Game Controller Report

1. **Description and explanations for the generated functions**
   1. **Overview of functions**

The functions developed for this project were designed to interact seamlessly with the game’s mechanics, ensuring game completion for levels 0, 1 and 2. These functions include:

* **Connect to server function:** Sets up the communication between the game and the game controller via tcp.
* **Communication function:** With this function the game controller receives data from the game in real time
* **Game interaction function:** Handles real-time interactions with the game, such as figuring out the current game level, and calculating cursor position in order to finish the game.
* **Disconnect from server function:** Closes the communication between the game and the game controller.
  1. **Function explanations**

Connect to server function – ConnectToServer():

* Follows a standard tcp client-server communication implementation
* Ip address is the standard localhost address and the port number is 6000
* If the connection is successfull, it calls the ReceiveDataAsync method

Communication function – ReceiveDataAsync(CancellationToken cancellationToken):

* Listens to incoming data packages from the game (packages have a JSON structure)
* Deserializes the data packages and calls the SolveTheGame method to interact with the game based on the received data

Game interaction function – SolveTheGame():

* Detects the current game level
* Calls the appropriate game solving method for the current level (SolveLevel0, SolveLevel1, or SolveLevel2)
* These three methods are responsible for moving the mouse cursor

Disconnect from server function – DisconnectFromServer():

* Closes the communication
* Frees up the allocated resources

1. **Description of interactions with the game**
   1. **Game-controller interaction**

This controller acts as an automated game solver for the first three levels of the game. It calculates the position of the cursor in real time based on the red dot postion, speed or acceleration, so the red dot reaches the designated target.

* 1. **Game responses to controller inputs**

Because the game controller can only interact with the mouse cursor position, this is considered the only game input. When the game controller calculates a new cursor position and it moves the mouse cursor, the game changes the red dot position based on a newly calculated position, speed or acceleration.

* 1. **Challenges in interaction design**

The only challenge in interaction design for this project was figuring out which pieces of data were relevant and how to bundle them together. After I managed to understand which parameters to export, the solution was to send them as a JSON package.

1. **Description of the considered controller and parameter choice**
   1. **Controller design**

* The controller for level 0 is only based on target position. The game controller receives the target position and moves the mouse cursor accordingly.
* The controller for level 1 is a P controller (proportional controller)
* The controller for level 2 is a PD controller (proportional-derivative controller)
  1. **Parameter experiments and justifications**

The parameters for this game controller are kp and kd:

* kp is the proportional gain, which determines the strength of the response to the current error
* kd is the derivative gain, which determines the strength of the response to the rate of change of the error (or the system's velocity).

Experiments (maxSpeed = 100, maxAcceleration = 100):

* Level 0 solver doesn’t use kp and kd parameters. Solving time is fixed for every target position.
* Level 1 uses only the kp parameter. After some testing, it turned out the optimal value for kp is 1. Any value lower than 1 results in the red dot reaching the target slower. Any value higher than 1 results in overshooting, which means the game controller takes longer to finish the level (due to a small, fixable overshoot), or it won’t finish the game at all (constant oscilation around the target.
* Level 2 uses both kp and kd parameters. The optimal value for kp is 1 and the optimal value for kd is 1.55. Any value lower than 1.55 results in oscilation. Any value higher than 1.55 results in slower solving time.

1. **Annex: Functions code**

public async Task ConnectToServer()

{

try

{

string serverIP = "localhost";

int port = 6000;

client = new TcpClient();

await client.ConnectAsync(serverIP, port);

stream = client.GetStream();

cancellationTokenSource = new CancellationTokenSource();

\_ = ReceiveDataAsync(cancellationTokenSource.Token);

DataReceived?.Invoke("Connected...");

}

catch (Exception ex)

{

DataReceived?.Invoke($"Error: {ex.Message}");

}

}

public async Task ReceiveDataAsync(CancellationToken cancellationToken)

{

try

{

while (!cancellationToken.IsCancellationRequested)

{

if (stream != null && stream.CanRead)

{

byte[] buffer = new byte[1024];

int bytesRead = await stream.ReadAsync(buffer, 0, buffer.Length);

if (bytesRead > 0)

{

string receivedMessage = Encoding.UTF8.GetString(buffer, 0, bytesRead);

try

{

data = JsonConvert.DeserializeObject<Data>(receivedMessage);

if (startSolving == false)

{

stopWatch.Start();

startSolving = true;

}

SolveTheGame();

}

catch (JsonException ex)

{

Console.WriteLine("Error parsing JSON: " + ex.Message);

}

}

}

}

}

catch (Exception ex)

{

DataReceived?.Invoke($"Error: {ex.Message}");

}

}

public void SolveTheGame()

{

try

{

DataReceived?.Invoke(Math.Floor(data.PositionXY.PositionX).ToString() + " " + Math.Floor(data.PositionXY.PositionY).ToString());

}

catch (Exception ex)

{

Console.WriteLine("Error displaying position: " + ex.Message);

}

switch (data.Level)

{

case 1:

SolveLevel0();

break;

case 2:

SolveLevel1();

break;

case 3:

SolveLevel2();

break;

}

}

private void SolveLevel0()

{

double mouseX = data.TargetXY.TargetX;

double mouseY = data.TargetXY.TargetY;

MoveMouseAbsolute(mouseX, mouseY);

}

private void SolveLevel1()

{

currentX = data.PositionXY.PositionX;

currentY = data.PositionXY.PositionY;

double errorX = data.TargetXY.TargetX - currentX;

double errorY = data.TargetXY.TargetY - currentY;

double controlX = kp \* errorX;

double controlY = kp \* errorY;

double maxSpeed = 100.0;

double controlMag = Math.Sqrt(controlX \* controlX + controlY \* controlY);

if (controlMag > maxSpeed)

{

double scale = maxSpeed / controlMag;

controlX \*= scale;

controlY \*= scale;

}

double mouseX = 1920 / 2 + controlX \* (1920 / 2) / maxSpeed;

double mouseY = 1080 / 2 + controlY \* (1080 / 2) / maxSpeed;

MoveMouseAbsolute(mouseX, mouseY);

}

private void SolveLevel2()

{

currentX = data.PositionXY.PositionX;

currentY = data.PositionXY.PositionY;

double velX = data.VelocityXY.VelocityX[1];

double velY = data.VelocityXY.VelocityY[1];

double errorX = data.TargetXY.TargetX - currentX;

double errorY = data.TargetXY.TargetY - currentY;

double controlX = kp \* errorX - kd \* velX;

double controlY = kp \* errorY - kd \* velY;

double maxAcceleration = 100.0;

double controlMag = Math.Sqrt(controlX \* controlX + controlY \* controlY);

if (controlMag > maxAcceleration)

{

double scale = maxAcceleration / controlMag;

controlX \*= scale;

controlY \*= scale;

}

double mouseX = 1920 / 2 + controlX \* (1920 / 2) / maxAcceleration;

double mouseY = 1080 / 2 + controlY \* (1080 / 2) / maxAcceleration;

MoveMouseAbsolute(mouseX, mouseY);

}

private void MoveMouseAbsolute(double mouseX, double mouseY)

{

Cursor.Position = new Point((int)mouseX, 1080 - (int)mouseY);

AddPoints(mouseX, mouseY);

}

private void AddPoints(double x, double y)

{

var time = (stopWatch.ElapsedMilliseconds / 0.1) \* 0.1;

lineSeriesX.Points.Add(new DataPoint(time, x));

lineSeriesY.Points.Add(new DataPoint(time, y));

}

public void DisconnectFromServer()

{

cancellationTokenSource?.Cancel();

stream?.Close();

client?.Close();

cancellationTokenSource?.Dispose();

}