УДК 004.05

АРХИТЕКТУРА КЛИЕНТ-СЕРВЕРНЫХ ИГРОВЫХ ПРИЛОЖЕНИЙ

Д.В. СУЩЕВСКИЙ
(Представлено: канд. тех. наук И.Б. БУРАЧЁНОК)

Исследована архитектура клиент-серверных игровых приложений и проанализированы подходы, которые помогают избавится от нечестных игроков, а также избежать проблемы с синхронизацией состояния игры на клиентских компьютерах с сервером.

Одной их самых сложных задач клиент-серверных игровых приложений является синхронизация всех игроков с состоянием сервера. Также острым вопросом всегда являлась тема «читеров», так называемых жульничающих игроков, ведь при разработки однопользовательской игры особо нет разницы, если игрок будет жульничать, ведь эти действия повлияют только на него. Но если это многопользовательская игра, нечестный игрок упрощает ее для себя, но ухудшает игровой процесс для других пользователей, таким образом это становится большой проблемой, с которой необходимо бороться, так как такое поведение отталкивает других игроков.

Решением проблем с «читерами» служит авторитарный сервер с наивным клиентом [1]. То есть, вся игровая логика происходит под контролем игрового сервера, а клиенты лишь отображают актуальное состояние, полученное от сервера и отправляют команды в виде нажатых клавиш. С таким подходом, игрок может подменять урон или здоровье, но сервер будет знать актуальное значение и это никак не скажется на игровом процессе других игроков, так как они будут видеть актуальные данные. Также нельзя доверять получаемым позициям игрока, так как он может прислать не верные или не корректные данные. Ситуации с неверными данными легко обработать. Отправленные данные могут содержать информацию, что игрок прошёл сквозь какую-то поверхность, хотя это невозможно или переместился на недостижимое расстояние, но если мы знаем где он находился и его новую позицию, то можно проверить корректность этих данных, а затем предпринять определенные действия в зависимости от их корректности.

Самый простой способ синхронизации, это чтобы клиенты отправляли серверу изменения с определенным интервалом. Обновления содержат пользовательские данные, которые также могут содержать информацию о том, что за прошедший период времени введенных данных не было. Но сразу можно заметить недостаток такого подхода. Например, если интервал составляет 50 мс, сервер получает значение, обрабатывает и возвращает назад результат, но во время ожидания ответа от сервера, у игрока просто застынет игровой процесс. Также общение может происходить с задержкой, из-за скорости сети, где получение или отправка может быть от 50 мс до 500 мс и более, это увеличит время простоя, что просто неприемлемо для современных игр. Это единственная проблема подхода с фиксированным интервалом. Например, есть сервер и два игрока A и B, у игрока A скорость интернета на порядок выше игрока B и при отправлении событий они приходят с различным интервалом, поэтому сервер продолжает расчёт только при получении обновлений от всех клиентов. Отсюда следует, что задержка игры равна задержке игрока с самым медленным интернетом, так как серверу приходится ожидать данных от клиента, чтобы обработать их и отправить обратно результат ожидающим игрокам.

Предсказание на стороне клиента является отличным улучшением подхода с фиксированным интервалом. Этот поход используется при условии, что игра достаточно детерминирована, т.е. результат зависит от предыдущего состояние и не является хаотичным, как например в так называемых «шутерах» или других динамических играх. При таком подходе клиент отправляет серверу свою команду, но локально продолжает эмитировать игровой процесс, так как в большинстве случаев предсказания, которые просчитаны сервером будут совпадать с действительностью. Проблема в том, что игрок видит мир в настоящий момент времени, но из-за задержки обновления, сервер отправляет состояние мира в прошлом, поэтому происходят скачки в предыдущее состояние, что хорошо отражается на позиции персонажа. Но при таком подходе можно не волноваться о нечестных игроках, так как все некорректные данные отображаются только у игрока, который жульничает, но не влияет на общее состояние игры, которое наблюдают другие клиенты [2].

Следующим этапом в совершенствовании архитектуры – согласование с сервером. Решением проблемы с предсказанием, является хранение на клиенте копий всех команд, которые отправлялись до этого на сервер. Теперь при отправке сообщений, у клиента хранится список команд, которые совершались им, когда приходит ответ по обработанной команде, то игрок восстанавливает игровой процесс до этого состояния, удаляет команду из своей очереди и выполняет команды, которые следовали за ней, чтобы вернутся в корректное состояние. Когда приходит последняя команда, клиент удаляет все копии команд до той, которая пришла включительно и применяет последнее состояние, в результате чего, ничего не изменится. Если необработанных команд больше нет, то на этом всё завершится.

Ранее описанный подход хорошо работал для небольшого количества пользователей. Но что, если клиентов много, и они часто отправляют команды? Обновление игры для каждой команды и оповещение всех игроков об изменении состояния основательно нагрузит сервер и сеть. Чтобы исправить эту ситуацию, можно обновлять мир с низкой частотой, например, 20 раз в секунду. Во время обновления, все накопленные команды применяются и меняется состояние игры клиентов на актуальное. Частота обновления мира зависит от динамики игры, если это шахматы, то имеет смысл поставить обновление меньше 20 раз в секунду, если игры является «шутером», то может потребоваться 100 или больше. Клиент будет также, как и в подходе согласования видеть гладкий игровой процесс, вне зависимости от скорости обновления, но клиент редко получает информацию о состоянии других игроков, что приводит к рывкам в геймплее [3]. Есть два самых распространенных способа для борьбы с этой ситуацией:

– экстраполяция;
– интерполяция.

Экстраполяция хорошо подходит для игр, где легко предсказать дальнейшее состояние игры. Например, в игре «Марио» мы знаем с какой скоростью двигается персонаж и что находится перед ним на определенном расстоянии. Следующее положение персонажа зависит от предыдущего, и мы почти наверняка можем предсказать его новое положение в игре. Клиент получает скорость и направление каждого игрока и в течении какого-то времени он не будет получать новых данных. В зависимости от полученных данных, игрок должен эмитировать движения других игроков. Самым простым способом является предположить, что направление и ускорение будут константными между обновлениями и локально воспроизводить физику игры с учетом этих данных, позже, когда придут новые данные, необходимо скорректировать состояние других игроков. Этот метод подойдёт для объектов, которые имеют большую инерцию.

Интерполяция идеально подходит для тех ситуаций, когда скорость и направление меняются быстро. Где необходимо быстро изменять позиции и дальнейшее поведение невозможно предугадать, используя экстраполяцию. Идея состоит в том, что есть позиции игроков за определенным промежуток времени и можно показывать, что происходило между обновлениями, то есть не в настоящий момент, а в прошлом. Игроки видят слегка разное состояние игрового мира, себя игроки видят в настоящий момент времени, но состояние других игроков, является состоянием из прошлого [4].

С интерполяцией могут возникнуть проблемы, если необходима высокая пространственная и временная точность, например, при выстреле. Игрок целится с задержкой и поэтому при выстреле у себя, игрок мог попасть в противника, но в действительности он не попал в цель, так как, то состояние противника, что он видел, было его состояние в прошлом. Здесь может помочь компенсация задержки. Клиент при выстреле, отправляет событие на сервер, с серверным временем состояния, которое отображает момент выстрела и направление оружия. Так как сервер рассылает состояние клиентам, то он может восстановить любой момент из прошлого, в том числе во время выстрела. Таким образом сервер воспроизводит ситуацию выстрела, проверяет, что произошло и с учетом полученного результата, он обновляет состояние мира и рассылает его клиентам. При таком подходе, подобные ситуации будут просчитаны верно. Правда, может возникнуть ситуация, когда игрок прошёл зону поражение, но через доли секунды погиб. Это альтернатива, на которую готовы идти разработчики, потому что лучше уничтожить игрока в прошлом, чем невозможность этого сделать вообще.

Таким образом, в данной статье представлены основные принципы проектирования архитектуры клиент-серверных игровых приложений, а также описанные способы борьбы с нечестными игроками и способы синхронизации, такие как интерполяция и интервальное обновление игрового мира. Детальное изучение представленных в данной статье вопросов позволило исключить ошибки при разработке собственного проекта.

ЛИТЕРАТУРА

1. Unity. Высокоуровневые понятия сетевого взаимодействия [Электронный ресурс] / Unity. – Режим доступа: https://docs.unity3d.com/ru/current/Manual/net-HighLevelOverview.html/. – Дата доступа: 17.09.2018.
2. Сетевое программирование в Source [Электронный ресурс] / Developer Community. – Режим доступа: https://developer.valvesoftware.com/wiki/Source_Multiplayer_Networking:ru/. – Дата доступа: 17.09.2018.
3. Мультиплеер в быстрых играх. Хабр [Электронный ресурс] / Хабрахабр. – Режим доступа: https://habr.com/post/302834/. – Дата доступа: 17.09.2018.
4. Player.IO [Электронный ресурс] / Player.IO. – Режим доступа: http://www.ant-karlov.ru/PlayerIO-interpolyatsiya-ili-udivitelniy-mir-obmana.html/. – Дата доступа: 17.09.2018.