Минобрнауки России

Юго-Западный государственный университет

Кафедра программной инженерии

КУРСОВАЯ РАБОТА (ПРОЕКТ)

по дисциплине « Языки объектно-ориентированного программирования »
(наименование дисциплины)

на тему « Приложение для моделирования работы электронной очереди в банке »

Направление подготовки (специальность) 09.03.04
(код, наименование)
Программная инженерия

Автор работы (проекта) Д.Р. Стрелкова _____
(инициалы, фамилия) (подпись, дата)

Группа ПО-32б

Руководитель работы (проекта) Е.И. Аникина _____
(инициалы, фамилия) (подпись, дата)

Работа (проект) защищена _____
(дата)

Оценка _____

Члены комиссии _____ _____
(подпись, дата) (инициалы, фамилия)

_____ _____
(подпись, дата) (инициалы, фамилия)

_____ _____
(подпись, дата) (инициалы, фамилия)

Курск 2024 г.

Минобрнауки России
Юго-Западный государственный университет

Кафедра программной инженерии

ЗАДАНИЕ НА КУРСОВУЮ РАБОТУ (ПРОЕКТ)

Студент Д.Р. Стрелкова (инициалы, фамилия) шифр 23-06-0146 группа ПО-32б

1. Тема Приложение для моделирования работы электронной очереди в банке

2. Срок представления работы (проекта) к защите « » 20 24

3. Исходные данные (*для проектирования, для научного исследования*):

Определяются требованиями пользователя.

4. Содержание пояснительной записки курсовой работы (проекта):

4.1 ВВЕДЕНИЕ

4.2 Анализ и моделирование предметной области приложения

4.3 Техническое задание

4.4 Технический проект

4.5 Рабочий проект

4.6 ЗАКЛЮЧЕНИЕ

4.7 СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

4.8 ПРИЛОЖЕНИЕ А

4.9 ПРИЛОЖЕНИЕ Б

5. Перечень графического материала: диаграмма прецедентов, диаграмма классов, внешний вид интерфейса, макет интерфейса, таблицы тестовых наборов, таблица описания объектов интерфейса.

Руководитель работы (проекта) Е.И. Аникина (инициалы, фамилия) _____ (подпись, дата)

Задание принял к исполнению Д.Р. Стрелкова (инициалы, фамилия) _____ (подпись, дата)

РЕФЕРАТ

Данный текстовый документ имеет объем 110 страниц и включает в себя 52 рисунка, 8 таблиц, 2 приложения, 15 библиографических источников.

Перечень ключевых слов: банк, терминал, инструкция, тип операции, талон, электронная очередь, экран, время ожидания, окно, обслуживание.

Целью работы является программная реализация на языке C# приложения для моделирования работы электронной очереди. Программный продукт предназначен для улучшения понимания работы электронной очереди в банке.

При создании программного продукта с локальной архитектурой применялись технология объектно-ориентированного программирования.

ABSTRACT

This text document has a volume of 110 pages and includes 52 figures, 8 tables, 2 appendices, 15 bibliographic sources.

The list of keywords: bank, terminal, instruction, type of operation, coupon, electronic queue, screen, waiting time, window, service.

The purpose of the work is a software implementation in C# of an application for modeling the operation of an electronic queue. The software product is designed to improve the understanding of the electronic queue in the bank.

When creating a software product with a local architecture, object-oriented programming technology was used.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ..... 7

1. Анализ и моделирование предметной области приложения 8

1.1. Описание предметной области 8

1.2. Моделирование вариантов использования 12

1.2.1. Варианты использования..... 12

1.2.2. Диаграмма прецедентов 13

1.2.3. Сценарии прецедентов программного изделия 15

1.2.4. Словарь предметной области приложения..... 20

1.2.5. Моделирование предметной области с помощью диаграммы классов……..22

2. Техническое задание 23

2.1. Основание для разработки 23

2.2. Назначение разработки 23

2.3. Требования к программе или программному изделию..... 23

2.3.1. Требования к функциональным характеристикам 23

2.3.2. Требования к входным и выходным данным..... 24

2.3.3. Требования пользователя к интерфейсу 24

2.3.4. Требования к надежности 31

2.3.5. Условия эксплуатации..... 31

2.3.6. Требования к составу и параметрам технических средств..... 31

2.3.7. Требования к информационной и программной совместимости. 31

2.4. Требования к программной документации..... 32

2.5. Стадии и этапы разработки..... 32

2.6. Порядок контроля и приемки 32

3. Технический проект 34

3.1. Моделирование последовательности действий 34

3.2. Проектирование архитектуры программной системы 35

3.3. Описание структур и форматов данных 35

3.4. Схемы алгоритмов 35

4. Рабочий проект 36

4.1. Спецификация компонентов и классов программ 36

4.1.1. Модули и объекты интерфейса пользователя 36

4.1.2. Описание объектов интерфейса программы 36

4.1.3. Описание классов 46

4.2. Тестирование программной системы 50

ЗАКЛЮЧЕНИЕ 78

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 79

ПРИЛОЖЕНИЕ А 82

ПРИЛОЖЕНИЕ Б 108

ВВЕДЕНИЕ

Целью курсовой работы является создание на языке C# компьютерного приложения для моделирования работы электронной очереди в банке.

Для достижения поставленной цели необходимо решить следующие задачи:

1. Разработать архитектуру приложения.
2. Разработать интерфейс приложения.
3. Разработка функции обработки данных.
4. Реализация хранения данных.
5. Реализация алгоритма генерации номеров очереди.
6. Возможность отслеживания своего места в очереди по номеру талона.
7. Реализация алгоритма вызова клиентов.
8. Реализация системы оповещения клиентов.
9. Спроектировать работу электронной очереди в банке.
10. Реализация прекращения работы приложения.

Основные результаты. В ходе разработки получены следующие результаты:

1. Была разработана архитектура приложения.
2. Был разработан интерфейс приложения.
3. Была разработана функция обработки данных.
4. Было реализовано хранение данных.
5. Был реализован алгоритм генерации номеров очереди.
6. Была реализована возможность отслеживания своего места в очереди по номеру талона.
7. Был реализован алгоритм вызова клиентов.
8. Была реализована система оповещения клиентов.
9. Была спроектирована работа электронной очереди в банке.
10. Реализовано прекращение работы приложения.

1. Анализ и моделирование предметной области приложения

1.1. Описание предметной области

Приложение разрабатывается с целью показать работу электронной очереди в банке. Приложение для моделирования электронной очереди в банке представляет собой программное обеспечение, имитирующее работу реальной системы обслуживания клиентов.

Пользователь попадает на стартовую форму. Пользователь (клиент банка) может начать обслуживание в Str-Банке при нажатии на кнопку.

После нажатия на кнопку, пользователь перемещается на форму, на которой отображается клиент идущий в банк. Клиент входит в банк и на следующей форме он видит терминал.

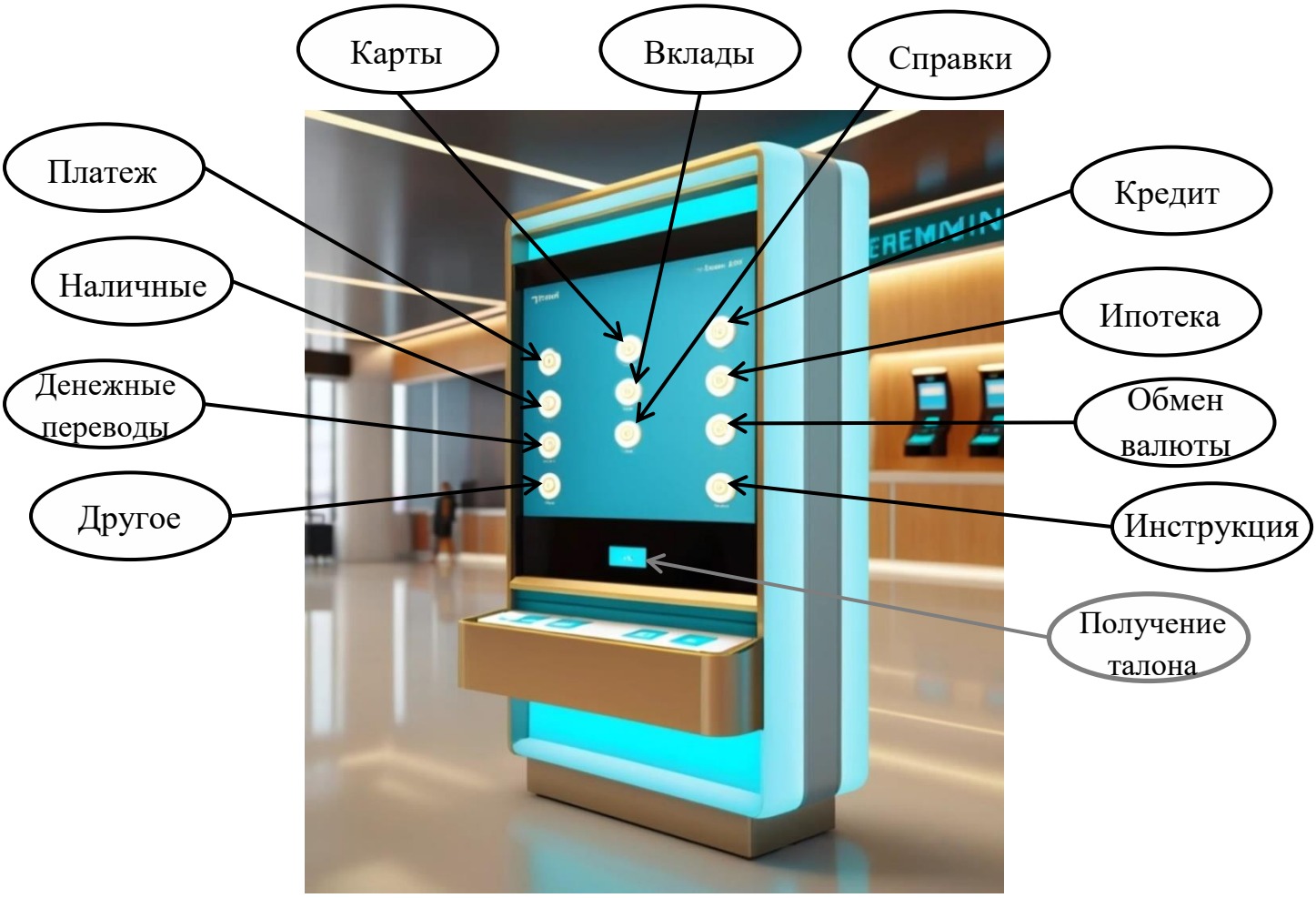


Рисунок 1.1 – Компоненты терминала.

Экран терминала содержит:

- 10 кнопок: Каждая кнопка соответствует определенному типу операции, и пользователь может выбрать нужную ему.
- Кнопка "Инструкция": Предоставляет возможность клиенту ознакомиться с руководством по использованию терминала в случае отсутствия понимания или навыков работы с устройством.

Клиент может воспользоваться функцией "Инструкция" для получения информации о том, как правильно работать с терминалом, если у него возникли какие-либо затруднения. Для этого на терминале надо нажать кнопку «Инструкция».

Клиент может выбрать одну из следующих операций, нажав соответствующую кнопку:

- Платежи (категория А): Позволяет клиенту оплачивать коммунальные услуги, штрафы и другие обязательные платежи.
- Наличные (категория Б): Возможность снять или внести наличные деньги на счет.
- Денежные переводы (категория В): Клиент может переводить деньги между счетами или знакомым.
- Карты (категория Г): Оформление кредитной или дебетовой карты, активация или блокировка карты, перевыпуск карты;
- Вклады (категория Д): Открытие нового вклада, проверка состояния текущих вкладов.
- Справки (категория Е): Получение информации о состоянии счета, процентах по вкладам и предыдущих операциях.
- Кредит (категория Ж): Подача заявки на кредит или просмотр статуса действующих кредитов.
- Ипотека (категория З): Запрос на получение ипотеки, работа с условиями и документами.

- Обмен валюты (категория И): Возможность обмена одной валюты на другую с актуальными курсами.
- Другое (категория К): Служит для доступа к нестандартным операциям или услугам.

После выбора одного из этих типов операций клиентом, система обрабатывает запрос, автоматически генерирует уникальный номер талона и ставит клиента в электронную очередь.

После успешной обработки пользователь перемещается на форму, где получает созданный талон с уникальным номером и видит его.

После получения талона пользователь перемещается на следующую форму, в зону ожидания. В ней расположены мониторы. На этих экранах отображается:

- Номер талона клиента;
- Текущая позиция в очереди;
- Ожидаемое время обслуживания для каждого клиента, что позволяет клиенту примерно оценить, когда его вызовут.

Внутри системы осуществляется управление порядком вызова клиентов. Система фиксирует время обслуживания каждого клиента, время ожидания для всех клиентов в очереди.

Когда подходит очередь для обслуживания клиента, система автоматически генерирует уведомление, которое отображается на экране в зоне ожидания. Уведомление отображается на экране в зоне ожидания, обеспечивая видимость для всех клиентов. Уведомление содержит:

- Номер талона клиента;
- Номер окна, к которому клиент должен пройти.

Когда клиент видит свой индивидуальный номер талона на экране, он направляется в указанное окно, где производится его обслуживание. После завершения процесса обслуживания клиент покидает банк, удовлетворенный предоставленной услугой.

На любом этапе клиент может начать процесс заново, получив новый талон и очередное место в системе. Это позволяет клиенту при необходимости изменять свои предпочтения или повторно запросить ту же услугу.

Так же на любом этапе приложения можно будет закончить обслуживание и прекратить работу проекта.

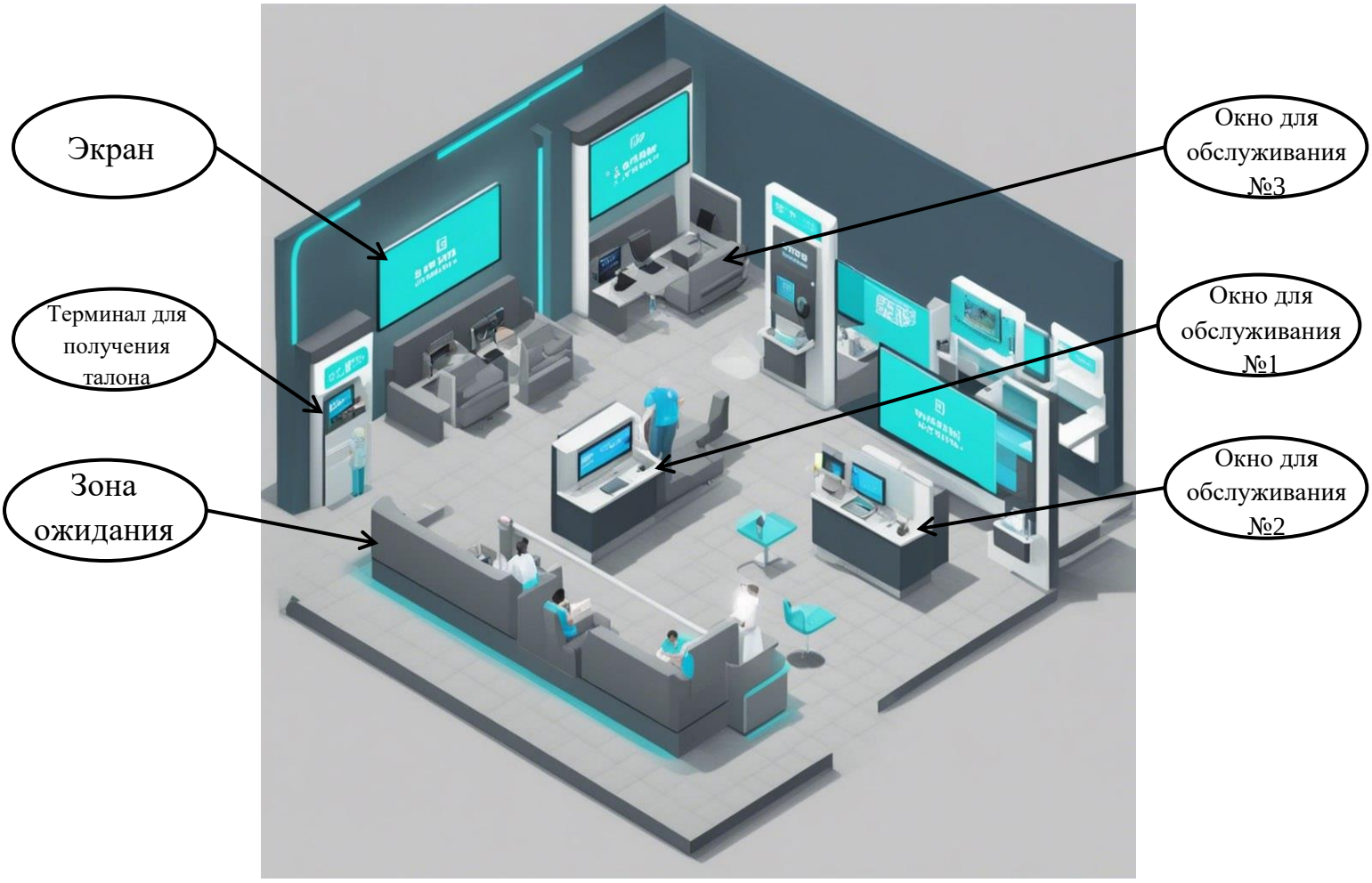


Рисунок 1.2 – Компоненты банка.

1.2. Моделирование вариантов использования

1.2.1. Варианты использования

Приложение должно моделировать следующие ситуации:

1. Начало обслуживания. Пользователь попадает на стартовую форму и нажимает кнопку «Начать обслуживание в Str-банке», что активирует процесс работы с приложением.
2. Пользователь видит, как клиент идет в банк. Клиент заходит в Str-банк и попадает на следующую форму с терминалом.
3. Клиент видит на экране терминала кнопки с различными типами операций, и кнопку для изучения инструкции.
4. Клиент может изучить инструкцию, чтобы понять, как работает терминал.
5. Пользователь выбирает тип операции и нажимает на него.

   Клиентские операции:

   - Платежи;
   - Наличные;
   - Денежные переводы;
   - Карты;
   - Вклады;
   - Справки;
   - Кредит;
   - Ипотека;
   - Обмен валюты;
   - Другое.
6. Система обрабатывает запрос, автоматически генерирует уникальный номер талона и ставит клиента в электронную очередь.

7. Пользователь получает талон с индивидуальным номером, который фиксируется в электронной очереди.
8. Пользователь проходит в зону ожидания. На экране отображается информация о текущей очереди и приблизительное время ожидания.
9. Система отправляет визуальный сигнал на экран, чтобы привлечь внимание пользователя с сообщение о том, что очередь клиента подошла.
10. Клиент понимает, что его время ожидания завершилось, и он может продолжить процесс обслуживания.
11. Клиент проходит к указанному на экране окну, где его готов обслужить сотрудник банка.
12. При начале обслуживания клиент получает уведомление.
13. При завершении обслуживания клиент получает уведомление.
14. После завершения обслуживания клиент покидает банк, и получает уведомление о выходе из Str-Банка.
15. На любом этапе использования приложения клиент может в любой момент вернуться к стартовому экрану.
16. На любом этапе использования приложения клиент может завершить обслуживание и прекратить работу проекта.

1.2.2. Диаграмма прецедентов

В системе должен быть представлен один действующий пользователь.

В программе должны быть реализованы следующие прецеденты:

1. Прецедент «Начать обслуживание». Позволяет пользователю начать обслуживание и зайти в Str-банк.
2. Прецедент «Просмотр инструкции». Позволяет пользователю изучить инструкцию, если клиенту не понятен принцип работы терминала.

3. Прецедент «Выбор типа операции». Позволяет пользователю на терминале выбрать тип операции.
4. Прецедент «Сгенерировать номер талона и поставить его в электронную очередь». Позволяет системе после выбора типа операции клиентом, сгенерировать индивидуальный номер талона и занести его в электронную очередь.
5. Прецедент «Получение талона». Позволяет клиенту получить талон с индивидуальным номером.
6. Прецедент «Отслеживание позиции в очереди». Позволяет пользователю отслеживать свою позицию в очереди по индивидуальному номеру талона, используя информационный экран.
7. Прецедент «Вызов клиента». Позволяет пользователю увидеть, что система вызывает его на обслуживание, узнать к какому окну ему нужно пройти.
8. Прецедент «Обслуживание клиента». Позволяет пользователю получить обслуживание и уйти из банка.
9. Прецедент «Начать обслуживание заново». Позволяет пользователю на любом этапе приложения начать обслуживание в банке заново.
10. Прецедент «Завершить работу приложения». Прецедент позволяет пользователю закончить обслуживание на любом этапе и прекратить работу проекта.

На рисунке 1.3 представлена диаграмма прецедентов для программы.

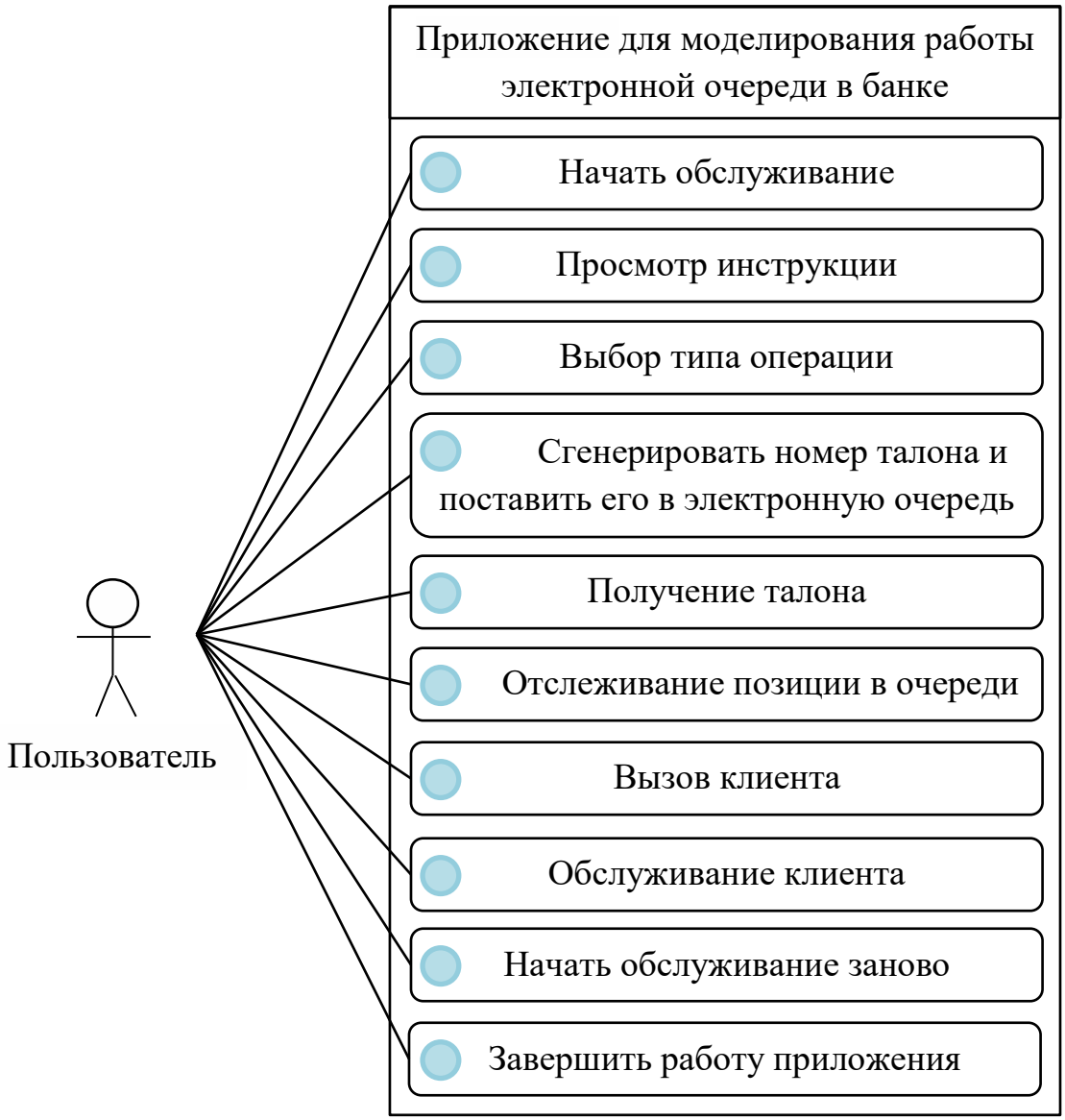


Рисунок 1.3 – Диаграмма прецедентов.

1.2.3. Сценарии прецедентов программного изделия

Сценарий для прецедента «Начать обслуживание»

Основной исполнитель: пользователь (клиент банка).

Заинтересованные лица и их требования: Пользователь хочет начать процесс обслуживания в Str-банке и зайти в него.

Основной успешный сценарий:

1. Пользователь активирует начало обслуживания нажатием на стартовой форме на кнопку «Начать обслуживание в Str-банке.».
2. Система отображает форму с клиентом идущим в банк.
3. Пользователь заходит в банк.

Сценарий для прецедента «Просмотр инструкции»

Основной исполнитель: пользователь (клиент банка).

Заинтересованные лица и их требования: Пользователь хочет изучить инструкцию по работе с терминалом.

Основной успешный сценарий:

1. Система отображает форму с терминалом.
2. Пользователь нажимает кнопку «Инструкция» на терминале.
3. Система отображает следующую форму с текстовой подсказкой с объяснением работы терминала и операциями, которые можно выполнить.
4. Пользователь имеет возможность вернуться к основному экрану.

Сценарий для прецедента «Выбор типа операции»

Основной исполнитель: пользователь (клиент банка).

Заинтересованные лица и их требования: Пользователь хочет выбрать тип операции.

Основной успешный сценарий:

1. Система отображает форму с терминалом для получения талона.
2. Пользователь может выбрать нужный ему тип операции из 10 представленных (Платежи, наличные, денежные переводы, карты, вклады, справки, кредит, ипотека, обмен валюты, другое).
3. Пользователь выбирает нажатием на кнопку нужный тип операции.

4. Терминал получает данные о выбранном типе операции клиентом и создает индивидуальный номер для клиента.
5. Терминал выдает талон с индивидуальным номером клиенту.

Сценарий для прецедента «Сгенерировать номер талона и поставить его в электронную очередь»

Основной исполнитель: система.

Заинтересованные лица и их требования: Система хочет сгенерировать индивидуальный номер талона и поставить его в электронную очередь, после выбора типа операции.

Основной успешный сценарий:

1. Пользователь выбирает тип операции на терминале.
2. Система генерирует уникальный номер талона.
3. Номер талона заносится в электронную очередь.
4. Номер талона появляется на экране.

Сценарий для прецедента «Получение талона»

Основной исполнитель: пользователь (клиент банка).

Заинтересованные лица и их требования: Пользователь хочет получить талон с индивидуальным номером.

Основной успешный сценарий:

1. После выбора типа операции и генерации номера талона, система печатает талон с уникальным номером.
2. Пользователь получает талон и проходит в зону ожидания.

Сценарий для прецедента «Отслеживание позиции в очереди»

Основной исполнитель: пользователь (клиент банка).

Заинтересованные лица и их требования: Пользователь хочет отслеживать свою позицию в очереди и примерное время ожидания по индивидуальному номеру талона, используя информационный экран в банке.

Основной успешный сценарий:

1. Система отображает форму, где клиент находится в зоне ожидания.
2. Система отображает текущую позицию пользователя в очереди и ожидаемое время ожидания на экране.
3. Пользователь смотрит на информационный экран.
4. На экране клиент находит свой индивидуальный номер талона, и узнает свою позицию в очереди и примерное время ожидания.
5. Клиент узнал нужную ему информацию.

Сценарий для прецедента «Вызов клиента»

Основной исполнитель: система.

Заинтересованные лица и их требования: Система предоставляет клиенту информацию о вызове его на обслуживание, включая указание на соответствующее окно для обращения.

Основной успешный сценарий:

1. Когда номер талона клиента подходит к концу электронной очереди, система отображает сообщение на экране.
2. На экране появляется номер клиента и окно, к которому ему следует пройти, чтобы получить нужное обслуживание.
3. Некоторое время на экране подается визуальный сигнал, чтобы клиент это заметил.
4. Пользователь переходит к указанному окну для получения услуги.

Сценарий для прецедента «Обслуживание клиента»

Основной исполнитель: пользователь (клиент банка).

Заинтересованные лица и их требования: Пользователь хочет получить обслуживание.

Основной успешный сценарий:

1. Клиент подходит к окну, указанному системой.
2. Сотрудник банка предоставляет клиенту необходимые услуги, связанные с выбранной операцией.
3. По завершении обслуживания клиент покидает банк, и система обновляет статус электронной очереди.

Сценарий для прецедента «Начать обслуживание заново»

Основной исполнитель: пользователь (клиент банка).

Заинтересованные лица и их требования: Пользователь хочет вернуться в начало процесса обслуживания в банке.

Основной успешный сценарий:

1. Пользователь на любом этапе приложения нажимает кнопку «Начать обслуживание заново».
2. Система отображает подтверждение о том, что все введенные данные и текущие процессы будут сброшены.
3. Пользователь подтверждает намерение начать процесс заново.
4. Система возвращает пользователя на стартовую форму приложения, готового к новому обслуживанию.

Сценарий для прецедента «Завершить работу приложения»

Основной исполнитель: пользователь.

Заинтересованные лица и их требования: Пользователь хочет завершить работу приложения и прекратить работу проекта.

Основной успешный сценарий:

1. Пользователь на любом этапе рабочего процесса нажимает кнопку «Завершить работу приложения».
2. Система отображает сообщение с запросом о подтверждении завершения работы.
3. Пользователь подтверждает свое желание завершить работу.
4. Система сохраняет все данные в базе данных.
5. Система закрывает приложение и прекращает работу программы.

1.2.4. Словарь предметной области приложения

На основе анализа описания предметной области были определены основные понятия и разработан словарь предметной области, представленный в таблице 1.

Термин	Определение
Str-Банк	Финансовое учреждение, где клиент получает нужное обслуживание.
Клиент	Физическое лицо, которое приходит в Str-Банк для получения услуги.
Терминал	Устройство, где клиент выбирает нужный ему тип операции и получает талон с индивидуальным номером.
Тип операции	Вид услуги, которую клиент хочет получить (Платежи, карты, вклады, наличные, денежные переводы, кредит, ипотека, справки, обмен валюты).
Инструкция	Документ, который содержит подробные указания, рекомендации и правила по выполнению определённых действий.

Термин	Определение
Электронная очередь	Система, которая автоматизирует процесс обслуживания клиентов в банке, управляя порядком очереди и информируя клиентов о своей очереди. Она включает в себя терминал для получения индивидуального номера талона, талон, оповещение на экране.
Талон	Документ, выдаваемый клиенту в бумажном виде с индивидуальным номером, определяющим его позицию в очереди.
Индивидуальный номер талона	Уникальный идентификатор, присваиваемый клиенту при получении талона. Индивидуальный номер талона используется для отслеживания позиции клиента в очереди и для оповещения о его очереди.
Время ожидания	Промежуток времени, который клиент проводит в зоне ожидания до момента обслуживания.
Экран	Электронное устройство, отображающее информацию о текущей позиции клиента и информацию о времени ожидания.
Визуальный сигнал	Сигнал, оповещающий клиента о его очереди.
Окно	Рабочее место оператора, где происходит обслуживание клиента.
Оператор	Обслуживает клиента.

Таблица 1 – Словарь предметной области.

1.2.5. Моделирование предметной области с помощью диаграммы классов

На основе анализа словаря предметной области приложения и описания вариантов использования были определены классы, атрибуты и методы классов и отношения классов, представленные на рисунке 1.4 в виде диаграммы концептуальных классов.

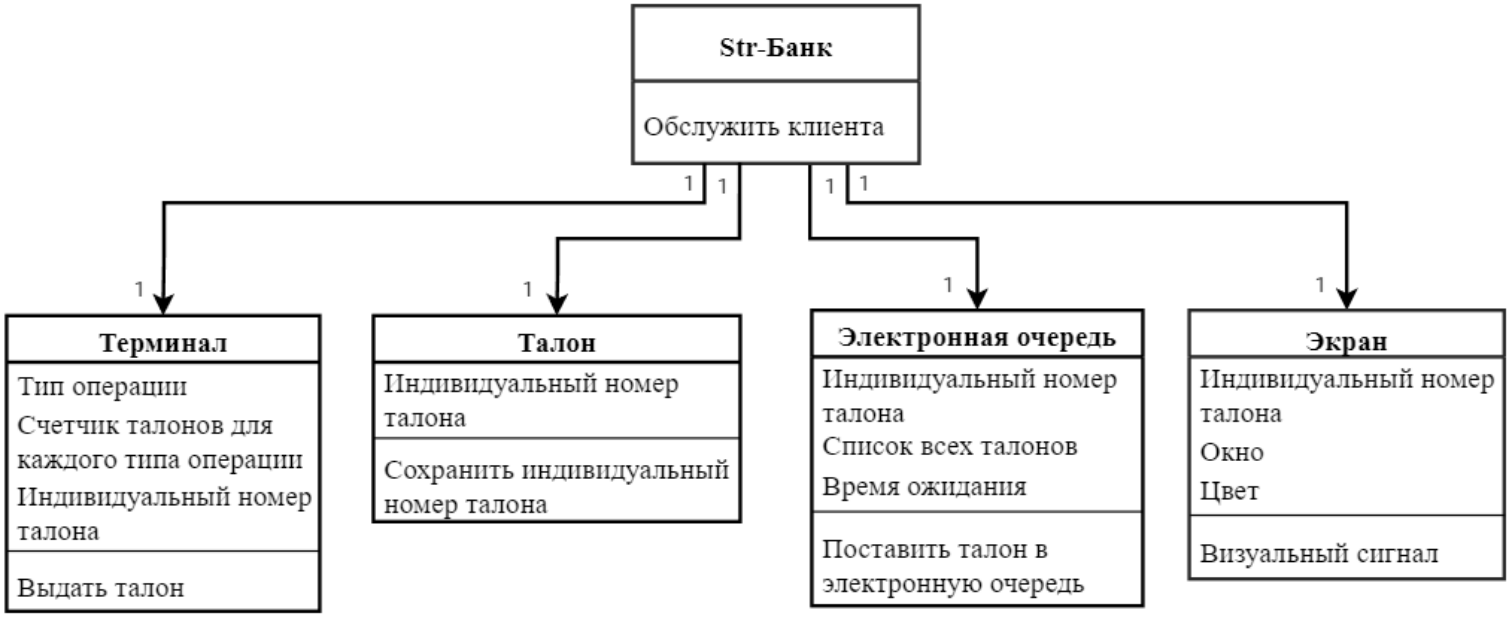


Рисунок 1.4 – Диаграмма концептуальных классов предметной области приложения.

2. Техническое задание

2.1. Основание для разработки

Основанием для разработки программного продукта служит задание по курсовой работе по дисциплине «Языки объектно-ориентированного программирования».

2.2. Назначение разработки

Программный продукт предназначен для улучшения понимания работы электронной очереди в банке.

2.3. Требования к программе или программному изделию

2.3.1. Требования к функциональным характеристикам

В программном продукте должен быть представлен один пользователь.

В разрабатываемом программном продукте для пользователя должны быть реализованы следующие функциональные требования.

1. Интуитивно понятный и приятный интерфейс для пользователя, для приложения для моделирования работы электронной очереди в банке.
2. Пользователь может начать обслуживание в банке.
3. Пользователь может ознакомиться с инструкцией на терминале.
4. Пользователь может ознакомиться с типами операций на терминале.
5. Пользователь может выбрать тип операции на терминале.
6. Пользователь может получить талон с индивидуальным номером.
7. Пользователь может отслеживать на экране свое положение в электронной очереди, а так же время ожидания.

8. Пользователь может увидеть визуальный сигнал на экране, и понять, что подошла его очередь.
9. Пользователь может пройти к окну обслуживания, и получить нужную помощь.
10. Пользователь может увидеть, что обслуживание в банке начато.
11. Пользователь может увидеть, что обслуживание в банке завершено.
12. Пользователь может увидеть, что вышел из банка.
13. Пользователь может начать обслуживание в банке заново.
14. Пользователь может завершить работу приложения.

2.3.2. Требования к входным и выходным данным

Формирование входных данных происходит через нажатие компьютерной мышки.

2.3.3. Требования пользователя к интерфейсу

Для работы пользователя с приложением был разработан интерфейс, со всеми необходимыми окнами для пользователя. Макет интерфейса представлен на рисунках 2.1-2.6.

Слева снизу 1 окна приложения расположена кнопка для управления приложением «Начать обслуживание в Str-Банке.» для начала обслуживания. Вверху окна расположено поле для отображения приветствия приложения.

Рисунок 2.1 - Макет интерфейса «Начальное окно приложения».

На 2 окне приложения, с помощью таймера, графически изображено как клиент Str-Банка идет по улице и заходит вовнутрь здания Str-Банка.

Рисунок 2.2 - Макет интерфейса «Окно приложения «Путь в банк»».

На 3 окне приложения расположены 13 кнопок для управления приложением. По центру окна находится 11 кнопок: 10 кнопок с типами операций на терминале для получения талона («Платежи», «Наличные», «Денежные переводы», «Карты», «Вклады», «Справки», «Кредит», «Ипотека», «Обмен валюты», «Другое»); 1 кнопка – «Инструкция». В верхнем правом углу находится 2 кнопки: «Начать обслуживание заново» – для того, чтобы начать процесс обслуживания с начального окна приложения; «Закрыть приложение» – для закрытия приложения и прекращения работы проекта.

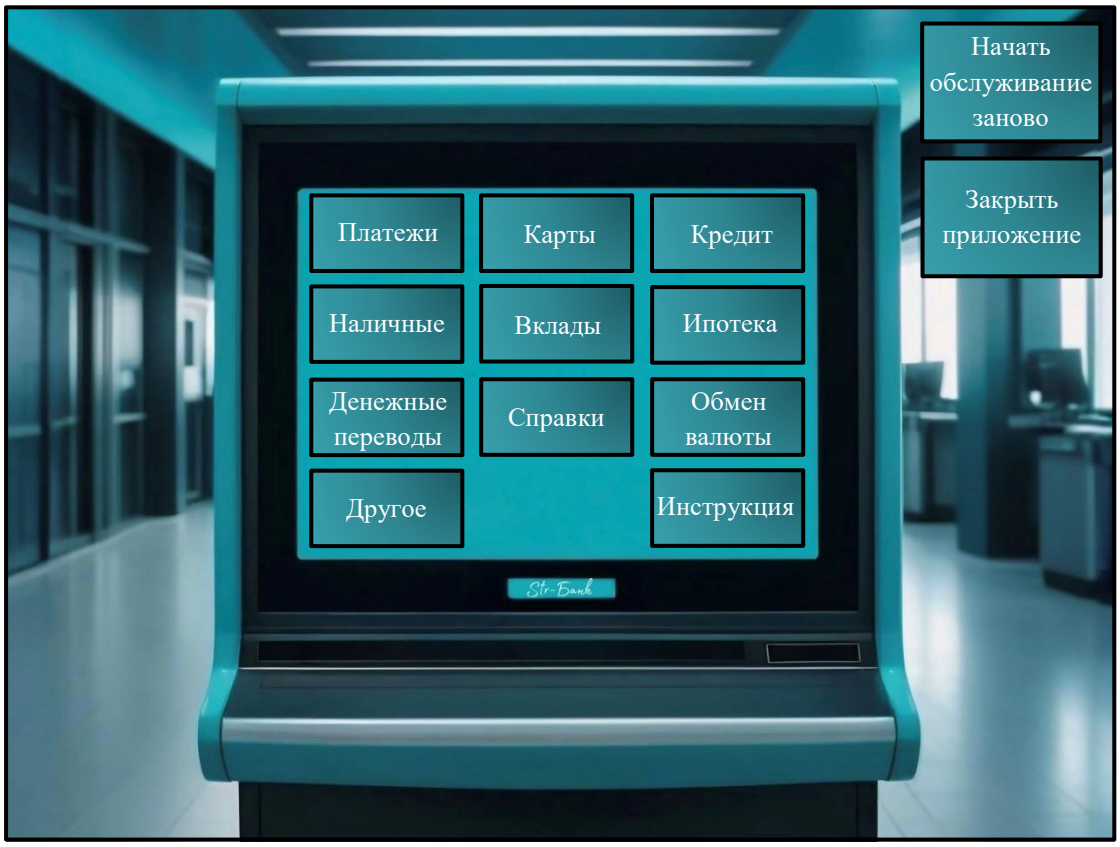


Рисунок 2.3 - Макет интерфейса «Главное окно приложения».

На 4 окне приложения слева расположено поле для отображения инструкции, и 2 кнопки в правом верхнем углу: «Начать обслуживание заново» – для того, чтобы начать процесс обслуживания с начального окна приложения; «Закрыть приложение» – для закрытия приложения и прекращения работы проекта.

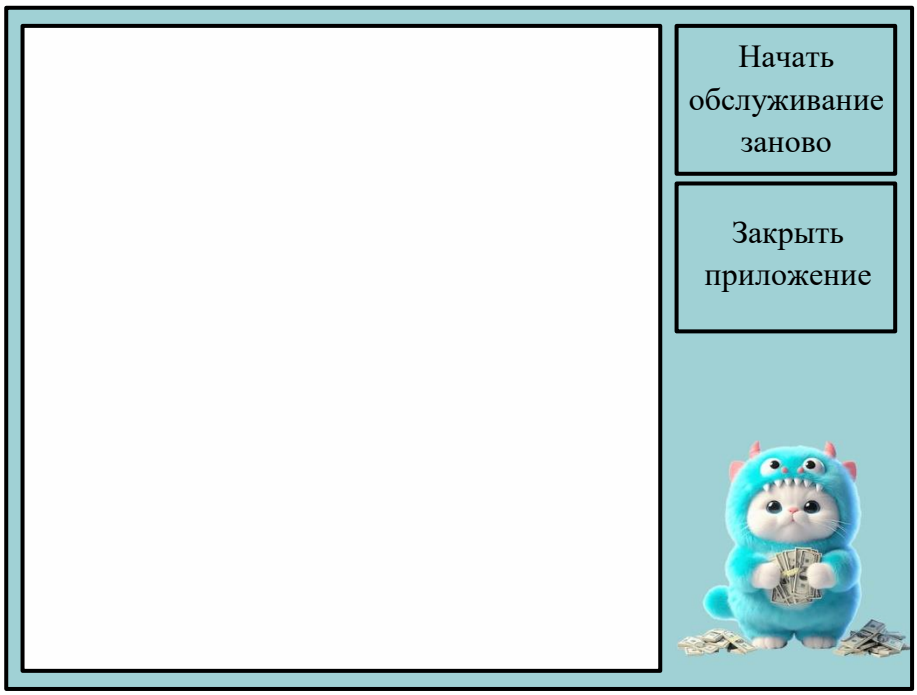


Рисунок 2.4 - Макет интерфейса «Окно приложения «Инструкция»».

На 5 окне приложения расположено поле для отображения индивидуального номера талона, и 2 кнопки в правом верхнем углу: «Начать обслуживание заново» – для того, чтобы начать процесс обслуживания с начального окна приложения; «Закрыть приложение» – для закрытия приложения и прекращения работы проекта.

Рисунок 2.5 - Макет интерфейса «Окно приложения «Получение талона»».

На 6 окне приложения расположено 4 поля для отображения информации: 1, 2 и 3 поля – для просмотра информации о примерном времени ожидания и количестве ожидающих клиентов, по категориям (1 поле – категории А, Б, В; 2 поле – категории Г, Д, Е; 3 поле – категории Ж, З, И, К). 4 поле – для того, чтобы узнать с помощью визуального сигнала, к какому окну надо пройти, когда подойдет время ожидания. Так же с помощью таймера, графически будет изображено, как клиент идет к нужному окну, указанному на экране (в 4 поле). Так же имеется 2 кнопки в левом нижнем углу: «✕» – для закрытия приложения прекращения работы проекта; «⟳» – для того, чтобы начать процесс обслуживания с начального окна приложения.

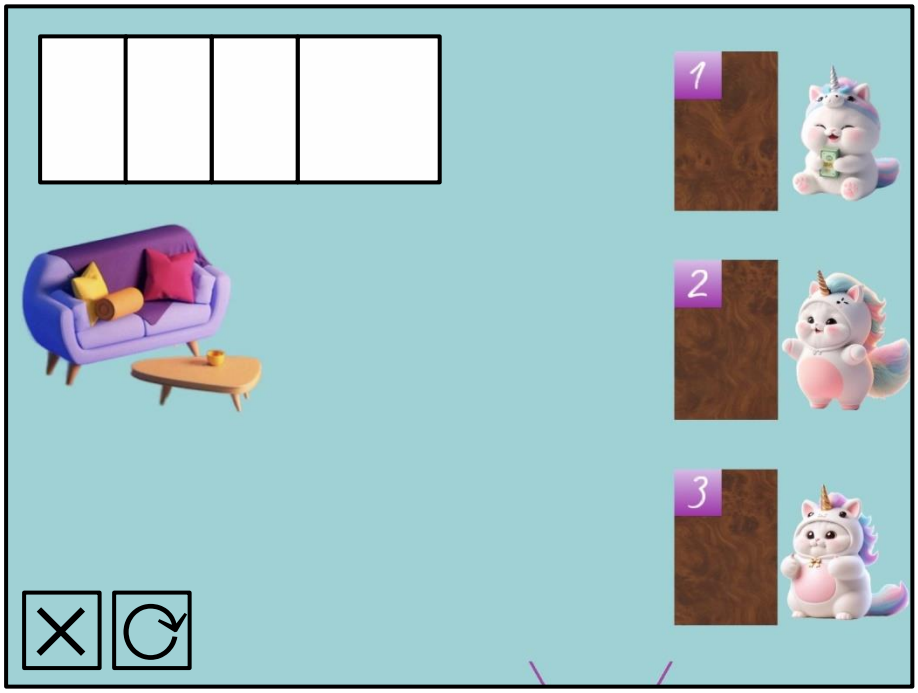


Рисунок 2.6 - Макет интерфейса «Окно приложения «Электронная очередь»».

2.3.4. Требования к надежности

Приложение должно функционировать на всех разработанных тестах. Тесты требуется разработать на этапе рабочего проекта.

2.3.5. Условия эксплуатации

Программное изделие будет эксплуатироваться на персональном компьютере пользователя под управлением операционной системы Windows.

2.3.6. Требования к составу и параметрам технических средств

Для работы клиентского приложения необходимо дисковое пространство не менее 256 Мб, свободная оперативная память в размере не менее 512 Мб, видеокарта с не менее 256 Мб видеопамяти, разрешение экрана не менее 1920*1080, операционная система не ниже Windows 7, клавиатура, мышь.

2.3.7. Требования к информационной и программной совместимости

Для разработки программного продукта необходимы использовать:

- язык C# – для разработки приложения.
- Microsoft Visual Studio не ниже версии 2022 – для написания кода приложения.

Программное изделие должно работать в операционных системах Windows с установленным .NET Framework версии не ниже 4.6.1. Для переноса программы не должны требоваться специальные аппаратные и программные средства.

2.4. Требования к программной документации

Разработка программной документации и программного изделия должна производиться согласно СТУ 02.030 – 2023 «Курсовые работы (проекты). Выпускные квалификационные работы. Общие требования к структуре и оформлению» ЮЗГУ, ГОСТ 19.001-77, ГОСТ Р 7.0.100–2018.

2.5. Стадии и этапы разработки

Выполнение разработки должно включать три стадии:

- техническое задание;
- технический проект;
- рабочий проект.

На стадии «Техническое задание» проводится постановка задачи, анализ данной предметной области, разработка требований к программному изделию, изучение литературы по задаче и оформление документа «Техническое задание».

На стадии «Технический проект» проводится проектирование программной системы. В заключение данного этапа оформляется документ "Технический проект".

На стадии «Рабочий проект» проводится реализация программной системы. В заключение данного этапа оформляется документ «Рабочий проект».

2.6. Порядок контроля и приемки

Приемка программного изделия осуществляется при сдаче документально оформленных этапов разработки и проведении испытаний на

основе установленных тестов. Тесты должны быть предоставлены поставщиком и согласованы с заказчиком.

3. Технический проект

3.1. Моделирование последовательности действий

Диаграмма последовательности для приложения представлена на рисунке 3.1.

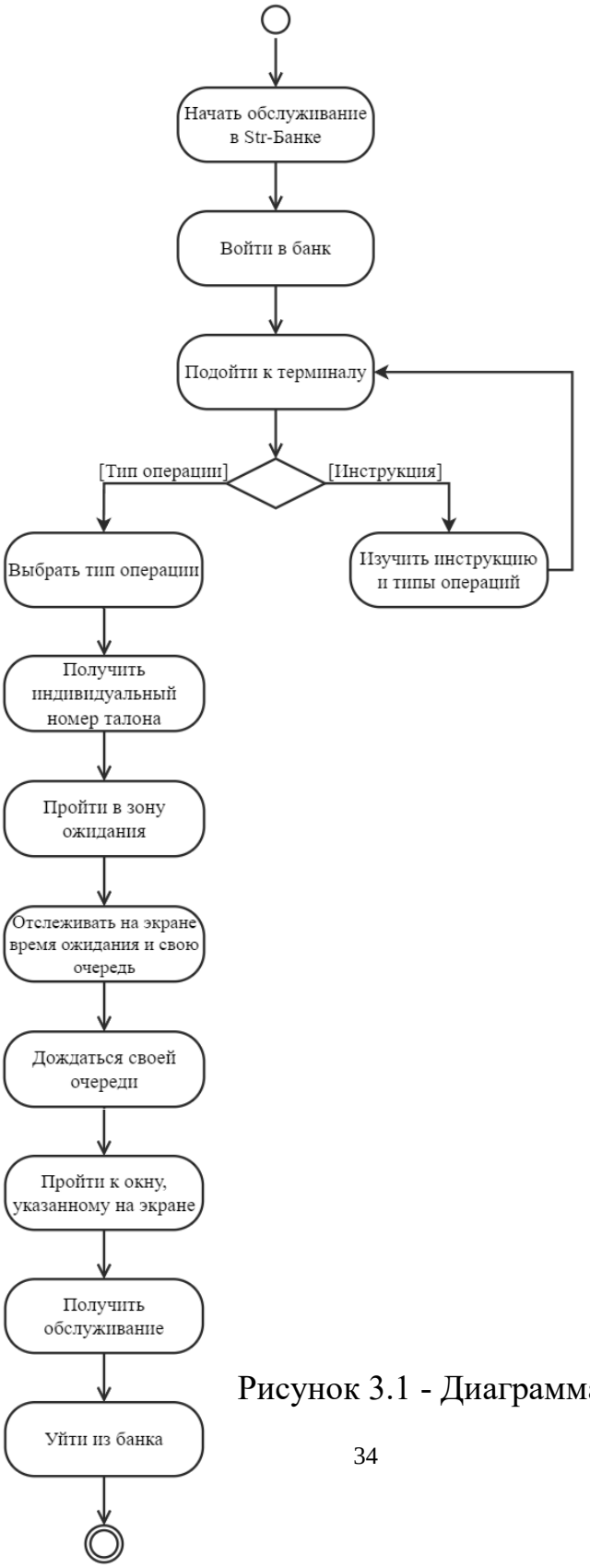


Рисунок 3.1 - Диаграмма последовательности.

3.2. Проектирование архитектуры программной системы

На рисунке 3.2 представлена диаграмма компонентов разрабатываемой программы.

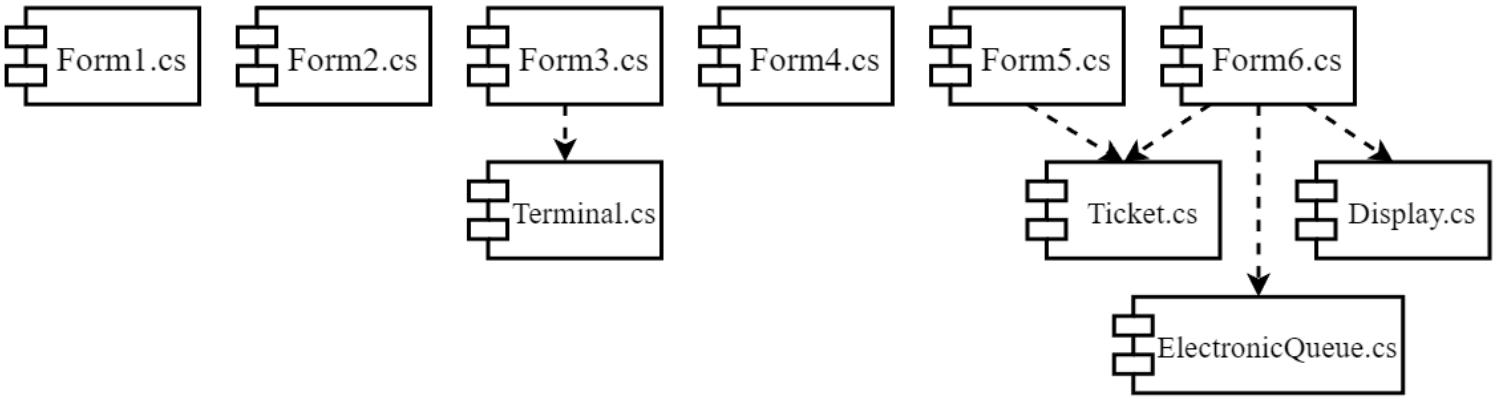


Рисунок 3.2 – Диаграмма компонентов разрабатываемой программы.

3.3. Описание структур и форматов данных

Программа основана на взаимодействии программных классов с данными, сохраненными на локальном диске пользователя – текстовые файлы, фото. В программе используются следующие форматы файлов для:

- Текстовых файлов - .txt;
- Фото - .png;

3.4. Схемы алгоритмов

В ПРИЛОЖЕНИИ Б представлены блок-схемы алгоритмов выполнения функций GetDisplayString и UpdateTalons.

4. Рабочий проект

4.1. Спецификация компонентов и классов программ

4.1.1. Модули и объекты интерфейса пользователя

Пользовательский интерфейс включает в себя следующие компоненты:

- Начальное окно приложения;
- Окно приложения «Путь в банк»;
- Главное окно приложения;
- Окно приложения «Инструкция»;
- Окно приложения «Получение талона»;
- Окно приложения «Электронная очередь»;
- Поле для отображения приветствия в банк;
- Поле для изучения инструкции и типов операции;
- Поле для индивидуального номера талона;
- Поля для просмотра примерного времени ожидания;
- Поле для визуального сигнала, с индивидуальным номером талона и окном, к которому надо пройти;
- Кнопка для изучения инструкции;
- Кнопки для выбора типа операции;
- Кнопки управления приложением.

4.1.2. Описание объектов интерфейса программы

На основе макета интерфейса, представленного в пункте 2.3.3 технического задания, с помощью языка программирования C# и IDE Visual Studio 2023 был создан пользовательский интерфейс для приложения для моделирования работы электронной очереди в банке.

На рисунках 4.1.1 – 4.1.6 представлен данный интерфейс с представленными на нём полями и кнопками. Числами на рисунках обозначены номера объектов интерфейса.

В таблицах 4.1.1 – 4.1.6 представлено описание объектов интерфейса. Взаимодействия пользователя и программы для интерфейса на рисунках 4.2.1 – 4.2.31. Прямоугольниками на рисунках обозначены объекты в окнах программы, а сплошными линиями зависимости между объектами типа «состоит из».

Программная реализация данного интерфейса представлена в виде исходного кода программы в ПРИЛОЖЕИИ А.

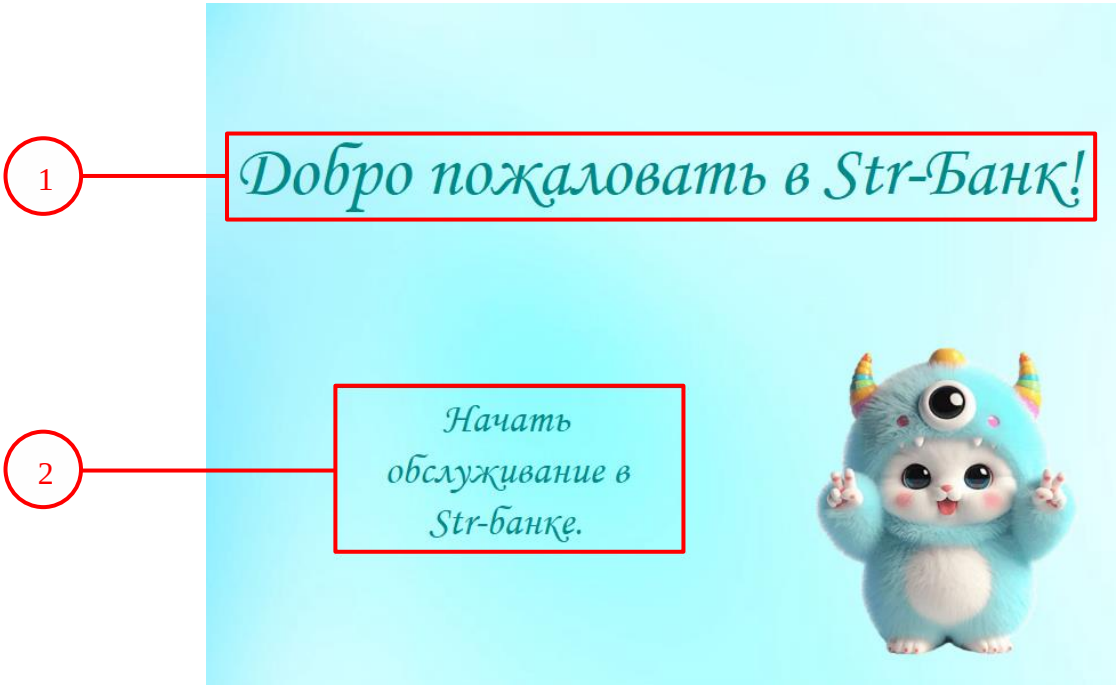


Рисунок 4.1.1 – Интерфейс взаимодействия пользователя с начальным окном приложения.

В таблице 4.1.1 представлено описание объектов интерфейса взаимодействия пользователя с начальным окном приложения.

№	Тип объекта	Имя объекта (name)	Действие/описание объекта
1	Label	label1	Поле для отображения приветствия.
2	Button	button1	Кнопка, которая начинает обслуживание в Str-Банке.

Таблица 4.1.1 – Описание объектов интерфейса взаимодействия пользователя и программы.

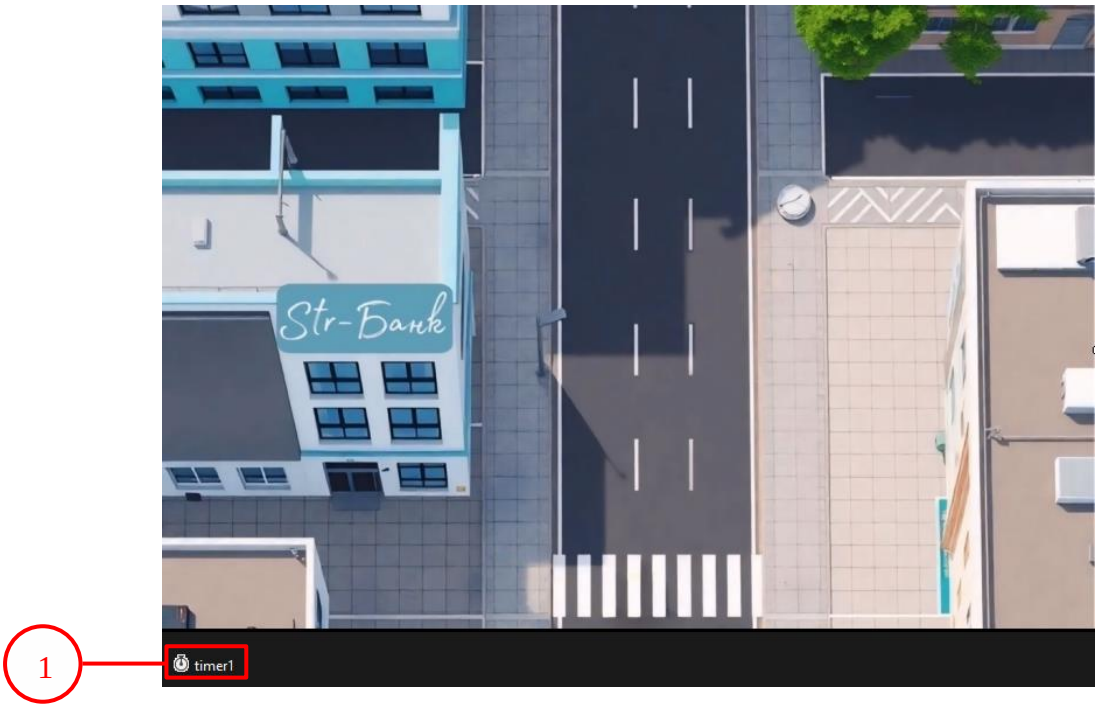


Рисунок 4.1.2 – Интерфейс взаимодействия пользователя с окном приложения «Путь в банк».

В таблице 4.1.2 представлено описание объектов интерфейса взаимодействия пользователя с окном приложения «Путь в банк».

№	Тип объ-екта	Имя объекта (name)	Действие/описание объекта
1	Timer	timer1	Таймер для управления анимацией изображения.

Таблица 4.1.2 – Описание объектов интерфейса взаимодействия пользователя и программы.

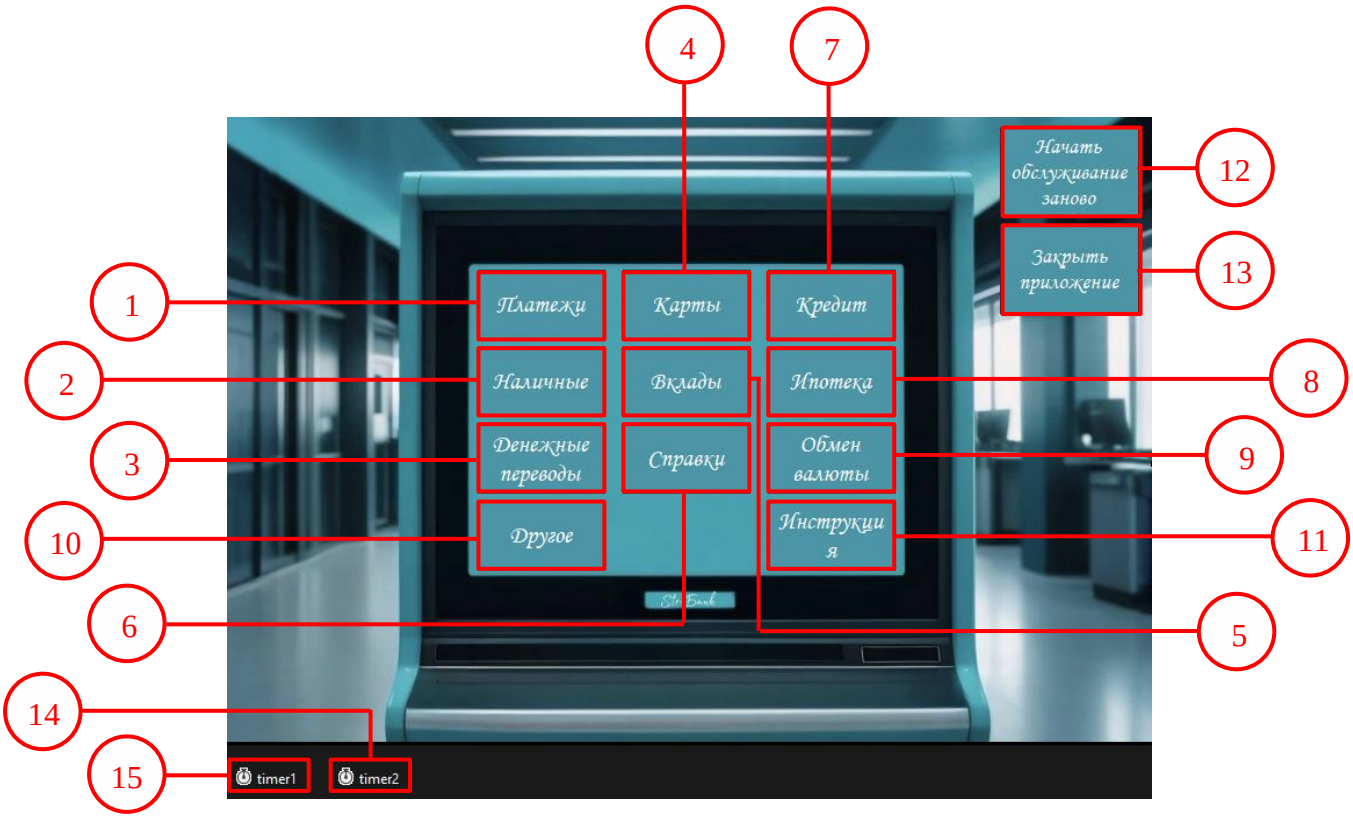


Рисунок 4.1.3 – Интерфейс взаимодействия пользователя с главным окном приложения.

В таблице 4.1.3 представлено описание объектов интерфейса взаимодействия пользователя с главным окном приложения.

№	Тип объекта	Имя объекта (name)	Действие/описание объекта
1	2	3	4
1	Button	button1	Кнопка «Платежи», которая присваивает талону категорию «А».
2	Button	button2	Кнопка «Наличные», которая присваивает талону категорию «Б».
3	Button	button3	Кнопка «Денежные переводы», которая присваивает талону категорию «В».
4	Button	button4	Кнопка «Карты», которая присваивает талону категорию «Г».

1	2	3	4
5	Button	button5	Кнопка «Вклады», которая присваивает талону категорию «Д».
6	Button	button6	Кнопка «Справки», которая присваивает талону категорию «Е».
7	Button	button7	Кнопка «Кредит», которая присваивает талону категорию «Ж».
8	Button	button8	Кнопка «Ипотека», которая присваивает талону категорию «З».
9	Button	button9	Кнопка «Обмен валюты», которая присваивает талону категорию «И».
10	Button	button10	Кнопка «Другое», которая присваивает талону категорию «К».
11	Button	button11	Кнопка, которая позволяет изучить инструкцию и типы операций.
12	Button	button12	Кнопка, которая позволяет начать обслуживание в банке заново.
13	Button	button13	Кнопка, которая завершает работу приложения.
14	Timer	timer1	Таймер, который создает анимацию появления талона, визуализируя его «выдачу».
15	Timer	timer2	Таймер, который после завершения анимации талона закрывает окно и открывает следующее.

Таблица 4.1.3 – Описание объектов интерфейса взаимодействия пользователя и программы.

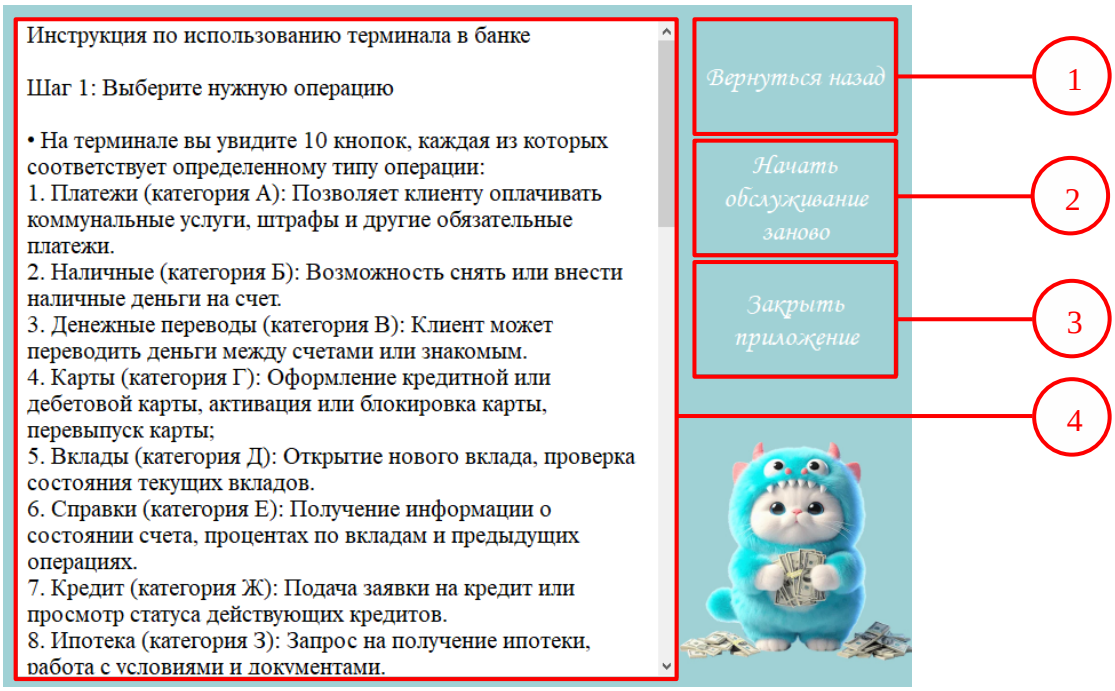


Рисунок 4.1.4 – Интерфейс взаимодействия пользователя с окном приложения «Инструкция».

В таблице 4.1.4 представлено описание объектов интерфейса взаимодействия пользователя с окном приложения «Инструкция».

№	Тип объекта	Имя объекта (name)	Действие/описание объекта
1	Button	button1	Кнопка, которая позволяет вернуться к главному окну.
2	Button	button2	Кнопка, которая позволяет начать обслуживание в банке заново.
3	Button	button3	Кнопка, которая завершает работу приложения.
4	TextBox	textBox1	Поле для изучения инструкции.

Таблица 4.1.4 – Описание объектов интерфейса взаимодействия пользователя и программы.

Рисунок 4.1.5 – Интерфейс взаимодействия пользователя с окном приложения «Получение талона».

В таблице 4.1.5 представлено описание объектов интерфейса взаимодействия пользователя с окном приложения «Получение талона».

№	Тип объекта	Имя объекта (name)	Действие/описание объекта
1	Button	button1	Кнопка, которая позволяет начать обслуживание в банке заново.
2	Button	button2	Кнопка, которая завершает работу приложения.
3	Label	label1	Поле с индивидуальным номером талона.
4	Timer	timer1	Таймер для перехода на следующую форму спустя время.

Таблица 4.1.5 – Описание объектов интерфейса взаимодействия пользователя и программы.

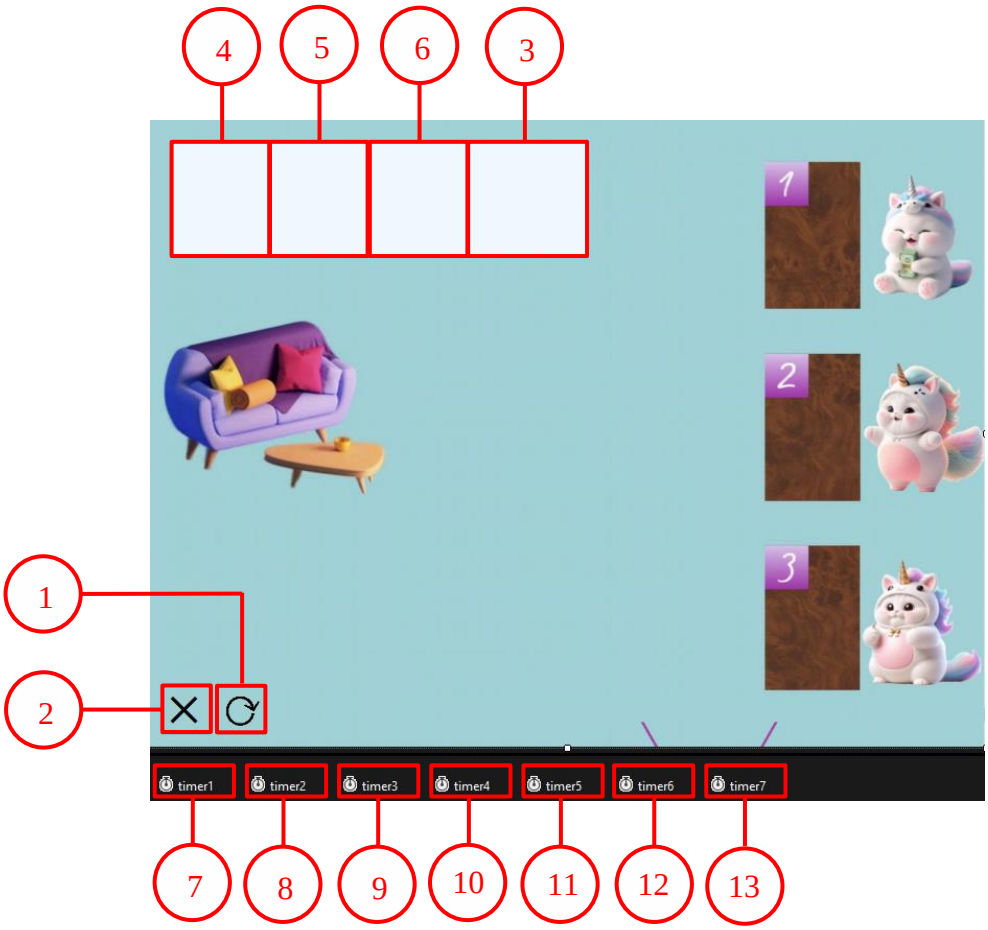


Рисунок 4.1.6 – Интерфейс взаимодействия пользователя с окном приложения «Электронная очередь».

В таблице 4.1.6 представлено описание объектов интерфейса взаимодействия пользователя с окном приложения «Электронная очередь».

№	Тип объ-екта	Имя объекта (name)	Действие/описание объекта
1	2	3	4
1	Button	button1	Кнопка, которая позволяет начать обслуживание в банке заново.
2	Button	button2	Кнопка, которая завершает работу приложения.

1	2	3	4
3	RichTextBox	richTextBox1	Поле для визуального сигнала и просмотра индивидуального номера талона с указанием окна, в которое надо пройти.
4	RichTextBox	richTextBox2	Поле для просмотра примерного времени ожидания для категорий «А», «Б», «В».
5	RichTextBox	richTextBox3	Поле для просмотра примерного времени ожидания для категорий «Г», «Д», «Е».
6	RichTextBox	richTextBox4	Поле для просмотра примерного времени ожидания для категорий «Ж», «З», «И», «К».
7	Timer	timer1	Таймер, который обновляет информацию об электронной очереди для категорий «А», «Б», «В».
8	Timer	timer2	Таймер, который обновляет информацию об электронной очереди для категорий «Г», «Д», «Е».
9	Timer	timer3	Таймер, который обновляет информацию об электронной очереди для категорий «Ж», «З», «И», «К».
10	Timer	timer4	Таймер, который очищает поле, очищает список currentTalons, останавливает подсветку.
11	Timer	timer5	Таймер для управления анимацией изображения.
12	Timer	timer6	Таймер для управления остановки анимации изображения.
13	Timer	timer7	Таймер для управления анимацией изображения.

Таблица 4.1.6 – Описание объектов интерфейса взаимодействия пользователя и программы.

4.1.3. Описание классов

Описание класса Terminal

– Внутренние поля класса:

string filePath: Путь к файлу, где будут храниться данные о счетчиках.

int[] counters: Массив целых чисел, представляющий собой сами счетчики.

– Внутренние методы класса:

void Load(): Загружает данные о счетчиках из файла. Проверяет, существует ли файл по указанному пути filePath. Если файл существует, считывает все строки из него. Проходит по каждой строке, пытается преобразовать ее в целое число. Если преобразование успешное, обновляет соответствующий элемент массива counters полученным значением.

void Save(): Сохраняет текущие значения счетчиков в файл. Преобразует массив counters в массив строк с помощью Array.ConvertAll. Записывает полученные строки в файл по указанному пути filePath.

Описание класса Ticket

– Внутренние поля класса:

string _filePath: Хранит путь к файлу, где будут сохраняться данные о талонах.

– Внутренние методы класса:

void ClearFile(): Очищает содержимое файла. Проверяет, существует ли файл по указанному пути _filePath. Если файл существует, записывает в него пустую строку string.Empty с помощью File.WriteAllText, тем самым очищая его содержимое.

void Save(string category, int ticketNumber): Сохраняет информацию о талоне в файл. Использует StreamWriter для записи в файл. Записывает в файл строку в формате "{category}{ticketNumber}". Использует using для автоматического закрытия StreamWriter после завершения работы с файлом.

Описание класса ElectronicQueue

– Внутренние поля класса:

string QueueName { get; private set; }: Название очереди.

List<string> Talons { get; private set; }: Список талонов, находящихся в очереди.

List<int> WaitTimes { get; private set; }: Список времени ожидания для каждого талона.

int _initialWaitTime: Начальное время ожидания для первого талона в очереди.

_waitTimeIncrement: Приращение времени ожидания для каждого последующего талона.

– Внутренние методы класса:

void AddTalon(string talon): Добавляет новый талон в очередь. Добавляет талон в список Talons. Добавляет в список WaitTimes время ожидания, которое рассчитывается как _initialWaitTime плюс _waitTimeIncrement, умноженное на количество талонов в очереди минус 1. Увеличивает _initialWaitTime на _waitTimeIncrement, чтобы следующее время ожидания для нового талона было больше.

string GetDisplayString(): Возвращает строку, представляющую собой текстовое отображение очереди. Использует StringBuilder для построения строки. Для каждого талона в списке Talons формирует строку вида "{talon} - {форматированное время ожидания}" и добавляет ее в StringBuilder. Возвращает сформированную строку.

string FormatTime(int seconds): Преобразует количество секунд в строку с часами и минутами в формате "MM:SS".

Описание класса Display

– Внутренние поля класса:

string talonFilePath: Путь к файлу, где будут сохраняться талоны.

RichTextBox _richTextBox1, RichTextBox _richTextBox2, RichTextBox _richTextBox3, RichTextBox _richTextBox4: Ссылки на RichTextBox-компоненты, которые будут использоваться для отображения информации.

List<string> _currentTalons: Список талонов, которые уже были обработаны.

Timer _timer4: Таймер, используемый для подсветки _richTextBox1.

ElectronicQueue _queue1, ElectronicQueue _queue2, ElectronicQueue _queue3: Ссылки на экземпляры класса ElectronicQueue, представляющие три электронные очереди для разных категорий.

bool isHighlighting: Флаг, указывающий, выделен ли _richTextBox1.

– Внутренние методы класса:

void UpdateTalons(ElectronicQueue queue, RichTextBox richTextBox): Обновляет информацию о талонах в заданной очереди и отображает ее в соответствующем RichTextBox. Проходит по списку WaitTimes очереди в обратном порядке. Уменьшает время ожидания каждого талона на 1 секунду. Если время ожидания талона истекло: определяет номер очереди, в которую передан талон, используя GetRichTextBoxNumber; добавляет запись в _richTextBox1 о переданном талоне; добавляет талон в список _currentTalons; запускает таймер _timer4, который будет подсвечивать _richTextBox1; удаляет талон из очереди; обновляет файл с талонами; обновляет отображение всех очередей; обновляет отображение талонов в указанном richTextBox.

int GetRichTextBoxNumber(RichTextBox richTextBox): Возвращает номер очереди, соответствующей заданному richTextBox.

void UpdateTalonDisplay(): Обновляет отображение всех трех очередей в соответствующих RichTextBox-компонентах.

void UpdateTalonFile(): Сохраняет все талоны из всех очередей в файл.

void StartHighlight(): Выделяет _richTextBox1, меняя его цвет фона.

void StopHighlight(): Снимает выделение с _richTextBox1, возвращая его цвет фона к исходному.

void ClearHighlight(): Очищает _richTextBox1 от записей о переданных талонах, удаляет эти талоны из списка _currentTalons, останавливает подсветку и останавливает таймер _timer4.

На рисунке 4.1.7 показана диаграмма программных классов.

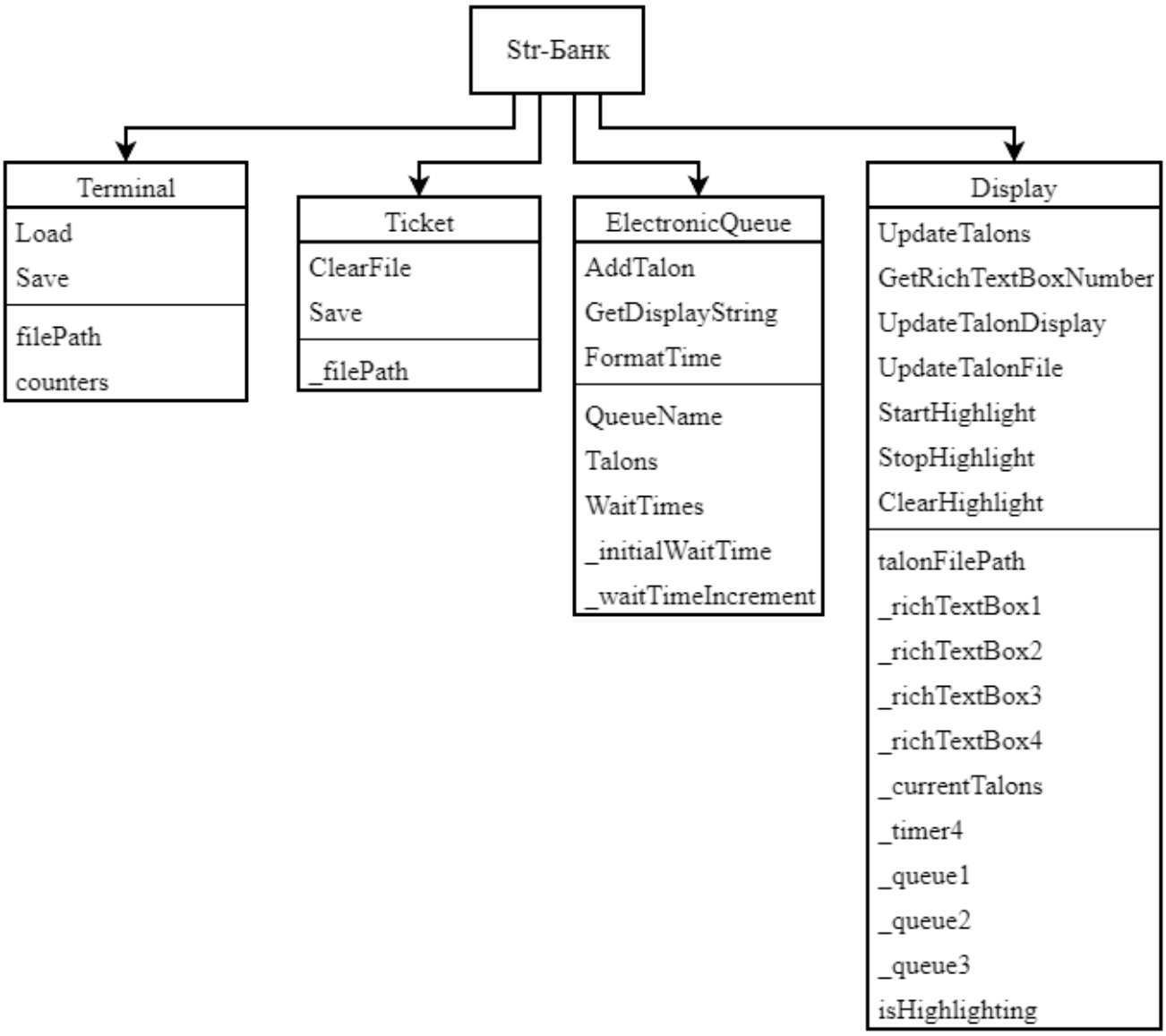


Рисунок 4.1.7 – Диаграмма программных классов.

4.2. Тестирование программной системы

Разработка тестовых наборов для интерфейсов будет рассмотрена на примере тестирования приложения для моделирования электронной очереди в банке.

В таблице 4.2 представлены тестовые наборы для отладки интерфейса.

Имя теста	Проверяемая ситуация	Действия пользователя	Входные данные	Реакция системы
1	2	3	4	5
T1	Пользователь открывает приложение.	Пользователь запустил приложение.	Открыто начальное окно приложения.	Отображается начальное окно приложения, с приветствием и кнопкой.
T2	Пользователь начинает обслуживание.	Пользователь нажимает на кнопку «Начать обслуживание в Str-Банке».	На начальном окне приложения нажата кнопка.	Отображается окно приложения «Путь в банк», и изображение с помощью графики движется в здание банка.
T3	Пользователь начинает работу с терминалом.		Открыто главное окно приложения.	Отображается главное окно приложения с 13 кнопками.
T4	Пользователь закрывает приложение.	Пользователь нажимает на кнопку закрытия приложения.	На любом этапе приложения нажата кнопка закрытия.	Приложение закрывается, и работа проекта прекращается.

1	2	3	4	5
Т5	Пользователь начинает обслуживание заново.	Пользователь нажимает на кнопку «Начать обслуживание заново».	На любом этапе приложения нажата кнопка.	Приложение открывает начальное окно.
Т6	Пользователь открывает инструкцию.	Пользователь нажимает на кнопку «Инструкция».	На главном окне приложения нажата кнопка «Инструкция»	Приложение открывает окно «Инструкция».
Т7	Пользователь возвращается на главное окно программы.	Пользователь нажимает на кнопку «Вернуться назад».	На окне приложения «Инструкция» нажата кнопка «Вернуться назад».	Приложение открывает главное окно.
Т8	Пользователь выбирает тип операции.	Пользователь нажимает на кнопку с названием типа операции.	На главном окне приложения выбран тип операции.	Отображается главное окно приложения и с помощью графики из терминала появляется талон.
Т9	Демонстрация индивидуального номера талона.		Открыто главное окно приложения и выбран тип операции.	Приложение открывает окно «Получение талона».
Т10	Демонстрация электронной очереди.	Пользователь получил индивидуальный номер талона.	Открыто окно приложения «Электронная очередь».	Приложение открывает окно «Электронная очередь».

1	2	3	4	5
Т11	Демонстрация визуального сигнала (сиреневого цвета) для категорий для категорий «А», «Б», «В».	Пользователь получил индивидуальный номер талона.	Открыто окно приложения «Электронная очередь».	В окне отображается индивидуальный номер талона и номер окна (1), к которому надо пройти.
Т12	Демонстрация визуального сигнала (сиреневого цвета) для категорий для категорий «Г», «Д», «Е».			В окне отображается индивидуальный номер талона и номер окна (2), к которому надо пройти.
Т13	Демонстрация визуального сигнала (сиреневого цвета) для категорий для категорий «Ж», «З», «И», «К».			В окне отображается индивидуальный номер талона и номер окна (3), к которому надо пройти.
Т14	Демонстрация движения клиента к окну 1.			Отображается, как с помощью графики изображение движется к окну 1.
Т15	Демонстрация движения клиента к окну 2.			Отображается, как с помощью графики изображение движется к окну 2.
Т16	Демонстрация движения клиента к окну 3.			Отображается, как с помощью графики изображение двигается к окну 3.

1	2	3	4	5
Т17	Начало получения обслуживания.	Пользователь получил индивидуальный номер талона.	Открыто окно приложения «Электронная очередь».	Отображается сообщение о начале обслуживания.
Т18	Завершение обслуживания.			Отображается сообщение о завершении обслуживания.
Т19	Демонстрация движения клиента к выходу из банка.			Отображается, как с помощью графики изображение движется к выходу из банка.
Т19	Выход клиента из банка.			Отображается сообщение о завершении обслуживания.

Таблица 4.2 – Тестовые наборы для отладки интерфейса.

Тестовые наборы для отладки работы приложения.

Для отладки работы программы разработаны следующие тестовые наборы:

1. **Случай использования: Пользователь открывает приложение.**

Предусловие: Приложение запущено, открыто начальное окно приложения;

Тестовый случай: Пользователь запустил приложение;

Ожидаемый результат: Отображается начальное окно приложения, с приветствием и кнопкой;

Результат представлен на рисунке 4.2.1.

Рисунок 4.2.1 – Запуск приложения.

2. **Случай использования: Пользователь начинает обслуживание.**

Предусловие: Приложение запущено, открыто начальное окно приложения;

Тестовый случай: Пользователь нажимает на кнопку «Начать обслуживание в Str-Банке»;

Ожидаемый результат: Отображается окно приложения «Путь в банк», и изображение с помощью графики движется в здание банка;

Результат представлен на рисунках 4.2.2 – 4.2.5.

Рисунок 4.2.2 – Путь в Банк.

Рисунок 4.2.3 – Путь в Банк.

Рисунок 4.2.4 – Путь в Банк.

Рисунок 4.2.5 – Путь в Банк.

3. **Случай использования: Пользователь начинает работу с терминалом.**

Предусловие: Приложение запущено, показано окно приложения «Путь в банк»;

Тестовый случай: Пользователь нажимает на кнопку «Начать обслуживание в Str-Банке»;

Ожидаемый результат: Отображается главное окно приложения с 13 кнопками;

Результат представлен на рисунке 4.2.6.

Рисунок 4.2.6 – Терминал для обслуживания в банке.

4. **Случай использования: Пользователь закрывает приложение.**

Предусловие: Приложение запущено, открыто любое окно приложения;

Тестовый случай: Пользователь нажимает на кнопку закрытия приложения;

Ожидаемый результат: Приложение закрывается, и работа проекта прекращается.

5. **Случай использования: Пользователь начинает обслуживание заново.**

   Предусловие: Приложение запущено, открыто любое окно приложения;

   Тестовый случай: Пользователь нажимает на кнопку «Начать обслуживание заново»;

   Ожидаемый результат: Приложение открывает начальное окно;

   Результат представлен на рисунке 4.2.1.

6. **Случай использования: Пользователь открывает инструкцию.**

Предусловие: Приложение запущено, открыто главное окно приложения;

Тестовый случай: Пользователь нажимает на кнопку «Инструкция»;

Ожидаемый результат: Приложение открывает окно «Инструкция»;

Результат представлен на рисунке 4.2.7.

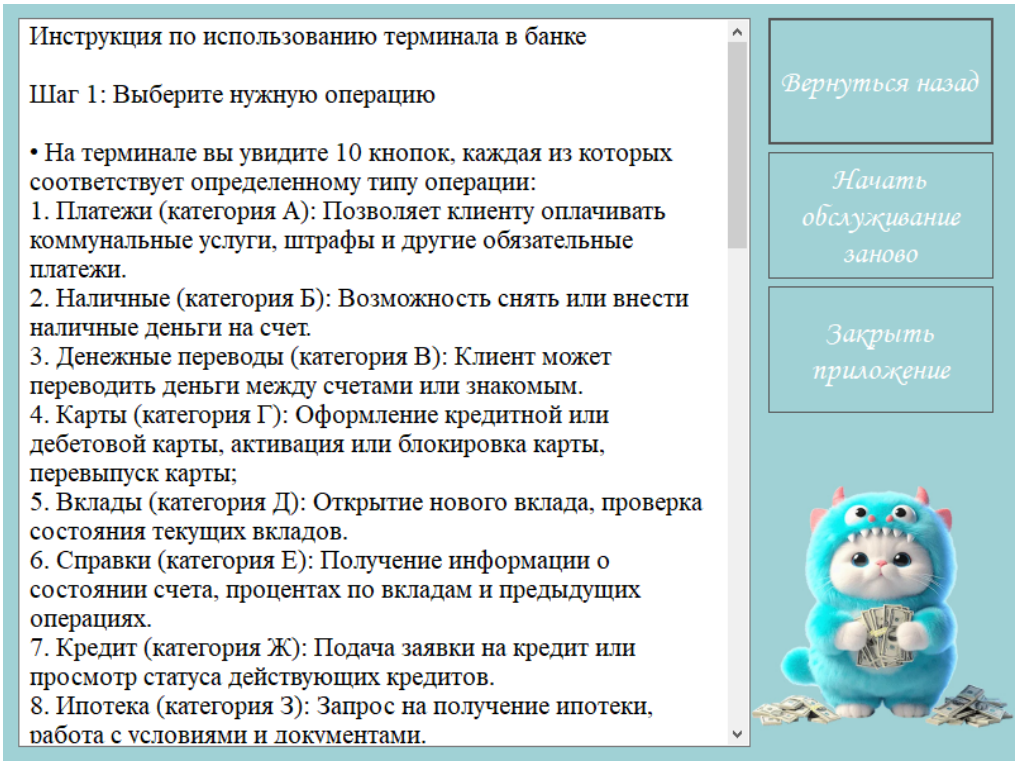


Рисунок 4.2.7 – Инструкция.

7. **Случай использования: Пользователь возвращается на главное окно программы.**

Предусловие: Приложение запущено, открыто окно приложения «Инструкция»;

Тестовый случай: Пользователь нажимает на кнопку «Вернуться назад»;

Ожидаемый результат: Приложение открывает главное окно;

Результат представлен на рисунке 4.2.6.

8. **Случай использования: Пользователь выбирает тип операции.**

Предусловие: Приложение запущено, открыто главное окно приложения;

Тестовый случай: Пользователь нажимает на кнопку с названием типа операции;

Ожидаемый результат: Отображается главное окно приложения и с помощью графики из терминала появляется талон;

Результат представлен на рисунке 4.2.8.

Рисунок 4.2.8 – Появление талона из терминала.

9. **Случай использования: Демонстрация индивидуального номера талона.**

Предусловие: Приложение запущено, открыто главное окно приложения;

Тестовый случай: Пользователь нажимает на кнопку с названием типа операции;

Ожидаемый результат: Приложение открывает окно «Получение талона»;

Результат представлен на рисунке 4.2.9.

Рисунок 4.2.9 – Получение талона с индивидуальным номером.

10. Случай использования: Демонстрация электронной очереди.

Предусловие: Приложение запущено, открыто окно приложения «Получение талона»;

Тестовый случай: Пользователь получил индивидуальный номер талона;

Ожидаемый результат: Приложение открывает окно «Электронная очередь»;

Результат представлен на рисунке 4.2.10.

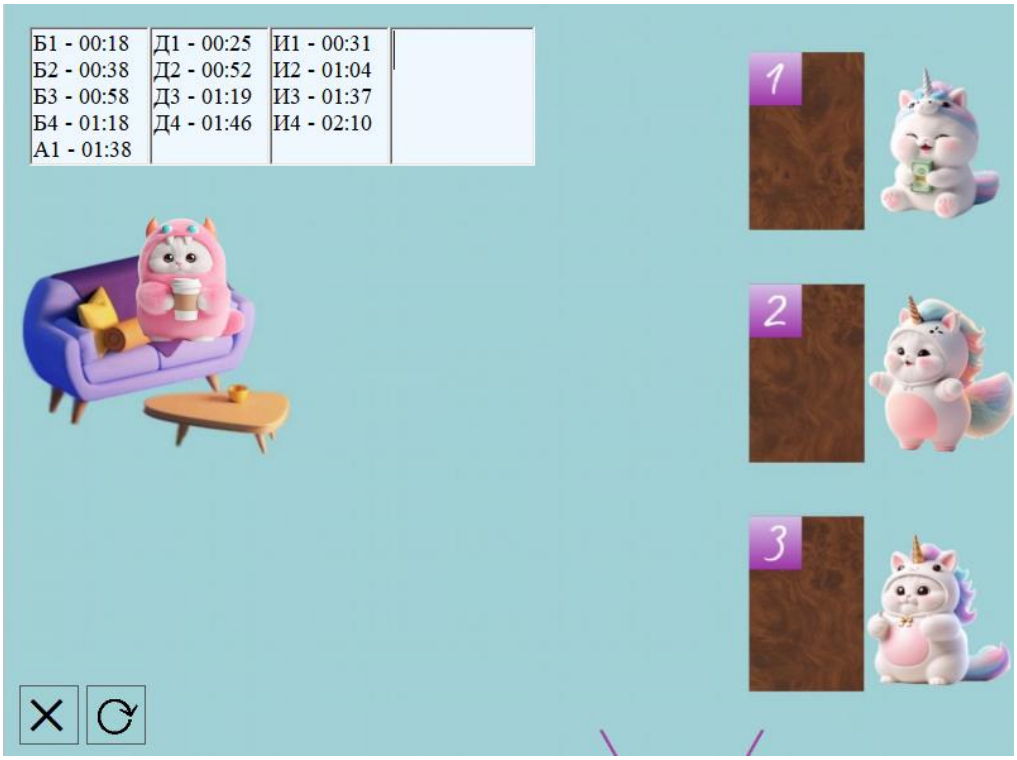


Рисунок 4.2.10 – Электронная очередь.

11. **Случай использования: Демонстрация визуального сигнала для категорий для категорий «А», «Б», «В».**

Предусловие: Приложение запущено, открыто окно приложения «Электронная очередь»;

Тестовый случай: Пользователь получил индивидуальный номер талона;

Ожидаемый результат: В окне отображается индивидуальный номер талона и номер окна (1), к которому надо пройти;

Результат представлен на рисунке 4.2.11.

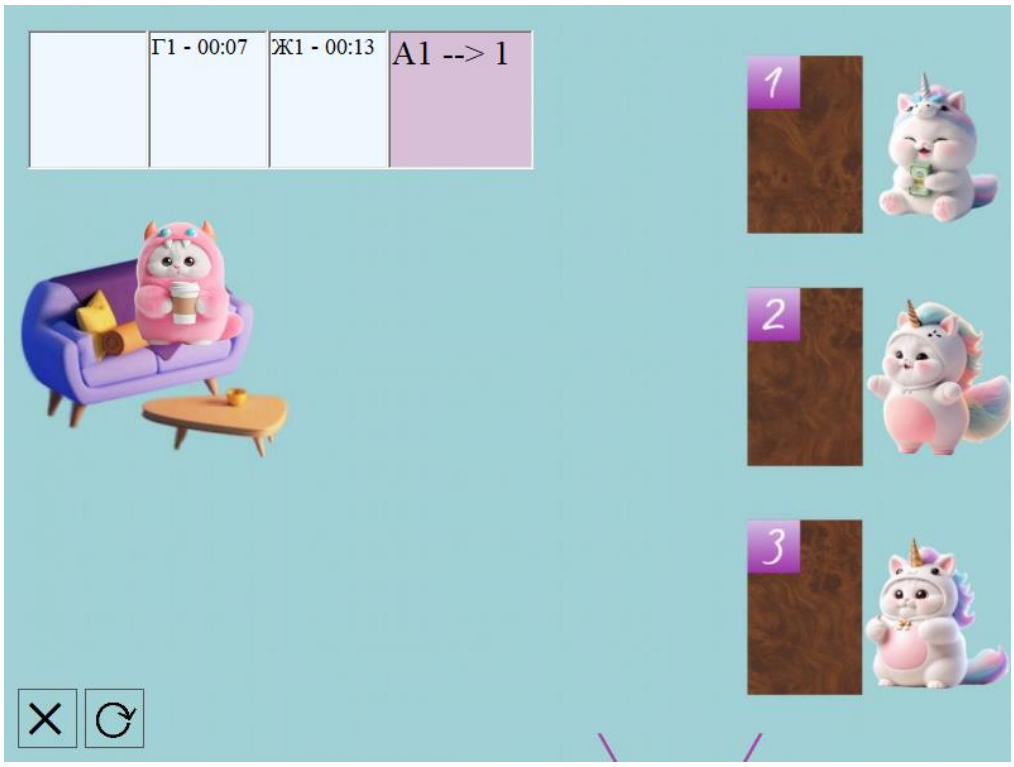


Рисунок 4.2.11 – Визуальный сигнал для категорий «А», «Б», «В».

12. **Случай использования: Демонстрация визуального сигнала для категорий для категорий «Г», «Д», «Е».**

Предусловие: Приложение запущено, открыто окно приложения «Электронная очередь»;

Тестовый случай: Пользователь получил индивидуальный номер талона;

Ожидаемый результат: В окне отображается индивидуальный номер талона и номер окна (2), к которому надо пройти;

Результат представлен на рисунке 4.2.12.

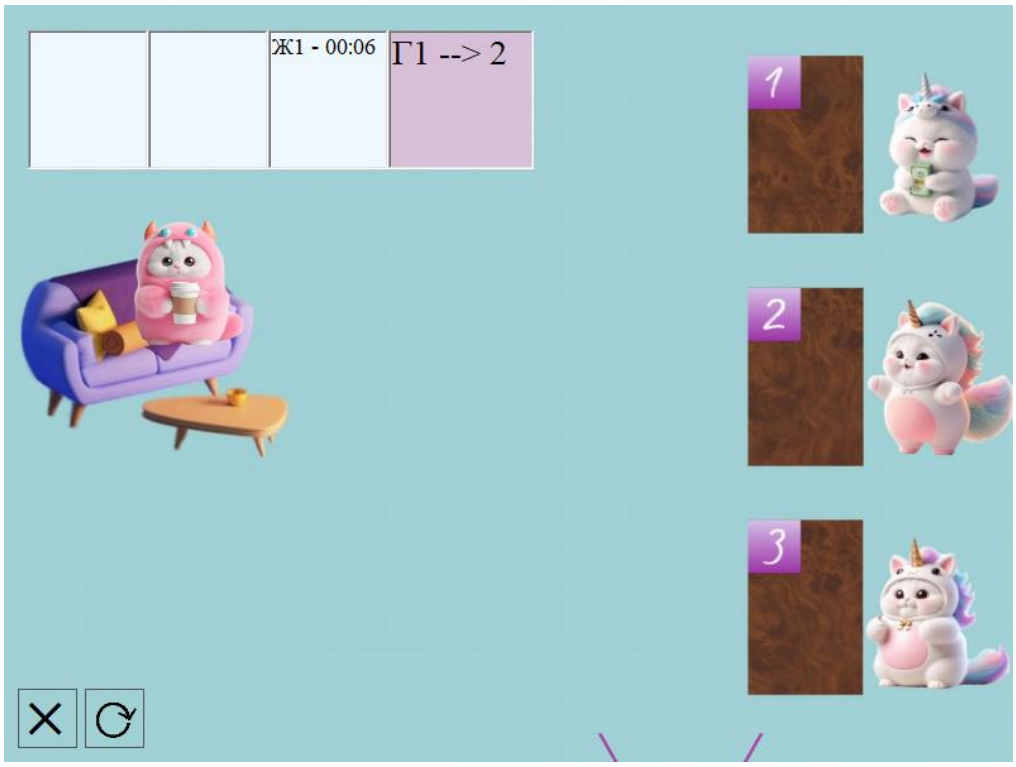


Рисунок 4.2.12 – Визуальный сигнал для категорий «Г», «Д», «Е».

13. **Случай использования: Демонстрация визуального сигнала для категорий для категорий «Ж», «З», «И», «К».**

Предусловие: Приложение запущено, открыто окно приложения «Электронная очередь»;

Тестовый случай: Пользователь получил индивидуальный номер талона;

Ожидаемый результат: В окне отображается индивидуальный номер талона и номер окна (3), к которому надо пройти;

Результат представлен на рисунке 4.2.13.

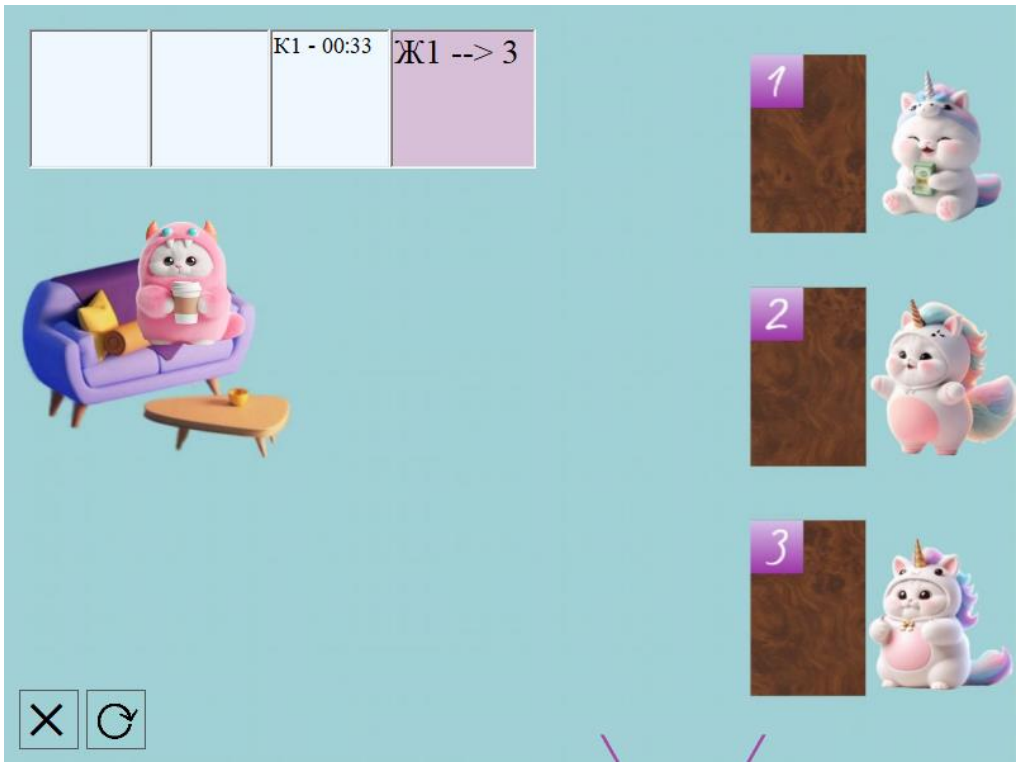


Рисунок 4.2.13 – Визуальный сигнал для категорий «Ж», «З», «И», «К».

14. Случай использования: Демонстрация движения клиента к окну 1.

Предусловие: Приложение запущено, открыто окно приложения «Электронная очередь»;

Тестовый случай: Пользователь получил индивидуальный номер талона;

Ожидаемый результат: Отображается, как с помощью графики изображение движется к окну 1;

Результат представлен на рисунках 4.2.14 – 4.2.17.

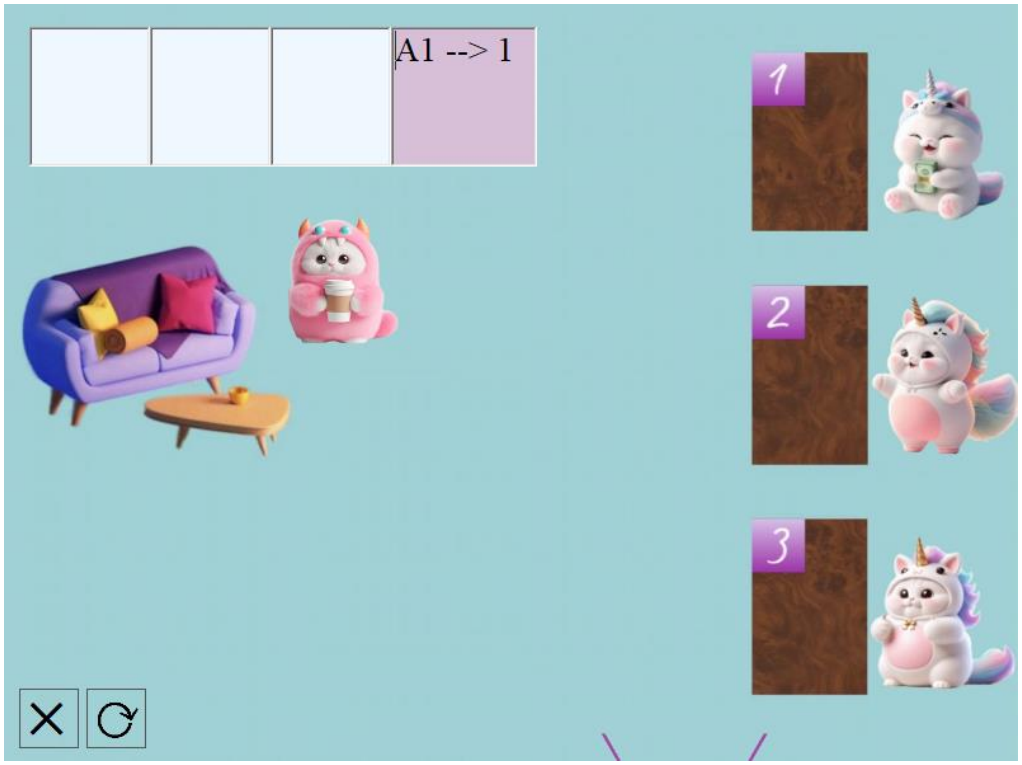


Рисунок 4.2.14 – Путь клиента к окну 1.

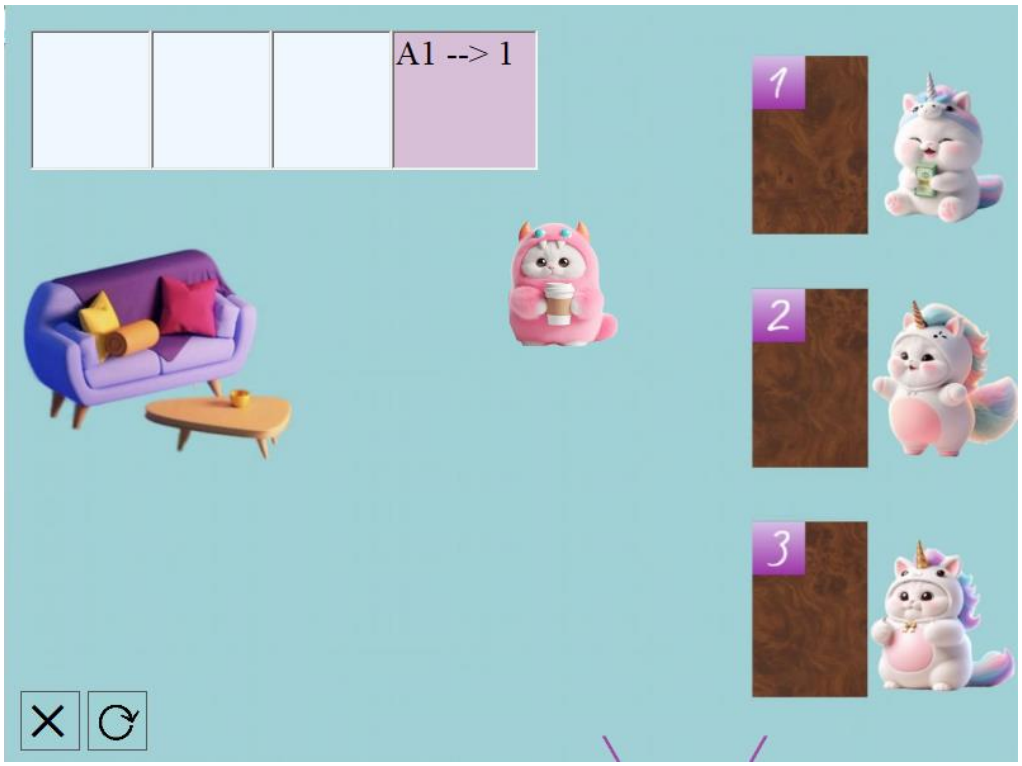


Рисунок 4.2.15 – Путь клиента к окну 1.

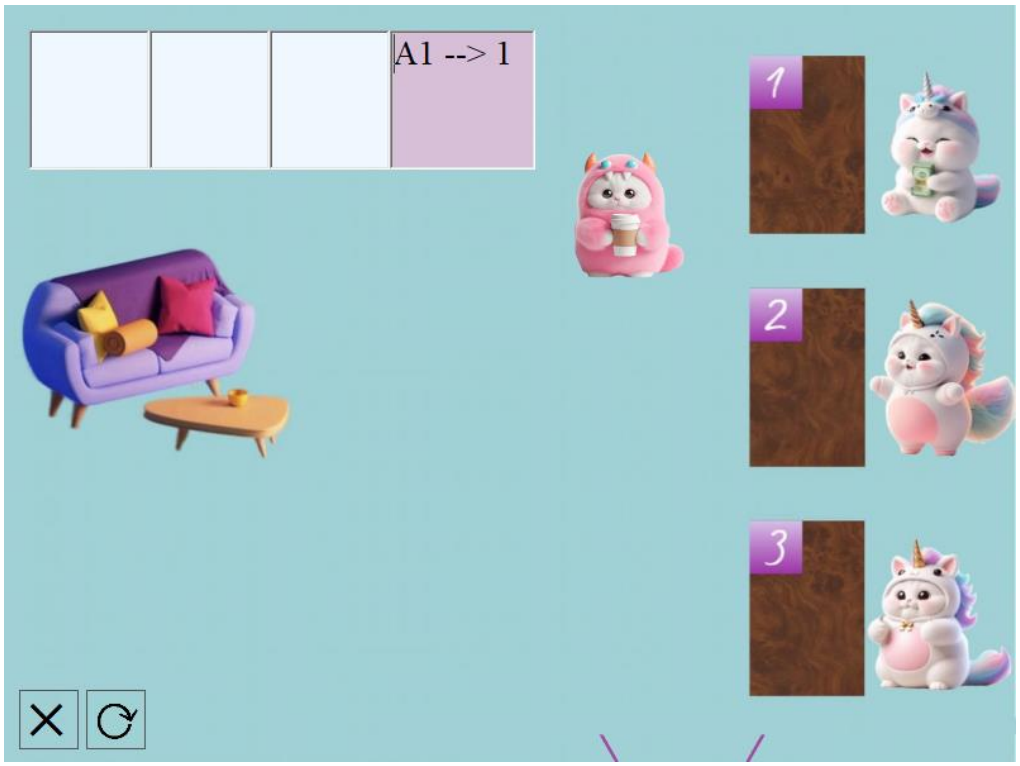


Рисунок 4.2.16 – Путь клиента к окну 1.

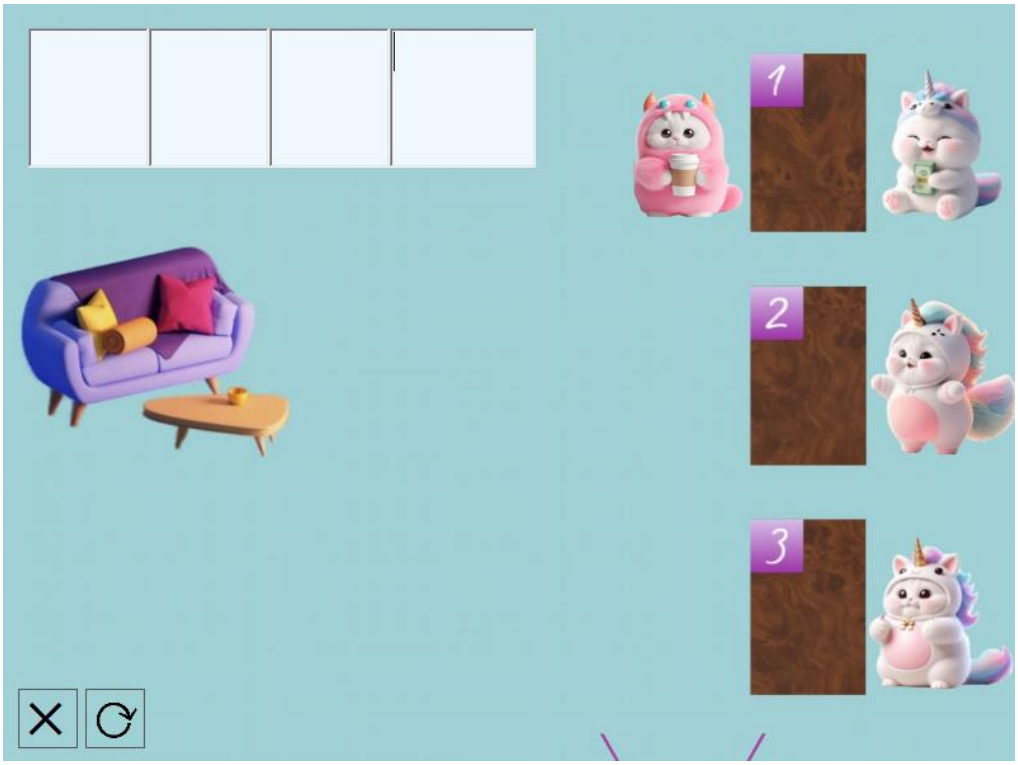


Рисунок 4.2.17 – Путь клиента к окну 1.

15. Случай использования: Демонстрация движения клиента к окну 2.

Предусловие: Приложение запущено, открыто окно приложения «Электронная очередь»;

Тестовый случай: Пользователь получил индивидуальный номер талона;

Ожидаемый результат: Отображается, как с помощью графики изображение движется к окну 2;

Результат представлен на рисунках 4.2.18 – 4.2.20.

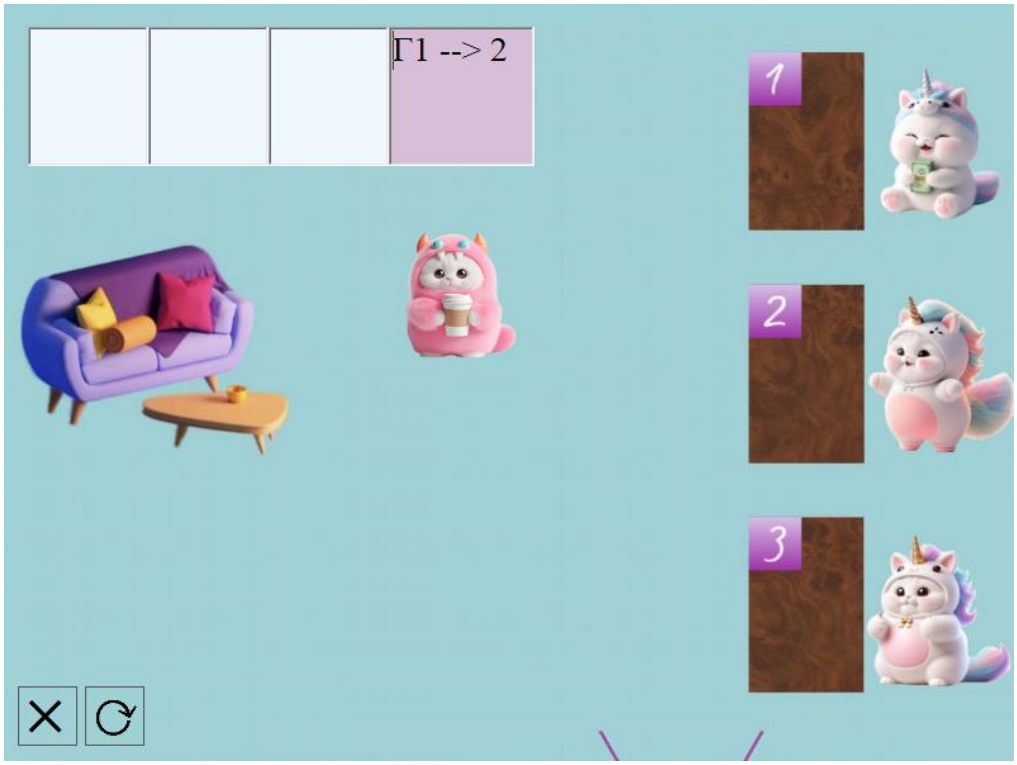


Рисунок 4.2.18 – Путь клиента к окну 2.

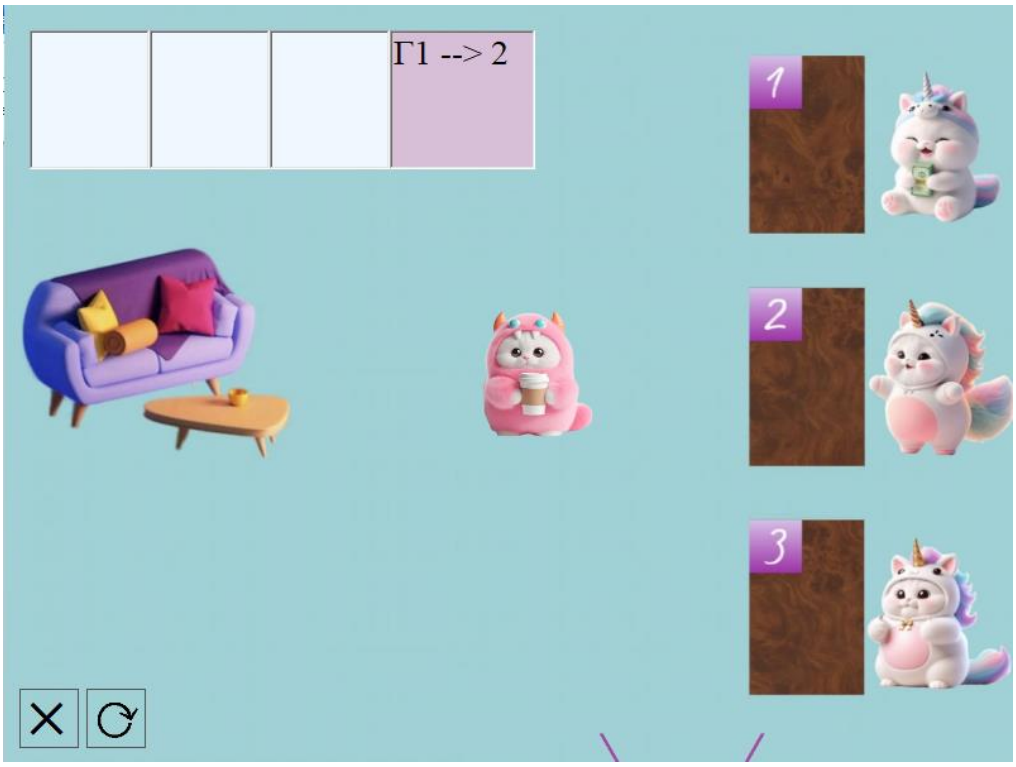


Рисунок 4.2.19 – Путь клиента к окну 2.

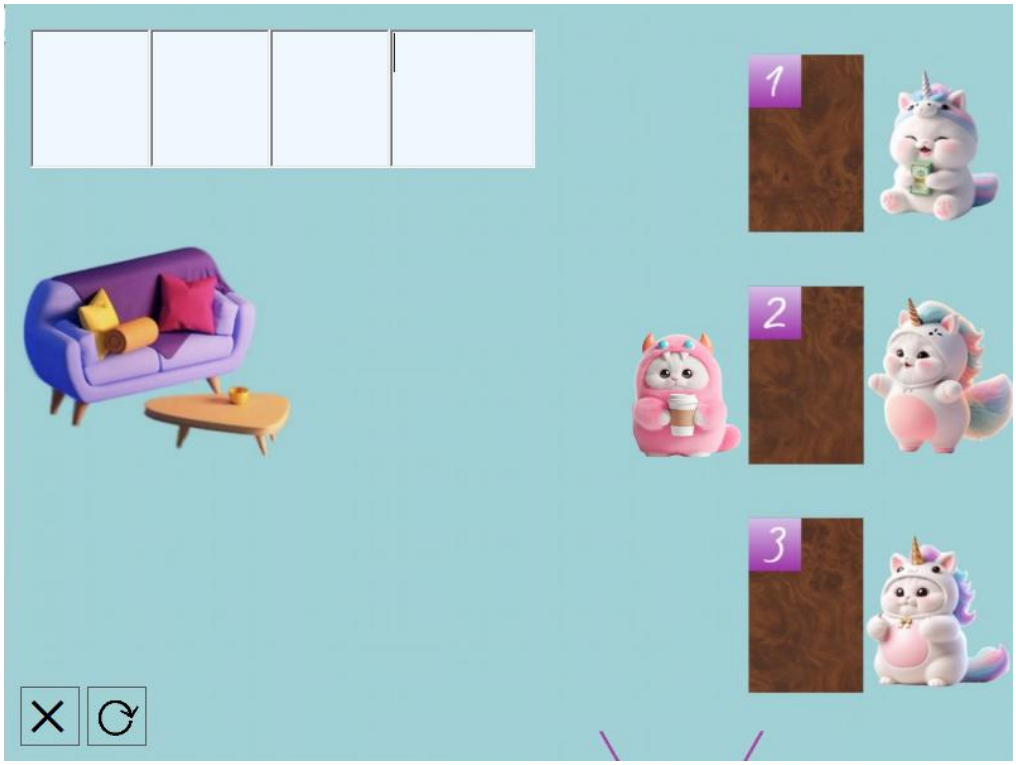


Рисунок 4.2.20 – Путь клиента к окну 2.

16. Случай использования: Демонстрация движения клиента к окну 3.

Предусловие: Приложение запущено, открыто окно приложения «Электронная очередь»;

Тестовый случай: Пользователь получил индивидуальный номер талона;

Ожидаемый результат: Отображается, как с помощью графики изображение движется к окну 3;

Результат представлен на рисунках 4.2.21 – 4.2.24.

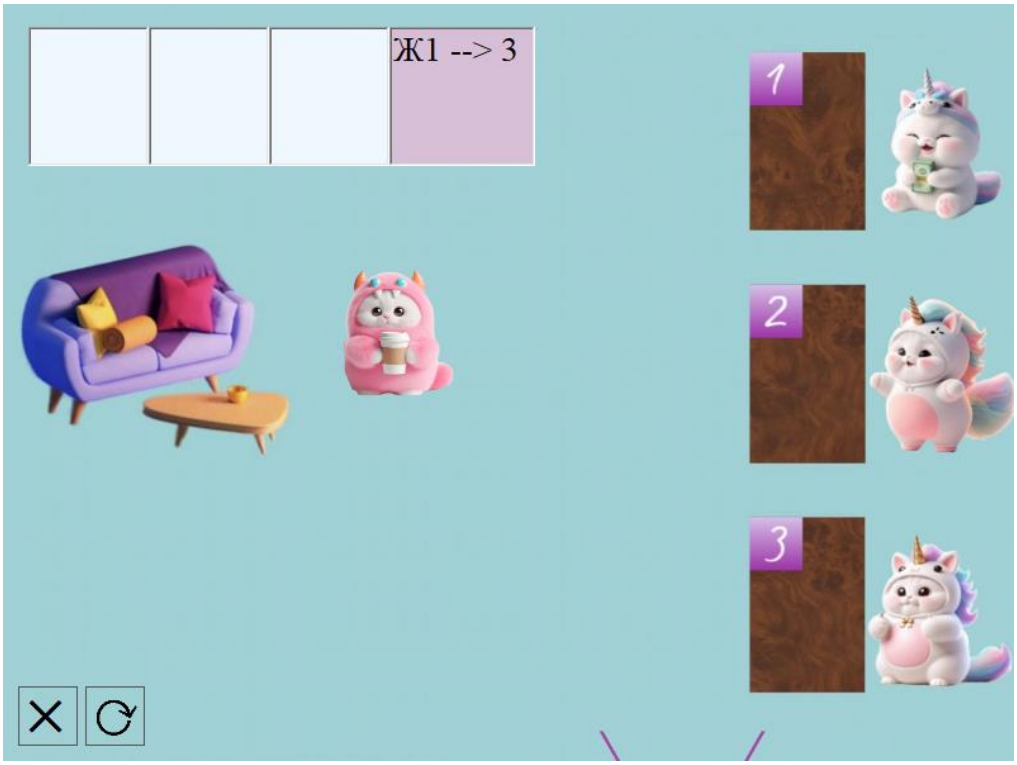


Рисунок 4.2.21 – Путь клиента к окну 3.

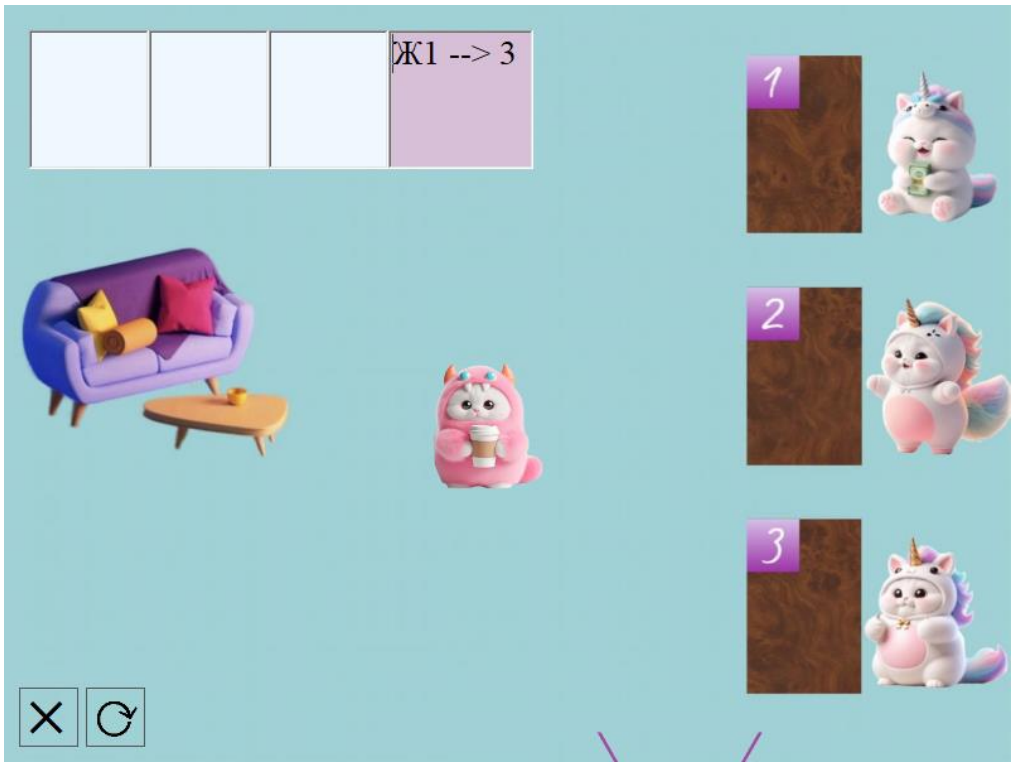


Рисунок 4.2.22 – Путь клиента к окну 3.

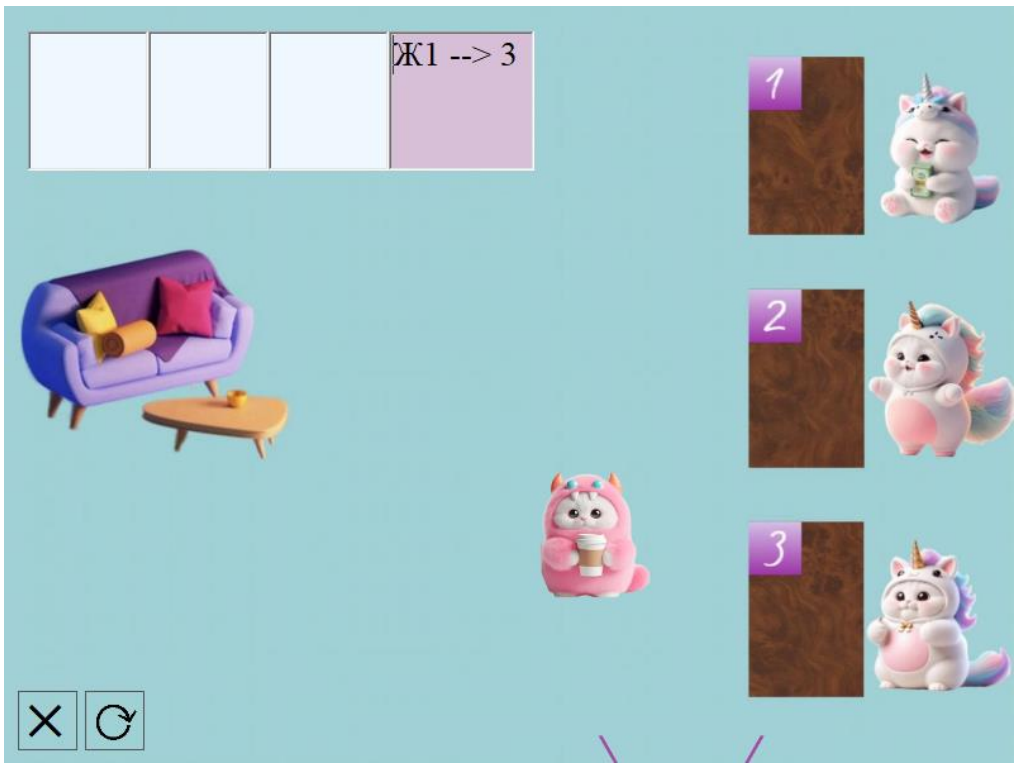


Рисунок 4.2.23 – Путь клиента к окну 3.

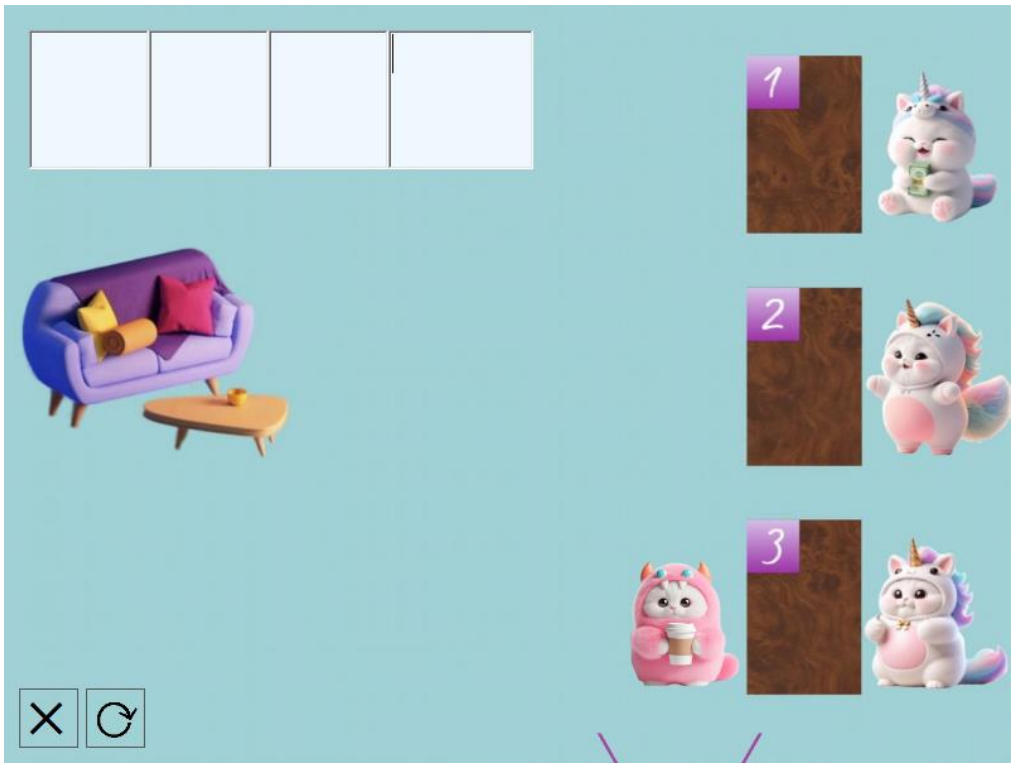


Рисунок 4.2.24 – Путь клиента к окну 3.

17. Случай использования: Начало получения обслуживания.

Предусловие: Приложение запущено, открыто окно приложения «Электронная очередь»;

Тестовый случай: Пользователь получил индивидуальный номер талона;

Ожидаемый результат: Отображается сообщение о начале обслуживания;

Результат представлен на рисунке 4.2.25.

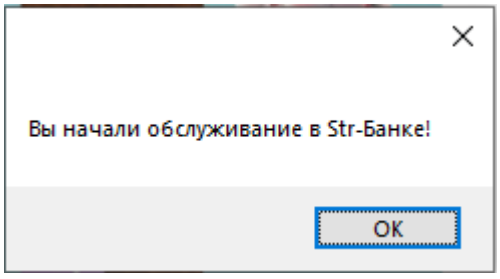


Рисунок 4.2.25 – Начало обслуживания.

18. Случай использования: Завершение обслуживания.

Предусловие: Приложение запущено, открыто окно приложения «Электронная очередь»;

Тестовый случай: Пользователь получил индивидуальный номер талона;

Ожидаемый результат: Отображается сообщение о завершении обслуживания;

Результат представлен на рисунке 4.2.26.

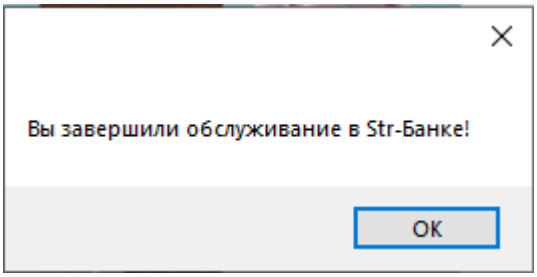


Рисунок 4.2.26 – Завершение обслуживания.

19. Случай использования: Демонстрация движения клиента к выходу из банка.

Предусловие: Приложение запущено, открыто окно приложения «Электронная очередь»;

Тестовый случай: Пользователь получил индивидуальный номер талона;

Ожидаемый результат: Отображается, как с помощью графики изображение движется к выходу из банка;

Результат представлен на рисунках 4.2.27 – 4.2.30.

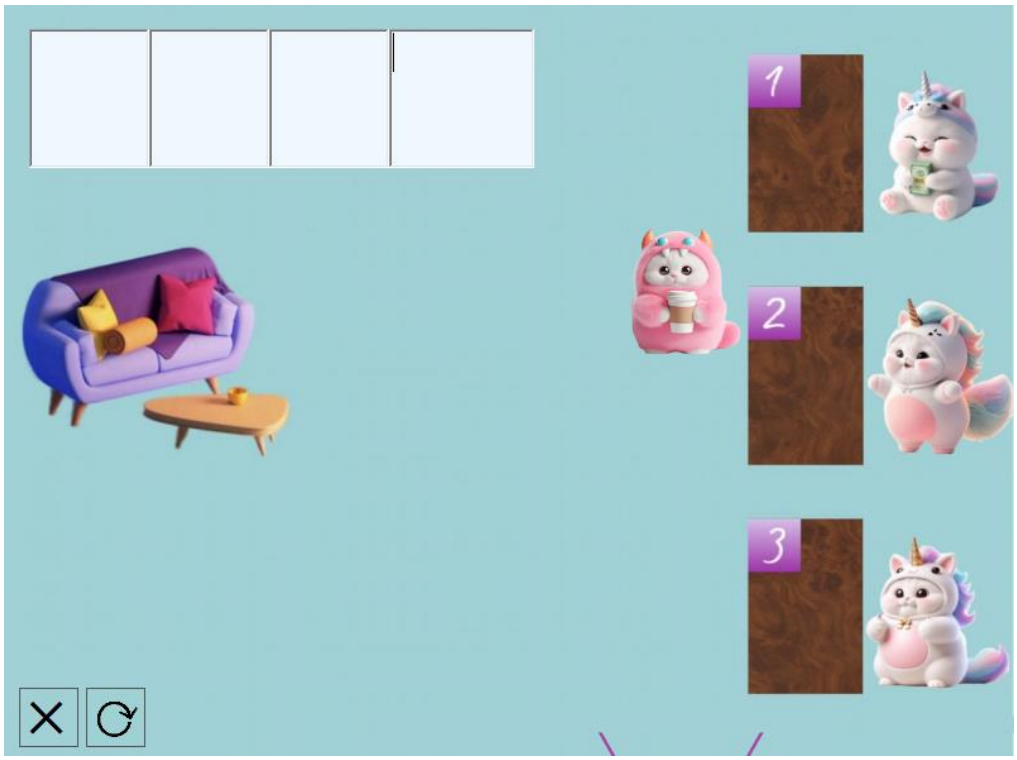


Рисунок 4.2.27 – Путь клиента к выходу.

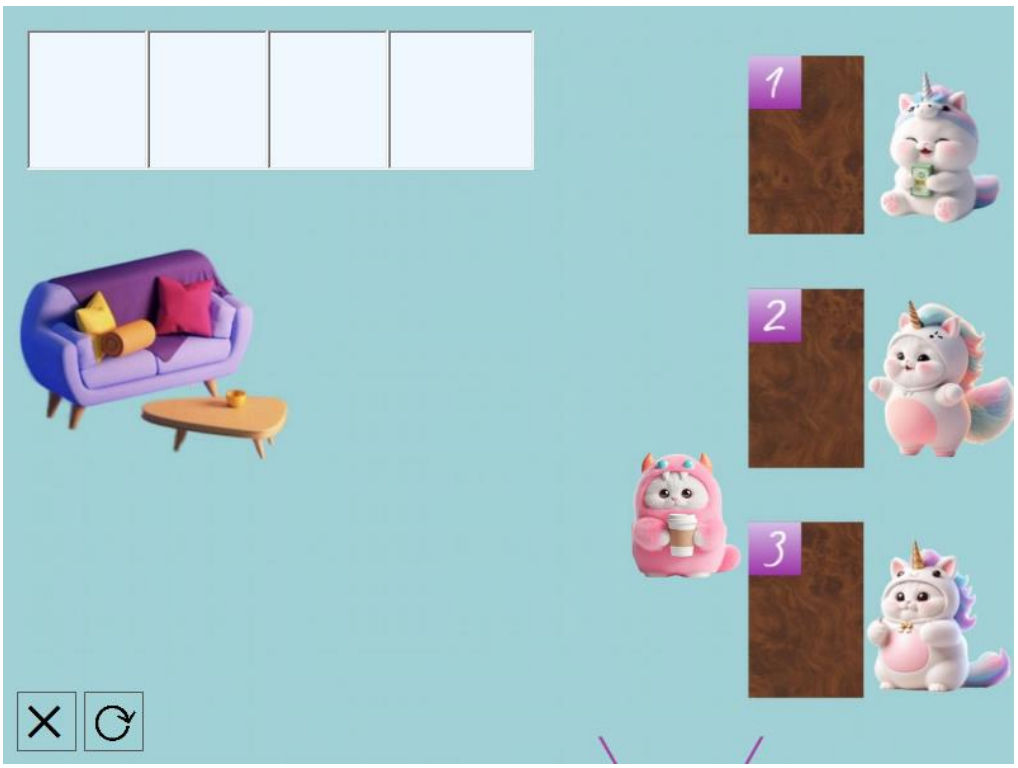


Рисунок 4.2.28 – Путь клиента к выходу.

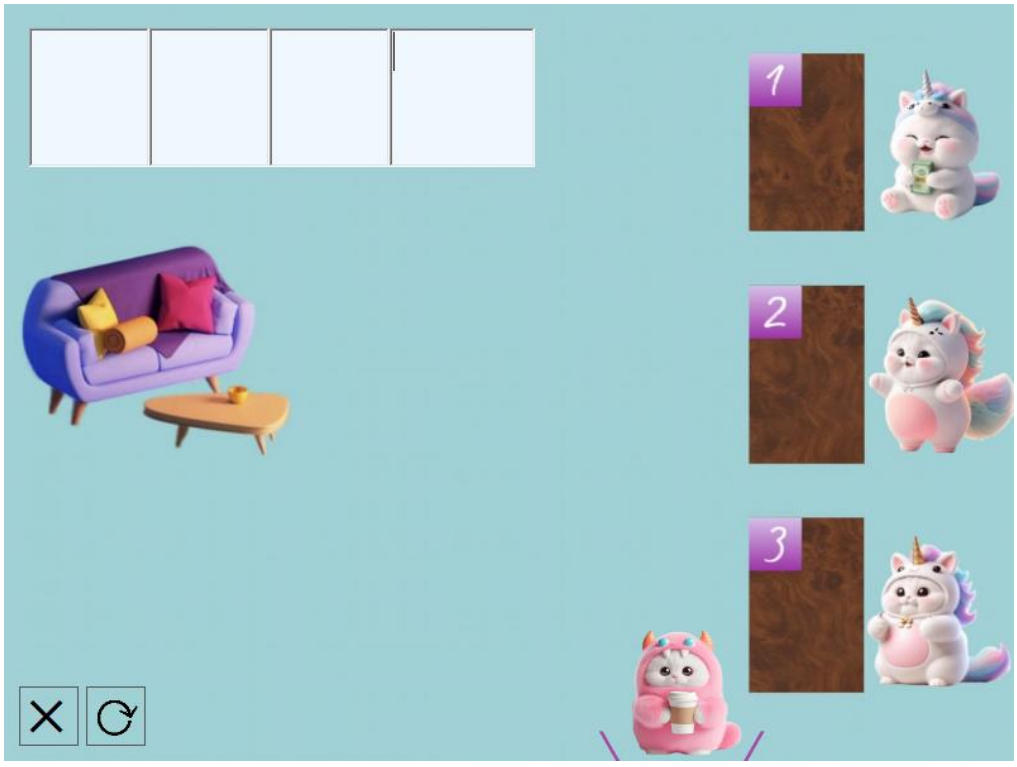


Рисунок 4.2.29 – Путь клиента к выходу.

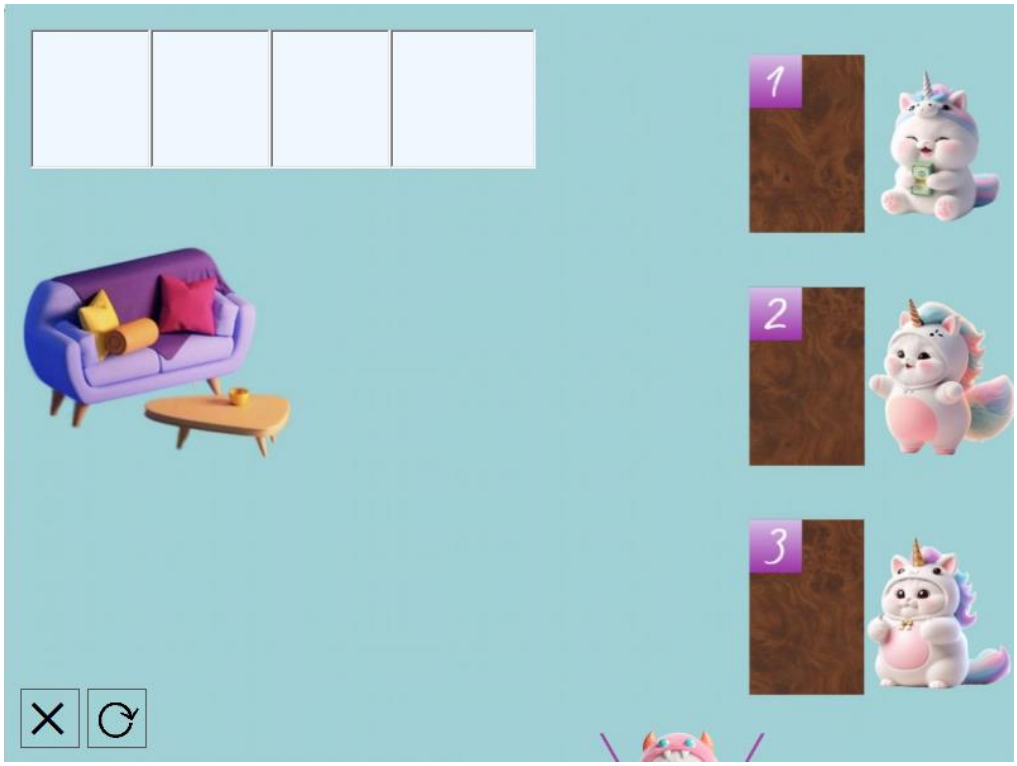


Рисунок 4.2.30 – Путь клиента к выходу.

20. Случай использования: Выход клиента из банка.

Предусловие: Приложение запущено, открыто окно приложения «Электронная очередь»;

Тестовый случай: Пользователь получил индивидуальный номер талона;

Ожидаемый результат: Отображается сообщение о завершении обслуживания;

Результат представлен на рисунке 4.2.31.

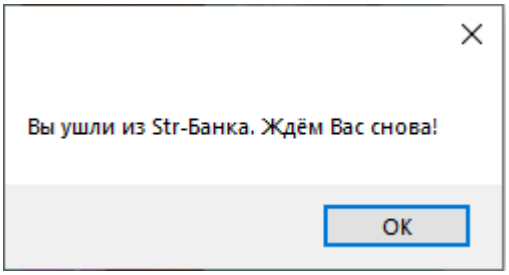


Рисунок 4.2.31 – Выход из банка.

ЗАКЛЮЧЕНИЕ

В результате выполнения курсовой работы было разработано приложение для моделирования работы электронной очереди в банке.

Разработанное приложение предназначено для улучшения понимания работы электронной очереди в банке.

В процессе выполнения курсовой работы на основе исследования предметной области приложения были определены требования к приложению. Для реализации требований к программе разработана логика моделирования; разработан основной алгоритм моделирования; разработана система взаимодействия пользователя и интерфейса программы; спроектирован и реализован графический интерфейс окон программы; проведено функциональное тестирование программы

Все требования, объявленные в техническом задании, были полностью реализованы в данном программном продукте.

Все задачи, поставленные в начале разработки проекта, были решены.

Таким образом, цель работы достигнута.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Медведев, М.А. Программирование на СИ# : учеб. пособие / М. А. Медведев, А. Н. Медведев. – Екатеринбург: Изд-во Урал. ун-та, 2015. – 64 с. ISBN 978-5-7996-1561-1. – Текст: непосредственный.
2. Белик, А. Г. Проектирование и архитектура программных систем: учебное пособие / А. Г. Белик, В. Н. Цыганенко. – Омск: ОмГТУ, 2016. – 96 с. - ISBN 978-5-8149-2258-8. - Текст: непосредственный.
3. Котов, О. М. Язык C#: краткое описание и введение в технологии программирования: учебное пособие / О. М. Котов. – Екатеринбург: Изд-во Урал. ун-та, 2014. – 208 с. ISBN 978-5-7996-1094-4. – Текст: непосредственный.
4. Абрамян М. Э. Visual C# на примерах. – СПб: БХВ-Петербург, 2008. – 496 с.: ил. ISBN 978-5-9775-0266-5. – Текст: непосредственный.
5. Либерти Д. Программирование на C#. – Пер. с англ. – СПб.: Символ-Плюс, 2003. – 688 с., ил. ISBN 5-93286-038-3. – Текст: непосредственный.
6. Биллиг, В. А. Объектное программирование в классах на C# 3.0 [Электронный ресурс] / В.А. Биллиг. - 2-е изд., испр. - Москва: Национальный Открытый Университет «ИНТУИТ», 2016 - 391 с. Полный текст находится в ЭБС "Университетская библиотека ONLINE". – Текст: непосредственный.
7. Мартин, Р. Чистая архитектура. Искусство разработки программного обеспечения / Р. Мартин; пер. с англ. А. Кисилева. – СанктПетербург: Питер, 2018. – 352 с. – ISBN 978-5-4461-0772-8. – Текст: непосредственный.

8. Рихтер, Джеффри CLR via C#. Программирование на платформе Microsoft .NET Framework 4.5 на языке C#. 4-е изд. / Джеффри Рихтер. – М. СПб.: Питер, 2022. – 896 с. – ISBN 978-5-4461-1102-2. – Текст: непосредственный.
9. Программирование на языке C# [Текст]: учебное пособие / И. Л. Александрова, Д. Н. Тумаков; Казанский федеральный университет. - Казань: Изд-во Казанского ун-та, 2017. - 111 с.; 21 см.; ISBN 978-5-00019-914-5: 100 экз. – Текст: непосредственный.
10. Мюллер. Джон Пол. Семпф. Билл, Сфер. Чак. C# для чайников.: Пер. с англ. — СПб. : ООО "Диалектика". 2019. — 608 с. : ил. — Парал. тит. англ. ISBN 978-5-907144-43-9 (рус.) – Текст: непосредственный.
11. Пахомов Б. И. C# для начинающих. — СПб.: БХВ-Петербург, 2014. — 432 с.: ил. ISBN 978-5-9775-0943-5. – Текст: непосредственный.
12. Маклецов С. В. Объектно-ориентированное программирование на языке C#: учебное пособие / С. В. Маклецов. - Казань: Казанский (Приволжский) федеральный университет, 2024. - 132 с. – Текст: непосредственный.
13. Залогова Л. А. Основы объектно-ориентированного программирования на базе языка C#: учебное пособие для СПО / Л. А. Залогова. — 2-е изд., стер. — Санкт-Петербург: Лань, 2021. - 192 с.: ил. ISBN 978-5-8114-7722-7. – Текст: непосредственный.
14. Толстых, С. Г. Объектно-ориентированное программирование с использованием C# [Электронное издание]: учебное пособие / С. Г. Толстых, И. Л. Коробова, Н. В. Майстренко. - Тамбов: Издательский центр ФГБОУ ВО «ТГТУ», 2023. - 1 электрон. опт. диск (CD-ROM). - ил. ISBN 978-5-8265-2615-6. – Текст: непосредственный.

15. Ермолаев, Ф. Б. Инновации в сфере операционного управления очередями розничных клиентов в современных банковских учреждениях. Система электронной очереди / Ф. Б. Ермолаев. – Текст: непосредственный // Молодой ученый. – 2017. – № 24 (158). – С. 31-34. – URL: https://moluch.ru/archive/158/44453/ (дата обращения: 05.10.2024).

ПРИЛОЖЕНИЕ А

Исходный код Form 1:

```
using System;
using System.Windows.Forms;

namespace Курсовая_работа
{
   public partial class Form1 : Form
   {
      public Form1()
      {
         InitializeComponent();
      }

      private void button1_Click(object sender, EventArgs e)
      {
         this.Hide();
         Form2 form2 = new Form2();
         form2.ShowDialog();
      }
   }
}
```

Исходный код Form 2:

```
using System;
using System.Drawing;
using System.IO;
using System.Windows.Forms;

namespace Курсовая_работа
{
   public partial class Form2 : Form
   {
      private int X;
      private int Y;
      private int step = 5;
```

```
        private bool goingUp = false;

        private Image img;

        public Form2()
        {
            InitializeComponent();

            X = this.ClientSize.Width - 200;
            Y = this.ClientSize.Height - 135;
            timer1.Interval = 60;
            timer1.Start();

            string imagePath =
Path.Combine(AppDomain.CurrentDomain.BaseDirectory, "1.png");
            img = Image.FromFile(imagePath);
        }

        private void timer1_Tick(object sender, EventArgs e)
        {
            if (X > 140)
            {
                X -= step;
            }

            else if (X <= 150 && !goingUp)
            {
                goingUp = true;
            }

            else if (goingUp && Y > this.ClientSize.Height - 200)
            {
                Y -= step;
            }

            else
            {
                this.Hide();
                Form3 form3 = new Form3();
```

```
            form3.Show();
            timer1.Stop();
        }
        this.Invalidate();
    }

    protected override void OnPaint(PaintEventArgs e)
    {
        base.OnPaint(e);
        e.Graphics.DrawImage(img, X, Y, 60, 60);
    }
  }
}
```

Исходный код Form 3:

```
using System;
using System.Drawing;
using System.IO;
using System.Windows.Forms;

namespace Курсовая_работа
{
  public partial class Form3 : Form
  {
    private int squareHeight = 0;
    private const int MaxHeight = 70;
    private bool isDrawing = false;

    private int[] counters = new int[10];
    private const string filePath = "counters.txt";

    private const string talonFilePath = "talon.txt";

    private string category;
    private int ticketNumber;

    public Form3()
    {
```

```
    InitializeComponent();

    this.DoubleBuffered = true;
    timer1.Interval = 60;
    timer2.Interval = 1500;
    timer1.Start();

    TerminalLoad();
}

private void TerminalLoad()
{
    Terminal terminal = new Terminal(filePath, counters);
    terminal.Load();
}

private void TerminalSave()
{
    Terminal terminal = new Terminal(filePath, counters);
    terminal.Save();
}

protected override void OnFormClosing(FormClosingEventArgs e)
{
    base.OnFormClosing(e);
    TerminalSave();
}

private void SaveTicketNumber(string category, int ticketNumber)
{
    using (StreamWriter writer = new StreamWriter(talonFilePath, true))
    {
        writer.WriteLine($"{category}{ticketNumber}");
    }
}

void timer1_Tick(object sender, EventArgs e)
{
    if (isDrawing)
```

```
        {
            squareHeight += 5;

            if (squareHeight == MaxHeight)
            {
                isDrawing = false;
                timer1.Stop();
                timer2.Start();
            }
            this.Invalidate();
        }
    }

    private void timer2_Tick(object sender, EventArgs e)
    {
        timer2.Stop();
        this.Hide();
        Form5 form5 = new Form5();
        form5.Category = category;
        form5.TicketNumber = ticketNumber;
        form5.Show();
    }

    protected override void OnPaint(PaintEventArgs e)
    {
        base.OnPaint(e);

        if (squareHeight > 0)
        {
            e.Graphics.FillRectangle(Brushes.White, 555, 466 - squareHeight, 56,
squareHeight);

            Font font = new Font("Times New Roman", 13);
            StringFormat stringFormat = new StringFormat();
            stringFormat.Alignment = StringAlignment.Center;
            stringFormat.LineAlignment = StringAlignment.Center;

            e.Graphics.DrawString("Талон", font, Brushes.Black, new Rectan-
gleF(555, 466 - squareHeight, 56, squareHeight), stringFormat);
```

```
    }
}

private void button1_Click(object sender, EventArgs e)
{
    int categoryIndex = 0;
    counters[categoryIndex] = (counters[categoryIndex] + 1) % 100;

    category = "A";
    ticketNumber = counters[categoryIndex];

    SaveTicketNumber(category, ticketNumber);

    isDrawing = true;
    timer1.Start();
}

private void button2_Click(object sender, EventArgs e)
{
    int categoryIndex = 1;
    counters[categoryIndex] = (counters[categoryIndex] + 1) % 100;

    category = "Б";
    ticketNumber = counters[categoryIndex];

    SaveTicketNumber(category, ticketNumber);

    isDrawing = true;
    timer1.Start();
}

private void button3_Click(object sender, EventArgs e)
{
    int categoryIndex = 2;
    counters[categoryIndex] = (counters[categoryIndex] + 1) % 100;

    category = "B";
    ticketNumber = counters[categoryIndex];
```

```
    SaveTicketNumber(category, ticketNumber);

    isDrawing = true;
    timer1.Start();
}

private void button4_Click(object sender, EventArgs e)
{
    int categoryIndex = 3;
    counters[categoryIndex] = (counters[categoryIndex] + 1) % 100;

    category = "Г";
    ticketNumber = counters[categoryIndex];

    SaveTicketNumber(category, ticketNumber);

    isDrawing = true;
    timer1.Start();
}

private void button5_Click(object sender, EventArgs e)
{
    int categoryIndex = 4;
    counters[categoryIndex] = (counters[categoryIndex] + 1) % 100;

    category = "Д";
    ticketNumber = counters[categoryIndex];

    SaveTicketNumber(category, ticketNumber);

    isDrawing = true;
    timer1.Start();
}

private void button6_Click(object sender, EventArgs e)
{
    int categoryIndex = 5;
    counters[categoryIndex] = (counters[categoryIndex] + 1) % 100;
```

```
    category = "E";
    ticketNumber = counters[categoryIndex];

    SaveTicketNumber(category, ticketNumber);

    isDrawing = true;
    timer1.Start();
}

private void button7_Click(object sender, EventArgs e)
{
    int categoryIndex = 6;
    counters[categoryIndex] = (counters[categoryIndex] + 1) % 100;

    category = "Ж";
    ticketNumber = counters[categoryIndex];

    SaveTicketNumber(category, ticketNumber);

    isDrawing = true;
    timer1.Start();
}

private void button8_Click(object sender, EventArgs e)
{
    int categoryIndex = 7;
    counters[categoryIndex] = (counters[categoryIndex] + 1) % 100;

    category = "З";
    ticketNumber = counters[categoryIndex];

    SaveTicketNumber(category, ticketNumber);

    isDrawing = true;
    timer1.Start();
}

private void button9_Click(object sender, EventArgs e)
{
```

```
    int categoryIndex = 8;
    counters[categoryIndex] = (counters[categoryIndex] + 1) % 100;

    category = "И";
    ticketNumber = counters[categoryIndex];

    SaveTicketNumber(category, ticketNumber);

    isDrawing = true;
    timer1.Start();
}

private void button10_Click(object sender, EventArgs e)
{
    int categoryIndex = 9;
    counters[categoryIndex] = (counters[categoryIndex] + 1) % 100;

    category = "К";
    ticketNumber = counters[categoryIndex];

    SaveTicketNumber(category, ticketNumber);

    isDrawing = true;
    timer1.Start();
}

private void button11_Click(object sender, EventArgs e)
{
    this.Hide();
    Form4 form4 = new Form4();
    form4.Show();
}

private void button12_Click(object sender, EventArgs e)
{
    this.Hide();
    Form1 form1 = new Form1();
    form1.Show();
}
```

```
        private void button13_Click(object sender, EventArgs e)
        {
            Application.Exit();
        }
    }
}
```

Исходный код Form 4:

```
using System;
using System.Windows.Forms;

namespace Курсовая_работа
{
    public partial class Form4 : Form
    {
        public Form4()
        {
            InitializeComponent();
        }

        private void button1_Click(object sender, EventArgs e)
        {
            this.Hide();
            Form3 form3 = new Form3();
            form3.Show();
        }

        private void button2_Click(object sender, EventArgs e)
        {
            this.Hide();
            Form1 form1 = new Form1();
            form1.Show();
        }

        private void button3_Click(object sender, EventArgs e)
        {
            Application.Exit();
        }
```

```
    }
}
```

Исходный код Form 5:

```
using System;
using System.Windows.Forms;

namespace Курсовая_работа
{
    public partial class Form5 : Form
    {
        public string Category { get; set; }
        public int TicketNumber { get; set; }

        private const string FilePath = "my.txt";

        public Form5()
        {
            InitializeComponent();

            timer1.Interval = 4000;
            timer1.Start();
        }

        private void Form5_Load(object sender, EventArgs e)
        {
            label1.Text = $"{Category}{TicketNumber}";
            SaveMy();
        }

        private void SaveMy()
        {
            Ticket ticket = new Ticket(FilePath);
            ticket.Save(Category, TicketNumber);
        }

        private void button1_Click(object sender, EventArgs e)
        {
```

```
            timer1.Stop();

            Ticket _fileManager = new Ticket(FilePath);
            _fileManager.ClearFile();

            this.Hide();
            Form1 form1 = new Form1();
            form1.Show();
        }

        private void button2_Click(object sender, EventArgs e)
        {
            Ticket ticket = new Ticket(FilePath);
            ticket.ClearFile();

            Application.Exit();
        }

        private void timer1_Tick(object sender, EventArgs e)
        {
            timer1.Stop();

            this.Hide();
            Form6 form6 = new Form6();
            form6.Show();
        }
    }
}
```

Исходный код Form 6:

```
using System;
using System.Collections.Generic;
using System.Drawing;
using System.IO;
using System.Linq;
using System.Windows.Forms;

namespace Курсовая_работа
```

```
{
    public partial class Form6 : Form
    {
        private const string talonFilePath = "talon.txt";
        private const string FilePath = "my.txt";

        private const int initialWaitTime1 = 20;
        private const int initialWaitTime2 = 27;
        private const int initialWaitTime3 = 33;

        private const int waitTimeIncrement1 = 20;
        private const int waitTimeIncrement2 = 27;
        private const int waitTimeIncrement3 = 33;

        private ElectronicQueue queue1;
        private ElectronicQueue queue2;
        private ElectronicQueue queue3;

        private List<string> currentTalons = new List<string>();
        private Display display;

        private Image img;
        private int X = 97;
        private int Y = 170;
        private bool moving = false;
        private int step = 6;
        private RichTextBox sourceRichTextBox = null;
        private bool firstMoveCompleted = false;
        private string currentTalon = "";
        private int destY;
        private int destX;


        public Form6()
        {
            InitializeComponent();

            timer1.Interval = 1000;
            timer2.Interval = 1000;
```

```
            timer3.Interval = 1000;
            timer4.Interval = 5000;
            timer5.Interval = 60;
            timer7.Interval = 60;

            timer1.Start();
            timer2.Start();
            timer3.Start();
            timer5.Start();

            SetStyle(ControlStyles.AllPaintingInWmPaint | Con-
trolStyles.OptimizedDoubleBuffer | ControlStyles.ResizeRedraw | Con-
trolStyles.UserPaint, true);

            string imagePath =
Path.Combine(AppDomain.CurrentDomain.BaseDirectory, "1.png");
            img = Image.FromFile(imagePath);
        }

        private void Form6_Load(object sender, EventArgs e)
        {
            List<string> allTalons = new
List<string>(File.ReadAllLines(talonFilePath));

            queue1 = new ElectronicQueue("Queue 1", initialWaitTime1, waitTimeIn-
crement1);
            queue2 = new ElectronicQueue("Queue 2", initialWaitTime2, waitTimeIn-
crement2);
            queue3 = new ElectronicQueue("Queue 3", initialWaitTime3, waitTimeIn-
crement3);

            foreach (string talon in allTalons)
            {
                if (talon.StartsWith("А") || talon.StartsWith("Б") || talon.StartsWith("В"))
                {
                    queue1.AddTalon(talon);
                }
                else if (talon.StartsWith("Г") || talon.StartsWith("Д") ||
talon.StartsWith("Е"))
```

```
            {
                queue2.AddTalon(talon);
            }
            else if (talon.StartsWith("Ж") || talon.StartsWith("З") || tal-
on.StartsWith("И") || talon.StartsWith("К"))
            {
                queue3.AddTalon(talon);
            }
        }
        richTextBox2.Text = queue1.GetDisplayString();
        richTextBox3.Text = queue2.GetDisplayString();
        richTextBox4.Text = queue3.GetDisplayString();
        display = new Display(richTextBox1, richTextBox2, richTextBox3, rich-
TextBox4, currentTalons, timer4, queue1, queue2, queue3);
    }

    private void timer1_Tick(object sender, EventArgs e)
    {
        UpdateTalons(queue1, richTextBox2);
        CompareTalonsWithFile("my.txt", richTextBox2);
    }

    private void timer2_Tick(object sender, EventArgs e)
    {
        UpdateTalons(queue2, richTextBox3);
        CompareTalonsWithFile("my.txt", richTextBox3);
    }

    private void timer3_Tick(object sender, EventArgs e)
    {
        UpdateTalons(queue3, richTextBox4);
        CompareTalonsWithFile("my.txt", richTextBox4);
    }

    private void timer4_Tick(object sender, EventArgs e)
    {
        display.ClearHighlight();
    }
```

```
    private void UpdateTalons(ElectronicQueue queue, RichTextBox richText-
Box)
    {
        display.UpdateTalons(queue, richTextBox);
    }

    private void CompareTalonsWithFile(string filePath, RichTextBox source)
    {
        string[] fileTalons = File.ReadAllLines(filePath);

        foreach (string line in richTextBox1.Lines)
        {
            string talon = line.Split(' ')[0];
            if (fileTalons.Contains(talon) && !moving)
            {
                moving = true;
                sourceRichTextBox = source;
                currentTalon = talon;
                Invalidate();
                break;
            }
        }
    }

    private void timer5_Tick(object sender, EventArgs e)
    {
        if (moving)
        {
            int initialX = 0;
            if (sourceRichTextBox == richTextBox2)
            {
                initialX = 390;
                destY = 70;
            }
            else if (sourceRichTextBox == richTextBox3)
            {
                initialX = 300;
                destY = 260;
            }
```

```
else if (sourceRichTextBox == richTextBox4)
{
   initialX = 220;
   destY = 440;
}

if (!firstMoveCompleted)
{
   X += step;
   if (X >= initialX)
   {
      firstMoveCompleted = true;
      X = initialX;
   }
}

else
{
   destX = 490;

   if (Y < destY)
   {
      Y += step;
      if (Y > destY) Y = destY;
   }
   else if (Y > destY)
   {
      Y -= step;
      if (Y < destY) Y = destY;
   }

   if (X < destX)
   {
      X += step;
      if (X > destX) X = destX;
   }

   if (X >= destX && Math.Abs(Y - destY) <= step)
   {
```

```
            X = destX;
            Y = destY;
            moving = false;
            firstMoveCompleted = false;
            currentTalon = "";
            timer5.Stop();

            timer6.Start();
            MessageBox.Show("Вы начали обслуживание в Str-Банке!");
          }
        }
        Invalidate();
    }
}

private void timer6_Tick(object sender, EventArgs e)
{
    if (Y == 70)
    {
        timer6.Interval = 10000;
    }
    else if (Y == 260)
    {
        timer6.Interval = 17000;
    }
    else if (Y == 440)
    {
        timer6.Interval = 24000;
    }
    timer6.Tick += (s, args) =>
    {
        timer6.Stop();
        timer7.Start();
        MessageBox.Show("Вы завершили обслуживание в Str-Банке!");
        moving = true;
    };
}

private void timer7_Tick(object sender, EventArgs e)
```

```
{
    destY = 610;
    destX = 490;

    if (X >= destX && Math.Abs(Y - destY) <= step)
    {
        X = destX;
        Y = destY;
        moving = false;
        timer7.Stop();
        MessageBox.Show("Вы ушли из Str-Банка. Ждём Вас снова!");
    }
    else
    {
        if (Y < destY)
        {
            Y += step;
            if (Y > destY)
            {
                Y = destY;
            }
        }
        else if (Y > destY)
        {
            Y -= step;
            if (Y < destY)
            {
                Y = destY;
            }
        }
    }
    Invalidate();
}

protected override void OnPaint(PaintEventArgs e)
{
    base.OnPaint(e);
    e.Graphics.DrawImage(img, X, Y, 100, 100);
}
```

```
        private void button1_Click(object sender, EventArgs e)
        {
            Ticket ticket = new Ticket(FilePath);
            ticket.ClearFile();

            this.Hide();
            Form1 form1 = new Form1();
            form1.Show();
        }

        private void button2_Click(object sender, EventArgs e)
        {
            Ticket ticket = new Ticket(FilePath);
            ticket.ClearFile();

            Application.Exit();
        }
    }
}
```

Исходный код класса Terminal:

```
using System;
using System.IO;

namespace Курсовая_работа
{
    internal class Terminal
    {
        private string filePath;
        private int[] counters;

        public Terminal(string filePath, int[] counters)
        {
            this.filePath = filePath;
            this.counters = counters;
        }

        public void Load()
```

```
        {
            if (File.Exists(filePath))
            {
                var lines = File.ReadAllLines(filePath);
                for (int i = 0; i < counters.Length && i < lines.Length; i++)
                {
                    if (int.TryParse(lines[i], out int count))
                    {
                        counters[i] = count;
                    }
                }
            }
        }

        public void Save()
        {
            File.WriteAllLines(filePath, Array.ConvertAll(counters, count =>
count.ToString()));
        }
    }
}
```

Исходный код класса Ticket:

```
using System.IO;

namespace Курсовая_работа
{
    internal class Ticket
    {
        private readonly string _filePath;

        public Ticket(string filePath)
        {
            _filePath = filePath;
        }

        public void ClearFile()
        {
```

```
            if (File.Exists(_filePath))
            {
                File.WriteAllText(_filePath, string.Empty);
            }
        }

        public void Save(string category, int ticketNumber)
        {
            using (StreamWriter writer = new StreamWriter(_filePath))
            {
                writer.WriteLine($"{category}{ticketNumber}");
            }
        }
    }
}
```

Исходный код класса ElectronicQueue:

```
using System.Collections.Generic;
using System.Text;

namespace Курсовая_работа
{
    internal class ElectronicQueue
    {
        public string QueueName { get; private set; }
        public List<string> Talons { get; private set; } = new List<string>();
        public List<int> WaitTimes { get; private set; } = new List<int>();

        private int _initialWaitTime;
        private int _waitTimeIncrement;

        public ElectronicQueue(string queueName, int initialWaitTime, int
waitTimeIncrement)
        {
            QueueName = queueName;
            _initialWaitTime = initialWaitTime;
            _waitTimeIncrement = waitTimeIncrement;
        }
```

```
        public void AddTalon(string talon)
        {
            Talons.Add(talon);
            WaitTimes.Add(_initialWaitTime);
            _initialWaitTime += _waitTimeIncrement;
        }

        public string GetDisplayString()
        {
            StringBuilder sb = new StringBuilder();
            for (int i = 0; i < Talons.Count; i++)
            {
                sb.Append($"{Talons[i]} - {FormatTime(WaitTimes[i])}\n");
            }
            return sb.ToString();
        }

        private string FormatTime(int seconds)
        {
            int minutes = seconds / 60;
            int secs = seconds % 60;
            return $"{minutes:D2}:{secs:D2}";
        }
    }
}
```

Исходный код класса Display:

```
using System.Collections.Generic;
using System.Drawing;
using System.Windows.Forms;
using System.IO;

namespace Курсовая_работа
{
    internal class Display
    {
        private const string talonFilePath = "talon.txt";
```

```
        private RichTextBox _richTextBox1;
        private RichTextBox _richTextBox2;
        private RichTextBox _richTextBox3;
        private RichTextBox _richTextBox4;
        private List<string> _currentTalons;
        private Timer _timer4;

        private ElectronicQueue _queue1;
        private ElectronicQueue _queue2;
        private ElectronicQueue _queue3;

        private bool isHighlighting = false;

        public Display(RichTextBox richTextBox1, RichTextBox richTextBox2,
RichTextBox richTextBox3, RichTextBox richTextBox4, List<string> cur-
rentTalons, Timer timer4, ElectronicQueue queue1, ElectronicQueue queue2, Elec-
tronicQueue queue3)
        {
            _richTextBox1 = richTextBox1;
            _richTextBox2 = richTextBox2;
            _richTextBox3 = richTextBox3;
            _richTextBox4 = richTextBox4;
            _currentTalons = currentTalons;
            _timer4 = timer4;
            _queue1 = queue1;
            _queue2 = queue2;
            _queue3 = queue3;
        }

        public void UpdateTalons(ElectronicQueue queue, RichTextBox richTextBox)
        {
            if (queue.WaitTimes.Count > 0)
            {
                for (int i = queue.WaitTimes.Count - 1; i >= 0; i--)
                {
                    queue.WaitTimes[i]--;

                    if (queue.WaitTimes[i] <= 0)
                    {
```

```
                int richTextBoxNumber = GetRichTextBoxNumber(richTextBox);

                _richTextBox1.Text += $"{queue.Talons[i]} --> {richText-
BoxNumber}\n";
                _currentTalons.Add(queue.Talons[i]);
                _timer4.Start();
                StartHighlight();

                queue.Talons.RemoveAt(i);
                queue.WaitTimes.RemoveAt(i);

                UpdateTalonFile();
                UpdateTalonDisplay();
            }
        }
    }
    UpdateTalonDisplay();
    richTextBox.Text = queue.GetDisplayString();
}

private int GetRichTextBoxNumber(RichTextBox richTextBox)
{
    if (richTextBox == _richTextBox2) return 1;
    else if (richTextBox == _richTextBox3) return 2;
    else if (richTextBox == _richTextBox4) return 3;
    else return 0;
}

private void UpdateTalonDisplay()
{
    _richTextBox2.Text = _queue1.GetDisplayString();
    _richTextBox3.Text = _queue2.GetDisplayString();
    _richTextBox4.Text = _queue3.GetDisplayString();
}

private void UpdateTalonFile()
{
    List<string> allTalons = new List<string>();
    allTalons.AddRange(_queue1.Talons);
```

```
            allTalons.AddRange(_queue2.Talons);
            allTalons.AddRange(_queue3.Talons);
            File.WriteAllLines(talonFilePath, allTalons);
        }

        private void StartHighlight()
        {
            if (!isHighlighting)
            {
                _richTextBox1.BackColor = Color.Thistle;
                isHighlighting = true;
            }
        }

        private void StopHighlight()
        {
            if (isHighlighting)
            {
                _richTextBox1.BackColor = Color.AliceBlue;
                isHighlighting = false;
            }
        }

        public void ClearHighlight()
        {
            foreach (var talon in _currentTalons)
            {
                _richTextBox1.Text = _richTextBox1.Text.Replace($"{talon} --> 1\n",
"");
                _richTextBox1.Text = _richTextBox1.Text.Replace($"{talon} --> 2\n",
"");
                _richTextBox1.Text = _richTextBox1.Text.Replace($"{talon} --> 3\n",
"");
            }
            _currentTalons.Clear();
            StopHighlight();
            _timer4.Stop();
        }
    }
}
```

ПРИЛОЖЕНИЕ Б

Блок-схемы алгоритмов GetDisplayString (ElectronicQueue) и UpdateTalons (Display).

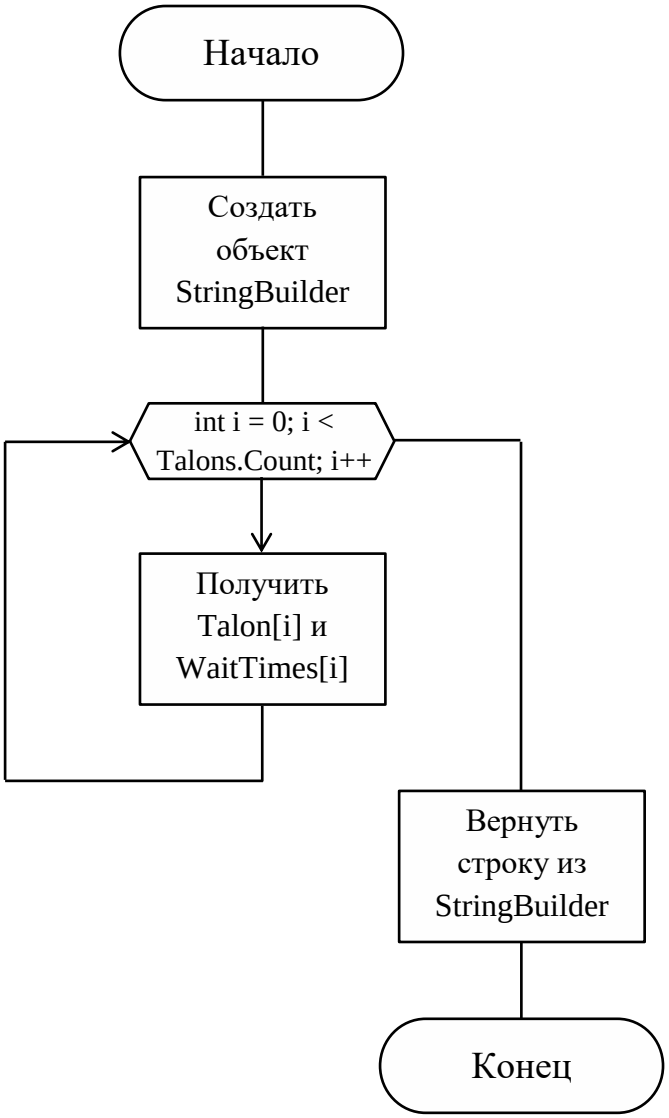
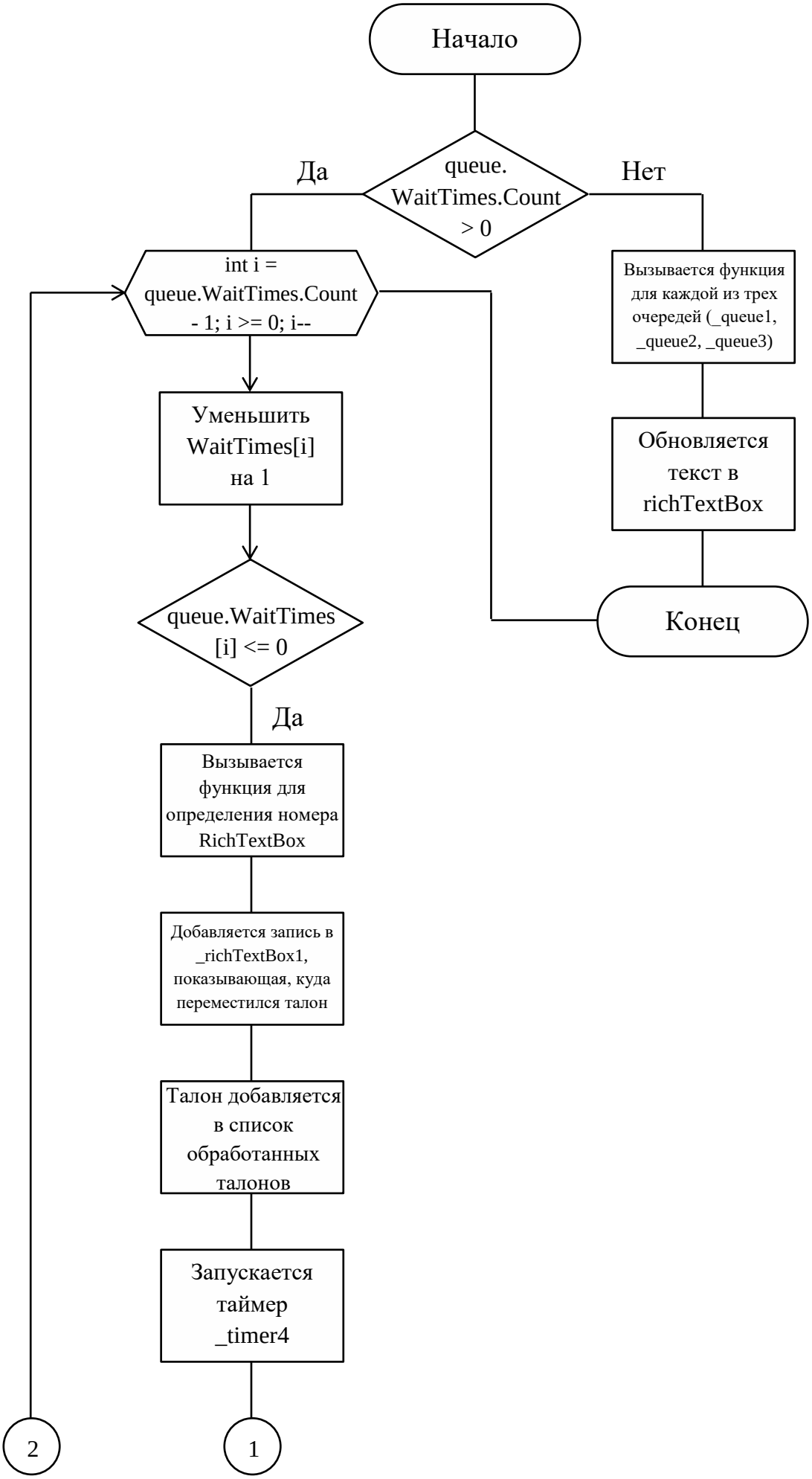


Рисунок Б.1 – Блок-схема алгоритма функции GetDisplayString (ElectronicQueue).


Начало
queue.
WaitTimes.Count
> 0
Да
Нет
int i =
queue.WaitTimes.Count
- 1; i >= 0; i--
Вызывается функция для каждой из трех очередей (_queue1, _queue2, _queue3)
Уменьшить
WaitTimes[i]
на 1
Обновляется
текст в
richTextBox
queue.WaitTimes
[i] <= 0
Конец
Да
Вызывается
функция для
определения номера
RichTextBox
Добавляется запись в
_richTextBox1,
показывающая, куда
переместился талон
Талон добавляется
в список
обработанных
талонов
Запускается
таймер
_timer4
2
1

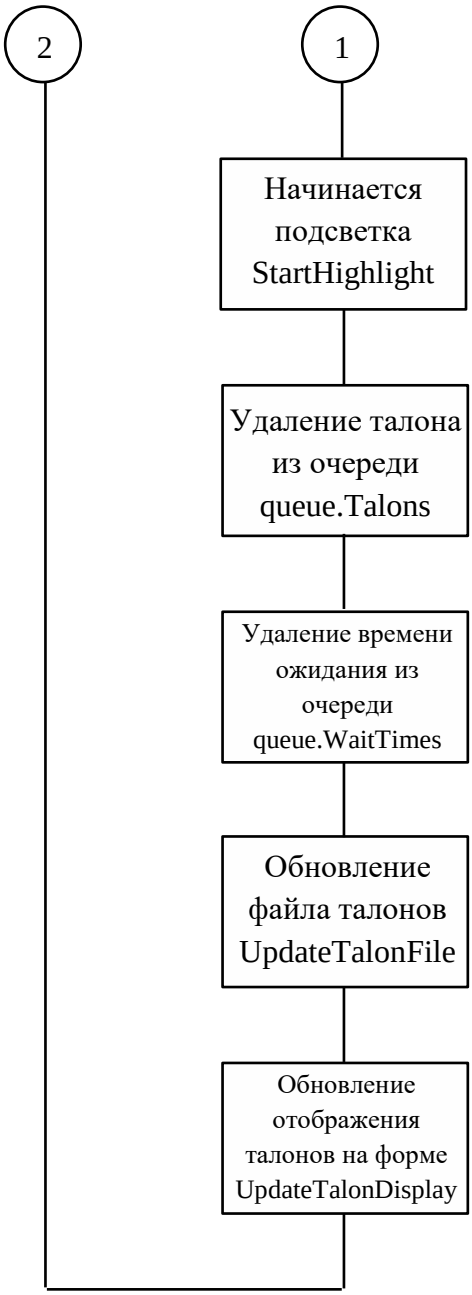


Рисунок Б.2 – Блок-схема алгоритма функции UpdateTalons (Display).