MULTIPLAYER DISTRIBUTED GAME

Coursework report for AG1107A – Network Game Development, University of Abertay, Dundee

Richa Sachdeva
M.Sc. Computer Games Technology
Student Id. – 1304869

21st May 2014

Introduction

Online games enable player across geographically different locations to play games together, this in part is made possible due to internet which is used to share game data among players. For an application distributed over internet with multiple players spread across different locations, sending game information from one player to another is possible due to networking, and such games which incorporate networking features are known as networked games. Networking in games is quite old, but due to the volatile nature of the internet some problems are persistent and main considerations while designing a networked games are -

1. The view presented to several players across internet needs to be consistent.
2. Each player has a local copy of the game, which is different for all, so to make game view consistent, the local copy needs to be updated periodically as per the server's copy, which is the master copy, and is used to provide consistent game view.
3. How to handle the processing of game in case data packets been sent are lost, or connection is interrupted.

The coursework associated with the module 'Network Game Development' required to construct a distributed application that synchronises the position of objects over a network connection, using dead reckoning to deal with latency.

For this, a two player distributed game was developed in which players can shoot at each other. The two players are represented by the server and the client. The server player is controlled algorithmically and fires at regular intervals. The client player is controlled by the player/user and can be moved left, right using the left/right arrow keys, stopped using the down arrow key, and can fire using the spacebar. In the center of the playing space in the game, coins are present, which blocks the fire from the server, and client needs to collect these coins and if client is able to collect all the coins, client wins. As it's a firing game, so the player dies if the opponent's bullet touches the player.

Rest of the report discusses the system architecture used, protocol designs used, algorithm used for position estimation, and code structure implemented.

System architecture

Architecture is the backbone of any system and it's an important factor in deciding how efficient the system will be in terms of its robustness to failure and efficiency in delivering the data and providing a uniform view.

In a client server architecture, server controls the flow of data in way that each client first sends the data to the server, from where the server will send the data back to all the clients. The problem with this approach is for multiple clients it doesn't scale well due to the latency involved, as server needs to first receive messages from all the clients, what if some clients leave, and second it has to send the data back to all the clients, what if server disconnects. So, scaling is an issue in such systems owing to high dependency on the server.

Another approach is hybrid client server, in which server is chosen from one of the players. It is similar to the client server plus it scales well, as server is chosen from one of the players, so in cases if server disconnects, another player is chosen to be the server. Also latency is less in this case, as first a limit is put on player server ratio, which results in the establishment of a smaller network amongst the players and the server. To further reduce the effects of latency, server sends messages after a set interval to sync the state of the game across different players. Sure, it does mean that for some players game state will go back in time, but due to periodic messages been sent, the difference isn't much and in most cases won't be visible. Thus, considering above factors, hybrid client server architecture is used in the current game.

Protocol design

1. To send data packets from one computer to another over internet, set of rules are established. The set of rules are known as protocols which set standards and rules for data communication. Two commonly used protocols are TCP/IP and UDP. In the current case, TCP/IP is used. TCP is a connection oriented, reliable and ordered protocol, which essentially means that TCP establishes the connection and waits for the acknowledgement from the other end. Until acknowledgement isn't received, connection establishment isn't complete. It is reliable and ordered which means that all the data that's been sent will reach the destination in the same order, and in cases where packets are dropped, they're resent. While sending packets again isn't desirable in case of real time games as resending packets would be the main reason for lag, or inconsistent behavior, but still TCP is chosen as, TCP is connection oriented, which means that there will be acknowledgement for the connection and TCP will send messages only when the client has connected, and will keep track of the connection, unlike UDP in which no

connection acknowledgement is received, so it is difficult to keep track when a client leaves, which results in sending data when there is no client, wasting bandwidth and other resources. Also, UDP isn't reliable and provides no mechanism in cases when the packets are been dropped by the network due to congestion, or any other reason, which contributes to the inconsistent behavior, so it's better to chose TCP which resend the messages and then the game can deal with duplicate values, rather than not getting any data. Another reason for selecting TCP was the amount of data that's been sent. In the current game, player position, speed (for both client and server), bullets fired, server time were the data elements that's communicated over the network. So, as data involved is less, TCP was a better choice.

2. Sockets represent endpoint of communication in a network and are required to transmit messages from one point to another. By default sockets are in blocking mode, which means that sockets will wait for data/event and will retain the control of the program till the operation isn't complete. In non blocking mode, socket will wait for the event or data as the case is, but the control will be returned to the main block so that other processes aren't kept waiting. In the game, non blocking sockets are used, as there is no point in keeping other process to wait till the network fetches the data.

3. Event based programming is one in which based on the occurrence/ non occurrence of the events, program flow is decided. That is, based on whether the client moves the player right or left by pressing the respective arrow keys, player will move, else it will not. In this, whenever an event has occurred, it will inform the program about it. Or the other way to say it is that main game loop keeps on listening for these events and performs as and when a particular event occurs. For the networking in the game, event based approach is used, in which when there is data to be sent/read, it is sent/read, else the game continues as per the last sent/read message. Also, as the sockets are non blocking, so they don't interrupt the program flow or take away the control from the main loop, when there is data to be sent/receive, respective task is performed, else it keeps on listening. Advantage of the event based approach is that control is with the main loop and no components are kept waiting, so the execution is fast.

4. For networking, timing of the events is essential, for which a timestamp, which is the server time has been sent to all of the clients, so that the clients can synchronise their clocks with respect to the server, keeping latency in check. Also, apart from regular networking messages, server also sends a periodic message to all the clients, so that the clients can update the current game state as per the server's game state, minimizing the differences, which help in reducing the lag.

Position estimation algorithm

Algorithm to estimate the current position of the client and the server, or the players in a networked game is essential in order to give a smooth, consistent view without any jitters, or delay in case data arrives late, and in general to deal with the main problem associated with the networks, that is of lag.

Lag or latency, is the noticeable delay between the action of client and the reaction of the server (Wiki). To deal with lag, position estimation algorithms are used in which based on the current position, next position is estimated. Dead Reckoning is prediction technique which is used to estimate the next position based on the current position. In this, as the movement of the player is restricted in single dimension, player being able to move only along x axis, a linear model is used, in which the position is estimated based on the past position and velocity, where if player is moving in one direction it continues doing so till either the end of the playing game screen is reached or a new message has been brought by the network. Thus, the player uses the data from the last message and the linear model to deduce the position.

Initially, only the velocity values were sent over the network and based on it the position was estimated, but as the network lag was increasing with time, both the position and speed values were sent to reduce the inconsistency, which helps in reducing the lag.

To further reduce the lag, server sends two types of messages. One is normal which goes once per game loop, which tells the client about the current position, second message is sent after 100ms, in which client updates itself and brought the game state to be at level of server. The parameters sent are more or less similar, but the parameters updated are different. In normal game loop, client updates the server based on it's current position, sent by the network (in case, network sends no data or incorrect data, server position is predicted using linear model). When after 100ms, client updates server speed so that it's the most recent value. As server is also a player, so same process is repeated at server end, where server updates the client position with each game loop, and after 100ms, updates the client speed with the most recent one. To synchronise the time clock between the server and the client, server sends timestamp with each data. The client receives the timestamp, compare it with it's internal clock, calculate the difference and actions are taken based on the time with respect to the server.

Another important point is that the periodic update messages are sent to take care of jerky movement of the player across networks due to lag, and constant updation of the client and server code. By sending the data periodically, the inconsistencies are removed within smaller interval and are "almost" invisible to the player. In network, several different factors can contribute towards latency, and it can not be totally avoided but the effects can be shielded from the player and an almost consistent movement can be achieved.

Code structure

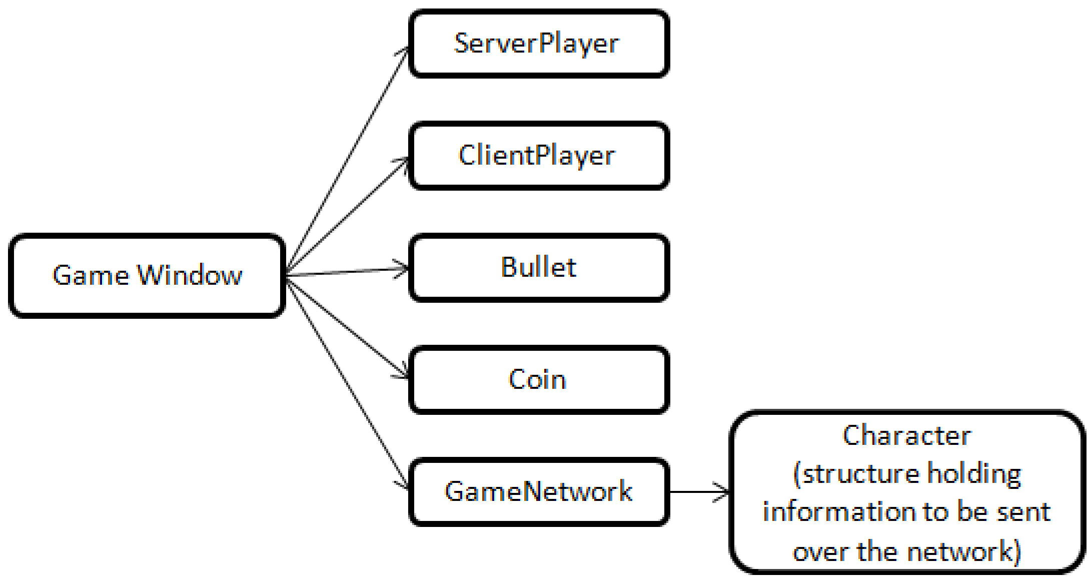
Game is developed using C++ and Sfml (Simple and fast media library). Sfml is a 2d C++ game library using which the graphical elements viz the players, coins, and bullets are rendered on the screen. Sfml uses OpenGl to render the graphics. The current library version is 2.1 which is used in the game. It also provides support for networking by implementing Socket class, which is the base class and defines the basic operations like create, close, set sockets as blocking or non blocking. TcpSocket class extends from the Socket class and has been used in the game for implementing Tcp Sockets. Sfml also provides a Packet class, which provides a safe and easy way to serialize data while sending it over the network using sockets. A structure specific to the game, holding server position, speed, time, bullets fired, client's position, speed, bullets fired, coins collected, score and whether game is running or not is constructed. Using packets, it is possible to overload the operators to handle the custom types (like the structure, constructed above).

The main goal was not to develop a 2d game, but a distributed application, for which networking is the core feature, but still it's a feature of the application and not the application on the whole, so it needs to be developed keeping it in mind, ensuring that networking code is contained in a separate class, from which it's called, instead of networking code being mixed with the main game loop. And that's why a lot of emphasis was given on keeping the code modular, where data to one aspect of game, say player is contained in player class. Likewise, networking code is present in Networking class, and a Networking object is present in the main game class, from where the networking functionality is accessed. This also ensures that networking class is reusable in any other different application with minimum changes.

As C++ is object oriented, so the application developed makes full use of object oriented features like classes, inheritance, polymorphism, and overloading. Every object in the game has been given a separate class file as it helps in better organization and keeps code modular. The objects present in the game are ServerPlayer, ClientPlayer, Coins, Bullet, Networking. From the main function, control goes to the GameWindow class, which manages the different game stages, store objects from above mentioned classes and manages the interaction between all of them.

When the application starts, a connection is established between the server and the client. Only after both parties receive the confirmation for the connection from the other player, game starts. From the time connection is established and game starts, there is gap of 2 seconds, in which both the server and client initialises the game state and gap is there to ensure when the game starts, both are at same state. Then control goes to GameWindow class, which initialises the players and game enters in the main game loop, where the control lives till the end. Thus, in a way it can be said that GameWindow is the starting point of the game.

Class structure -


ServerPlayer
ClientPlayer
Game Window
Bullet
Coin
GameNetwork
Character (structure holding information to be sent over the network)

ServerPlayer class stores information regarding the server player, which is rhombus shaped, positioned at the top. It's class consists of methods to initialise the player using CircleShape class (present in Sfml). CircleShape class is used to draw different shapes. ServerPlayer class initialises, draw, and moves the player from one end of the screen to another. ClientPlayer class is same as ServerPlayer, but the client is circular in shape and is positioned at the bottom of the screen. The ClientPlayer also has methods to move the player as the player presses the arrow keys. The two classes are similar in behavior, and ideally they should inherit from an abstract Player class, but that hasn't been done in current game, as even though the behavior is similar, purpose is different, so are the controls with server class being controlled algorithmically and player controlled by player's movements.

Bullet class draws, and moves bullets on the screen. Coin class is responsible for the behavior of the coins class. By using different classes for each of the above, it was easy to manage the behavior of objects.

GameNetwork class is responsible for providing the network features in the class. It consists of the structure that holds all the information that is communicated between the client and the server. It also consists of Tcp Socket which is required to establish a connection and then send and receive messages. The class consists of methods to establish a connection, send messages, receive messages and terminate the connection, when the game ends.

The interaction between the players and other game objects is handled by the GameWindow class. Interaction is of two types - 1. Player controls how the player would move and when to fire. 2. Interaction between the game objects. Bullet fired from server is blocked by the coins, but client get a point for the same and bullets from both kill each other. Collision detection is employed for the same, in which a bounding box is constructed for the two objects, say a bullet and a coin and the distance between them is calculated, if it's less than their size, then collision has occurred, else not.

Result

The networked game was developed with minimal effect from latency. The client has smooth convincing behavior, where it's speed and position is synchronised and bullets are fired with 'almost' no lag, or within the acceptable lag limits. In the server's case, firing was at the approximate position, but the movement was bit delayed, so the server player was modified to deal with the inconsistencies and to ensure a smooth, synchronised movement. The position from where the bullets are fired was synchronised, but as there speed's were not synchronised (they were the same value) so the bullets move with different speed. The coins collected by the player, and the information of the player dying, to end the game, this information was sent within the lag limits, so overall the application developed worked successfully in a network. Also, as the network class was developed as a standalone class, it can be reused in another application with little modifications.

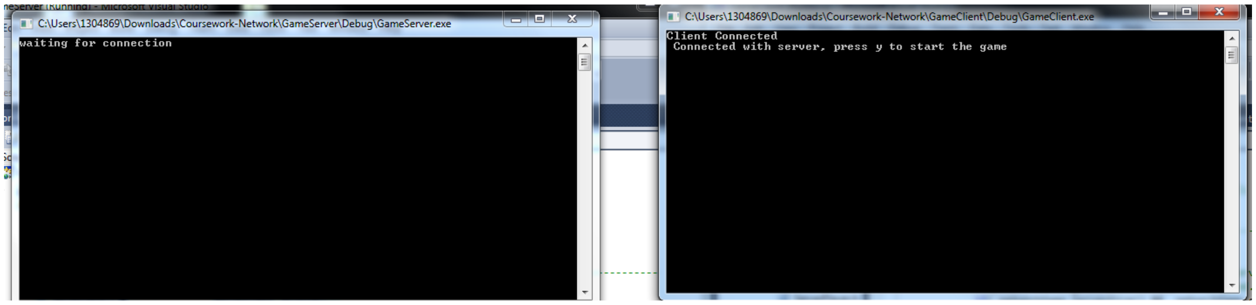
Dealing with the network problems was challenging and informative, as first it was fun and exciting to develop a networked game, second by encountering and trying to devise a way to resolve the problems one understands the working of internet in a better way. The problem of lag is there from the start of the internet, it still persists, and maybe it will remain for a long time, as networks are of volatile nature, with packets being dropped and network congestion. Thus, to develop distributed application, one has to not only understand the working of internet, but also to find ways to minimize it and provide a consistent, smooth view for the user. The coursework so developed has been immensely useful in providing in-depth understanding of the network architecture, how the packet transmission worked and how to resolve the network issues to synchronise various game objects.

References

- Sockets (Online) *Blocking vs Non Blocking Sockets* [online available at:] http://www.scottklement.com/rpg/socktut/nonblocking.html [last accessed: 14th May '14]
- SFML (Online) *Simple and Fast Multimedia Library* [online available at:] http://www.tikkle.in/2014/05/alia-bhatt-memes-trending-on-twitter/ [last accessed: 16th May '14]
- Fortuna, H. (2014) *AG1107A Lecture Notes,* University of Abertay, Dundee

Screenshots of the game

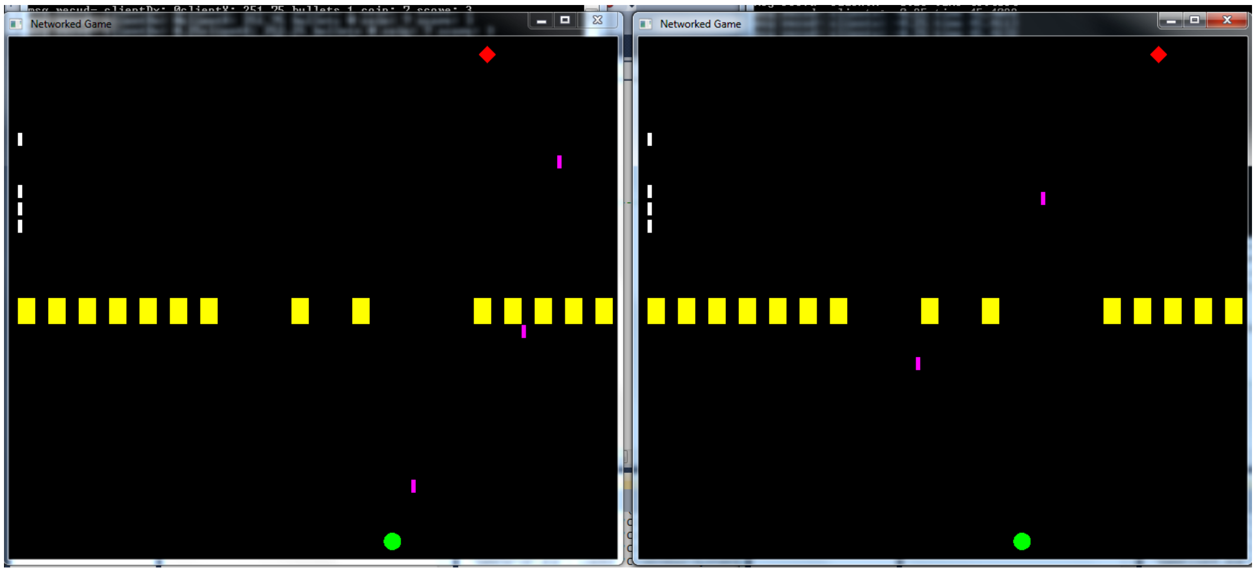
When both the Server and Client starts


C:\Users\1304869\Downloads\Coursework-Network\GameServer\Debug\GameServer.exe
C:\Users\1304869\Downloads\Coursework-Network\GameClient\Debug\GameClient.exe

```
waiting for connection
```

```
Client Connected
 Connected with server, press y to start the game
```

Game in progress


Networked Game
Networked Game