Multiplayer Tank Game

Scott Austin

Department of Computer Science and Networking

Wentworth Institute of Technology

Boston, MA 02115, USA

[saustin@wit.edu](mailto:saustin@wit.edu)

Introduction

The goal of my project is to implement an application making use of networking protocols. My project is a real-time multiplayer tank game using the UDP network protocol. Two users of the application on separate host systems will be able to connect to one another and participate in simple tank arena combat. Both players can move around the arena, aim their turrets and fire shells.

Actions

To make the game work in a networked environment, the game’s actions and program state must be explicitly defined. The arena consists of the 1080 x 720 screen space. There are two tanks present in the arena, one controlled by a “host” player and one controlled by a “client” player. Tanks can be moved forward or backward relative to their rotation by pressing the ‘W’ and ‘S’ keys, respectively. Tanks can rotate left and rotate right by pressing the ‘A’ and ‘D’ keys, respectively. The tank’s turret will rotate equally with the tank’s rotation. The tank’s turret can be rotated independently left and right with left arrow key and the right key respectively. A tank can fire a shell from their turret using the ‘space’ key. Only one shell per tank can be on screen at a time; if a tank’s shell is already active, the ‘space’ key does nothing. If a tank is hit by an opponent’s shell, it is destroyed.

While there are multiple keys that perform various actions, these can be broken down into three major types of actions: movement, firing and hit. With the game reduced to 3 actions, we can define what goes into our messages.

Messages

There are three main actions to account for in our game, therefore we should have 3 different message types. We’ll define the message types as TG\_MOVE, TG\_FIRE, and TG\_HIT. In move and fire, we should send all the data about the player’s tank to the other client. This data includes it’s x-coordinate, y-coordinate, rotation in degrees, and the turret rotation in degrees. All of this data are floating point values, stored in 4-byte float variables. In the hit message, we don’t need to include any additional data since the only noteworthy task is to remove the hit tank from the game. In our header, we will include an integer denoting the message type as well as an integer to act as a timestamp of when the message was sent. Since we are using the UDP protocol, we have no guarantee when or if a message will be received, nor any guarantee that they will be received in order. We’ll use the timestamp to record how recent our last message is and discard any messages that are old and out of date. We will also have a TG\_START message to indicate that a client has connected to a host and the game can begin. Figure 1 shows a table detailing all our defined messages.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Figure 1** |  |  |  |  |  |
| TankGameByteBuffer | byte-offset |  |  |  |  |
| int | 0 | time\_stamp | time\_stamp | time\_stamp |  |
| int | 4 | TG\_START | TG\_MOVE | TG\_FIRE | TG\_HIT |
| float | 8 | X | xpos | xpos | X |
| float | 12 | X | ypos | ypos | X |
| float | 16 | X | rotation | rotation | X |
| float | 20 | X | turret\_rotation | turret\_rotation | X |
| END DATA | 24 |  |  |  |  |

Issues

Using the UDP protocol, we have no guarantee that messages have been received or that they arrive in order. Unlike the TCP protocol, we must handle errors like this in the application layer. However, due to the nature of this application, TCP is insufficient because the overhead involved in guaranteeing delivery would greatly impact the latency of the actions.

Because this application has two real-time players that are acting separately, each player is likely to see different game states on their screens. Even using UDP, there is still a natural latency in transmitting data over the network. It is possible that the two applications could go out of sync if, for example, one player’s application believes that a tank was hit by an opponent’s shell but the other player’s application believes that the tank moved out of the way and dodged it. There are many ways to handle this kind of conflict resolution, for example by making one player application the authoritative server that says other applications must differ to what it says is the truth. No such methods are implemented in this project however.

Conclusion

A real-time game is a demanding network application to take on. There are many concerns involving how to handle the shortcomings of the UDP protocol and how to handle conflict resolution when multiple applications do not agree on the game state.