DESIGN AND IMPLEMENTATION OF A MULTIPLAYER THIRD-PERSON SHOOTER GAME USING UNITY3D

Premsai,Sriram,Abhishek and Chandrasekhar

Department of Computer Science

Indian Institute of Technology,Kharagpur

ABSTRACT

The main idea behind this paper is to take advantage of game implementation to learn several principles of distributed systems. This approach represents the other end of the usual meaning of edutainment where the educational part is building of a game, instead of playing the game. This paper shows how several main principles of distributed systems have a relation with the distributed games. The entire gaming environment has been designed utilizing the Unity 3D video game engine. We have developed a 3D environment of the physical plan in the format of a networked multi player third person shooter. 3D gaming technology provides tools to create an environment that is both visually familiar to the player as well display immense amounts of system data in a meaningful and easy to absorb format.

INTRODUCTION

Networked games are rapidly evolving from small 4-8 person, one-time play games to large-scale games involving thousands of participants and persistent game worlds. However, like most Internet applications, current networked games are centralized. Players send control messages to a central server and the server sends state update to all other active players. This design suffers from robustness and scalability problems of the single server designs. For example, complex game-play and AI computation prevent even well provisioned servers from supporting more than several tens of players for third person shooter games. Further, client-server game designs often force players to rely on infrastructure provided by the game manufacturers. These infrastructures are sometimes not well provisioned or long-lived; thus, they either provide poor performance or prevent users from playing their game long after their purchase.

A distributed design can potentially address the above shortcomings. However, architecting distributed applications is difficult. The most fundamental challenge in distributing an application is in partitioning the application’s state (e.g., the game world state) and execution (e.g., the logic to simulate player and game AI actions) among the participating nodes. Distributing a networked game is made even more difficult by the performance demands of the real-time game-play. In addition, since the game-play of an individual player translates to updates to the shared state of the game application.

Distributing systems presents several abstract concepts, that could be difficult to introduce into the scenario without resulting artificial. An example, consider the event ordering and the fault tolerance concepts. On the contrary, the architecture of a distributed game naturally contains several of these aspects. A game clearly has well known rules and goals. Therefore, the corresponding architecture and functionalities expose relevant problems in the field of distributed systems. As an example a board game needs at least a reliable communication scheme, a shared common state, a leader election mechanism, and ordering of the players that possibly leave the game due to disconnection, or to a slow network response. The strongest requirement is that the architecture has to be a kind of peer-to-peer, avoiding any form of centralization during the play.

MAIN REQUIREMENTS

Shared common state:

It strictly depends on the kind of game, and represents the local view of each player. It shows the same data to any player, and it has to be strictly coherent. In our game which we are designing we are strictly in need of the sharing as every player must be able to see another player shooting at the enemies so that he can shoot at other enemies to get points to himself.

Reliable communication:

To avoid the impossibility of consensus. Also the communication uses standard TCP protocol, excluding UDP. The latter is usually blocked by firewalls, resulting scarcely usable over internet.

Fault model:

The only kind of processes fault is the player crash i.e. a player that suddenly do not responds to any message. All the other components of the system are reliable. This condition is to introduce the fault tolerance issues. Note that, in the case of gaming, arbitrary faults to manage. Note that, in the case of gaming, arbitrary faults are equivalent to cheating.

Tolerant to multiple faults:

Managing a single fault could be tricky sometimes and misleading with respect to a general approach. Therefore the game must be correctly continue even if all but one player crashes.

Imposing reliable communication allow to focus on processes fault solely, and keeps affordable the complexity of the system.

CONCEPTS RELATED TO THE GAME

Non centralized server:

The client–server paradigm offers an easy solution to on–line gaming. A centralized server could store and manage the common state of the game, answering to the client moves. This kind of architecture is not interesting from a distributed point of view, therefore it is forbidden. The absence of a game server , commonly leads to a leads to peer-to-peer architecture.

Shared global state:

In general, the shared state in a third person shooter multiplayer game, comprehensive of any position of all the other players. In this type of game, the shared state is the arena in which the shooting is taking place, including the movement of the players and positions of the players as well as the enemies.

Message passing:

The players could communicate with each other solely by exchange of messages. The absence of both a central server and a shared memory, forces this kind of communication. In our case of gaming, we need message passing in order to let all the players know that an enemy is being shot dead by a player and the other players will have no chance to shoot at the enemy making them to move onto shooting at another enemy to gain the score. Also each time the bullet hits the enemy the newly updated health of the enemy is being broadcasted to all the players to maintain transparency.

Event Ordering:

A casual ordering of events will make much more sense rather than trying to implement a clock synchronized model. Any change in the game has to be delivered to the players preserving its order. Usually, the adoption of broadcast methods solves that problems.

Fault tolerance:

Assumption of reliable communication channels provide simple fault detection mechanism. If a player does not answer to a ping message before a predefined deadline, we can assume that the player has crashed. On the other hand, we can assume that the player has crashed if it does not send a periodic ping before a predefined deadline. A player re-joins a game that it has abandoned because of the crash, while a crashed player usually remains unavailable until the end of the game. Also dealing with the crashes of the master client should also be taken care of by choosing a leader election method such that even when the leader crashes, there will be alternative of continuing the game by changing the leadership to the second joined player and so on.

Leader election:

The player who has initially joined the game will be serving as the leader managing as the master client. In the above discussed architecture, a crash of the leader is not a problem, because it can be simply detected and managed by its successor in the logical ring.

DESCRIPTION OF THE DESIGN AND IMPLEMENTATION OF THE GAME

We used Photon Unity Networking library for all the network functions we have used. Messages are passed about the states of each player to all other players for each update of the player's transform or player's rotation. As the game involves animation of the characters passing the options when to play certain animations is also an important part. We need to synchronize the health of the enemy bots after every shooting occurs. So instead of updating these changes for every frame we used RPC to damage the health of a bot if any player shoots at it. All the players have the same RPC's to call from. Whenever a player shoots and hits a bot, a RPC to decrease the health of the bot is called for all other players.

The need to order the events of the game plays an important role in multiplayer shooter games where an enemy might be hit by both the players but only the last player to shoot will get the points. PUN has taken care of that issue in its library to order the events. The last player to shoot the bot before it dies will call RPC across all other players and the info about the caller will be passed along with the arguments of the RPC, this helps to update the player's score correctly. This is possible because the RPC's called will come in order at other player's end.

We have choosen the PUN's inbuilt leader election process where the first player to join the room will be the Master Client and he is responsible for spawning bots automatically in the environment. These bots will be synced with all other players. When a Master Client leaves the room the next player to join the room after master Client becomes the leader. This Solves the issue of Fault tolerance unlike the case in a centralized server game.

CONCLUSIONS

` In this paper we presented a different approach to educational games: the design and implementation of a game as a means to learn abstract concepts of distributed systems. In the presented case we analyzed how a course in Distributes System could take advantage from such an approach. We have also learnt the features of UNITY3D in alliance with c#.we have observed how it proves to be an excellent framework and how it can be effectively used to develop 3D games. We understood how the concepts of RPC, Leader election, Shared Global states and Fault tolerance in distributed systems.

Assets used:

1. **http://unity3d.com/learn/tutorials/projects/survival-shooter**
2. **https://www.exitgames.com/en/PUN**

REFERENCES:

1. Distributed Gaming by Omar A. Abdelwahed Revision 1.1
2. Colyseus: A Distributed Architecture for Online Multiplayer Games BY Ashwin Bharambe, Jeffrey Pang and Srinivasan Seshan.
3. TAN, S. A., ET AL. Networked game mobility model for first-person-shooter games. In NetGames (Oct. 2005).
4. BHARAMBE, A., ET AL. A Distributed Architecture for Interactive Multiplayer Games. Tech. Rep. CMU-CS-05- 112, CMU, Jan. 2005.