Dokumentation von Mutiplayer Autorennspiel “Drift”

Geliederung

1. Einleitung
2. Physik und Bewegung
3. Netzwerk und Sycronisierung
4. Graphik und Modell
5. UI und Interaktion
6. Fazit
7. Einleitung

Das Spiel “Drift” ist ein Autorennspiel für mehrere Spieler in eine LAN. Der Spiel ist mit Unity-Engine (https://unity3d.com) programmiert. Die Modelle wie die Häuse und Autos sind aus Unity-Assetsstore.

1. Physik

Die Rechnungen der Physik des Autos sind mit dreidimensionale Vectoren(Vector3 [1]) in Script CarController. Ein Auto hat immer paar physikalische Attribute:

Masse (float mass)

Bewegungsrichtung und Geschwindigkeit (Vector3 velocity)

Position (Vector3 position)

Forwarts Richtung (Vector3 forward)

Aufwarts Richtung (Vector3 upward)

Ein Auto hat auch paar Attribute für die Bewegung (Ein AnimationCurve [2] ist ein kurve mit Graphik bearbeiten kann, damit kann man ein Wert von dem kurve auslesen):

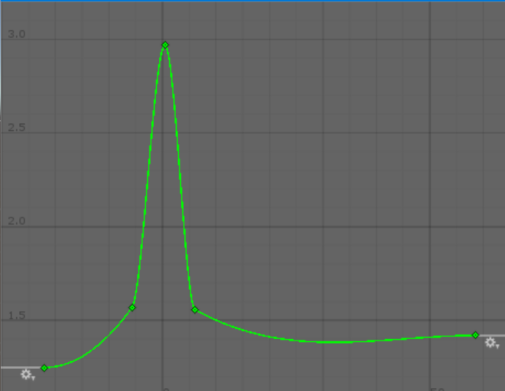
Beschleunigung/rückwarts Beschleunigung (float acceleration/backAcceleration)

Trägheit der Beschleunigung (AnimationCurve accelerationDrag)

Beschleunigung der Drehung (AnimationCurve torque)

Trägheit der Rotation(float rotateDrage)

Die accelerationDrag und torque siehen so aus:

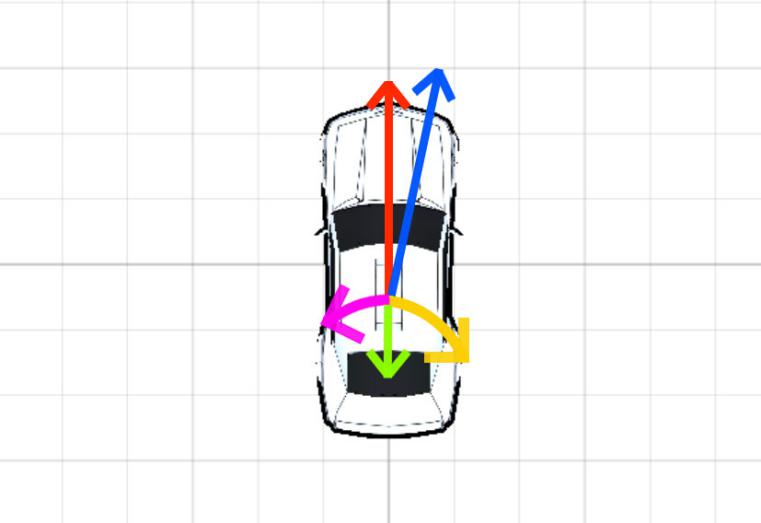
 

Die Rechnung der Bewegung ist:

velocity += InputY() \* forward \* acceleration \* deltaTime (deltaTime ist die Zeit jeder Frame, InputY/InputY ist die horizontale/vertikale Input des Spieler zwischen -1 und 1)

AddTorque (InputX() \* upward \* torque.Evaluate(velocity.magnitude) \* deltaTime)

(AddTorque() ist ein Funktion, die ein Geschwindigkeit zu dem Rotation des Autos machen. Evaluate() ist nimmt den Wert der X-Achse und gibt den Wert der Y-Achse der Kurve zurück. magnitude ist die Länge einer Vector)



Rot: anfangs velocity, Grün: accelerationDrag, Gelb: torque, Lila: torqueDrag, Blau: neue velocity

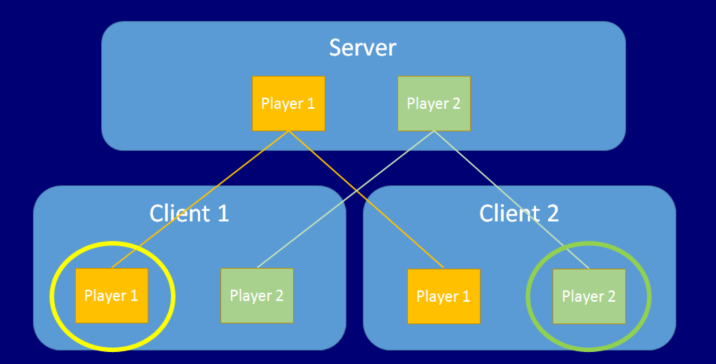
Nach jeder Frame rechnen wir die Reibung.

Velocity \*= (1-accelerationDrag.Evaluate(velocity.z)) \* deltaTime

angularVelocity \*= (1-rotateDrag) \* deltaTime

1. Netzwerk und Syncronisierung

Ich habe die Netzwerk-Module Unet für meine Projekt benutzt. Unet ist ein high-level-scripting-API [3]. In Unet ist ein Client von alle Spieler auch als ein Server. Unet controliert den Netzwerk mit eine NetworkManager [4]. Unet kann auch die Informationen und Befehle zwischen die Spieler schicken.



Server-Client Diagramm für UNet. Die Spieler in der Kreise sind die Spiel von dem Client krontroliert wird[5]

Wenn die Spieler anfangen in ein LAN zu spielen, die wissen nicht die IP-Addresse von der andere Spieler, daswegen brauchen wir eine Script, die IP zu alle Port in diese Lan broadcastet, und die IP von dem Anderen zu empfangen.

In die Hauptmenü (MainMenu) ist ein Script heißt ConnectionManager, welche NetworkDiscovery [6] erbt. Der hat vier Funktionen: StartServer(), StartClient(), CloseConnection() und überschriebende Funktion OnReceivedBroadcast(). Die erste drei Funktionnen können ein Server oder Client für die IP-Broadcast erstellen und schließen, die letzte Funktion kann die IP Adresse empfangen und behandeln.

In die haupte Szene (Town) habe ich zwei Scripts für die Netzwerk, eine ist NetworkManager. Die andere ist AutoConnection, welche mit dem API von NetworkManager zu mit dem IP, die wir vorher in Hauptmenü empfangt, der Server verbindet, wenn der Spieler als Client ist, oder ein Server erstellt, wenn der Spieler als Server ist.

Die CarController erbt NetworkBehavior [7]. die kontroliert nicht nur die Physik der Spieler auch die Sycronisierung der Positionen und Bewegungen. Es hat zwei Funktion heißt RpcSyncPos() mit Attribute [ClientRpc][8] und CmdSendServer() mit Attribute [Command][9], CmdSendServer() schickt die Position, Rotation, Geschwindigkeit und ID der Spieler von einem Client zu dem Server, und auf dem Server wird RpcSyncPos() aufgerufen und die Informationen zu die andere Client weiter schicken. Wenn die Andere Client die Informationen bekommen, dann wird LerpPosition() aufgerufen, die die Position der Spieler mit Lerp() [10] flüssig auf die neue Position stellen.

1. Graphik und Modell

Die Modelle sind alle kostenlose Materiale vom AssetStore [11].



[12]



MainMenu



Town(haupt Szene)

Quelle:

1. <https://docs.unity3d.com/ScriptReference/Vector3.html>
2. <https://docs.unity3d.com/ScriptReference/AnimationCurve.html>
3. <https://docs.unity3d.com/Manual/UNet.html>
4. <https://docs.unity3d.com/ScriptReference/Networking.NetworkManager.html>
5. <https://docs.unity3d.com/uploads/Main/NetworkLocalPlayers.png>
6. <https://docs.unity3d.com/ScriptReference/Networking.NetworkDiscovery.html>
7. <https://docs.unity3d.com/ScriptReference/Networking.NetworkBehaviour.html>
8. <https://docs.unity3d.com/ScriptReference/Networking.ClientRpcAttribute.html>
9. <https://docs.unity3d.com/ScriptReference/Networking.CommandAttribute.html>
10. <https://docs.unity3d.com/ScriptReference/Vector3.Lerp.html> und <https://docs.unity3d.com/ScriptReference/Quaternion.Lerp.html>
11. <https://assetstore.unity.com/>