Lappeenranta University of Technology
School of Industrial Engineering and Management
Software Engineering and Information Management

2D multiplayer racing game

Jari Kauppila
Toni Martikainen
Jiri Musto
Shaghayegh Royaee
Joonas Salminen

Table of contents

1 Game description

The game is a 2D top-down racing game with 4 players. When joining the game the player is directed to lobby where it is possible to chat with other players. The server provides players with their ids and usernames. When 4 players have joined the lobby the actual game will start. In the game each player has a control of 1 individual car. There are 3 laps in the race and the first player to pass the finish line is the winner. The game will end after one of the players finishes the race.

2 Game mechanics and rules

- 4 players are able to take part in the race
  - Each player has control over 1 car
- Players start simultaneously from the start line
- Tracks consist of asphalt and grass
  - Driving on grass will significantly slow down the car
- The tracks have several checkpoints which the players need to pass to complete the lap
- To win the race a player has to finish 3 laps faster that his opponents
  - The game will end after there is a winner

2.1 Gameplay

Each player controls a car of a unique color, which is to identify the car from other players' cars. Game starts from the start line after 4 players have joined the server. The players race against each other and the first one to complete 3 laps is the winner. The tracks have both grass and asphalt elements. It is advantageous for the players to drive on asphalt because the grass will slow down the car. Several checkpoints on track also prevent the players from cutting the corners. Currently players are not able to collide with each other and the cars will go through each other in case of a collision. The cars can be controlled by using the arrow keys. Up-key is the throttle, down-key is reverse/break and the car can be steered with left- and right-keys.

3 Game design

The 2D race game consists of client and server application and it can be played through networks that support UDP over IPv4/IPv6. The game is implemented with Java in the Netbeans environment and it can be executed on several different Java supporting operating systems. The graphics are implemented by utilizing Swing/AWT libraries and the modeling of the cars and tracks is done with Paint.net. Networking is implemented by using multithreading and basic socket programming.

The game has a lobby which the players join before the game starts. The lobby system is implemented so that immediately after 4 players have joined the lobby the game starts. Players are able to chat with each other in the lobby. Chatting is also possible during the game through the terminal that the lobby runs in. The chat is also implemented in UDP-protocol.

The game itself is very simple. The game physics are basic arcade game physics: the cars can accelerate and there is some friction but these are highly simplified. Furthermore, there is no collision detection between the cars to keep the calculations in the minimum and thus ensure better performance. Finally, the checkpoint system is basically just server checking whether players have passed certain track points. The similar method is also used for checking whether the finish line has been passed.

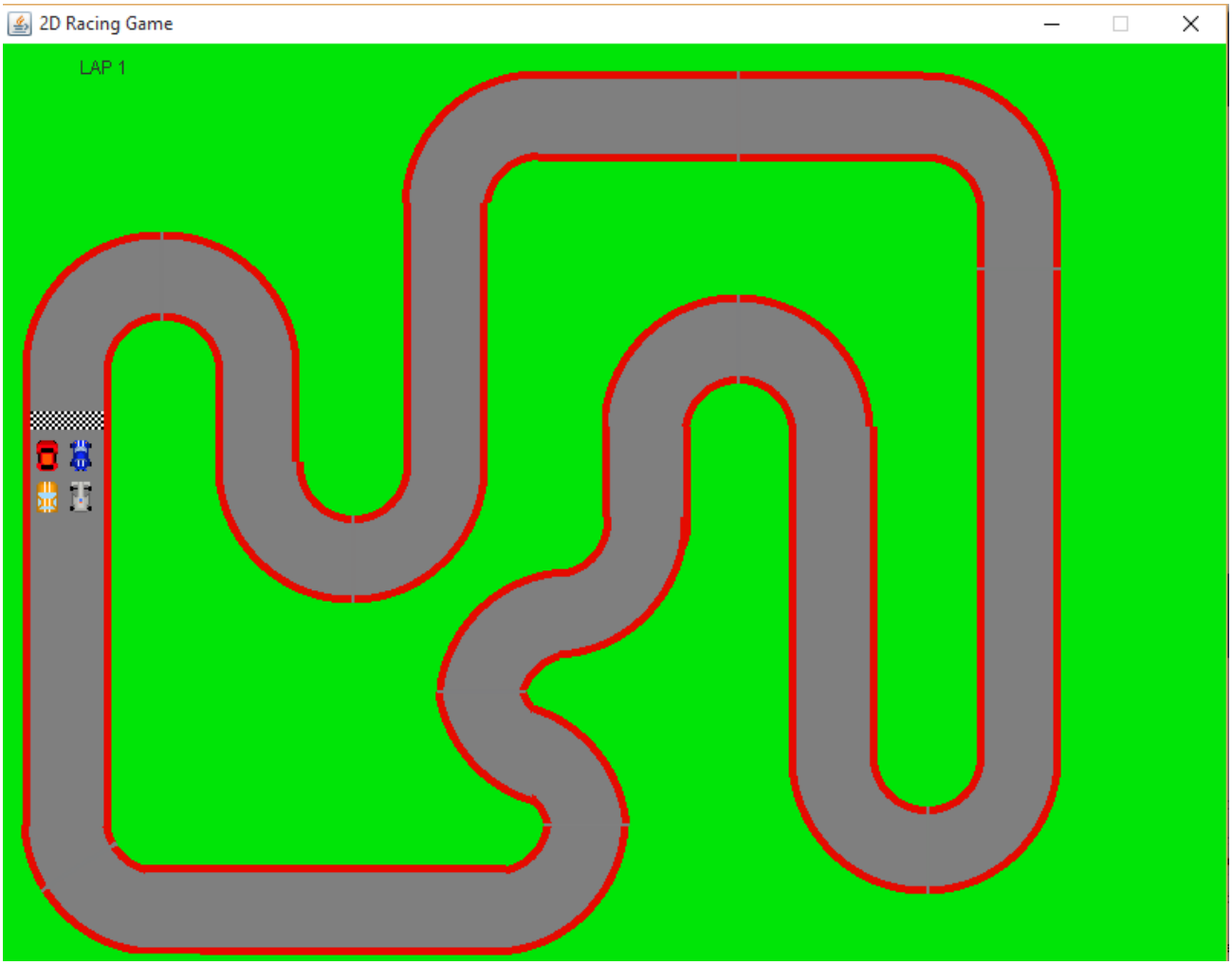

Figure 1: Sample UI - Multiplayer version

Game is made with multiple classes which contain the appropriate functions for each part. The game is divided into two runnable applications, RacingGameClient and RacingGameServer. Server consists of main class and Car class to calculate each players movements on the server-side. Client consists of main class, Car class, GameSurface class and Prediction class. Car class calculates the player movements on the client side, Prediction class is used to store the inputs for client-side prediction and server reconciliation and GameSurface class handles the graphics and draws each player on the screen.

4 Protocol definitions

The networking is based on UDP-protocol. There are 8 amount of different kind of packets sent between the client and the server. When joining client sends a packet containing username to the server and server answers with a packet containing the user ID thus accepting the connection. When the game starts server sends packet to client containing the game start message and map data. In the game loop client sends packets that contain the client ID and position coordinates of the client to the server and server sends packets that contain the location of other connected clients to the client. When the game ends server will send a packet that contains the end message to all the connected clients. There will be a chat running parallel to the game itself so players can chat with each other. The packages used in the chat contain the chat message, packet ID, user ID and username. The server will then broadcast the message sent from 1 client to all of the clients. The packets sent betweent the clients and server each contain an identifier. The protocol of the game is explained in figure 2.

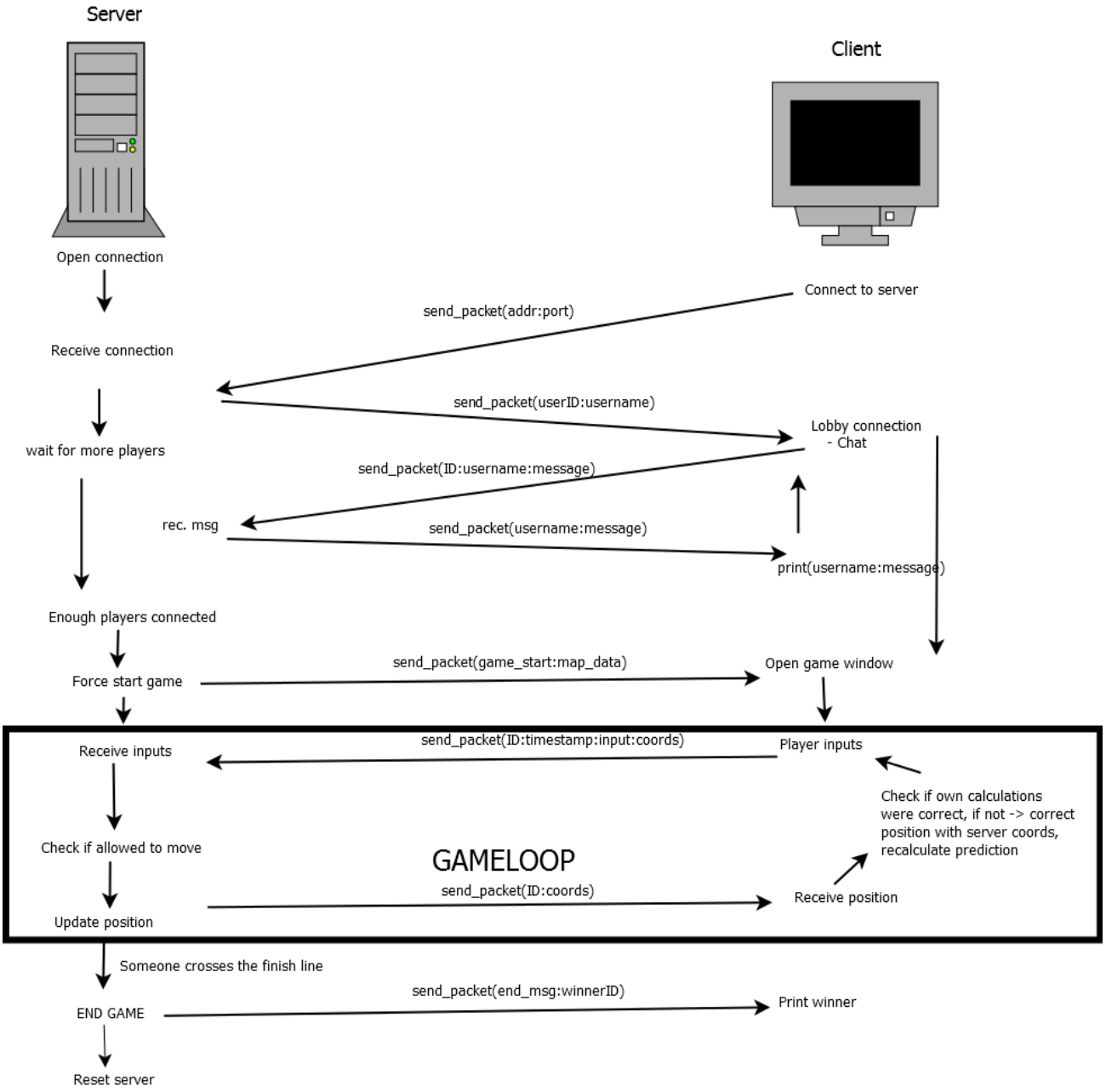

Figure 2. Game protocol

Figure 3 describes the joining protocol. Client sends login_packet to server and server responds with login_response_packet and ID_data_packet. When enough clients have joined the server sends game_start_packet to the client and the game starts.

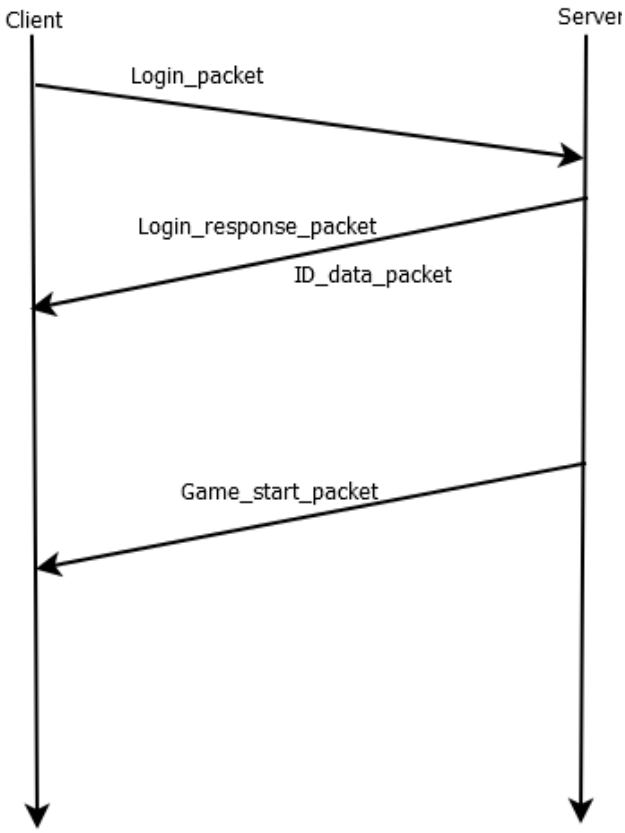

Figure 3. Client-Server connection handshake sequence chart.

In figure 4 the typical game situation is described. Client sends packet containing it's coordinates and server answers with packets containing other clients coordinates.

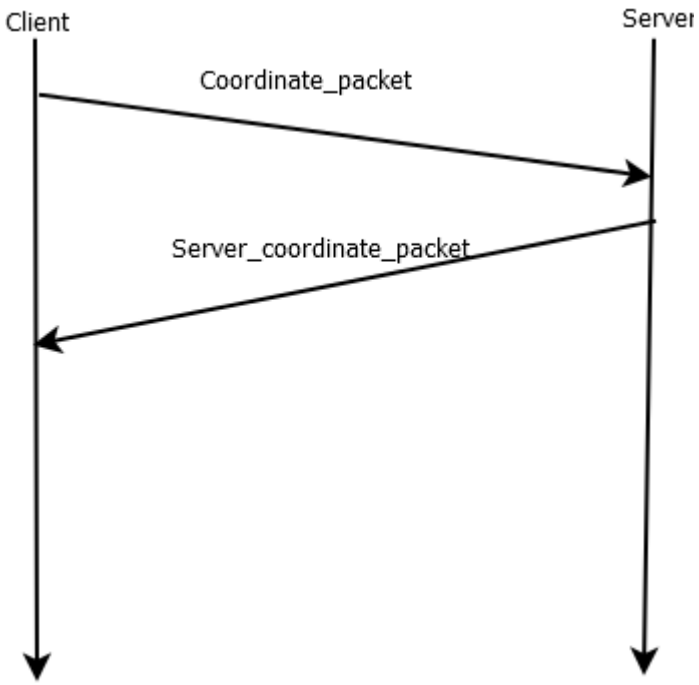

Figure 4. Game situation

In figure 5, the game end procedure is explained. Client sends coordinate packet containing the coordinates that tell the client has passed the finish line. Server sends a packet to all other clients telling who has won the race, ending the game and resetting the server.

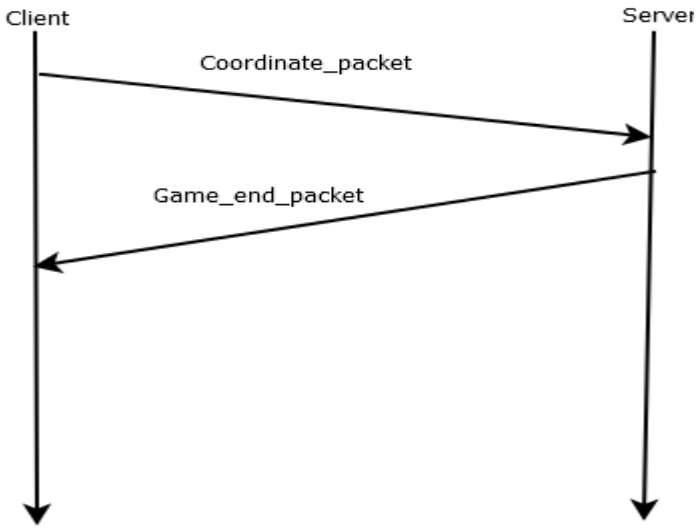

Figure 5. Game end

5 Advanced features

As advanced feature we implemented client-side prediction with intention to compensate latency issues. The principle of client-side prediction is to perform the client's movement locally and just assume, temporarily, that the server will accept and acknowledge the client commands directly. The general procedure for client-side prediction is as follows: client inputs are sampled and a user command is generated. Also as before, this user command is sent off to the server. However, each user command (and the exact time it was generated) is stored on the client. The prediction algorithm uses these stored commands. (Bernier, 2001)

6 Work schedule and division of work

Milestones:

Deadline	**Description of work**
1.12.2015	**Working single player game**
18.12.2015	**Game design document 1st version**
18.12.2015	**Start Skype sessions for the project**
5.1.2016	**Better graphics for cars and tracks**
15.1.2016	**Chat part done**
31.1.2016	**Primitive networking done (works but might not have advanced techniques)**
15.2.2016	**Beta release or first final release**

Division of work:

Jari Kauppila: Idea, programming, testing

Toni Martikainen: Idea, programming, testing

Jiri Musto: Idea, programming, graphics, testing

Shaghayegh Royaee: Idea, documentation, testing

Joonas Salminen: Idea, programming, testing

7 Listing of additional libraries used

None besides the Swing and AWT libraries mentioned in chapter 3. Game design.

8 Situations not handled in implementation

Currently the game will not notice when players disconnect during the chatting phase and will start with less players. This trick can be useful to start the game with less players, though. Cars belonging to disconnected clients will not be there once the game begins.

There is also no handling of player disconnections during a game other than their car staying still where they left. Because there is no car collision, this does not matter much. Sadly if all players disconnect during a race the server will be stuck in the game mode forever and must be restarted.

The server restarts itself when a game ends, but the game client does not have functionality for this and must be restarted.

9 References

Bernier, Y. (2001). Latency Compensating Methods in Client/Server In-game Protocol Design and Optimization [Article]. Available: <http://web.cs.wpi.edu/~claypool/courses/4513-B03/papers/games/bernier.pdf>