

Zwischenpräsentation

Von Xue Ming Chen und Moncif Souhali

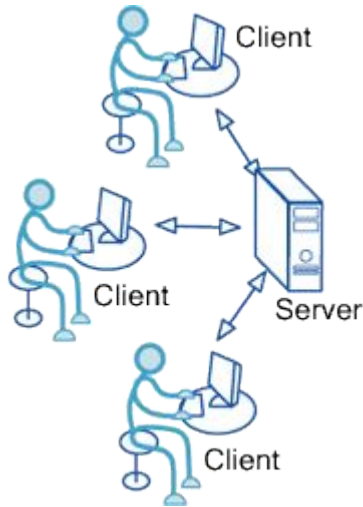
Unser Spiel

- 2D jump-and-run
- Multiplayer
- intelligente Bosse
- Wandsprung, Dash-Angriff
- übernehmbare Fähigkeiten (Feuer, Eis, Wasser, Stein)

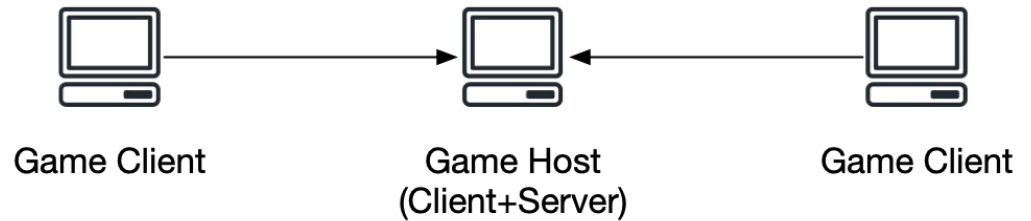


Schwerpunkt 1: Multiplayer Architekturen

Server - Client



Host - Client



Netcode - Unity Multiplayer Bibliothek

Netcode for GameObjects

- sehr einfach zu benutzen
- gut geeignet für Co-op oder slow-paced player vs player Spiele

Netcode for Entities (DOTS)

- Einarbeitung in DOTS notwendig
- gut geeignet für fast-paced player vs player Spiele

Funktionsweise

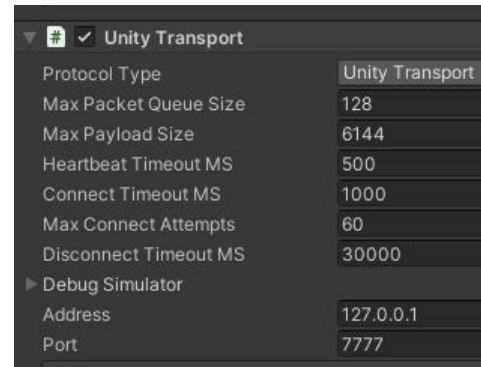
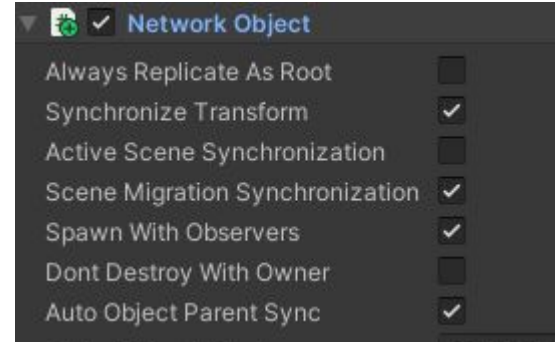
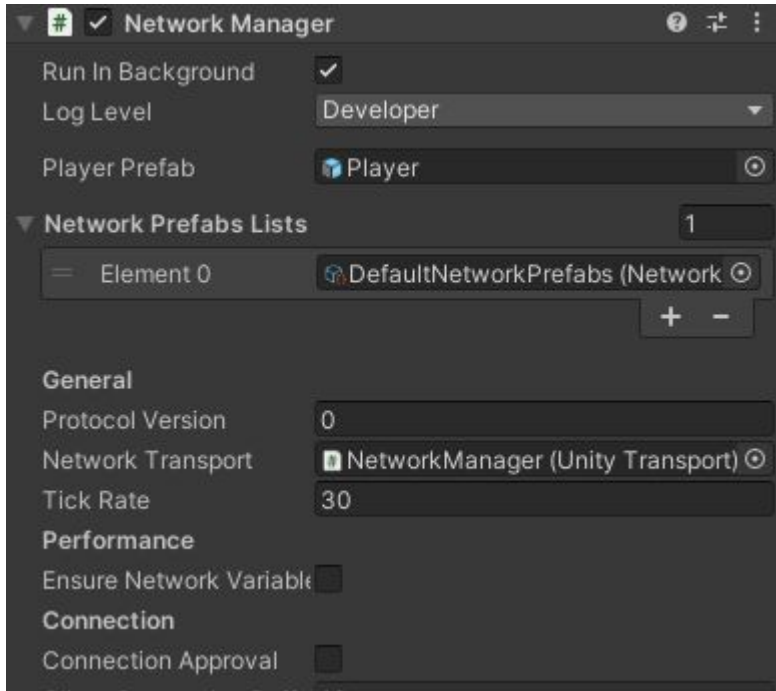
- Der Host hostet unter seiner IP-Adresse einen Raum
- Der Client verbindet sich mit Host
- Alle für Daten notwendig für Synchronisierung werden umher geschickt

Beispieldaten:

Spielerposition/ -rotation/ -animation, Fähigkeiten, Interaktionen mit der Umgebung(Schalter betätigen), Interaktionen untereinander(Heilung), gemeinsamer Szenenwechsel, Kombo-Attacken auf den Gegner

Implementierung

Network Manager (Transport Layer)



Verbindung aufbauen

```
hostButton.onClick.AddListener(call: () => {  
    ipAddress = IPAddressInput.text;  
    if (ipAddress == "") {  
        ipAddress = "127.0.0.1";  
    }  
    networkManager.GetComponent<UnityTransport>().ConnectionData.Address=ipAddress  
    NetworkManager.Singleton.StartHost();  
});
```

Host

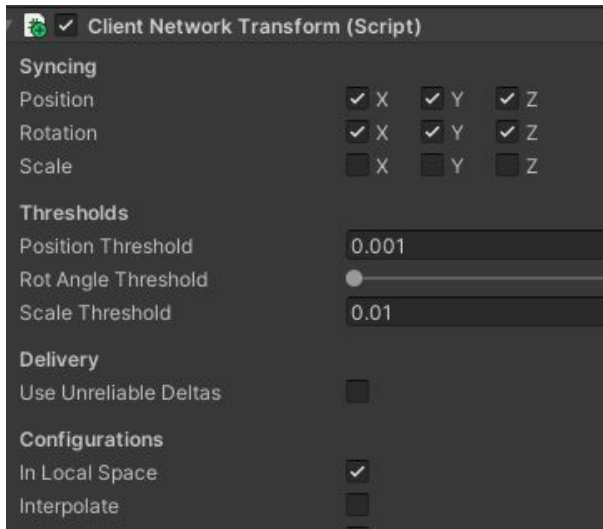
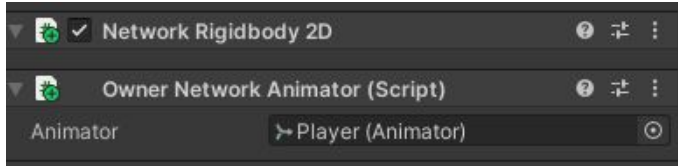
Client

Enter IP Address...

Leave



Datensynchronisierung

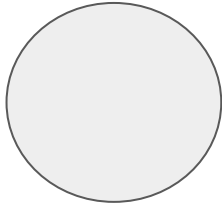


```
void Update()
{
    if (!IsOwner && SceneManager.GetActiveScene().name == "Lobby") {
        return;
    }
}
```

```
public class ClientNetworkTransform : NetworkTransform
{
    [Moncif]
    protected override bool OnIsServerAuthoritative() {
        return false;
    }
}
```


Exkurs: Automaten

Zustände :



Transition: Kante zwischen zwei Zustände

Übergang zwischen zwei Zustände



Behavior Tree

Weiterentwicklung von Automaten

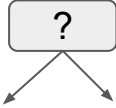
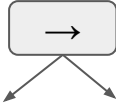

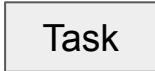
- hierarchische Strukturen
- Entkoppeln der Zustände in einzelne Aufgaben

Ähnlich zu anderen Baum Strukturen

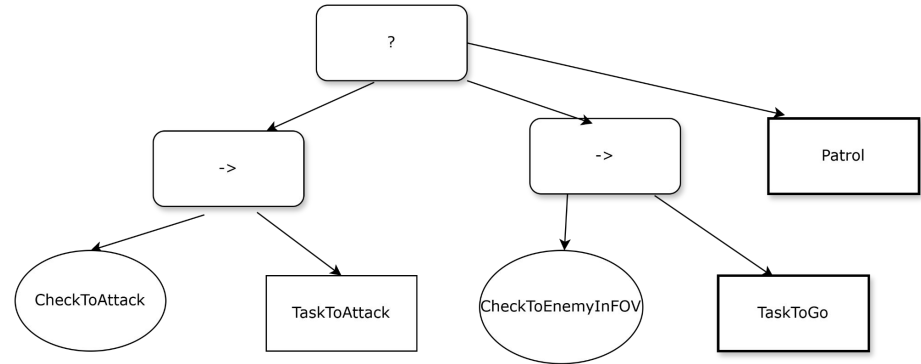
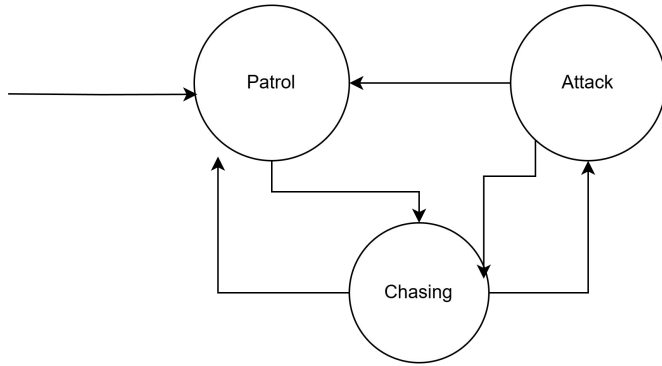
- Wurzel, Knoten, Kinder , Eltern
- Knoten haben States: Running , Success, Failure
- arbeitet mit Prioritäten

Verwendung: Spieleentwicklung, Robotik und Simulationen

Knotenarten

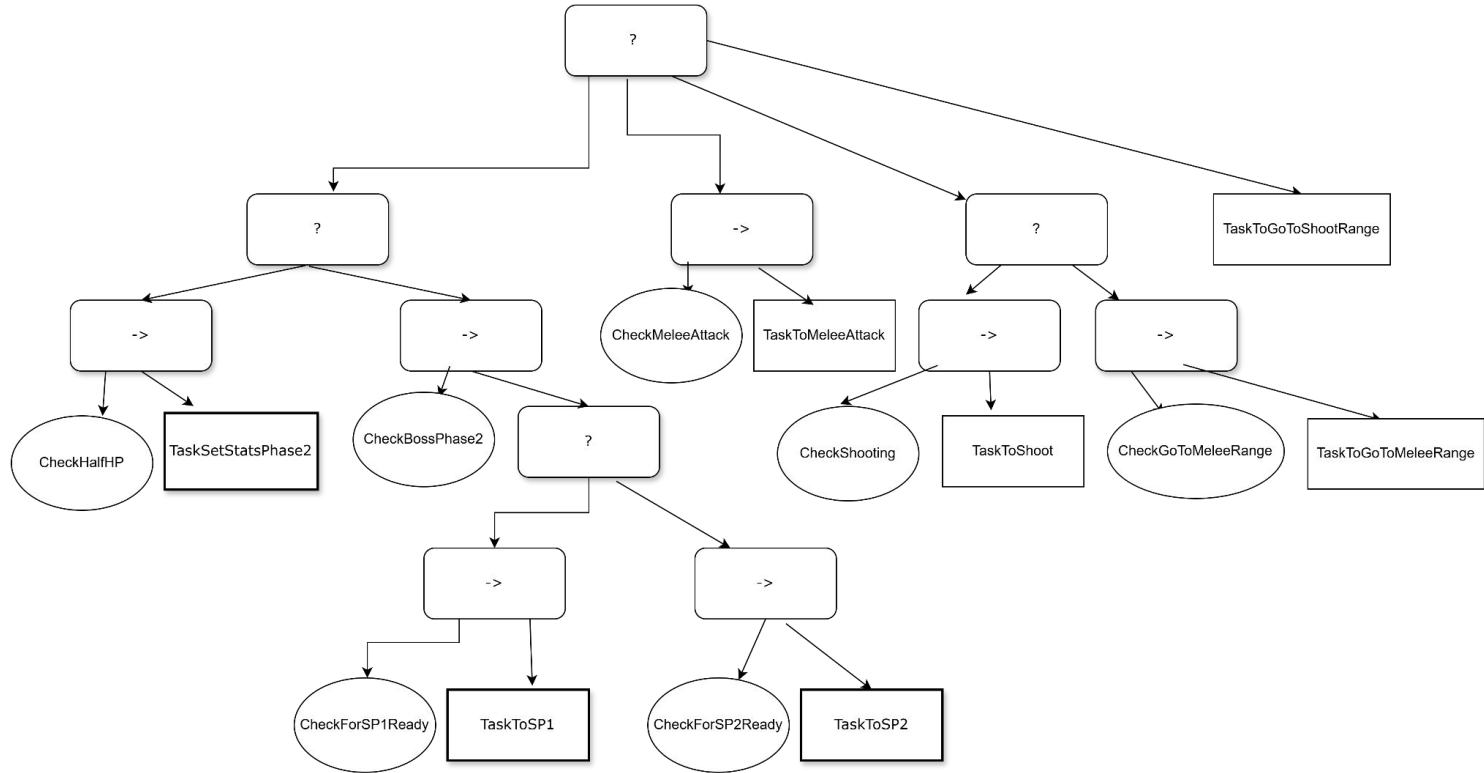
Knoten	Success	Failure	Running	Notation
Fallback/Selector	Ein Kind gibt Success zurück	Alle Kinder geben Failure zurück.	Ein Kind in Ausführung	
Sequence	Alle Kinder geben Success zurück.	Ein Kind gibt Failure zurück	Ein Kind in Ausführung	
Condition	Condition erfüllt	Condition nicht erfüllt	/	
Task/Action	Erfolgreiche Ausführung	Ausführung erfolglos	in Ausführung	

Vergleich Automaten und Behavior Tree



Behavior Tree

Bsp.:



Ausblick

	Bis 23.05	Bis 30.05	Bis 06.06	Bis 13.06	Bis 20.06	Bis 27.06	Bis 04.07	Bis 11.07	Bis 18.07
<u>Aufgabe</u>	<u>1. Woche</u>	<u>2. Woche</u>	<u>3. Woche</u>	<u>4. Woche</u>	<u>5. Woche</u>	<u>6. Woche</u>	<u>7. Woche</u>	<u>8. Woche</u>	<u>9. Woche</u>
<u>Charakter</u>									
<u>Leveldesign</u>									
<u>Texturen</u>									
<u>Multiplayer</u>									
<u>Gegner</u>									
<u>Bosse (KI)</u>									
<u>Fähigkeiten</u>									
<u>Animationen</u>									
<u>Puzzle</u>									
<u>Sound</u>									
<u>Partikeleffekte</u>									
<u>Bug-Fixing</u>									

Puffer