Online Multiplayer mit SFML

In dem neuem Template (2.3) sind zwei neue Klassen eingefügt. NetworkClient und NetworkServer. Diese beiden Klassen enthalten eine simple Implementierung eines Client-Server-Modells.

Wir konzentrieren uns hier nur auf den NetworkClient:

Der Network-Client besitzt folgende relevante Methoden/Felder:

**ConnectTo:**

NetworkClient.connectTo("127.0.0.1", 8888);

Der Client versucht sich mit einem Server mit der gegebenen IP Adresse und dem gegebenen Port zu verbinden. Falls der Versuch fehlschlägt, passiert nichts.

**isConnected:**

bool isConnected = NetworkClient.isConnected

Ein Boolwert, der angibt ob eine funktionierende Verbindung besteht oder nicht.

**read:**

List<String> responses = NetworkClient.read();

Gibt eine Liste aller empfangen Nachrichten vom Server zurück.

Die Liste wird danach geleert, also wird keine Nachricht doppelt auftauchen.

**send:**

NetworkClient.send("Hello Server");

Sendet eine Nachricht an den Server, wenn eine Verbindung besteht.

Diese wollen wir nun nutzen, um uns mit einem Server zu verbinden und zu kommunizieren.

Zuerst, versuchen wir eine Verbindung mit dem Server aufzubauen. Die IP-Adresse eines Computers in einem Netzwerk kann über den CommandLine Befehl "ipconfig" herausgefunden werden.

Der Port sollte vom Server festgelegt sein. In der AG werden wir immer den Port 8888 benutzen.

public override void setup()

{

// Initialisere Beispiel (das führt automatisch den Konstruktor aus)

example = new Entity(this);

NetworkClient.connectTo("xxx.xxx.xxx.xxx", 8888);

}

Sobald eine Verbindung besteht, können wir über NetworkClient.read() die Nachrichten vom Server empfangen.

public override void loop()

{

List<String> responses = NetworkClient.read();

example.loop();

example.draw();

}

Nun muss sich auf einen Verschlüsselungscode von Informationen geeinigt werden. Ähnlich wie bei den Savegames(siehe #13.2) müssen wir die Entity Informationen in einen String schreiben.

Wir nehmen also die Positionsdaten unserer Entity und schreiben sie in einen String:

String position = example.graphic.Position.X.ToString() + "x" + example.graphic.Position.Y.ToString();

Das gleiche tun wir für Rotation und Size.

public override void loop()

{

List<String> responses = NetworkClient.read();

String position = example.graphic.Position.X.ToString() + "x" + example.graphic.Position.Y.ToString();

String rotation = example.graphic.Rotation.ToString();

String size = example.graphic.Size.X.ToString() + "x" + example.graphic.Size.Y.ToString();

example.loop();

example.draw();

}

Jetzt fügen wir die Strings mit Trennzeichen zusammen in einen langen String. Diesen senden wir dann mit NetworkClient.send() an den Server.

public override void loop()

{

List<String> responses = NetworkClient.read();

String position = example.graphic.Position.X.ToString() + "x" + example.graphic.Position.Y.ToString();

String rotation = example.graphic.Rotation.ToString();

String size = example.graphic.Size.X.ToString() + "x" + example.graphic.Size.Y.ToString();

String entitiyData = position + "&" + rotation + "&" + size;

NetworkClient.send(entitiyData);

example.loop();

example.draw();

}

Jetzt müssen wir die Daten, die wir empfangen nur noch auswerten, dafür müssen wir die Daten aus dem String lesen und daraus neue Entitys erzeugen.

Die "responses" Liste enthält möglicherweise 0 Nachrichten, falls der Server langsam ist, oder mehr als 1 wenn der Server schneller ist. Wir müssen alle Möglichkeiten abfangen.

Als erstes Erstellen wir eine Liste von RectangleShapes. Diese enthält dann die Entitys der anderen Spieler auf dem Server.

// Deklariere hier Objekte oder Klassenvariablen!

Entity example;

List<RectangleShape> shapes;

public override void setup()

{

// Initialisere Beispiel (das führt automatisch den Konstruktor aus)

example = new Entity(this);

shapes = new List<RectangleShape>();

NetworkClient.connectTo("xxx.xxx.xxx.xxx", 8888);

}

Wir wollen nur etwas an unserem Spiel ändern, wenn wir Nachrichten erhalten haben. Wenn responses also keine Nachricht enthält, tuen wir nichts.

public override void loop()

{

List<String> responses = NetworkClient.read();

if (responses.Count > 0)

{

}

Falls "responses" mehr als eine Nachricht enthält, interessiert uns nur die neuste Nachricht. Also lesen wir nur diese aus:

if (responses.Count > 0)

{

//Nehme nur die letzte/neuste Nachricht aus responses

String message = responses[responses.Count - 1];

}

Jetzt müssen wir die einzelnen EntityDaten voneinander trennen. Dafür müssen wir das Trennzeichen kennen. Hier nehmen wir an, es wäre eine "+". Mit split('+') erzeugen wir einen Array der die EntityDaten enthält. Diese müssen wir jetzt nurnoch alle einzeln abarbeiten mit einer for-schleife.

if (responses.Count > 0)

{

//Nehme nur die letzte/neuste Nachricht aus responses

String message = responses[responses.Count - 1];

//Spalte die message in einzelne Entitydaten auf

String[] data = message.Split('+');

//Lese jede Entitydata aus

for (int i = 0; i < data.Length; i++)

{

}

}

Jetzt spalten wir die EntityData in ihre einzelnen Komponenten auf(wir machen das Rückwärts, was wir beim senden gemacht haben).

//Lese jede Entitydata aus

for (int i = 0; i < data.Length; i++)

{

String posData = data[i].Split('&')[0];

String rotData = data[i].Split('&')[1];

String sizeData = data[i].Split('&')[2];

String posX = posData.Split('x')[0];

String posY = posData.Split('x')[1];

String sizeX = sizeData.Split('x')[0];

String sizeY = sizeData.Split('x')[1];

}

Jetzt brauchen wir ein RectangleShape, das diese Informationen darstellt.

for (int i = 0; i < data.Length; i++)

{

String posData = data[i].Split('&')[0];

String rotData = data[i].Split('&')[1];

String sizeData = data[i].Split('&')[2];

String posX = posData.Split('x')[0];

String posY = posData.Split('x')[1];

String sizeX = sizeData.Split('x')[0];

String sizeY = sizeData.Split('x')[1];

RectangleShape shape = new RectangleShape();

shape.Position = new Vector2f(float.Parse(posX),float.Parse(posY));

shape.Position = new Vector2f(float.Parse(posX),float.Parse(posY));

shape.Rotation = float.Parse(rotData);

shape.Size = new Vector2f(float.Parse(sizeX), float.Parse(sizeY));

}

Jetzt fügen wir dieses RectangleShape unserer Liste hinzu. Allerdings sollten wir unsere Liste immer vor der for-schleife Clearen, damit keine RectangleShapes doppelt gezeichnet werden.

shapes.Clear();

//Lese jede Entitydata aus

for (int i = 0; i < data.Length; i++)

{

String posData = data[i].Split('&')[0];

String rotData = data[i].Split('&')[1];

String sizeData = data[i].Split('&')[2];

String posX = posData.Split('x')[0];

String posY = posData.Split('x')[1];

String sizeX = sizeData.Split('x')[0];

String sizeY = sizeData.Split('x')[1];

RectangleShape shape = new RectangleShape();

shape.Position = new Vector2f(float.Parse(posX),float.Parse(posY));

shape.Rotation = float.Parse(rotData);

shape.Size = new Vector2f(float.Parse(sizeX), float.Parse(sizeY));

shapes.Add(shape);

}

Jetzt fehlt nurnoch, dass wir unsere Liste von RectangleShapes mit einer For-Schleife zeichnen.

for (int i = 0; i < shapes.Count; i++)

{

draw(shapes[i]);

}

example.loop();

example.draw();

(Damit man etwas sieht, sollten wir unsere Entity bewegen können.)