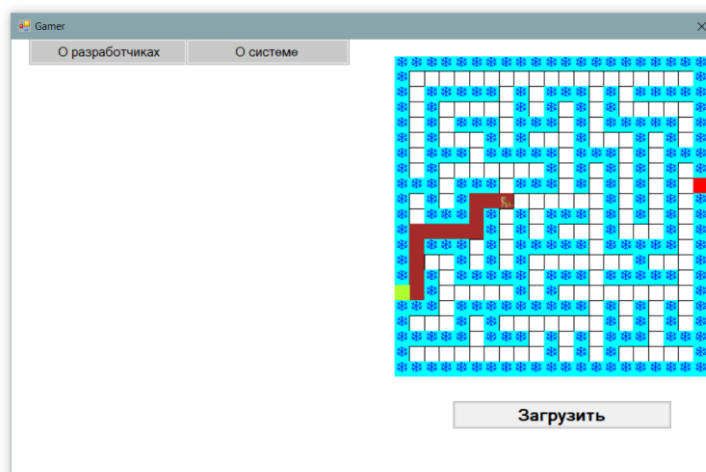


Рисунок 44 – Прохождение лабиринта

Пример окна со сведениями о разработчиках показан на рисунке 45.

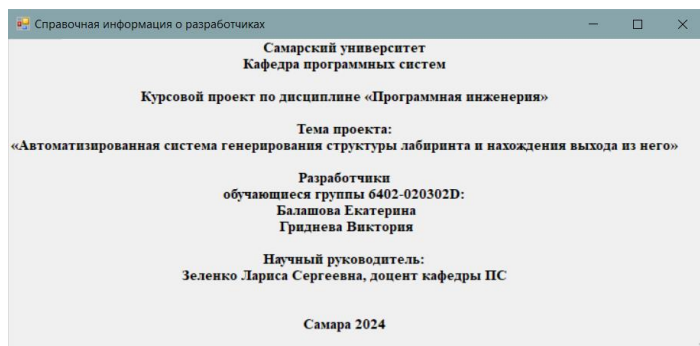


Рисунок 45 – Сведения о разработчика

3.2 Диаграммы реализации

Диаграммы реализации предназначены для отображения состава компилируемых и выполняемых модулей системы, а также связей между ними. Диаграммы реализации разделяются на два конкретных вида: диаграммы компонентов (component diagrams) и диаграммы развертывания (deployment diagrams) [40].

3.2.1 Диаграмма компонентов

Диаграмма реализации – это диаграмма, определяющая физическое представление модели, это совокупность связанных физических сущностей, включая программное и аппаратное обеспечение. В данной диаграмме акцент сделан на физическую организацию классов в виде компонентов и подсистем.

На рисунке 46 приведена диаграмма компонентов, их описание приведено в таблице 5.

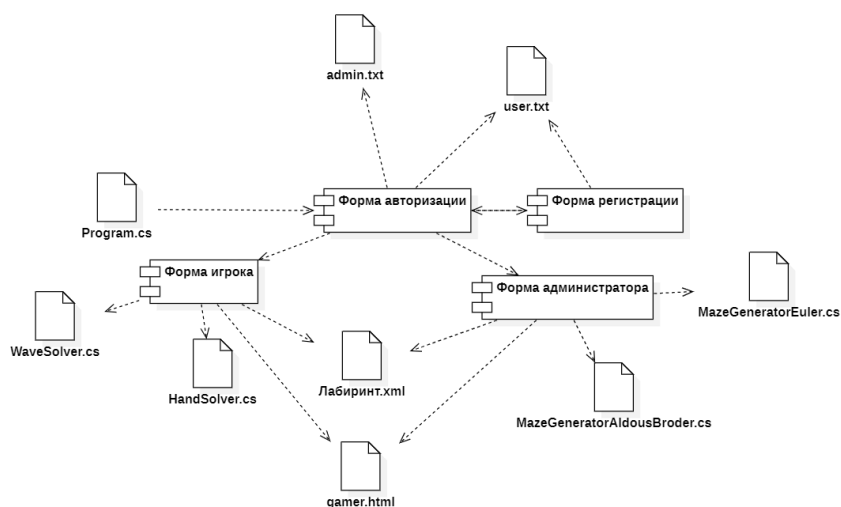


Рисунок 46 – Диаграмма компонентов системы

Таблица 5 – Описание компонентов системы

Название компонента	Назначение компонента	Подсистема
1	2	3
Форма авторизации	С учетом роли авторизовать пользователя в системе	Подсистема авторизации
Форма регистрации	Зарегистрировать нового пользователя в системе под ролью игрок	Подсистема регистрации
Форма администратора	Настроить параметров лабиринта, сгенерировать его и сохранить файл	Подсистема «Администратор»
Форма игрока	Выбрать файл, настроить параметры игры и пройти лабиринт	Подсистема «Игрок»

– помимо основных компонентов корректную работу программы обеспечивают следующие файлы:

– admin.txt. Файл с логинами и паролями всех администраторов, зарегистрированных в системе;

– user.txt. Файл с логинами и паролями всех игроков, зарегистрированных в системе;

– MazeGeneratorEuler.cs. Файл с алгоритмом Эйлера для построения лабиринта;

– MazeGeneratorAldousBroder.cs. Файл с алгоритмом Олдоса-Бордера для построения лабиринта;

– лабиринт.xml. Файл с лабиринтом для его визуального отображения;

– gamer.html. Файл для отображения в браузере справочной информации о системе;

– HandSolver.cs. Файл с реализованным алгоритмом одной руки;

– WaveSolver.cs. Файл с реализованным волновым алгоритмом;

– Program.cs. Файл с кодом программы, который выполняется при запуске системы.

3.2.2 Диаграмма развертывания

Диаграмма развертывания – диаграмма, на которой представлены узлы выполнения программных компонентов реального времени, а также процессов и объектов. Диаграмма развертывания показывает наличие физических соединений – маршрутов передачи информации между аппаратными устройствами, задействованными в реализации системы. Применяется для представления общей конфигурации и топологии распределенной программной системы и содержит изображение размещения компонентов по отдельным узлам системы.

На рисунке 47 приведена диаграмма развертывания системы. Прежде всего у пользователя должен быть персональный компьютер, а вместе с ним операционная система Windows 10, файловая система, браузер и

структурный файл Maze.exe для запуска и загрузки проекта. Помимо этого, для полноценной работы игры необходимы артефакты, а именно:

- admin.txt. Файл с логинами и паролями всех администраторов;
- user.txt. Файл с логинами и паролями всех игроков;
- файл со всеми лабиринтами с расширением .xml;
- gamer.html. Файл для отображения в браузере справочной информации о системе.

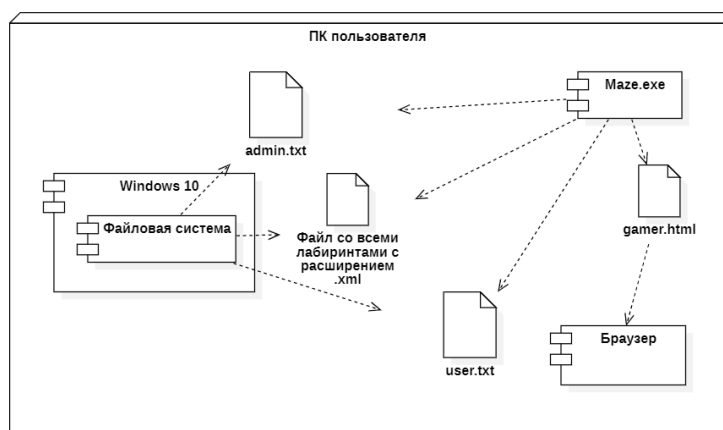


Рисунок 47 – Диаграмма развертывания системы

3.2.3 Диаграмма классов

В соответствии со спецификацией, приведенной в п. 2.5.6, и с учетом выбранного языка программирования (см. п. 2.7.1) разработана диаграмма классов системы (этап реализации), приведенная на рисунке 48.

3.3 Выбор и обоснование комплекса технических средств

3.3.4 Расчет объема занимаемой памяти

Расчет объема внешней памяти

Для расчета необходимого объема свободной внешней памяти, необходимой для функционирования системы, воспользуемся следующей формулой:

$$V_{\text{ЖД}} = V_{\text{ОС}} + V_{\text{ПР}} + V_{\text{ф}} + V_{\text{браузера}} + V_{\text{справки}},$$

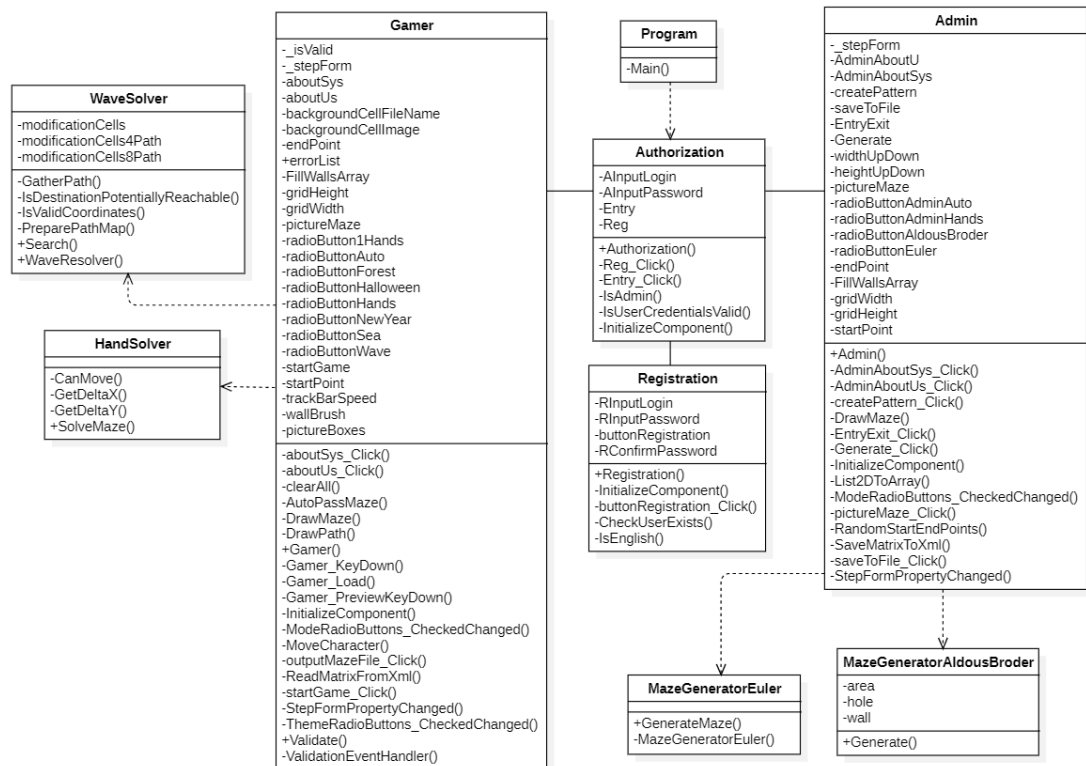


Рисунок 48 – Описание классов системы (этап реализации)

где V_{OC} – объем памяти, занимаемый операционной системой (операционная система Windows 10 Home 64 бит, $V_{OC} = 16$ Гб);

$V_{ПР}$ – объем памяти, занимаемый непосредственно файлами приложения ($V_{ПР} = 0,88$ Мб);

V_{Φ} – объем памяти, занимаемый файлами хранения данных пользователей и лабиринтов, дадим оценку сверху $V_{\Phi} = 0,01$ Мб;

$V_{браузера}$ – объем памяти, занимаемый браузером Google Chrome; дадим оценку сверху $V_{браузера}$ в 800 Мб;

$V_{справки}$ – объем памяти, необходимый для хранения файла справки ($V_{справки} = 0,008$ Мб).

Таким образом, суммарный объем внешней памяти составит:

$$V_{ЖД} = 16 \text{ Гб} + 0,88 \text{ Мб} + 0,01 \text{ Мб} + 800 \text{ Мб} + 0,008 \text{ Мб} \sim 16,801 \text{ Гб}.$$

Расчет объема ОЗУ

Для расчета необходимого объема ОЗУ воспользуемся следующей формулой:

$$V_{\text{ОЗУ}} = V_{\text{ОС}} + V_{\text{ПР}} + V_{\text{браузера}},$$

где $V_{\text{ОС}}$ – ОЗУ, занимаемое операционной системой (2 Гб);

$V_{\text{ПР}}$ – ОЗУ, которое займет само приложение (не превысит 30 Мб);

$V_{\text{браузера}}$ – ОЗУ, занимаемое браузером (оценим его сверху значением в 680 Мб).

Суммарные объемы ОЗУ составит:

$$V_{\text{ОЗУ}} = 2 \text{ Гб} + 30 \text{ Мб} + 680 \text{ Мб} \sim 2.71 \text{ Гб}.$$

Таким образом, 2.76 Гб оперативной памяти можно считать минимально необходимым для функционирования системы.

3.3.5 Минимальные требования, предъявляемые к системе

Для корректного функционирования системы необходимо:

- тип ЭВМ: x86-64 совместимый;
- объем ОЗУ – не менее 3,5 Гб;
- объем свободного дискового пространства – не менее 18 Гб;
- клавиатура или иное устройство ввода;
- мышь или иное манипулирующее устройство;
- процессор – AMD Ryzen 5 не менее 2,10 ГГц;
- дисплей с разрешением не менее 1920×1080 пикселей;
- операционная система Windows 10 и выше;
- браузер Google Chrome.

ЗАКЛЮЧЕНИЕ

В процессе выполнения курсового проекта была разработана автоматизированная система генерирования структуры лабиринта и нахождения выхода из него, позволяющая сгенерировать лабиринт с помощью двух алгоритмов, расставить вход и выход в ручном или автоматическом режиме, а также сохранить созданный лабиринт в файл заданной структуры. Система позволяет загрузить лабиринт из файла, выбрать тему оформления лабиринта, пройти лабиринт в автоматическом режиме с помощью двух алгоритмов, выбрав скорость перемещения, а также пройти лабиринт в ручном режиме.

В первом разделе приведены основные понятия предметной области, описаны характеристики систем-аналогов, приведен их сравнительный анализ. На основе проведенного анализа выполнена объектная декомпозиция, отраженная в диаграмме объектов, и сформулирована постановка задачи.

Во втором разделе выполнено проектирование системы. Выбрана и обоснована архитектура системы, построена структурная схема системы, разработана спецификация требований, разработаны прототипы интерфейса пользователя системы, разработан информационно-логический проект системы, который отображен в соответствующих диаграммах. Разработаны и описаны алгоритмы обработки данных, выбран и обоснован комплекс программных средств.

В третьем разделе разработан и описан интерфейс пользователя, построены диаграммы реализации, выполнены ресурсные расчеты, выбран и обоснован комплекс технических средств.

Разработанная система может использоваться в играх, где требуется лабиринт и нахождение выхода из него. Также система может использоваться в образовательных целях, где нужно изучать алгоритмы генерации лабиринтов и нахождения выхода из них.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

- 1 Лабиринт [Электронный ресурс]. URL: <https://ru.wikipedia.org/wiki/Лабиринт> (дата обращения: 17.09.2024).
- 2 Игровые лабиринты [Электронный ресурс]. URL: <https://dskrnd.ru/blog/sovety-pokupatelyam/igrovye-labirinty-kakuyu-polzu-dlya-detskogo-razvitiya-oni-prinosyat/#:~:text=%20Лабиринт%20-%20это%20не%20только%20интересное,игра%20способствует%20выплеску%20лишней%20энергии> (дата обращения: 17.09.2024).
- 3 RAD (программирование) [Электронный ресурс]. URL: [https://ru.wikipedia.org/wiki/RAD_\(программирование\)](https://ru.wikipedia.org/wiki/RAD_(программирование)) (дата обращения: 17.09.2024).
- 4 Краткая история UML [Электронный ресурс]. URL: <http://techn.sstu.ru/kafedri/подразделения/1/MetMat/murashev/oor/lec/lec12.htm#:~:text=UML> (дата обращения: 17.09.2024).
- 5 Буч Г., Рамбо Д., Якобсон А. Язык UML. Руководство пользователя. Изд. 2-е. М.: ДМК Пресс, 2006. 546 с
- 6 Технология объектно-ориентированного анализа и проектирование информационных систем [Электронный ресурс]. URL: <https://textarchive.ru/c-1890952.html> (дата обращения: 18.09.2024).
- 7 Объекто-ориентированный анализ и проектирование [Электронный ресурс]. URL: https://studopedia.ru/3_33047_ob-ektno-orientirovanniy-analiz-i-proektirovanie.html (дата обращения: 18.09.2024).
- 8 Лабиринты: классификация, генерирование, поиск решений [Электронный ресурс]. URL: <https://habr.com/ru/articles/445378/> (дата обращения: 17.09.2024).
- 9 Лабиринт [Электронный ресурс]. URL: <https://ru.wikipedia.org/wiki/Лабиринт> (дата обращения: 18.09.2024).
- 10 Алгоритмы генерации лабиринтов [Электронный ресурс]. URL: <https://tproger.ru/articles/maze-generators?ysclid=m183grujpj942114810+,+> (дата обращения: 18.09.2024).

11 Gotcha [Электронный ресурс]. URL: [https://en.wikipedia.org/wiki/Gotcha_\(video_game\)](https://en.wikipedia.org/wiki/Gotcha_(video_game)) (дата обращения: 19.09.2024).

12 Plottersvg [Электронный ресурс]. URL: <https://plottersvg.ru/maze-generator> (дата обращения: 19.09.2024).

13 Принципы объектно-ориентированного проектирования [Электронный ресурс]. URL: https://studopedia.su/20_118710_printsipi-obektno-orientirovannogo-proektirovaniya.html?ysclid=m18zcakru9268504810 (дата обращения: 19.09.2024).

14 UML: обзор основных типов диаграмм, диаграмма объектов [Электронный ресурс]. URL: <https://habr.com/ru/articles/800849/> (дата обращения: 19.09.2024).

15 Системный анализ [Электронный ресурс]. URL: <https://victor-safronov.ru/systems-analysis/lectures/rodionov/00.html> (дата обращения: 29.09.2024).

16 Проектирование [Электронный ресурс]. URL: <https://okbm.ru/proektirovanie/> (дата обращения: 19.09.2024).

17 Немного об архитектурах программного обеспечения [Электронный ресурс]. URL: <https://habr.com/ru/companies/mws/articles/276297/> (дата обращения: 19.09.2024).

18 Виды архитектур ПО [Электронный ресурс]. URL: <https://dzen.ru/a/ZdeO8pAHE1CaSJnz> (дата обращения: 19.09.2024).

19 Что такое микросервисы и микросервисная архитектура? [Электронный ресурс]. URL: <https://www.diasoft.ru/about/publications/20743/> (дата обращения: 19.09.2024).

20 Как написать спецификацию требований к программному обеспечению [Электронный ресурс]. URL: <https://stfalcon.com/ru/blog/post/How-to-Write-a-Software-Requirements-Specification> (дата обращения: 16.10.2024).

21 Спецификация [Электронный ресурс]. URL: <https://ru.wikipedia.org/wiki> (дата обращения: 16.10.2024).

22 Класс Exception. Стандартные классы исключений. Операторы обработки исключений. [Электронный ресурс]. URL: <https://saliyna.narod.ru/OOP/Lesson23/Lecture.html> (дата обращения: 17.10.2024).

23 Интерфейс [Электронный ресурс]. URL: <https://blog.skillfactory.ru/glossary/interface/> (дата обращения: 07.10.2024).

24 Зачем нужно прототипирование интерфейсов? [Электронный ресурс]. URL: <https://webdoka.ru/blog/zachem-nuzhno-prototipirovanie-ux/> (дата обращения: 20.10.2024).

25 Пользовательский интерфейс: правила и этапы разработки [Электронный ресурс]. URL: <https://bangbangeducation.ru/point/ux-ui-dizain/polzovatelskij-interfejs/#5> (дата обращения: 07.10.2024).

26 Проектирование информационной системы [Электронный ресурс]. URL: <https://3uch.ru/textbooks/quire/xio/quide> (дата обращения: 27.10.2024).

27 Соммервиль, И. Инженерия программного обеспечения/ Иан Соммервиль. М., СПб, Киев: Издательский дом «Вильямс», 2002. 626 с.

28 ГОСТ 12.1.007-76. Система стандартов безопасности труда. Вредные вещества. Классификация и общие требования безопасности [Текст] . Дата введения 01.01.1977. М.: Стандартиформ, 2007. 7 с.

29 Унифицированный язык моделирования UML [Электронный ресурс]. URL: <https://samara.mgpu.ru/~dzhadzha/dis/15/200.html> (дата обращения: 27.10.2024).

30 UML [Электронный ресурс]. URL: <https://blog.skillfactory.ru/glossary/uml/> (дата обращения: 27.10.2024).

31 Элементы графической нотации диаграммы вариантов использования [Электронный ресурс]. URL: <https://intuit.ru/studies/courses/32/32/lecture/1004> (дата обращения: 27.10.2024).

32 Спецификация требований и рекомендации по написанию эффективных вариантов использования [Электронный ресурс]. URL: <https://intuit.ru/studies/courses/32/32/lecture/1006> (дата обращения: 09.11.2024).

33 Унифицированный язык программирования UML. Базовые понятия диаграмм классов [Электронный ресурс]. URL: https://it.kgsu.ru/UML/uml_0076.html (дата обращения: 09.11.2024).

34 Диаграмма состояний [Электронный ресурс]. URL: https://pikabu.ru/story/karera_v_it_sistemnyiy_analitik_chast_3_diagrammyi_uml_9980766 (дата обращения: 18.11.2024).

35 Что находится между идеей и кодом? Обзор 14 диаграмм UML [Электронный ресурс]. URL: <https://habr.com/ru/articles/508710/> (дата обращения: 03.12.2024).

36 Диаграммы деятельности [Электронный ресурс]. URL: https://ciu.nstu.ru/kaf/persons/1914/page47048/diagramm_deyatelnosti (дата обращения: 03.12.2024).

37 Диаграмма последовательности [Электронный ресурс]. URL: https://ru.wikipedia.org/wiki/Диаграмма_последовательности (дата обращения: 03.12.2024).

38 Диаграмма последовательности (sequence diagrams) [Электронный ресурс]. URL: https://itonboard.ru/analysis/394-diagramma_posledovatelnosti_sequence_diagrams_uml/ (дата обращения: 03.12.2024).

39 Мишель А.А. Информатика и программирование: программные средства реализации информационных процессов. Томск, 2013. 51 с.

40 Диаграмма реализации UML [Электронный ресурс]. URL: https://ru.wikipedia.org/wiki/Windows_10 (дата обращения: 03.12.2024).

ПРИЛОЖЕНИЕ А

Руководство пользователя

А.1 Назначение системы

Система обладает функциональностью генерации лабиринта при помощи двух алгоритмов, установки входа и выхода в ручном или автоматическом режиме, а также сохранения созданного лабиринта в файл заданного формата. Система позволяет загружать лабиринты из файлов, выбирать их тематическое оформление, проходить лабиринт автоматически с установкой скорости движения и в ручном режиме.

Система может использоваться в играх, где требуется лабиринт и нахождение выхода из него. Также система может использоваться в образовательных целях, где нужно изучать алгоритмы генерации лабиринтов и нахождения выхода из них.

А.2 Условия работы системы

Для корректной работы системы необходимо наличие соответствующих программных и аппаратных средств.

1 Требования к техническому обеспечению:

- тип ЭВМ: x64 совместимый;
- объем ОЗУ – не менее 4 Гб;
- объем свободного дискового пространства – не менее 20 Гб;
- клавиатура или иное устройство ввода;
- мышь или иное манипулирующее устройство;
- процессор – AMD Ryzen 5 не менее 60 Гц;
- дисплей с разрешением не менее 1920×1080 пикселей.

2 Требования к программному обеспечению:

- тип операционной системы – Windows 10 и выше;
- браузер – Google Chrome 86.0.4240.183 (64-битный) и выше.

А.3 Установка системы

Система поставляется в виде zip-архива. Данный файл необходимо распаковать в любую директорию на жестком диске. Запускаемым файлом системы является файл Maze.exe

А.4 Работа с системой

А.4.1 Авторизация и регистрация в системе

При запуске приложения открывается форма авторизации, чтобы пользователь смог войти в систему под соответствующей ему ролью, она изображена на рисунке А.1.

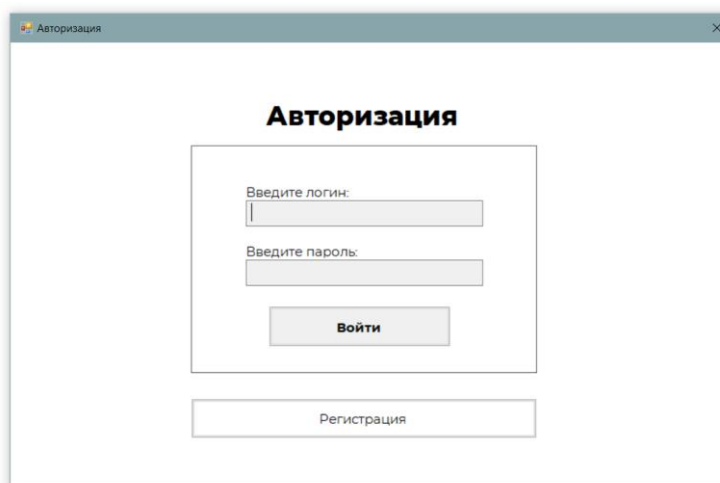
The image shows a screenshot of a software window titled "Авторизация" (Authorization). The window has a light blue header bar with the title and a close button. The main content area is white and contains the following elements: a bold title "Авторизация" at the top center; a group box containing two text input fields, the first labeled "Введите логин:" (Enter login) and the second labeled "Введите пароль:" (Enter password), with a "Войти" (Login) button positioned below them; and a separate "Регистрация" (Registration) button located below the login group box.

Рисунок А.1 – Экранная форма авторизации

Пользователю необходимо заполнить поля для ввода логина и пароля и нажать кнопку «Войти» для входа в систему или кнопку «Регистрация», если пользователь ранее не создавал аккаунт.

Если введен логин от несуществующего в системе аккаунта, то появится ошибка «Логин не найден». Пример этой ошибки изображен на рисунке А.2.

Если ввести неверный пароль от существующего в системе аккаунта, то появится ошибка «Неверный пароль», она изображена на рисунке А.3.

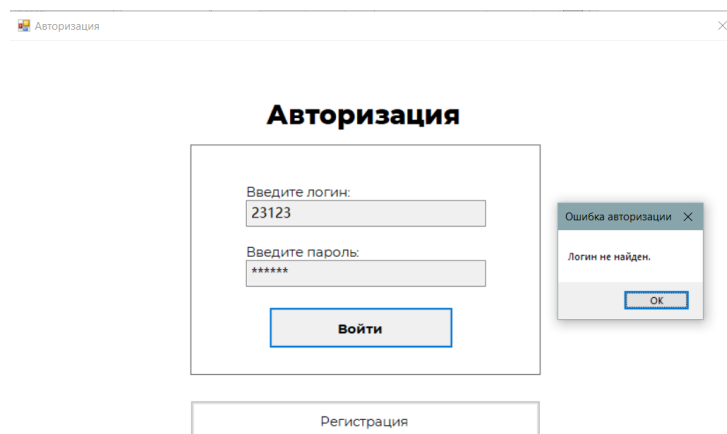


Рисунок А.2 – Пример ошибки «логин не найден»

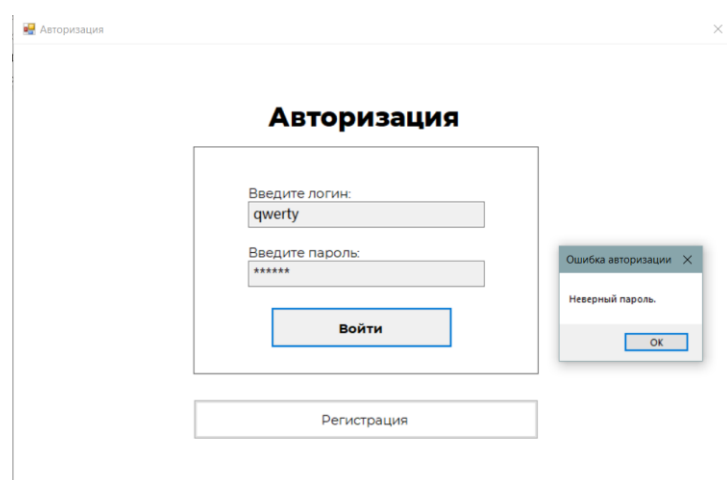


Рисунок А.3 – Пример ошибки «Неверный пароль»

Если пользователь не был ранее зарегистрирован, то при нажатии на кнопку «Регистрация» осуществляется переход на форму регистрации, она изображена на рисунке А.4.

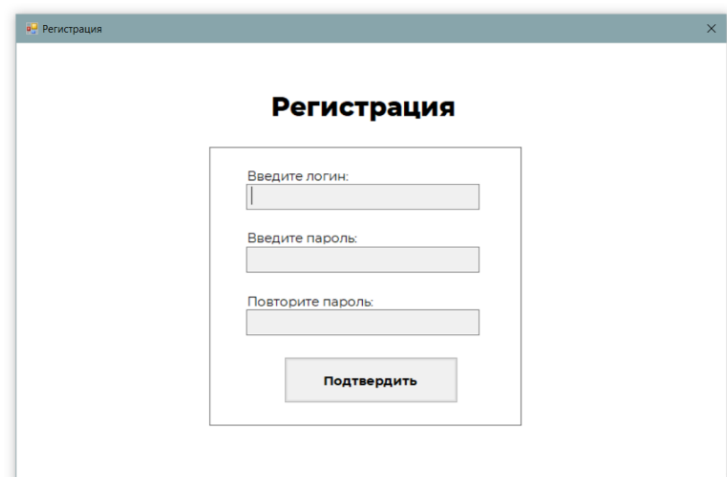


Рисунок А.4 – Экранная форма регистрации

При регистрации пользователю нужно заполнить поля логина, пароля и повтора пароля, затем нажать кнопку «Подтвердить» для перехода на форму авторизации. По умолчанию создается аккаунт для игрока.

Если при регистрации ввести пароль меньше 6 или больше 20 символов, то появится ошибка регистрации, изображенная на рисунке А.5.

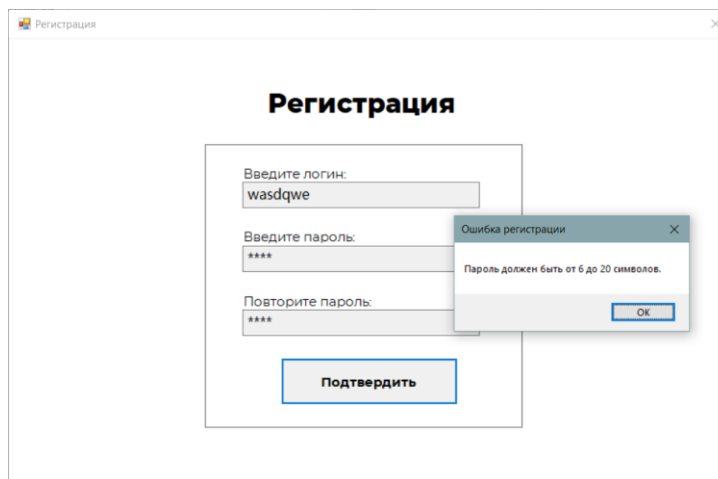


Рисунок А.5 – Пример ошибки «Неверная длина пароля»

Если при регистрации введенные пароли не совпадают, то появится соответствующая ошибка, изображенная на рисунке А.6.

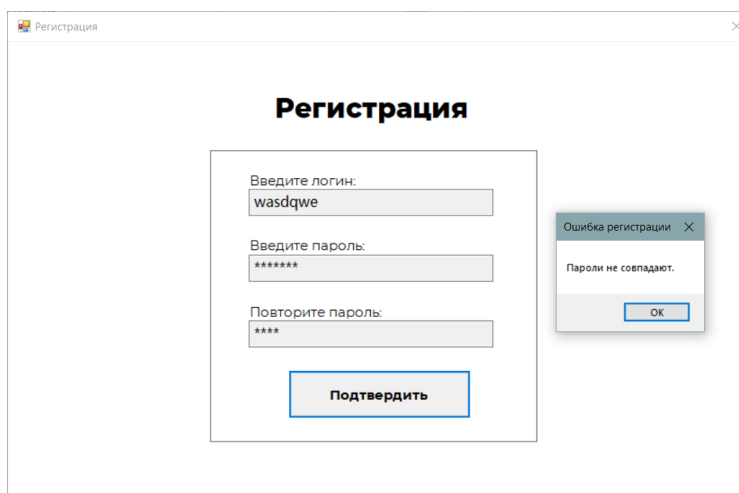


Рисунок А.6 – Пример ошибки «Пароли не совпадают»

А.4.1 Работа с системой в режиме администратора

Если при авторизации ввести данные от аккаунта администратора, то при нажатии на кнопку «Войти» произойдет переход на форму «Admin», в

которой происходит создание лабиринта. На рисунке А.7 приведена экранная форма администратора.

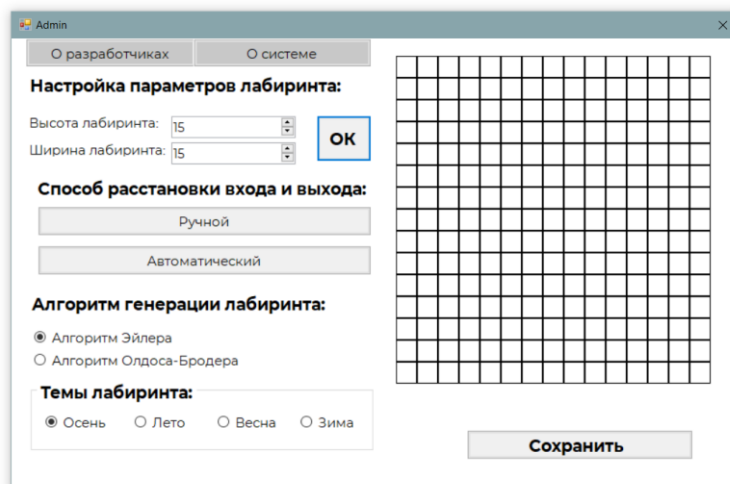


Рисунок А.7 – Экранная форма администратора

Для создания лабиринта администратор должен указать его размеры (ширину и высоту) и подтвердить действие кнопкой «ОК». Значения меньше 7 и больше 21 заменяются этими значениями. Затем в соответствующем поле отобразится сетка указанного размера.

Следующим шагом является выбор метода размещения входа и выхода: вручную или автоматически. Вход и выход должны располагаться исключительно по периметру лабиринта, исключая угловые клетки. Пример неправильного расположения выхода приведен на рисунке А.8.

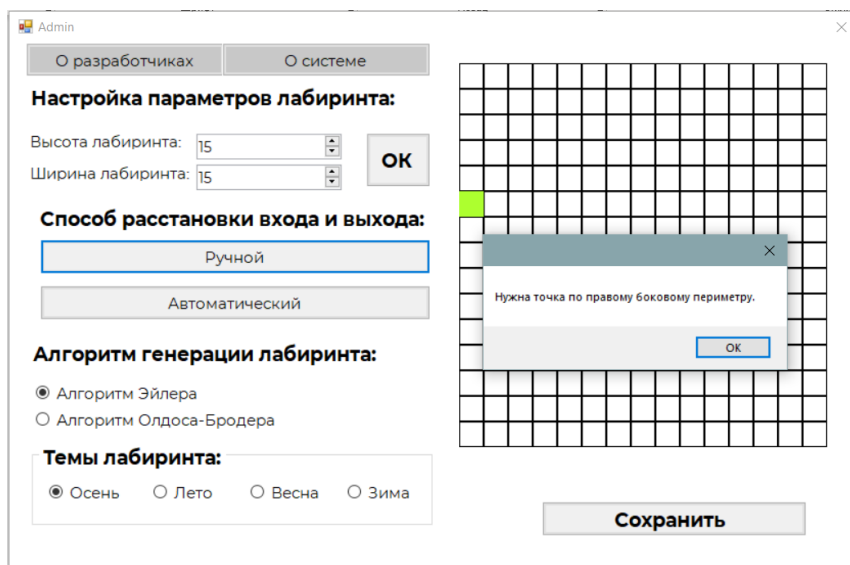


Рисунок А.8 – Пример ошибки неправильного расположения выхода

Администратор также имеет возможность выбрать алгоритм генерации и тему оформления. По завершении настроек, при нажатии на кнопку «Сгенерировать», будет создан лабиринт с заданными параметрами. Пример сгенерированного лабиринта изображен на рисунке А.9.

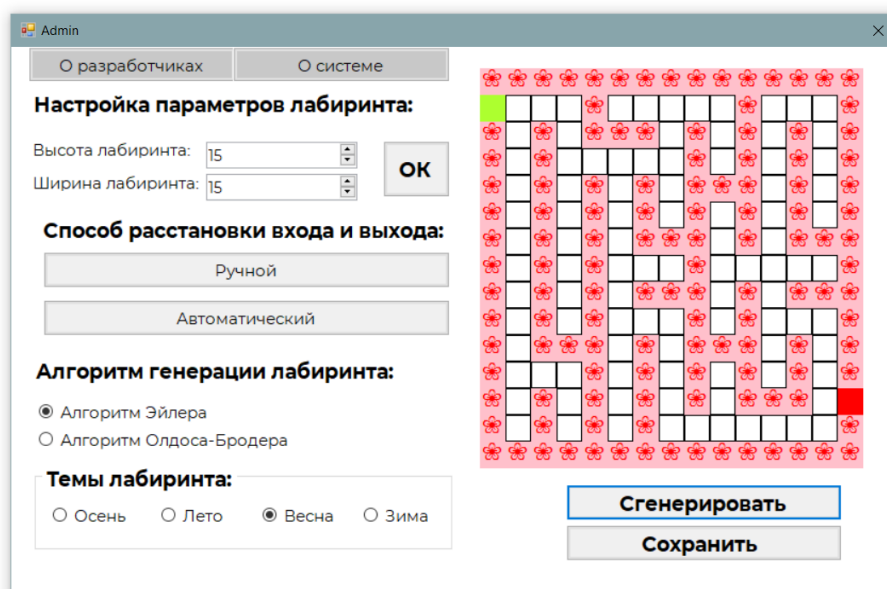


Рисунок А.9 – Пример генерации лабиринта

Администратор может повторно нажать кнопку «Сгенерировать» для получения другого результата или нажать кнопку «Сохранить» для того, что оставить этот вариант лабиринта у себя в системе в файле формата *.xml, задав имя и адрес сохранения.

При нажатии кнопки «О системе» может появиться ошибка, если файл не будет найден. Пример ошибки изображен на рисунке А.10.

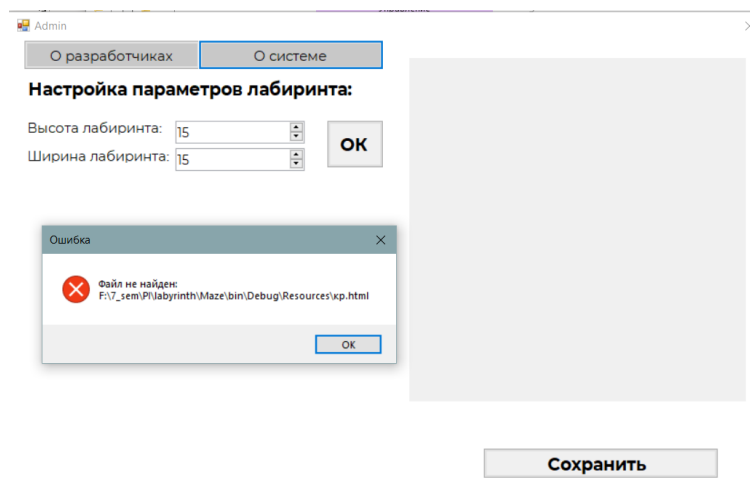


Рисунок А.10 – Пример ошибки «Файл не найден»

А.4.2 Работа с системой в режиме пользователя

Если при авторизации ввести логин и пароль от аккаунта игрока, то откроется форма «Gamer», изображенная на рисунке А.11.



Рисунок А.11 – Форма игрока

Для начала работы нужно загрузить лабиринт из файла, нажав кнопку «Загрузить» и выбрав соответствующий файл. В случае выбора файла неправильного формата или структуры система выдаст сообщение об ошибке. Пример изображен на рисунке А.12.

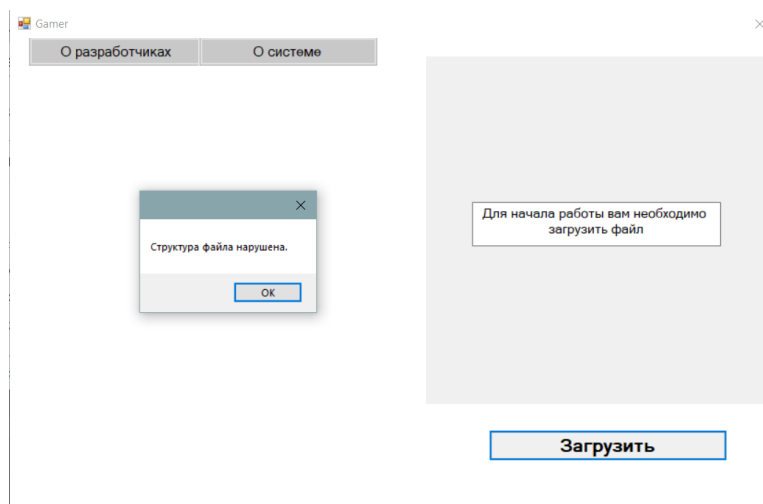


Рисунок А.10 – Пример ошибки «Структура файла нарушена»

Успешная загрузка отображает лабиринт в специальном окне, как показано на рисунке А.13.

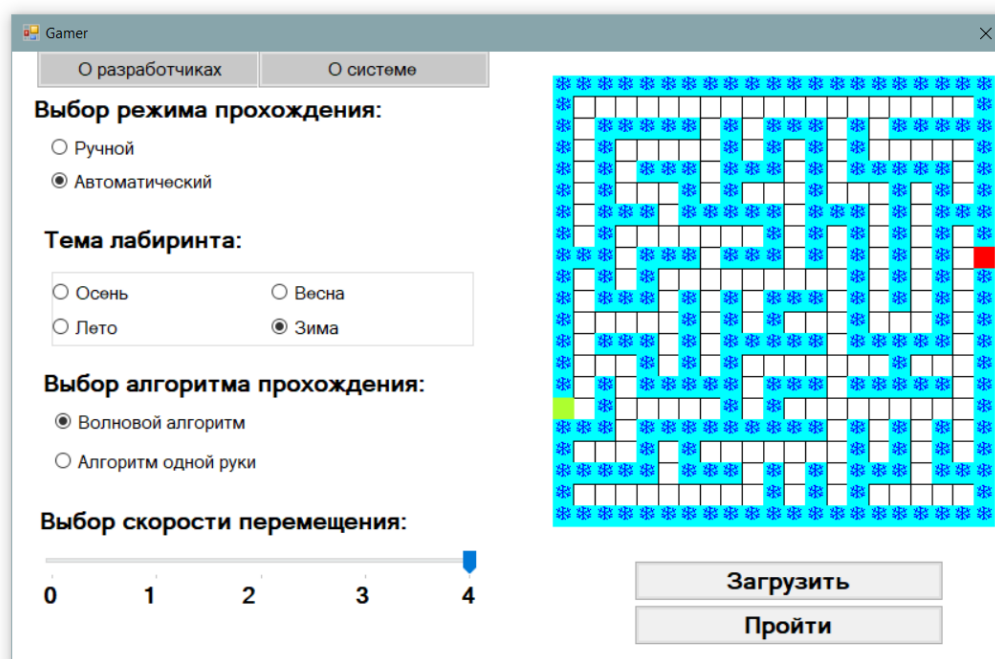


Рисунок А.13 – Настройка параметров игры

Пользователю предоставляется возможность выбрать тему оформления и способ прохождения лабиринта: ручной или автоматический. В ручном режиме достаточно нажать кнопку «Пройти» для старта игры; управление осуществляется клавишами-стрелками на клавиатуре.

В автоматическом режиме пользователь может настроить параметры игры: выбрать алгоритм и скорость прохождения.

Пользователь в любой момент может повторно нажать кнопку «Загрузить» и выбрать другой файл.

Для подтверждения выбора всех настроек и запуска игры необходимо нажать кнопку «Пройти». При прохождении лабиринта настройки параметров игры скрываются от игрока.

ПРИЛОЖЕНИЕ Б

Листинг модулей программы

```
using MazeGenerator;
using Syroot.Windows.IO;
using System;
using System.Collections.Generic;
using System.Drawing;
using System.IO;
using System.Windows.Forms;
using System.Xml;

namespace Maze
{
    public partial class Admin : Form
    {
        private protected enum EStepForm : sbyte
        {
            NOTCREATETEMPLATE = 1,
            CREATEDTEMPLATE,
            BEGINSETPOINTS,
            SETPOINTENTRY,
            ENDSETPOINTS,
            GENERATEDMAZE,
            EXPORTEDMAZE
        }

        /* private StepForm stepForm = StepForm.NOTCREATETEMPLATE;*/
        SolidBrush wallBrush = new SolidBrush(Color.Orange);
        private bool[,] FillWallsArray;
        private Point? startPoint = null;
        private Point? endPoint = null;
        private uint gridWidth;
        private uint gridHeight;
        private bool buttonAutoExit = false;

        private EStepForm _stepForm = EStepForm.NOTCREATETEMPLATE;

        private protected EStepForm StepForm
        {
            get { return _stepForm; }
            set
            {
                _stepForm = value;
                StepFormPropertyChanged();
            }
        }
        private void StepFormPropertyChanged()
        {
            switch (StepForm)
            {
                case EStepForm.NOTCREATETEMPLATE:

                    pictureMaze.Image = null;
                    pictureMaze.BackColor = Color.White;
                    pictureMaze.Invalidate();
                    FillWallsArray = null;
                    startPoint = null;
                    endPoint = null;
                    gridWidth = default;
                    gridHeight = default;
            }
        }
    }
}
```

```

        widthUpDown.Value = 15;
        heightUpDown.Value = 15;

        Generate.Visible = false;

        break;
    case EStepForm.CREATEDTEMPLATE:
        if (buttonAutoExit)
            Generate.Visible = true;
        else
            Generate.Visible = false;
        startPoint = null;
        endPoint = null;
        FillWallsArray = null;
        gridWidth = (uint)widthUpDown.Value;
        gridHeight = (uint)heightUpDown.Value;
        DrawMaze();
        break;
    case EStepForm.BEGINSETPOINTS:
        Generate.Visible = false;

        break;
    case EStepForm.SETPOINTENTRY:
        Generate.Visible = false;
        break;
    case EStepForm.ENDSETPOINTS:
        Generate.Visible = true;
        break;
    case EStepForm.GENERATEDMAZE:
        if (buttonAutoExit)
        {
            (startPoint, endPoint) = RandomStartEndPoints(FillWallsArray);
            DrawMaze();
        }
        break;
    case EStepForm.EXPORTEDMAZE:
        StepForm = EStepForm.NOTCREATETEMPLATE;
        break;
    }
}

public Admin()
{
    InitializeComponent();
}

private void createPattern_Click(object sender, EventArgs e)
{
    settingMazePanel.Visible = true;
    DrawMaze();
    StepForm = EStepForm.CREATEDTEMPLATE;
}

private bool[,] List2DToArray(List<List<bool>> listBoolean)
{
    int rows = listBoolean.Count;
    int cols = listBoolean[0].Count;

    bool[,] array2D = new bool[rows, cols];

    for (int i = 0; i < rows; i++)
    {
        for (int j = 0; j < cols; j++)
        {

```



```

        array2D[i, j] = listBoolean[i][j];
    }
}
return array2D;
}
private void Generate_Click(object sender, EventArgs e)
{
    if (radioButtonEuler.Checked)
    {
        bool[,] maze = List2DToArray(MazeGeneratorEuler.GenerateMaze(gridWidth / 2, gridHeight /
2));
        DrawMaze(maze);
    }
    else if (radioButtonAldousBroder.Checked)
    {
        bool[,] maze = List2DToArray(new MazeGeneratorAldousBroder().Generate((int)gridWidth,
(int)gridHeight));
        DrawMaze(maze);
    }

    StepForm = EStepForm.GENERATEDMAZE;
}
private Tuple<Point, Point> RandomStartEndPoints(bool[,] maze)
{
    int randomRowStart, randomRowEnd;
    Point startPoint, endPoint;
    do
    {
        Random random = new Random();
        randomRowStart = random.Next(0, maze.GetLength(0));
        startPoint = new Point(randomRowStart, 0);

        randomRowEnd = random.Next(0, maze.GetLength(0));
        endPoint = new Point(randomRowEnd, (int)gridWidth - 1);

    } while (randomRowStart <= 0 ||
        randomRowStart >= gridWidth - 1 ||
        randomRowEnd <= 0 ||
        randomRowEnd >= gridWidth - 1 ||
        maze[randomRowStart, 1] == true ||
        maze[randomRowEnd, (int)gridWidth - 2] == true);

    return new Tuple<Point, Point>(startPoint, endPoint);
}
private void DrawMaze(bool[,] mazeMatrix = null)
{
    float cellWidth = (float)Math.Floor((double)pictureMaze.Width / gridWidth);
    float cellHeight = (float)Math.Floor((double)pictureMaze.Height / gridHeight);

    Pen wallPen = new Pen(Color.Black);
    SolidBrush cellBrush = new SolidBrush(Color.White);
    //SolidBrush wallBrush = new SolidBrush(Color.Orange);
    SolidBrush startPointBrush = new SolidBrush(Color.GreenYellow);
    SolidBrush endPointBrush = new SolidBrush(Color.Red);

    FillWallsArray = mazeMatrix is null ? FillWallsArray is null ? null : FillWallsArray : mazeMatrix;
    if (pictureMaze.Image == null || pictureMaze.Image.Width != pictureMaze.Width ||
pictureMaze.Image.Height != pictureMaze.Height)
    {
        if (pictureMaze.Image != null)

```

```

        {
            pictureMaze.Image.Dispose();
        }
        pictureMaze.Image = new Bitmap(pictureMaze.Width, pictureMaze.Height);
    }
    using (Graphics g = Graphics.FromImage(pictureMaze.Image))
    {
        g.Clear(Color.White);
        for (int row = 0; row < gridHeight; row++)
        {
            for (int col = 0; col < gridWidth; col++)
            {
                int x = (int)(col * cellWidth);
                int y = (int)(row * cellHeight);

                int nextX = (int)((col + 1) * cellWidth);
                int nextY = (int)((row + 1) * cellHeight);

                g.FillRectangle(cellBrush, x, y, cellWidth, cellHeight);
                g.DrawRectangle(wallPen, x, y, nextX - x - 1, nextY - y - 1);

                if (FillWallsArray != null && FillWallsArray[row, col] == true)
                    g.FillRectangle(wallBrush, x, y, cellWidth, cellHeight);
            }
        }

        if (startPoint != null)
        {
            int x = (int)(startPoint?.Y * cellWidth);
            int y = (int)(startPoint?.X * cellHeight);
            g.FillRectangle(startPointBrush, x, y, cellWidth, cellHeight);
        }

        if (endPoint != null)
        {
            int x = (int)(endPoint?.Y * cellWidth);
            int y = (int)(endPoint?.X * cellHeight);
            g.FillRectangle(endPointBrush, x, y, cellWidth, cellHeight);
        }
    }

    pictureMaze.Invalidate();
}

private void AdminAboutUs_Click(object sender, EventArgs e)
{
    Form customMessageBox = new Form();
    customMessageBox.Size = new Size(850, 400);
    customMessageBox.Text = "Справочная информация о разработчиках";
    customMessageBox.MaximizeBox = false;
    customMessageBox.MinimizeBox = false;
    customMessageBox.FormBorderStyle = FormBorderStyle.FixedDialog;

    Label label = new Label();
    label.Text = "Самарский университет\nКафедра программных систем\nКурсовой проект по дисциплине «Программная инженерия»\nТема проекта:\n«Автоматизированная система генерирования структуры лабиринта и нахождения выхода из него»\nРазработчики\побучающиеся группы 6402-020302D:\n Балашова Екатерина\n Гриднева Виктория\n\nНаучный руководитель:\nЗеленко Лариса Сергеевна, доцент кафедры ПС\n\n\nСамара 2024";
    label.AutoSize = true;
    label.Font = new Font("Monserat", 11, FontStyle.Bold); // Настройки шрифта
    label.TextAlign = ContentAlignment.MiddleCenter; // Выравнивание текста по центру
    label.Dock = DockStyle.Fill; // Занимает всю доступную площадь
}

```

```

        customMessageBox.Controls.Add(label);

        customMessageBox.ShowDialog();
    }
    private void AdminAboutSys_Click(object sender, EventArgs e)
    {
        string htmlFilePath = $"{Environment.CurrentDirectory}\\admin.html";

        if (File.Exists(htmlFilePath))
        {
            System.Diagnostics.Process.Start(htmlFilePath);
        }
        else
        {
            MessageBox.Show("Файл не найден", "Ошибка", MessageBoxButtons.OK,
MessageBoxIcon.Error);
        }
    }
    private bool SaveMatrixToXml(string filePath)
    {
        XmlWriterSettings settings = new XmlWriterSettings();
        settings.Indent = true;

        using (XmlWriter writer = XmlWriter.Create(filePath, settings))
        {
            writer.WriteStartDocument();
            writer.WriteStartElement("Matrix");

            for (int i = 0; i < FillWallsArray.GetLength(0); i++)
            {
                writer.WriteStartElement("Row");

                for (int j = 0; j < FillWallsArray.GetLength(1); j++)
                {
                    writer.WriteElementString("Cell", FillWallsArray[i, j].ToString());
                }

                writer.WriteEndElement();
            }
            if (startPoint != null)
            {
                writer.WriteElementString("StartPoint", $"{startPoint?.X}:{startPoint?.Y}");
            }
            if (endPoint != null)
            {
                writer.WriteElementString("EndPoint", $"{endPoint?.X}:{endPoint?.Y}");
            }
            writer.WriteEndElement();
            writer.WriteEndDocument();
            MessageBox.Show("Лабиринт успешно сохранён в файл!");
        }
        return true;
    }
    private void saveToFile_Click(object sender, EventArgs e)
    {
        switch (StepForm)
        {
            case EStepForm.NOTCREATETEMPLATE:
                MessageBox.Show("Создайте шаблон!");
                break;
            case EStepForm.CREATEDTEMPLATE:
                MessageBox.Show("Расставьте точки входа и выхода!");
                break;
        }
    }

```

```

case EStepForm.GENERATEDMAZE:
    string applicationPath = AppDomain.CurrentDomain.BaseDirectory;
    string mazesFolderPath = Path.Combine(applicationPath, "Mazes");
    if (!Directory.Exists(mazesFolderPath))
    {
        Directory.CreateDirectory(mazesFolderPath);
    }

    if (FillWallsArray is null || FillWallsArray?.Length == 0)
    {
        MessageBox.Show("Генерация лабиринта не выполнена!");
        break;
    }

    using (var saveFileDialog = new SaveFileDialog())
    {
        saveFileDialog.Filter = "XML файлы (*.xml)|*.xml";
        saveFileDialog.InitialDirectory = mazesFolderPath;

        if (saveFileDialog.ShowDialog() == DialogResult.OK)
        {
            SaveMatrixToXml(saveFileDialog.FileName);
            StepForm = EStepForm.EXPORTEDMAZE;
        }
    }
    break;
}
}
private void pictureMaze_Click(object sender, EventArgs e)
{
    float cellWidth = (float)pictureMaze.Width / gridWidth;
    float cellHeight = (float)pictureMaze.Height / gridHeight;
    MouseEventArgs me = (MouseEventArgs)e;
    int cellRowIndex = Convert.ToInt32(Math.Floor(me.Y / cellHeight));
    int cellColumnIndex = Convert.ToInt32(Math.Floor(me.X / cellWidth));

    switch (StepForm)
    {
        case EStepForm.BEGINSETPOINTS:

            if (cellRowIndex > 0 && cellRowIndex < gridHeight - 1 && cellColumnIndex == 0)
            {
                startPoint = new Point(cellRowIndex, cellColumnIndex);
                DrawMaze(FillWallsArray);
                StepForm = EStepForm.SETPOINTENTRY;
                MessageBox.Show("Входная точка установлена.");
            }
            else
            {
                MessageBox.Show("Нужна точка по левому боковому периметру без угла.");
            }

            break;
        case EStepForm.SETPOINTENTRY:
            if (cellRowIndex > 0 && cellRowIndex < gridHeight - 1 && cellColumnIndex == gridWidth - 1)
            {
                endPoint = new Point(cellRowIndex, cellColumnIndex);
                DrawMaze(FillWallsArray);
                StepForm = EStepForm.ENDSETPOINTS;

                MessageBox.Show("Выходная точка установлена.");
            }
    }
}

```

1)

```

        else
        {
            MessageBox.Show("Нужна точка по правому боковому периметру без угла.");
        }

        break;
    case EStepForm.ENDSETPOINTS:
        MessageBox.Show("Точки входа и выхода расставлены!");
        break;

    }

}

private void handed_Click(object sender, EventArgs e)
{
    buttonAutoExit = false;

    if (gridWidth > 0 && gridHeight > 0)
        StepForm = EStepForm.CREATEDTEMPLATE;
    else
        StepForm = EStepForm.NOTCREATETEMPLATE;
    startPoint = null;
    endPoint = null;
    DrawMaze();

    MessageBox.Show("Установите точку входа и выхода по боковому периметру лабиринта.");
    StepForm = EStepForm.BEGINSETPOINTS;
}

private void nonhanded_Click(object sender, EventArgs e)
{
    buttonAutoExit = true;

    if (gridWidth > 0 && gridHeight > 0)
        StepForm = EStepForm.CREATEDTEMPLATE;
    else
        StepForm = EStepForm.NOTCREATETEMPLATE;
    startPoint = null;
    endPoint = null;
    DrawMaze();
}

private void Gamer_Load(object sender, EventArgs e)
{
    Control.ControlCollection themes = ThemeGroupBox.Controls;

    foreach (RadioButton rdb in themes)
    {
        rdb.MouseUp += ThemeRadioButtons_CheckedChanged;
    }

}

private void ThemeRadioButtons_CheckedChanged(object sender, EventArgs e)
{
    RadioButton radioButton = sender as RadioButton;

    if (radioButton.Checked == true)
    {

```

```

        switch (radioButton.Name)
        {
            case "radioButtonAU":
                wallBrush = new SolidBrush(Color.Orange);
                DrawMaze();
                break;
            case "radioButtonSU":
                wallBrush = new SolidBrush(Color.Green);
                DrawMaze();
                break;
            case "radioButtonSP":
                wallBrush = new SolidBrush(Color.Pink);
                DrawMaze();
                break;
            case "radioButtonWI":
                wallBrush = new SolidBrush(Color.Aqua);
                DrawMaze();
                break;
        }
    }
}

private void heightUpDown_ValueChanged(object sender, EventArgs e)
{
    if (heightUpDown.Value % 2 == 0)
    {
        heightUpDown.Value++;
        //MessageBox.Show("Высота изменена на ближайшее нечётное значение.");
    }
}

private void widthUpDown_ValueChanged(object sender, EventArgs e)
{
    if (widthUpDown.Value % 2 == 0)
    {
        widthUpDown.Value++;
        //MessageBox.Show("Ширина изменена на ближайшее нечётное значение.");
    }
}
}

using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using System.IO;

namespace Maze
{
    public partial class Authorization : Form
    {
        public Authorization()
        {
            InitializeComponent();
        }

        private void Reg_Click(object sender, EventArgs e)
        {

```

```

        Registration registration = new Registration();
        registration.Show();
        this.Hide();
    }

    private void Entry_Click(object sender, EventArgs e)
    {
        string login = AInputLogin.Text;
        string password = AInputPassword.Text;
        string userFilePath = $"{Environment.CurrentDirectory}\\users.txt";
        string adminFilePath = $"{Environment.CurrentDirectory}\\admin.txt";
        if (File.Exists(userFilePath) && File.Exists(adminFilePath))
        {
            var errorType = CheckCredentials(login, password, userFilePath, adminFilePath);
            if (errorType == CredentialError.None)
            {
                if (IsAdmin(login, password, adminFilePath))
                {
                    MessageBox.Show("Вы успешно авторизовались.");
                    Admin adminForm = new Admin();
                    this.Hide();
                    adminForm.Show();
                }
                else
                {
                    MessageBox.Show("Вы успешно авторизовались.");
                    Gamer gamerForm = new Gamer();
                    this.Hide();
                    gamerForm.Show();
                }
            }
            else
            {
                switch (errorType)
                {
                    case CredentialError.LoginNotFound:
                        MessageBox.Show("Логин не найден.", "Ошибка авторизации");
                        break;
                    case CredentialError.IncorrectPassword:
                        MessageBox.Show("Неверный пароль.", "Ошибка авторизации");
                        break;
                }
            }
        }
        else
        {
            MessageBox.Show("Файлы с учетными данными не найдены.", "Ошибка");
        }
    }

    private CredentialError CheckCredentials(string login, string password, string userFilePath, string
adminFilePath)
    {
        // Проверяем, существует ли логин среди пользователей
        bool loginExistsInUsers = File.Exists(userFilePath) && File.ReadAllLines(userFilePath)
            .Any(line => line.Split(':')[0] == login);
        bool loginExistsInAdmins = File.Exists(adminFilePath) && File.ReadAllLines(adminFilePath)
            .Any(line => line.Split(':')[0] == login);
        if (!loginExistsInUsers && !loginExistsInAdmins)
        {
            return CredentialError.LoginNotFound; // Логин не найден
        }
        // Проверяем корректность пароля
        bool isPasswordCorrectForUser = File.Exists(userFilePath) &&

```

```

        File.ReadAllLines(userFilePath)
        .Any(line => line == $"{login}:{password}");
    bool isPasswordCorrectForAdmin = File.Exists(adminFilePath) &&
        File.ReadAllLines(adminFilePath)
        .Any(line => line == $"{login}:{password}");
    if (loginExistsInUsers || loginExistsInAdmins)
    {
        if (isPasswordCorrectForUser || isPasswordCorrectForAdmin)
        {
            return CredentialError.None; // Все корректно
        }
        return CredentialError.IncorrectPassword; // Пароль неверный
    }
    return CredentialError.LoginNotFound;
}
private enum CredentialError
{
    None,
    LoginNotFound,
    IncorrectPassword
}
private bool IsUserCredentialsValid(string login, string password, string userFilePath, string adminFilePath)
{
    if (File.Exists(adminFilePath))
    {
        string[] adminLines = File.ReadAllLines(adminFilePath);
        string credentials = $"{login}:{password}";
        if (adminLines.Any(line => line == credentials))
        {
            return true;
        }
    }
    if (File.Exists(userFilePath))
    {
        string[] userLines = File.ReadAllLines(userFilePath);
        string credentials = $"{login}:{password}";
        if (userLines.Any(line => line == credentials))
        {
            // Это обычный пользователь
            return true;
        }
    }
    return false;
}
private bool IsAdmin(string login, string password, string adminFilePath)
{
    if (File.Exists(adminFilePath))
    {
        string[] adminLines = File.ReadAllLines(adminFilePath);
        string credentials = $"{login}:{password}";
        if (adminLines.Any(line => line == credentials))
        {
            return true;
        }
    }
    return false;
}
}
}

```