Ду Гомин, магистрант,
Амурский государственный университет

ПРОЕКТИРОВАНИЕ И РЕАЛИЗАЦИЯ СЕТЕВОЙ АРХИТЕКТУРЫ МНОГОПОЛЬЗОВАТЕЛЬСКОЙ ИГРЫ «ТАНКОВЫЙ БОЙ» НА JAVA

Аннотация. Представлена реализация многопользовательской версии «Танкового боя» на языке Java с клиент-серверной архитектурой и обменом через сокеты. Описаны организация сетевого взаимодействия, модель синхронизации общего состояния и интеграция подсистем управления, коллизий, визуальных эффектов и чата в единый канал обмена без деградации производительности.

Ключевые слова: Java; клиент-сервер; сокеты; синхронизация состояния; игровая логика; обработка столкновений; реальное время.

Игра опирается на идею централизованного управления миром: все критические решения принимаются на сервере, а клиенты отвечают за сбор ввода и визуализацию подтверждённых изменений. Такой подход упрощает поддержание согласованности между участниками и уменьшает количество неустранимых конфликтов. Прежде чем перейти к деталям, зафиксируем общую картину обмена: клиент инициирует соединение, сервер регистрирует участника и начинает принимать поток событий, после чего регулярно рассылает подтверждённые изменения всем подключённым. Именно сервер остаётся единственным источником истины – это допущение лежит в основе правил синхронизации, коллизий и начисления урона.

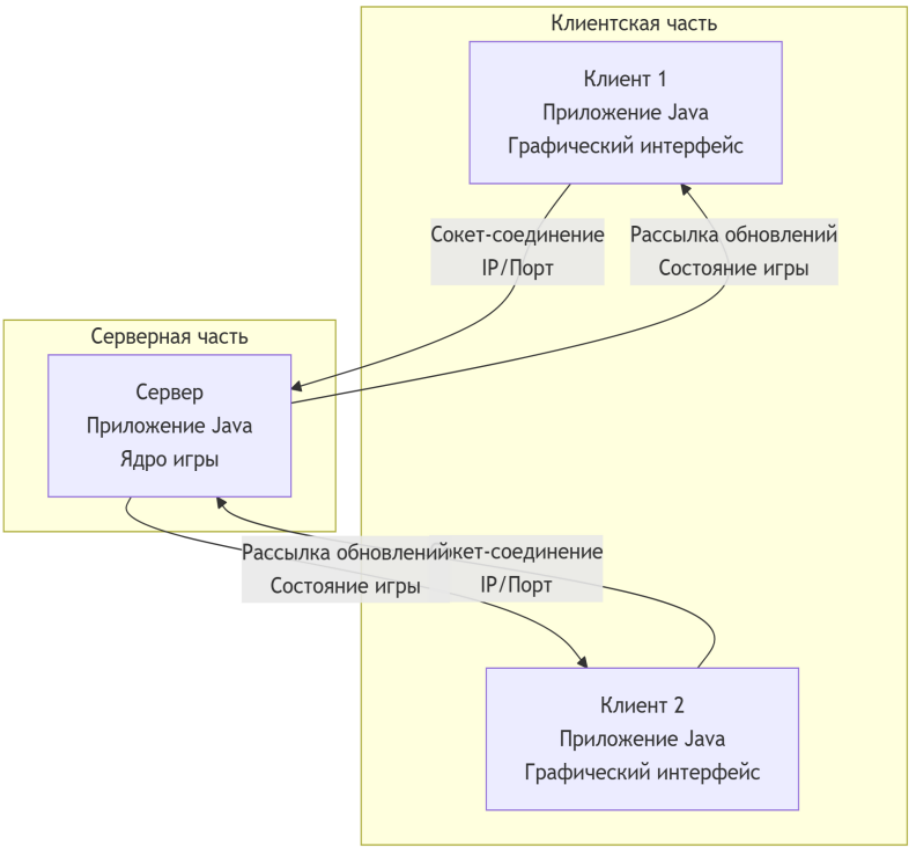

Рисунок 1 – Сетевая архитектура: сервер, клиенты и рассылка состояния

Архитектура системы реализована по клиент-серверной модели с четким разделением ответственности. Серверное ядро на Java содержит авторитетное состояние игрового мира и механизм арбитража, в то время как клиентское приложение отвечает за взаимодействие с пользователем и визуализацию.

Для обеспечения отзывчивости управления применяется оптимизированный протокол обмена, передающий только измененные параметры объектов, в сочетании с механизмом клиентского предсказания. Этот подход позволяет сохранять плавность геймплея даже при сетевых задержках, поскольку клиент локально прогнозирует непрерывные действия, а серверные корректировки мягко нивелируют возможные расхождения.

Критичные с точки зрения игрового баланса события, такие как применение урона и разрушение объектов, обрабатываются исключительно на сервере. Это гарантирует детерминированность и честность игрового процесса для всех участников, сохраняя при этом высокую отзывчивость интерфейса.

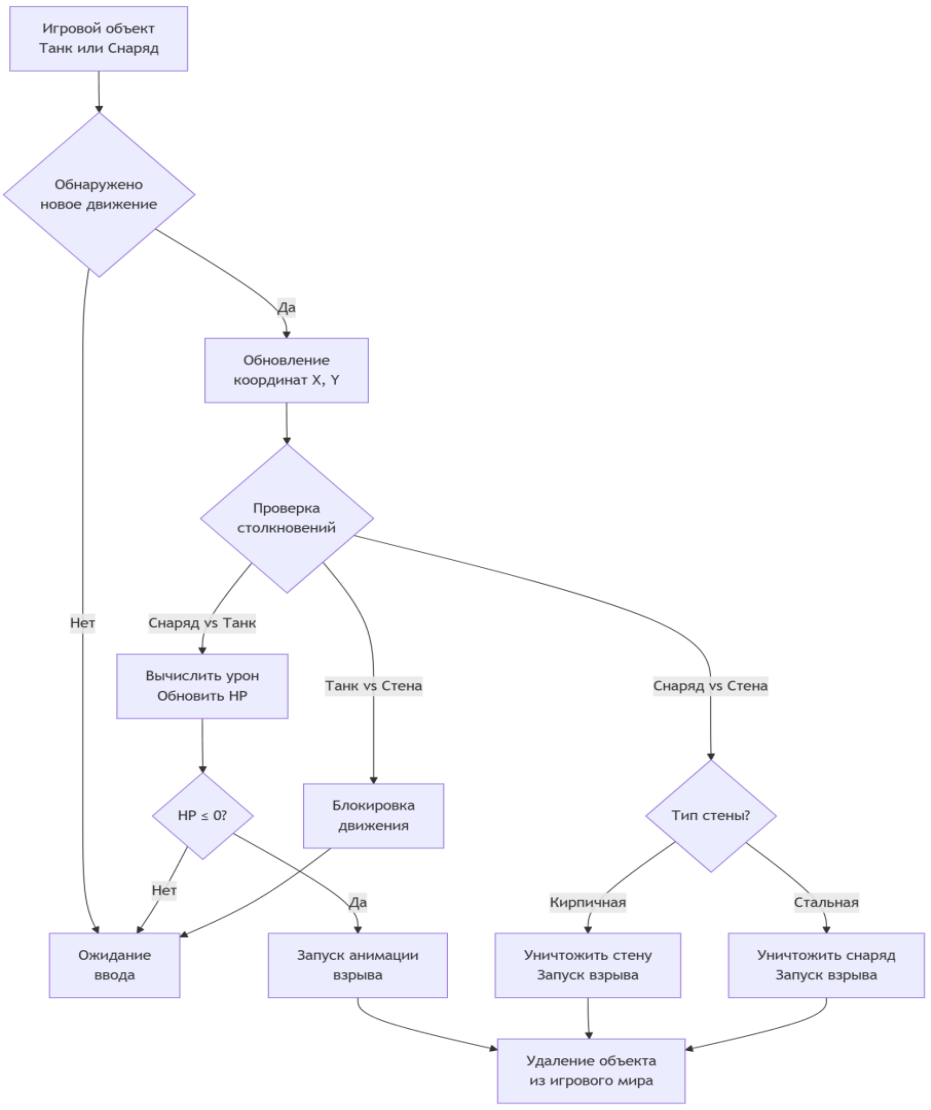

Рисунок 2 – Логика столкновений и обработки событий: танк/снаряд/стена

Сетевой канал обслуживает не только игровой цикл, но и сервисные функции: встроенный чат идёт по тому же сокет-соединению как низкоприоритетные пакеты, что

избавляет от дополнительных портов, упрощает маршрутизацию и не влияет на частоту рассылки состояния. Потери компенсируются ретраями, а идемпотентные обработчики на сервере предотвращают повторное применение уже учтённых событий.

Тестирование в реальных сетевых условиях с разными задержками и пропускной способностью позволило подобрать частоту серверных «тиков» и темп дельта-рассылок так, чтобы балансировать свежесть данных и объём трафика. Практика показала: частота, кратная скорости изменения объектов, даёт оптимум «плавность/нагрузка», а мягкая коррекция позиций на клиентах практически устраняет визуальные скачки. При обрывах клиент сохраняет последний подтверждённый снимок, пытается переподключиться и после восстановления получает накопленные изменения без перезапуска.

В кодовой базе выдержана строгая декомпозиция: серверное ядро инкапсулирует правила игры и не зависит от UI, сетевой слой изолирован от формата сериализации, клиент разделён на представление и состояние, где хранятся подтверждённые данные и предсказания. Такой каркас облегчает добавление режимов (например, кооператив или командные бои) без изменения базовой сетевой схемы.

Практические ограничения связаны с компенсацией лагов и экономией трафика: избыточное предсказание ведёт к заметным корректировкам, слишком редкая рассылка – к потере отзывчивости. Компромисс достигается короткими командами движения, приоритетными дискретными событиями, агрессивной упаковкой дельт и аккуратным сглаживанием расхождений; конфликтные случаи обрабатываются на сервере детерминированно. В качестве дальнейших шагов целесообразны адаптивная компенсация задержек, серверные боты для нагрузочного тестирования и расширенная событийная телеметрия.

В сумме выбранная архитектура показывает, что стандартных средств Java достаточно для стабильной игры реального времени: центральный арбитраж, дельта-снимки и унифицированные типы пакетов с мягкой синхронизацией на клиенте обеспечивают плавный геймплей и сохраняют расширяемость без роста инфраструктурной сложности.

Список литературы:

1. Ищенко Е. А., Борзенкова С. Ю., Баранов А. Н. «Разработка сетевого игрового клиент-серверного приложения на основе клиент-серверной архитектуры». // Информационные технологии и вычислительные системы. 2021.

2. Танатканова А. К. «Построение клиент-серверных приложений». // Системный анализ, управление и обработка информации. 2019.

3. ИС Рямов. «Разработка игрового клиент-серверного приложения». // ВКР или статья. 2024.

4. Баранский И. В. «Технология разработки клиент-серверных приложений на языке Java». // ВКР, 2016.