IIT Delhi

Pong!: A Networked Multi-Player Game

Design Document

Author:	*Entry No.:*
Ashley Jain	2014CS50280
Dasari Chandu	2013EE10449
M.Sreevatsa Saurabh	2013EE10475

Software design document:
for Desktop Game Application using Swing Graphics

on

April 2016

iit delhi

0.1 Introduction

This Design Document aims to provide a complete picture of how we plan to implement a multiplayer peer-to-peer ping-pong game for desktop environments called "Pong!".

- **Scope and Features:**
  Our application is an extension to the existing classic arcade game *Pong* by *Atari*. It includes many features which the original game doesn't have. They are:

  - **The Classic single-player mode** in which the user can play against a computer player. In doing this he will also have an option to select the computer player's *difficulty level*. More about how this is implemented is desribed in section X.X
  - **Multiplayer mode** where a maximum of 4 players with atleast one of them human can play against each other.
  - During multiplayer gameplay, if any of the existing users is disconnected, remaining users are given an option of continuing with their game by substituting the disonnected player with a computer player. In such a case, the difficulty level of computer player is fixed at 'medium'.

  Coming to the user control of the paddle and the ball,

  - The user can move his paddle by using either the mouse pointer or keyboard keys.
  - On hitting the paddle, the ball is reflected back with a velocity which can be influenced by the paddle's velocity when it hits the ball, producing a spin-like effect to the ball movement.
  - We also plan to have some additional special powers given to the user's paddle like *increasing the size of his paddle* or the power to *capture the ball* and release it at some place else, etc. as rewards for completing some achievements.
  - Users will also be given an option to vary the *speed of the ball* and the *number of balls* in play, though it will make the gameplay difficult.

- **Project Overview**
  This application is to be written entirely in *Java*. It uses Java's *Swing* library for implementing the User Interface (GUI) and *Socket programming* for the network communications which are required for the multiplayer mode.

0.2 Algorithm for computer player

- Computer player algorithm:

  As soon as the user hit the ball with the paddle,the AI will calculate the velocity of the ball after the collision and with the velocity it will predict where the ball hits the floor.

  After predicting where the ball hits the floor to prevent it the AI has to move the paddle to hit the ball.

  Depending on the difficulty level which the player chooses, the AI paddle's maximum speed is fixed.

  Difficulty level is also manipulated by varying the probability that the ball will take the required velocity i.e., at easier levels, the probability that the ai uses its maximum velocity is low. So there is a higher chance that it misses the ball.

  As we are designing the network communication to be Peer-to-Peer communication so the computer player runs in all the machines in a distributed fashion.

  There can also be a situation where the user strikes the ball towards a wall in order to make the ball bounce off it. In such cases, the computer player which was only considering the ball's velocity when it hit the player may predict differently. In such cases, the computer has to consider the velocity of ball even when it hits the wall and recalculate the position.

0.3 Physics of the game

- The ball's collision **with the wall** follows an *elastic collision* where the ball's velocity of approach and velocity of seperation are the same along the

direction perpendicular to the wall. The component of velocity along the direction of the wall is left unchanged.

- The ball's collision **with the paddle**. The ball's collision with the paddle, is different from that with wall, in the sense that the paddle's velocity at the time of collision, also influences the ball's component of velocity along the paddle's length.
  After hitting moving paddle,the new speed parallel component would be equal to some factor value of parallel component speed of moving paddle with same direction and perpendicular component of ball speed would be in same logic as given in previous point.

$$V_{ball,horizontal} = U_{ball,horizontal} + f \times V_{paddle,horizontal}$$

  where f is some fraction ($f \in [0, 1]$), which we plan to decide on, after the implementation is done, from testing.
  This is done so as to ensure that the ball doesn't go into an infinite loop of volleying back and forth vertically when the computer player doesn't move.

0.4 Network Communication

In order to implement the multiplayer game mode, we will be using a network with *full mesh topology* which connects each computer to every other one in the game. The game will use a non-authoritative peer-to-peer approach.

0.4.1 Information that is sent

Each player in the network, sends information only about the events which are directly related to him (instead of sending his entire game state). These events when received by other players, are simulated in their machines. The rest of the game state for that peer, is made entirely by itself. This is done so as to reduce the bulkiness of each message, which in turn speeds up the process of packaging and unpacking the messages, which is the major contributor to the lag in receiving messages. So, only the messages containing the following info are sent:

- The **paddle movements** of each player. Only the movements ('right' or 'left' key presses) are broadcast to all the other players in the game.
- **Special Powers**. when a player obtains or loses special powers, he sends a message having the following.
  - special power type
  - attained or lost
- **Collisions**. This is the information about collision of the ball with paddle or the wall. This message consists of the following:
  - position of the paddle at the time of collision.
  - final velocity of the ball(both the components).
  - points lost or gained.
- **Connection messages**. Each player will keep sending an additional message periodically which will indicate that the player is connected. This helps in detecting

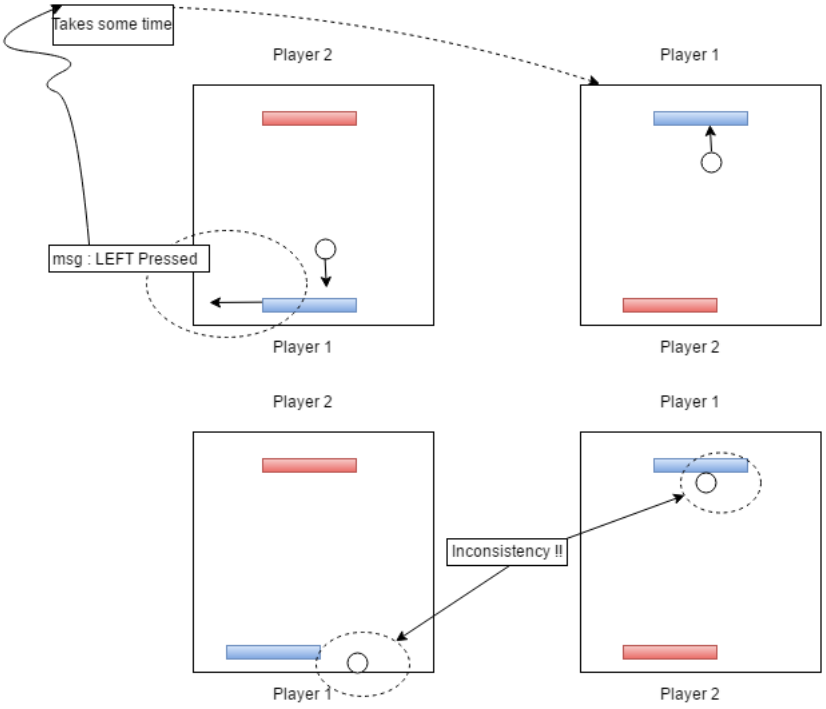
0.4.2 Maintaining a consistent game state

Inconsistent game states between players may occur in the following way.

To prevent such inconsistencies from snowballing into bigger differences, the player2's score is updated only when the collision message from player 1 is received, so that there is no difference in scores seen by each player. As mentioned in the previous subsection, all significant events relating to a player are simulated by other players based on the messages they receive.

Also, a reliable protocol(TCP/IP) is used for sending information about collisions and such rarely occurring, but important phenomena. On the other hand events such as movements of the paddle can be sent using less reliable and more fast protocol (UDP).

When more noticeable events like updating the score or giving a special power to the user experience such glitches, it leads to an inconvenient game-play. So such events are done only after reliable information about them is obtained, whereas we can afford some unnoticeable glitches in the movement of paddle etc. since there are many such commands and only the most recent ones matter the most.



0.4.3 Network message event flow

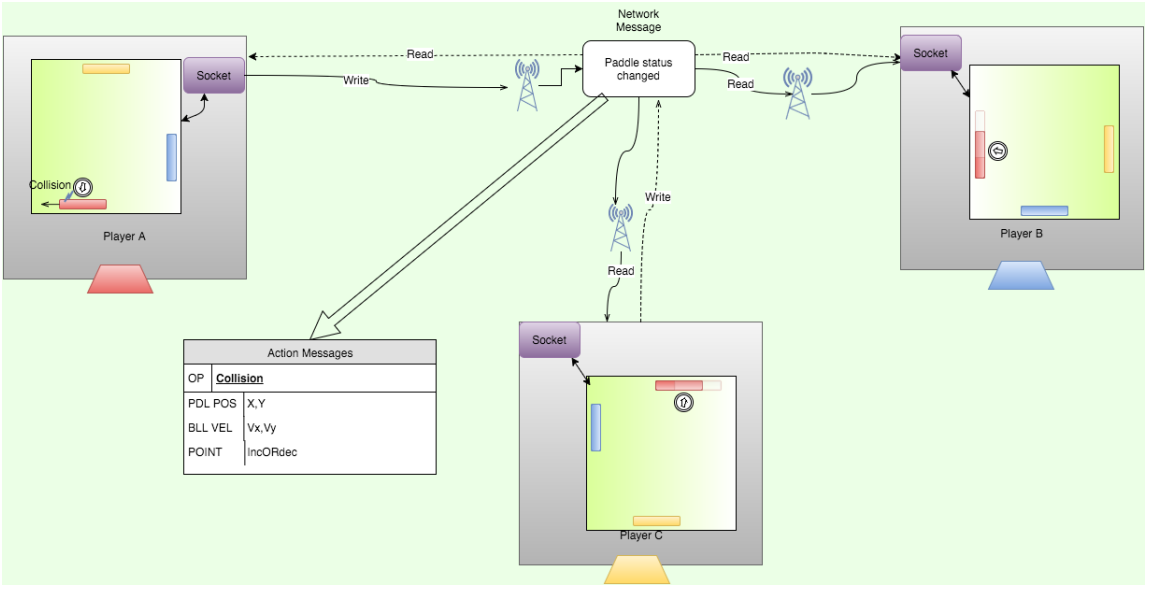


FIGURE 1: Event Flow

0.4.4 Handling a Disconnected user

Detecting a connection failure

As mentioned above each player sends a periodic connection message which informs all other players that he is connected. when such a message is not received from

a player for a fixed amount of time, the player is said to be *timed out*.

Handling it

When a player is detected to be timed out, all the remaining players are given an option to terminate the game or continue playing with the disconnected player replaced by a computer. This decision is made based on majority decision which is taken from a vote on a pop-up screen.