Hochschule für Technik und Wirtschaft Berlin
University of Applied Sciences

UDK: Vorlesung 2 – Advanced Topics

Tobias Müller

GT4: Games & Simulation Technology

Wintersemester 2012/2013

Gliederung

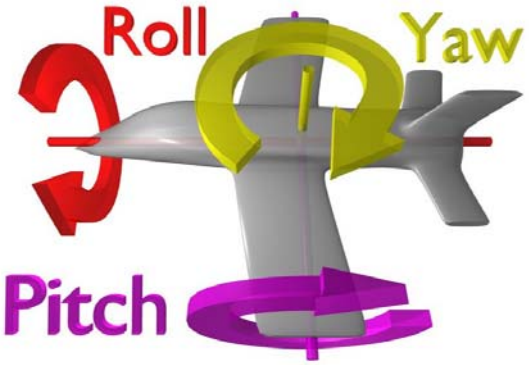
- UnrealScript (basiert auf voriger Vorlesung)
- Physics
- AI

Hochschule für Technik und Wirtschaft Berlin
University of Applied Sciences

UnrealScript

UnrealScript

- `Log(„Hello World“);
  - Erfordert Parameter „-log“ bei Ausführung der UDK.exe
- String-Concatination:
  - `Log(„Name: “$Pawn.Name)
- Rotationen im UDK
  - Yaw, Pitch & Roll



  - Rotation.Yaw = 90 * DegToUnrRot;
  - Weitere: RadToDeg, RadToUnrRot, UnrRotToDeg, UnrRotToRad

UnrealScript & Objekterzeugungen

- Begin Object & End Object

```
class TestPawn extends Pawn;

DefaultProperties
{
    Begin Object Class=SkeletalMeshComponent Name=MyMesh
        CastShadow=true
        SkeletalMesh=SkeletalMesh'BeispielPackage.Mesh.BeispielMesh'
    End Object
    Mesh=MyMesh
```

- Objekterzeugung im laufenden Code
  - Actor: `Spawn(class'UDKPawn',,,,);`
  - Object:

```
local SkeletalMesh exampleMesh;
exampleMesh = new class'SkeletalMesh';
```

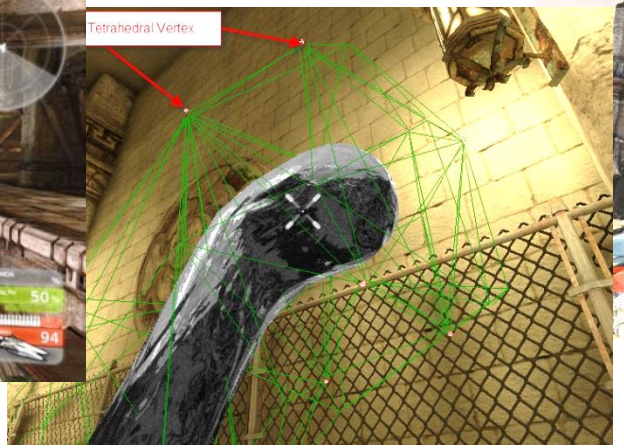
Physik:

Unreal PhysX

Unreal Collision

Physik - PhysX

- UDK nutzt PhysX Engine von NVIDIA
  - Collision Detection
  - Physikalische Interaktionen zwischen Spielobjekten
  - Partikelsysteme
  - Destructible Environments, Soft Bodies, Cloths



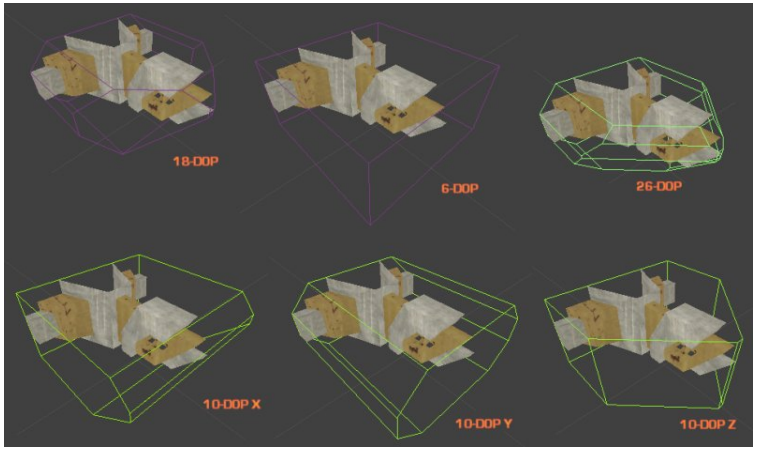
Physik - PhysX

- Jeder Actor kann physikalische Eigenschaften bekommen
  - Rigid Body oder KActor
- Einfach über Editor möglich (Präsentation)
- Konfigurationsmöglichkeiten:
  - Reaktion auf Schaden
  - PhysicalMaterial
    - Gewicht, Dämpfung, Reibung
    - Reaktion auf Aufprall oder Rutschen
      - Sounds, Partikeleffekte
  - Constraints
    - Limitieren Physik auf bestimmte Achsen
- UnrealPhat Editor
  - Komplexere physikalische Assets
  - Bone-Tree, weitere Constraints

Constraints

Physik – Unreal Collision

- Seit Unreal Engine 1 im Einsatz
- Collision Hulls
  - Automatische Generierung
  - Tools



K-DOP tool

- Actor Properties:
  - CollideActors
  - BlockActors
  - BlockZeroExtent (Geschosse)
  - bPathColliding
  - CollisionComponent
    - Unreal erzeugt CollisionCylinder für Actor
    - Kann durch physicsAsset ersetzt werden
  - Properties teilweise in Actor UND Component (SkeletalMesh) vorhanden
    - Actor-Properties überschreiben die des Meshes!

Physik - Wichtige Pawn-Events

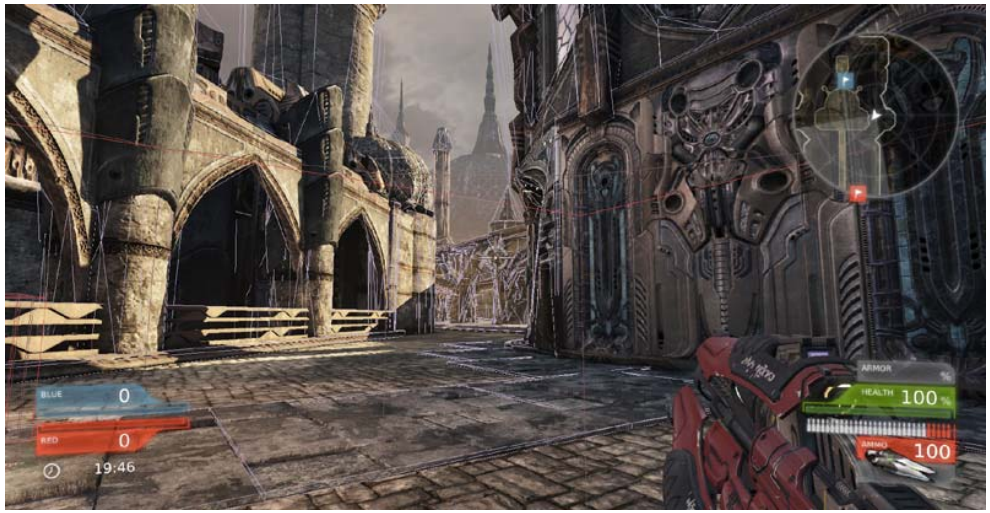
- Event Touch(Actor Other, …)
  - Für „Trigger“
  - Beide Actor-Objekte müssen CollideActors auf true haben
  - Einer der beiden Aktoren muss BlockActors auf false haben
  - Other: Actor, der berührt wurde
- Event UnTouch(Actor Other)
  - Gleiche Bedingungen wie Touch
  - Wird aufgerufen, wenn die beiden Actor-Objekte sich nicht mehr überlagern
- Event Bump(Actor Other,…)
  - Für „echte“ Kollision zwischen Aktoren
  - Beide Actor-Objekte müssen CollideActors und BlockActors auf true haben

Physik - Traces

- Actor Trace(out Vector HitLoc, out Vector HitNorm, Vector End, Vector Start, …)
  - Strecke wird auf Kollision mit Aktoren überprüft

- Bool FastTrace()
  - Schneller als Trace ☺
  - Liefert nur zurück, ob nicht mit etwas kollidiert wurde
  - Sinnvoll zur Überprüfung einer freien Sichtlinie

- Debugging
  - Console Commands
  - z.B. „Show Collision“



Hochschule für Technik und Wirtschaft Berlin
University of Applied Sciences

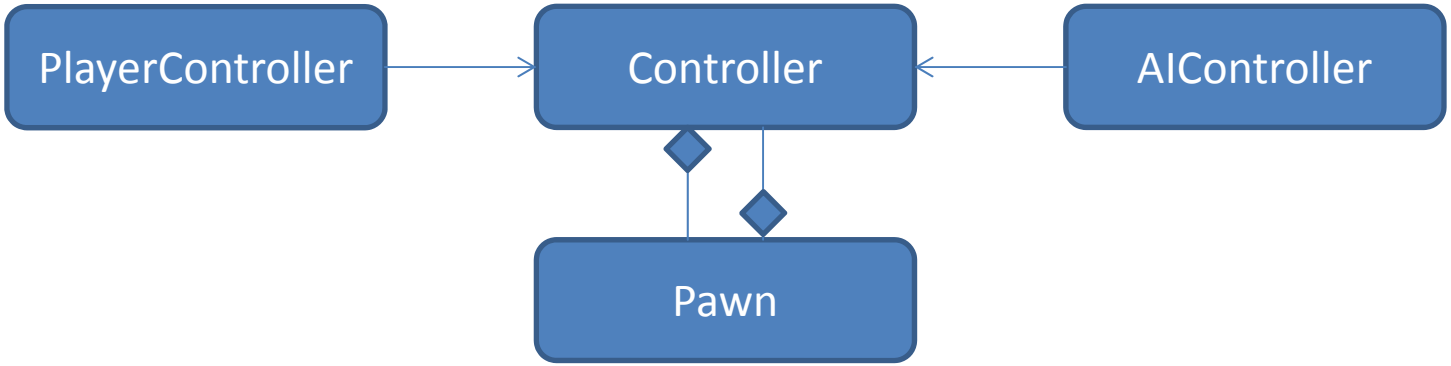
AI:

Grundlagen und States

Pathfinding

AI - Grundlagen

- Erinnerung: Hierarchie



AI – States

- States fest in UnrealScript integriert
- Beispiel Pawn
  - States: „Walking“, „Attacking“, „Dying“, …
- auto gibt initialen Zustand an

```
state Attacking
{
   function Touch(Actor Other, PrimitiveComponent OtherComp, Vector HitLoc, Vector HitNorm)
   {
   }
}

auto state Walking
{
   function Touch(Actor Other, PrimitiveComponent OtherComp, Vector HitLoc, Vector HitNorm)
   {
   }
}
```

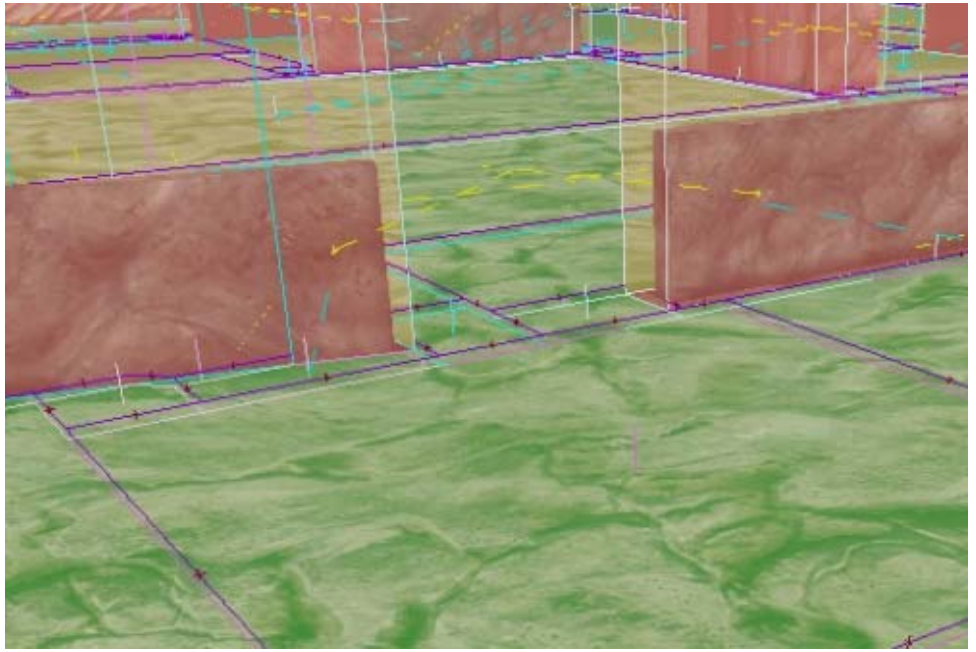
AI – States

- Labels
- Latent functions
  - z.B: sleep(), FinishAnim(), …
  - sind nur in State Code zugänglich!
- Goto(LabelName)
- GotToState(StateName)
- States können erben (extends)
- PushState(StateName) & PopState()
  - Ermöglichen leichtere Rückkehr zu vorigem State

```
auto state MyState
{
    Begin:
    `log( "MyState has just begun!" );
    Sleep( 2.0 );
   `log( "MyState has finished sleeping" );
    goto('Begin');
}
```

AI – Waypoints & Navigation Meshs

- Basis
  - Früher: Waypoints
  - Heute: Navigation Mesh (NavMesh)
    - Automatische Generierung
    - Anpassung



AI – Navigation

- Einfacher Ansatz
  - MoveTo(Vector Position)
    - Gerader Weg zum Ziel
    - Return bei Ankunft
  - MoveTowards(Pawn Ziel)
    - Gerader Weg Richtung Ziel
    - Return nach einem Schritt
    - Möglichkeit der Anpassung des Ziels
- NavigationHandle
  - Befindet sich in Controller-Klasse
  - Verwaltet NavMesh und gefundene Routen

AI – NavigationHandle & GoalEvaluator

- Suchen eines Pfades durch GoalEvaluator:

```
class'NavMeshGoal_At'.static.AtLocation(NavigationHandle, target, 32);

class'NavMeshGoal_At'.static.AtActor(NavigationHandle, target, 32);

bPathFound =  NavigationHandle.FindPath();
```

- Navigieren durch gefundenen Pfad:

```
local Vector TempDest;
if(NavigationHandle.GetNextMoveLocation(TempDest, 100))
{
    MoveTo(TempDest);
}
```

- Weitere GoalEvaluator & Constraints existieren
  - z.B.Deckung suchen
  - GoalEvaluator sind nativ
  - GoalEvaluator können nicht in UnrealScript geschrieben werden

Fragen?

Quellen

- http://udn.epicgames.com/
- http://1.bp.blogspot.com/_nibSxQl2aCk/TBB5L_xzS1I/AAAAAAAABZc/EtJ6-FXd5G4/s1600/YawPitchRoll.jpg