МБОУ ДОД Дворец творчества детей и молодежи
города Ростова-на-Дону
Донская академия наук юных исследователей им. Ю.А. Жданова

Наименование секции/подсекции: информатика/программирование

Исследовательская работа

# Тема: «Сетевая игра World Of Sea Battle»

Автор работы:
Качмар Сергей Александрович 10А класс
МБОУ СОШ №37

Руководитель:
Рогудеева Татьяна Ивановна,
учитель математики и информатики

г. Ростов-на-Дону
2014 г.

## Оглавление

## Введение

Большинство Онлайн игр приносят пользу, давая возможность узнать что-то новое или развивая стратегическое мышление. Детские игры способствуют развитию логики и моторики ребенка. Ролевые игры дают возможность побыть в роли любимого героя или персонажа понравившегося фильма, моделируя действия и поступки своего героя. Также есть командные, которые учат объединяться и находить общий язык с разными людьми. Это приносит пользу, ведь умение вести диалог еще никому не вредило. В виртуальных мирах большое количество различных тематик и жанров; встречаются также и игрушки, несущие в себе юмор. Любые, даже самые лучшие Онлайн игры, как правило, предназначены лишь для того, чтобы дать возможность отдохнуть, расслабиться, отвлечься от повседневных проблем и скоротать свободное время, получив при этом удовольствие. Всегда приятно расслабиться после трудного дня, погрузившись в бесконечные просторы морских пучин или исследовать неизвестные планеты. А любители адреналина в любой момент абсолютно безопасно для своего здоровья могут утолить жажду опасности, соревнуясь с такими же экстрималами.

Данный проект посвящен проектированию архитектуры в онлайн-играх. В последнее время онлайн игры стали набирать популярность, так как интернет-технологии развиваются быстрыми темпами, и разработка подобных проектов стала более доступной.

## Описание проекта

Проект носит название "Игра World of Sea Battle" (в переводе с англ. "Мир морских сражений"). В этой игре предстоит путешествовать на уникальных кораблях, открывать и завоевывать новые территории, участвовать в крупных сражениях с другими игроками.

У каждого игрока есть свой аккаунт, хранящийся на сервере, в который он заходит из клиента игры. Его приветствует сцена входа в игру, изображенная на рисунке 1.


Рисунок 1 – Скриншот программы, отображающий вход в игру.

Затем игрок входит на глобальную карту, изображенную на рисунке 2, на которой находятся острова, порты, нпс и другие игроки.


Рисунок 2 – Скриншот программы, отображающий карту.

Окружающая среда выглядит очень реалистично: смена времени суток и погоды, шум ветра, динамические волны на воде [6]. Все природные явления взаимодействуют с объектами на сцене. Это изображено на рисунках 3 и 4.


Рисунок 3 – Скриншот игры в дневное время.


Рисунок 4 – Скриншот игры в ночное время.

На каждой из восьми карт есть свой порт, в котором игрок может выбрать, оснастить или отремонтировать корабль, поторговаться на рынке. В каждом из них есть свой уникальный ассортимент предложений.

### *Техническое вступление*

Проект World of Sea Battle разрабатывается на языке C# платформы .NET, с использованием Microsoft XNA [5] и Windows Presentation Foundation.

Он состоит из трех частей, каждая из которых имеет свое функциональное предназначение. Это сборка сервера, сборка клиента и сборка единых компонентов. Сборка единых компонентов отличается тем, что библиотеки, из которых она состоит, используются и на серверной, и на клиентской частях. Рассмотрим эти библиотеки подробнее.

Первая из них - это библиотека "Common", в которой находятся игровое ядро, включая все ресурсы, база данных игровых объектов, рабочие классы, контент, который содержит модели и многие другие элементы.

Игровой движок "UWEngine", который начал разрабатываться еще за долго до создания World of Sea Battle, компактно расположился в трех библиотеках: непосредственно UWEngine (все компоненты игровой архитектуры, визуализации, геометрии; функциональная библиотека интерфейса, и т.п.); UWUtilites (набор вспомогательных утилит и компонентов) и UWPhysics (небольшой физический движок).

Библиотека "NetworkTransfer" хранит все необходимое для организации сетевого взаимодействия: начиная от сокетов и ресиверов, заканчивая классами пакетов.

Библиотека "ManualPacketSerialization" будет рассмотрена позже.

Клиентская сборка – это несколько библиотек, которые объединяют и по-своему используют все упомянутые ресурсы. Задача клиента - это отражение того, что происходит на сервере. Поэтому клиент не содержит важных компонентов игровой логики. Он содержит лишь интерфейс, визуализацию и то, что ему понадобится для вспомогательных операций.

Серверная сборка состоит из двух отдельных и ограниченных друг от друга частей: ядро и GUI (интерфейс), изображенных на рисунке 5.

Ядро сервера это важнейший компонент всей игровой системы: именно там происходит выполнение всей игровой логики, обработка состояния, обновление единой удаленной игровой сцены и всех находящихся на ней объектов. Интерфейс нужен лишь для мониторинга и управления работой этого ядра.

### *Описание архитектуры*

Обмен данными в подобных системах происходит следующим образом. Все функции обмена и подключения реализованы сокетами, с использованием надежного TCP-протокола. Во время входа в игру клиент подключается к серверу, отправляя логин и пароль. Сервер добавляет клиента во внутренний список клиентов, привязывает к нему аккаунт и корабль (основной персонаж игры), а затем отправляет информацию обратно клиенту.

Обмен информацией происходит с помощью пакетов. Пакет – это объект, представленный классом языка C#. Он содержит набор полей, описывающих его содержимое. Эти классы пакетов хранятся в выше упомянутой библиотеке "NetworkTarnsfer". Таким образом, они доступны и на сервере, и на клиенте. Для того, чтобы отправить пакет, необходимо совершить целый ряд операций. Прежде всего, необходимо сам этот пакет преобразовать в массив байтов. С этим успешно справляется полуавтоматическая сериализация, которая реализована объектами вышеупомянутой библиотеки "ManualPacketSerialization". Она содержит интерфейс "IMPSerializable", а так же статический класс "Serializator" с некоторыми вспомогательными компонентами. IMPSerializable имеет два

метода: Boxing и Unboxing. Все классы пакетов реализуют эти методы, указывая в каждом из них по порядку поля, которые нужно считать или записать. Класс Serializator содержит два метода для преобразования объекта в массив байтов, и наоборот.

В операциях преобразования огромную роль играет рефлексия - возможность работать с типами и их содержимым удаленно (на уровне кода). Для того, чтобы создать объект, нужно знать его тип. Поэтому все пакеты (в байтовом представлении) имеют заголовок. Уникальность заключается в том, что этот заголовок имеет размер всего в два байта. Перед началом работы формируется общий массив типов из всех пакетов. Именно индекс определенного типа из этого массива и записывается в два байта (16-разрядное целое число). Когда нужно записать пакет, система ищет его тип в этом массиве, получает и записывает индекс. Когда нужно считать тип пакета, система считывает сначала индекс из байт-буфера, а потом получает тип из массива по его индексу.

Данная система с успехом отправляет различные типы данных: структуры, перечисления, строки или ссылочные типы, реализующие "IMPSerializable". Также система способна передавать массивы с этими типами данных.

На следующем шаге вызывается соответствующая функция в сокете [2]. В проекте используются два типа сокетов: серверный и клиентский. Серверный сокет нужен лишь для поиска и подключения новых клиентов. Клиентский сокет используется как в клиенте, так и на сервере. Он содержит функции для получения и отправки данных.

Все вышеупомянутые функции сокетов являются асинхронными [1], а для каждого подключенного клиента выделяется свой поток. Во всех необходимых местах существуют объекты для взаимодействия между потоками. Таким образом, достигается оптимальная производительность и надежность.

Байт-буферы для отправки и приема данных создаются один раз при запуске системы, а при вызове операций чтения/записи просто перезаписываются без создания нового экземпляра.

### *Синхронизация объектов*

Для данной игры необходимо проводить синхронизацию объектов на игровой сцене. Этот вопрос достаточно сложный и до сих пор не разработано его универсальное решение. Однако, разработаны многие способы для реализации синхронизации [3], [4]. Рассмотрим два способа, которые оказались наиболее подходящими для данного проекта. Представим, что существует некоторый объект (например, НПС), который начал движение, а затем через некоторое время остановился. Необходимо уведомить клиентов об этом.

Первый способ - это просто отправить клиентам пакеты, когда НПС начинает движение или останавливается. Однако, из-за того, что каждый раз

пакет доходит от сервера до клиента за разное время, могут возникать определенные проблемы при визуализации.

Для реализации следующих способов, нужно точно знать это время или просто пинг. В данной работе для измерения пинга используется следующая схема: через определенный временной интервал сервер отправляет клиентам специальный пакет, на который клиент отвечает. Сервер замеряет время, прошедшее с момента отправки этого пакета до момента получения ответа, и делит его пополам. Зная пинг, мы можем воспользоваться следующими вариантами синхронизации.

Первый из них - асинхронная синхронизация с фиксированной ступенью. Данный вариант стремиться к тому, чтобы событие произошло одновременно на всех клиентах. Для этого вводится дополнительный показатель - максимальное время. Таким образом, сервер создает таймер с этим значением, отправляет пакет клиентам и начинает отсчет. Как только клиент получает пакет, он не выполняет изменения, а тоже начинает ожидание, но время ожидания уменьшается на время пинга. Когда на всех хостах таймер доходит до конца, событие применяется, т.е. НПС начинает движение или останавливается. Недостатки очевидны: во-первых, большинство клиентов с малым пингом будут тратить лишнее время на ожидание, а во вторых, если пинг будет больше максимального времени, то произойдет рассинхронизация, т.к. событие произойдет позже, чем нужно.

Второй - моделируемая синхронизация. Данный вариант стремится к тому, чтобы событие происходило на всех клиентах не одновременно, а как можно быстрее. Для реализации моделирования нужно знать, что за время X объект, который начал движение, пройдет путь Y. Соответственно, используя формулу, учитывающую поведение объекта, можно предсказать, в какой позиции будет находиться объект через время Z. В пакет, который отправляется при начале движения или остановке, нужно добавить дополнительное поле: позицию. Она устанавливается как позиция, которую будет иметь объект в тот момент, когда пакет дойдет до конкретного клиента. Таким образом, данная система является наиболее быстродействующей и обладает возможностью самокорректировки. Поэтому она используется в данном проекте.

## Заключение

В результате выполнения этой работы была создана игра под названием „World of Sea Battle”. В проекте разработана быстродействующая и гибкая сетевая архитектура. В работе было реализовано программное средство, которое выполняет следующие функции: обмен данными между клиентами и сервером, преобразование объектов C# в массив байтов, синхронизация объектов в режиме онлайн. Используются такие графические возможности, как прорисовка динамической жидкости, модели кораблей, состоящие из подвижных элементов, сглаживание, shadow mapping (карты теней) и т.д. Были изучены такие средства программирования, как паттерны

проектирования, рефлексия, сокеты, методы расширения, небезопасный код и многие другие возможности языка C#.

Создание различных игр привлекает внимание пользователей, и формируют интерес к изучению языков программирования, а так же развивает математическое мышление, так как в данной работе использовалось множество математических, а особенно геометрических расчетов.

Подобные игры пользуются большой популярностью, а эта игра способствует развитию стратегического и логического мышления у пользователей. Данный проект планируется использовать в коммерческих целях.

## Используемая литература


1. http://msdn.microsoft.com/ru-ru/library/fx6588te(v=vs.110).aspx - асинхронный сокет
2. http://round-angle-net.blogspot.ru/2010/09/tcpip-c.html - пример использования сокета
3. http://www.ant-karlov.ru/PlayerIO-reshenie-problemi-zaderzhek.html - синхронизация
4. http://enepomnyaschih.livejournal.com/1610.html - синхронизация
5. http://unmail22.narod.ru/Books/3D_XNA4/Content.html - книга по Microsoft XNA
6. http://www.gamedev.ru/code/articles/HLSL - HLSL шейдеры