**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ  
РОССИЙСКОЙ ФЕДЕРАЦИИ**

БРЯНСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

Кафедра «Информатика и программное обеспечение»

**КУРСОВОЙ ПРОЕКТ**

**Игровая программа “Тамагочи”**

**Всего листов 26**

Преподаватель:Д.И. Булатицкий

«\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2025 г.

Студент гр. О-24-ИВТ-1-ПО-Б

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ И.А. Школин

«\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_ 2025 г.

**БРЯНСК 2025**

[Введение 3](#_Toc1)

[1. Анализ предметной области 4](#_Toc2)

[1.1. Историческая справка 4](#_Toc3)

[1.1.1. Зарождение концепции виртуальных питомцев 4](#_Toc4)

[1.1.2. Популяризация Tamagotchi 4](#_Toc5)

[1.1.3. Эволюция и влияние на современные технологии 4](#_Toc6)

[1.2. Основные термины и определения 5](#_Toc7)

[1.3. Типовые подходы к решению 6](#_Toc8)

[1.3.1. Модульная архитектура и разделение функциональности 7](#_Toc9)

[1.3.2. Обработка событий и управление состояниями 7](#_Toc10)

[1.3.3. Использование сторонних библиотек и готовых решений 8](#_Toc11)

[2. Конструкторская часть 9](#_Toc12)

[2.1. Общая структура проекта 9](#_Toc13)

[2.1.1. Корневые файлы 9](#_Toc14)

[2.1.2. Папка assets 9](#_Toc15)

[2.1.3. Папка include 9](#_Toc16)

[2.1.4. Папка src 10](#_Toc17)

[2.2. Обобщённый алгоритм программы 11](#_Toc18)

[2.3. Технические решения 14](#_Toc19)

[2.3.1. Менеджер сцен 14](#_Toc20)

[2.3.2. Изображение питомца 15](#_Toc21)

[2.3.3. Анимации 17](#_Toc22)

[2.3.4. Шкалы параметров 18](#_Toc23)

[2.3.5. Меню персонализации (кастомиазция питомца смена его скинов) 19](#_Toc24)

[3. Тестирование 21](#_Toc25)

[4. Итоги проделанной работы 25](#_Toc26)

[Список литературы 26](#_Toc27)

# Введение

**Актуальность разработки игр-симуляторов.** На сегодняшний день игры-симуляторы, имитирующие уход за виртуальным питомцем, остаются актуальными как в образовательном, так и в развлекательном плане. Такие игры позволяют не только весело провести время, но и развивать навыки планирования, ответственности и эмпатии, особенно среди детей и подростков. Разработка подобных приложений на языке Си с использованием SDL2 позволяет получить высокопроизводительный и кроссплатформенный продукт, что важно для охвата широкой аудитории и дальнейшей интеграции с другими системами.

**Цели и задачи проекта "Тамагочи".** Основной целью данного проекта является создание интерактивного приложения, моделирующего поведение виртуального питомца, с богатым функционалом по уходу, взаимодействию и развитием питомца. Для достижения этой цели необходимо решить следующие задачи:

* Разработать модульную архитектуру проекта, включающую обработку ввода, графику, анимацию и пользовательский интерфейс.
* Реализовать систему состояний питомца (здоровье, настроение, голод) и их динамическое и статическое обновление.
* Создать удобный и интуитивно понятный интерфейс с использованием элементов управления (кнопок, шкал, уведомлений).
* Обеспечить кроссплатформенность и оптимизацию производительности приложения на языке Си с использованием SDL2.

Таким образом, данный проект направлен на объединение образовательных и развлекательных аспектов, позволяя пользователю учиться заботиться о виртуальном питомце и одновременно наслаждаться интерактивным игровым процессом.

# Анализ предметной области

## Историческая справка

### Зарождение концепции виртуальных питомцев

В начале 1990-х годов на фоне бурного развития цифровых технологий возникла идея создания интерактивных игрушек, способных имитировать живых питомцев[1]. Первые эксперименты в этой области были связаны с простыми электронными устройствами, которые реагировали на внешние воздействия и требовали минимального ухода. Это положило начало созданию концепции виртуального питомца, способного развиваться, нуждаться в заботе и взаимодействовать с пользователем.

### Популяризация Tamagotchi

В 1996 году японская компания Bandai выпустила устройство Tamagotchi – карманного цифрового питомца, которое мгновенно завоевало популярность во всём мире. Tamagotchi предлагал пользователю заботиться о виртуальном существе: кормить, играть, чистить и лечить его. Такая модель взаимодействия оказалась не только развлекательной, но и обучающей, поскольку способствовала развитию ответственности у владельцев. Успех Tamagotchi оказал значительное влияние на массовую культуру, вдохновив создание многочисленных аналогов и развитие целого жанра игр-симуляторов.

### Эволюция и влияние на современные технологии

Со временем концепция виртуальных питомцев претерпела значительные изменения. Первоначально простые электронные игрушки уступили место компьютерным играм с более сложной логикой поведения, расширенными возможностями кастомизации и интеграцией мультимедийных элементов. Современные приложения в жанре «Тамагочи» включают не только уход за питомцем, но и развитие его характеристик, реалистичную анимацию и возможность взаимодействия через различные сенсорные интерфейсы. Эти игры стали популярными не только как развлечение, но и как образовательные платформы, стимулирующие интерес к цифровым технологиям и программированию.

## Основные термины и определения

В рамках проекта «Игровая программа Тамагочи» используются ключевые понятия, определяющие структуру и логику работы приложения. Описания приведены с учётом делового стиля и направлены на формирование единого понятийного поля для всех участников процесса разработки.

Центральным элементом приложения является **виртуальный питомец** — цифровой объект, поведение которого имитирует действия живого существа. Он обладает динамически изменяемыми параметрами, такими как **здоровье**, **голод** и **настроение**. Эти характеристики реагируют на действия пользователя, формируя процесс взаимодействия и развития питомца в рамках игрового сценария. Цель — предоставить пользователю возможность заботиться о питомце, наблюдать его прогресс и выстраивать с ним условную эмоциональную связь.

Управление структурой приложения организовано через систему сцен, каждая из которых представляет собой отдельный экран — будь то главное меню, игровой процесс, настройки или меню кастомизации. За переключение между сценами отвечает специальный компонент, условно называемый **менеджером сцен**. Он контролирует загрузку и выгрузку ресурсов в зависимости от действий пользователя, обеспечивая стабильную и эффективную навигацию по приложению.

Неотъемлемой частью взаимодействия является **пользовательский интерфейс**. Он включает кнопки, шкалы, меню и прочие визуальные компоненты, с помощью которых осуществляется навигация, ввод команд и получение обратной связи. Особое внимание при разработке интерфейса уделяется удобству и эргономике, особенно с учётом использования приложения на мобильных устройствах.

Для визуализации поведения и отклика приложения используется **анимация**. Это механизм последовательного отображения кадров или спрайтов, создающий иллюзию движения. Анимация применяется как для оживления питомца, так и для улучшения пользовательского опыта при работе с элементами интерфейса, например, при нажатии кнопок.

Дополнительно предусмотрена возможность **кастомизации** питомца — пользователь может изменять его внешний вид, выбирая скины. Этот функционал расширяет возможности персонализации, повышает вовлечённость и позволяет пользователю адаптировать образ питомца под собственные предпочтения.

## Типовые подходы к решению

В данном разделе рассматриваются типовые методологии и проектные решения, применяемые при разработке игровых приложений на языке C с использованием SDL2. Эти подходы направлены на достижение гибкости, масштабируемости и простоты поддержки проекта, а также на эффективное решение типовых задач игровой разработки.

### Модульная архитектура и разделение функциональности

Одним из ключевых подходов является разделение проекта на независимые модули, каждый из которых отвечает за свою часть функциональности. Это позволяет:

* **Инкапсулировать логику**: отделить обработку анимаций, графики, пользовательского интерфейса и логики игровых сцен.
* **Упростить поддержку и расширение**: при необходимости добавить новый функционал (например, экран настроек, паузы или дополнительное поведение питомца), разработчик может внести изменения в соответствующий модуль, не затрагивая другие компоненты.
* **Повысить читаемость кода**: четкое разделение задач способствует созданию структурированного и понятного кода.

Типовой пример – реализация менеджера сцен, где каждое состояние игры (меню, игровой процесс, настройки) оформлено как отдельный модуль с собственными функциями и структурой, а переключение между ними осуществляется через центральный менеджер.

### Обработка событий и управление состояниями

В играх на SDL2 широко применяется событийно-ориентированная модель:

* **Обработка пользовательского ввода**: использование SDL\_Event для отслеживания клавиатуры, мыши и сенсорного ввода позволяет реализовать динамическое взаимодействие с игроком.
* **Состояния объектов и сцен**: для управления поведением игровых объектов (например, питомца) и экранов используется механизм конечных автоматов (state machine), что обеспечивает корректное переключение между различными режимами работы приложения.

Такой подход позволяет реализовать плавные переходы, реагировать на события в реальном времени и организовать гибкое управление игровым процессом.

### Использование сторонних библиотек и готовых решений

Для решения типовых задач разработчики часто прибегают к использованию готовых библиотек:

* **SDL2 и его расширения (SDL2\_image, SDL2\_ttf, SDL2\_gfx):** эти библиотеки значительно упрощают работу с графикой, звуком, текстом и примитивами, позволяя не реализовывать базовые функции с нуля.
* **Анимационные модули:** для создания анимаций объектов (например, питомца или интерфейсных элементов) используется организация кадра из спрайт-листов и управление временем переключения кадров, что является стандартным решением в 2D-играх.

Использование проверенных библиотек позволяет сократить время разработки, повысить надежность кода и сосредоточиться на уникальных аспектах проекта.

При разработке в команде ключевым подходом является использование систем контроля версий, таких как Git[2]:

* **Параллельная разработка:** каждый участник команды может работать над своим модулем, не создавая конфликтов, благодаря ветвлению и последующему слиянию изменений.
* **Отслеживание изменений:** история коммитов позволяет проследить эволюцию проекта, быстро обнаруживать и исправлять ошибки.
* **Интеграция с сервисами:** использование платформ вроде GitHub или GitLab облегчает проведение code review и обмен информацией между разработчиками.

Такой подход значительно упрощает управление проектом и повышает качество кода при коллективной разработке.

# Конструкторская часть

## Общая структура проекта

### Корневые файлы

**README.md** — инструкции по сборке проекта под Windows и Linux.

### Папка assets

Содержит ресурсы игры: изображения, анимации, шрифты и звуки.

**animations/** — спрайт‑листы для анимации элементов интерфейса и кнопок

**fonts/** — шрифты в формате TTF (используются для рендеринга текста).

**sounds/** — звуковые эффекты.

Также просто в папке **assets/** — набор PNG‑файлов с кнопками, фоном и иконками питомца.

**txt/info.txt** — текст об авторстве (отображается в титульном экране).

### Папка include

Заголовочные файлы объявляют интерфейсы модулей:

**graphics.h** — описание функций инициализации SDL, загрузки/рендеринга текстур, вспомогательные методы для рисования (скруглённые прямоугольники).

**scene\_manager.h** — управление текущей «сценой» (экраном): инициализация, обработка событий, обновление, отрисовка и уничтожение.

**title\_scene.h, menu\_scene.h, game\_scene.h, menu\_pet.h, dead\_scene.h**,  **menu\_pet.h, saves\_scene.h** — структуры Scene и прототипы функций для соответствующих экранов приложения.

**animation.h** — структура анимаций а также описания функций для взаимодействия с ними.

**ui.h** — абстракция для кнопок: создание, отрисовка, анимация и обработка событий.

**pet.h** — структура Pet с полями состояния (здоровье, голод, настроение, путь к изображению) и описанием функциий инициализации, обновления, сохранения.

**globals.h** — глобальные переменные (WINDOW\_WIDTH, WINDOW\_HEIGHT, флаг звука IS\_SOUND).

**notify.h** — кросс‑платформенная обёртка для всплывающих уведомлений.

**text\_input.h** — обертка для работы с вводом.

**file\_manager.h** — обертка для кроссплатформенной работы с файлами.

### Папка src

Реализация логики модулей.

**main.c** — точка входа: инициализация SDL2 (графика, шрифты, аудио), загрузка питомца, игровой цикл (обработка событий, вычисление времени, обновление питомца, текущей сцены, отрисовка, вывод FPS), финальное сохранение и очистка ресурсов.

**graphics.c —** реализация функций рисования, вспомогательных фигур, разбиения текстур на части и очистки SDL.

**scene\_manager.c —** хранит указатель на текущую сцену и вызывает её методы.

**animation.c** — представляет удобное взаимодействия с анимация SDL2 в совокупности с animation.h.

**title\_scene.c —** отображает титульный экран: рендерит текст из info.txt с автоматическим переносом строк, кнопку «Start», по клику переходит в меню.

**menu\_scene.c —** главное меню: кнопки «Start», «PowerOff», «Help», «Sound On/Off». Переключение в игровые сцены или выход из программы.

**saves\_scene.c** — сцена вфбора сохранения.

**game\_scene.c —** основная игровая сцена: фон, кнопки «гладить», «кормить», «кастомизация», отрисовка прогресс‑бара здоровье/голод/настроение, музыки, обновление кнопок и питомца.

**menu\_pet.c —** экран выборки скинов: листаем доступные текстуры питомца, просматриваем «предпросмотр», применяем выбранный скин или возвращаемся в игру.

**pet.c —** управление состоянием питомца: загрузка/сохранение, расчёт деградации показателей по времени, функции (переход в сцену смерти при нуле здоровья) .

**notify.c —** платформо‑зависимые уведомления.

**text\_input.c —** кросплатформенные использование ввода с клавиатуры.

**file\_manager.с** — кроссплатформенные функции для работы с файлами.

## Обобщённый алгоритм программы

Программа представляет собой классическую реализацию аркадной тамагочи-игры, построенной на SDL2 с модульной архитектурой и чёткой организацией логики взаимодействия пользователя и цифрового питомца. После запуска приложения инициализируются все ключевые подсистемы: графика, аудио, шрифты, а также загружаются необходимые ресурсы (спрайты, звуки, тексты). Одновременно из файла восстановления загружается текущее состояние питомца — его здоровье, голод, настроение и другие параметры, что позволяет продолжить игру с последнего момента.

Пользователю первоначально предоставляется главное меню, содержащее краткую информацию об игре и базовые навигационные кнопки — «Start», «Help» и «Exit». При выборе старта происходит переход в стартовую игровую сцену, где перед началом проверяются временные параметры, и происходит перерасчёт состояния питомца с учётом времени, прошедшего с момента последнего запуска. Если здоровье питомца оказывается равным нулю, игроку демонстрируется сцена смерти с возможностью начать заново, что приводит к сбросу всех сохранений. В случае, если питомец жив, запускается основной игровой цикл.

Игровая сцена — это центральная часть программы, где пользователь взаимодействует с питомцем через наглядный интерфейс. Здесь отображается сам питомец, фон, и шкалы параметров (например, здоровье, настроение), а также кнопки управления — «Кормить», «Гладить», «Кастомизация» и «Выход». Каждое действие пользователя (например, кормление) напрямую влияет на параметры питомца, что реализуется через систему пересчёта значений в зависимости от действия и прошедшего времени. Меню кастомизации предоставляет возможность изменить внешний вид питомца путём выбора одного из доступных скинов. Все изменения применяются немедленно, возвращая игрока обратно в игровой процесс.

Цикл работы сцены построен по типовой схеме: обработка событий, вычисление логики и обновлений, рендеринг всех элементов сцены и вывод кадра на экран. Такой подход обеспечивает плавность, отзывчивость и визуальную целостность происходящего. При выходе пользователя из игрового режима или при закрытии приложения вызывается процедура сохранения текущего состояния питомца, освобождаются ресурсы, и игра завершает свою работу корректно и безопасно.Алгоритм в виде блок-схемы можно рассмотреть на Рис. 1

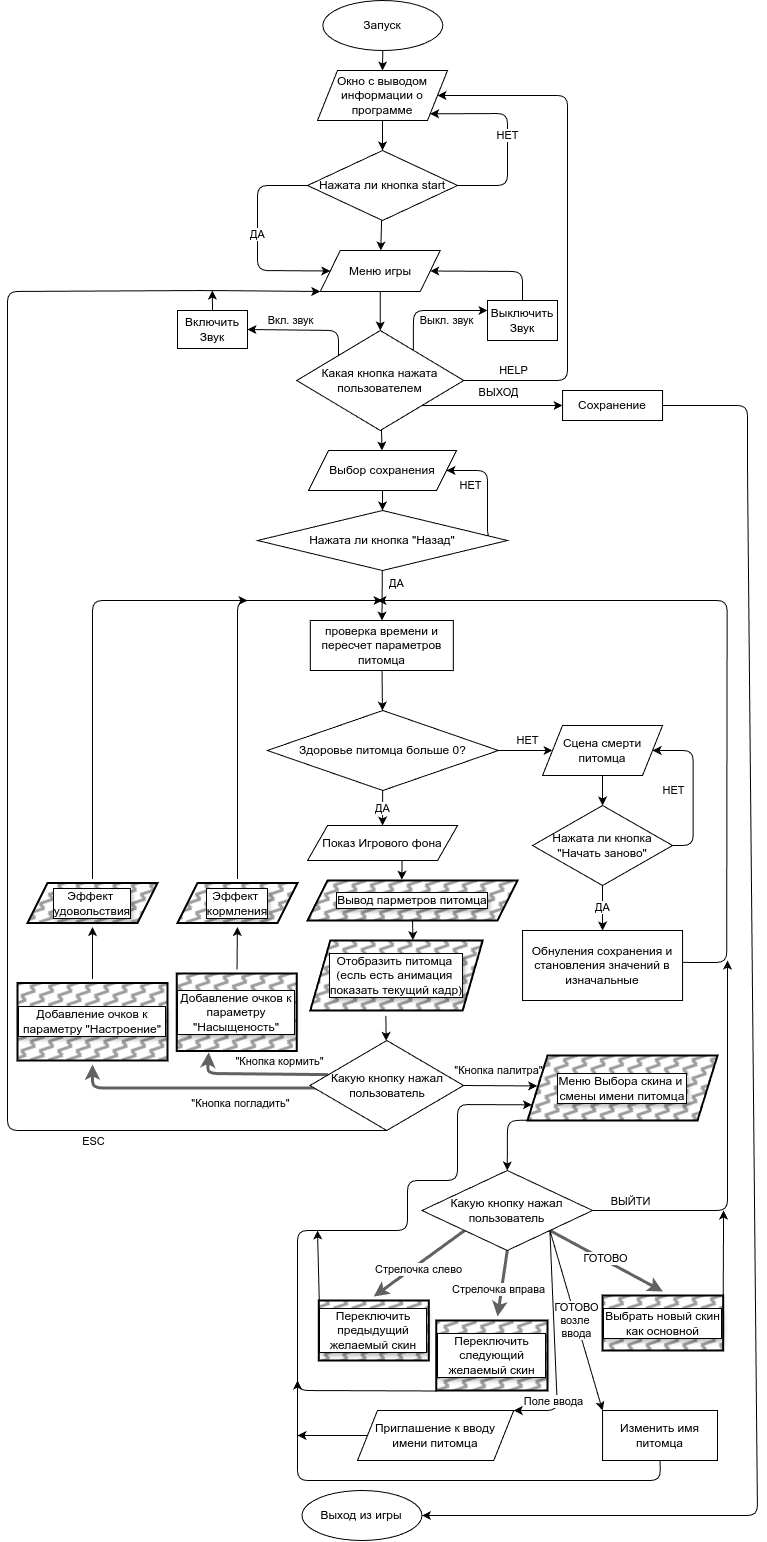


Рис. 1 Блок-схема

## Технические решения

### Менеджер сцен

Позволяет легко и быстро создавать различные сцены просто следуя структуре[3] которая описана в scene\_manager.h. (Рис. 2)

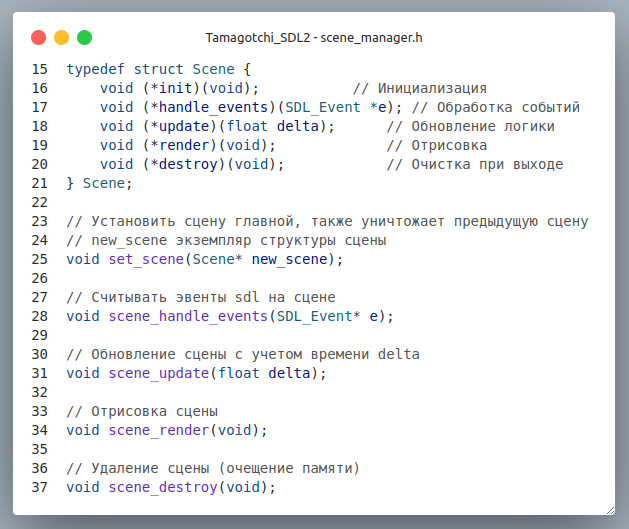


Рис. 2 Описание менеджера сцен

### Изображение питомца

Изображение зависит от размеров экрана, чтобы отображать его в центре экрана, а также учитывает изначальные размеры изображения, чтобы сохранить пропорции разных скинов питомца. В этой де функции show\_pet() также происходит проверку на здоровье, чтобы случае здоровье=0 вызвать экран смерти и предложить игроку начать заново (обнулив предыдущий прогресс). (См. Рис. 3 )



Рис. 3 Отображение питомца

Также данный код представлен в виде блок схемы на Рис. 4 .

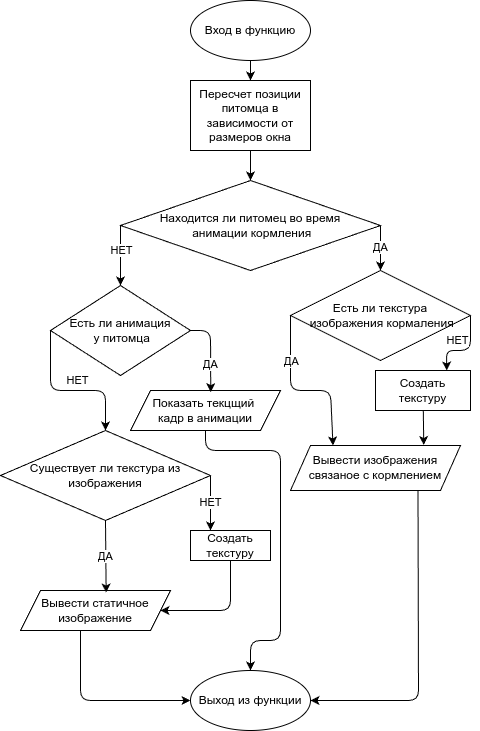


Рис. 4 Блок-схема функции show\_pet.

### Анимации

Анимации реализованы также с помощью структуры[4] и методы для взаимодействия с ней описанные в animation.h (См. Рис. 4 )

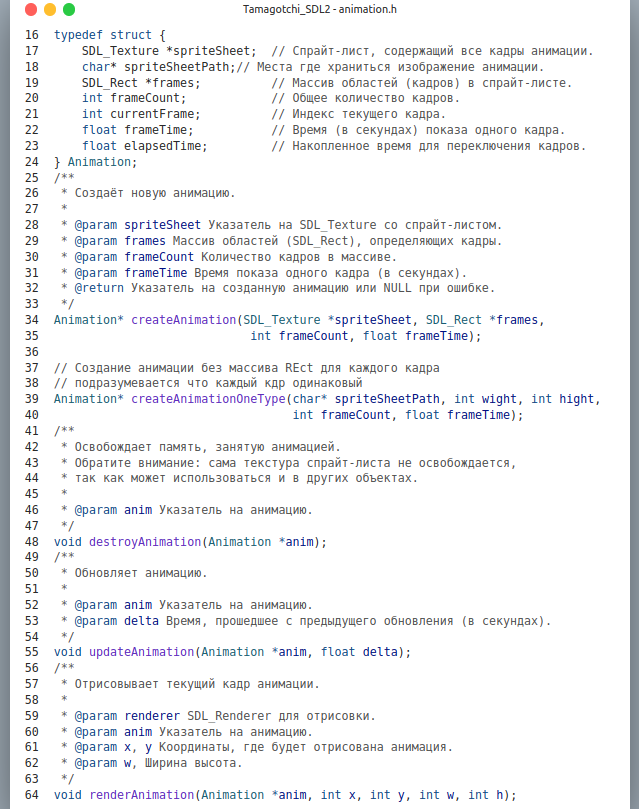


Рис. 5 Описание анимаций

### Шкалы параметров

Шкалы реализованы с помощью модуля SDL2\_gfx[5] для ровного и красивого изображения, используя сглаживание линий и встроенные методы округления[6]. В файле graphics.c есть функция для создания такой шкалы, в функции отрисовки она повторяется трижды, чтобы отразить три шкалы (См. Рис. 5)

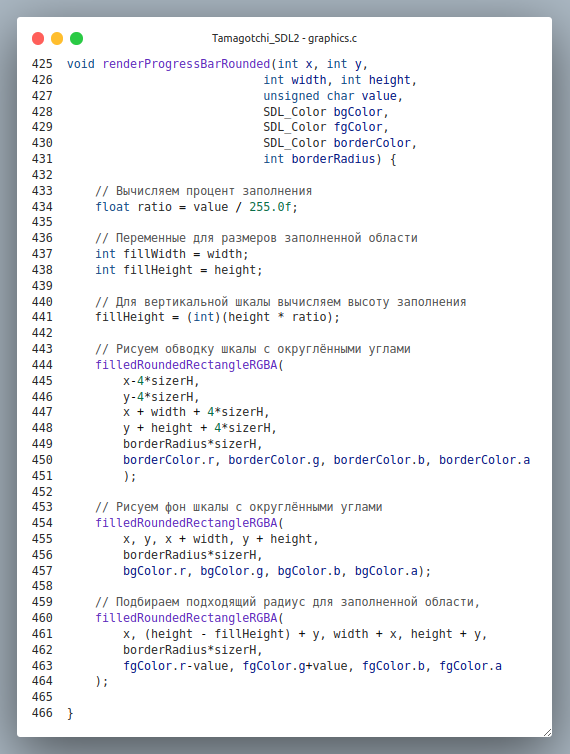


Рис. 6 Код шкалы параметров

### Меню персонализации (кастомиазция питомца смена его скинов)

Для изменения скина питомца была создана отдельная сцена menu\_pet которая описана в файле menu\_pet.c, а каждый скин, который можно выбрать для своего питомца, был описан в массиве структур SkinPet (См. Рис. 7 )

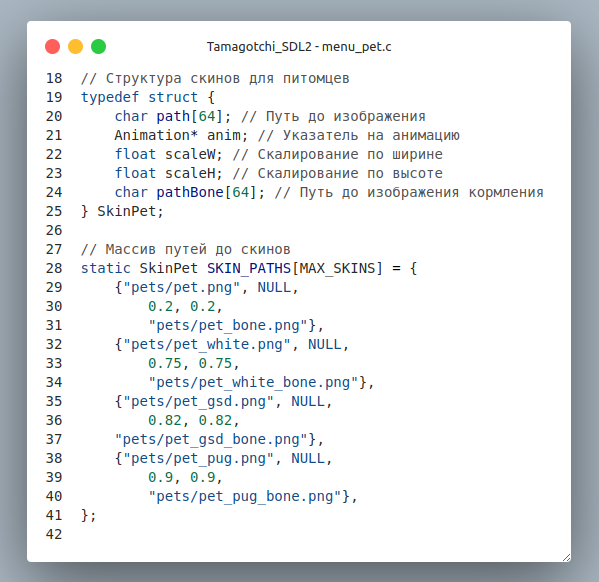


Рис. 7 Массив изображений питомца

Также более подробная логика сцены показана на Рис. 8.

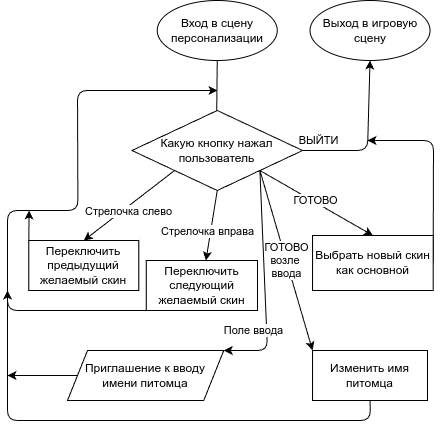


Рис. 8 Блок схема сцены для смены скинов питомца.

# Тестирование

Проверим работу программы в соответствии с заданными требованиями.

При открытии программы должно появится окно с основными данными о программе. После нажатия кнопки start должно появится меню игры.

При запуске программы видим, что, действительно, открылось окно со сведениями (См Рис. 9 )

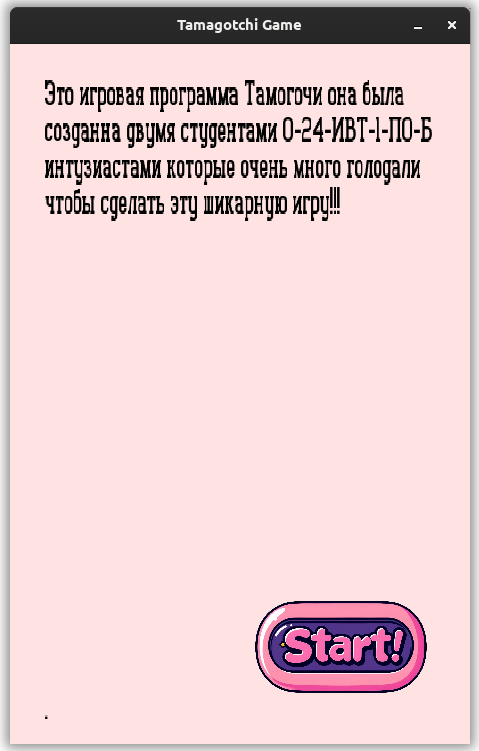


Рис. 9 Открытие игры

После его закрытия появилось меню игры (См. Рис. 10 )



Рис. 10 Меню игры

После нажатия на кнопку “start”, должно появиться окно с питомцем: тремя дисками и надписью о количестве шагов. Как показано на Рис. 11 , так и произошло.



Рис. 11 Игровая сцена скриншот

Можно также понажимать на кнопки, подождать и увидеть как параметры изменяются (См. Рис. 12 )



Рис. 12 Наглядное изменения параметров питомца

# Итоги проделанной работы

В ходе разработки игрового проекта «Тамагочи» на языке C с использованием библиотеки SDL2 была проанализирована предметная область, выбраны инструменты разработки, спроектирована структура кода и учтены возможные риски. Разработанная программная архитектура позволяет легко масштабировать проект, а использование Git обеспечивает удобную совместную работу над кодом.

Игровая программа оказалась максимально близка к своему оригиналу. Отличаясь новым визуалом и музыкально-звуковым сопровождением. Использовались игровые сцены для скорости разработки и читабельности итогового программного кода.

При разработке особое внимание уделялось кроссплатформенности, оптимальному управлению памятью и эффективности работы с графикой. Благодаря этому удалось создать игровую программу, обеспечивающую стабильную работу и корректную обработку пользовательского взаимодействия.

В случае дальнейшего развития программы можно ввести систему достижений, игровую валюту для покупки скинов и мини‑игры для ее заработка.

# Список литературы

1. **GameSpot.** Tamagotchi (Game Boy) [электронный ресурс] / GameFAQs. – Режим доступа: https://gamefaqs.gamespot.com/gameboy/562640-tamagotchi (дата обращения: 24.04.2025).
2. **Chacon, S., Straub, B.**  
   Pro Git: [учебное пособие] / S. Chacon, B. Straub. – 2-е изд. – New York: Apress, 2014. – 456 с.
3. **Kernighan, B. W., Ritchie, D. M.**  
   The C Programming Language: [учебное пособие] / B. W. Kernighan, D. M. Ritchie. – 2-е изд. – Englewood Cliffs: Prentice Hall, 1988. – 272 с. – ISBN 978-0131103627.
4. **Nystrom R. Game Programming Patterns**[Электронный ресурс] / R. Nystrom. — 2014. — 354 с. — Режим доступа:   
   http://gameprogrammingpatterns.com/ (дата обращения: 24.04.2025).
5. **SDL2\_gfx Documentation.**  
   [Электронный ресурс]. – Режим доступа:  
   http://www.ferzkopp.net/joomla/content/view/19/14/ (дата обращения: 24.04.2025).
6. **Lazy Foo’ Productions.**  
   SDL Tutorials: [электронный ресурс] / Lazy Foo’ Productions. – 2012. – Режим доступа:  
   http://lazyfoo.net/tutorials/SDL/ (дата обращения: 24.04.2025).