SEMINARARBEIT

im Studiengang XXX

Lehrveranstaltung XXX

Arbeitstitel

Arbeitstitel

Ausgeführt von: Titel Vorname Name Titel  
Personenkennzeichen:

BegutachterIn: Titel Vorname Name Titel

Ort, Datum

Inhaltsverzeichnis

[1 Aufgaben 3](#_Toc367364234)

[1.1 Erweiterung der Gameplay Rules 3](#_Toc367364235)

[1.1.1 add a player login with name and password (simple hash is enough) 3](#_Toc367364236)

[1.1.2 add a persistent 'known player management' to the server 3](#_Toc367364237)

[1.1.3 allow up to 16 players login to the game server simultaneously 3](#_Toc367364238)

[1.1.4 only 4 players can play at one time / the rest become spectators 3](#_Toc367364239)

[1.1.5 once one player dies the next spectator becomes a active player 3](#_Toc367364240)

[1.1.6 the dead player becomes a spectator 3](#_Toc367364241)

[1.1.7 add a 'kill' count for each player 3](#_Toc367364242)

[1.1.8 add the 'kill-count' and the name of the active (not spectating) players to the client UI 3](#_Toc367364243)

[1.2 Replication 3](#_Toc367364244)

[1.2.1 use compressed datagrams for transmitted state structures 3](#_Toc367364245)

[1.2.2 do not continuously send projectile updates 3](#_Toc367364246)

[1.2.3 implement client side interpolation 4](#_Toc367364247)

[1.2.4 implement client side prediction for the controlling client 4](#_Toc367364248)

[1.2.5 check if it is possible to implement server side lag compensation 4](#_Toc367364249)

[1.2.6 it might make sense to invent one additional replicated object called 'GameState' 4](#_Toc367364250)

[1.3 Additional functionality (bonus points) 4](#_Toc367364251)

[1.3.1 notify spectatores about how many rounds they have to wait 4](#_Toc367364252)

[1.3.2 add chat functionality to the game 4](#_Toc367364253)

# 

# Aufgaben

## Erweiterung der Gameplay Rules

### add a player login with name and password (simple hash is enough)

UI-Controls als Properties in TankVsTankPluginContent definieren:

class TankVsTankPluginContent : public PluginLayerContent {

**CC\_SYNTHESIZE(CCControlEditBox\*, \_ctrlName, CtrlName)**

**CC\_SYNTHESIZE(CCControlEditBox\*, \_ctrlPassword, CtrlPassword)**

**CC\_SYNTHESIZE(CCControlEditBox\*, \_ctrlMessage, CtrlMessage)**

}

Die Login-UI in addGameContentUI hinzufügen:

void TankVsTankPluginContent::addGameContentUI( SLSize idx, PeerNode\* peerNode, TankVsTankGameplayLayer\* parentLayer )

{

**if(peerNode->getPeer()->getTopology() == CLIENT)**

**{**

**\_ctrlName = ControlUtils::createEditBox("Name:", ctrlsPreferredSize);**

**ctrls->addObject(\_ctrlName);**

**\_ctrlPassword = ControlUtils::createEditBox("Password:", ctrlsPreferredSize);**

**ctrls->addObject(\_ctrlPassword);**

**\_ctrlMessage = nl::ControlUtils::createEditBox("Message:", ctrlsPreferredSize);**

**ctrls->addObject(\_ctrlMessage);**

**}**

}

### add a persistent 'known player management' to the server

where player properties are:

name

password

killcount

numberOfGamesPlayed

hint: PersistentDictionary

### allow up to 16 players login to the game server simultaneously

Antwort auf meine Frage zur Umsetzung: am besten dazu mehrere Instanzen starten mit Visual Studio!

Ansonsten viele GUI changes erforderlich und debugging wird sehr heftig

### only 4 players can play at one time / the rest become spectators

### once one player dies the next spectator becomes a active player

### the dead player becomes a spectator

### add a 'kill' count for each player

### add the 'kill-count' and the name of the active (not spectating) players to the client UI

## Replication

### use compressed datagrams for transmitted state structures

Padding ist ein wichtiges Thema. Unterschiede ergeben sich durch Anordnung von Variablen in Strukturen oder aber auch durch das Zielsystem und den Compiler (sowie dessen Optimierungseinstellungen.

nlProtocolStructures.h

ControllerValues: 16 Bytes statt 24 Bytes -> 8 Bytes eingespart

|  |  |
| --- | --- |
| Original: 24 Bytes | Komprimiert: 16 Bytes |
| typedef struct TControllerValues  {  float \_leftRight;  float \_forwardBackward;  float \_shoot;  RakNet::NetworkID \_controlledReplicaNetworkId;  SLSize \_updateTick;  } ControllerValues; | typedef struct TCompressed\_ControllerValues  {  char \_leftRight;  char \_forwardBackward;  bool \_shoot;  RakNet::NetworkID \_controlledReplicaNetworkId;  SLSize \_updateTick;  } TCompressed\_ControllerValues; |

Dynamic2DActorDatagram 24 statt 32 Bytes -> (zuvor 28 bzw. mit Padding 32 Bytes) -> 8 Bytes möglich

|  |  |
| --- | --- |
| Original: 28 Bytes | Komprimiert: 24 Bytes TODO |
| typedef struct TDynamic2DActorDatagram  {  float \_x;  float \_y;  float \_fx;  float \_fy;  float \_lvx;  float \_lvy;  float \_avz;  } Dynamic2DActorDatagram; | typedef struct TCompressed\_Dynamic2DActorDatagram  {  short \_x;  short \_y;  float \_lvx;  float \_lvy;  float \_fx;  float \_fy;  float \_avz;  } TCompressed\_Dynamic2DActorDatagram; |

Dann noch in der BitStream.h die Compressed-Methoden für die jeweiligen Datentypen (eignet sich z.B. besonders gut für Quaternions) verwenden:

Anstatt:

bitStream.WriteAlignedBytes( (const unsigned char \*)&\_ctrlValues, sizeof(ControllerValues) );

Mit Kompression:

Compressed\_ControllerValues comValues;

comValues.\_forwardBackward = TCompressedFixpoint<float,char,8>::writeCompress(\_ctrlValues.\_forwardBackward , -1.0f, 1.0f );

comValues.\_leftRight = TCompressedFixpoint<float,char,8>::writeCompress(\_ctrlValues.\_leftRight , -1.0f, 1.0f );

comValues.\_shoot = TCompressedFixpoint<float,char,8>::writeCompress(\_ctrlValues.\_shoot , -1.0f, 1.0f );

comValues.\_updateTick = \_ctrlValues.\_updateTick;

comValues.\_controlledReplicaNetworkId = \_ctrlValues.\_controlledReplicaNetworkId;

bitStream.WriteAlignedBytes( (const unsigned char \*)&comValues, sizeof(Compressed\_ControllerValues));

### do not continuously send projectile updates

a projectile spawns with an initial position/orientation/velocity and lifetime

(that's it on the clientside)

### implement client side interpolation

### implement client side prediction for the controlling client

### check if it is possible to implement server side lag compensation

One way of compensating server side lag could be to **keep track of the game state on the client** itself and send its absolute state to the server. The **drawbacks** of this method are that a) the current game state maintenance is not in the single responsibility of the server anymore b) the game state handling needs to be remodeled and b) this method is prone to allow cheating, as clients can send whatever they like (e.g. new positions that vary greater than the maximum speed allows them to move). Therefore, in our opinion, this lag compensation is **impracticable**.

Another way could be **to buffer the game states** and calculate the new state based on the game state the client had at the time the action was performed (= minus the lag time). This has the benefit that the **client’s actions correspond to what the player sees**, e.g. firing a bullet and hitting the enemy leads to the death of the enemy. The drawback of this approach is that with the lag of the client the **enemy might have already moved**, but the player doesn’t see it yet. Therefore, the enemy would die because he was hit at the client’s calculations, although actually the enemy had already moved to another position which is safe (worst case from the dying player: he has moved behind a wall or a shield and like out of nowhere he dies due to the calculations with the old position at the other client). In our opinion, the drawback of this approach clears the benefit, which makes this implementation not lucrative enough.

To wrap it up, server-side lag compensation is definitely possible, but it doesn’t come without drawbacks, especially if you’re not fully investigating in this topic and make it a full-time project with just the focus on accomplishing an acceptable lag compensation. In our opinion the benefits of a non-professional server-side lag compensation cannot outperform the drawbacks, which is why we don’t intra- or extrapolate or buffer game states and leave server-side lag “as-is”.

### it might make sense to invent one additional replicated object called 'GameState'

## Additional functionality (bonus points)

### notify spectatores about how many rounds they have to wait

### add chat functionality to the game