# CS4344 Assignment 2 Report

*Qing Cheng A0091747W*

*Yang Zhixing A0091726B*

## 1. Local Lag

```
var LOCAL_LAG = 200;
var PERCENTAGE_LOCAL_DELAY = 0.8;
```

```
185         // Short circuiting the paddle movement, with a
186         // local lag.
187         setTimeout(function() {myPaddle.x = newMouseX;}, Math.min(PERCENTAGE_LOCAL_DELAY * delay, LOCAL_LAG));
```

LOCAL_LAG is the maximum local lag that can be perceived and tolerated by our group members. However, if the client receives update from the server before LOCAL_LAG, the client should update the position of the paddle immediately instead of waiting for the LOCAL_LAG. As the delay has a 20% jitter, so we set it to be 0.8 * delay to make it safe.

## 2. Hold and Wait

```
167     function freezeBall(){
168         that.vx = 0;
169         that.vy = 0;
170     }
```

This function freezes the ball at the current location.

```
this.checkForBounceForGeneral = function(topPaddle, bottomPaddle, isFromServer) {
    // Check for bouncing
    if ((that.isMovingLeft() && that.x <= Ball.WIDTH/2) ||
        (that.isMovingRight() && that.x >= Pong.WIDTH - Ball.WIDTH/2)) {
        // Bounds off horizontally
        that.vx = -that.vx;
    } else if ((that.isMovingDown() && that.y + Ball.HEIGHT/2 > Pong.HEIGHT) ||
            (that.isMovingUp() && that.y - Ball.HEIGHT/2 < 0)) {
        // Goes out of bound! Lose point and restart.
        that.reset();
        that.outOfBound = true;
    } else if (that.isMovingUp() && that.y - Ball.HEIGHT/2 < Paddle.HEIGHT) {
        // Chance for ball to collide with top paddle.
        if (isFromServer) {
            updateVelocity(topPaddle.x);
        } else {
            freezeBall();
        }

    } else if (that.isMovingDown() && that.y + Ball.HEIGHT/2 > Pong.HEIGHT - Paddle.HEIGHT) {
        // Chance for ball to collide with bottom paddle.
        if (isFromServer) {
            updateVelocity(bottomPaddle.x);
        } else {
            freezeBall();
        }
    }
}
```

When the ball hits the paddle, freeze the ball and let it stick with the paddle. Only when the client receives a velocity update, will it update the velocity of the ball. This should only apply to client side, so we implemented another function called checkForBounceForGeneral to solve this problem. Thus we can keep the server code unchanged.

```
var gameLoop = function() {
    ball.updatePosition();
    if (myPaddle.y < Paddle.HEIGHT) {
        // my paddle is on top
        ball.checkForBounceForGeneral(myPaddle, opponentPaddle, false);
    } else {
        // my paddle is at the bottom
        ball.checkForBounceForGeneral(opponentPaddle, myPaddle, false);
    }
    render();
}
```

At the client side, we call the correct function to check for bounce, so that the paddle can stick the ball without affecting the server side.

## 3. Local Perception Filter

When the “updateVelocity” message is received by the client, the client will check the ball state. If it’s stuck on the board (that’s when it’s ballVY!=0 and velocity is zero), we apply local perception filter by invoking the computeBallVelocity method.

```
case "updateVelocity":
    var t = message.timestamp;
    if (t < lastUpdateVelocityAt)
        break;
    lastUpdateVelocityAt = t;
    //ball.vx = message.ballVX;
    //ball.vy = message.ballVY;
    // Periodically resync ball position to prevent error
    // in calculation to propagate.
    //ball.x = message.ballX;
    //ball.y = message.ballY;
    if (message.ballVY !== 0 && ball.vx === 0 && ball.vy === 0) {
        computeBallVelocity(message.ballX, message.ballY, message.ballVX, message.ballVY);
    } else{
        ball.vx = message.ballVX;
        ball.vy = message.ballVY;
        //Periodically resync ball position to prevent error
        //in calculation to propagate.
        ball.x = message.ballX;
        ball.y = message.ballY;
    }

    break;
```

```
function computeBallVelocity(nPositionX, nPositionY, nVelocityX, nVelocityY) {

    console.log("====================================");

    var t, collideX, collideY;

    if (nVelocityX < 0 && nVelocityY < 0) {
        if ((nPositionX - Ball.WIDTH/2)/-nVelocityX < (nPositionY - Paddle.HEIGHT - Ball.HEIGHT/2) /-nVelocityY) {
            t = (nPositionX - Ball.WIDTH/2)/-nVelocityX;
            collideX = Ball.WIDTH/2;
            collideY = nPositionY + nVelocityY * t;
        } else {
            t = (nPositionY - Paddle.HEIGHT - Ball.HEIGHT/2)/-nVelocityY;
            collideX = nPositionX + nVelocityX * t;
            collideY = Paddle.HEIGHT + Ball.HEIGHT/2;
        }
    } else if (nVelocityX < 0 && nVelocityY > 0) { //nVelocityY can never be 0;
        if ((nPositionX - Ball.WIDTH/2)/-nVelocityX < (Pong.HEIGHT - nPositionY - Paddle.HEIGHT - Ball.HEIGHT/2) /nVelocityY) {
            t = (nPositionX - Ball.WIDTH/2)/-nVelocityX;
            collideX = Ball.WIDTH/2;
            collideY = nPositionY + nVelocityY * t;
        } else {
            t = (Pong.HEIGHT - nPositionY - Paddle.HEIGHT - Ball.HEIGHT/2)/nVelocityY;
            collideX = nPositionX + nVelocityX * t;
            collideY = Pong.HEIGHT - Paddle.HEIGHT - Ball.HEIGHT/2;
        }

    } else if (nVelocityX >=0 && nVelocityY < 0) {
        if (nVelocityX == 0) {
            t = (nPositionY - Paddle.HEIGHT - Ball.HEIGHT/2)/-nVelocityY;
            collideX = nPositionX;
            collideY = Paddle.HEIGHT + Ball.HEIGHT/2;
        } else {
            if ((Pong.WIDTH - nPositionX - Ball.WIDTH/2)/nVelocityX < (nPositionY - Paddle.HEIGHT - Ball.HEIGHT/2) /-nVelocityY) {
                t = (Pong.WIDTH - nPositionX - Ball.WIDTH/2)/nVelocityX;
                collideX = Pong.WIDTH - Ball.WIDTH/2;
                collideY = nPositionY + nVelocityY * t;
            } else {
                t = (nPositionY - Paddle.HEIGHT - Ball.HEIGHT/2)/-nVelocityY;
                collideX = nPositionX + nVelocityX * t;
                collideY = Paddle.HEIGHT + Ball.HEIGHT/2;
            }
        }
    } else {
        if (nVelocityX == 0) {
            t = (Pong.HEIGHT - nPositionY - Paddle.HEIGHT - Ball.HEIGHT/2)/nVelocityY;
            collideX = nPositionX;
            collideY = Pong.WIDTH - Paddle.HEIGHT - Ball.HEIGHT/2;
        } else {
            if ((Pong.WIDTH - nPositionX - Ball.WIDTH/2)/nVelocityX < (Pong.HEIGHT - nPositionY - Paddle.HEIGHT - Ball.HEIGHT/2) /nVelocityY) {
                t = (Pong.WIDTH - nPositionX - Ball.WIDTH/2)/nVelocityX;
                collideX = Pong.WIDTH - Ball.WIDTH/2;
                collideY = nPositionY + nVelocityY * t;
            } else {
                t = (Pong.HEIGHT - nPositionY - Paddle.HEIGHT - Ball.HEIGHT/2)/nVelocityY;
                collideX = nPositionX + nVelocityX * t;
                collideY = Pong.HEIGHT - Paddle.HEIGHT - Ball.HEIGHT/2;
            }
        }
    }
```

```
    var oldt = t;
    var oldvx = ball.vx;
    var oldvy = ball.vy;

    t = t - delay/1000 * Pong.FRAME_RATE;

    ball.vx = (collideX - ball.x)/t;
    ball.vy = (collideY - ball.y)/t;

    console.log("t: " + oldt + " ===> " + t);
    console.log("vx: " + nVelocityX + " ===> " + ball.vx);
    console.log("vy: " + nVelocityY + " ===> " + ball.vy);
}
```

When the computeBallVelocity function is called, the client computes t, the time it takes from the position given from the server to the next collision point (which might be the paddle or the wall).

The client will apply local perception filter to move the ball to the next collision point with a different t, say t'. Their relationship is as follows:

t' = t - average delay

The average delay here will be delay/1000 * Pong.FRAME_RATE in our code, because we need to coordinate the units.

## 4. Tolerable Network Delay

With all three techniques above implemented, the game is able tolerate a latency as much as 550ms.