



# Media Engineering

## Game Engines



R. Weller

University of Bremen, Germany

[cgvr.cs.uni-bremen.de](http://cgvr.cs.uni-bremen.de)

- Musik
  - Film
  - Bücher
  - Videospiele
  - Fernsehen
  - Zeitungen
  - Zeitschriften
  - Hörfunk
  - Onlinewerbung
- Gesamtumsatz 2013: ca. 50 Mrd

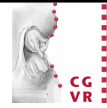
## Mediennutzung 2012

Medium	Nutzungsminuten pro Tag	Veränderung zu 2002 (gerundet)
<b>Fernsehen</b>	205	+9%
<b>Radio</b>	149	-7%
<b>Internet</b>	107	+350%
<b>PC/Videospiele</b>	38	+50%
<b>Buch</b>	33	-1%
<b>Zeitung</b>	19	-20%
<b>Video/DVD</b>	18	-16%
<b>Zeitschrift</b>	8	-50%
<b>Teletext</b>	3	>1%
<b>Kino</b>	3	>1%

Daten: SevenOne Media, bezogen auf Zielgruppe 14-49 Jahre



# Marktübersicht (Digitale) Medien in Deutschland

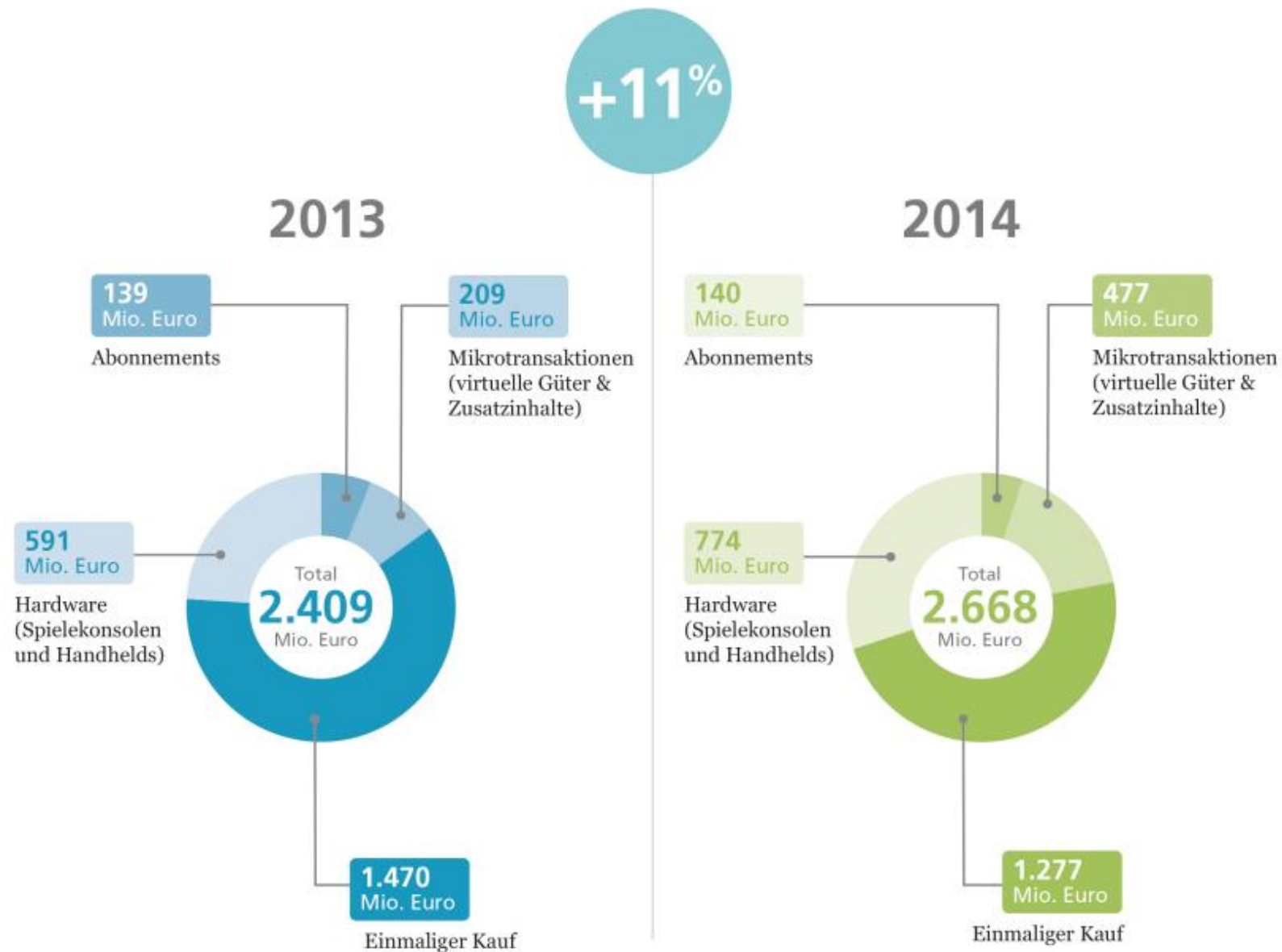


- Fernsehen(12,9M)
- Buch(9,5M)
- Zeitungen(8,0M)
- Zeitschriften(5,6M)
- Onlinewerbung(5,1M)
- Hörfunk(3,5M)
- Film(2,9M)
- Videospiele(1,9M)
- Musik(1,5M)

## Umsatzerlöse in den einzelnen Marktsegmenten

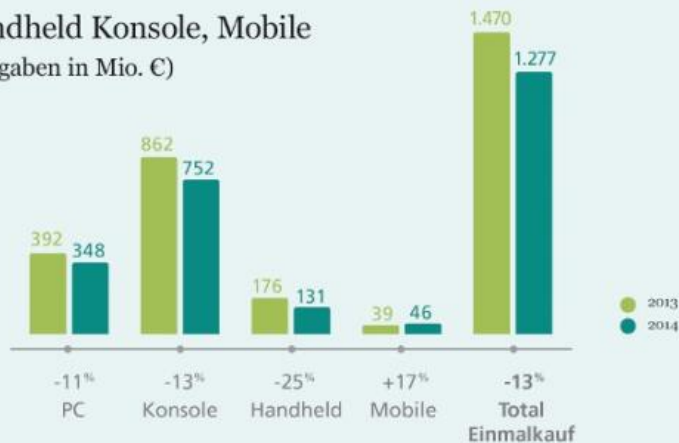
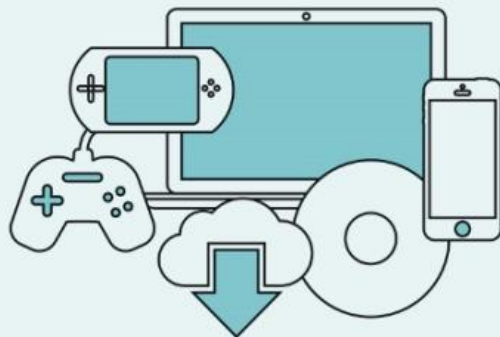
	2009	2010	2011	2012	2013	2014	2015	2016	2017	2018	2014–2018 Ø jährliches Wachstum
Film (in Mio. €)	2.707	2.659	2.746	2.827	2.861	2.891	2.966	3.069	3.161	3.252	
Veränderung (in %)		-1,7	3,3	3,0	1,2	1,1	2,6	3,5	3,0	2,9	2,6
Fernsehen (in Mio. €)	11.225	11.733	12.023	12.511	12.949	13.456	13.595	13.882	14.125	14.324	
Veränderung (in %)		4,5	2,5	4,1	3,5	3,9	1,0	2,1	1,8	1,4	2,0
Musik (in Mio. €)	1.575	1.489	1.483	1.435	1.452	1.454	1.459	1.468	1.475	1.478	
Veränderung (in %)		-5,4	-0,4	-3,2	1,2	0,1	0,3	0,6	0,5	0,1	0,3
Hörfunk (in Mio. €)	3.502	3.501	3.507	3.497	3.588	3.711	3.660	3.688	3.718	3.745	
Veränderung (in %)		0,0	0,2	-0,3	2,6	3,4	-1,4	0,8	0,8	0,7	0,9
Außenwerbung (in Mio. €)	811	843	897	868	891	917	942	956	979	999	
Veränderung (in %)		3,9	6,4	-3,2	2,6	2,9	2,7	1,5	2,4	2,0	2,3
Onlinewerbung (in Mio. €)	3.332	3.769	4.249	4.670	5.126	5.541	5.932	6.317	6.653	7.004	
Veränderung (in %)		13,1	12,7	9,9	9,8	8,1	7,1	6,5	5,3	5,3	6,4
Internetzugang (in Mio. €)	10.161	10.657	11.926	12.869	13.475	13.893	14.383	14.863	15.388	16.067	
Veränderung (in %)		4,9	11,9	7,9	4,7	3,1	3,5	3,3	3,5	4,4	3,6
Zeitschriften (in Mio. €)	5.750	5.832	5.824	5.690	5.609	5.504	5.412	5.355	5.322	5.299	
Veränderung (in %)		1,4	-0,1	-2,3	-1,4	-1,9	-1,7	-1,1	-0,6	-0,4	-1,1
Zeitungen (in Mio. €)	8.588	8.673	8.692	8.441	8.076	7.796	7.635	7.522	7.401	7.285	
Veränderung (in %)		1,0	0,2	-2,9	-4,3	-3,5	-2,1	-1,5	-1,6	-1,6	-2,0
Bücher (in Mio. €)	9.695	9.734	9.601	9.523	9.540	9.547	9.583	9.663	9.758	9.848	
Veränderung (in %)		0,4	-1,4	-0,8	0,2	0,1	0,4	0,8	1,0	0,9	0,6
Videospiele (in Mio. €)	1.895	2.013	2.096	1.967	1.932	2.066	2.157	2.227	2.277	2.324	
Veränderung (in %)		6,2	4,1	-6,2	-1,8	6,9	4,4	3,2	2,2	2,1	3,8
<b>Umsatzerlöse der gesamten Marktsegmente (in Mio. €)</b>	<b>59.065</b>	<b>60.668</b>	<b>62.752</b>	<b>63.948</b>	<b>65.130</b>	<b>66.385</b>	<b>67.306</b>	<b>68.560</b>	<b>69.773</b>	<b>71.102</b>	
Veränderung (in %)		2,7	3,4	1,9	1,8	1,9	1,4	1,9	1,8	1,9	1,8

Quellen: FFA, BVV, ZAW, OVK, GfK-Gruppe, Bundesverband Musikindustrie, VDZ, IWW, Fachpresse-Statistik, BDZV, BIU, Börsenverein des deutschen Buchhandels, PwC, Ovum.



# Videospielmekmarkt in Deutschland

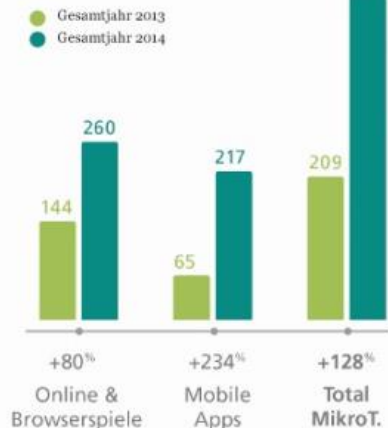
Teilmarkt: Erwerb PC, Stationäre Konsole, Handheld Konsole, Mobile  
 Geschäftsmodell: Einmaliger Kauf von digitalen Spielen (Angaben in Mio. €)



Teilmarkt: Virtuelle Güter und Zusatzinhalte für digitale Spiele



Geschäftsmodell: Mikrotransaktionen für virtuelle Güter, Zusatzinhalte, In-App-Käufe (Angaben in Mio. €)



Browser- und Online-Spiele (Mikrotransaktionen und Abo-Gebühren, Angaben in Mio. €)



Teilmarkt: Abonnements für digitale Spiele

Geschäftsmodell: Abo-Gebühren für Spiele-Software (Angaben in Mio. €)

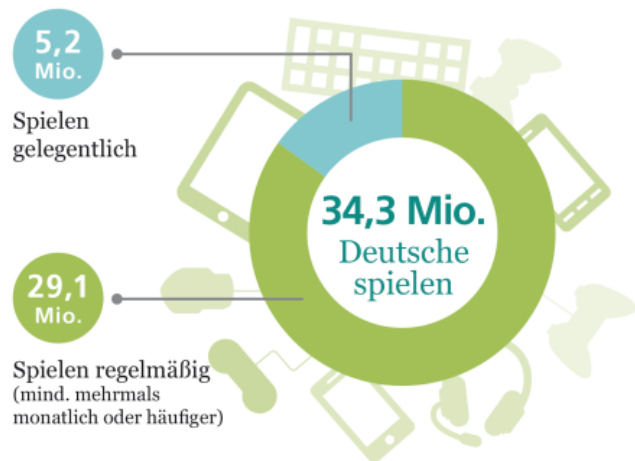


**BIU**   
 Bundesverband Interaktive  
 Unterhaltungssoftware  
[www.biu-online.de](http://www.biu-online.de)

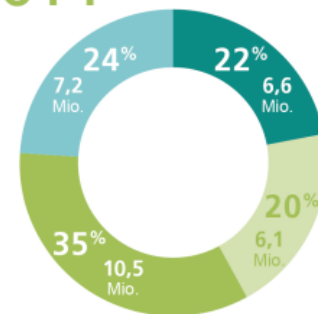
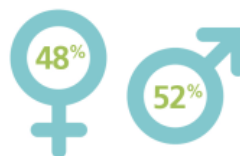
Quelle: basierend auf GfK Consumer Panel Jan-Dec 2014. Visualisierung: visualize.my

# Videospielemarkt in Deutschland

## Nutzer digitaler Spiele in Deutschland 2014



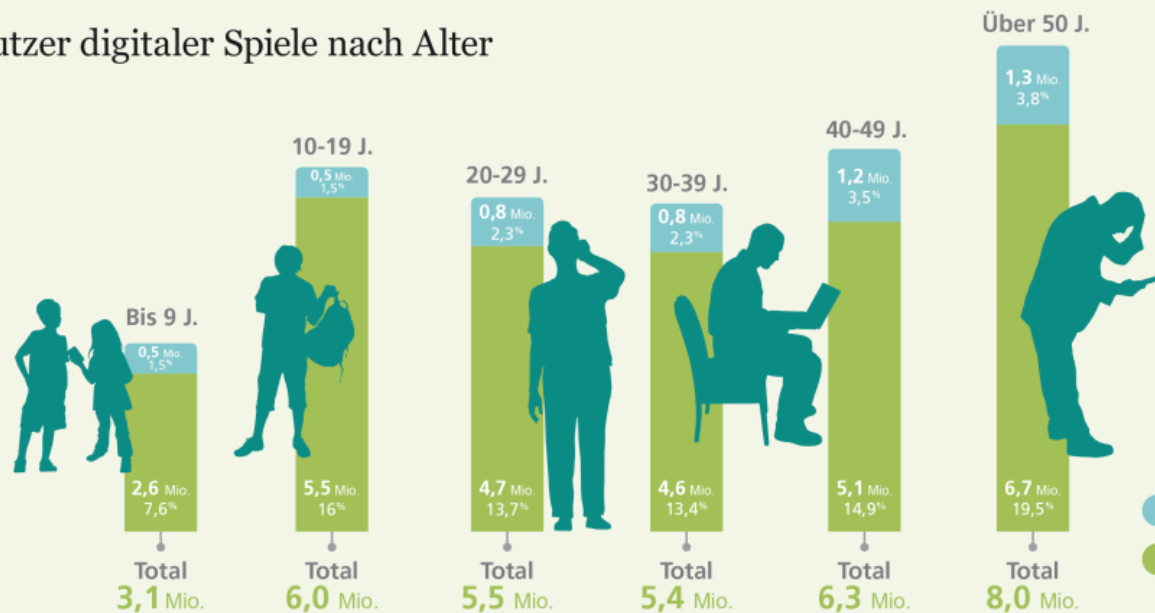
"Knapp jeder zweite Deutsche spielt mittlerweile digitale Spiele - Tendenz steigend. Das zeigt, dass Spiele für immer mehr Menschen fester Bestandteil ihres Medienalltags geworden sind."



### Bildungsgrad

- Hochschule
- Mittlere Reife
- Abitur
- Hauptschule

### Nutzer digitaler Spiele nach Alter

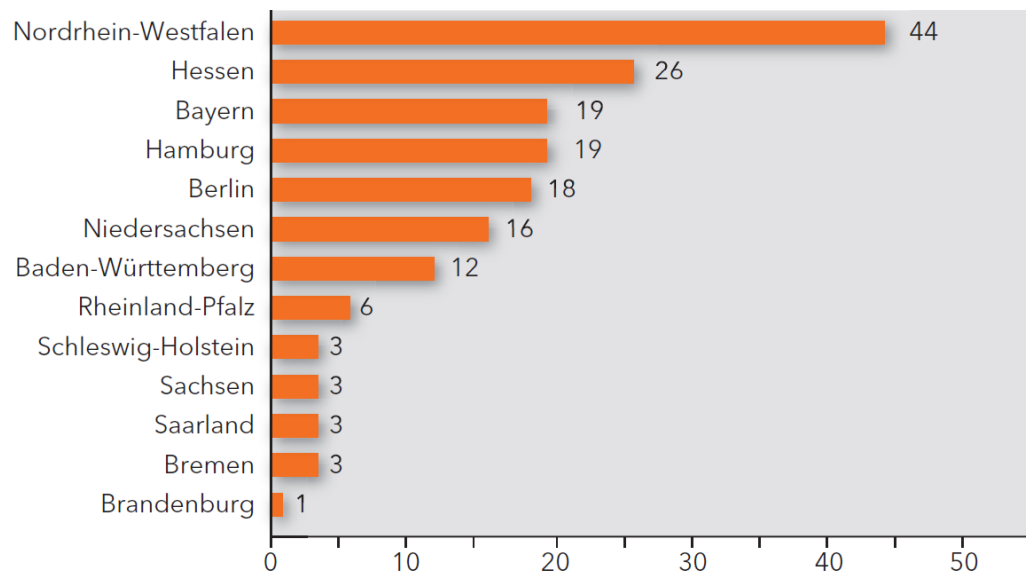


Das Durchschnittsalter der Nutzer digitaler Spiele beträgt **34,5 Jahre**.

- Spielen gelegentlich
- Spielen regelmäßig

- 686 Unternehmen, ca 10000 Beschäftigte
  - Kerngeschäft Entwickler und Publisher:
    - 275 Betriebe (mit > 5 Angestellten)
    - 6000 Beschäftigte
    - > 1500 offene Stellen (auf games-career.com)

## Entwicklerstudios für Videospiele in Deutschland



[Quelle: mediabiz.de]



[Quelle: games-career.com]



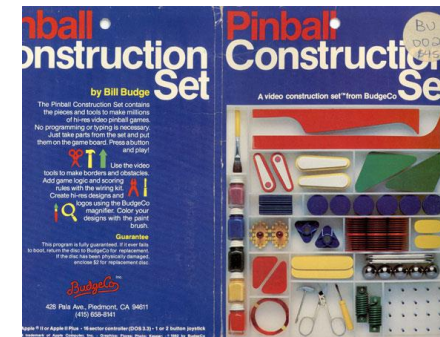
- Tools werden nicht nur in Spielen eingesetzt, sondern zunehmend auch in anderen Anwendungen
  - Serious Games
  - VR, AR
  - Mobile Apps
  - Kunst (Interaktive Installationen), Architektur
  - Forschung (z.B. Datenvisualisierung)
- Ähnliche Tools werden auch beim Film verwendet
  - Renderman (Pixar), Cinema 4D, 3ds Max
  - Blender hat sogar eine Game Engine eingebaut
  - Unterschied: Echtzeitfähigkeit (wobei Grafik-Engines für Filme oft einen reduzierte Echtzeitvorschau bieten)



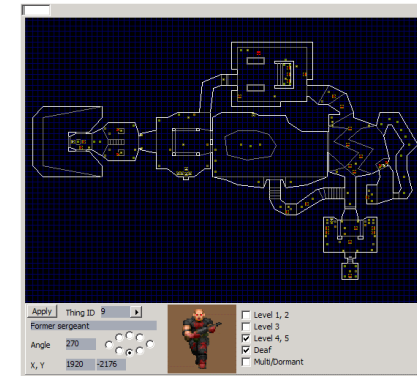
[Quelle: Ilumens, 2013]

# Spieleentwicklung - Prä-Historisches

- Ab ca 1977: Arcade-Spiele
  - Hardware sehr langsam
    - Spezialhardware
  - Zeitaufwändige Anpassung an Zielmaschine  
=> Kaum Wiederverwendung von Code möglich
  
- Ab ca 1983: Game Creation Systeme
  - Sehr spezielle Tools für bestimmte Genres
  - Meist für kleinere Indie-Titel
  - Beispiele: Pinball Construction KIT (1983),  
Adventure Construction Set (1984),  
Wargame Construction Set (1986), Shoot'Em-Up Construction Kit  
(1987), RPG Maker (1988)



- Der Begriff „**Game Engine**“ wurde ab Mitte der 1990er im Zusammenhang mit 3D-FPS (First-Person-Shootern) wie Wolfenstein 3D und Doom verwendet.
  - **Trennung von Technik** und eigentlichem „**Game Content**“ wie Texturen und Leveln
  - Doom bot den Benutzern die Möglichkeit, eigene **Modifikationen** (z.B. Level, Waffen, Gegner) des Contents zu entwerfen und miteinander zu teilen
  - Kommerzielle Anwender konnten den Programmcode **lizensieren** und damit erstellte Spiele auch verkaufen
    - Doom II (1994), Heretic (1996), Hexen (1995), Strife (1996), Check Quest (1996)
    - HacX: Twitch n Kill (1997)
    - Cruis'n Velocity (2001)
    - Dark Arena (2002)

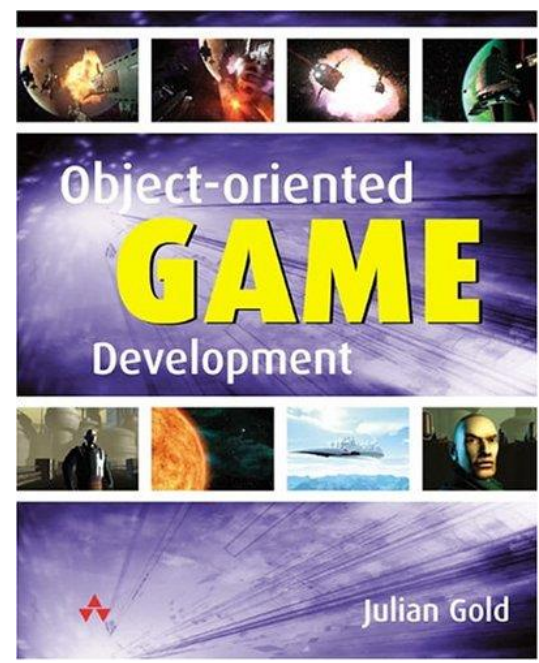


# Historisches – Die Entwicklung bis heute

- Der Erfolg von Doom und Quake wurde von vielen Spieleherstellern kopiert (z.B. Beigabe von Leveleditoren für Nutzercontent)
  - Z.B.: Duke Nukem 3D, Warcraft II
- Spätere Game Engines wurden extra für die Lizenzierung entwickelt (aber immer noch von Firmen, die hauptsächlich durch den Verkauf ihrer eigenen Spiele Geld verdienen)
  - Z.B. Quake II (1997) und Unreal (1998)
- Später wurden spezielle Game Engines nur für den Verkauf entwickelt und die Spiele waren nur noch Demo für die Engine
  - CryEngine (2004)
  - Oder die Engineentwickler stellten gar keine Spiele mehr her (Vollständige Trennung von Game Engine und Game Content)
    - Unity (seit 2005)

# Definition: Game Engine

“A series of **modules** and interfaces that allows a development team to focus on produce *game-play content*, rather than *technical content*.” - **Julian Gold, Object-Oriented Game Dev.**

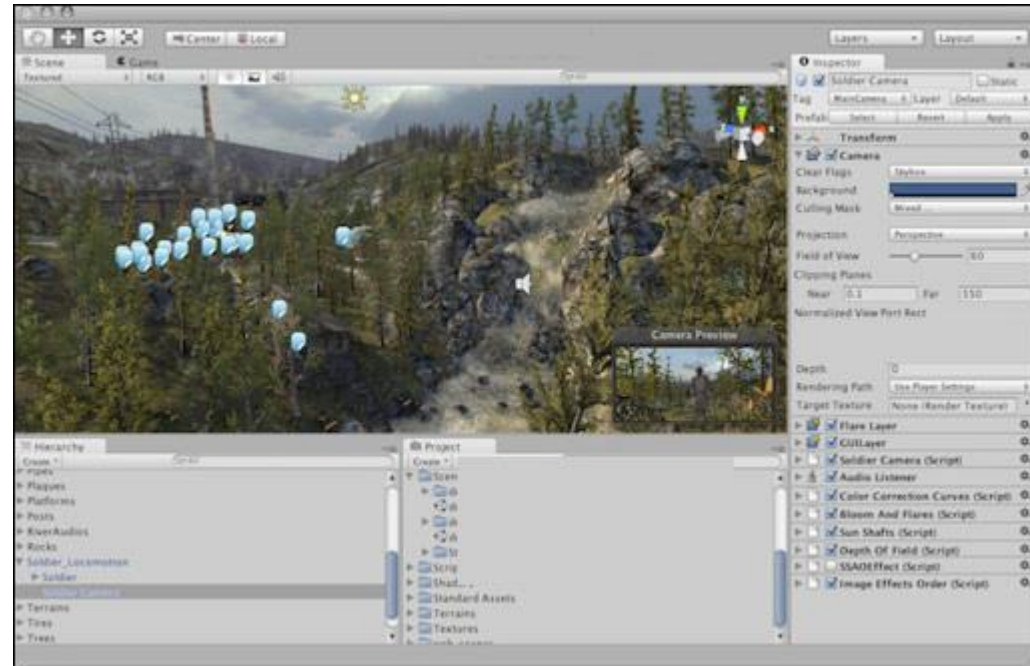
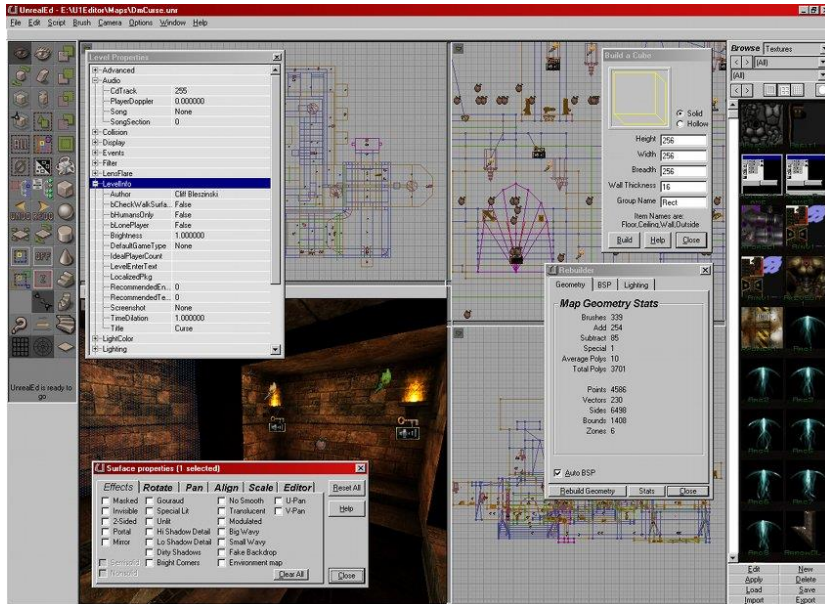
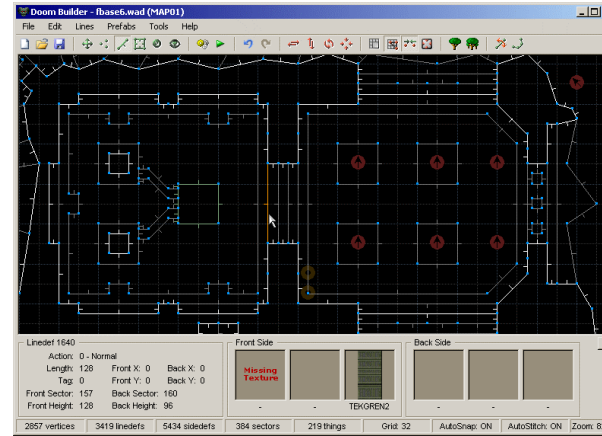
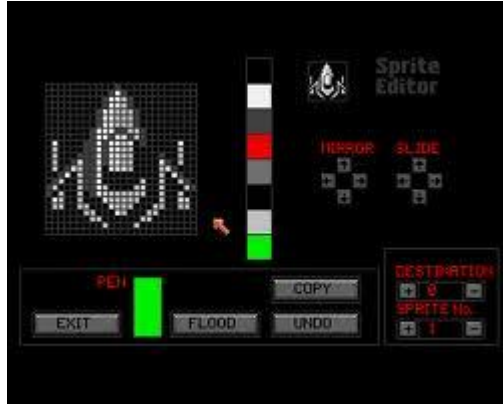


# Warum verwendet man Game Engines?

- Bieten einen einfachen Ansatz die Entwicklung des Spieleinhalts von den darunter liegenden technischen Aspekten zu abstrahieren
- Erleichtert die Wiederverwendbarkeit von Code
- Ermöglicht die gleichzeitige Entwicklung für mehrere Zielplattformen
  - PC/Mac
  - Konsolen
  - Mobile Geräte
    - Android/iOS



# Game Engines im Wandel der Zeit



id Tech 1 (1999)	79k
id Tech 2 (2001)	138k
id Tech 3 (2005)	329k
id Tech 4 (2011)	586k
UE4 v4.6 (2015)	1964k



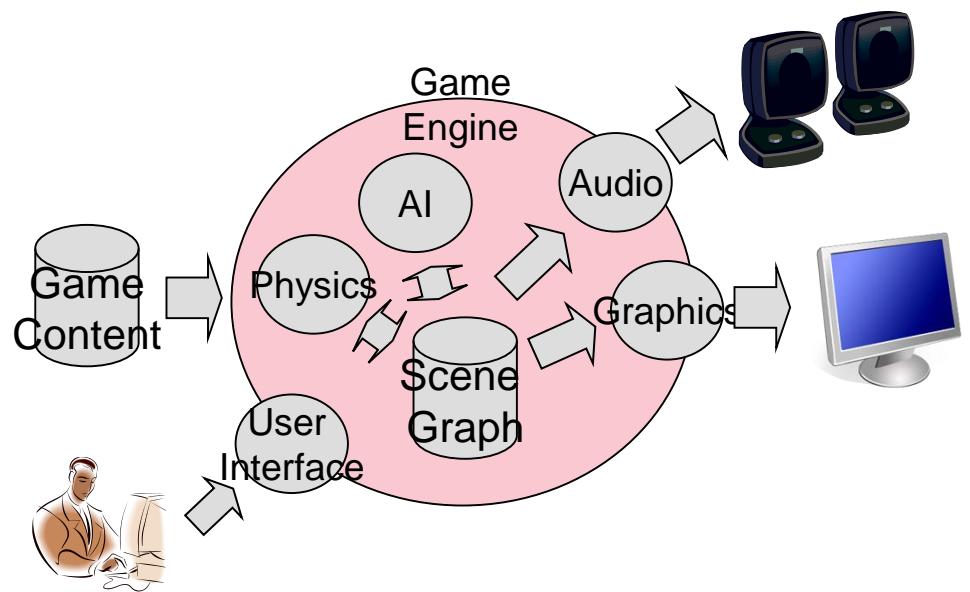
- Used cloc
- Only counting C, C++ and header files.

*“Measuring software productivity by lines of code is like measuring progress on an airplane by how much it weighs.” - Bill Gates*



# Was ist eine Game Engine?

- Softwaretechnologie zum Erstellen von Computerspielen
- Behinhaltet unter anderem zahlreiche **Softwaremodule** die für die meisten Spiele benötigt werden:
  - Graphik
  - Physik
  - Sound
  - Scripting
  - Animation
  - Künstliche Intelligenz (KI)
  - Netzwerkunterstützung
  - Benutzerschnittstellen



# Die Module im Überblick: Rendering Engine

- Aufgabe der Rendering Engine ist es, die vom Benutzer erstellten 3D-Objekte hübsch darzustellen
- Sie ist meist das größte und komplexeste Modul einer Game Engine
- Oft wird sie in mehrere Untermodule unterteilt:
  - Renderer
  - Szenengraph
  - Visuelle Effekte
  - GUI

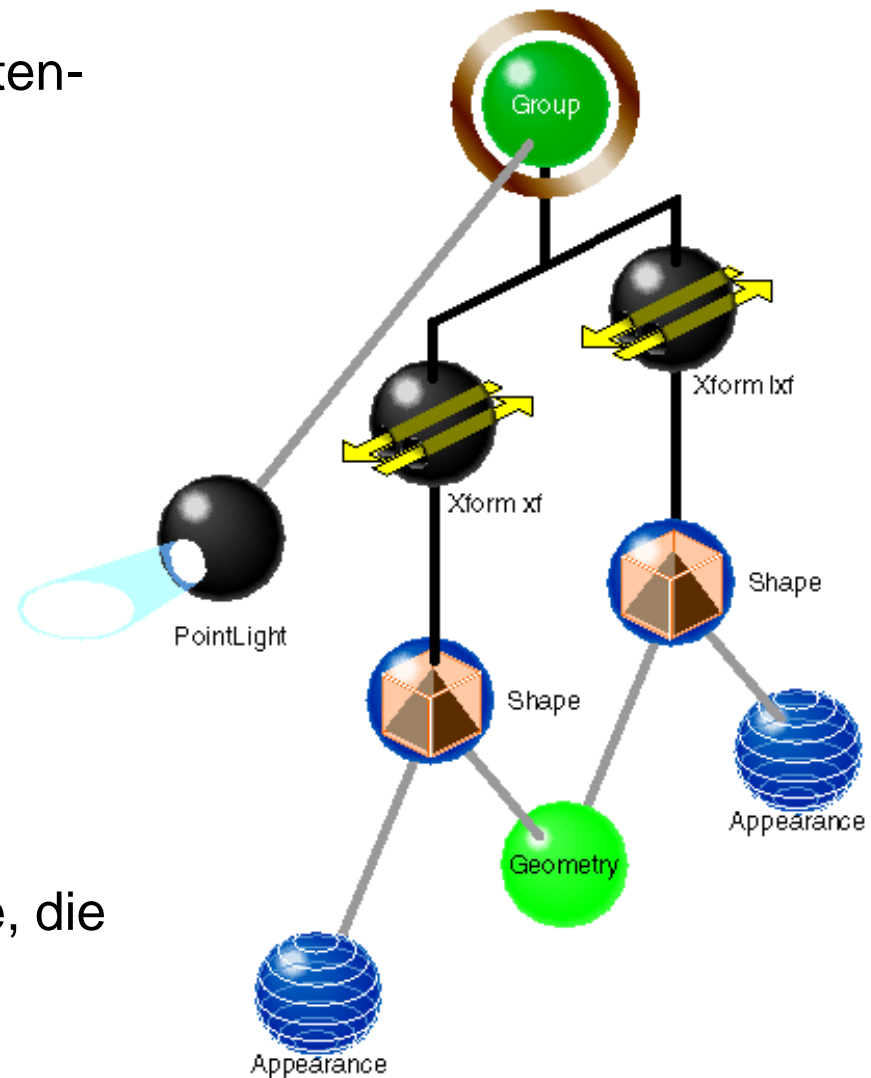


- Die Aufgabe des Renderers ist es, eine Menge von geometrischen Primitiven so schnell wie möglich darzustellen
  - Geometrische Primitiven sind beispielsweise: 3D Dreiecksnetze, Linien, Punkte, Partikel
- Der Renderer stellt darüber hinaus ein Basissystem für die Beleuchtung und die Materialverwaltung bereit
- Dabei greift er meist direkt auf die Grafikhardware zu oder er verwendet Plattform-unabhängige Zwischenschichten wie OpenGL oder DirectX

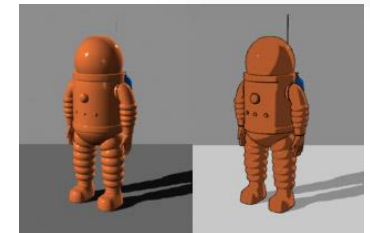


# Rendering Engine: Szenengraph

- Üblicherweise eine baumartige Datenstruktur um die Objekte einer Spielszene für das Rendering (und andere Spielaspekte) zu verwalten
  - Objekte
    - Transformationen
    - Materialien
  - Lichtquellen
  - Kameras
- Unterstützt meist Culling (d.h. Das Abschneiden von Teilen der Szene, die nicht im Sichtbereich)



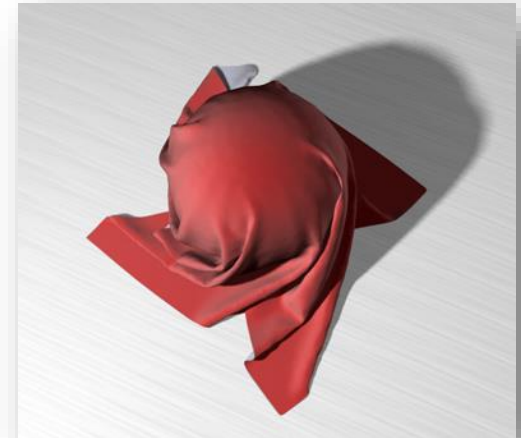
- Spezialeffekte, die nicht direkt vom Renderer (bzw der Zwischenschicht OpenGL oder DirectX) unterstützt werden
- Oft durch mitgelieferte Shader-Programme implementiert
- Beispiele:
  - Partikeleffekte (Feuer, Rauch, Explosionen)
  - Dynamische Schatten
  - Non-Photorealistic-Rendering (z.B. Cel Shading)
  - Environmental Mapping (Spiegelungen der Umgebung)
  - Post-Render-Effekte
    - HDR, FSAA, Farbkorrektur



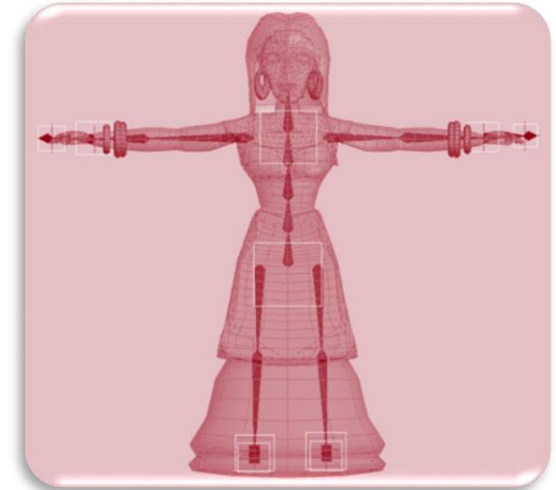
- Darstellung von 2D-Elementen
  - Z.B: Menus
  - Heads-Up-Display (HUD)
  - In-Game 2D-Benutzerinterfaces (z.B. Inventar)
  - Darstellung von Debugging-Informationen im Spiel



- Sorgt dafür, dass sich Objekte in einer Szene gemäss den Newton'schen Gesetzen der Physik verhalten (also, z.B., dass einem der Apfel auf den Kopf fällt)
- Definiert Masse, Kräfte und Geschwindigkeiten für die Objekte
- Kollisionsdetektion erkennt, wann Objekte zusammenstossen
- Verwendet oft vereinfachte Modelle der Physik um Echtzeitfähigkeit zu garantieren



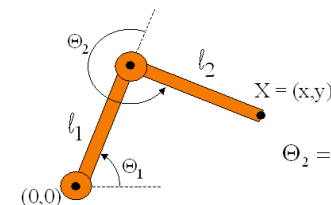
- Animation der Spielcharaktere
  - Meist Skelettanimation
  - Oft werden Daten unterstützt, die per Motion-Capturing gewonnen werden
- Überblenden einzelner Posen (Keyframes)
- Übertragen der Bewegung auf das darüberliegende Dreiecksnetz (Skinning)
- Teilweise auch inverse Kinematiken oder Forward-Kinematiken



## Inverse Kinematics



- Animator specifies end-effector positions:  $X$
- Computer finds joint angles:  $\Theta_1$  and  $\Theta_2$ :



$$\Theta_2 = \cos^{-1} \left( \frac{x^2 + y^2 - l_1^2 - l_2^2}{2l_1l_2} \right)$$

$$\Theta_1 = \frac{-(l_2 \sin(\Theta_2))x + (l_1 + l_2 \cos(\Theta_2))y}{(l_2 \sin(\Theta_2))y + (l_1 + l_2 \cos(\Theta_2))x}$$

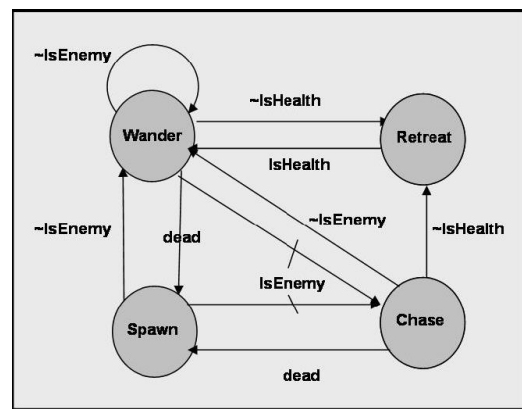


- Abspielen von Sounds
- Hintergrundmusik (evtl szenenabhängiges Überblenden)
- Spielgeräusche (Motorengeräusche, Reifenquietschen, Waffenklingen, Schreie, Sprache,...)
  - Triggerbasiert
  - Abhängig von Spielphysik (Kollisionen)
- Spielgeräusche werden im 3D-Raum positioniert und mit Effekten versehen (Hall, Delay,...)
- Entscheidend ist hier oft die Anzahl und Art der unterstützten Formate und Effekte (MP3, WAV, ogg,...)

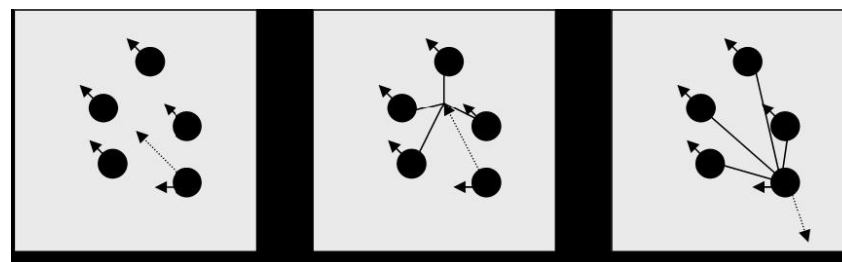
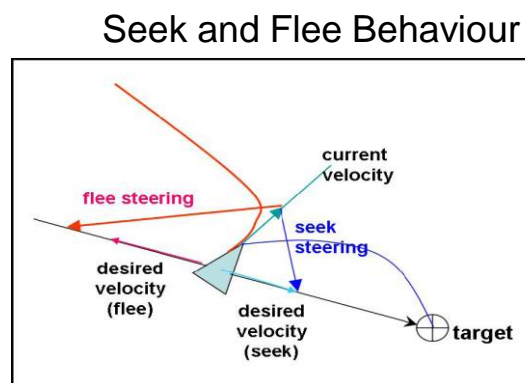


# Künstliche Intelligenz (KI)

- Sorgt dafür, dass sich NPCs (aber auch Spieler) glaubhaft in der Spielwelt bewegen und verhalten
- Bessere Engines bieten Algorithmen aus der KI
  - Endliche Zustandsautomaten
  - Neuronale Netzwerke (lernen aus dem Verhalten des Spielers)
- Gruppenverhalten
  - Schwarmverhalten
  - Fluchtverhalten
- Wegfindung

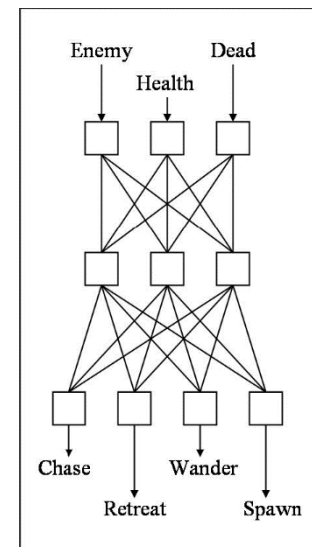


Finite State Automata



Flocking

Neural Network



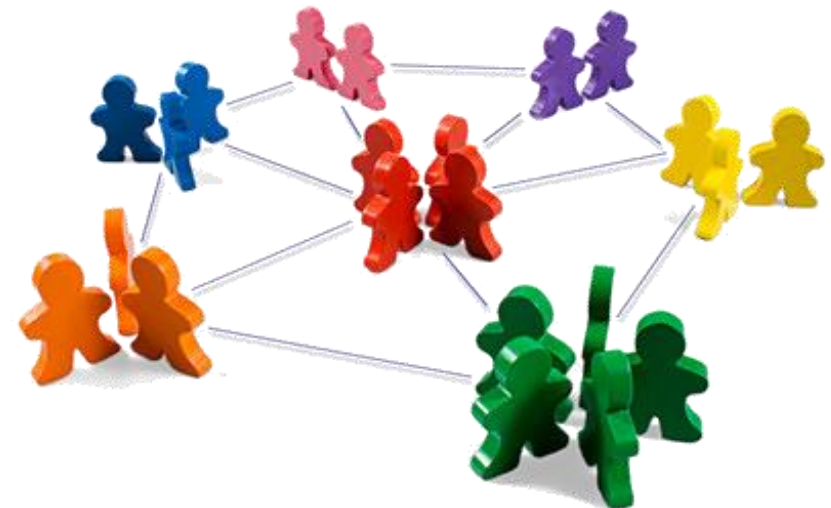
- Meist einfache, interpretierte Skriptsprache die einfachen Zugriff auf Teile (oder die gesamte) Engine erlaubt (dadurch muss das Spiel nicht bei jeder Änderung neu kompiliert werden)
- Z.B. Veränderung der Spielmechaniken durch Definition von
  - Events
  - Auszeichnungen
  - KI-Erweiterungen
- Kann eine Engine-unabhängige Sprache sein oder auch eine selbstentwickelte (Unreal-Scripting Language)



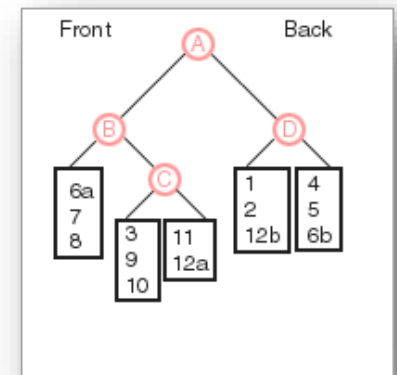
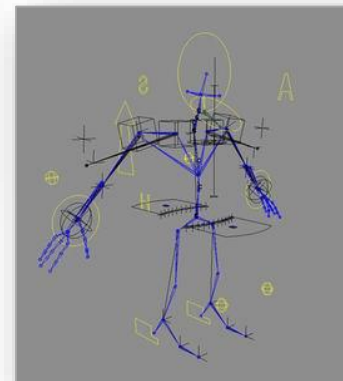
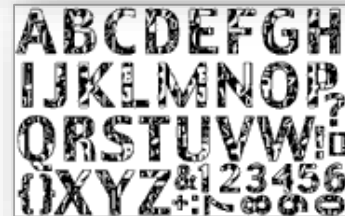
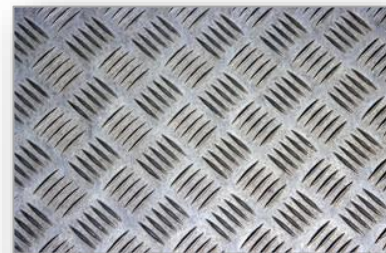
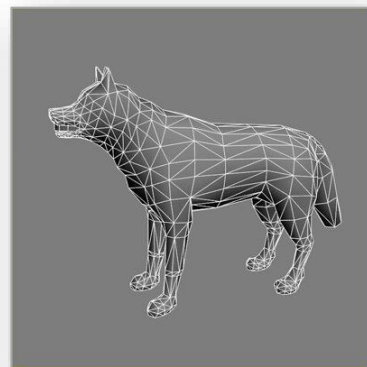
- Spiele unterstützen oft verschiedene Eingabegeräte
  - Keyboard und Maus
  - Gamepad
  - Spezialcontroller (Kinect, Lenkräder, Flighsticks, billige Plastikgitarrenimitate mit vier bunten Knöpfen, Wii Fit Board,...)
- Dieses Modul abstrahiert das Mapping von logischen Spielfunktionen und dem physikalischen Controller



- Unterstützung von Multiplayer-Spielen, wobei meist nur direkte Kommunikation zwischen den Spielern unterstützt wird
  - Dies ist zu unterscheiden von Massively Multiplayer Games (MMOGs), bei denen tausende Spieler gleichzeitig in einer gemeinsamen Welt interagieren. Dazu werden meist riesige Serverfarmen benötigt
- Stellt einen gemeinsamen, eindeutigen Zustand auf allen beteiligten Rechnern sicher
  - Dazu werden meist ausgefeilte Caching-Strategien verwendet

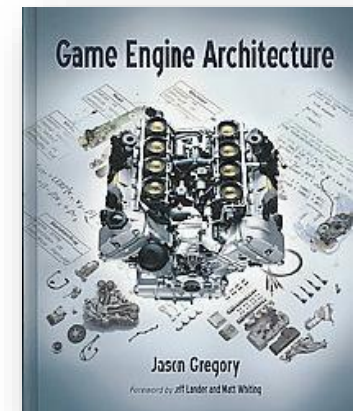
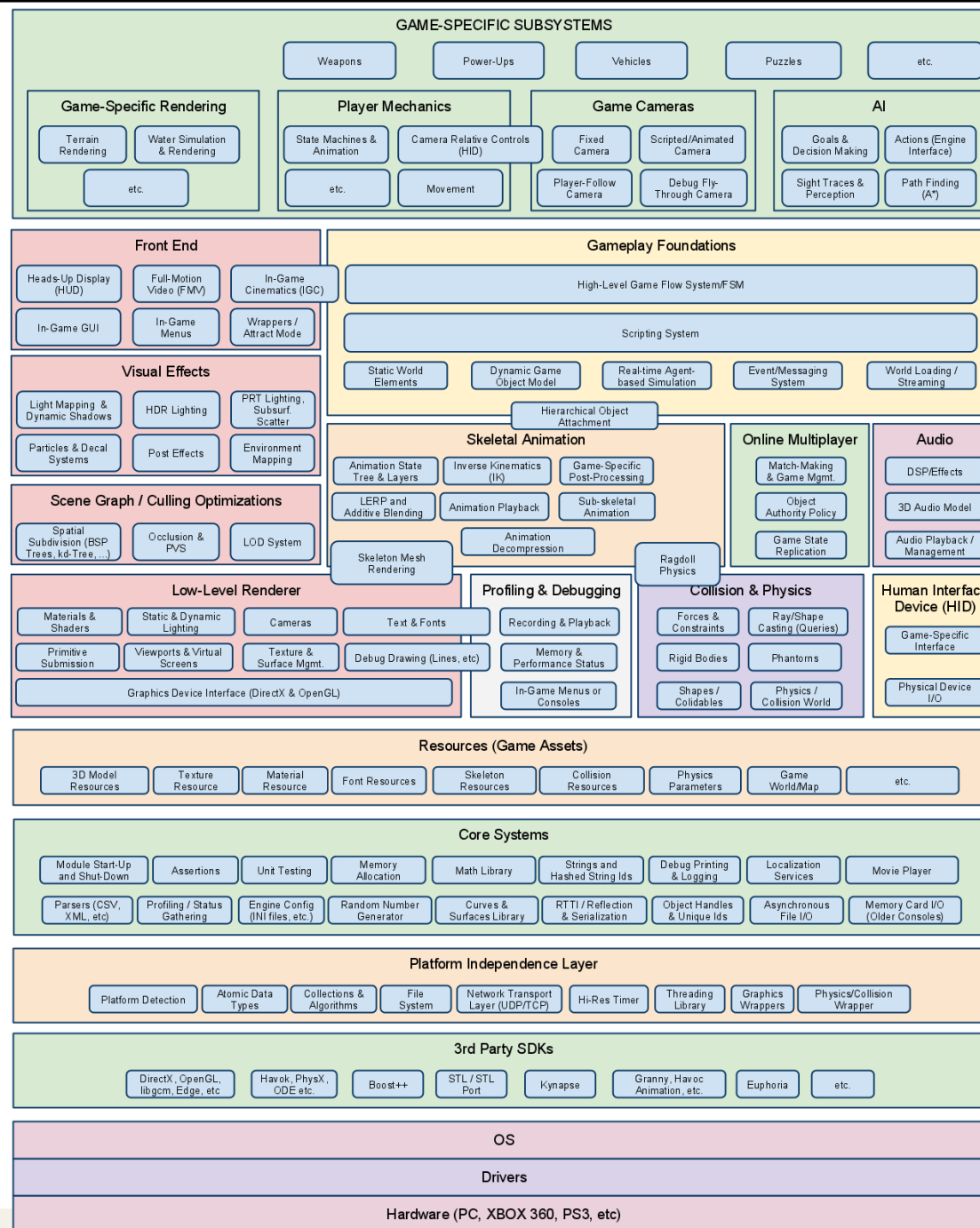


- Game Engine-eigenes Tool zum verwalten des Contents
  - 3D Objekte
  - Texturen
  - Fonts
  - Skeletanimationen
  - Levelbeschreibungen (Maps)
  - Sounds



- Meist sehr hardwarenahe Utilities und Support-Klassen
  - Mathematik-Library
  - Zufallszahlengenerator
  - Funktionen zum Abspielen von Filmen
  - Parser
  - File I/O
  - Tools für die Übersetzung in andere Sprachen
  - Debugging/Profiling-Tools
  - Multithreading
  - Speicherverwaltung
  - Assertions
  - ...

# Game Engine Architecture Blocks (complete?)



**Game Engine Architecture**, by Jason Gregory, 2009, AK Peters, ISBN: 1-5688-1413-5.



# Die Game-Loop

- While user input not exit
  - Update Scene-Graph via User-Input
  - Update Scene-Graph with network cache
  - Update Scene-Graph via KI
  - Update Scene-Graph via Physics and Animation
  - Render Scene-Graph to Screen via Graphics System
- Endwhile

# Game Engines vs. SDKs und APIs

- Game Engines benötigen oft eine enge Anbindung an die Hardware
  - Realisierung durch Plattform-unabhängige Extraschichten oder betriebssystemeigene APIs
- Kaum ein Game-Engine-Entwickler erfindet das Rad komplett neu. Meist Rückgriff auf bekannte und etablierte SDKs und APIs:
  - Datenstrukturen und Algorithmen: STL, Boost
  - Graphikhardware: DirectX, OpenGL
  - Physik: Havok, PhysX, ODE
  - Sound: Fmod, Irrklang





# One Game Engine to Rule Them All?



# “It’s a Jungle Out There”

- Fast mehr Game-Engines als Spiele
- 375 Engines auf DevMaster.net
- 444 Engines auf IndieDB.com

100 Most Popular Engines Today						
Search ...		Any Status ▾	Any Licence ▾	Search		Reset
Name	Released	Licence	Last Updated	Visits (today/total)	Trend	
1. Unity	✓	Commercial	17hours 48mins ago	16   609,201	↑ 2	
2. S2ENGINE HD	✓	Commercial	Mar 15 2015, 5:32am	11   59,047	↓ 1	
3. Unreal Engine 4	✓	Commercial	19hours 4mins ago	11   109,046	↓ 1	
4. Torque Game Engine	✓	Commercial	8hours 57mins ago	10   51,118	↑ 124	
5. GameMaker: Studio	✓	Commercial	9hours 12mins ago	8   289,109	↑ 3	
6. Wave Engine	✓	Proprietary	Feb 26 2015, 5:20am	7   64,913	↓ 1	
7. 2D Fighter Maker 2nd	—	Commercial	Oct 7 2013, 12:05pm	5   25,858	↓ 3	
8. Blender Game Engine	✓	GPL	Feb 4 2015, 8:04pm	5   136,744	↑ 9	
9. C4 Engine	✓	Commercial	Feb 21 2015, 4:26pm	5   49,030	↑ 24	
10. Construct 2	✓	Commercial	Mar 16 2015, 11:37am	4   87,265	↑ 9	

# Zahlreiche Features zur Auswahl

## General Info

### Graphics API

[OpenGL](#) | [DirectX](#) | [Glide](#) | [Software](#) | [Other](#)

### Operating Systems

[Windows](#) | [Linux](#) | [MacOS](#) | [Solaris](#) | [SunOS](#) | [HP/UX](#) | [FreeBSD](#) | [Irix](#) | [OS/2](#) | [Amiga](#) | [DOS](#) | [Xbox](#) | [Playstation](#) | [GameCube](#) | [GBA](#) | [PSP](#) | [N-Gage](#) | [BeOS](#) | [Xbox360](#) | [PS2](#) | [PS3](#) | [Nintendo Wii](#) | [Nintendo DS](#)

### Programming Language

[C/C++](#) | [Java](#) | [C#](#) | [D](#) | [Delphi](#) | [Pascal](#) | [BASIC](#) | [Ada](#) | [Fortran](#) | [Lisp](#) | [Perl](#) | [Python](#) | [Visual Basic 6](#) | [VB.NET](#)

### Status

[Alpha](#) | [Beta](#) | [Productive/Stable](#) | [Inactive](#)

### Misc

[Documentation](#)

### General Features

[Object-Oriented Design](#) | [Plug-in Architecture](#) | [Save/Load System](#) | [Other](#)

## Game Features

### Networking System

[Client-Server](#) | [Peer-to-Peer](#) | [Master Server](#)

### Tools & Editors

[Scripting](#) | [Built-in Editors](#)

### Sound & Video

[2D Sound](#) | [3D Sound](#) | [Streaming Sound](#)

### Physics

[Basic Physics](#) | [Collision Detection](#) | [Rigid Body](#) | [Vehicle Physics](#)

### Artificial Intelligence

[Pathfinding](#) | [Decision Making](#) | [Finite State Machines](#) | [Scripted](#) | [Neural Networks](#)

## Graphics Features

### Lighting

[Per-vertex](#) | [Per-pixel](#) | [Volumetric](#) | [Lightmapping](#) | [Radiosity](#) | [Gloss maps](#) | [Anisotropic](#) | [BRDF](#)

### Shadows

[Shadow Mapping](#) | [Projected planar](#) | [Shadow Volume](#)

### Texturing

[Basic](#) | [Multi-texturing](#) | [Bumpmapping](#) | [Mipmapping](#) | [Volumetric](#) | [Projected](#) | [Procedural](#)

### Shaders

[Vertex](#) | [Pixel](#) | [High Level](#)

### Rendering

[Fixed-function](#) | [Stereo Rendering](#) | [Raytracing](#) | [Raycasting](#) | [Deferred Shading](#) | [Render-to-Texture](#) | [Voxel](#) | [Fonts](#) | [GUI](#)

### Scene Management

[General](#) | [BSP](#) | [Portals](#) | [Octrees](#) | [Occlusion Culling](#) | [PVS](#) | [LOD](#)

### Animation

[Inverse Kinematics](#) | [Forward Kinematics](#) | [Keyframe Animation](#) | [Skeletal Animation](#) | [Morphing](#) | [Facial Animation](#) | [Animation Blending](#)

### Meshes

[Mesh Loading](#) | [Skinning](#) | [Progressive](#) | [Tessellation](#) | [Deformation](#)

### Surfaces & Curves

[Splines](#) | [Patches](#)

### Special Effects

[Environment Mapping](#) | [Lens Flares](#) | [Billboarding](#) | [Particle System](#) | [Depth of Field](#) | [Motion Blur](#) | [Sky](#) | [Water](#) | [Fire](#) | [Explosion](#) | [Decals](#) | [Fog](#) | [Weather](#) | [Mirror](#)




### Terrain



[Rendering](#) | [CLOD](#) | [Splatting](#)

[ [DevMaster.net](#) ]

# Einige wichtige Unterscheidungsmerkmale

- Kommerziell vs. Open Source vs. Kostenlos
- Unterstützte Plattformen
- Unterstützte Features/Module
- Support/Community (Online Foren, Mailing-Listen)/Dokumentation
- Unterstützte Programmiersprachen
- Extensible IDE vs. Open Class Library

- Code-orientierte Entwicklung
- Meist sehr bedacht aufgebautes internes Modulsystem (Da der Entwickler damit direkt in Berührung kommt)
- Können sehr leicht modifiziert werden
- Oftmals Open Source
  - Bemerkung: Für die Unreal Engine ist der Source Code vollständig erhältlich, er darf aber nicht frei weiter verteilt werden
- Oftmal größere Hürde für Einsteiger und Gelegenheitsprogrammierer
- Beispiele:
  - Irrlicht, C4, Unreal (teilweise)   
  - Nahezu alle reinen 3D Engines wie Ogre 3D, OpenSG, Open Scene Graph

- GUI-orientierte Entwicklung
  - Einsteigerfreundlich
  - Eher “Content”-orientiert
- Einfache Verwaltung des Contents (Texturen, Objekte,...)
  - Direkt in die Engine-eigene IDE integriert
- Eingeschränkter (bzw kontrollierter) Zugriff auf die Kernfunktionen
  - Verhindert Missbrauch/Fehler
  - Verhindert aber auch eigene Erweiterungen und kann Kreativität einschränken
- Beispiele: Unity, Unreal  

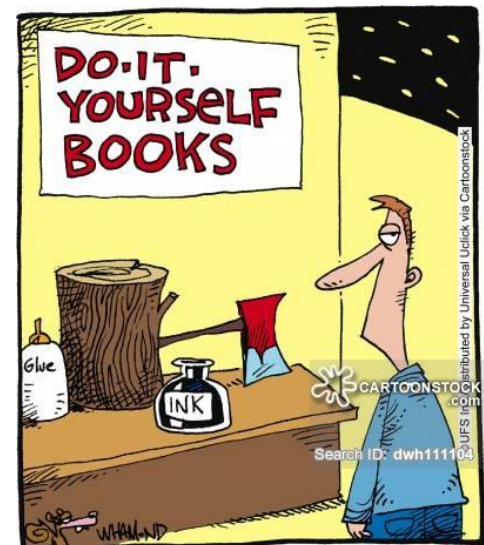


# DIE “beste” Engine gibt es nicht

- Die Auswahl der Engine für ein bestimmtes Projekt ist immer sehr situationsabhängig
  
- Einige wichtige Auswahlkriterien
  - Plattform, Programmiersprache
  - Kosten
  - Spezielle technische Features
  - Vorerfahrung mit der Engine
  - Support
  - Bösertige Vorgaben durch den Dozenten

# Vielleicht lieber alles selbst machen?

- Abhängig von Anforderungen, Resources und anderen Nebenbedingungen
  - Technische Anforderungen (z.B., will man das letzte Quäntchen Performance herausholen?)
  - Finanzielle Ressourcen (z.B., gibt es genügend Startkapital?)
  - Zeitliche Nebenbedingungen (z.B., soll das Spiel in einem Monat oder erst in zwei Jahren fertig werden?)
  - Anforderungen an die Zielplattform (z.B., Browsergames mit Flash?)
  - Andere Faktoren (z.B., ist das Spiel ein Sequel und es gibt bereits Code vom Vorgänger?)
- Realität: Die meisten heute entwickelten Spiele verwenden irgendeine Art von “Engine Layer”



# Gründe um komplett neu zu entwickeln

- Die technischen Anforderungen werden von keiner existierende Game Engine erfüllt (z.B. Minecraft)
- Pädagogische Gründe (weil man in erster Linie die Technologie erlernen will und erst in zweiter ein Spiel entwickeln)
- Bietet besseres Verständnis der Game Engine
  - Kann bei Bedarf einfach erweitert oder angepasst werden
- Eine eigene Engine passt genau zum Genre, in welchem man das Spiel entwickelt (kaum Performanceverluste durch Overhead, man implementiert nur Features, die man auch wirklich benötigt)
- Man will keine Lizenzgebühren zahlen
  - Eigentlich kein wirkliches Argument mehr – es gibt zahlreiche günstige oder gar kostenlose Engines → Es wird immer mehr kosten, eine eigene Engine zu entwickeln

# Gründe um eine Game Engine zu verwenden

- Finanzielle – Zeit/Geld reichen nicht, eine eigene Engine zu entwickeln
- Support – Bestehende Engines haben schon eine User Community und/oder eine Dokumentation
- Robustheit – Bestehende Engine haben weniger Bugs und die Codebasis ist bereits getestet
- Vorkenntnisse – Die Entwickler verfügen bereits über Erfahrungen mit einer bestehenden Game Engine

# Warum Unreal Engine in dieser Vorlesung?

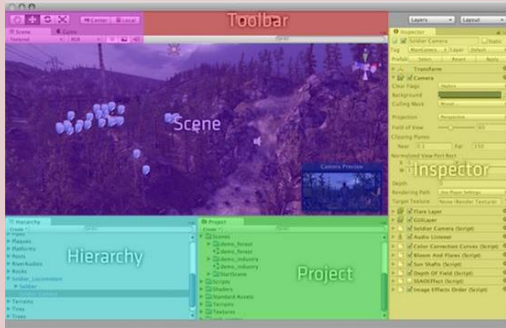



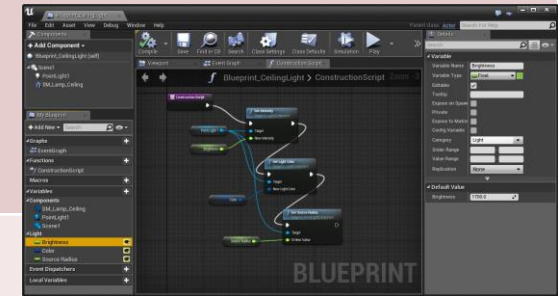
- Bietet sowohl GUI-orientierte Entwicklung als auch Quellcode
  - Leichter Einstieg als beispielsweise mit C4
  - Mehr Kontrollmöglichkeiten als Unity
- Programmierung in C++
  - Nach wie vor Goldstandard im Bereich der Spieleentwicklung, schon aus Performancegründen
  - Objektorientierte Programmierung wichtiges Konzept des Software Engineering
- Direkte Integration in vollständige IDEs
  - Microsoft Visual Studio (Windows), Xcode (Mac)
- Quellcode vollständig erhältlich
  - Vereinfacht das Debugging
  - Leichte Erweiterbarkeit




VS



	Unity	Unreal
Plattform	Windows PC, Mac OS X, Linux, Web Player, WebGL, VR(including Hololens), SteamOS, iOS, Android, Windows Phone 8, SMART TVs, as well as Xbox One & 360, PS4, Playstation Vita, and Wii U	Windows PC, Mac OS X, iOS, Android, VR, Linux, SteamOS, HTML5, Xbox One, and PS4
Code	JavaScript, C#, Boo (kein graphischer Script-Editor)	Blueprint, C++
Editoren		
Graphik	2D/3D	Nur 3D. Mehr Features



# Abschließende Bemerkung

- Ziel der Übungen (und dieser Vorlesungsstunde) ist es nicht, EINE Game Engine (in unserem Fall der  Engine) bis ins letzte Detail zu durchdringen, sondern
- das Grundkonzept von Game Engines zu verstehen
  - Letztendlich sind alle Game Engines ziemlich ähnlich aufgebaut
  - Ebenso wie bei Programmiersprachen gilt: Kennt man eine, kennt man alle
- den Umgang mit diesen wichtigen Tools für die Erstellung digitaler Medien zu erlernen
- Kriterien kennen zu lernen, die Euch bei der Auswahl für die beste Engine für ein bestimmtes Projekt unterstützen