Multiplayer
Danny Kart Racing

Jeremy Hauschildt
Advisor: Dr. Judy Challinger
Overview

Background and Motivation
What is Multiplayer DannyKart Racing?
Requirements and Features
Design Issues and Problems
Methods and Design
Network Interaction and Results
Demo and Analysis
The Future
Background and Motivation

- DannyKart Racing¹ was created last semester (Fall 2010) by Chico State student Daniel Phelps
- 3D racing game written in C#® using XNA Game Studio® and the JigLibX® physics engine
  - Single player, race against time
  - Level building
- Intended to be extended by future students
- Next step seemed to be allowing multiple people to play together
What is Multiplayer DannyKart Racing?

Multiplayer racing game

- Game Client (extends previous project)
- Game Server
Requirements and Features

Multiple people racing each other over a network

DannyKart Server handles multiple races simultaneously

Client can choose a specific server and game

Deal with latency

Easily configured via XML
Design Issues

First design (Midterm Documents)
- Server accepts requests to move from clients and calculates positions with its own physics engine
- Too complicated and resource intensive for the scope of my project
  - Didn’t fit into existing project as well as I thought
- Have to keep multiple physics engines synchronized

Second design
- Clients keep physics engine and send positions at regular intervals to the server
- Clients send prediction information so they can be drawn in between updates (velocity)
- Server now just has to relay these positions fast and fair
Reasons for using UDP

- Faster than TCP (TCP has lots of overhead)
- We don’t care if all position packets get there, only newest

Problems

- Unreliable
- Packets are not guaranteed delivery or order

Lidgren

- Networking library
- Throw away old packets
Methods

Research: Network programming, XNA®, C#®, 3D, previous code base

Bottom up implementation at first
• Implemented and tested basic underlying network components

Then iterated between bottom of project and top for each use case

Object Oriented
• Encapsulate network architecture
• Fit my project into previous object hierarchy

Tools
• Visual Studio®
• XNA Game Studio®
• Mercurial® (version control)
Game Server Class Diagram
Packet Structure: Examples

Position Packet (sent while actively racing)
- 8 bit unsigned player ID
- Vector3 with position (3 floats)
- Matrix with orientation (3 floats)
- Vector3 with velocity (3 half precision floats)

GameState Packet (sent before a game starts)
- MpNetGameState object
  - Game Id
  - List of MpPlayer Objects (name, id, car, position, etc.)
  - Actions that can be preformed on a game (remove players, update players and add players)

GameList Packet
- List of MpGameInfo objects (information about a game)
Results

Client output is a steady 2.89 kilobytes/second
• Default update rate is 15 ms for clients
• Server can specify what update rate its clients need

Server needs to be able to send/receive this from each client
• EG: 3 racers, server needs to be able to send and receive 8.67KB/s
• Server requirements go up with each connected player

Largest position packet is 49 bytes

Current maps support 5 players at a time
• Larger maps can be added to support more users
## Example Interaction

<table>
<thead>
<tr>
<th>Protocol</th>
<th>Data/Contents</th>
<th>Sent By Type</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>[Connect]</td>
<td>Client Name1</td>
<td>Client &lt;String&gt;</td>
<td>Client connects to server</td>
</tr>
<tr>
<td>[Connect]</td>
<td>15ms</td>
<td>Server &lt;int&gt;</td>
<td>Server tells client update rate</td>
</tr>
<tr>
<td>[GameList]</td>
<td></td>
<td>Client &lt;&gt;</td>
<td>Client requests a list of games</td>
</tr>
<tr>
<td>[GameList]</td>
<td>&lt;GameListObject&gt;</td>
<td>Server &lt;Object&gt;</td>
<td>Server compiles and sends list of games</td>
</tr>
<tr>
<td>[JoinGame]</td>
<td>Game # 12</td>
<td>Client &lt;int&gt;</td>
<td>Server requests to join a game</td>
</tr>
<tr>
<td>[GameState]</td>
<td>&lt;GameStateObject&gt;</td>
<td>Server &lt;Object&gt;</td>
<td>Client gets state of current game</td>
</tr>
<tr>
<td>[Start]</td>
<td></td>
<td>Client &lt;&gt;</td>
<td>When client is ready, let server know</td>
</tr>
<tr>
<td>[Start]</td>
<td></td>
<td>Server &lt;&gt;</td>
<td>When all clients are ready server says go</td>
</tr>
<tr>
<td>[Position]</td>
<td>&lt;PositionObject&gt;</td>
<td>Client &lt;Object&gt;</td>
<td>Game started, client sends its position</td>
</tr>
<tr>
<td>[PositionState]</td>
<td>2, PositionObjects[]</td>
<td>Server &lt;ushort&gt;&lt;Objects&gt;</td>
<td>Server sends out positions of all players</td>
</tr>
<tr>
<td>[Position]</td>
<td>&lt;PositionObject&gt;</td>
<td>Client &lt;Object&gt;</td>
<td>Client updates its position</td>
</tr>
<tr>
<td>[FinishedRace]</td>
<td>&lt;TimeObject&gt;</td>
<td>Client &lt;Object&gt;</td>
<td>Client finished so let server know time</td>
</tr>
<tr>
<td>[FinishedRace]</td>
<td>2, TimeObjects[]</td>
<td>Server &lt;int&gt;&lt;Objects&gt;</td>
<td>Server sends compiled list of times</td>
</tr>
<tr>
<td>[LeaveGame]</td>
<td></td>
<td>Client &lt;&gt;</td>
<td>Client leaves the game</td>
</tr>
<tr>
<td>[Disconnect]</td>
<td></td>
<td>Client &lt;&gt;</td>
<td>Client disconnects</td>
</tr>
</tbody>
</table>
Demo

Connected Players:
Jeremy
Batman
Bruce Wayne
Person
Racer1

GO!
Final Analysis

All use cases from original requirements document are complete

Software runs fairly smooth
• Server handles, gracefully, clients who drop unexpectedly

Most limitations are based on server resources
• Commercial multiplayer games have higher requirements
• Servers run on consumer connections do not perform

Local network play runs well as bandwidth is not a concern

More testing is needed
Testing

Already done extensive lag simulation with Lidgren

Server load testing
  • Testing with more clients
  • Testing with more clients in different games

Test with more users over the internet

Work on further reducing packet size

Work on dynamically changing update intervals based on latency to clients
Summary
References

1 Daniel Phelps. DannyCart Racing. CSU Chico, Fall 2010.

   http://unreal.epicgames.com/Network.htm

Resources:

- C#
- XNA Game Studio 3.1
- Lidgren.Network – Generation 3
- Visual Studio 2008
Thanks

Advisor: Dr. Judy Challinger
Dan Phelps
Dr. Renner
Faculty
Multiplayer
Danny Kart Racing

Jeremy Hauschildt
Advisor: Dr. Judy Challinger

Questions?
End
Supplementary Slides

To help explain possible questions
Future Work

There is a lot left that can be added to DannyKart

Some features I wanted to add but were out of scope:
  • Mini overview map of where players are
  • More maps
  • Multiplayer leader board (partially in place)
  • Global, centralized server list

Possible future directions:
  • Different game modes (both single and multiplayer)
  • AI
Sequence Diagrams (client)

- Assume state is already past menus and starting to play a networked game
- Separate thread updates variables as they are received over network
Sequence Diagrams (server)
Client Interaction Diagram
Game Interaction Diagram
Game Client Class Diagram (relationships)
Methods

Mix between the waterfall method and iterative

• Specified requirements
• Designed
• Implemented/prototyped
• Designed
• Implemented/tested

Only needed to iterate back once, when the design slightly changed
Demo

Video times, just in case

1:30
3:20