СОДЕРЖАНИЕ

ВВЕДЕНИЕ

В современном мире становится всё сложнее уследить за огромным количеством дел. Умелое владение своим временем позволяет эффективнее работать, а также качественно отдыхать. С ростом количества повседневных обязанностей всё больше людей нуждаются в удобных инструментах, которые помогут не только распланировать свой день, но и замотивировать на их выполнение. В связи с этим возникает необходимость в создании интеллектуальных планировщиков, которые будут сочетать в себе функциональность, удобство и игровые механики для повышения вовлечённости.

Цель работы: разработка интеллектуального планировщика для ПК на языке C++, который позволит распределять задачи и поддерживать мотивацию через визуализацию прогресса.

Задачи работы:

1. провести анализ существующих решений в области планировщиков;
2. сформулировать требования к программируемой системе и выбрать подходящие технологии реализации;
3. разработать программный продукт в соответствии с поставленными требованиями;
4. провести функциональное тестирование приложения;
5. составить отчёт о проделанной работе;
6. сдать отчёт и представить его к защите.

Объектом данного исследования является процесс планирования и мотивации пользователя при управлении личными задачами с использованием современных технологий программирования.

Предметом исследования в данной работе является программная реализация логики интеллектуального планировщика задач с элементами визуализации для ПК на языке C++.

В качестве основных методов исследования применены анализ, синтез, сравнение и моделирование. Практическая реализация поставленной задачи

соответствует основным подходам к разработке программного обеспечения, включая объектно-ориентированный подход.

Информационной базой исследования являются открытые источники, в том числе доступные в сети Интернет, а также материалы курса «Технологии индустриального программирования», доступные через систему дистанционного обучения МИРЭА — российского технологического университета.

В данном отчете будет представлен процесс разработки программного продукта, в том числе теоретический обзор области и системы, технологическое проектирование и описание системы. а также непосредственно результаты разработки.

1 ОБЗОР СУЩЕСТВУЮЩИХ РЕШЕНИЙ

Сам по себе планер — блокнот для организации своего времени, задач и встреч. Люди, которые используют бумажные планеры традиционно придерживаются следующей структуры:

1. страничка для личной информации;
2. календарь на год;
3. планы на месяц;
4. планы на неделю;
5. планы на день;
6. дополнительные листы для личных нужд.

Подобная организация способствует эффективному распределению времени, оптимизируя работу. В особенности ведение ежедневника помогает творческим людям снизить стресс после рабочего дня и сконцентрироваться на поставленных задачах. Бумажный носитель безусловно справляется с базовыми функциями планирования, однако имеет ряд существенных недостатков:

1. ограниченность физического пространства;
2. подверженность механически повреждениям;
3. неисправимость ошибок, нарушающих общую структуру записей;
4. значительные временные затраты на ведение и оформление;
5. необходимость постоянного ношения с собой;

В современном мире цифровых технологий существует множество приложений для планирования и организации времени. Рынок программного обеспечения предлагает различные решения: от классических планировщиков до инновационных разработок с элементами геймификации. Данный подход способствует вовлечённости в процесс ежедневного пополнения планера и обеспечивает устойчивую мотивацию. Существующие приложения-планировщики можно разделить на несколько условных категорий:

1. минималистичные с упрощённым дизайном и списком;
2. классические с традиционным подходом к организации задач;

3. игровые с элементами геймификации и визуализации прогресса.

Большинство существующих приложений следуют более строгому деловому стилю. При этом на рынке практически отсутствуют решения, объединяющие современный подход к мотивации и функциональность стандартных планировщиков. Рассмотрим некоторые популярные примеры.

Todoist [1].

Todoist является одним из наиболее популярных представителей категории классических планировщиков. Данное приложение, разработанное компанией Doist, демонстрирует традиционный подход к управлению задачами и организации времени.

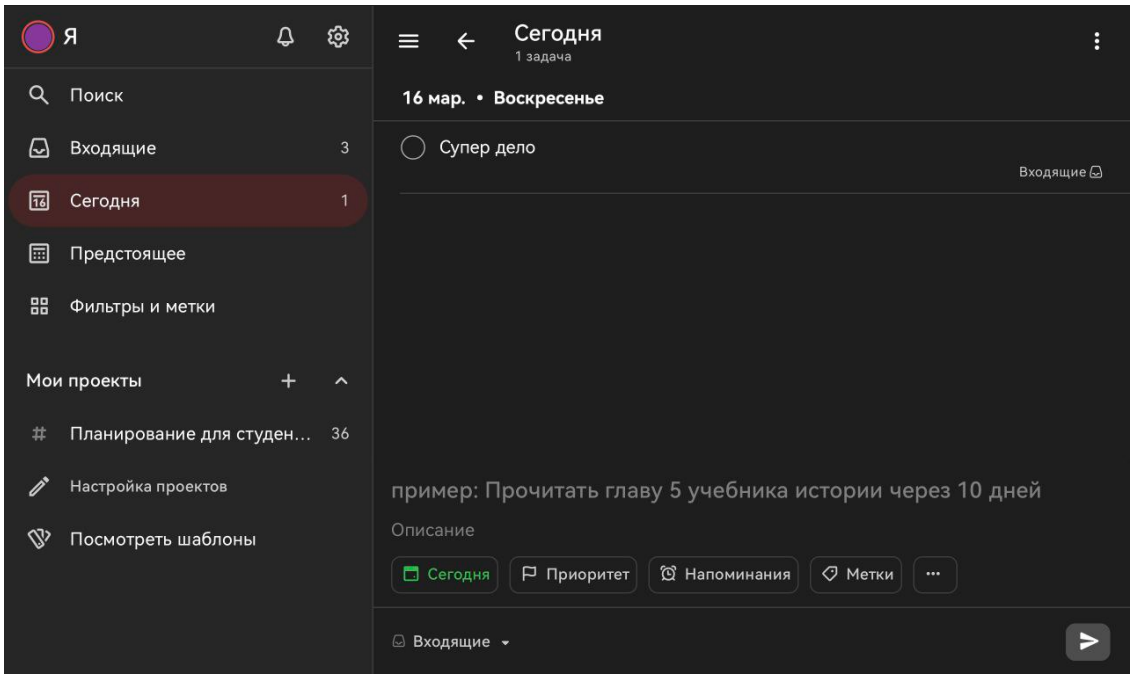

Рисунок 1.1 – Интерфейс Todoist

Основные преимущества:

1. интуитивно понятная система приоритета задач;
2. минималистичный интерфейс с акцентом на функциональность;
3. иерархическая система организации задач и проектов;
4. гибкие настройки повторяющихся задач;
5. продвинутая система тегов и фильтров;
6. кроссплатформенность;
7. встроенные инструменты для совместной работы.

Ограничения:

1. отсутствие элементов геймификации и визуализации;
2. излишне формальный интерфейс без элементов персонализации;
3. ограниченный функционал в бесплатной версии;
4. отсутствие управления временными блоками;
5. высокий порог входа для неподготовленных пользователей.

Habitica

Habitica представляет собой инновационное решение в категории игровых планировщиков, трансформирующее процесс управления задачами в ролевую игру с элементами фэнтези. Разработчики приложения успешно интегрировали механики RPG-игр в систему планирования.

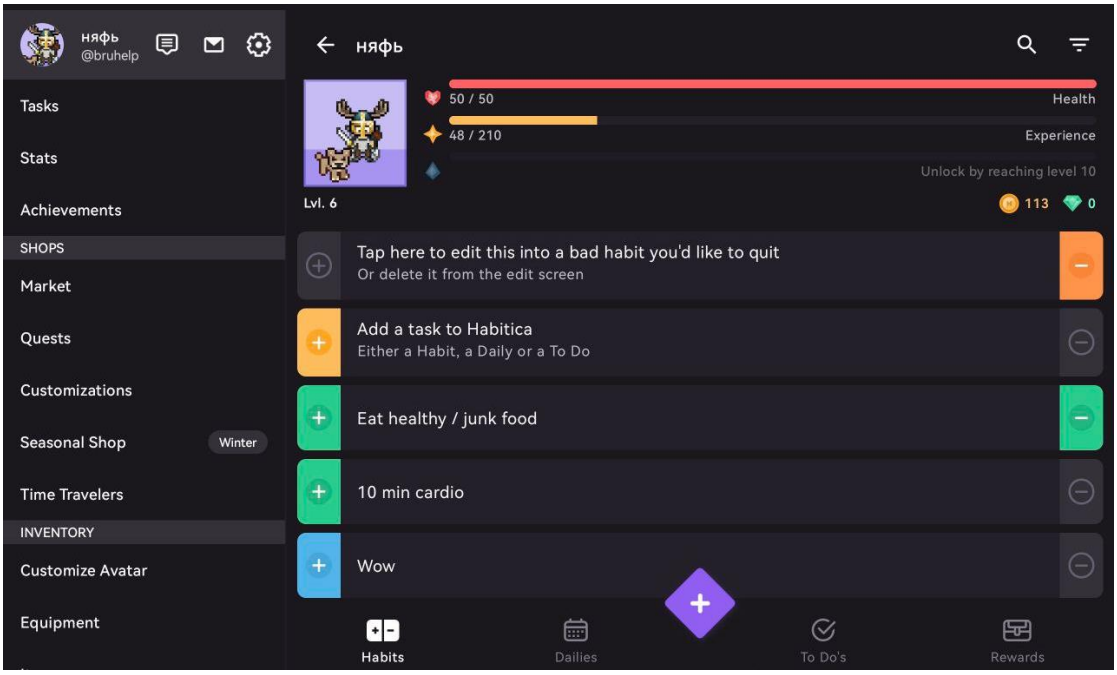

Рисунок 1.2 – Интерфейс мобильного приложения “Habitica”

Основные преимущества:

1. визуальное представление пользователя в виде игрового персонажа;
2. система уровней и характеристик, развивающихся при выполнении задач;
3. высокий уровень вовлеченности пользователей;
4. дополнительная мотивация через систему достижений;
5. социальное взаимодействие и взаимная поддержка;

6. разнообразие игровых механик;
7. элементы соревнования между пользователями.

Ограничения:

1. чрезмерно нагруженный интерфейс;
2. задачи направлены на создание привычек, не рассчитаны на далёкое планирование;
3. система штрафов за невыполненные задачи может демотивировать;
4. средневековая тематика может не соответствовать вкусам целевой аудитории.

Forest [3].

Forest представляет собой уникальное приложение для тайм-менеджмента, сочетающее функции планировщика с экологической концепцией. Данное решение использует метафору выращивания деревьев как визуализацию продуктивно проведенного времени.

Рисунок 1.3 – Интерфейс мобильного приложения “Forest”

Основные преимущества:

1. наглядная визуализация прогресса;
2. таймер фокусировки с визуализацией в виде растущего дерева;
3. простой и интуитивно понятный интерфейс;
4. система блокировки отвлекающих приложений;
5. позитивная система мотивации.

Ограничения:

1. ограниченный функционал планирования задач;
2. фокус преимущественно на краткосрочных целях;
3. отсутствие детальной статистики и аналитики;
4. монотонность игрового процесса при длительном использовании.

Проведённый анализ существующих решений показывает, что на рынке существует явный пробел между классическими планировщиками с их строгой функциональностью и игровыми приложениями с избыточной геймификацией. Большинство приложений либо пренебрегают элементами вовлечения пользователя, либо чрезмерно усложняют процесс планирования игровыми механиками.

Разрабатываемое приложение призвано заполнить эту нишу, предлагая сбалансированное решение, которое сочетает:

1. отсутствие детальной статистики и аналитики;
2. функциональность классического планировщика;
3. эстетичный минималистичный дизайн;
4. умеренную геймификацию через космическую тематику;
5. позитивную систему мотивации без штрафов;
6. интуитивно понятный интерфейс.

Такой подход позволит создать приложение, отвечающее современным требованиям пользователей и устраняющее основные недостатки существующих решений.

2 ОПИСАНИЕ ПРОГРАММИРУЕМОЙ СИСТЕМЫ

Для корректного выполнения курсовой работы мной были разработаны требования программируемой системы. В таблице 2.1 представлены требования к программируемой системе.

Таблица 2.1 – Требование к продукту

№	Требование	Значение
1	Язык программирования	C++
2	Корректность работы	Приложение запускается и поддерживает стабильный цикл работы от момента старта до завершения
3	Применение принципов объектно-ориентированного программирования	При написании приложения, как минимум, были использованы классы в C++, объектный подход к проектированию системы, а также инкапсуляция
4	Интерфейс пользователя	Создан интерфейс пользователя, поддерживающий корректный пользовательский опыт и содержащий все необходимые пояснения к работе и эксплуатации
5	Инструкция по эксплуатации	Написана инструкция по эксплуатации, содержащая, в том числе, основные рекомендации по использованию и пояснения к возможным ошибкам в программе
6	Фреймворк	Qt
7	Управление вводом	Клавиатура ПК и мышь ПК
8	Визуализация прогресса	Система выполненных задач посредством визуализации в виде звёздного неба, где каждый объект является выполненной задачей
9	Реализация сохранения данных	При выходе из оконного приложения записанные и выполненные задачи будут сохраняться локально в папке с ресурсами проекта, что обеспечивает их корректную загрузку без указания абсолютных путей
10	Возможность просматривать историю выполнения задач	Программа должна позволять отмеченные задачи просмотреть не только в окне с визуализацией процесса в виде звёздного неба, но и в окне с выполненными задачами

Для описания работы системы были использованы диаграммы состояний (Рисунок 2.1) и классов системы (Рисунок 2.2).

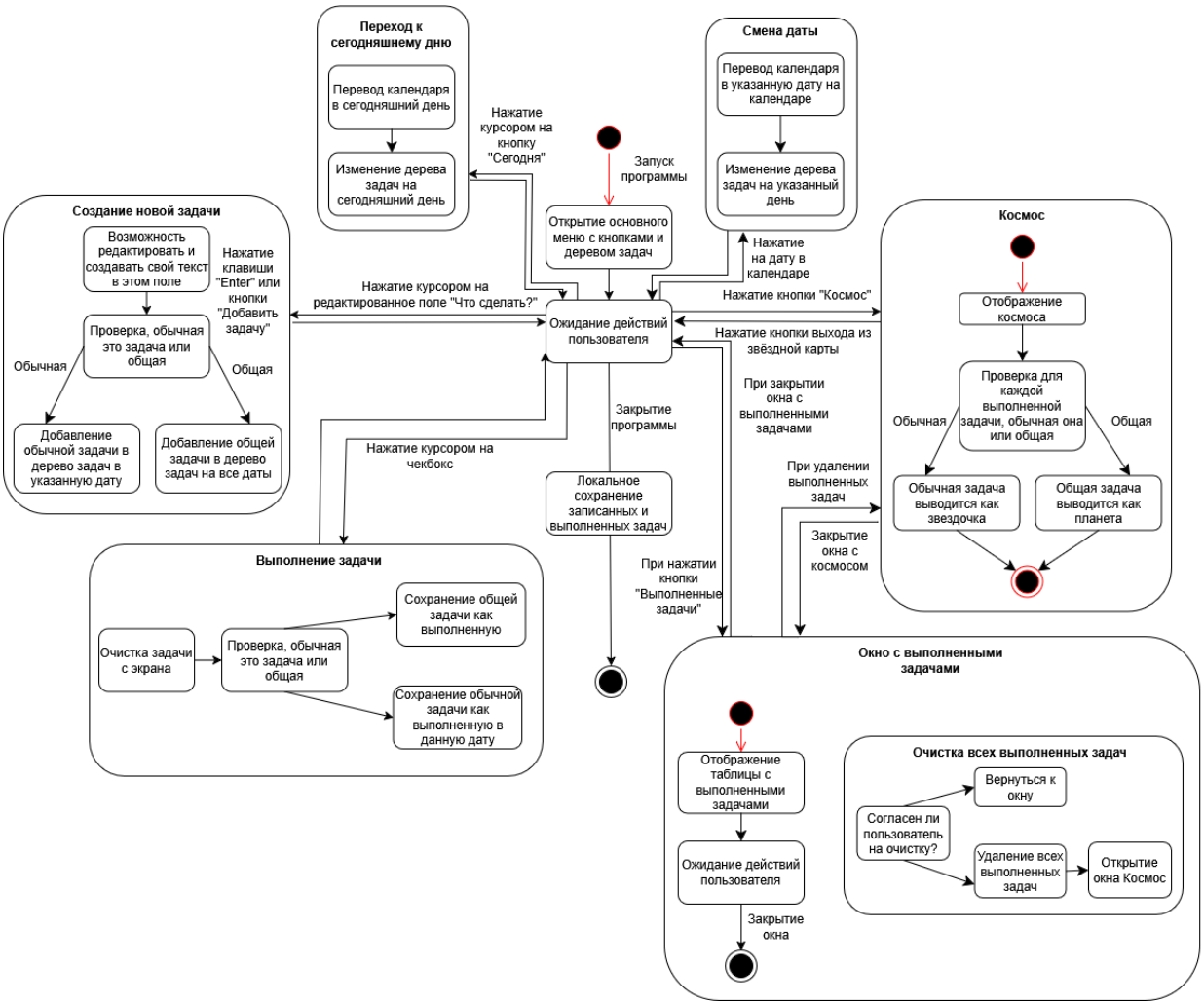

Рисунок 2.1 – Диаграмма состояний системы

Данная диаграмма описывает состояния системы во время работы приложения для планирования задач. После запуска приложения открывается основное меню, где отображается дерево задач, и система переходит в состояние ожидания действий пользователя. В этом состоянии пользователь может выполнять различные действия: создавать новые задачи, удалять предыдущие, отмечать задачи как выполненные, переключаться между датами в календаре, переходить к сегодняшнему дню, а также открывать дополнительные окна: «Космос» (визуализация прогресса) и выполненные задачи.

Если пользователь нажимает кнопку «Сегодня», календарь автоматически переходит к текущей дате, а дерево задач обновляется, отображая задачи, запланированные на этот день. Аналогично, при выборе другой даты в календаре происходит переход к этой дате, и дерево задач обновляется, чтобы показать задачи, относящиеся к выбранному дню.

При создании новой задачи пользователь может выбрать её тип: «Обычная» или «Общая». Если задача обычная, она добавляется к списку задач на выбранную дату. Если задача общая — к списку общих задач, которые не привязаны к дате. Добавление задачи происходит либо нажатием кнопки, либо клавишей Enter.

Выполнение задачи происходит, когда пользователь отмечает задачу как выполненную через чекбокс. Если это обычная задача, она переносится в список выполненных задач на выбранную дату. Если задача общая — она попадает в список выполненных общих задач без даты.

Пользователь может открыть окно «Космос», где отображается визуализация всех выполненных задач — каждая из них представлена в виде звезды или планеты в зависимости от типа.

При открытии окна выполненных задач пользователю отображается список всех завершённых задач. Здесь можно очистить весь список — при этом система спросит подтверждение пользователя, и, если он согласится, список выполненных задач будет полностью очищен, в таком случае ему покажут пустое окно «Космос».

В любой момент пользователь может завершить работу приложения. Перед закрытием все изменения и состояние задач сохраняются.

Таким образом, система постоянно находится в состоянии ожидания действий пользователя, переходя в другие состояния только на время выполнения конкретных операций, после чего возвращается обратно в основное состояние. Все действия пользователя отражаются на интерфейсе.

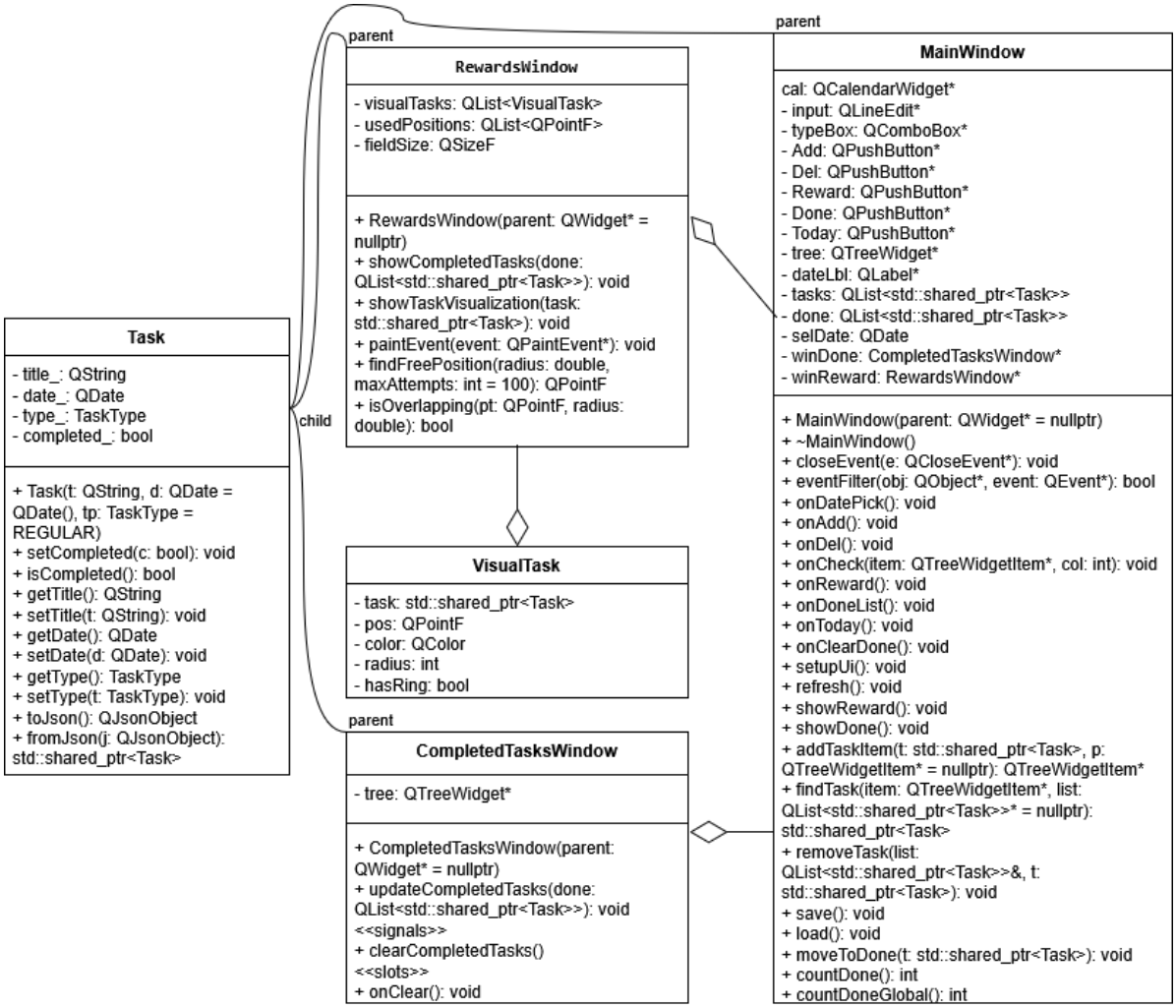

Рисунок 2.2 – Диаграмма состояний системы

Диаграмма классов отражает архитектуру основной логики приложения и взаимосвязи между ключевыми сущностями. Центральный класс системы — MainWindow, который реализует главное окно пользовательского интерфейса и управляет всеми основными процессами работы с задачами. Внутри MainWindow хранятся коллекции объектов Task, которые представляют отдельные задачи пользователя, как активные, так и выполненные. Task — это самостоятельный класс данных, в котором содержится вся информация о задаче: название, дата, тип, статус выполнения, а также методы для сохранения состояния.

MainWindow также агрегирует два вспомогательных окна: RewardsWindow и CompletedTasksWindow. Оба они создаются с указанием MainWindow в качестве родителя, что обеспечивает корректное управление их жизненным циклом. RewardsWindow отвечает за визуализацию выполненных задач в виде космоса — каждая задача отображается в виде планеты или звезды в зависимости от типа. Для

этого внутри RewardsWindow используется коллекция структур VisualTask, каждая из которых хранит указатель на связанную задачу и параметры отображения. Таким образом, RewardsWindow агрегирует VisualTask, а VisualTask, в свою очередь, содержит ссылку на Task. Это позволяет гибко связывать визуальные элементы с данными о задачах.

CompletedTasksWindow отвечает за отображение списка завершённых задач. Он получает на вход коллекцию выполненных задач для отображения их в виде дерева.

Оба вспомогательных окна не имеют собственных данных о задачах, а только отображают или визуализируют те объекты Task, которыми управляет MainWindow.

Связи между классами на диаграмме показывают, что MainWindow напрямую управляет окнами RewardsWindow и CompletedTasksWindow, а также коллекциями Task. RewardsWindow агрегирует VisualTask, причём VisualTask полностью контролируется этим окном. VisualTask связан с Task: каждый визуальный элемент соответствует конкретной задаче. CompletedTasksWindow и RewardsWindow используют Task для отображения, но не управляют их созданием или удалением.

В целом, такая структура классов обеспечивает чёткое разделение ответственности: MainWindow — логика и управление задачами, Task — хранение данных о задачах, RewardsWindow и CompletedTasksWindow — визуализация и отображение. За счёт этого приложение легко поддерживать и расширять.

3 РЕЗУЛЬТАТЫ РАЗРАБОТКИ ПРОГРАММНОГО ПРОДУКТА

3.1 Реализация требований к системе

1. **Язык программирования**

В рамках курсовой работы было разработано приложения для отслеживания привычек, полностью написанное на языке программирования C++. Увидеть реализацию данного требования можно обратившись к исходному коду программы (Приложение А).

2. **Корректность работы**

Работа приложения сопровождается стабильным поведением во всех предусмотренных сценариях: от добавления и удаления задач до переключения между днями и визуализации неба. Обеспечена корректность функционирования приложения. Программа сохраняет состояние при закрытии и восстанавливает данные при следующем запуске. Увидеть реализацию данного требования можно обратившись к скриншотам работы программы (Часть 3.2, Рисунки 3.1-3.5).

3. **Применение принципов объектно-ориентированного программирования**

В ходе разработки интеллектуального планировщика были применены ключевые принципы объектно-ориентированного программирования (ООП), включая инкапсуляцию (например, доступ к приватным полям класс Task), наследование (для графических компонентов Qt) и полиморфизм (через сигналы и слоты). Основные сущности системы (привычки, статистика, трекер, пользовательский интерфейс) представлены в виде отдельных классов. Увидеть реализацию данного требования можно обратившись к исходному коду программы (Приложение А).

4. **Интерфейс пользователя**

Интерфейс пользователя был разработан с учетом простоты и наглядности: есть кнопки управления, визуальное пространство прогресса в стиле космического пространства и понятный список задач с удобным календарём. Пользователь может легко переключаться между днями с помощью календаря. Увидеть реализацию данного требования можно обратившись к скриншотам работы программы (Часть 3.2, Рисунки 3.1-3.5).

5. Инструкция по эксплуатации

Для обеспечения понимания пользователем всех аспектов работы с интеллектуальным планировщиком была подготовлена подробная инструкция по эксплуатации. Она содержит информацию по запуску приложения, добавлению задач, отметке их выполнения, открытие космического пространства и просмотр выполненных задач. Увидеть реализацию данного требования можно обратившись к специально отведенной части (3.3 Инструкция по эксплуатации).

6. Фреймворк

Вся программа выполнялась в фреймворк QT, что позволило свободно пользоваться стандартными виджетами и создавать полноценную программу, которую можно открыть на любом устройстве. Увидеть реализацию данного требования можно обратившись к исходному коду программы (Приложение А).

7. Управление вводом

Управление вводом в приложении реализовано с использованием клавиатуры ПК, что обеспечивает точный и отзывчивый прием команд пользователя. А также использование мыши понятно любому обывателю, с помощью которой можно взаимодействовать с элементами интерфейса. Увидеть реализацию данного требования можно обратившись к исходному коду программы (Приложение А).

8. Визуализация процесса

Каждая задача в зависимости от её типа приобретает в окне «Космос» материальное преображение либо в качестве планеты различных цветов и размеров, либо звезды в пастельных неярких тонах. Это требование позволяет пользователю анализировать свой прогресс через интересное космическое

оформление и мотивировать себя на дальнейшие достижения целей. Реализацию этого требования можно увидеть, обратившись к скриншотам работы программы (Часть 3.2, Рисунок 3.3).

9. Реализация сохранения данных

Даже при выходе из оконного приложения данные о текущих и выполненных задачах сохраняются в файлах JSON, что обеспечивает их сохранность и корректную загрузку без указания путей. Увидеть реализацию данного требования можно обратившись к исходному коду программы (Приложение А).

10. Возможность просматривать историю выполнения задач

Программа обеспечивает доступ к списку всех завершённых задач за весь период времени. Пользователь имеет отдельное окно, в котором может просмотреть название, тип и дату задачи. Есть возможность очищения истории при необходимости, в таком случае сбрасывается весь прогресс. Реализацию этого требования можно увидеть, обратившись к скриншотам работы программы (Часть 3.2, Рисунок 3.4).

3.2 Функциональное тестирование программного продукта

Именно этот класс создаёт и управляет всеми основными компонентами интерфейса, хранит коллекции задач, обрабатывает пользовательские действия, а также координирует взаимодействие между окнами (RewardsWindow, CompletedTasksWindow) и задачами (Task). (Приложение А, Листинг А.4).

После запуска программы пользователю отображается окно интерфейса меню (рис 3.1), содержащее следующие основные элементы:

1. календарь, который по умолчанию выставлен на сегодняшний день;
2. поле для ввода задачи;
3. пользователь с помощью выпадающего списка выбирает необходимый тип задачи: обычный или общий, где общий будет распространяться на каждый день, а обычный — только на текущий;
4. справа видно лист, в котором будет выводится активные задачи;

5. под календарём и полем располагаются пять кнопок: добавление задачи, удаление задачи, показ окна «Космос», показ окна «Выполненные задачи», и переход на сегодняшний день.

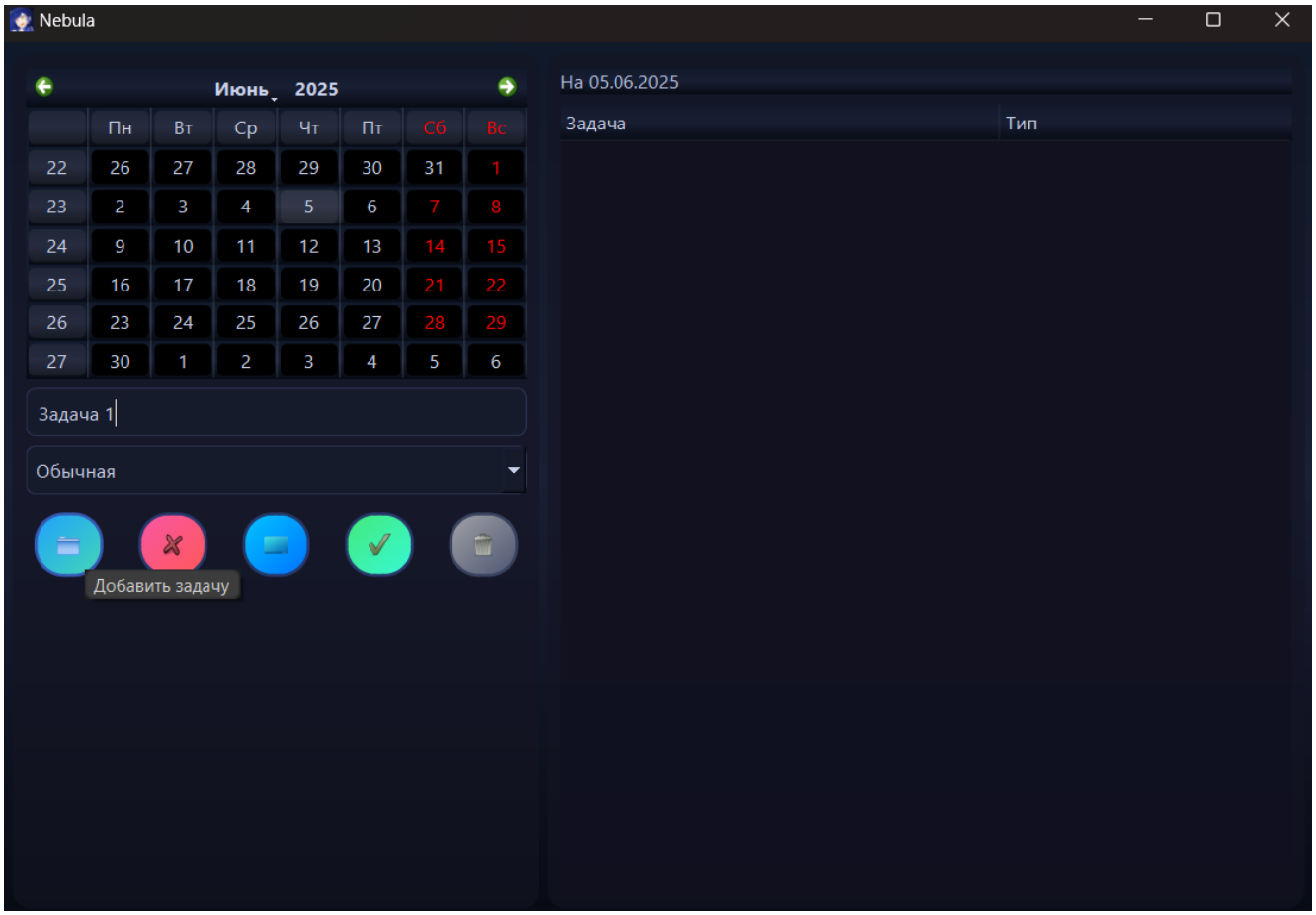

Рисунок 3.1 – Окно интерфейса меню

Для добавления новой задачи пользователь для начала вводит текст задачи, затем выбирает её тип, а после нажимает кнопку «Добавить задачу». В списки задач она появляется вместе с чекбоксом, также выводится выбранный предварительно

тип, по умолчанию стоит «Обычный» (Рисунок 3.2).

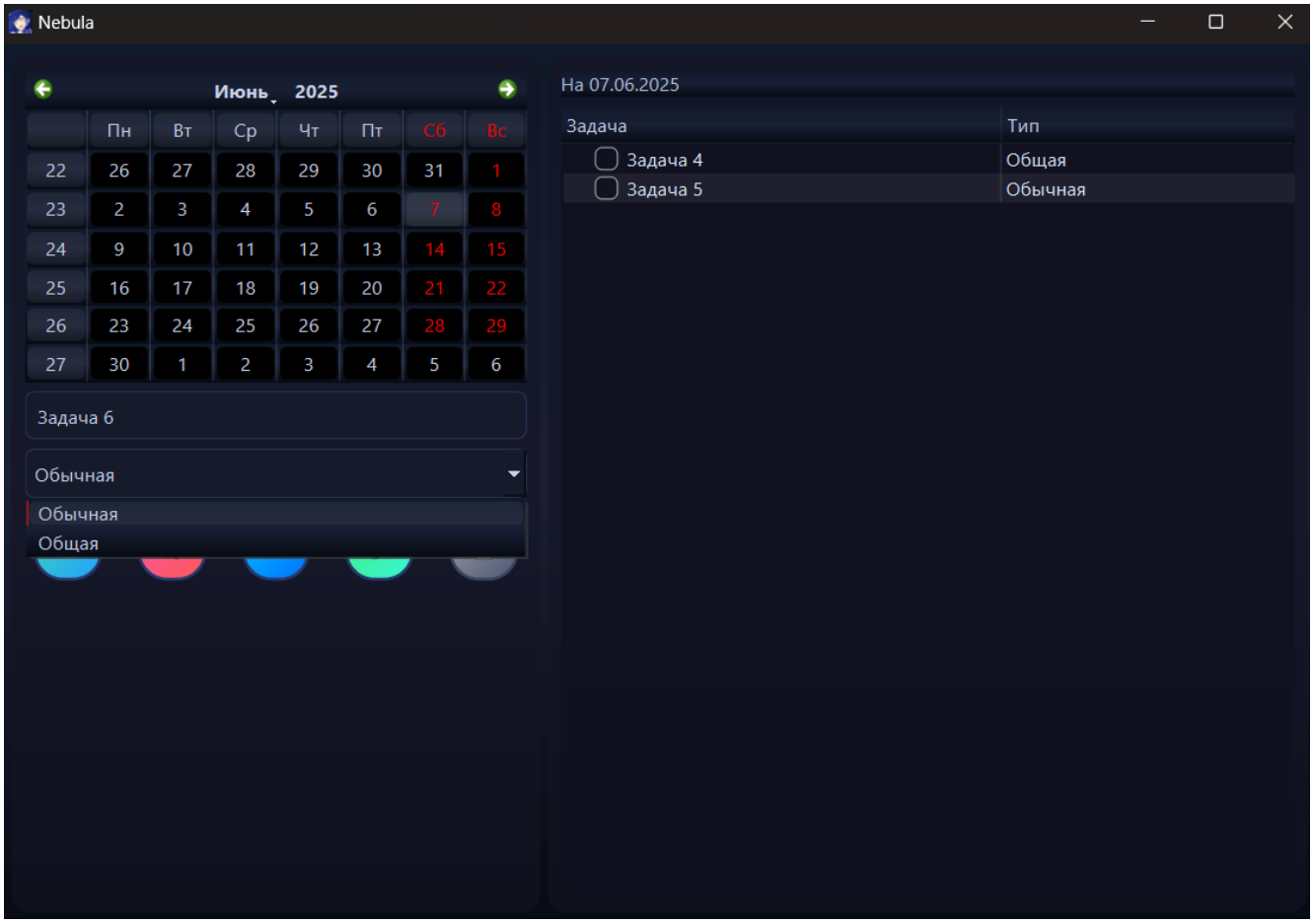

Рисунок 3.2 – Ввод нескольких задач

Удаление задачи происходит посредством нажатия на задачу в списке задач, а затем кнопки удаления. На выполненныхе задачи это никак не повлияет. Все отмеченные задачи по нажатию чекбокса тут же пропадают с экрана. Пользователь их может наблюдать через окно «Космос» по одноимённой кнопке. Планеты обозначают общие, глобальные, задачи, а звёзды — обычные. Планеты формируются самых разных размеров и цветов, как и звёзды. Программа автоматически располагает объекты на звёздном небе без наслаивания их друг на друга. Сверху окна показано, сколько задач выполнил пользователь на данный момент (Рисунок 3.3).

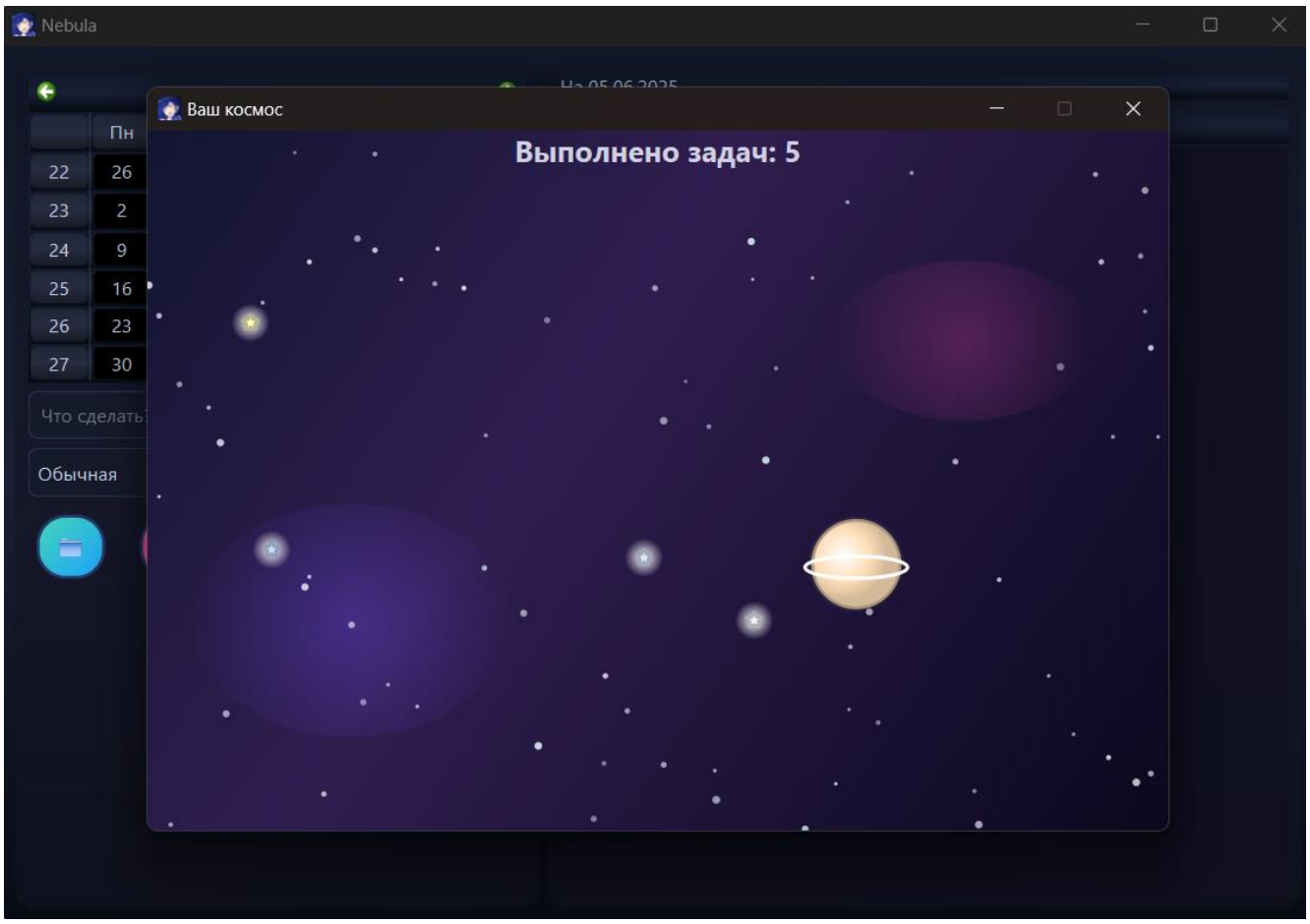

Рисунок 3.3 – Окно «Космос»

При нажатии на кнопку открытия окна «Выполненные задачи» открывается соответствующее окно. В нём будет такой же список задач, как и в основном меню, только с указанием даты выполнения, если задача была типа «Обычная» и, если типа «Общая», то без даты (Рисунок. 3.4).

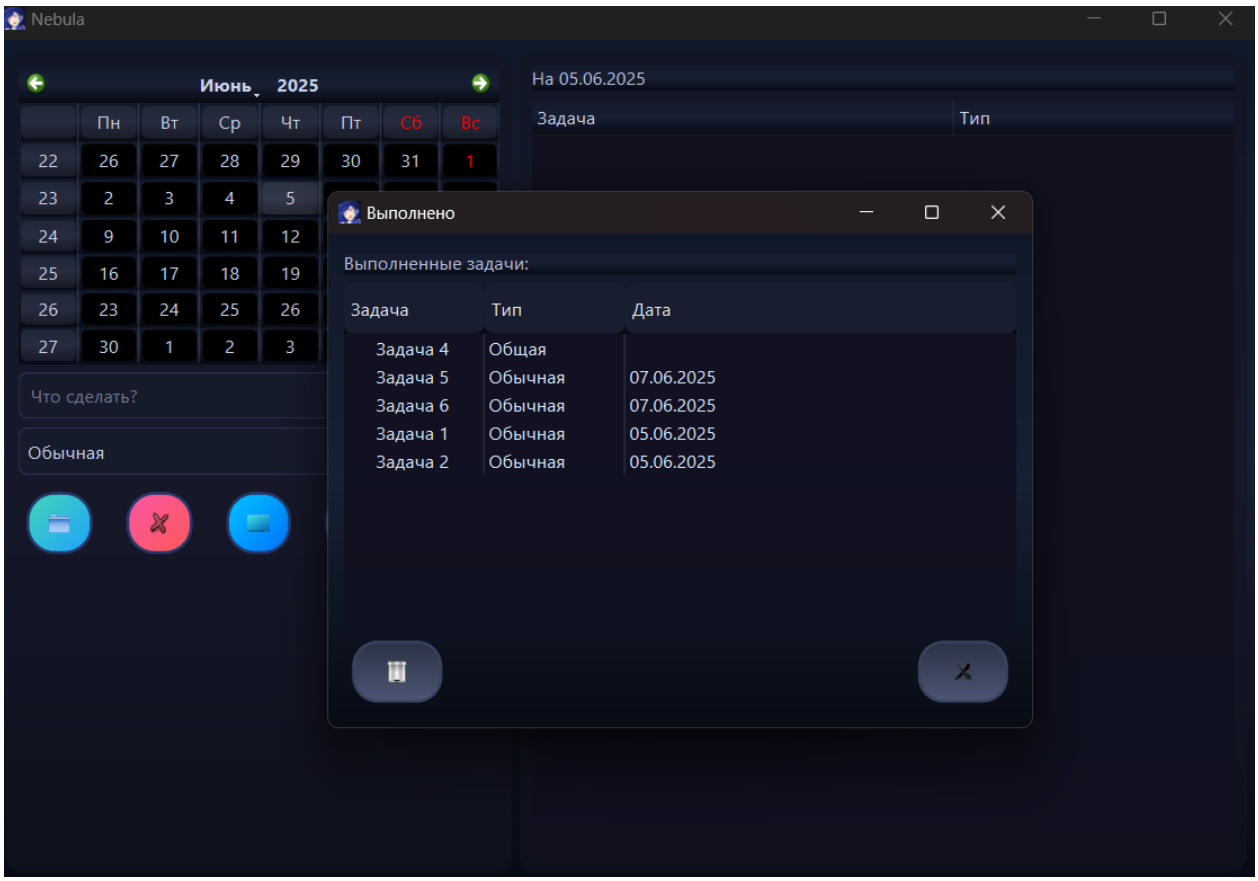

Рисунок 3.4 – Окно «Выполненные задачи»

Также перед пользователем будут две кнопки: «Очистить список» и «Закрыть окно». При очищения истории выполненных задач, пользователя спросят, уверен ли он. В случае, если он уверен, то список пустеет, а пользователю открывается пустое звёздное небо окна «Космос» (Рисунок 3.5)

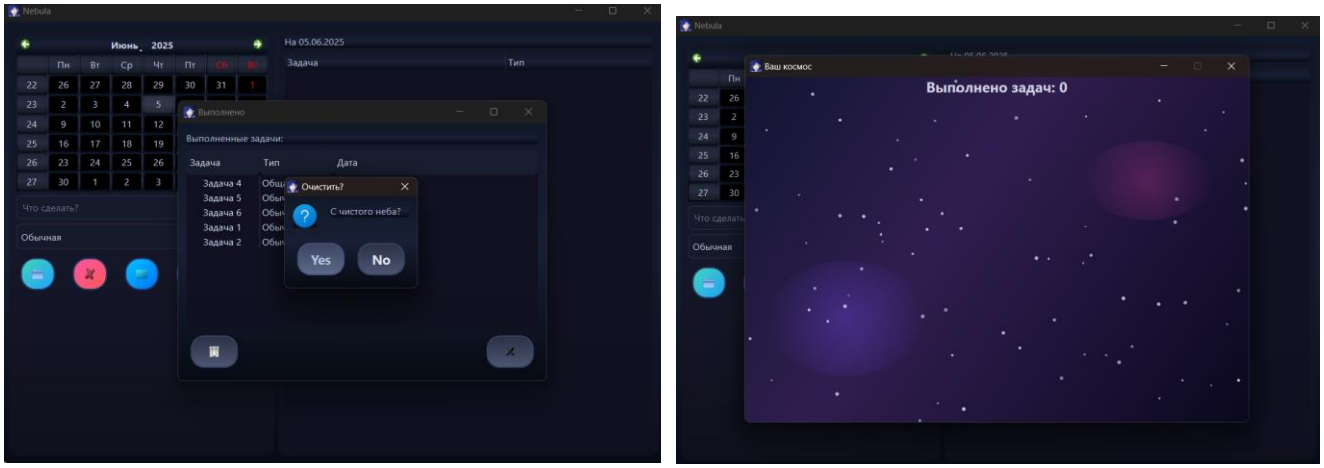

Рисунок 3.5 – Удаление всех выполненных задач

3.3 Инструкция по эксплуатации

После запуска программы пользователь попадает в главное окно приложения. На экране отображаются список задач на текущий день, а также календарь и основные элементы управления.

Пользователю доступны следующие действия:

1. Добавление новой задачи.

Для добавления новой задачи необходимо нажать кнопку «Добавить задачу», ввести название задачи и выбрать её тип (по умолчанию стоит «Обычная»).

2. Удаление задачи.

Чтобы удалить задачу, необходимо нажать на нужную задачу курсором, а после на кнопку «Удалить задачу».

3. Отметка выполнения задачи.

Чтобы выполнить задачу, нужно нажать на чекбокс возле нужной задачи. Задача автоматически перейдёт в список выполненных.

4. Выбор даты.

Для того, чтобы выбрать необходимую дату, требуется нажать на календаре нужную дату. Если после этого нужно вернуться в сегодняшний день, то для этого есть соответствующая крайняя правая кнопка «Сегодня».

5. Просмотр прогресса через визуализацию.

Чтобы просмотреть звёздное небо, создаваемое выполнением задач, необходимо нажать кнопку «Космос», после чего появится соответствующее окно.

6. Просмотр прогресса через список.

Для просмотра списка выполненных задач в главном меню надо нажать «Выполненные задачи». Перед пользователем откроется окно. В случае необходимости можно стереть прогресс с помощью кнопки в этом же окне «Очистить список».

ЗАКЛЮЧЕНИЕ

В результате выполнения курсовой работы был разработан программный продукт — интеллектуальный планировщик Nebula на языке C++ с использованием фреймворка Qt. В ходе работы был проведён анализ существующих решений в области планеров, что позволило выявить их сильные и слабые стороны и сформулировать требования к собственной системе. На этапе проектирования были созданы диаграммы состояний и классов, отражающие логику функционирования приложения и взаимодействие его компонентов.

Разработанная система реализует все заявленные функциональные и технические требования, включая сохранение и визуализацию прогресса, понятный пользовательский интерфейс, а также стабильную работу при взаимодействии с пользовательскими данными. Программа была протестирована на корректность выполнения основных операций, а также снабжена подробной инструкцией по эксплуатации. В ходе работы были применены ключевые принципы объектно-ориентированного программирования, что обеспечило удобство масштабирования и поддержки продукта.

Практическая значимость проекта заключается как в создании удобного инструмента для управления повседневными задачами пользователя и мотивации через элементы геймификации (визуализация прогресса в виде «космического пространства»), так и в приобретении опыта проектирования, разработки и тестирования программных продуктов на C++ с использованием современных библиотек и технологий.

В дальнейшем приложение Nebula может быть расширено за счёт внедрения новых функций, таких как система достижений, персонализация интерфейса, а также больше визуальных эффектов. Полученные в ходе работы знания и навыки станут основой для дальнейшего профессионального роста и разработки более сложных программных систем.

Таким образом, поставленные цели и задачи курсового проекта были успешно достигнуты, а разработанный планировщик Nebula представляет собой

стабильное, интуитивно понятное и современное программное решение, способное служить основой для дальнейших исследований и развития в области цифровой продуктивности и саморазвития.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. todoist / [Электронный ресурс] // Todoist: Список дел для организации работы и жизни : [сайт]. — URL: https://www.todoist.com/ru/home (дата обращения: 16.03.2025).
2. Habitica: Gamify Your Life / [Электронный ресурс] // Habitica : [сайт]. — URL: https://habitica.com (дата обращения: 16.03.2025).
3. Forest / [Электронный ресурс] // Лайфхакер : [сайт]. — URL: https://lifehacker.ru/forest/ (дата обращения: 16.03.2025).

ПРИЛОЖЕНИЯ

Приложение А – Исходный код программы

Листинг А.1 – Файл CompletedTasksWindow.h

```
#ifndef COMPLETEDTASKSWINDOW_H
#define COMPLETEDTASKSWINDOW_H

#include <QWidget>
#include <QTreeWidget>
#include <memory>
#include "Task.h"

// Класс окна для отображения завершённых задач
class CompletedTasksWindow : public QWidget {
    Q_OBJECT

public:
    explicit CompletedTasksWindow(QWidget *parent = nullptr);
    void updateCompletedTasks(const QList<std::shared_ptr<Task>>&);

signals:
    void clearCompletedTasks();

private slots:
    void onClear();

private:
    QTreeWidget* tree = nullptr;
};

#endif
```

Листинг А.2 – Файл CompletedTasksWindow.cpp

```
#include "CompletedTasksWindow.h"
#include <QPushButton>
#include <QVBoxLayout>
#include <QHBoxLayout>
#include <QLabel>
#include <QMessageBox>

// Создание и настройка окна выполненных задач
CompletedTasksWindow::CompletedTasksWindow(QWidget *parent) : QWidget(parent) {
    setWindowTitle("Выполненые задачи");
    setMinimumSize(500, 350);

    auto v = new QVBoxLayout(this); // Вертикальная компоновка
    auto lbl = new QLabel("Выполненные задачи:");
    v->addWidget(lbl);

    tree = new QTreeWidget; // Дерево для задач
    tree->setHeaderLabels({"Задача", "Тип", "Дата"}); // Заголовки столбцов
    v->addWidget(tree); // Добавление дерева в компоновку

    auto h = new QHBoxLayout; // Горизонтальная компоновка для кнопок
    auto Clear = new QPushButton; // Кнопка для очистки списка
    Clear->setToolTip("Очистить список");
    Clear->setIcon(style()->standardIcon(QStyle::SP_TrashIcon));
    Clear->setText("");
    auto Close = new QPushButton; // Кнопка для закрытия окна
```

```
    Close->setToolTip("Закрыть окно");
    Close->setIcon(style()->standardIcon(QStyle::SP_DialogCloseButton));
    Close->setText("");

    h->addWidget(Clear);
    h->addStretch(); // Пространство между кнопками
    h->addWidget(Close);
    v->addLayout(h);

    // Соединение сигналов и слотов
    connect(Clear, &QPushButton::clicked, this, &CompletedTasksWindow::onClear); //
Очистка списка
    connect(Close, &QPushButton::clicked, this, &QWidget::close); // Закрытие окна

    // Оформление окна
    setStyleSheet(R"(
QWidget {
    background: qlineargradient(x1:0, y1:0, x2:0, y2:1,
        stop:0 #101929, stop:0.5 #191f34, stop:1 #070b14);
    color: #c9d1ea;
    font-family: Segoe UI, Arial, sans-serif;
    font-size: 13px;
}
QTreeWidget {
    background: rgba(16,16,32,0.92);
    border-radius: 7px;
    color: #dbeafe;
}
QHeaderView::section {
    background: #181e2f;
    color: #e7e7ef;
    border: none;
    padding: 5px;
    border-radius: 7px;
    font-size: 13px;
}
QLabel {
    color: #b6c1e1;
}
QPushButton {
    background: qlineargradient(x1:0, y1:0, x2:0, y2:1,
        stop:0 #2b3246, stop:1 #4d5673);
    border-radius: 18px;
    min-width: 40px;
    min-height: 40px;
    max-width: 44px;
    max-height: 44px;
    color: #e2e8f0;
    font-size: 18px;
    font-weight: bold;
    border: 1.5px solid #3a4360;
    margin: 6px;
    padding: 0 10px;
}
QPushButton:hover {
    background: qlineargradient(x1:0, y1:0, x2:0, y2:1,
        stop:0 #3c4661, stop:1 #5f6a8a);
    color: #b6c6e0;
    border: 1.5px solid #495382;
}
QPushButton:pressed {
    background: #232a3a;
    color: #fff;
```

```
    border: 1.5px solid #323b57;
}
)");

}

// Обновление списка задач в окне выполненных задач
void CompletedTasksWindow::updateCompletedTasks(const QList<std::shared_ptr<Task>>&
done) {
    tree->clear(); // Очищение дерева перед заполнением
    for (auto& t : done) {
        if (!t->isCompleted()) continue; // Невыполненные задачи пропускаются
        auto item = new QTreeWidgetItem; // Создание элемента дерева для задачи
        item->setText(0, t->getTitle()); // Название задачи
        item->setText(1, t->getType() == TaskType::GLOBAL ? "Общая" : "Обычная");
// Тип задачи
        item->setText(2,           t->getDate().isValid()           ?           t-
>getDate().toString("dd.MM.yyyy") : ""); // Дата задачи, если есть
        tree->addTopLevelItem(item); // Добавление задачи на верхний уровень дерева
    }
    tree->expandAll(); // Разворачивание элементов дерева
}

// Сообщение для пользователя
void CompletedTasksWindow::onClear() {
    if   (QMessageBox::question(this,   "Очистить?",   "С   чистого   неба?")   ==
QMessageBox::Yes) {
        emit clearCompletedTasks();
        tree->clear(); // Очищение дерева
    }
}
```

Листинг А.3 – Файл MainWindow.h

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include <QMainWindow>
#include <QTreeWidget>
#include <QTreeWidgetItem>
#include <QHeaderView>
#include <QJsonArray>
#include <QList>
#include <memory>
#include "Task.h"

class QCalendarWidget;
class QLineEdit;
class QComboBox;
class QPushButton;
class QLabel;
class CompletedTasksWindow;
class RewardsWindow;

// Главный класс окна приложения
class MainWindow : public QMainWindow {
    Q_OBJECT
public:
    MainWindow(QWidget *parent = nullptr);
    ~MainWindow();

protected:
    void closeEvent(QCloseEvent* e) override; // Метод, вызываем при закрытии окна
(сохраняет состояние)
```

```
    bool eventFilter(QObject *obj, QEvent *event) override;

private slots:
    // Слоты:
    void onDatePick(); // Выбор даты в календаре
    void onAdd(); // Добавить задачу
    void onDel(); // Удалить выбранную задачу
    void onCheck(QTreeWidgetItem*, int); // Отметить задачу как выполненную
    void onReward(); // Окно с визуализацией
    void onDoneList(); // Окно со списком завершённых задач
    void onToday(); // Выбрать сегодняшнюю дату
    void onClearDone(); // Очистить список завершённых задач

private:
    // Интерфейс
    QCalendarWidget* cal;
    QLineEdit* input;
    QComboBox* typeBox;
    QPushButton* Add;
    QPushButton* Del;
    QPushButton* Reward;
    QPushButton* Done;
    QPushButton* Today;

    QTreeWidget* tree;
    QLabel* dateLbl;

    QList<std::shared_ptr<Task>> tasks;
    QList<std::shared_ptr<Task>> done;
    QDate selDate;

    // Окна
    CompletedTasksWindow* winDone = nullptr;
    RewardsWindow* winReward = nullptr;

    // Внутренние методы
    void setupUi();
    void refresh();
    void showReward();
    void showDone();

    // Добавление, поиск и удаление задачи в дереве задач
    QTreeWidgetItem* addTaskItem(const std::shared_ptr<Task>&, QTreeWidgetItem* p =
nullptr);
    std::shared_ptr<Task> findTask(QTreeWidgetItem*, QList<std::shared_ptr<Task>>*
list = nullptr);
    void removeTask(QList<std::shared_ptr<Task>>&, std::shared_ptr<Task> t);

    void save();
    void load();

    void moveToDone(std::shared_ptr<Task>);
    int countDone() const;
    int countDoneGlobal() const;
};

#endif
```

Листинг А.4 – Файл MainWindow.cpp

```
#include "MainWindow.h"
#include <QCalendarWidget>
#include <QLineEdit>
#include <QComboBox>
```

```
#include <QPushButton>
#include <QTreeWidget>
#include <QTreeWidgetItem>
#include <QHeaderView>
#include <QLabel>
#include <QVBoxLayout>
#include <QHBoxLayout>
#include <QSplitter>
#include <QGroupBox>
#include <QFile>
#include <QJsonDocument>
#include <QMessageBox>
#include <QCloseEvent>
#include <QKeyEvent>
#include "CompletedTasksWindow.h"
#include "RewardsWindow.h"

#define FILE_TASKS "tasks.json" // Файл для хранения всех задач
#define FILE_DONE "completed.json" // Файл для хранения выполненных задач

// Главное окно
MainWindow::MainWindow(QWidget *parent) : QMainWindow(parent) {
    setupUi(); // Создание и настройка интерфейса
    load(); // Загрузка задачей из файлов
    refresh(); // Обновление отображения задач
}

MainWindow::~MainWindow() {}

// Сохранение задачей перед выходом
void MainWindow::closeEvent(QCloseEvent* e) {
    save();
    e->accept();
}

// Создание и настройка интерфейса
void MainWindow::setupUi() {
    auto main = new QWidget;
    setCentralWidget(main);

    auto split = new QSplitter(Qt::Horizontal, this); // Разделитель на левую и
правую часть

    auto left = new QGroupBox;
    auto vleft = new QVBoxLayout(left);

    cal = new QCalendarWidget;
    vleft->addWidget(cal);
    connect(cal, &QCalendarWidget::clicked, this, &MainWindow::onDatePick); // По
клике в календаре обновляется дата

    // Поле для ввода задачи
    input = new QLineEdit;
    input->setPlaceholderText("Ваше космическое дело");
    vleft->addWidget(input);

    // Enter для создания задачи
    input->installEventFilter(this);

    typeBox = new QComboBox; // Выпадающий список для типа задачи
    typeBox->addItem("Обычная", static_cast<int>(TaskType::REGULAR));
    typeBox->addItem("Общая", static_cast<int>(TaskType::GLOBAL));
    vleft->addWidget(typeBox);
```

```
    // Кнопки задач
    Add = new QPushButton;
    Del = new QPushButton;
    Reward = new QPushButton;
    Done = new QPushButton;
    Today = new QPushButton;

    // Иконки для кнопок
    Add->setIcon(style()->standardIcon(QStyle::SP_DirHomeIcon));
    Del->setIcon(style()->standardIcon(QStyle::SP_DialogCancelButton));
    Reward->setIcon(style()->standardIcon(QStyle::SP_DesktopIcon));
    Done->setIcon(style()->standardIcon(QStyle::SP_DialogApplyButton));
    Today->setIcon(style()->standardIcon(QStyle::SP_DialogDiscardButton));

    // Подсказки к кнопкам
    Add->setToolTip("Добавить задачу");
    Del->setToolTip("Удалить выбранную задачу");
    Reward->setToolTip("Посмотреть ваш космос");
    Done->setToolTip("Выполненные задачи");
    Today->setToolTip("Сегодня");

    // Иконки без текста
    Add->setText("");
    Del->setText("");
    Reward->setText("");
    Done->setText("");
    Today->setText("");

    // Стилизация для кнопок
    Add->setStyleSheet(R"(
QPushButton {
    background: qlineargradient(x1:0, y1:0, x2:1, y2:1,
        stop:0 #44d7b6, stop:1 #1fa2ff);
    border-radius: 20px;
    min-width: 40px; min-height: 40px;
    max-width: 44px; max-height: 44px;
    color: #e4eaff;
    font-size: 18px;
    font-weight: bold;
    border: 1.5px solid #27396b;
    margin: 6px;
}
QPushButton:hover {
    background: qlineargradient(x1:0, y1:0, x2:1, y2:1,
        stop:0 #1fa2ff, stop:1 #44d7b6);
    color: #8ecfff;
    border: 1.5px solid #365ab6;
}
QPushButton:pressed {
    background: #111a33;
    color: #fff;
    border: 1.5px solid #223264;
}
)");
    Del->setStyleSheet(R"(
QPushButton {
    background: qlineargradient(x1:0, y1:0, x2:1, y2:1,
        stop:0 #f857a6, stop:1 #ff5858);
    border-radius: 20px;
    min-width: 40px; min-height: 40px;
    max-width: 44px; max-height: 44px;
    color: #e4eaff;
```

```
    font-size: 18px;
    font-weight: bold;
    border: 1.5px solid #27396b;
    margin: 6px;
}
QPushButton:hover {
    background: qlineargradient(x1:0, y1:0, x2:1, y2:1,
        stop:0 #ff5858, stop:1 #f857a6);
    color: #8ecfff;
    border: 1.5px solid #365ab6;
}
QPushButton:pressed {
    background: #111a33;
    color: #fff;
    border: 1.5px solid #223264;
}
)");
    Reward->setStyleSheet(R"(
QPushButton {
    background: qlineargradient(x1:0, y1:0, x2:1, y2:1,
        stop:0 #00c6ff, stop:1 #0072ff);
    border-radius: 20px;
    min-width: 40px; min-height: 40px;
    max-width: 44px; max-height: 44px;
    color: #e4eaff;
    font-size: 18px;
    font-weight: bold;
    border: 1.5px solid #27396b;
    margin: 6px;
}
QPushButton:hover {
    background: qlineargradient(x1:0, y1:0, x2:1, y2:1,
        stop:0 #0072ff, stop:1 #00c6ff);
    color: #8ecfff;
    border: 1.5px solid #365ab6;
}
QPushButton:pressed {
    background: #111a33;
    color: #fff;
    border: 1.5px solid #223264;
}
)");
    Done->setStyleSheet(R"(
QPushButton {
    background: qlineargradient(x1:0, y1:0, x2:1, y2:1,
        stop:0 #43e97b, stop:1 #38f9d7);
    border-radius: 20px;
    min-width: 40px; min-height: 40px;
    max-width: 44px; max-height: 44px;
    color: #e4eaff;
    font-size: 18px;
    font-weight: bold;
    border: 1.5px solid #27396b;
    margin: 6px;
}
QPushButton:hover {
    background: qlineargradient(x1:0, y1:0, x2:1, y2:1,
        stop:0 #38f9d7, stop:1 #43e97b);
    color: #8ecfff;
    border: 1.5px solid #365ab6;
}
QPushButton:pressed {
    background: #111a33;
```

```
    color: #fff;
    border: 1.5px solid #223264;
}
)");
    Today->setStyleSheet(R"(
QPushButton {
    background: qlineargradient(x1:0, y1:0, x2:1, y2:1,
        stop:0 #a2a3a6, stop:1 #4d5673);
    border-radius: 20px;
    min-width: 40px; min-height: 40px;
    max-width: 44px; max-height: 44px;
    color: #e4eaff;
    font-size: 18px;
    font-weight: bold;
    border: 1.5px solid #3a4360;
    margin: 6px;
}
QPushButton:hover {
    background: qlineargradient(x1:0, y1:0, x2:1, y2:1,
        stop:0 #4d5673, stop:1 #a2a3a6);
    color: #b6c6e0;
    border: 1.5px solid #495382;
}
QPushButton:pressed {
    background: #232a3a;
    color: #fff;
    border: 1.5px solid #323b57;
}
)");

    auto Layout = new QHBoxLayout; // Горизонтальное расположение кнопок
    Layout->addWidget(Add);
    Layout->addWidget(Del);
    Layout->addWidget(Reward);
    Layout->addWidget(Done);
    Layout->addWidget(Today);
    Layout->setSpacing(12);
    vleft->addLayout(Layout);

    vleft->addStretch(); // Чтобы элементы были сверху

    // Соединение сигналов и слотов
    connect(Add, &QPushButton::clicked, this, &MainWindow::onAdd);
    connect(Del, &QPushButton::clicked, this, &MainWindow::onDel);
    connect(Reward, &QPushButton::clicked, this, &MainWindow::onReward);
    connect(Done, &QPushButton::clicked, this, &MainWindow::onDoneList);
    connect(Today, &QPushButton::clicked, this, &MainWindow::onToday);

    auto right = new QGroupBox;
    auto vright = new QVBoxLayout(right);

    dateLbl = new QLabel; // Выбранная дата
    vright->addWidget(dateLbl);

    tree = new QTreeWidget; // Дерево задач
    tree->setHeaderLabels(QStringList() << "Задача" << "Тип");
    tree->setAlternatingRowColors(true);
    tree->header()->setSectionResizeMode(0, QHeaderView::Interactive);
    tree->header()->setSectionResizeMode(1, QHeaderView::Interactive);
    tree->setColumnWidth(0, 300);
    tree->setColumnWidth(1, 70);
    vright->addWidget(tree);
```

```
    connect(tree, &QTreeWidget::itemChanged, this, &MainWindow::onCheck);

    split->addWidget(left); // Левая панель
    split->addWidget(right); // Правая панель
    split->setSizes({300, 500});

    auto lay = new QHBoxLayout(main);
    lay->addWidget(split);

    // Стилизация интерфейса
    setStyleSheet(R"(
QMainWindow, QWidget {
    background: qlineargradient(x1:0, y1:0, x2:0, y2:1,
        stop:0 #101929, stop:0.5 #191f34, stop:1 #070b14);
    color: #c9d1ea;
    font-family: Segoe UI, Arial, sans-serif;
    font-size: 13px;
}
QGroupBox {
    border-radius: 12px;
    background: rgba(20,20,35,0.60);
    font-size: 15px;
}
QLineEdit, QComboBox {
    background: #161b2c;
    border-radius: 7px;
    border: 1px solid #2a3150;
    color: #c9d1ea;
    padding: 6px;
}
QTreeWidget {
    background: rgba(16,16,32,0.92);
    border-radius: 7px;
    color: #dbeafe;
}
QLabel {
    color: #b6c1e1;
}
QMenuBar {
    background: #181e2f;
    color: #e7e7ef;
}
QMenuBar::item:selected {
    background: #232b45;
    color: #0ff1ce;
}
QMenuBar::item:pressed {
    background: #101929;
}
QMenu {
    background: #1b2135;
    color: #e7e7ef;
}
QMenu::item:selected {
    background: #28304e;
    color: #0ff1ce;
}
)");

    setWindowTitle("Nebula");
    resize(900, 600);
    selDate = QDate::currentDate();
}
```

```
// Нажатие на дату в календаре
void MainWindow::onDatePick() {
    selDate = cal->selectedDate();
    refresh();
}

// Кнопка "Сегодня"
void MainWindow::onToday() {
    selDate = QDate::currentDate();
    cal->setSelectedDate(selDate);
    refresh();
}

// Кнопка "Добавить задачу"
void MainWindow::onAdd() {
    auto text = input->text().trimmed();
    if (text.isEmpty()) {
        QMessageBox::warning(this,   "Упси",   "Введите   задачу");   //   Сообщение
пользователю, если пусто
        return;
    }
    auto   type   =   static_cast<TaskType>(typeBox->currentData().toInt());   //
Определение типа задачи
    auto date = type == TaskType::GLOBAL ? QDate() : selDate; // Для глобальной
задачи даты нет
    tasks << std::make_shared<Task>(text, date, type); // Добавление задачи в список
    input->clear(); // Очищение поля ввода
    refresh(); // Обновление интерфейса
}

// Создание задачи по Enter
bool MainWindow::eventFilter(QObject *obj, QEvent *event) {
    if (obj == input && event->type() == QEvent::KeyPress) {
        QKeyEvent *keyEvent = static_cast<QKeyEvent *>(event);
        if (keyEvent->key() == Qt::Key_Return || keyEvent->key() == Qt::Key_Enter)
{
            onAdd();
            return true;
        }
    }
    return QMainWindow::eventFilter(obj, event);
}

// Кнопка "Удалить задачу"
void MainWindow::onDel() {
    auto item = tree->currentItem(); // Получаем выделенный элемент дерева
    if (!item) return;
    auto t = findTask(item); // Находим связанную с ним задачу
    if (!t) return;
    if (QMessageBox::question(this, "Удалить?", "Вы уверены?") == QMessageBox::Yes)
{ // Сообщение для пользователя
        removeTask(tasks, t);
        refresh();
    }
}

// Выполнение задачи по чекбоксу
void MainWindow::onCheck(QTreeWidgetItem* item, int col) {
    if (col != 0) return;
    auto t = findTask(item);
    if (!t) return;
    if (item->checkState(0) == Qt::Checked) {
```

```
        t->setCompleted(true); // Отметка, что задача выполнена
        moveToDone(t); // Задача перемещается в выполненные
        if (t->getType() == TaskType::GLOBAL)
            removeTask(tasks, t);
    }
    refresh();
}

// Перемещение задачи в список выполненных
void MainWindow::moveToDone(std::shared_ptr<Task> t) {
    if (!done.contains(t)) done << t;
    removeTask(tasks, t);
}

// Кнопка "Космсо"
void MainWindow::onReward() { showReward(); }
// Кнопка "Выполненные задачи"
void MainWindow::onDoneList() { showDone(); }

// Открытие окна "Космос"
void MainWindow::showReward() {
    if (!winReward) winReward = new RewardsWindow;
    winReward->showCompletedTasks(done);
    auto item = tree->currentItem();
    auto t = item ? findTask(item) : nullptr;
    winReward->showTaskVisualization(t);
    winReward->show();
    winReward->raise();
}


// Открытие окна завершённых задач
void MainWindow::showDone() {
    if (!winDone) {
        winDone = new CompletedTasksWindow;
        connect(winDone,     &CompletedTasksWindow::clearCompletedTasks,     this,
&MainWindow::onClearDone);
    }
    winDone->updateCompletedTasks(done);
    winDone->show();
    winDone->raise();
}

// Очистка выполненных задач
void MainWindow::onClearDone() {
    done.clear();
    if (winDone) winDone->updateCompletedTasks(done);
    showReward();
}

// Обновление дерева задач в интерфейсе
void MainWindow::refresh() {
    tree->clear();
    dateLbl->setText("На " + selDate.toString("dd.MM.yyyy"));
    for (auto t : tasks) {
        bool show = false;
        if (t->getType() == TaskType::GLOBAL)
            show = true;
        else if (t->getDate() == selDate)
            show = true;
        if (show)
            addTaskItem(t);
    }
```

```
    tree->expandAll();
}

// Добавление задачи в дерево задач
QTreeWidgetItem*    MainWindow::addTaskItem(const    std::shared_ptr<Task>&    t,
QTreeWidgetItem* p) {
    Q_UNUSED(p);
    auto item = new QTreeWidgetItem;
    item->setText(0, t->getTitle());
    item->setText(1, t->getType() == TaskType::GLOBAL ? "Общая" : "Обычная");
    item->setCheckState(0, t->isCompleted() ? Qt::Checked : Qt::Unchecked);
    item->setFlags(item->flags() | Qt::ItemIsUserCheckable | Qt::ItemIsSelectable |
Qt::ItemIsEnabled);
    tree->addTopLevelItem(item);
    return item;
}

// Поиск задачи
std::shared_ptr<Task>          MainWindow::findTask(QTreeWidgetItem*          item,
QList<std::shared_ptr<Task>>* list) {
    if (!item) return nullptr;
    if (!list) list = &tasks;
    for (auto& t : *list) {
        if (t->getTitle() == item->text(0)) return t;
    }
    return nullptr;
}

// Удаление задачи из списка
void           MainWindow::removeTask(QList<std::shared_ptr<Task>>&           list,
std::shared_ptr<Task> t) {
    for (int i = 0; i < list.size(); ++i) {
        if (list[i] == t) { list.removeAt(i); return; }
    }
}

// Сохранение всех задач в файл Json
void MainWindow::save() {
    QJsonArray arr, arrDone;
    for (auto t : tasks) arr.append(t->toJson());
    for (auto t : done) arrDone.append(t->toJson());

    QFile f(FILE_TASKS);
    if (f.open(QIODevice::WriteOnly))
        f.write(QJsonDocument(arr).toJson());

    QFile f2(FILE_DONE);
    if (f2.open(QIODevice::WriteOnly))
        f2.write(QJsonDocument(arrDone).toJson());
}

// Загрузка задач из файла
void MainWindow::load() {
    QFile f(FILE_TASKS);
    if (f.open(QIODevice::ReadOnly)) {
        auto doc = QJsonDocument::fromJson(f.readAll());
        tasks.clear();
        for (auto val : doc.array())
            tasks << Task::fromJson(val.toObject());
    }
    QFile f2(FILE_DONE);
    if (f2.open(QIODevice::ReadOnly)) {
        auto doc = QJsonDocument::fromJson(f2.readAll());
```

```
        done.clear();
        for (auto val : doc.array())
            done << Task::fromJson(val.toObject());
    }
}

// Подсчёт количества выполненных обычных задач
int MainWindow::countDone() const {
    int c = 0;
    for (auto t : done)
        if (t->getType() == TaskType::REGULAR && t->isCompleted())
            ++c;
    return c;
}

// Подсчёт количества выполненных глобальных задач
int MainWindow::countDoneGlobal() const {
    int c = 0;
    for (auto t : done)
        if (t->getType() == TaskType::GLOBAL && t->isCompleted())
            ++c;
    return c;
}
```

Листинг А.5 – Файл RewardsWindow.h

```
#ifndef REWARDSWINDOW_H
#define REWARDSWINDOW_H

#include <QWidget>
#include <memory>
#include <QList>
#include <QPointF>
#include <QColor>
#include <QRandomGenerator>
#include "Task.h"

// Структура, отвечающая за визуализацию задачи в "космосе"
struct VisualTask {
    std::shared_ptr<Task> task;
    QPointF pos;
    QColor color;
    int radius = 0;
    bool hasRing = false;
};

// Класс окна для визуализации задач ("космос")
class RewardsWindow : public QWidget {
    Q_OBJECT
public:
    explicit RewardsWindow(QWidget *parent = nullptr);
    void showCompletedTasks(const QList<std::shared_ptr<Task>>& done);
    void showTaskVisualization(const std::shared_ptr<Task>& task);

protected:
    // Перерисовка окна
    void paintEvent(QPaintEvent*) override;

private:
    QPointF findFreePosition(double radius, int maxAttempts = 100); // Поиск
свободной точки
    bool isOverlapping(const QPointF& pt, double radius); // Проверка на пересечение
с другими объектами
```

```
    QList<VisualTask> visualTasks; // Визуализация
    QList<QPointF> usedPositions; // Занятые позиции
    QSizeF fieldSize;
};

#endif
```

Листинг А.6 – Файл RewardsWindow.cpp

```
#include "RewardsWindow.h"
#include <QPainter>
#include <QPaintEvent>
#include <QtMath>
#include <QRandomGenerator>

namespace {
// Цвета для планет
QList<QColor> Planets() {
    return {
        QColor(255, 223, 186), QColor(202, 231, 255), QColor(187, 222, 214),
        QColor(255, 236, 179), QColor(220, 198, 224), QColor(200, 210, 255),
        QColor(178, 235, 242), QColor(255, 213, 230)
    };
}
// Цвета для звёзд
QList<QColor> Stars() {
    return {
        QColor(255, 255, 190), QColor(220, 235, 255), QColor(245, 245, 255),
        QColor(200, 220, 255), QColor(255, 252, 210), QColor(220, 228, 255)
    };
}
}

// Окно "Космсо"
RewardsWindow::RewardsWindow(QWidget *parent) : QWidget(parent) {
    setWindowTitle("Ваш космос");
    setFixedSize(700, 480);
    setStyleSheet("background: transparent;");
}

// Отображение завершённых задач как "планеты" и "звёзды"
void RewardsWindow::showCompletedTasks(const QList<std::shared_ptr<Task>>& done) {
    visualTasks.clear(); // Очистка старых объектов
    usedPositions.clear(); // Очистка старых занятых позиций
    fieldSize = QSizeF(width()-80, height()-80);

    int planetCount = 0, starCount = 0;
    QList<QColor> planets = Planets();
    QList<QColor> stars = Stars();

    QRandomGenerator* rng = QRandomGenerator::global(); // Генератор случайных чисел

    for (const auto& t : done) {
        if (!t->isCompleted()) continue; // Только завершённые задачи

        // Для обычных задач
        if (t->getType() != TaskType::REGULAR && t->getType() != TaskType::GLOBAL)
            continue;

        VisualTask vt;
        vt.task = t;

        if (t->getType() == TaskType::GLOBAL) {
            // Размер планеты случайный
```

```
            vt.radius = 22 + rng->bounded(21);
            vt.hasRing = rng->bounded(100) < 60;
            vt.color = planets.at(planetCount % planets.size());
            ++planetCount;
        } else {
            vt.radius = 6;
            vt.hasRing = false;
            vt.color = stars.at(starCount % stars.size());
            ++starCount;
        }

        QPointF pos = findFreePosition(vt.radius); // Поиск свободной позиции
        vt.pos = pos;

        visualTasks.append(vt);
        usedPositions.append(pos);
    }

    update();
    show();
    raise();
}

// Поиск свободной позиции для объекта
QPointF RewardsWindow::findFreePosition(double radius, int maxAttempts) {
    QRectF field(40, 40, fieldSize.width(), fieldSize.height());
    for (int attempt = 0; attempt < maxAttempts; ++attempt) {
        double x = field.left() + QRandomGenerator::global()-
>bounded(field.width());
        double y = field.top() + QRandomGenerator::global()-
>bounded(field.height());
        QPointF pt(x, y);
        if (!isOverlapping(pt, radius)) return pt; // Если не пересекается, то
вернуть
    }
    return QPointF(field.center());
}

// Проверка на пересечение
bool RewardsWindow::isOverlapping(const QPointF& pt, double radius) {
    for (const auto& other : usedPositions) {
        if (QLineF(pt, other).length() < radius*2.2)
            return true;
    }
    return false;
}

// Рисует элементы "космоса"
void RewardsWindow::paintEvent(QPaintEvent*) {
    QPainter p(this);
    p.setRenderHint(QPainter::Antialiasing);

    // Фон
    QLinearGradient grad(0, 0, width(), height());
    grad.setColorAt(0.0, QColor(18, 21, 50));
    grad.setColorAt(0.4, QColor(48, 30, 80));
    grad.setColorAt(1.0, QColor(10, 8, 30));
    p.fillRect(rect(), grad);

    // Туманности
    QRadialGradient nebula1(QPointF(width()*0.2, height()*0.7), 110);
    nebula1.setColorAt(0.0, QColor(120, 80, 255, 80));
    nebula1.setColorAt(1.0, Qt::transparent);
```

```
    p.setBrush(nebula1); p.setPen(Qt::NoPen);
    p.drawEllipse(QPointF(width()*0.2, height()*0.7), 110, 80);

    QRadialGradient nebula2(QPointF(width()*0.8, height()*0.3), 90);
    nebula2.setColorAt(0.0, QColor(255, 70, 180, 60));
    nebula2.setColorAt(1.0, Qt::transparent);
    p.setBrush(nebula2);
    p.drawEllipse(QPointF(width()*0.8, height()*0.3), 90, 55);

    // Звезды на фоне
    QRandomGenerator* localRng = QRandomGenerator::global();
    for (int i = 0; i < 72; ++i) {
        double x = localRng->bounded(width());
        double y = localRng->bounded(height());
        double rad = 1.2 + localRng->bounded(1.5);
        QColor c = QColor(220+localRng->bounded(35), 220+localRng->bounded(35),
255, 120+localRng->bounded(100));
        p.setPen(Qt::NoPen);
        p.setBrush(c);
        p.drawEllipse(QPointF(x, y), rad, rad);
    }

    for (const auto& vt : visualTasks) {
        if (!vt.task) continue;
        const auto& t = vt.task;
        QPointF pt = vt.pos;

        if (t->getType() == TaskType::GLOBAL) {
            int planetR = vt.radius;
            QColor c = vt.color;

            // Ореол
            QRadialGradient aura(pt, planetR * 1.6);
            aura.setColorAt(0, c.lighter(170));
            aura.setColorAt(0.7, QColor(0, 0, 0, 0));
            p.setBrush(aura); p.setPen(Qt::NoPen);
            p.drawEllipse(pt, planetR * 1.6, planetR * 1.6);

            // Тень
            QRadialGradient shadow(pt + QPointF(planetR * 0.7, planetR * 0.5),
planetR * 1.1);
            shadow.setColorAt(0, QColor(0, 0, 0, 60));
            shadow.setColorAt(1, Qt::transparent);
            p.setBrush(shadow); p.setPen(Qt::NoPen);
            p.drawEllipse(pt, planetR * 1.1, planetR * 1.1);

            // Планета
            QRadialGradient grad(pt - QPointF(planetR * 0.3, planetR * 0.2),
planetR);
            grad.setColorAt(0.0, c.lighter(130));
            grad.setColorAt(0.6, c);
            grad.setColorAt(1.0, c.darker(120));
            p.setBrush(grad);
            p.setPen(QPen(c.darker(150), 2));
            p.drawEllipse(pt, planetR, planetR);

            // Кольцо — только если оно есть у этой планеты
            if (vt.hasRing) {
                p.setBrush(Qt::NoBrush);
                p.setPen(QPen(c.lighter(170), 2.5));
                QRectF ring(pt.x() - planetR*1.17, pt.y() - planetR*0.18,
planetR*2.34, planetR*0.39 + planetR*0.2 * (localRng->bounded(100)/100.0));
                p.drawEllipse(ring);
```

```
            }
        }
        else {
            int starR = vt.radius;
            QColor c = vt.color;

            // Свечение
            QRadialGradient glow(pt, starR*2.2);
            glow.setColorAt(0, c.lighter(150));
            glow.setColorAt(1, Qt::transparent);
            p.setBrush(glow); p.setPen(Qt::NoPen);
            p.drawEllipse(pt, starR*2.2, starR*2.2);

            // Звезда
            QPolygonF star;
            for (int j = 0; j < 10; ++j) {
                double a = j * M_PI / 5.0;
                double r = (j % 2 == 0) ? starR : starR * 0.47;
                star << QPointF(pt.x() + r * qCos(a - M_PI/2), pt.y() + r * qSin(a
- M_PI/2));
            }
            p.setBrush(c);
            p.setPen(QPen(c.darker(140), 2));
            p.drawPolygon(star);
        }
    }

    // Заголовок
    p.setPen(QColor(210, 210, 230));
    QFont f = font();
    f.setBold(true); f.setPointSize(15);
    p.setFont(f);
    QString txt = QString("Выполнено задач: %1").arg(visualTasks.size());
    p.drawText(rect(), Qt::AlignTop | Qt::AlignHCenter, txt);
}

void RewardsWindow::showTaskVisualization(const std::shared_ptr<Task>& task) {
    update();
    show();
    raise();
}
```

Листинг А.7 – Файл Task.h

```
#ifndef TASK_H
#define TASK_H

#include <QString>
#include <QDate>
#include <QList>
#include <QJsonObject>
#include <memory>

enum class TaskType { REGULAR, GLOBAL };

// Класс задачи
class Task {
public:
    Task(const QString& t, const QDate& d = QDate(), TaskType tp =
TaskType::REGULAR);

    void setCompleted(bool); // Отметить задачу выполненной
    bool isCompleted() const; //Проверка, выполнена задача
```

```
    QString getTitle() const { return title_; } // Получить название
    void setTitle(const QString& t) { title_ = t; } // Задать название
    QDate getDate() const { return date_; } // Получить дату
    void setDate(const QDate& d) { date_ = d; } // Задать дату
    TaskType getType() const { return type_; } // Получить тип задачи
    void setType(TaskType t) { type_ = t; } // Задать тип задачи

    QJsonObject toJson() const;
    static std::shared_ptr<Task> fromJson(const QJsonObject&);

private:
    QString title_;
    QDate date_;
    TaskType type_;
    bool completed_ = false;
};

#endif
```

Листинг А.8 – Файл Task.cpp

```
#include "Task.h"
#include <QJsonArray>

// Конструктор задачи (название, дата и тип)
Task::Task(const QString& t, const QDate& d, TaskType tp)
    : title_(t), date_(d), type_(tp) {}

void Task::setCompleted(bool c) { completed_ = c; }
bool Task::isCompleted() const { return completed_ ; }

// Сохранение задачи в Json
QJsonObject Task::toJson() const {
    QJsonObject j;
    j["title"] = title_;
    j["date"] = date_.isValid() ? date_.toString(Qt::ISODate) : QString();
    j["type"] = static_cast<int>(type_);
    j["completed"] = completed_;
    QJsonArray subs;
    j["subtasks"] = subs;
    return j;
}

// Восстановить задачу из Json
std::shared_ptr<Task> Task::fromJson(const QJsonObject& j) {
    QString title = j["title"].toString();
    QDate date;
    if             (!j["date"].toString().isEmpty())             date             =
QDate::fromString(j["date"].toString(), Qt::ISODate);
    auto type = static_cast<TaskType>(j["type"].toInt());
    auto t = std::make_shared<Task>(title, date, type);
    t->setCompleted(j["completed"].toBool());
    return t;
}
```