# ТЕРМИНЫ И ОПРЕДЕЛЕНИЯ

# СОДЕРЖАНИЕ

[ТЕРМИНЫ И ОПРЕДЕЛЕНИЯ 2](#_Toc198500620)

[СОДЕРЖАНИЕ 3](#_Toc198500621)

[**Глава 1. ИСТОРИЯ И ОСОБЕННОСТИ ЖАНРА** RTS **(Real-Time Strategy)** 6](#_Toc198500622)

[**Возникновение и становление жанра** 6](#_Toc198500623)

[**Ключевые механики и требования к геймплею в** RTS**-играх** 7](#_Toc198500624)

[**Архитектуры многопользовательских игр** 11](#_Toc198500625)

[Клиент-серверная архитектура 11](#_Toc198500626)

[Peer-to-peer (равный-равному) 13](#_Toc198500627)

[Авторитарный сервер 14](#_Toc198500628)

Современная индустрия компьютерных игр представляет собой динамично развивающуюся сферу цифровых технологий, которая оказывает значительное влияние на культурные, социальные и экономические аспекты современного общества. В условиях стремительной цифровизации различных областей человеческой деятельности компьютерные игры перестали быть просто развлекательным продуктом, превратившись в сложные программно-аппаратные комплексы, сочетающие в себе элементы искусства, технологии и науки. Особое место в этом многообразии занимает жанр стратегий реального времени (RTS), который на протяжении нескольких десятилетий остается одним из наиболее сложных и требовательных к технической реализации направлений игровой индустрии.

Актуальность разработки многопользовательской компьютерной игры в жанре RTS "Боярский турнир" обусловлена рядом значимых факторов. Во-первых, современный рынок игровой продукции демонстрирует устойчивый спрос на качественные стратегические игры, что подтверждается коммерческим успехом таких проектов, как Age of Empires IV, Company of Heroes 3 и других представителей жанра. Во-вторых, наблюдается постоянное технологическое развитие инструментария для создания игровых приложений, включая появление новых языков программирования и специализированных движков, что открывает перед разработчиками ранее недоступные возможности.

Особый интерес в контексте данной работы представляет использование языка программирования Rust и игрового движка Bevy. Rust, появившийся в 2010 году и достигший стабильной версии 1.0 в 2015 году, за последнее десятилетие зарекомендовал себя как надежное решение для разработки высокопроизводительных и безопасных систем. Его уникальная система владения и заимствования (ownership system) обеспечивает защиту от целого класса ошибок, характерных для системного программирования, что особенно важно при создании многопользовательских игровых приложений. Bevy, в свою очередь, представляет собой современный data-driven игровой движок, который, хотя и является относительно молодым проектом (первый стабильный релиз состоялся в 2020 году), уже демонстрирует значительный потенциал для создания сложных игровых систем.

Техническая сложность разработки RTS-игр традиционно связана с необходимостью решения целого комплекса взаимосвязанных задач. Среди них можно выделить: реализацию сложного искусственного интеллекта для управления юнитами, создание эффективной системы сетевой синхронизации для многопользовательского режима, а также разработку сбалансированной экономической модели игры. Именно эти аспекты делают процесс создания RTS-игры исключительно интересным с точки зрения исследования возможностей современных технологий игровой разработки.

Историческая тематика проекта "Боярский турнир" добавляет дополнительный уровень сложности и одновременно открывает новые возможности для исследования. Реализация достоверного исторического антуража требует тщательной работы с источниками и внимания к деталям, в то время как игровая составляющая предполагает определенную степень условности и абстракции. Этот баланс между исторической достоверностью и игровой условностью представляет собой отдельную исследовательскую задачу в рамках разработки проекта.

WНаучная новизна исследования заключается в комплексном анализе возможностей языка Rust и движка Bevy для разработки игр жанра RTS, что представляет особый интерес в свете относительно небольшого количества подобных реализаций на текущий момент. Практическая значимость работы определяется тем, что разработанный прототип может служить основой для дальнейшего развития в полноценный коммерческий продукт, а полученные в ходе разработки результаты могут быть полезны другим исследователям и разработчикам, работающим с аналогичными технологиями.

Методологическую основу исследования составляют современные подходы к разработке программного обеспечения, принципы объектно-ориентированного и системного программирования, а также специализированные методики, применяемые в игровой индустрии. В процессе работы использовались такие методы как прототипирование, модульное тестирование, профилирование производительности и итеративная разработка.

Структура работы соответствует классической схеме научно-исследовательской работы и включает введение, аналитический обзор, описание методики разработки, представление результатов, заключение и список использованных источников. Такая организация материала позволяет последовательно раскрыть все аспекты проведенного исследования и представить полученные результаты в систематизированном виде.

## **Глава 1. ИСТОРИЯ И ОСОБЕННОСТИ ЖАНРА** RTS **(Real-Time Strategy)**

### Возникновение и становление жанра

Жанр стратегий в реальном времени (англ. **Real-Time Strategy**, сокращённо RTS) начал формироваться в конце 1980-х — начале 1990-х годов на фоне стремительного развития персональных компьютеров. Первые эксперименты в жанре имели форму пошаговых стратегий, однако с развитием вычислительных мощностей появилась возможность реализовать игры, где действия происходят в реальном времени, без пауз между ходами игроков.

Одной из первых полноценных RTS-игр считается **Herzog Zwei (1989)** на консоли **Sega Mega Drive**. Она представила основы жанра: управление юнитами, строительство базы, сбор ресурсов и ведение боевых действий в реальном времени. Однако настоящий расцвет RTS пришёлся на 1990-е годы.

Прорывной стала игра **Dune II: The Building of a Dynasty (1992)** от студии **Westwood Studios**. Именно эта игра ввела многие каноны жанра, ставшие стандартом:

* Управление юнитами с помощью мыши
* Разделение на фракции с уникальными особенностями
* Строительство базы и добыча ресурсов
* Постепенное развитие технологий

Следующим этапом стало появление **Warcraft: Orcs & Humans (1994)**, а затем **Command & Conquer (1995)** и **StarCraft (1998)**, каждая из которых добавляла новшества в геймплей и углубляла стратегические аспекты. Эти игры получили не только коммерческий успех, но и заложили основы для киберспорта, особенно в Южной Корее, где **StarCraft** стал национальной игрой.

В 2000-х жанр продолжал развиваться, появились:

* **Age of Empires II** — с упором на историческую достоверность и экономику
* **Warcraft III** — с добавлением ролевых элементов (герои, прокачка)
* **Company of Heroes** — с более реалистичной боевой системой и физикой

С началом 2010-х годов классические RTS стали терять популярность из-за роста интереса к **MOBA**, шутерам и мобильным играм. Тем не менее, жанр не исчез. Он начал развиваться в более нишевом направлении. Появились инди-RTS, такие как **They Are Billions**, **Northgard** и **Zero-K**, а крупные студии возвращались к истокам через ремастеры (**StarCraft: Remastered, Age of Empires: Definitive Edition**). Вышел **Age of Empires IV** (2021), вновь привлекший внимание к классическим RTS.

Также стоит отметить, что RTS оказали большое влияние на развитие других жанров. Из **Warcraft III** выросла **Dota**, ставшая родоначальницей **MOBA**, а элементы RTS встречаются в стратегиях жанра **4X** и даже в современных **выживалках** и градостроительных симуляторах.

### Ключевые механики и требования к геймплею в RTS-играх

Жанр стратегий в реальном времени (RTS) обладает уникальной структурой геймплея, отличающей его от других игровых жанров. Основу любой RTS составляют взаимосвязанные ключевые механики, обеспечивающие стратегическую и тактическую глубину игрового процесса. От качества реализации этих механик зависит баланс, общая заинтересованность и реиграбельность проекта. Осмотрев несколько реализованных проектов в жанре RTS**,** можно выделить основные компоненты геймплея RTS-игр, а также требования, предъявляемые к каждой из них:

**Сбор ресурсов** — один из базовых элементов геймплея. Ресурсы служат основой для постройки зданий, создания юнитов и исследований. Наиболее часто встречающиеся типы ресурсов:

* Добываемые (золото, руда, нефть, древесина и т.п.)
* Производимые (энергия, еда, рабочая сила)
* Временные (мана, очки технологий, таймеры)

**Требования**:

* Ресурсная система должна быть сбалансированной, чтобы ограничивать чрезмерный рост игрока.
* Добыча должна быть сопряжена с рисками (например, ближнее месторождение — безопасное, но быстро истощается; дальнее — богатое, но требует защиты).

**Постройка и развитие базы** — это стратегическая составляющая игры, обеспечивающая игрока инфраструктурой. Игрок размещает здания, каждое из которых выполняет определённую функцию (обработка ресурсов, военное производство, исследование технологий, защита).

**Требования**:

* Базы должны быть уязвимыми — их разрушение должно играть ключевую роль в сражении.
* Игрок должен иметь тактический выбор: оборонительная база, агрессивная экспансия, технологическое развитие.
* Должна быть чёткая архитектура зависимости: нельзя построить высокоуровневое здание без предварительных условий (техдрево).

**Тренировка войск.** Игроку предоставляется возможность производить юниты различных классов: пехота, техника, авиация, морские силы и т.д. Юниты используются для захвата территории, защиты базы, разведки и атаки противника.

**Требования**:

* Разнообразие юнитов должно обеспечивать тактическую вариативность.
* Должна быть реализована система классов. Все юниты должны иметь чётко определённые роли (танк — бронирован, но медлителен; разведчик — быстр, но уязвим и т.д.).
* В игре должен быть механизм группирования и массового управления отрядами.

**Боевые столкновения** — ключевой элемент RTS. В отличие от шутеров или файтингов, здесь бои происходят между группами юнитов, часто — на нескольких фронтах.

**Требования**:

* Баланс сил: ни один юнит или тактика не должен доминировать.
* Влияние рельефа и расположения (например, высоты, леса, мосты).
* Влияние времени реакции игрока — игрок, быстрее отдавший команды, получает преимущество.
* Поддержка **приоритетов атак** и **поведения юнитов** (например, “атаковать всех”, “удерживать позицию”, “преследовать”).

Механика **технологического дерева** позволяет игрокам открывать новые юниты, улучшения и возможности. Исследования требуют времени и ресурсов и влияют на дальнейший стиль игры.

**Требования**:

* Исследования должны давать ощутимое преимущество, но не нарушать баланс.
* Игрок должен выбирать стратегию развития: быстрые дешёвые технологии или медленное, но мощное развитие.
* Древо должно быть визуально понятно и логично структурировано.

**Информативность.** Знание позиции противника, местонахождения ресурсов и возможных угроз — основа стратегического планирования.

**Требования**:

* **Туман войны** — область карты, недоступная для обзора, если там нет юнитов или зданий игрока.
* Механики разведки: лёгкие юниты, воздушные шары, радары, обсерватории и пр.

**ИИ** реализует поведение как вражеских, так и союзных фракций в одиночной игре. Важно, чтобы он был предсказуем, но не тривиален.

**Требования**:

* Гибкая реакция на действия игрока (например, адаптация стратегии).
* Планирование атак, контратак и обороны.
* Возможность настраивать уровень сложности.

**Эффективное управление** — важнейшая часть геймплея RTS. Игрок должен уметь быстро отдавать приказы, анализировать ситуацию и контролировать несколько процессов одновременно.

**Требования**:

* Поддержка “горячих клавиш” и макросов.
* Миникарта с возможностью управления.
* Интуитивный интерфейс для отображения ресурсов, производства и юнитов.

**Мультиплеер.** RTS часто имеют мультиплеерные режимы, включая PvP и командную игру.

**Требования**:

* Надёжная синхронизация состояний игроков (часто используется **lock-step** или **deterministic simulation**).
* Баланс игровых сторон.
* Система матчмейкинга, перезапуска, наблюдения.

### Архитектуры многопользовательских игр

Организация сетевого взаимодействия между игроками является одной из ключевых задач при разработке многопользовательских игр. От выбранной архитектуры зависит не только производительность и масштабируемость проекта, но и его уязвимость к читам, устойчивость к сбоям и даже восприятие отклика управления со стороны пользователя. На практике наиболее широко применяются три архитектурных подхода: **клиент-сервер**, **peer-to-peer (равный-равному)** и **авторитарный сервер**. Каждый из них имеет свои особенности, сценарии применения и компромиссы, которые важно учитывать при проектировании.

### Клиент-серверная архитектура

Наиболее традиционный подход — это организация игры по модели клиент-сервер, при которой существует выделенный узел (сервер), отвечающий за координацию всех подключённых клиентов. Каждый игрок управляет своей копией клиента, который передаёт свои действия (например, движение, атака или использование способности) на сервер. Сервер, в свою очередь, обрабатывает эту информацию, рассчитывает результат и рассылает его остальным клиентам, обеспечивая согласованность и синхронность игрового мира. Часто именно сервер в этой архитектуре отвечает за значительную часть логики игры, включая физику, коллизии и правила взаимодействия. Клиенты же в таком случае являются в основном визуализаторами — они отображают состояние, полученное от сервера, и передают пользовательский ввод. Однако возможны и гибридные реализации, в которых часть логики (например, анимации или локальные предсказания движения) остаётся на клиенте для снижения визуальных задержек.

**Преимущества:**

* Централизованное управление игрой упрощает синхронизацию состояния между всеми участниками.
* Более высокий уровень защиты от несогласованных действий клиентов и потенциального читерства.
* Относительная простота масштабирования за счёт управления потоками и балансировки нагрузки на уровне сервера.

**Недостатки:**

* Сервер является единой точкой отказа: при его отключении вся игровая сессия становится недоступной.
* Возможна повышенная латентность при удалённом расположении серверов относительно игроков.
* Увеличенные требования к хостингу и сетевым ресурсам, особенно при большом числе одновременно подключённых игроков.

### Peer-to-peer (равный-равному)

Архитектура peer-to-peer строится на прямом взаимодействии между клиентами без участия центрального сервера. Все игроки в такой модели связаны друг с другом и обмениваются данными напрямую. Каждый участник может как отправлять, так и получать данные, причём в некоторых реализациях все клиенты обладают равными правами и возможностями — отсюда и название. Одним из классических примеров такой архитектуры является реализация с использованием алгоритма **lockstep**: каждый клиент сначала отправляет свои действия, дожидается получения команд от других участников, и лишь затем одновременно выполняет игровой шаг. Такой подход особенно популярен в старых RTS-играх, где каждый игровой тик требует согласованных действий всех игроков. Однако в современных условиях peer-to-peer сталкивается с рядом серьёзных ограничений, включая сложности при пробивании **NAT**, защиту от модификации данных и обеспечение справедливости.

**Преимущества:**

* Отсутствие необходимости в выделенном сервере позволяет уменьшить затраты на инфраструктуру.
* Низкая задержка при игре в локальной сети или между игроками вблизи друг друга.
* Возможность децентрализации — при выходе одного игрока другие могут продолжать сессию при наличии механизмов восстановления.

**Недостатки:**

* Уязвимость к читам: каждый клиент может вмешиваться в игровой процесс, а обнаружить нарушения крайне трудно.
* Трудности синхронизации и восстановления согласованность и синхронность игрового состояния.
* Плохая масштабируемость и растущая нагрузка при увеличении числа участников.

### Авторитарный сервер

Архитектура с авторитарным сервером представляет собой усиленный вариант клиент-серверной модели. В этой конфигурации сервер является единственным источником истины: все игровые вычисления, включая проверку корректности действий, физику, столкновения и последствия команд, происходят исключительно на его стороне. Клиенты в такой архитектуре выполняют исключительно роль интерфейса — они отправляют запросы на сервер и отображают полученный от него результат. Для обеспечения плавности игрового процесса клиент может применять локальную предикцию действий и интерполяцию состояний. Однако в случае расхождений между предсказанным и полученным от сервера состоянием требуется механизм отката и повторной синхронизации. Такой подход особенно актуален для соревновательных игр, где честность и согласованность важнее мгновенного отклика.

**Преимущества:**

* Максимальная защита от несанкционированных изменений: весь игровой процесс контролируется на стороне сервера.
* Предсказуемое поведение системы даже при попытках сетевых атак или вмешательства в клиентский код.
* Полная централизованная аналитика и контроль, что облегчает отладку и поддержку.

**Недостатки:**

* Повышенная задержка между вводом игрока и реакцией игрового мира (особенно без предикции на клиенте).
* Увеличенные требования к производительности сервера, особенно при большом количестве активных игроков.
* Повышенная сложность реализации, требующая точной синхронизации состояний и продуманной архитектуры rollback-процессов.

### Протоколы и технологии

Сетевые протоколы играют критическую роль в разработке многопользовательских игр, особенно в жанре RTS, где важны как своевременная доставка данных, так и их корректность. Правильный выбор протокола может существенно повлиять на плавность геймплея, устойчивость к сетевым потерям, а также общую производительность игры. Традиционно основными транспортными протоколами, лежащими в основе сетевых приложений, являются **TCP** и **UDP**. Однако в последнее десятилетие на фоне развития технологий появился более универсальный подход — протокол **QUIC**, который постепенно вытесняет классические решения. Именно его применяет современная библиотека **bevy\_quinnet**, которая используется в рамках твоего проекта на **Bevy**.

**TCP (Transmission Control Protocol)** — это ориентированный на соединение протокол, предназначенный для надёжной передачи данных между двумя конечными точками. **TCP** гарантирует, что все отправленные данные дойдут до получателя в точности в том же порядке, в каком были отправлены. Этот протокол реализует механизмы подтверждений доставки (**ACK**), повторной передачи утерянных пакетов, контроль перегрузки сети (congestion control) и автоматическую адаптацию скорости передачи. Эти свойства делают **TCP** надёжным, но одновременно накладывают ограничения, связанные с задержками. В случае потери пакета, передача приостанавливается до тех пор, пока потерянный сегмент не будет повторно получен. Такая ситуация называется head-of-line blocking — блокировка следующей информации из-за недоставки одного предыдущего элемента. Это неприемлемо в играх реального времени, где задержка даже в несколько сотен миллисекунд может привести к рассинхронизации игрового мира или нарушению взаимодействия игроков. Тем не менее, **TCP** по-прежнему широко используется для обмена важной, но нечасто изменяющейся информацией — например, для аутентификации игроков, загрузки игровых данных, чатов или обновлений инвентаря.

В противоположность ему, **UDP (User Datagram Protocol)** — это протокол без установления соединения, ориентированный на минимальные задержки и скорость доставки. Он не гарантирует ни доставки, ни порядка пакетов, ни их целостности. Именно поэтому **UDP** идеально подходит для быстрой передачи игровых состояний, координат объектов, движений юнитов — всех тех данных, которые устаревают через доли секунды и не требуют повторной отправки в случае потери. Благодаря своей простоте, **UDP** обеспечивает минимальную задержку и не страдает от блокировок, как **TCP**. Однако из-за отсутствия встроенных механизмов надёжности разработчику приходится реализовывать собственные алгоритмы для обработки потерь, восстановления порядка сообщений и синхронизации состояний. Это приводит к усложнению логики, особенно в условиях нестабильных сетей. В результате возникает дилемма: либо использовать **TCP** и страдать от задержек, либо использовать **UDP** и самому решать проблемы сетевой нестабильности.

В ответ на эти ограничения в 2013 году компания Google начала разработку нового транспортного протокола под названием **QUIC (QUICk UDP Internet Connections)**. **QUIC** работает поверх **UDP**, но реализует на уровне пользовательского пространства те же функции, что и **TCP**, включая надёжную доставку, порядок сообщений, мультиплексирование потоков и шифрование (через встроенный **TLS 1.3**). Главное отличие **QUIC** от **TCP** заключается в том, что он устраняет проблему head-of-line blocking: если один поток сталкивается с потерей пакета, остальные потоки продолжают передаваться. Это делает **QUIC** особенно привлекательным для многопоточной передачи в играх, где одновременно передаются координаты, игровые события и пользовательские действия. Более того, **QUIC** обеспечивает очень быстрое установление соединения — в большинстве случаев за один-единственный раунд обмена, в отличие от трёхступенчатого рукопожатия в **TCP**. В итоге **QUIC** сочетает надёжность и контроль, присущий **TCP**, с низкой задержкой и гибкостью, характерной для **UDP**.

На фоне распространения **QUIC** в современных браузерах, мобильных приложениях и игровых движках появилась потребность в библиотечной реализации протокола на разных языках. Одной из таких реализаций является **Quinn** — асинхронная реализация **QUIC** на языке Rust. Она легла в основу сетевой библиотеки bevy\_quinnet, используемой в игровом движке Bevy. Библиотека bevy\_quinnet предоставляет высокоуровневый API для организации клиент-серверного взаимодействия на базе **QUIC**, абстрагируя разработчика от всех низкоуровневых деталей шифрования, мультиплексирования и обработки сетевых пакетов. Она автоматически управляет сессиями, подключениями и безопасной передачей данных, позволяя разработчику сосредоточиться на логике синхронизации игрового состояния. Её архитектура идеально подходит для RTS-игр, где требуется одновременно высокая скорость, надёжность и масштабируемость.

### Аналоги

### Clash Royale



Рисунок 1 — Эмблема игры

Приложение ***Clash Royale*** представляет собой мобильную игру в жанре стратегической карточной арены с элементами защиты башен и боя в реальном времени. Игра разработана студией **Supercell** и предназначена для устройств на платформах **Android** и **iOS**. На рисунке 1 изображена эмблема игры. Игровой процесс заключается в противостоянии двух игроков на арене, разделённой на две половины с тремя башнями на каждой стороне. Цель игрока — разрушить центральную башню соперника, либо нанести максимальный урон в течение ограниченного времени. Игрок использует заранее собранную **колоду из восьми карт**, каждая из которых представляет собой боевого юнита, защитное здание или заклинание. Для использования карт требуется ресурс **эликсир**, который восстанавливается с течением времени. Игра проходит в реальном времени, что требует от пользователя стратегического мышления и быстрой реакции.

Основные особенности приложения:

* **Сбор и улучшение карт** — каждая карта может быть улучшена при помощи внутриигровых ресурсов, что повышает её характеристики;
* **Быстрые PvP-сражения** — матч между двумя игроками длится в среднем 3–5 минут, что позволяет запускать игру в любое удобное время;
* **Система рейтингов и арен** — в зависимости от уровня и побед игрока, он открывает новые арены с уникальными наборами карт;
* **Минималистичный интерфейс** — управление осуществляется с помощью одного касания, а весь бой проходит на одном экране;
* **Постоянное подключение к серверу** — игра требует интернет-соединения, так как все матчи проходят в режиме онлайн;
* **Кланы и социальное взаимодействие** — игроки могут объединяться в кланы, делиться картами, общаться и участвовать в клановых войнах;

Пользовательский интерфейс основных экранов игры продемонстрирован на рисунке 2.

--------------------------------КАРТИНКА---------------------------------

Рисунок 2 — Интерфейс главного экрана игры ***Clash Royale***

### Rush Royale



Рисунок 3 — Эмблема игры Rush Royale

Приложение **Rush Royale** представляет собой мобильную стратегическую игру с упором на быстрые карточные бои в режиме реального времени. Разработана студией **RushGames** и доступна на платформах **Android** и **iOS**. Игроки сражаются друг с другом на ограниченной арене, используя колоду из 10 карт, каждая из которых представляет юнита, защитное сооружение или заклинание. Основная задача — уничтожить главную базу соперника. Матчи длятся в среднем 4–6 минут, что обеспечивает динамичный игровой опыт. Для разыгрывания карт используется ресурс — мана, которая постепенно восстанавливается во время матча. Различные карты требуют разного количества маны, что заставляет игрока грамотно планировать свои ходы и управлять ресурсами. Юниты обладают уникальными характеристиками — уровнем здоровья, уроном, скоростью передвижения и атакой. Также на поле боя присутствуют здания с функцией обороны или генерации дополнительных ресурсов.

Матчи проходят в реальном времени, что требует от игрока не только стратегического мышления, но и быстрой реакции для контратак и адаптации к действиям соперника.

Основные особенности приложения:

* Колода состоит из 10 карт, которые можно собирать и улучшать. Каждая карта имеет уровень, влияющий на её боевые характеристики (урон, здоровье, скорость атаки). Улучшение карт требует внутриигровых ресурсов — золота и кристаллов.
* Все матчи проходят в режиме реального времени с постоянной синхронизацией состояния арены между клиентами и сервером. Используется клиент-серверная архитектура с сервером авторитетным по состоянию игры для предотвращения читинга.
* Игроки получают опыт и внутриигровые ресурсы за победы, что позволяет повышать уровень аккаунта и открывать новые карты. В игре реализована система уровней карт, влияющая на эффективность юнитов.
* В игре предусмотрена система подбора соперников по рейтингу (MMR), чтобы обеспечить равные условия боя. Также реализованы периодические обновления баланса карт, чтобы избежать доминирования определённых стратегий.
* Карты делятся на классы юнитов: ближний бой, дальний бой, защитные сооружения, заклинания с разными эффектами (замедление, урон по области и т.д
* Интерфейс адаптирован под сенсорное управление, все действия выполняются простыми касаниями и перетаскиваниями карт на поле. Вся игровая информация о состоянии карты, запасе маны и времени матча отображается на одном экране.
* Игра содержит систему лиг и рейтингов, обновляемых после каждого матча. Лиги открывают доступ к новым картам и турнирам с призами.
* Игроки могут вступать в кланы, где происходит обмен картами, участие в совместных клановых турнирах и выполнение клановых заданий. Это расширяет социальный аспект игры.

Пользовательский интерфейс основных экранов игры продемонстрирован на рисунке 4.

* --------------------------------КАРТИНКА---------------------------------
* Рисунок 4 — Интерфейс главного экрана игры ***Clash Royale***

### Боярский турнир

**«Боярский турнир»** — это стратегия в реальном времени с древнерусской тематикой, вдохновлённая механикой **Clash Royale**. Игроки управляют отрядами боевых юнитов, сражаясь на арене, призывая на поле боя персонажей древнерусской тематики с помощью предоставленной боевой колоды.

Визуальная часть **Боярского турнира** состоит из:

* **Юнитов** — боевых персонажей с уникальным внешним видом и характеристиками:
  + Рус (воин с мечом, синий с золотом)
  + Гигант (крупный юнит в рубахе)
  + Жрец (с посохом, в красной рубахе с амулетом)
  + Бомбер (подрывник в чёрной броне, кидающий бомбы)
  + Летучей мыши (миньоны похожие на летучих мышей)
  + Мушкетер (стрелок с огнестрельным оружием)
* **Спрайтов** — для каждого юнита создаются изображения в 4 направлениях (С, Ю, В, З) для анимации и ориентации на поле боя.
* **Арены и карты** — игровое поле, где происходит сражение, с контролем территорий и стратегическим размещением юнитов.
* **Интерфейса** — меню управления, отображение информации о картах в руке, следующей боевой карте после розыгрыша и текущем количестве эликсира.

Основные возможности приложения:

* Сетевая игра с подключением и синхронизацией действий двух игроков в реальном времени. Она позволяет обмениваться игровыми событиями и состояниями без задержек.
* Призыв юнитов на поле боя с помощью боевых карт из руки. Последующее передвижение и действия юнитов на поле реализовано с помощью искусственного интеллекта. При этом реализуются атаки, защита и взаимодействия между юнитами.
* Реализованы союзные и вражеские базы в виде 3-х основных башен: королевской башни – основного, главного здания; двух башней лучников – защитные пристройки для защиты королевской башни. Башни автоматически атакуют врагов в пределах радиуса действия, причем королевская башня начинает атаковать только после того, как одна из башен лучников падёт. Игра контролирует стрельбу и нанесение урона по целям.
* Различные типы юнитов имеют уникальные количества здоровья, типа атаки, типа передвижения и урона.
* Для юнитов с дальним типом атаки реализованы снаряды. Обработка столкновений отслеживает попадания снарядов в юнитов и башни. На основе этого рассчитывается нанесённый урон и обновляется состояние здоровья.
* Интерфейс отображает информацию о текущей руке (4-х доступных картах боевой колоды), следующей карте и количестве накопленного эликсира. Он позволяет игроку взаимодействовать с текущей рукой для призыва юнитов, если позволяет текущее количество эликсира.

**СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**