DESIGN AND DEVELOPMENT OF A PEER-TO-PEER ONLINE MULTIPLAYER GAME USING DIRECTX AND C#

Loh Sau Meng, Kelvin, Edmond C. Prakash and Peter K. K. Loh

School of Computer Engineering
Nanyang Technological University, Singapore
Contact email: Asprakash@ntu.edu.sg

ABSTRACT

This paper highlights the accomplishments in designing a peer-to-peer online multiplayer game. The design of a turn-based fantasy game in terms of game design: game-balancing, non-linearity, artificial intelligence, level design and overall game-play; software design: object-oriented analysis, class modeling, sprite design and animation. The development of the game using C# and DirectX APIs with emphasis on DirectPlay for networking to achieve peer-to-peer playability in a .Net environment.

1. INTRODUCTION

Computer game programming has been evolving ever since the dawn of the software development. Playing against real human opponents is simply more challenging and motivating [1, 14]. However, a common problem for developers is the maintenance of an effective network of entities and system, given the low bandwidth, high latency infrastructure we have today. Furthermore, even with many promising technologies such as Digital Subscriber Line (DSL) or Cable which can provide up to 2 Mbps download and 256 Kbps upload bandwidth, most home users are still limited to 56Kbps voice modems [3]. Hence, distributed simulation can only be achieved with efficient communication synchronization between players, system and distributed databases so that the global state of the game is replicated accurately in every end-system [9]. But with the clever technologies that are currently available from various Internet technology developers can now start a trend of media-rich multiplayer gaming easily [6-8,10-12]. The project is split into two sub-parts, with each sub-project being reused to achieve software reusability.

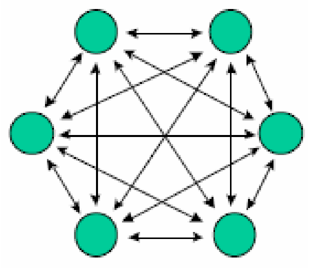


Figure 1: Peer-to-Peer

1.1 Sub-Project One – "BattleGround"

A turn-based, strategic, 2D fantasy game where the player is presented a variety of moves for maneuvering his character to a finishing flag across a board filled with obstacles. The moves are executed when the player confirms its selection or when the timer runs out.

1.2 Sub-Project Two – "BattleGround Online"

An extension of the previous sub-project, it is installed locally but runs only after the user has logged in to the remote server. Multiplayer games are organized by the remote server using DirectPlay to provide peer-to-peer session to host these games. Fitted with chatting functions, users can also decide from the game menu to play either a single player game or host/join a multiplayer game. Unlike single player game, the objective of the multiplayer game is to be the sole survivor. During game play, all the players must select their moves at the same time within the same time limit. All their moves will be executed at the same time with the priority given to the player that confirms the moves the earliest.

2. GAME DESIGN

The design reflects the combination of both the sub-projects after integration. The initial design of the game engine is adapted from an existing Java real-time strategic game engine. The base classes are built generically to fit with any game while reusability is maintained so that the same classes can be used by future updates.

For the graphical implementation of this project, we adapted the use of a C#-DirectX-Visual Basic wrapper library known as DxVBLib.dll that uses DirectX 7.0 interfaces, enabling a sense of backward compatibility to the game. This implementation came from two open-source C#-DirectX games we discovered in the "C# Corner" forum. Since the project emphasis is on networking [2,4], optimal graphical presentation is sufficient, advanced graphics found in the later versions of DirectX was not required.

Sprite class: Anything that is shown onscreen is a BitmapObject (an image). Hence, Sprite is modeled as a sub class of BitmapObject. From the above analysis, it is also obvious that Sprite will be the parent for many other classes (see Figure 2) in the game, so it has to be

0-7803-8560-8/04/$20.00©2004IEEE

designed generically to provide all the common functionalities.

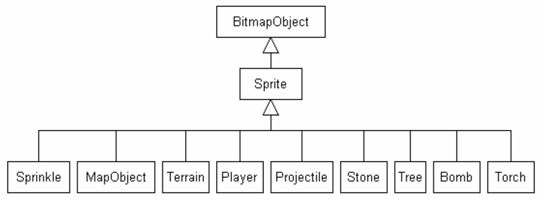


Figure 2: Sprite Class Generalization Diagram

States: A sprite has five states: Attacking, Attacked, Dying, Idle and Walking. Changes to the character's attributes happen instantaneously but the animation does not. A sprite stays in each state until the animation sequence has ended (except the idle state which draws the first image of the Attacked sequence). Not every sprite succumbs to this rule since they may require only one image.

Player Class: The second most popular class is the player ("character") class that represents the actual player in the game board. Players are sprites with more special traits. A player would generally possess 16 animation sequences (4 directions x 4 actions) for the Attacking, Attacked, Dying and Walking states. Idle state uses the first image for each direction from the Attacked sequences.

A player contains a list that stores all the possible items a player captured in the game. With the ability to store items, a player can also use these items. In the game, every different character is a subclass of Player. Each bears unique characteristics in terms of appearance (image), soulstones usage effects and attacking/attacked procedure (e.g. a rat should not behave like the wolf, a wise mage would have stronger magical capabilities compared to a barbarian, an archer would not attack another sprite directly but with an arrow compared to a swordsman). Based on the list of items, a character can use an item to perform "magical spells" that would give the character an advantage (e.g. the blue soulstone is commonly used for healing, increasing health points while the red soulstone is for conjuring a bomb).

Board Class: A board is a virtual playing field where sprites are congregated to portray a realistic battleground. Serves as the sprite manager, it ensures that all the sprites have a fair share of the graphic display time slice. During initialization, it populates itself with four lists of sprites. The Mapvector list contains all the dormant (terrain) sprites while the Spritevector list contains all the active sprites. During initialization, a simple terrain management scheme is used to deicide how the battlefield should look. A priority scale is used for comparison between different terrain types. The game is loaded when the board is initialized with a board data file that is formatted using XML schema, containing the details of the battlefield.

BattleGround Class: The Window application that holds the entire game is BattleGround class. It handles the overall layout of the game from the GUI down to the game-play sequence. An infinite loop is used to run though these states. Some states (like the loading states) are run only once to load the necessary resources while the rest are ran repeatedly to render images on the screen all the time.

Game Play: A single player game differs from a multiplayer game in terms of game-play. In the single player game, the game is completed once the player reaches or passes by a finishing point but in a multiplayer game, the winner is the last survivor.

To begin the single player game, a quest is chosen. Each quest varies in difficulties depending on the complexity of the puzzle. The layout of the battleground is specifically designed to make the game player think. The elements on the field and the NPC actions can be anticipated thus solving the problem should be simple enough. But the real challenge is to beat the expert par provided for each Quest; only by matching or beating the par does one deserve the win.
Six moves must be selected for each turn and the selection ends when the confirmation button is clicked within the given time limit. If the time limit has been exceeded, for each unselected moves, a one heath point penalty is inflicted on the player. The confirmation button turns yellow in color as a warning to indicate that time is running. After selection, the game will collect all six moves in a list known as player_actions. Unselected moves will have a penalizing action inserted as a replacement. The actions are stored as a string with the following format: *Object name, action*

An additional_actions list is also created to store all the actions performed by the NPCs and Traps. The player_actions list is then combined with the additional_actions list to form the compiled_actions list which is then sorted according to an action scheme and inserted into a board_action list. In the list, an integer is appended to the list for every move, acting as a separator between moves for the entire turn. When the board_action list is completed, every action is executed sequentially according to the order of the list. Each action is executed only when the board's update() reflects that there is no sprite in motion or animation. Different actions are given different priorities to be executed earlier or later than others. The list is also appended occasionally with new actions during the action execution because some actions create more actions. E.g. "Bomb, explode" is appended to the list when Player successfully used the red soulstone.

The move selection interface becomes complicated when the player could collect items. When a player collects an item in a move, the possession of the item must be reflected on the interface. When the player selects the item for a move, the chosen item is removed from view and selection. When the player undoes its selection, the item is return to sight for selection again.

The game is over when a player's health points and lives are reduced to zero. The user can choose to replay or try another Quest.

DirectPlay: The rise of the Peer-to-Peer phenomenon opens up new grounds for exploration as it provides an alternate path toward the building of a multiplayer game. To embrace this new technology, BattleGround utilizes DirectX's Directplay API as the backbone for networking.

P2P (acronym for Peer-To-Peer), known as a form of communication architecture, that allows messages to be relayed between players or sets of receiving players without any server, is for message passing or game management. Unlike conventional client-server topology, communication and calculations are performed solely by the clients, hence removing the bottlenecks associated with servers; messages can be delivered twice as fast compared to client-server [1].

However, a lobby server is still required to perform match-making services for players to log on. The lobby is a virtual environment where players can chat and exchange information. Players can create their own game sessions or view/join others' game sessions.

Managed DirectX libraries are used which makes the multiplayer mode accessible only to those who possesses DirectX 9.0. DirectPlay uses TCP/IP connections as channels to deliver the messages. It provided easy to use functions based on the original C++ interfaces that greatly simplified the tasks for interfacing with COM objects and middleware.

The design of the multiplayer game is split into two categories: the game session host and the game session client. After logging on to the lobby, a player can choose to create a game session or view/join another.

Multiplayer Game Design: The multiplayer game design has 22 states of which two states are reserved for connection error handling. The initial states are designed to establish network connection with the lobby and the various game players. After detecting the remote database and retrieving the address of the lobby server (state 0), the user can select which lobby server to join. An enumeration of hosts will be conducted on the lobby server's address at state 2 and the user will be required to log in at state 20 (the logging module was added in much later). The user can choose to log in as a registered user or as a guest. In either case, the user will be brought to state 2 whereby a connection is established with the lobby server and proceed to state 3. At state 3, the user is presented with the lobby and the options to chat with other players, create a new game session or view/join others' game sessions.

Game Host: When a user chooses to create a game session, they proceed to state 4 by clicking on the "Create" button in the lobby. The user will choose from an array of battlefields (boards) for the game to begin. Then the user is brought to state 5 to allocate port 3167 to host their game session and then inform everyone about the new game session through broadcasting.

Game Client: To join a game, the player must select a game session in the lobby to view. The player will then be brought to state 15 to retrieve the game information and display a preview of the game at state 16. If the player is satisfied, it can choose to join or return to the lobby. Joining will put the player into state 17 where it will wait for the game host to start.

Scoring: The score of the player is computed according to the following algorithm shown in Fig. 3.
:

```
If (there are two players i.e. a duel)    {
        // Let NA and NB be the scores of player A and player B.
        if (the winner is A) {
            if (NA - NB > 625) result= 0; // A is too strong
            else result= (25 - ((NA-NB)/25));
        }
        else result= -(25 - ((NA-NB)/25));          // loser
}
else if (more than 2 players) {
    // Let NA be the score of player A and AVG be the average
score of all players.
      If (winner is A) {
            if (NA-AVG > 625) result= 0;          // A is too strong
            else result= (25 - ((NA-AVG)/25));
      }
      else result = -losers; // return the no. of players killed
}
```

Figure 3: Scoring Algorithm

The objective of this scoring scheme is to ensure fairness and equality between the players' challenge. A high score player should not earn more points from a win against a weak player. On contrary, a weak player will earn high points by defeating a high scoring player.

3. IMPLEMENTATION

Sub-project #1 covered basic game-playing features such as moving sprites from point to point. Improvements and adaptations are made according to the requirements. A text version lacks the complexity of a graphical version so development was much easier. However, when DirectX was used to develop the graphical module, problems ensued.

4. CONCLUSION

The first version has been completed (snapshot shown in Fig. 4), but for the next version of game there are always improvements that can be made. The installation requirement of having Microsoft .Net Framework 1.1 and DirectX 9.0b is cumbersome since these components are not pre-installed and have to be installed manually by the end-users themselves and not through the game.

5. REFERENCES

[1] M. Sabadello, (2001, April/May). *Small group multiplayer games*. Vienna University of Technology, Austria. http://www.cg.tuwien.ac.at/courses/Seminar/SS2001/multiplayer/small_group_multiplayer.pdf

[2] Remotesoft, (2002, 15 June). *FAQs for Remotesoft .NET Obfuscator*, http://www.remotesoft.com/salamander/obfuscator/obfuscator_faq.html

[3] E. Serin, (2003, March). *Design And Test Of The Cross-Format Schema Protocol (Xfsp) For Networked Virtual Environments*, Naval Postgraduate School,Monterey, California. http://theses.nps.navy.mil/03Mar_Serin.pdf

[4] L. O'Brien, (1997, March). *The Game Network API Slalom*, Game Developer. http://nasla.yonsei.ac.kr/~neurok/link_docs/OnlineGame/199703_TheGameNetworkAPISlalorn.pdf

[5] Microsoft.com, (2004, 15 April). *Developing Multiplayer Add-Ons for Flight Simulator 2000* http://www.microsoft.com/games/flightsimulator/inc/fs2000/sdk/fs2000_multiplayer_sdk.pdf

[6] Intel Corporation, Developer Relations Group (1997). *Multiplayer Internet Gaming,* Developer Relations Group, Hybrid Application Cookbooks. http://www.intel.com/business/bss/industry/media/games.pdf

[7] A. Istvan, B. Zoltan, C. Hassan, et al., (2004, 15 April). *Challenging .NET as a multiplayer game platform – minerArena game,* http://minerarena.aut.bme.hu/Download/minerArena.pdf

[8] J. Sanneblad and L. E. Holmquist, (2003*).* *OpenTrek: A Platform for Developing Interactive Networked Games on Mobile Devices*, Future Applications Lab Viktoria Institute, Göteborg, Sweden. http://www.viktoria.se/fal/publications/2003/mobilehci2003-opentrek.pdf

[9] M.Joao Monteiro, J. Pereira, L. Rodrigues, (2002). *Integration of Flight Simulator 2002 with an epidemic multicast protocol*, Proc. of International Workshop on Large-Scale Group Communication, Florence, Italy. http://www.di.fc.ul.pt/~ler/reports/Srds03Workshop.pdf

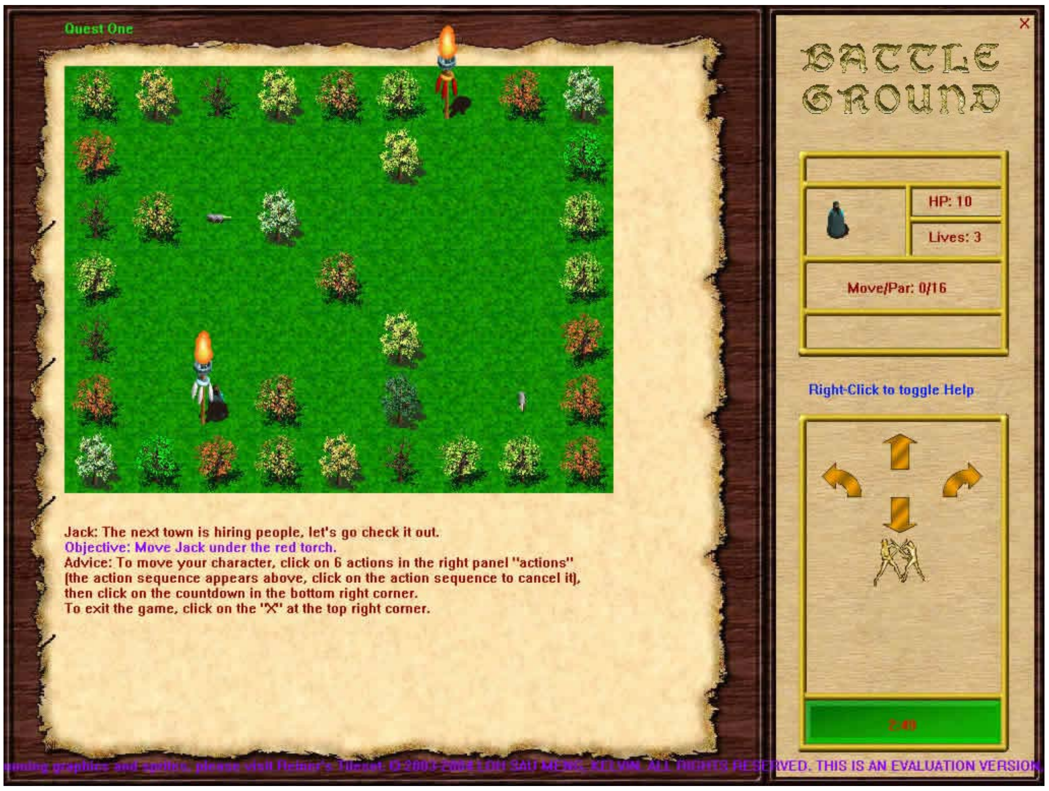
[10] T. S. Leithead, (2003, 21 April). *User Datagram Protocol: A Real-time Multicast Gaming System*, CS 460: Computer Communications Final Report. http://students.cs.byu.edu/~tleithea/school/papers/TechPaper-UDP_A_Real-time_Multicast_Gaming_System_28_10_2003.pdf

[11] P. Bettner, M. Terrano (2001, 22 March). *1500 Archers on a 28.8: Network Programming in Ageof Empires and Beyond*, Game Developer's Conference 2001, http://zoo.cs.yale.edu/classes/cs538/readings/papers/terrano_1500arch.pdf

[12] K. Grönlund, (2004). *Gamenet, a Game Network Library*, Dept. of Computing Science, Umeå University. http://www.cs.umu.se/~c99kgd/exjobb/thesis.20040121.pdf

[13] L. Pantel, L. C. Wolf (2002, 12 May). *On the Impact of Delay in Real-Time Multiplayer Games*, in Proc. of the 12th International Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV 2002), Miami Beach, Florida, USA, 2002.

[14] Dan Kegel, (1999, 17 July). *NAT and Peer-to-peer networking*, Caltech Alumni Association, 1200 East California Blvd., Pasadena, CA 91125. http://www.alumni.caltech.edu/~dank/peer-nat.html



Jack: The next town is hiring people, let's go check it out.
Objective: Move Jack under the red torch.
Advice: To move your character, click on 6 actions in the right panel "actions"
[the action sequence appears above, click on the action sequence to cancel it],
then click on the countdown in the bottom right corner.
To exit the game, click on the "X" at the top right corner.

[illegible] VED. THIS IS AN EVALUATION VERSION

Figure 4: Snapshot of the Battle Ground game