Nicholas Pierson

Parker Mathewson

CSC 422

Project 2 – Distributed Pong

5/6/2015

**Introduction**

Our project keeps the initial implementation of sequential and parallel collisions of bodies (or in this case balls) and extending the previous project to a version of Pong that is distributed across multiple computers. Each client connected to the server displays half the screen and redraws all the information according to the server, which does all the calculations.

**Programs**

This Distributed project has both a sequential and a parallel solution written in java that calculates the collisions on both other bodies as well as the walls and paddles. The main part of this project is the server and socket implementation that allows this version of Pong to be played across a maximum of three computers. One computer is the server computer, which calculates the position and collisions of all the bodies. The other two computers are client computers that display a graphical user interface and displays the balls and paddles according to the positions passed to them through the server socket relation.

The project was made within Eclipse and has three packages: controller, model, and view. The controller package contains PongController, which calls the appropriate server and client according to command line arguments, and from there, calls the appropriate constructors for each computer. If the server command line argument is specified along with an IP of the current machine, the server starts up and waits for two computers to connect to begin a game. To start a client, all that is needed is an IP, no client command line argument is needed. Without any other command line arguments, the sequential program will start with a single ball once both clients have connected. If other command line arguments are specified, specifically with more workers, a separate constructor will be called producing the parallel version. If a command line argument to specify more balls is entered, then the balls will all start in the center of the screen with varying velocities initially.

Also within the controller package are the PongServer and PongClient. These classes specifically are the communication controllers between the computers. These are where the server socket implementation is produced and multiple threads start allowing the communication between the computers. The threads are created using the ClientHelper and the PongSubServer classes within this package. Messages are passed back and forth using objects specified by classes named ClientMessage and ServerMessage within the control package and are utilized within their respective Server or Client using the bufferedReader resource from Javas libraries.

ServerMessage is a class specifically designed to hold the messages that the server needs to send the client, and the client has a method to parse out the message and send the information to its respective area of the project. The ClientMessage class is the same process, but rather sending from Client to Server. These message classes allow us to be able to send one large message with all the information needed to update both clients in a timely manner, and in a well organized way to where anyone looking at the code can see much easier what is being attempted.

Within the model package, you will find the base of our project, mostly all available from the parallel project from earlier in the semester. The Body, Point, and CollisionWorker classes are all identical from the previous project, but Collision was altered to fit the scope of the game of Pong. In line of checking for collisions with other balls, the Collisions class now also checks for collisions of walls and of paddles used within the game using all elastic collisions, or non-energy consuming collisions. Gravity was also taken out of the equation (as well as forces) as all the balls are of the same radius and mass. The model class reads in from a file currently to initialize up to 10 balls (any more and the game became overwhelming). From here, the code is looped through on a continuous game loop that allows the game to be run until someone wins.

In the view package, this is where the CircleBody class lives, which is a JPanel object that is used to create the balls and paint them on the screen using the position given by the Collision class. This class is the exact same from the previous project. There is a new Paddle class that creates a rectangle on the screen and is movable, like the balls, and is repainted on the main game GUI. There is the startup popup JFrame, which is the class OptionGUI which has the title of the game (Distributed Pong!) and allows the user to select left or right. The implementation was supposed to not allow the user to connect to an already connected side so the users can enjoy the game as it was proposed. Once the users are connected, the main glass of the view package comes up, which is the PongGUI. This class takes input from the client through the form of the position of the balls passed through the socket from the Collisions class. With this, each client is shown one half of the whole game board. From here, the games difficulty increases as if you are sitting apart from the other player, you have half the screen to predict where the ball will be, rather than the full screen. This was a design choice by us to show the complete separation of the computers that are clients, rather than displaying the same board and have individual controls only. We thought about changing this to allow for easier gameplay, but decided to show that we can also implement a GUI that is dependent on each other while being independent. Within the main GUI as well, rather than having a 180 degree change in velocity if the corner of the paddle is hit, a calculation was made to send the ball flying and increase its trajectory to help trip up your opponent.

**Verification**

This project was verified through the reusing of old functions that were previously verified in the last project. Of course, we re-verified since we altered code by creating points with set positions and velocities that should hit the walls, or paddles, or other balls in various, yet calculated positions. All the calculations seemed to work well, and gave correct numbers within a range of acceptable error (since some calculations within the game are rough calculations rather than full physics force equations). We also verified that the game worked by setting the balls to go automatically and as soon as the first ball went behind a paddle, the round ended with the opponent winning the point for that round.

**Conclusion**

Thus far, on May 6th at 9pm, we had not been able to fully connect our components together, as one of our members was unable to work on the project for a time while trying to catch back up in school. Our main base of code was done, but many of the implementations we wanted to put into our project were left unable to be implemented fully. Distributed programming isnt always the best option to go about things, but sometimes it is the only way to communicate through programs, rather than relying on one computer to have all the necessary components.