



Leseprobe

Carsten Seifert

Spiele entwickeln mit Unity 5

2D- und 3D-Games mit Unity und C# für Desktop, Web & Mobile

ISBN (Buch): 978-3-446-44563-5

ISBN (E-Book): 978-3-446-44580-2

Weitere Informationen oder Bestellungen unter

<http://www.hanser-fachbuch.de/978-3-446-44563-5>

sowie im Buchhandel.

Inhalt

Vorwort	XXIII
1 Einleitung	1
1.1 Multiplattform-Publishing	1
1.2 Das kann Unity (nicht)	2
1.3 Lizenzmodelle	2
1.4 Aufbau und Ziel des Buches	3
1.5 Weiterentwicklung von Unity	4
1.6 Online-Zusatzmaterial	4
2 Grundlagen	7
2.1 Installation	7
2.2 Oberfläche	7
2.2.1 Hauptmenü	9
2.2.2 Scene View	10
2.2.2.1 Navigieren in der Scene View	11
2.2.2.2 Scene Gizmo	11
2.2.3 Game View	12
2.2.4 Toolbar	13
2.2.5 Hierarchy	15
2.2.5.1 Parenting	16
2.2.6 Inspector	17
2.2.6.1 Kopfinformationen im Inspector	18
2.2.6.2 Variablen Werte zuweisen	18
2.2.6.3 Komponenten-Menüs	19
2.2.6.4 Preview-Fenster	20
2.2.7 Project Browser	20
2.2.7.1 Assets suchen und finden	21
2.2.7.2 Assets importieren	22
2.2.8 Console	22

2.3	Das Unity-Projekt	23
2.3.1	Neues Projekt anlegen	23
2.3.2	Bestehendes Projekt öffnen	24
2.3.2.1	Ältere Projekte öffnen	25
2.3.3	Projektdateien	25
2.3.4	Szene	25
2.3.5	Game Objects	26
2.3.6	Components	27
2.3.7	Tags	28
2.3.8	Layer	29
2.3.9	Assets	29
2.3.9.1	UnityPackage	30
2.3.9.2	Asset Import	30
2.3.9.3	Asset Export	31
2.3.9.4	Asset Store	31
2.3.10	Frames	32
2.4	Das erste Übungsprojekt	33
3	C# und Unity	35
3.1	Die Sprache C#	35
3.2	Syntax	36
3.3	Kommentare	37
3.4	Variablen	37
3.4.1	Namenskonventionen	37
3.4.2	Datentypen	38
3.4.3	Schlüsselwort var	39
3.4.4	Datenfelder/Array	39
3.4.4.1	Arrays erstellen	39
3.4.4.2	Zugriff auf ein Array-Element	40
3.4.4.3	Anzahl aller Array-Items ermitteln	40
3.4.4.4	Mehrdimensionale Arrays	40
3.5	Konstanten	41
3.5.1	Enumeration	41
3.6	Typkonvertierung	42
3.7	Rechnen	43
3.8	Verzweigungen	44
3.8.1	if-Anweisungen	44
3.8.1.1	Komplexere if-Anweisungen	45
3.8.2	switch-Anweisung	46
3.9	Schleifen	47
3.9.1	for-Schleife	47
3.9.1.1	Negative Schrittweite	48
3.9.1.2	break	48

3.9.2	Foreach-Schleife	48
3.9.3	while-Schleife	49
3.9.4	do-Schleife	49
3.10	Klassen	50
3.10.1	Komponenten per Code zuweisen	50
3.10.2	Instanziierung von Nichtkomponenten	51
3.10.3	Werttypen und Referenztypen	53
3.10.4	Überladene Methoden	53
3.11	Der Konstruktor	54
3.11.1	Konstruktoren in Unity	54
3.12	Lokale und globale Variablen	55
3.12.1	Namensverwechslung verhindern mit this	55
3.13	Zugriff und Sichtbarkeit	55
3.14	Statische Klassen und Klassenmember	56
3.15	Parametermodifizierer out/ref	57
3.16	Array-Übergabe mit params	58
3.17	Eigenschaften und Eigenschaftsmethoden	59
3.18	Vererbung	60
3.18.1	Basisklasse und abgeleitete Klassen	60
3.18.2	Vererbung und die Sichtbarkeit	61
3.18.3	Geerbte Methode überschreiben	61
3.18.4	Zugriff auf die Basisklasse	62
3.18.5	Klassen versiegeln	62
3.19	Polymorphie	63
3.20	Schnittstellen	63
3.20.1	Schnittstelle definieren	63
3.20.2	Schnittstellen implementieren	64
3.20.2.1	Unterstützung durch MonoDevelop	65
3.20.3	Zugriff über eine Schnittstellen	65
3.21	Namespaces	66
3.21.1	Eigene Namespaces definieren	66
3.22	Generische Klassen und Methoden	67
3.22.1	List	68
3.22.2	Dictionary	69
4	Skript-Programmierung	71
4.1	MonoDevelop	71
4.1.1	Hilfe in MonoDevelop	72
4.1.2	Syntaxfehler	72
4.2	Nutzbare Programmiersprachen	73
4.2.1	Warum C#?	73
4.3	Unitys Vererbungsstruktur	74
4.3.1	Object	74

4.3.2	GameObject	75
4.3.3	ScriptableObject	75
4.3.4	Component	75
4.3.5	Transform	76
4.3.6	Behaviour	76
4.3.7	MonoBehaviour	76
4.4	Skripte erstellen	76
4.4.1	Skripte umbenennen	77
4.5	Das Skript-Grundgerüst	77
4.6	Unitys Event-Methoden	78
4.6.1	Update	78
4.6.2	FixedUpdate	79
4.6.3	Awake	80
4.6.4	Start	80
4.6.5	OnGUI	80
4.6.6	LateUpdate	81
4.7	Komponentenprogrammierung	81
4.7.1	Auf GameObjects zugreifen	82
4.7.1.1	FindWithTag	82
4.7.1.2	FindGameObjectsWithTag	83
4.7.1.3	Find	83
4.7.2	GameObjects aktivieren und deaktivieren	83
4.7.3	GameObjects zerstören	84
4.7.4	GameObjects erstellen	84
4.7.5	Auf Components zugreifen	84
4.7.5.1	GetComponent	85
4.7.5.2	SendMessage	85
4.7.5.3	Direktzugriff	86
4.7.6	Components hinzufügen	86
4.7.7	Components entfernen	87
4.7.8	Components aktivieren und deaktivieren	87
4.7.9	Attribute	87
4.8	Zufallswerte	89
4.9	Parallel Code ausführen	90
4.9.1	WaitForSeconds	91
4.10	Verzögerte und wiederholende Funktionsaufrufe mit Invoke	91
4.10.1	Invoke	91
4.10.2	InvokeRepeating, IsInvoking und CancelInvoke	92
4.11	Daten speichern und laden	93
4.11.1	PlayerPrefs-Voreinstellungen	93
4.11.2	Daten speichern	94
4.11.3	Daten laden	94
4.11.4	Key überprüfen	95

4.11.5	Löschen	95
4.11.6	Save	95
4.12	Szeneübergreifende Daten	96
4.12.1	Werteübergabe mit PlayerPrefs	96
4.12.1.1	Startmenüs zur Initialisierung nutzen	97
4.12.2	Zerstörung unterbinden	98
4.12.2.1	DontDestroyOnLoad als Singleton	98
4.13	Debug-Klasse	99
4.14	Kompilierungsreihenfolge	100
4.14.1	Programmsprachen mischen und der sprachübergreifende Zugriff	100
4.15	Ausführungsreihenfolge	101
4.16	Plattformabhängig Code kompilieren	101
4.17	Eigene Assets mit ScriptableObject	102
4.17.1	Neue ScriptableObject-Subklasse erstellen	103
4.17.2	Instanzen eines ScriptableObjects erstellen	104
4.17.2.1	Eigener Menü-Knopf zum Erstellen von Assets	104
5	Objekte in der zweiten und dritten Dimension	107
5.1	Das 3D-Koordinatensystem	107
5.2	Vektoren	108
5.2.1	Ort, Winkel und Länge	109
5.2.2	Normalisieren	110
5.3	Das Mesh	111
5.3.1	Normalenvektor	112
5.3.2	MeshFilter und MeshRenderer	112
5.4	Transform	114
5.4.1	Kontextmenü der Transform-Komponente	114
5.4.2	Objekthierarchien	115
5.4.3	Scripting mit Transform	116
5.4.4	Quaternion	116
5.5	Shader und Materials	117
5.5.1	Der Standard-Shader	118
5.5.1.1	Rendering Mode	120
5.5.1.2	Main Maps	121
5.5.1.3	Secondary Maps	127
5.5.1.4	Metallic vs. Specular	129
5.5.2	Texturen	133
5.5.2.1	Import von Texturen	134
5.5.2.2	Normalmaps erstellen	135
5.5.3	UV Mapping	136
5.6	3D-Modelle einer Szene zufügen	138
5.6.1	Primitives	138

5.6.2	3D-Modelle importieren	139
5.6.2.1	Model Import Settings	140
5.6.3	In Unity modellieren	140
5.6.4	Prozedurale Mesh-Generierung	141
5.6.5	Level Of Detail	141
5.6.5.1	LODGroups	142
5.6.5.2	LOD-Qualitätseinstellungen	143
5.7	2D in Unity	143
5.7.1	Sprites	144
5.7.1.1	Sprite Editor	146
5.7.1.2	Sprite Packer	147
5.7.2	SpriteRenderer	149
5.7.2.1	Darstellungsreihenfolgen von Sprites	150
5.7.2.2	Sprite-Shader	151
5.7.3	Parallax Scrolling	152
6	Kameras, die Augen des Spielers	155
6.1	Die Kamera	155
6.1.1	Komponenten eines Kamera-Objektes	157
6.1.2	HDR – High Dynamic Range-Rendering	157
6.1.2.1	Rückwandeln mit Tonemapping	158
6.1.2.2	Abarbeitungsreihenfolge festlegen	159
6.1.3	Linearer- und Gamma-Farbraum	160
6.1.3.1	Die Gamma-Korrektur	160
6.1.3.2	Linear-Space und Gamma-Space in Unity	161
6.2	Kamerasteuerung	163
6.2.1	Statische Kamera	163
6.2.2	Parenting-Kamera	163
6.2.3	Kamera-Skripte	164
6.2.3.1	Kamera-Skripte programmieren	164
6.3	ScreenPointToRay	166
6.4	Mehrere Kameras	166
6.4.1	Kamerawechsel	166
6.4.2	Split-Screen	168
6.4.3	Einfache Minimap	169
6.4.4	Render Texture	170
6.5	Image Effects	172
6.5.1	Beispiel: Haus bei Nacht	172
6.6	Skybox	174
6.6.1	Mehrere Skyboxen gleichzeitig einsetzen	175
6.6.2	Skybox selber erstellen	176
6.7	Occlusion Culling	177
6.7.1	Occluder Static und Occludee Static	178
6.7.2	Occlusion Culling erstellen	179

7	Licht und Schatten	181
7.1	Ambient Light	181
7.2	Lichtarten	182
7.2.1	Directional Light	183
7.2.2	Point Light	184
7.2.3	Spot Light	184
7.2.4	Area Light	185
7.3	Schatten	186
7.3.1	Einfluss des MeshRenderers auf Schatten	187
7.4	Light Cookies	188
7.4.1	Import Settings eines Light Cookies	188
7.4.2	Light Cookies und Point Lights	189
7.5	Light Halos	190
7.5.1	Unabhängige Halos	191
7.6	Lens Flares	191
7.6.1	Eigene Lens Flares	192
7.7	Projector	192
7.7.1	Standard Projectors	192
7.8	Lightmapping	194
7.8.1	Light Probes	196
7.9	Rendering Paths	198
7.9.1	Forward Rendering	199
7.9.2	Vertex Lit	200
7.9.3	Deferred Lighting	201
7.10	Global Illumination	202
7.10.1	Baked GI	202
7.10.2	Precomputed Realtime GI	203
7.11	Reflexionen (Spiegelungen)	203
7.11.1	Reflection Probes	204
7.12	Qualitätseinstellungen	207
7.12.1	Quality Settings	207
7.12.2	Qualitätsstufen per Code festlegen	207
8	Physik in Unity	209
8.1	Physikberechnung	209
8.2	Rigidbody	210
8.2.1	Rigidbody kennenlernen	211
8.2.2	Massenschwerpunkt	212
8.2.3	Kräfte und Drehmomente zufügen	213
8.2.3.1	AddForce-Methode	213
8.2.3.2	AddTorque-Methode	214
8.2.3.3	ConstantForce-Komponente	216

8.3	Kollisionen	216
8.3.1	Collider	216
8.3.1.1	Mesh Collider	217
8.3.1.2	Collider modifizieren	218
8.3.1.3	OnCollision-Methoden	219
8.3.2	Trigger	220
8.3.3	Static Collider	221
8.3.4	Kollisionen mit schnellen Objekten	221
8.3.5	Terrain Collider	223
8.3.6	Layer-basierende Kollisionserkennung	223
8.3.7	Mit Layer-Masken arbeiten	223
8.4	Wheel Collider	225
8.4.1	Wheel Friction Curve	226
8.4.2	Entwicklung einer Fahrzeugsteuerung	228
8.4.2.1	CarController.cs	233
8.4.3	Autokonfiguration	235
8.4.4	Fahrzeugstabilität	237
8.5	Physic Materials	238
8.6	Joints	239
8.6.1	Fixed Joint	239
8.6.2	Spring Joint	239
8.6.3	Hinge Joint	239
8.7	Raycasting	240
8.8	Character Controller	241
8.8.1	SimpleMove	242
8.8.2	Move	243
8.8.3	Kräfte zufügen	244
8.8.4	Einfacher First Person Controller	245
8.9	2D-Physik	247
8.9.1	OnCollision2D- und OnTrigger2D-Methoden	248
8.9.2	2D Physic Effectors	250
9	Maus, Tastatur, Touch	251
9.1	Virtuelle Achsen und Tasten	251
9.1.1	Der Input-Manager	251
9.1.2	Virtuelle Achsen	253
9.1.3	Virtuelle Tasten	253
9.1.4	Steuern mit Mauseingaben	254
9.1.5	Joystick-Inputs	254
9.1.6	Anlegen neuer Inputs	255
9.2	Achsen- und Tasteneingaben auswerten	255
9.2.1	GetAxis	255
9.2.2	GetButton	256

9.3	Tastatureingaben auswerten	257
9.3.1	GetKey	257
9.3.2	anyKey	257
9.4	Mauseingaben auswerten	258
9.4.1	GetMouseButton	258
9.4.2	mousePosition	259
9.4.3	Mauszeiger ändern	260
9.5	Touch-Eingaben auswerten	261
9.5.1	Der Touch-Typ	261
9.5.2	Input.touches	262
9.5.3	TouchCount	262
9.5.4	GetTouch	263
9.5.5	CrossPlatformInput	263
9.6	Beschleunigungssensor auswerten	265
9.6.1	Input.acceleration	265
9.6.2	Tiefpass-Filter	266
9.7	Steuerungen bei Mehrspieler-Games	267
9.7.1	Split-Screen-Steuerung	267
9.7.2	Netzwerkspiele	268
10	Audio	269
10.1	AudioListener	269
10.2	AudioSource	270
10.2.1	Durch Mauern hören verhindern	272
10.2.2	Sound starten und stoppen	274
10.2.3	Temporäre AudioSource	275
10.3	AudioClip	276
10.3.1	Länge ermitteln	276
10.4	Reverb Zone	276
10.5	Filter	278
10.6	Audio Mixer	278
10.6.1	Das Audio Mixer-Fenster	278
10.6.1.1	Die Audio-Steuerungseinheit	280
10.6.2	Audiosignalwege	282
10.6.2.1	Send & Receive	283
10.6.2.2	Duck Volume	284
10.6.3	Mit Snapshots arbeiten	286
10.6.4	Views erstellen	287
11	Partikeleffekte mit Shuriken	289
11.1	Editor-Fenster	290
11.2	Particle Effect Control	291
11.3	Numerische Parametervarianten	291

11.4	Farbparameter-Varianten	292
11.5	Default-Modul	292
11.6	Effekt-Module	293
11.6.1	Emission	294
11.6.2	Shape	294
11.6.2.1	Sphere	294
11.6.2.2	HemiSphere	294
11.6.2.3	Cone	294
11.6.2.4	Box	295
11.6.2.5	Mesh	295
11.6.3	Velocity over Lifetime	295
11.6.4	Limit Velocity over Lifetime	296
11.6.5	Force over Lifetime	296
11.6.6	Color over Lifetime	296
11.6.7	Color by Speed	296
11.6.8	Size over Lifetime	297
11.6.9	Size by Speed	297
11.6.10	Rotation over Lifetime	297
11.6.11	Rotation by Speed	297
11.6.12	External Forces	297
11.6.13	Collision	298
11.6.13.1	Planes	298
11.6.13.2	World	299
11.6.14	Sub Emitter	299
11.6.15	Texture-Sheet-Animation	299
11.6.16	Renderer	300
11.6.16.1	Billboard	301
11.6.16.2	Stretched Billboard	301
11.6.16.3	Horizontal Billboard	301
11.6.16.4	Vertical Billboard	301
11.6.16.5	Mesh	302
11.7	Partikelemission starten, stoppen und unterbrechen	302
11.7.1	Play	302
11.7.2	Stop	303
11.7.3	Pause	303
11.7.4	enableEmission	303
11.8	OnParticleCollision	303
11.8.1	GetCollisionEvents	304
11.9	Feuer erstellen	304
11.9.1	Materials erstellen	305
11.9.2	Feuer-Partikelsystem	306
11.9.3	Rauch-Partikelsystem	309
11.10	Wassertropfen erstellen	312
11.10.1	Tropfen-Material erstellen	312

11.10.2	Wassertropfen-Partikelsystem	313
11.10.3	Kollisionspartikelsystem	315
11.10.4	Kollisionsound	317
12	Landschaften gestalten	319
12.1	Was Terrains können und wo die Grenzen liegen	320
12.2	Terrainhöhe verändern	320
12.2.1	Pinsel	321
12.2.2	Oberflächen anheben und senken	321
12.2.3	Plateaus und Schluchten erstellen	322
12.2.4	Oberflächen weicher machen	323
12.2.5	Heightmaps	323
	12.2.5.1 Reale Landschaften als Vorlagen nutzen	324
12.3	Terrain texturieren	325
12.3.1	Textur-Pinsel	326
12.3.2	Texturen verwalten	326
12.4	Bäume und Sträucher	328
12.4.1	Bedienung des Place Tree-Tools	328
12.4.2	Wälder erstellen	329
12.4.3	Mit Bäumen kollidieren	329
12.5	Gräser und Details hinzufügen	330
12.5.1	Detail-Meshs	330
12.5.2	Gräser	331
12.5.3	Quelldaten nachladen	332
12.6	Terrain-Einstellungen	332
12.6.1	Base Terrain	333
12.6.2	Resolution	333
12.6.3	Tree & Details Objects	334
12.6.4	Wind Settings	334
12.6.5	Zur Laufzeit Terrain-Eigenschaften verändern	335
12.7	Der Weg zum perfekten Terrain	336
12.8	Gewässer	337
13	Wind Zones	339
13.1	Spherical vs. Directional	340
13.2	Wind Zone - Eigenschaften	341
13.3	Frische Brise	342
13.4	Turbine	342
14	GUI	343
14.1	Das UI-System uGUI	344
14.1.1	Canvas	344
	14.1.1.1 Render Modes	345
	14.1.1.2 Canvas Scaler	347

14.1.2	RectTransform	348
14.1.2.1	Anchors	350
14.1.3	UI-Sprite Import	352
14.1.4	Grafische Controls	353
14.1.4.1	Text	353
14.1.4.2	Image	354
14.1.4.3	Raw Image	355
14.1.4.4	Panel	355
14.1.4.5	Effects	355
14.1.5	Interaktive Controls	356
14.1.5.1	Button	359
14.1.5.2	InputField	360
14.1.5.3	Toggle und Toggle Group	360
14.1.5.4	Slider	361
14.1.5.5	Scrollbar	362
14.1.6	Controls designen	362
14.1.7	Animationen in uGUI	363
14.1.8	Event Trigger	364
14.2	Screen-Klasse	365
14.2.1	Schriftgröße dem Bildschirm anpassen	366
14.3	OnGUI-Programmierung	366
14.3.1	GUI	367
14.3.1.1	Label	367
14.3.1.2	Button und RepeatButton	368
14.3.1.3	TextField und TextArea	368
14.3.1.4	Subfenster	369
14.3.2	GUILayout	370
14.3.2.1	Ausrichtung festlegen	370
14.3.3	GUIStyle und GUISkin	371
15	Prefabs	373
15.1	Prefabs erstellen und nutzen	373
15.2	Prefab-Instanzen erzeugen	373
15.2.1	Instanzen per Code erstellen	374
15.2.2	Instanzen weiter bearbeiten	374
15.3	Prefabs ersetzen und zurücksetzen	375
16	Internet und Datenbanken	377
16.1	Die WWW-Klasse	377
16.1.1	Rückgabewert-Formate	378
16.1.2	Parameter übergeben	379
16.1.2.1	WWWForm	379
16.2	Datenbank-Kommunikation	380
16.2.1	Daten in einer Datenbank speichern	380

16.2.2	Daten von einer Datenbank abfragen	381
16.2.3	Rückgabewerte parsen	383
16.2.4	Datenhaltung in eigenen Datentypen	383
16.2.5	HighscoreCommunication.cs	385
16.2.6	Datenbankverbindung in PHP	387
17	Animationen	389
17.1	Allgemeiner Animation-Workflow	390
17.2	Animationen erstellen	390
17.2.1	Animation View	391
17.2.2	Curves vs. Dope Sheet	392
17.2.3	Animationsaufnahme	392
17.2.3.1	Ansicht im Animation-Fenster anpassen	393
17.2.3.2	Sprite-Animationen	393
17.2.3.3	Root Motion-Kurven erstellen	396
17.2.4	Beispiel Fallgatter-Animation	397
17.3	Animationen importieren	398
17.3.1	Rig	399
17.3.1.1	Generic	399
17.3.1.2	Humanoid	399
17.3.2	Animationen	401
17.3.2.1	Animationen aufteilen	401
17.3.2.2	Importierte Animation-Clips bearbeiten	402
17.3.2.3	Mask	404
17.3.2.4	Events	404
17.4	Animationen einbinden	404
17.4.1	Animator Controller	404
17.4.1.1	Animation States erstellen	405
17.4.1.2	Transitions – Animation States wechseln	406
17.4.1.3	Any State nutzen	408
17.4.1.4	Blend Trees	408
17.4.1.5	Layer und Avatar Masks	409
17.4.1.6	Sub-State Machines	411
17.4.1.7	State Machine Behaviours	416
17.4.2	Animator-Komponente	420
17.4.3	Beispiel Fallgatter: Animator Controller	421
17.5	Controller-Skripte	423
17.5.1	Parameter des Animator Controllers setzen	423
17.5.1.1	Mit Hash-Werten arbeiten	424
17.5.2	Animation States abfragen	424
17.5.3	Beispiel Fallgatter Controller-Skript	425
17.6	Animation Events	427

18	Künstliche Intelligenz	429
18.1	NavMeshAgent	430
18.1.1	Eigenschaften der Navigationskomponente	431
18.1.2	Zielpunkt zuweisen	431
18.1.3	Pfadsuche unterbrechen und fortsetzen	432
18.2	NavigationMesh	432
18.2.1	Object Tab	434
18.2.2	Bake Tab	434
18.2.3	Areas Tab	435
18.3	NavMeshObstacle	436
18.4	Off-Mesh Link	437
18.4.1	Automatische Off-Mesh Links	437
18.4.2	Manuelle Off-Mesh Links	438
18.5	Point & Click-Steuerung für Maus und Touch	439
19	Fehlersuche und Performance	443
19.1	Fehlersuche	443
19.1.1	Breakpoints	444
19.1.2	Variablen beobachten	445
19.1.3	Console Tab nutzen	445
19.1.4	GUI- und GUILayout nutzen	446
19.2	Performance	446
19.2.1	Rendering-Statistik	447
19.2.2	Batching-Verfahren	448
19.2.3	Analyse mit dem Profiler	450
19.2.4	Echtzeit-Analyse auf Endgeräten	451
19.2.4.1	Webplayer-Game remote analysieren	451
19.2.4.2	Android-Game remote analysieren	452
19.2.4.3	iOS-Game remote analysieren	452
20	Spiele erstellen und publizieren	453
20.1	Der Build-Prozess	453
20.1.1	Szenen des Spiels	454
20.1.2	Plattformen	455
20.1.3	Notwendige SDKs	455
20.1.4	Plattformspezifische Optionen	456
20.1.5	Developer Builds	456
20.1.5.1	Autoconnect Profiler	456
20.1.5.2	Script Debugging	456
20.2	Publizieren	457
20.2.1	App	458
20.2.2	Browser-Game	458
20.2.3	Desktop-Anwendung	459

21	Erstes Beispiel-Game: 2D-Touch-Game	461
21.1	Projekt und Szene	461
21.1.1	Die Kamera	463
21.1.2	Texturen importieren und Sprites definieren	463
21.2	Gespenster und Hintergrund	465
21.2.1	Gespenster animieren	468
21.2.2	Gespenster laufen lassen	472
21.2.3	Gespenster-Prefab erstellen	474
21.3	Der GameController	474
21.3.1	Der Spawner	475
21.3.2	Level-Anzeige	477
21.3.3	Der Input-Controller	478
21.3.4	Game Over-UI	480
21.3.5	Hintergrundmusik	486
21.4	Punkte zählen	487
21.5	Spielende	488
21.6	Spiel erstellen	489
22	Zweites Beispiel-Game: 3D Dungeon Crawler	491
22.1	Level-Design	492
22.1.1	Modellimport	493
22.1.1.1	Allgemeine Import Settings	493
22.1.1.2	bat_03 Import Settings	493
22.1.1.3	crate_01 Import Settings	494
22.1.1.4	stone_01 Import Settings	494
22.1.1.5	floor_01 Import Settings	494
22.1.2	Materials konfigurieren	494
22.1.3	Prefabs erstellen	495
22.1.3.1	Wall Prefab	495
22.1.3.2	Torch Prefab	495
22.1.3.3	Wall_Torch Prefab	496
22.1.3.4	Floor Prefab	496
22.1.3.5	Ceiling Prefab	497
22.1.4	Dungeon erstellen	497
22.1.4.1	Boden erstellen	497
22.1.4.2	Wände erstellen	498
22.1.4.3	Decke erstellen	499
22.1.4.4	Allgemeine Licht-Einstellungen	499
22.1.4.5	Fackelbeleuchtung	499
22.1.4.6	Akzent-Beleuchtung	501
22.1.5	Dekoration erstellen	502
22.1.5.1	Crate Prefab	502
22.1.5.2	Barrel Prefabs	502

22.2	Inventarsystem erstellen	503
22.2.1	Verwaltungslogik	503
22.2.1.1	InventoryItem.cs	507
22.2.1.2	CreateInventoryItem.cs	507
22.2.1.3	Inventory.cs	508
22.2.1.4	PickableItem.cs	510
22.2.2	Oberfläche des Inventarsystems	510
22.2.3	Inventar-Items	514
22.2.3.1	HoverEffects.cs	519
22.3	Game Controller	520
22.4	Spieler erstellen	521
22.4.1	Lebensverwaltung	522
22.4.1.1	LifePointController.cs	530
22.4.1.2	HealthController.cs	531
22.4.1.3	PlayerHealth.cs	531
22.4.2	Spielersteuerung	533
22.4.2.1	PlayerController.cs	538
22.4.2.2	Footsteps.cs	539
22.4.2.3	Shooting.cs	540
22.4.3	Wurfstein entwickeln	541
22.4.3.1	StoneBehaviour.cs	544
22.4.3.2	Zerberstenden Stein konfigurieren	545
22.4.4	Lautstärke steuern	547
22.5	Quest erstellen	548
22.5.1	Erfahrungspunkte verwalten	548
22.5.1.1	EPController.cs	549
22.5.2	Questgeber erstellen	550
22.5.2.1	WaterQuest.cs	555
22.5.2.2	WaterdropSound.cs	557
22.5.2.3	InGameMenu.cs	557
22.5.3	Sub-Quest erstellen	558
22.5.3.1	AnimateDoor.cs	562
22.6	Gegner erstellen	564
22.6.1	Model-, Rig- und Animationsimport	564
22.6.2	Komponenten und Prefab konfigurieren	565
22.6.3	Animator Controller erstellen	567
22.6.4	NavMesh erstellen	569
22.6.5	Umgebung und Feinde erkennen	570
22.6.5.1	EnemySonar.cs	571
22.6.6	Gesundheitszustand verwalten	572
22.6.6.1	EnemyHealth.cs	575
22.6.7	Künstliche Intelligenz entwickeln	576
22.6.7.1	EnemyAI.cs	582

22.7	Eröffnungsszene	585
22.7.1	Szene erstellen	585
22.7.2	Startmenü-Logik erstellen	586
22.7.2.1	MainMenu.cs	587
22.7.3	Menü-GUI erstellen	588
22.8	Web-Player-Anpassungen	590
22.8.1	Web-Player-Template ändern	590
22.8.2	Quit-Methode im Web-Player abfangen	590
22.9	Finale Einstellungen	591
22.10	So könnte es weitergehen	594
23	Der Produktionsprozess in der Spieleentwicklung	595
23.1	Die Produktionsphasen	595
23.1.1	Ideen- und Konzeptionsphase	596
23.1.2	Planungsphase	596
23.1.3	Entwicklungsphase	596
23.1.4	Testphase	597
23.1.5	Veröffentlichung und Postproduktion	597
23.2	Das Game-Design-Dokument	597
24	Schlusswort	599
Index	601

Vorwort

Für viele von uns sind Computerspiele heutzutage allgegenwärtige Wegbegleiter. Egal wo, auf dem Smartphone, dem Tablet oder dem heimischen PC sind sie installiert und täglich in Benutzung. Manchmal dienen sie als Zeitvertreib, bis der nächste Bus kommt, manchmal sind sie aber auch Bestandteil eines intensiven Hobbys.

Aber nicht nur das Spielen kann Spaß machen, auch das Entwickeln dieser Games kann begeistern. Sowohl im Freizeitbereich als auch in der Arbeitswelt wird der Beruf des Spieleentwicklers immer beliebter. Es ist also kein Wunder, dass sich in den letzten Jahren bereits ganze Studiengänge dem Entwickeln von Computerspielen gewidmet haben.

In diesem Buch möchte ich Ihnen Unity, eine weit verbreitete Entwicklungsumgebung für Computerspiele, näherbringen und erläutern, wie Sie mit diesem Werkzeug Spiele selber entwickeln können. Dabei richtet sich das Buch sowohl an Einsteiger, Umsteiger und auch an Spieleentwickler, die mit Unity nun richtig durchstarten möchten.

An dieser Stelle möchte ich mich ganz besonders bei meiner Frau Cornelia bedanken, die mich während des Schreibens so geduldig unterstützt hat und mir jederzeit beim Formulieren und Korrigieren hilfsbereit zur Seite stand.

Auch danke ich ganz herzlich Alexej Bodemer, der für das Beispiel-Game dieses Buches alle 3D-Modelle, Texturen und Musikdateien entworfen und zur Verfügung gestellt hat.

Zudem gilt mein Dank Will Goldstone und Unity Technologies, die mir die bis dato aktuellsten Beta-Versionen zur Verfügung gestellt haben.

Weiter möchte ich Sieglinde Schär, Kristin Rothe und dem gesamten Hanser-Verlag-Team danken, die mir nicht nur das Schreiben dieses Buches ermöglicht haben, sondern auch jederzeit mit Rat und Tat zur Seite standen.

Nicht zuletzt danke ich auch meiner gesamten Community, die mich während des Schreibens auf meinem Blog und meinen sozialen Kanälen so konstruktiv begleitet hat.

Süderbrarup, Juni 2015

Carsten Seifert

7

Licht und Schatten

Damit eine Kamera die Textur eines *Mesh* und damit auch dessen Form darstellen kann, benötigt Ihre Szene zunächst einmal Licht. Licht hat wiederum einen enormen Einfluss auf die Darstellung der Textur, man denke nur an die Helligkeit, Position und Ausrichtung der Lichtquelle.

Unity bietet hierfür verschiedene Arten von sogenannten *Light*-Objekten an, die z. B. Echtzeitschatten erzeugen und mit anderen Effekten ausgestattet werden können. Neben dieser Echtzeitbeleuchtung unterstützt Unity auch *Lightmapping*, ein Verfahren, mit dem Sie Texturen generieren können, die beleuchtete Flächen vortäuschen. Da Lichtberechnungen sehr rechenintensiv sind, kann durch das Nutzen von *Lightmapping* die Performance erheblich gesteigert werden.

■ 7.1 Ambient Light

Unity besitzt eine globale Beleuchtung namens *Ambient Light*, die die komplette Szene mit einer Grundhelligkeit ausstattet. Die Einstellungen für diese Beleuchtung finden Sie im Lighting-Fenster, das Sie über das Menü **WINDOWS/LIGHTING** erreichen. Im Bereich „Environment Lighting“ (siehe Bild 7.1) finden Sie die Parameter des *Ambient Light*:

- **Ambient Source** legt die Quelle bzw. die Farbe(n) des Lichts fest. Sie haben die Wahl zwischen „Skybox“ (hier werden die Farben der *Skybox* als Grundlage genommen), „Gradient“ (erlaubt Ihnen, einen Farbübergang mit maximal drei Farben zu bestimmen) und „Color“ (bei dem eine einzige Farbe festgelegt wird).
- **Ambient Intensity** bestimmt die Lichtstärke des *Ambient Light*.
- **Ambient GI** legt fest, ob das *Ambient Light* zur Laufzeit berechnet wird oder beim *Baken* der *Lightmaps* erstellt wird. Da das Licht entweder über die *Global Illumination*-Verfahren *Precomputed Realtime GI* oder *Baked GI* gesteuert wird, müssen Sie hier nur eine Auswahl treffen, wenn beide im *Lighting*-Fenster aktiv sind. Ist nur eine aktiviert, wird diese automatisch genommen. Mehr zu diesen Verfahren lesen Sie im Abschnitt „Global Illumination“.

Möchten Sie ein Spiel entwickeln, das keine Grundbeleuchtung besitzen soll, weil es beispielsweise im Dunkeln spielt, können Sie die *Ambient Intensity* auf 0 stellen. Oder Sie stellen die *Ambient Source* auf „Color“ und setzen die Farbe auf Schwarz.



Bild 7.1
Lighting-Fenster

■ 7.2 Lichtarten

Über das Menü **GAMEOBJECT/LIGHT** können Sie vier unterschiedliche Beleuchtungsobjekte Ihrer Szene zufügen: *Directional Light*, *Point Light*, *Spot Light* und das *Area Light*. Letzteres nimmt eine Sonderrolle ein, worauf ich noch eingehen werde. Alle Objekte besitzen eine *Light*-Komponente, die das Herzstück einer Lichtquelle darstellt. Die Komponente besitzt unterschiedliche Parameter, die je nach *Type* der *Light*-Komponente zur Verfügung stehen. Die folgenden Parameter stehen aber mit Ausnahme des *Area Lights* immer zur Auswahl:

- **Type** bestimmt die Art des Lichtes und damit auch die zur Verfügung stehenden Parameter.
- **Baking** bestimmt das Verhalten beim Erstellen und Nutzen von *Lightmaps*. *Realtime* schließt dieses Licht beim Erstellungsprozess der *Lightmaps* (auch *baken* genannt) aus. *Mixed* berücksichtigt das Licht beim *Lightmapping*, ist aber auch zur Laufzeit im normalen Spiel aktiv, um die nichtstatischen Objekte zu beleuchten. *Baked* bindet das Licht in das *Baken* ein, deaktiviert es aber während des normalen Spiels.
- **Range** setzt die Reichweite der Lichtquelle fest.
- **Intensity** definiert die Lichtstärke.
- **Bounce Intensity** bestimmt die Helligkeit des Lichts, das von angestrahlten Flächen zurückgeworfen wird, auch indirekte Beleuchtung genannt. Mehr zu diesem Parameter erfahren Sie im Abschnitt „Global Illumination“.
- **Color** legt die Lichtfarbe fest.

- **Cookie** ermöglicht, eine Lichtschablone vor eine Lichtquelle zu legen. Dies ist je nach *Type* entweder eine einzelne Schwarz-Weiß-Grafik oder eine *Cubemap*, bestehend aus sechs solcher Grafiken.
- **Shadow Type** legt die Art des Schattens fest. Allgemein stehen *No Shadows*, *Hard Shadows* und *Soft Shadows* zur Verfügung. Auf die Details werde ich gleich noch eingehen.
- **Draw Halo** bestimmt, ob ein *Light Halo* dargestellt werden soll.
- **Flare** legt ein *Flare*-Objekt zum Darstellen eines Lichtscheins für diese Lichtquelle fest (siehe „Flare“).
- **Render Mode** legt fest, ob dieses Licht beim *Forward Rendering* per Pixel oder per Vertex gerendert werden soll. *Auto*: Unity bestimmt, auf welche Art gerendert wird. *Important* bedeutet per Pixel, *Not Important* bedeutet per Vertex. Der Wert *Pixel Light Count* (zu finden in den *Quality Settings*) bestimmt bei *Auto*, wie viele Lichtquellen insgesamt per Vertex gerendert werden können.
- **Culling Mask** definiert, welche Objekte eines *Layers* nicht von dieser Lichtquelle beeinflusst/beleuchtet werden.

7.2.1 Directional Light

Ein *Directional Light* beleuchtet die komplette Szene aus einer Richtung. Die Position der Lichtquelle spielt dabei keine Rolle, nur die Rotation der Quelle ist hierbei wichtig. Ein *Directional Light* kann sowohl mit *Forward Rendering* als auch *Deferred Lighting* (siehe Abschnitt „Rendering Paths“) Echtzeitschatten erzeugen.

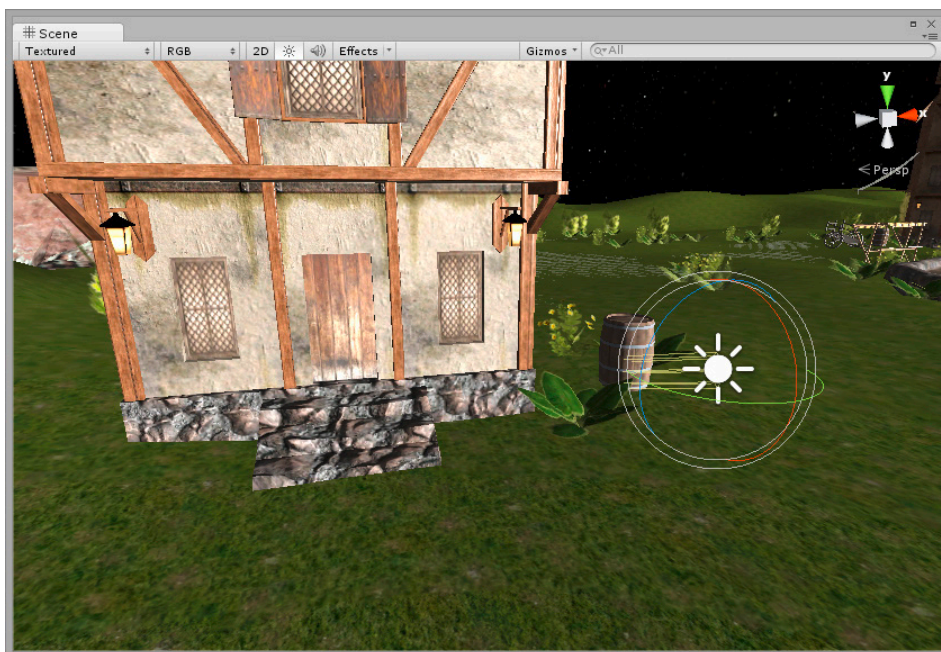


Bild 7.2 Directional Light

7.2.2 Point Light

Das *Point Light* ist eine Lichtquelle, die in alle Richtungen gleichmäßig abstrahlt, vergleichbar mit einer Glühlampe an der Decke. Im Gegensatz zum *Directional Light* ist hier die Position, aber nicht die Rotation wichtig. Über den Parameter *Range* legen Sie die Reichweite der Lichtquelle fest. Echtzeitschatten von *Point Lights* werden nur in Kombination mit dem *Rendering Path Deferred Lighting* unterstützt.

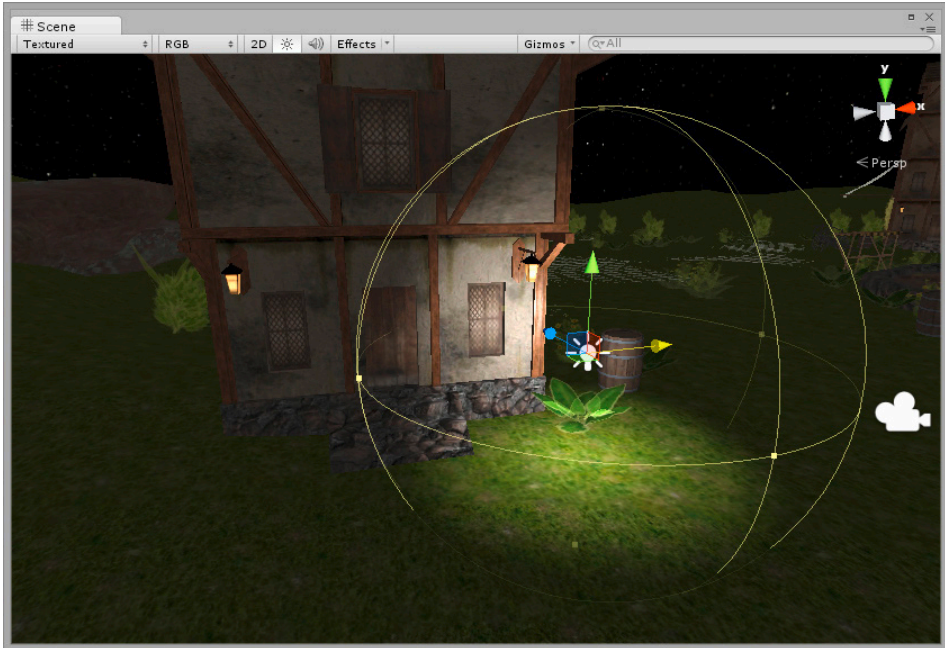


Bild 7.3 Point Light

7.2.3 Spot Light

Ein *Spot Light* scheint trichterförmig von der Lichtquelle in eine bestimmte Richtung, ähnlich wie eine Taschenlampe. Über den Parameter *Range* legen Sie fest, wie weit sie scheint. *Spot Angle* legt den äußeren Abstrahlwinkel fest. Echtzeitschatten von *Spot Lights* werden wie beim *Point Light* ebenfalls nur in Kombination mit dem *Rendering Path Deferred Lighting* unterstützt.

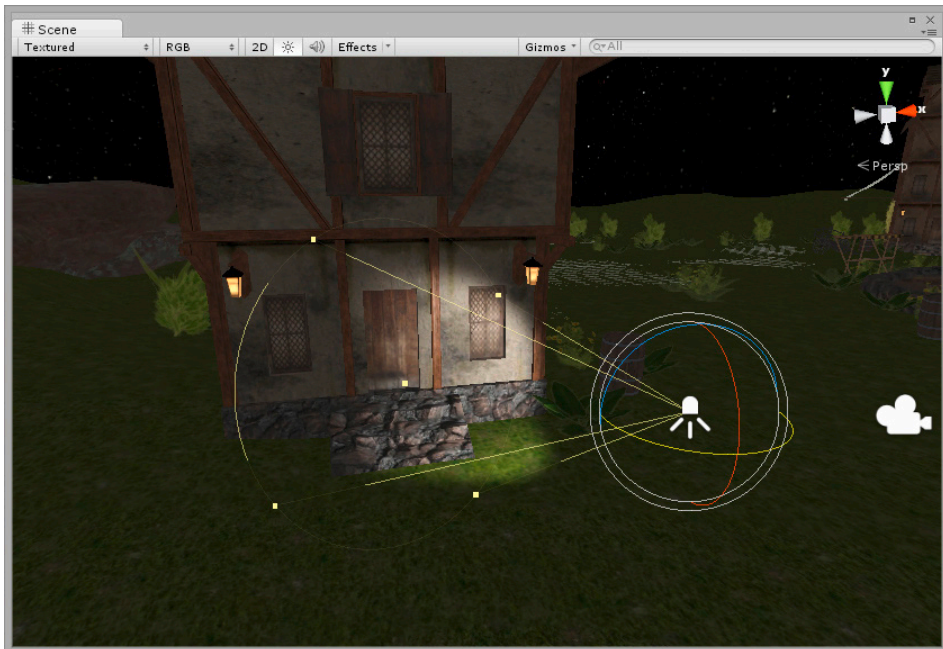


Bild 7.4 Spot Light

7.2.4 Area Light

Ein *Area Light* nimmt eine Sonderstellung bei den *Light Types* ein, da es ausschließlich beim Erstellen von *Lightmaps* berücksichtigt wird, nicht aber zur Echtzeit virtuelles Licht emittiert. Diese Lichtquellen arbeiten also ausschließlich so, als wenn der *Baking*-Parameter auf „Baken“ gestellt wäre.

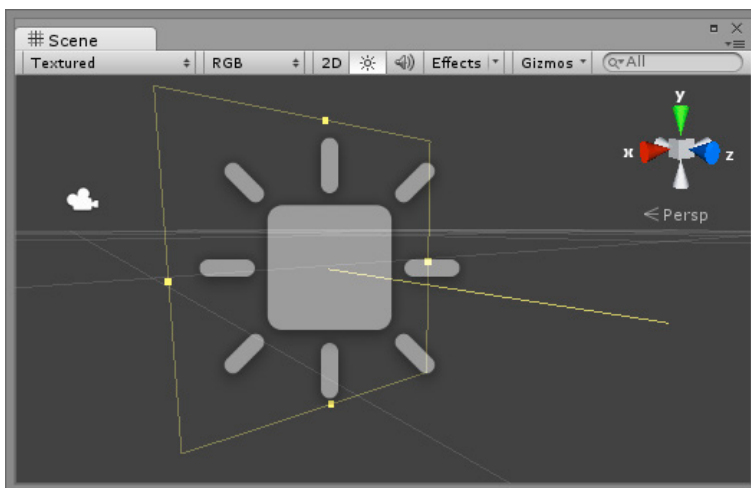


Bild 7.5 Area Light

Ein *Area Light* erscheint nach dem Erstellen der Lightmap wie eine Rechteckfläche, die in alle Richtungen einer Seite scheint, vergleichbar mit dem Bildschirm eines Fernsehers. Die Richtung wird mit einer Linie dargestellt (siehe Bild 7.5). Die Fläche wird durch die Parameter *Width* und *Height* festgelegt. Und auch hier wird die Reichweite der Beleuchtung über die *Intensity* definiert.

■ 7.3 Schatten

Unity bietet zwei unterschiedliche Echtzeitschattenarten an: *Hard* und *Soft Shadows*. *Hard Shadows* sind eine performance-schonende Variante, *Soft Shadows* sind dafür detaillierter. Zudem können Sie einer Lichtquelle auch den Parameter *No Shadows* mitgeben. In dem Fall werfen angestrahlte Objekte überhaupt keine Schatten.

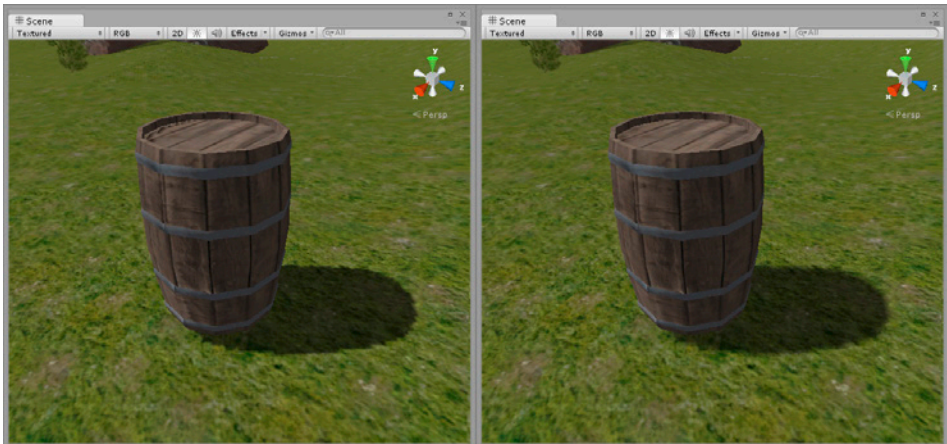


Bild 7.6 Vergleich: Hard Shadows vs. Soft Shadows

Neben der Schattenart bieten die Lichtquellen noch weitere Parameter an.

- **Strength** legt die Dunkelheit des Schattens fest.
- **Resolution** definiert die Qualität bzw. die Auflösung des Schattens. Standardmäßig wird hier für die Einstellung auf die *Quality Settings (EDIT/PROJECT SETTINGS/QUALITY)* verwiesen. Sie kann aber auch überschrieben werden.
- **Bias** hat einen Einfluss auf den Abstand des Objektes zum Schatten. Beginnt der Schatten zu nah am Objekt, kann dies zu optischen Fehlern führen. Deshalb ist standardmäßig ein Wert von 0,05 vorgegeben.

7.3.1 Einfluss des MeshRenderers auf Schatten

Über die *MeshRenderer*-Komponente eines jeden sichtbaren Objektes haben Sie die Möglichkeit zu steuern, ob ein Objekt überhaupt Schatten erzeugen soll oder nicht. Dies können Sie über die Eigenschaft **Cast Shadows** steuern. Genauso können Sie über die Eigenschaft **Receive Shadows** definieren, ob auf dem Objekt selber Schatten anderer Objekte dargestellt werden sollen. Wenn Sie beispielsweise einem Spieler einen Unsichtbarkeitszauber zuführen, darf dieser selber keine Schatten werfen, aber auch keine Schatten darstellen, die andere Objekte auf ihn werfen. In diesem Fall wird der Schatten einfach zum nächsten Objekt „durchgeleitet“, sodass dort der Schatten dargestellt wird. Das Bild 7.7 zeigt zwei Fässer. Während beim linken Fass beide Parameter aktiv sind und dieses einen Schatten wirft, wurden beim rechten Fass beide Parameter deaktiviert. Der Schatten des linken Objektes wird deshalb nicht auf dem rechten Fass dargestellt. Zudem wirft das rechte Fass auch selber keinen Schatten, sodass nur der Schattenwurf des linken gezeigt wird.

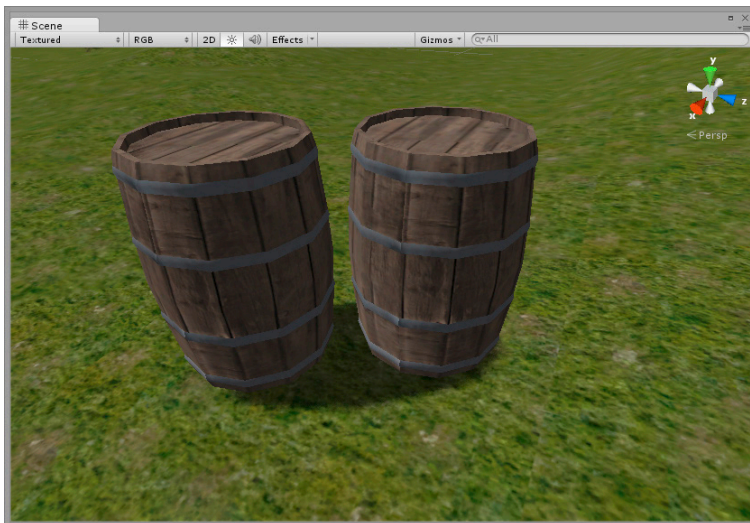


Bild 7.7 Einfluss des „Cast Shadows“- und „Receive Shadows“-Parameter

Der Parameter *Cast Shadows* bietet noch weitere Einstellmöglichkeiten an. Neben *On* und *Off* können Sie auch *Two Sided* auswählen. In diesem Fall wirft das *Mesh* in beide Richtungen einen Schatten. Haben Sie beispielsweise eine Ebene, um die Sie eine Lichtquelle rotieren lassen, würde der Schatten sich genauso verhalten, wie Sie es erwarten würden, auch wenn das *Mesh* selber vielleicht nur aus einer Richtung zu sehen ist (siehe „Normalenvektor“ im Kapitel 5 „Objekte in der zweiten und dritten Dimension“). Als dritte Auswahlmöglichkeit können Sie *Shadows Only* auswählen. In diesem Fall wird nur der Schatten dargestellt, nicht aber das *Mesh*.

■ 7.4 Light Cookies

Bei einem *Light Cookie* handelt es sich um eine Lichtschablone, die das abgegebene Licht in bestimmte Formen bringt. Wenn Sie ein Comic-Fan sind, dann kennen Sie sicher das Batman-Zeichen, das die Polizei von Gotham-City an den Himmel wirft, um Batman zu rufen. Genau das ist ein *Light Cookie*, genauer gesagt ein *Spot Light* mit einem *Light Cookie*. Das Bild 7.8 zeigt Ihnen einen ähnlichen Effekt. Dort wurde im rechten Motiv ein *Spot Light* mit einem *Light Cookie* versehen, das einen taschenlampenähnlichen Effekt erzeugt. Diesen wie auch weitere *Light Cookies* liefert Unity in seinen *Standard Assets* „Effects“ mit.

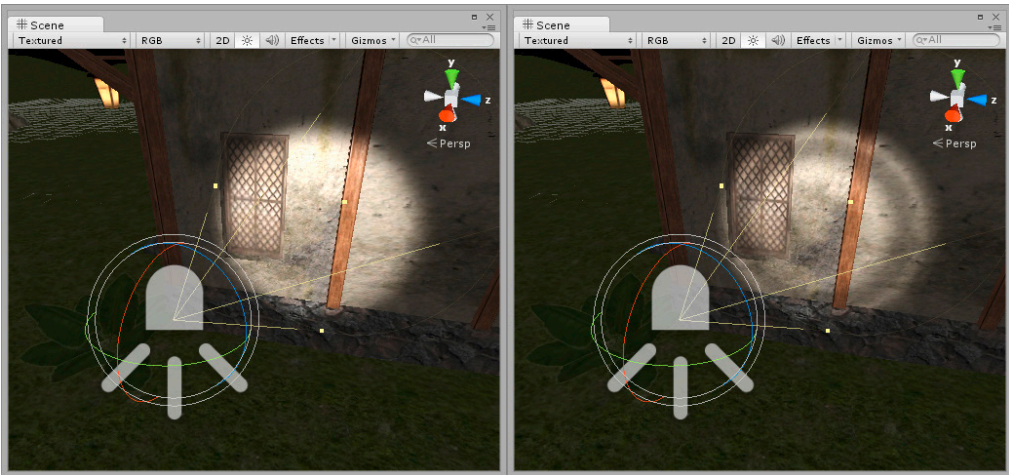


Bild 7.8 Spot Light mit einem taschenlampenähnlichen Light Cookie

7.4.1 Import Settings eines Light Cookies

Der Kern eines *Light Cookies* ist eine Schwarz-Weiß-Grafik, auch *Grayscale Texture* genannt. Schwarz bedeutet hierbei keine Lichtdurchlässigkeit, Weiß lässt das komplette Licht durch. Eine wichtige Rolle bei einer Cookie-Textur spielt der *Wrap Mode*.

- Bei *Spot Lights* wird häufig der *Wrap Mode* auf *Clamp* gestellt, um auf diese Weise nur eine Abbildung des *Cookies* zu erhalten.
- Bei *Directional Lights* wird häufig der *Wrap Mode* auf *Repeat* gestellt, um so das *Cookie*-Motiv zu wiederholen. Ein *Directional Light* bietet hierfür noch den zusätzlichen Parameter *Cookie Size* an, der die Größe eines einzelnen Motivs festlegt. Auf diese Weise können zum Beispiel in einer gesamten Landschaft Lichtunregelmäßigkeiten erzeugt werden, die etwa durch Wolken erzeugt werden.

7.4.2 Light Cookies und Point Lights

Da *Point Lights* in alle Richtungen strahlen, reicht es nicht aus, eine einfache Textur als *Cookie* zu nutzen. Hierfür werden *Cubemaps* genutzt. Cubemaps sind Texturen, die eine Rundumsicht darstellen. Wie der Name schon verrät, können Sie sich das vorstellen wie einen aufgeklappten Würfel, Unity nennt diese Darstellung „6 Faces Layout“. Sie können aber auch andere Formate für eine *Cubemap* nutzen. In den Import-Settings der Textur können Sie dies im *Mapping*-Parameter hinterlegen, wenn Sie den *Texture Type Cubemap* gewählt haben.

Bei einem *Point Light* können Sie sich das nun so vorstellen, dass diese Textur wieder zu einem Würfel zusammengeklappt wird und die Lichtquelle in der Mitte ist. Dadurch wirkt sich der *Light Cookie* nun in alle Richtungen aus.

Zusätzlich unterstützt Unity noch sogenannte Legacy Cubemaps. Dies ist eine Asset-Art, der Sie sechs getrennte Texturen zuweisen. So eine Legacy *Cubemap* erzeugen Sie über **ASSET/CREATE/LEGACY/CUBEMAP**. Auch wenn diese Art von *Cubemaps* vielleicht einfacher zu erstellen ist, so unterstützen diese leider nicht die gleichen grafischen Funktionen wie die oberen (die aber bei *Light Cookies* nicht so relevant sind).

Den *Texture*-Slots werden nun je nach Effektwunsch verschiedene oder gleiche *Cookie*-Texturen zugewiesen. Anschließend wird die *Cubemap* dann dem *Cookie*-Slot der *Light*-Komponente vom *Point Light* zugewiesen, und schon haben wir auch dort den *Light Cookie*-Effekt, allerdings dieses Mal in alle Richtungen.

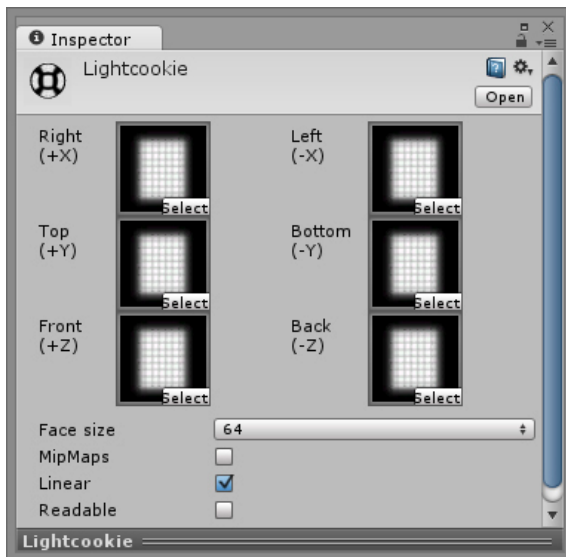


Bild 7.9
Beispiel eines Cubemaps
Light Cookie

Das Bild 7.10 zeigt den Vergleich eines *Point Lights* ohne und mit einem *Light Cookie*.

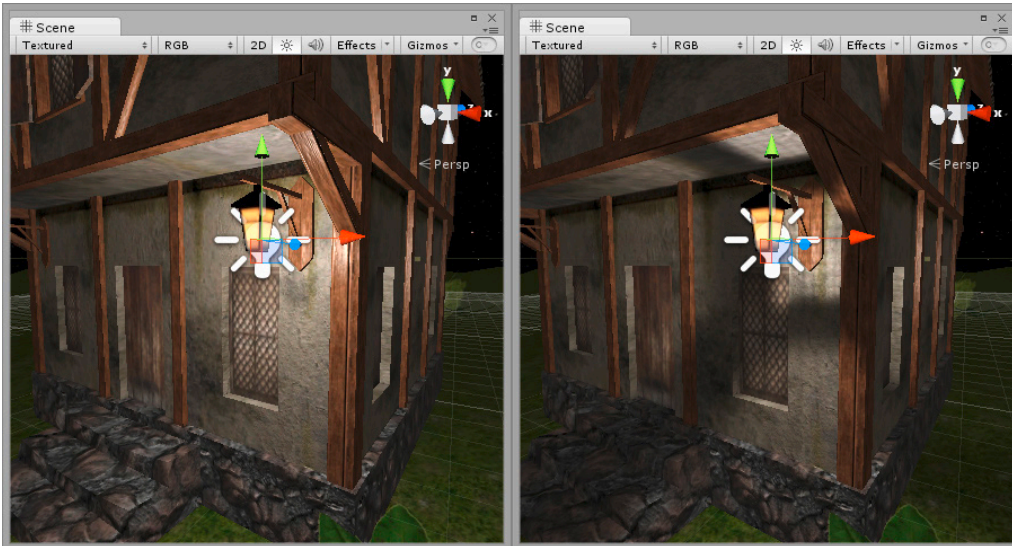


Bild 7.10 Einsatz eines Light Cookies bei einem Point Light

■ 7.5 Light Halos

Ein *Light Halo* ist ein Lichtschleier, der in der realen Welt durch Staubpartikel in der Luft entsteht. Da in Unity natürlich kein Staub vorhanden ist, wird dieser eben durch ein solches *Halo* simuliert. Über die *Light*-Komponenten-Eigenschaft *Draw Halo* können Sie einen Standard-*Halo* aktivieren, der sich an der Lichtstärke (*Intensity*), der Lichtfarbe (*Color*) sowie an der Reichweite (*Range*) der *Light*-Komponente orientiert.

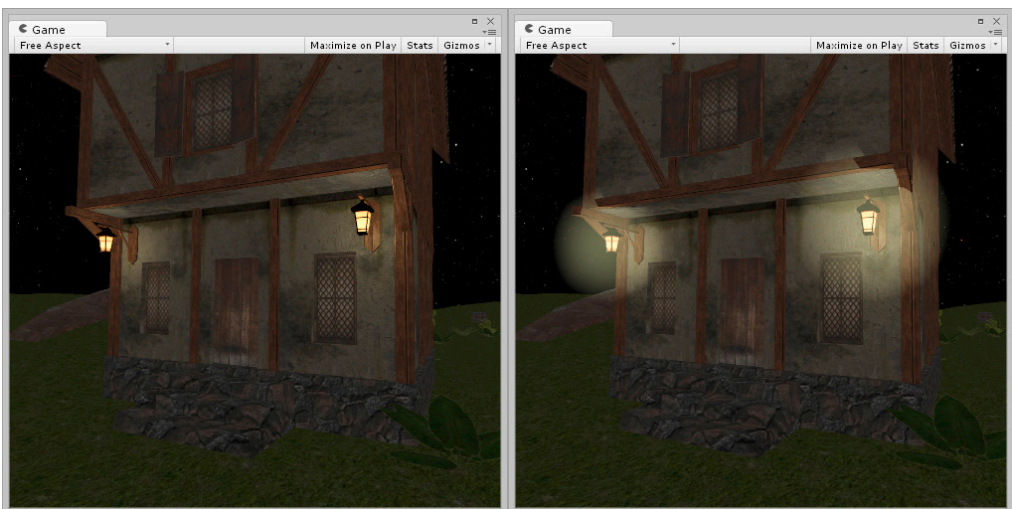


Bild 7.11 Light Halos

Das Bild 7.11 zeigt den Vergleich zweier *Point Lights* mit deaktiviertem und aktiviertem *Draw Halo*-Parameter.

7.5.1 Unabhängige Halos

Sie können neben den Standard-*Halos*, die Sie an den *Light*-Komponenten aktivieren können, auch unabhängige *Halos* erzeugen. Hierfür fügen Sie einem beliebigen *GameObject* über **COMPONENT/EFFECTS/HALO** (oder über **ADD COMPONENT** im *Inspector*) eine *Halo*-Komponente zu. Dies kann auch das gleiche Objekt sein, das bereits eine *Light*-Komponente besitzt. Der Unterschied ist nur, dass sich dieses *Halo* nicht an den Eigenschaften der *Light*-Komponente orientiert, sondern frei konfigurierbar ist.

7.6 Lens Flares

Lens Flares simulieren Linsenreflexionen, also Effekte, die bei Gegenlicht in Kameralinsen entstehen. Das Bild 7.12 zeigt zwei unterschiedliche *Lens Flare*-Effekte, die Unity bereits in den *Standard Assets* „Effects“ bereitstellt. Das linke Motiv zeigt einen kleinen Effekt, der sich lediglich an der Lichtquelle selber zeigt, das rechte Teilbild zeigt einen sehr ausgeprägten Effekt, der auch verschobene Linseneffekte erzeugt.

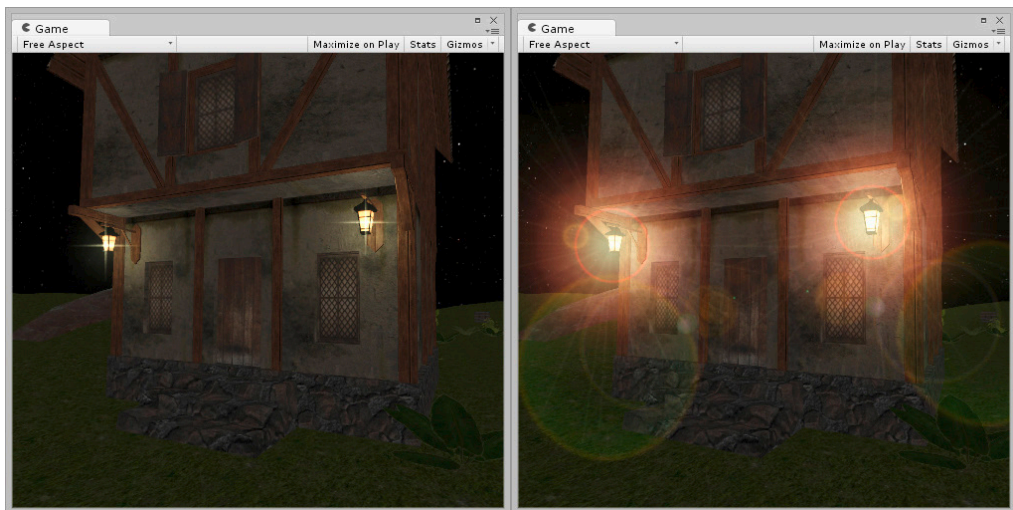


Bild 7.12 Lens Flares „Small Flare“ und „50 mm Zoom“

Zum Nutzen von *Lens Flares* sind mehrere Dinge wichtig. Zunächst benötigen Sie ein *Flare*-Objekt, das den eigentlichen Effekt und dessen Verhalten beschreibt (siehe *Standard Assets*). Da *Flare*-Objekte aber nicht alleine existieren können, müssen Sie diese nun einem *GameObject* zuweisen. Dies können Sie entweder über die *Flare*-Variable einer *Light*-Komponente

machen oder aber über eine separate *Lens Flare*-Komponente, die Sie einem beliebigen *GameObject* zuweisen können. Diese finden Sie über **COMPONENT/EFFECTS/LENS FLARE**. Als Letztes muss die Kamera noch eine *Flare Layer*-Komponente besitzen. Standardmäßig ist dies aber der Fall (siehe Kapitel 6 „Kameras, die Augen des Spielers“).

7.6.1 Eigene Lens Flares

Sie können natürlich nicht nur die mitgelieferten *Lens Flares* nutzen, sondern auch eigene erzeugen. Dies machen Sie über das Menü **ASSETS/CREATE** bzw. über die rechte Maustaste im *Project Browser*.

Dem *Flare-Asset* können Sie einen *Texture-Atlas* zuweisen und anschließend das Verhalten an sich festlegen. Ein *Texture-Atlas* ist eine große Textur, die aus vielen kleinen Bildern besteht. Damit Unity weiß, an welcher Stelle sich ein Unterbild befindet, müssen diese *Flare-Textures* einen bestimmten Aufbau besitzen. Unity bietet hierfür sechs unterschiedliche Layouts an. Über die *Texture Layout*-Eigenschaft teilen Sie schließlich dem *Flare-Objekt* mit, welchen Aufbau Sie auf der Textur nutzen. Details erfahren Sie über den Hilfe-Button oben rechts im *Inspector* des *Flare-Objekts*.

■ 7.7 Projector

Eine weitere Möglichkeit, Licht und Schatten zumindest optisch darzustellen, sind sogenannte Projektoren bzw. *Projectors*. Wie der Name schon vermuten lässt, arbeitet dieser wie ein Beamer (oder ein Tageslichtprojektor oder DIA-Projektor), der ein Bild bzw. Material abstrahlt. Alle Objekte, die diesen Projektierungskegel schneiden, werden dann mit diesem Material überlagert. Hierdurch können sehr interessante Effekte erzielt werden.

7.7.1 Standard Projectors

In den *Standard Assets* gibt es unter anderem einen *Blob Light Projector*, der eine weiße, kreisförmige Textur auf die angestrahlten Objekte projiziert. Dies wirkt wie ein Lichtkegel, nur dass es von der Berechnung her eben kein Licht, sondern nur eine halbtransparente Textur ist. Objekte können also auch keine Schatten werfen.

Als Gegenstück gibt es ebenso auch noch den *Blob Shadow Projector*. Dieser wirft keine helle Textur auf die Objekte, sondern eine schwarze. Platzen Sie diesen *Projector* über einem Objekt und strahlen Sie auf diesen hinab, können Sie damit einen Schatten simulieren.

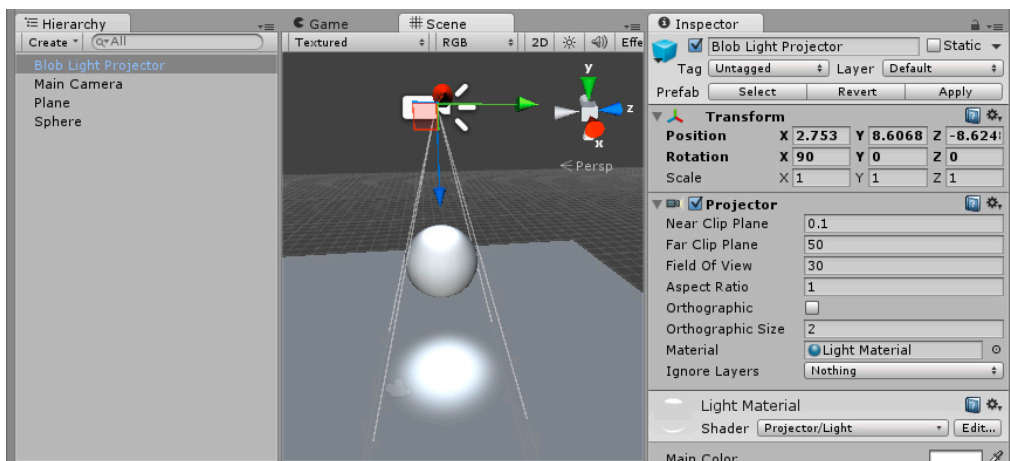


Bild 7.13 Blob Light Projector

Hierfür müssen Sie dem schattenwerfenden Objekt einen Layer zuweisen, den Sie in der *Ignore Layers*-Eigenschaft des *Projectors* hinterlegen. Hierdurch wird die schwarze Textur nicht mehr auf diesem Objekt, sehr wohl aber auf dem Untergrund angezeigt (siehe Bild 7.14). Am besten ist es natürlich, wenn Sie dem *Material*, das dem *Projector* zugewiesen wird, eine Textur zuweisen, die der Form des Objektes entspricht.

Damit der Schatten sich nun auch mit dem Objekt mitbewegt, empfiehlt es sich, den *Projector* als Kind-Objekt dem schattenwerfenden Objekt zuzuweisen – in Bild 7.14 wäre das die Kugel.

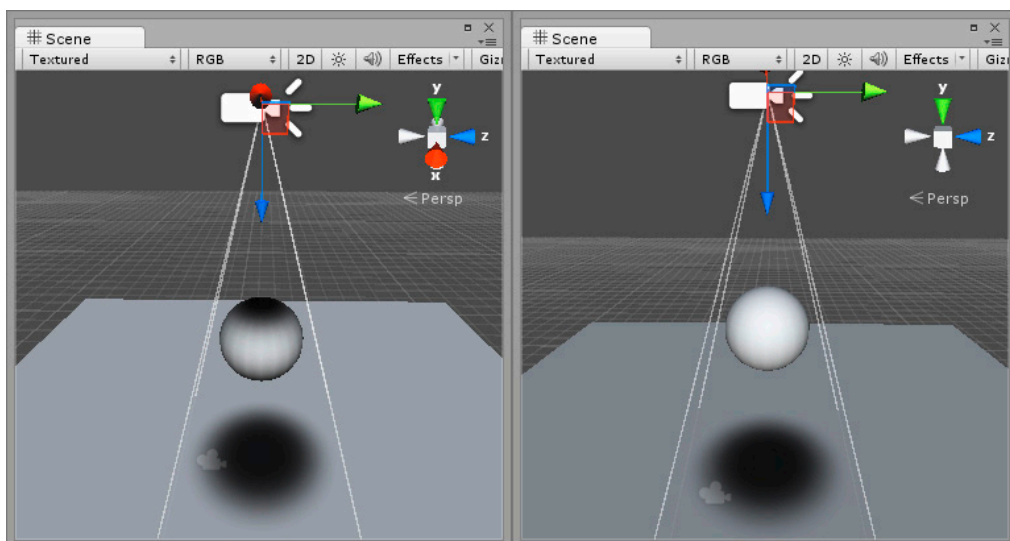


Bild 7.14 Blob Shadow Projector

■ 7.8 Lightmapping

Licht- und Schattenberechnungen sind sehr rechenintensiv. Und umso detaillierter diese dargestellt werden sollen, desto mehr muss gerechnet werden. Um trotzdem in einem Spiel sehr detaillierte Schatten und Lichtszenarien zu erhalten, gibt es das sogenannte *Lightmapping*. Dieses Verfahren berechnet bereits zur Entwicklungszeit die Licht- und Schatteneffekte und erstellt Texturen, die dann über die Modelle gelegt werden. Danach können die Lichtquellen deaktiviert werden, und trotzdem wirken die Objekte, als würden diese angestrahlt werden.

Ein wichtiger Punkt beim herkömmlichen *Lightmapping* ist der, dass nur Objekte berücksichtigt werden, die sich nicht bewegen. Das betrifft sowohl die Lichtobjekte als auch die beleuchteten Objekte. Der Grund hierfür ist ganz einfach: Stellen Sie sich vor, Sie berechnen den Schatten eines Autos und „brennen“ dessen Schatten in die Textur der Straße ein. Nun fährt das Auto weg und die Straßentextur mit dem Schatten bleibt an der gleichen Stelle. Dies ist natürlich nicht gerade das, was man als realistisch bezeichnen würde. Deshalb berücksichtigt Unity beim normalen *Lightmapping* nur Objekte, die statisch sind, also Objekte, die sich nicht bewegen, und Lichtquellen, bei denen der Baking-Modus auf *Mixed* oder *Baken* gestellt ist. Allerdings bietet Unity auch eine Möglichkeit, bewegliche Objekte vom Lightmapping profitieren zu lassen. Hierbei werden sogenannte *Light Probes* eingesetzt, die ich aber noch im folgenden Abschnitt „Light Probes“ behandeln werde.

Zum Markieren statischer Objekte besitzen alle *GameObjects* in einer Szene eine kleine Checkbox oben rechts im *Inspector* mit dem Namen *Static*. Setzen Sie diesen Haken bei allen Objekten, die sich nicht bewegen und beim *Lightmapping* berücksichtigt werden sollen.

Da sich mittlerweile mehrere Funktionen dieser *Static*-Eigenschaft bedienen, können Sie über den zusätzlichen Pfeil an der rechten Seite der *Static*-Checkbox ein weiteres Menü aufklappen. Hier können Sie definieren, für welche Funktionen diese *Static* Eigenschaft gilt. Achten Sie darauf, dass hier *Lightmap Static* aktiviert ist.

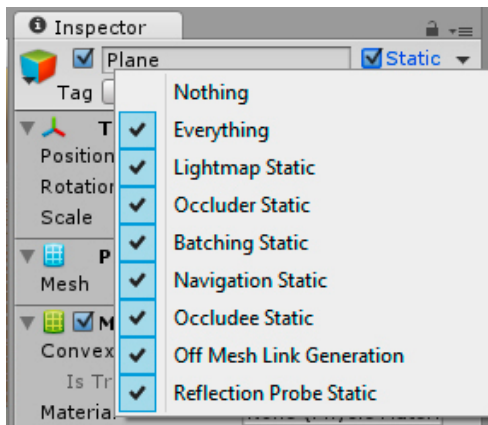


Bild 7.15
„Lightmap Static“-Option im Static-Menü

Beachten Sie, dass in Unity per Default *Continuous Baking* aktiviert ist. Das bedeutet, dass Unity automatisch die *Lightmaps* neu berechnet, sobald in einer Szene eine Aktion gemacht wurde, die berechnete *Lightmaps* beeinflussen könnten.

Um dies zu ändern, öffnen Sie das *Lighting-Fenster* (Window/Lighting) und entfernen Sie den Haken bei dieser Eigenschaft. Diese finden Sie ganz unten im Reiter „Scene“. Haben Sie *Continuous Baking* (ab Unity 5.1 wird voraussichtlich die Bezeichnung „Auto“ angezeigt) deaktiviert, wird das *Baken* nur noch manuell über den daneben befindlichen Knopf **BUILD** gestartet. Diese Einstellung sollten Sie auf jeden Fall dem Automatismus vorziehen, da das ständige Erstellen von *Lightmaps* den Arbeitsfluss doch erheblich ausbremst. Und umso größer das Projekt wird, desto länger dauert das Erstellen der *Lightmaps*.

Achten Sie darauf, dass beim *Baken* nur *Area Lights*, Lichtquellen mit dem *Baking-Modus Mixed* oder *Baked*, sowie *Materials* mit *Emission*-Werten bzw. Texturen, deren *Global Illumination*-Eigenschaften auf „Baked“ stehen (siehe „Standard-Shader“ im Kapitel 5 „Objekte in der zweiten und dritten Dimension“), berücksichtigt werden.



Emissionswerte und Texturen

Durch eine Emission-Textur (oder einen einfachen Wert) verleiht der Standard-Shader einem Material das Aussehen, als würde es leuchten. Für gewöhnlich werden diese *Shader* deshalb auch für Lichtquellen-Meshes wie Lampen, Laternen etc. genutzt. Steht nun zusätzlich der *Global Illumination*-Parameter des Materials auf „Baked“, werden beim Lightmapping-Verfahren die Objekte mit diesen Materialien ebenfalls als Lichtquellen berücksichtigt und in die *Lightmaps* „gebrannt“.

Da das Erstellen der *Lightmaps* je nach Einstellung und *Szenenaufbau* durchaus einige Zeit dauern kann, wird der Fortschritt des Vorgangs unten rechts als Balken angezeigt. Nach dem *Baken* werden die fertigen *Lightmaps* schließlich im *Lightmaps*-Bereich des *Lighting*-Fensters angezeigt und in der Szene über die statischen Objekte gelegt.

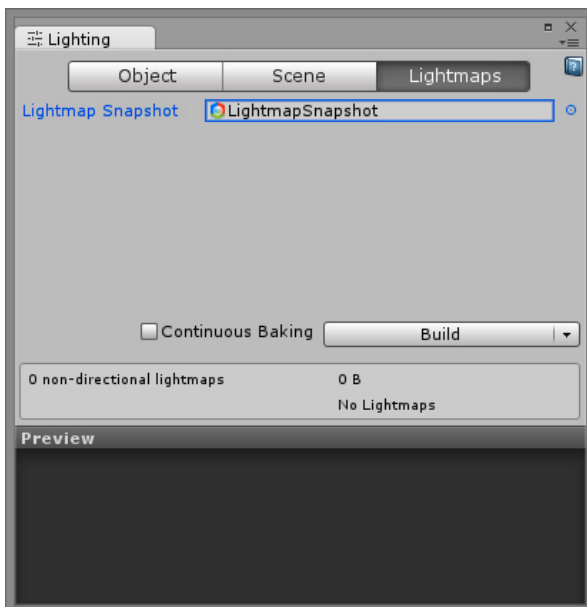


Bild 7.16
Lighting-Fenster mit
Lightmaps-Bereich

Mithilfe der *Lightmap Snapshot*-Eigenschaft (siehe Bild 7.16) können Sie auch zwischen verschiedenen *Lightmaps* wechseln oder auch gar kein *Lightmapping* zuweisen („None“). Auf diese Weise können Sie zu Testzwecken schnell zwischen verschiedenen *Lightmapping*-Szenarien oder auch die Szene ohne *Lightmapping* betrachten – vorausgesetzt natürlich, Sie haben *Continuous Baking* deaktiviert.

Beachten Sie zudem auf dem *Scene*-Reiter des *Lighting*-Fensters die Funktion *Baked GI*. Sie berechnet die *Global Illumination* (siehe Abschnitt „Global Illumination“), also die indirekte Beleuchtung beim *Lightmapping*. Ist diese deaktiviert, können Sie auch keine *Lightmaps* erstellen.



Lightmapping-Beispiel

Im Online-Bereich dieses Buches finden Sie eine Video-Demonstration des *Lightmapping*-Verfahrens, in der die wichtigsten Parameter vorgestellt werden. Weiter werden auch *Emissive Materials* und das *Area Light* gezeigt.

7.8.1 Light Probes

Lightmapping bietet Ihnen in Sachen Performance, aber auch in Sachen der Detailauflösung große Vorteile. Ein großer Nachteil des herkömmlichen *Lightmappings* ist hierbei, dass es nur statische Objekte berücksichtigen kann.

Mithilfe von *Light Probes* können nun auch bewegliche Objekte vom *Lightmapping* profitieren. Beachten Sie dabei, dass bei jedem beweglichen Objekt, welches von den *Light Probes* beeinflusst werden soll, die *Use Light Probes*-Eigenschaft vom *MeshRenderer* aktiviert sein muss.

Hinter den *Light Probes* steckt der Gedanke, dass die Beleuchtung im Vorwege an strategischen Stellen gespeichert wird. Befindet sich ein Objekt nun zwischen verschiedenen Messstellen, dann wird die Beleuchtung näherungsweise berechnet und dem Objekt zugewiesen. *Light Probes* sind nun genau diese Messstellen und werden, sobald sie in der *Hierarchy* selektiert werden, in der *Scene View* gelb dargestellt (siehe Bild 7.17).



Auch wenn Sie mit *Light Probes* vom *Lightmapping* profitieren, sollten Sie trotzdem statische Objekte auch immer als solche definieren. Denn Lichtberechnungen des normalen *Lightmappings* sehen immer besser aus als über *Light Probes* interpolierte Lichtwerte. Zudem sollten Sie bedenken, dass durch *Light Probes* selber keine Schatten entstehen, durch *Lightmapping* aber schon.

Das Bild 7.17 zeigt eine kleine Szene mit einem statischen Cube, der ein Material mit einer grünen *Emission*-Farbe besitzt (siehe „Der Standard-Shader“ im Kapitel 5 „Objekte in der zweiten und dritten Dimension“). Hinzu kommen eine statische Ebene (*Lightmap Static* ist hier aktiv) sowie ein bewegliches Capsule-Objekt, bei dem der *Use Light Probes*-Parameter des *MeshRenderers* aktiviert ist. Dank der *Light Probes* (gelb dargestellt) wird nun auch die Kapsel beleuchtet.

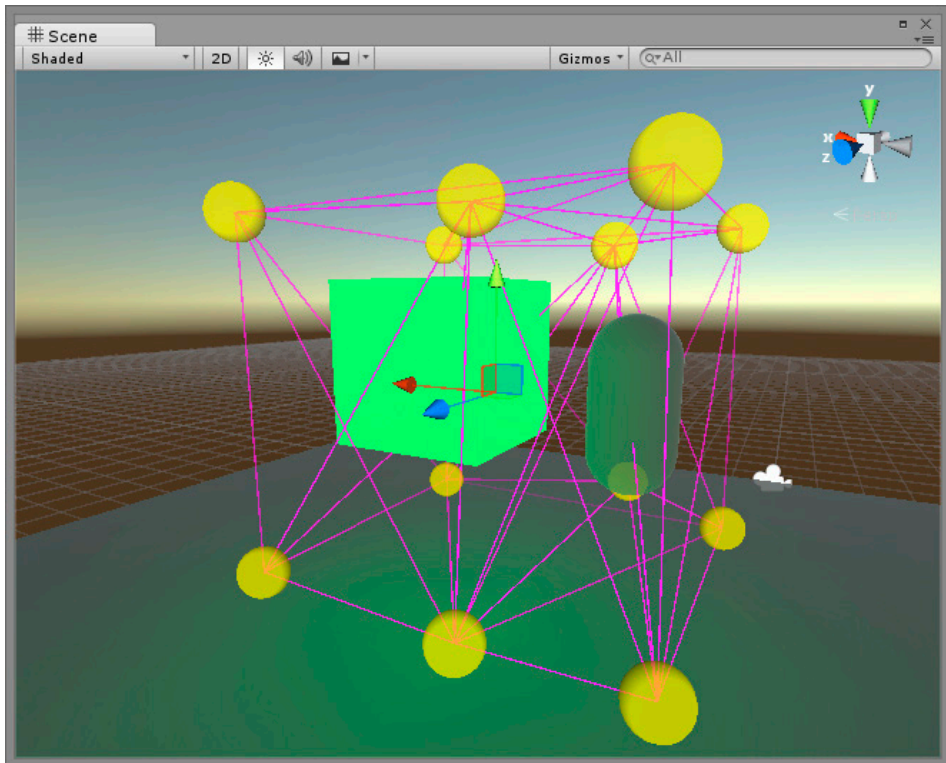


Bild 7.17 Im 3D-Raster angeordnete Light Probes

Light Probes fügen Sie Ihrer Szene über **GAMEOBJECT/LIGHT/LIGHT PROBE GROUP** zu. Wenn Sie dann die *Light Probe Group* in der *Hierarchy* selektieren, können Sie jedes einzelne *Light Probe* separat noch in Ihrer Szene verschieben. Zudem können Sie über ein kleines Menü im *Inspector* zusätzliche *Light Probes* zu der Gruppe hinzufügen als auch existierende löschen.

Am Anfang ist es sinnvoll, *Light Probes* zu positionieren, dass sie wie in Bild 7.17 in einem gleichmäßigem 3D-Gatter angeordnet werden. Später sollten Sie aber darauf achten, die Anzahl soweit es geht zu reduzieren und die Positionen entsprechend zu optimieren, da jedes einzelne *Light Probe* einiges an Speicher kostet. So ist es z. B. empfehlenswert, größere Abstände zwischen den *Light Probes* zu halten, bei denen es keine großen Unterschiede in der Beleuchtung gibt. Sind die Unterschiede sehr stark, sollten sie wiederum näher positioniert werden.

Zum Testen der *Light Probe*-Positionen können Sie ein beliebiges nichtstatisches Objekt in Ihrer Szene selektieren und dieses verschieben. Ihnen wird dabei nicht nur die spätere Beleuchtung angezeigt, es wird Ihnen auch grafisch dargestellt, welcher Bereich Ihrer *Light Probe Group* aktuell zum Berechnen der Ausleuchtung herangezogen wird (siehe Bild 7.18). Dabei können Sie zum einen an den *Light Probes* erkennen, welche Lichtinformationen diese besitzen (also von wo welches Licht auf diese trifft), zum anderen zeigt der gelb markierte Bereich die *Light Probes*, die die Berechnung beeinflussen. Beachten Sie hierbei, dass die *Light Probes* selber nur dann gelb dargestellt werden, wenn Sie die *Light Probe Group* auch in der *Hierarchy* selektieren.

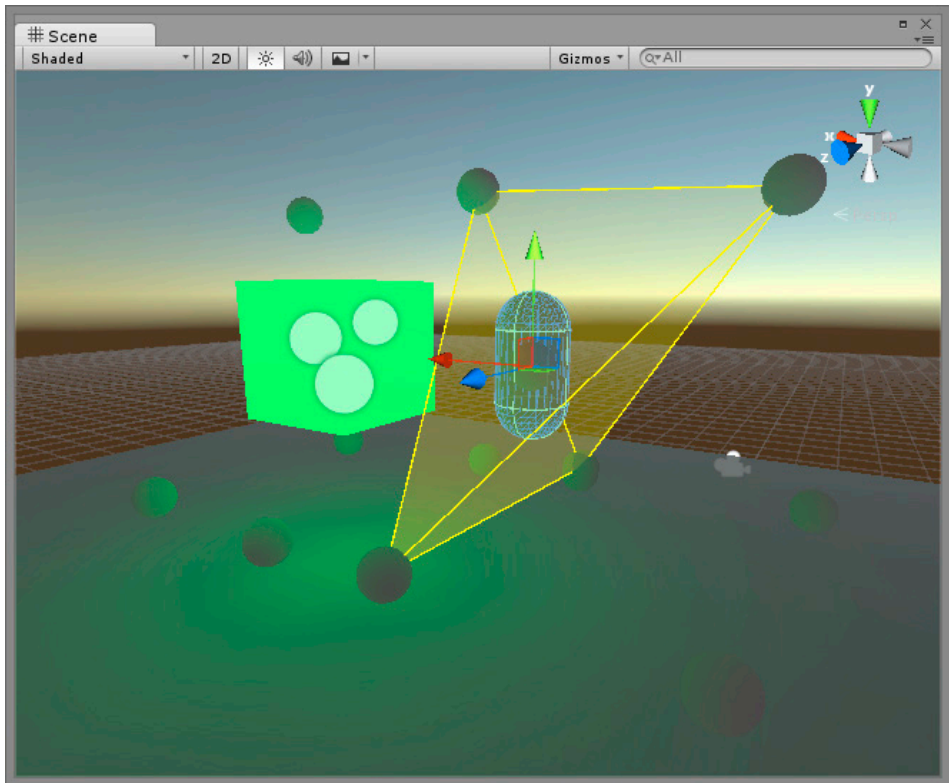


Bild 7.18 Mit Light Probes ausgeleuchtetes Objekt

Umso mehr sich die bewegliche Kapsel in Bild 7.18 einem der Light Probes annähert, desto stärker ist dessen Einfluss auf die Ausleuchtung des Objektes. Das bedeutet, dass ein im Zentrum des gelb markierten Bereiches befindliches Objekt von allen vier Light Probes gleichermaßen beeinflusst wird.

■ 7.9 Rendering Paths

Um Objekte mit Licht und Schatten ansehnlich darzustellen, ist die Wahl des richtigen Rendering-Verfahrens ein wichtiger Punkt. Das Rendern berechnet hierbei das Bild, das am Ende auf dem Bildschirm des Spielers dargestellt wird.

Unity unterstützt hier gleich drei unterschiedliche *Rendering*-Techniken, die ich im Folgenden noch weiter vorstellen werde. Sie können für jede Plattform individuell eine Default-Rendering-Technik definieren. Dies machen Sie in den *Player Settings* (**PROJECT SETTINGS / PLAYER**) im Bereich *Other Settings*. Zusätzlich können Sie noch einmal bei jeder Kamera ein zu nutzendes *Rendering*-Verfahren hinterlegen. Hierfür nutzen Sie die Eigenschaft *Rende-*

ring Path. Wird hier die Default-Einstellung „Use Player Settings“ belassen, wird die Einstellung aus den *Player Settings* übernommen.

Ein Hauptunterschied dieser Rendering-Verfahren sind die eingesetzten Methoden zum Berechnen der Beleuchtung.

- Beim **Vertex Lighting** wird die Auswirkung einer Lichtquelle lediglich anhand der *Vertices* der beleuchteten 3D-Modelle bzw. der *Meshes* berechnet und auf die gesamten Flächen hochgerechnet.
- Beim **Pixel Lighting** wird die Beleuchtung für jeden einzelnen Pixel separat berechnet. Diese Vorgehensweise ist natürlich ressourcenhungriger, ermöglicht aber z.B. Normal-Mapping oder auch Echtzeitschatten. Allerdings sollten Sie beachten, dass einige ältere Grafikkarten *Pixel Lighting* nicht unterstützen.

7.9.1 Forward Rendering

Forward Rendering ist ein Misch-Rendering-Verfahren, das verschiedene Berechnungsarten der Lichtquellen nutzt. In den *Quality Settings* (**EDIT/PROJECT SETTINGS/QUALITY**) können Sie für jede Plattform über den Parameter *Pixel Light Count* bestimmen, wie viele Lichtquellen im *Pixel Rendering*-Verfahren berechnet werden dürfen. Die anderen Lichtquellen werden mit dem *Vertex Lit*-Verfahren oder als *Spherical Harmonics* berechnet. Letzteres ist ein Verfahren, das ebenfalls auf Basis von *Vertices* arbeitet und aufgrund von Näherungen sehr schnell berechnet werden kann.

Unity wählt bei diesem Verfahren abhängig vom *Pixel Light Count*-Wert selbstständig die wichtigsten Lichtquellen aus und rendert diese dann entsprechend im *Pixel-Lighting-Modus*. Dabei zählt das hellste *Directional Light* automatisch zu den wichtigen.

Sie können aber auch die Auswahlmöglichkeiten selber bestimmen. Stellen Sie den *Render Mode* einer Lichtquelle auf *Important*, so wird diese per Pixel berechnet. Stellen Sie diese auf *Not Important*, wird sie auf jeden Fall per Vertex oder als *Spherical Harmonics* berechnet. Nur bei *Auto* wählt Unity selbstständig.

Beim *Forward Rendering* werden neben der hinterlegten Anzahl an Per-Pixel-Lichtern noch bis zu vier Lichter nach dem *Vertex Lighting*-Verfahren berechnet. Der Rest wird schließlich als *Spherical Harmonics* kalkuliert.

In Bild 7.19 sehen Sie eine kleine Szene, die mit dem *Forward Rendering*-Verfahren dargestellt wird. Sie besitzt zwei Lichtquellen, wobei in den *Quality Settings* als *Pixel Light Count* ein Wert von 1 hinterlegt wurde. Hierdurch wird nur bei einer Lichtquelle das *Pixel Rendering*-Verfahren eingesetzt, weshalb auch nur ein Schatten zu sehen ist.

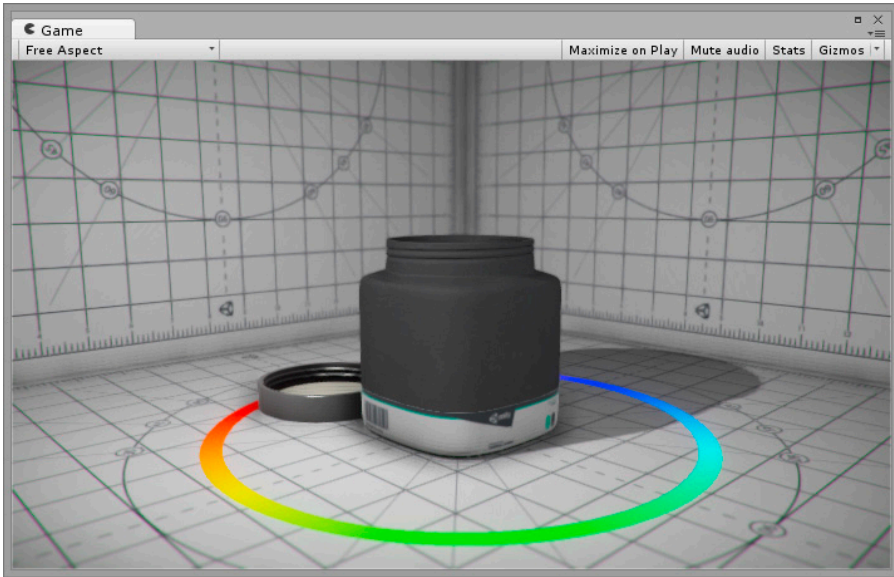


Bild 7.19 Mit Forward Rendering dargestellte Szene

7.9.2 Vertex Lit

Vertex Lit nutzt ausschließlich das *Vertex Lighting*. Es unterstützt keine Echtzeitschatten und auch keine *Shader*-basierten Effekte wie *Normalmaps* oder Echtzeitschatten. Dafür ist es aber mit Abstand am performantesten und bietet die umfassendste Hardwareunterstützung. Allerdings wird es nicht von Konsolen unterstützt.

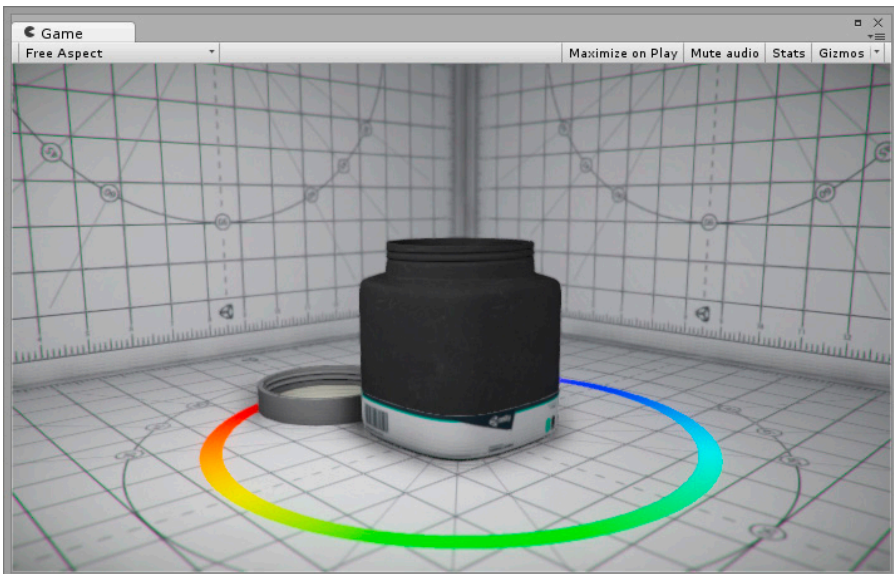


Bild 7.20 Gerenderte Szene mit Vertex Lit

Das Bild 7.20 zeigt eine Szene mit zwei Lichtquellen und ein Material, das zur Darstellung einer rauen Oberfläche eine *Normalmap* nutzt. Aufgrund des *Vertex Lit*-Verfahrens werden sowohl die Schatten der Lichtquellen als auch der Effekt der *Normalmap* nicht dargestellt.

7.9.3 Deferred Lighting

Deferred Lighting ist die anspruchsvollste Rendering-Art mit den detailreichsten Darstellungen von Schatten und Licht, da alle Lichtquellen nach dem Pixel Lighting-Verfahren berechnet werden. Im Gegensatz zum *Forward Rendering* unterstützt es deshalb auch beliebig viele Lichtquellen mit Echtzeitschatten, was sich aber natürlich auch im Performance-Bedarf bemerkbar macht. Außerdem wird dieses Verfahren nicht von jedem Gerät unterstützt, weshalb gerade im Mobile-Bereich die Verwendung vorher genauer geprüft werden sollte.

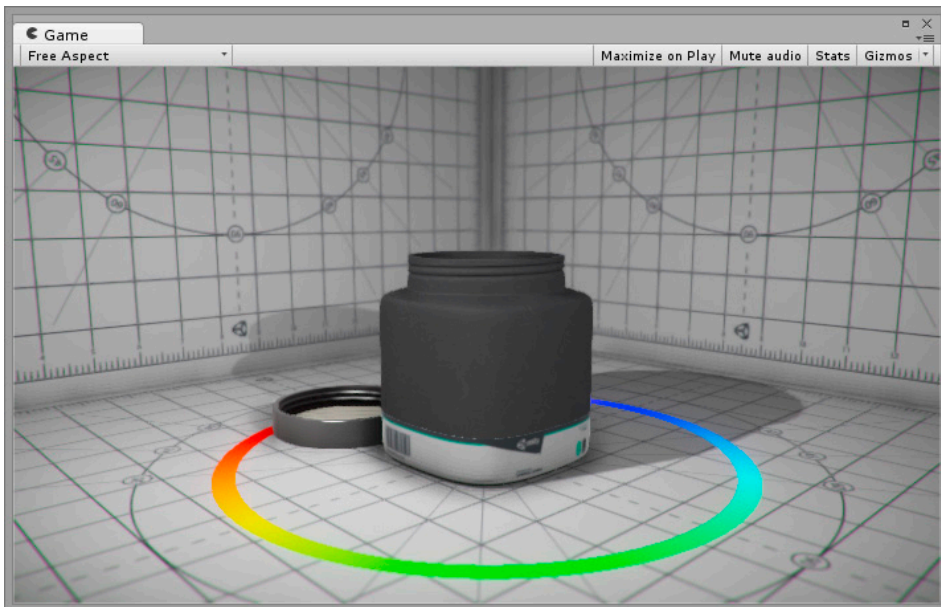


Bild 7.21 Gerenderte Szene mit Deferred Lighting

In Bild 7.21 sehen Sie eine Szene mit zwei Lichtquellen. Aufgrund des gewählten *Deferred Lighting*-Verfahrens wird bei beiden Lichtquellen das *Pixel Rendering*-Verfahren genutzt, sodass auch zwei Schatten zu sehen sind. *Normalmaps* und andere *Shader*-Effekte werden ebenfalls dargestellt.

Neben den höheren Performancekosten hat das *Deferred Lighting* noch einen weiteren Nachteil, den Sie aber ausgleichen können. Im Gegensatz zum *Forward Rendering* wird bei diesem Verfahren kein hardwareseitiges *Antialiasing*, also keine Kantenglättung, unterstützt. Stattdessen müssen Sie hier der Kamera den „Antialiasing“-*Image Effect* zufügen, den Sie in den Standard Assets „Effects“ finden.

■ 7.10 Global Illumination

In der realen Welt erreichen Lichtstrahlen nicht nur die Stellen, wo sie auf direktem Wege hinkommen. Sie werden auch von den Flächen reflektiert, auf die sie stoßen, und gelangen so über Umwege auch an verstecktere Stellen. Ansonsten würde es z. B. an einem Sommertag im Schatten komplett schwarz sein, was es aber nun mal nicht ist.

In Unity nennt sich dieses Verfahren zur indirekten Beleuchtung *Global Illumination*. Wie intensiv die indirekte Beleuchtung dabei ist, können Sie bei jeder Lichtquelle separat einstellen. Dort regeln Sie über den *Bounce Intensity*-Parameter, wie stark das Licht von einer Fläche abgestrahlt wird und die Umgebung erhellt. Das Bild 7.22 zeigt Ihnen, wie durch die reflektierende (indirekte) Beleuchtung eines Spotlights auch der Dachkasten und die Seitenbalken eines Holzhauses angeleuchtet werden.

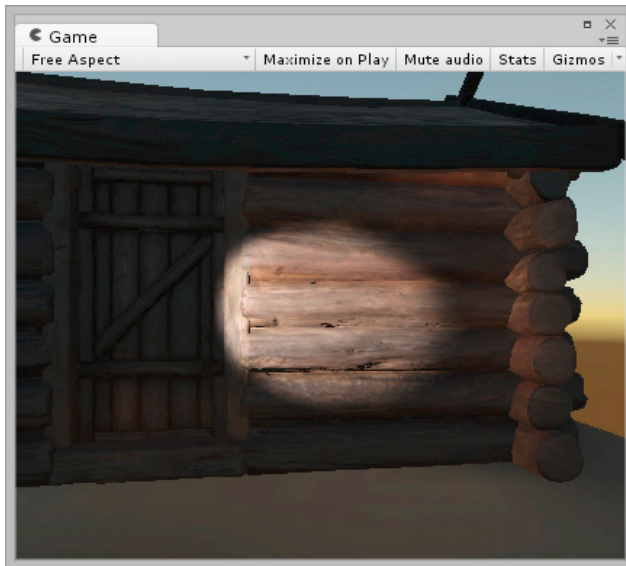


Bild 7.22
Indirekte Beleuchtung
durch ein Spotlight

Da die Berechnung indirekter Beleuchtung sehr rechenintensiv sein kann, gibt es in Unity zwei unterschiedliche Verfahren, die *Global Illumination* in Spielen ermöglichen: *Baked GI* und *Precomputed Realtime GI*.

7.10.1 Baked GI

Die erste *Global Illumination*-Variante wird beim *Lightmapping* genutzt und wird auch *Baked GI* genannt. Hier wird die Ausbreitung der indirekten Beleuchtung von entsprechend markierten Lichtquellen zur Entwicklungszeit berechnet und über die Objekte gelegt. Die *Baking*-Eigenschaft der Lichtquellen muss deshalb auf *Mixed* oder *Baken* gestellt sein. Allerdings werden bei diesem Verfahren nur statische Objekte berücksichtigt, die mit *Static* oder *Lightmap Static* gekennzeichnet wurden.

Im *Lighting-Fenster (WINDOW/LIGHTING)* gibt es für *Baked GI* einen extra Bereich, in dem Sie für dieses Verfahren einige Grundeinstellungen vornehmen und auch die komplette Funktion deaktivieren können. Dort finden Sie auch die *Ambient Occlusion*-Eigenschaft, mit der Sie den Lichteinfluss in verdeckten Stellen wie z.B. Innenecken einschränken und kleine Schatten erzeugen können. Beachten Sie, dass Sie kein *Lightmapping* machen können, wenn Sie *Baked GI* deaktiviert haben.

7.10.2 Precomputed Realtime GI

Das zweite *Global Illumination*-Verfahren ist das sogenannte *Precomputed Realtime GI*, also vorberechnetes Echtzeit-GI. Auch hier werden nur statische Objekte berücksichtigt. Allerdings wird hier dieses Mal alles zur Laufzeit berechnet, sodass Sie jede Lichtquelle mit den *Baking-Modus Realtime* auch während des Spiels verschieben oder anderweitig ändern können, und trotzdem funktioniert die indirekte Beleuchtung. Allerdings ist diese Variante nicht ganz so detailliert wie *Baked GI* und natürlich auch nicht ganz so performance-freundlich. Beachten Sie deshalb, dass bei sehr großen Beleuchtungsänderungen die Berechnungen auch mal über mehrere Frames dauern können.

Diese Funktion ist zu Anfang eines Projektes genauso aktiviert wie *Baked GI*. Dies macht auch Sinn, weil auf diese Weise sowohl Realtime- als auch in *Lightmaps* integrierte Lichtquellen beim GI berücksichtigt werden. Möchten Sie *Precomputed Realtime GI* allerdings deaktivieren, können Sie dies im *Lighting*-Fenster machen, wo Sie auch noch einige weitere Einstellmöglichkeiten zu diesem Verfahren finden.

■ 7.11 Reflexionen (Spiegelungen)

Möchten Sie in Unity Reflexionen bzw. Spiegelungen darstellen, gibt es hierfür verschiedene Verfahren.

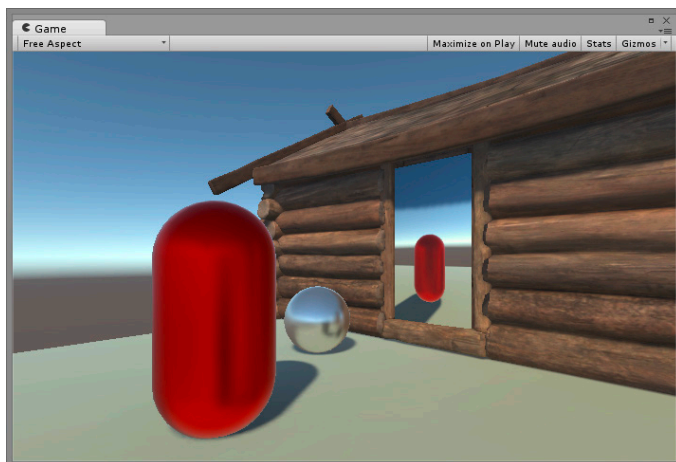


Bild 7.23

Szene mit einem Spiegel und anderen spiegelnden Objekten

Wenn ein korrektes Spiegelungsverhalten wichtig ist, wie z. B. bei einem Spiegel oder einer Wasseroberfläche, werden hierzu *Render Textures* eingesetzt. In dem Fall wird eine Kamera vor dem spiegelnden Objekt in die entgegengesetzte Richtung platziert und stellt dann auf der *Render Texture* das Kamerabild dar. Tipps zum Nutzen von *Render Textures* für Spiegel erhalten Sie im Abschnitt „Render Texture“ von Kapitel 6 „Kameras, die Augen des Spielers“.

Bei Objekten, wo kein 100% korrektes Bild notwendig ist, wie z. B. bei einem Metallbecher oder den Radkappen eines Autos, ist dieses Verfahren natürlich zu aufwendig und auch zu rechenintensiv. Deshalb werden bei solchen Anwendungszwecken einfach dreidimensionale Bilder genutzt, die statt einer echten Reflexion auf diesen Objekten als Spiegelung dargestellt werden.

Per Default wird dabei die prozedural erzeugte Skybox der aktuellen Szene auf den Objekten dargestellt. Sie können aber auch im *Lighting*-Fenster ein individuelles Bild in Form einer *Cubemap* festlegen, das stattdessen standardmäßig genutzt wird.

Dies können Sie über den *Reflection Source*-Parameter im „Environment Lighting“-Bereich des *Lighting*-Fensters einstellen. Dort können Sie auch weitere Grundeinstellungen für Reflexionen vornehmen.

Damit Unity die dort hinterlegte *Reflection Source* automatisch bei reflektierenden Materialien nutzt, um die Spiegelungen/Reflexionen darzustellen, brauchen Sie nichts weiter zu machen. Das Einzige, was Sie tun müssen, ist, beim *Shader* eines Materials zu definieren, wie das Reflexionsverhalten sein soll, also wie glatt eine Oberfläche ist und wie metallisch sie wirken soll (siehe „Der Standard-Shader“ im Kapitel 5 „Objekte in der zweiten und dritten Dimension“).

Neben dieser ganz grundsätzlichen Reflexionsvorlage können Sie aber auch noch individuelle Reflexions-*Cubemaps* auf den Objekten darstellen. Hierbei kommen sogenannte *Reflection Probes* zum Einsatz, die ich im Folgenden etwas genauer erläutere.

7.11.1 Reflection Probes

Mit *Reflection Probes* können Sie raumabhängige Reflexions-*Cubemaps* erzeugen, die dann automatisch den Materialien zugewiesen werden, die sich in deren Umgebung befinden. Die auf diese Weise generierten *Cubemaps* können sowohl statisch sein als auch in Echtzeit ihre Inhalte aktualisieren, sodass sich die Reflexionen auf den Materialien sogar während des Spiels ändern können.

Reflection Probes sind kugelförmige Gebilde (siehe Bild 7.24), die im Grunde eine 360°-Kamera darstellen, die ihre komplette Umgebung in Bildform festhält. Dabei können Sie über den *Type*-Parameter definieren, zu welchem Zeitpunkt und wie oft dieses Bild generiert wird.

- **Baken** legt fest, dass das Bild zur Entwicklungszeit während des *Lightmapping-Bakens* erstellt wird. Hierbei werden nur *statische* Objekte im Bild erfasst, bei denen *Static* oder *Reflection Probe Static* aktiviert ist.
- **Realtime** bedeutet, dass das Erstellen des Bildes zur Laufzeit des Spiels stattfindet.
- **Custom** ermöglicht, eine eigene *Cubemap* diesem *Reflection Probe* zuzuweisen, die dann in dem Einflussbereich dieses *Reflection Probes* als Reflexion dargestellt wird. Alternativ

können Sie über einen **BAKE**-Button eine *Cubemap* erstellen. Über die zusätzliche *Dynamic Objects*-Option können Sie zudem auch die nichtstatischen Objekte mit in die *Cubemap* einfließen lassen.

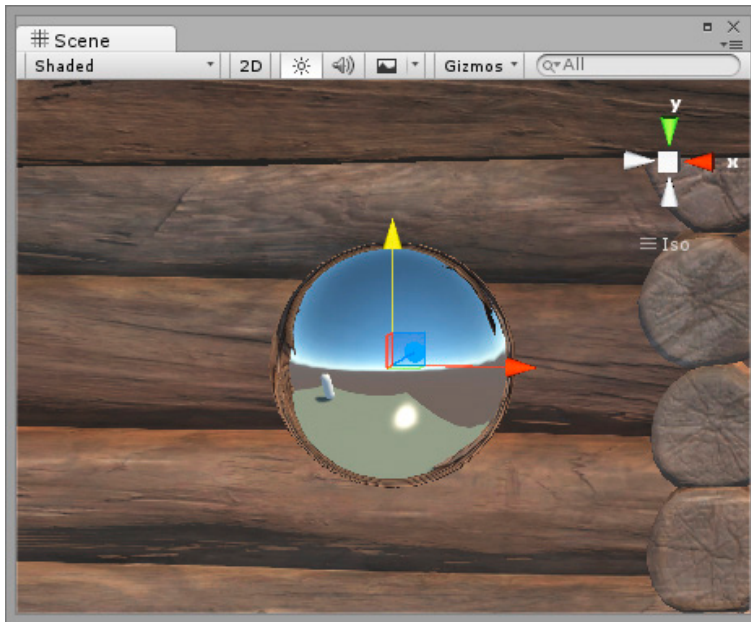


Bild 7.24 Light Probe

Wählen Sie die *Type*-Option *Realtime*, stehen Ihnen folgende Einstellmöglichkeiten zur Wahl:

- **On Awake** erstellt das Bild einmal, wenn der *Reflection Probe* erstmalig in einer Szene aktiv wird.
- **Every Frame** bewirkt, dass der Probe wie bei einer Kamera in jedem Frame das Bild erneut erstellt.
- **Via Scripting** bedeutet, dass das Bild nur dann erstellt wird, wenn per Code die Methode *RenderProbe* eines *Reflection Probes* ausgelöst wird.

Das Listing 7.1 zeigt Ihnen ein Beispiel, wie Sie bei der *Type*-Option *Via Scripting* das Erstellen eines neuen *Reflection Probe*-Bildes auslösen.

Listing 7.1 Per Code Reflection Probe-Bild erstellen

```
using UnityEngine;
using System.Collections;
public class RefreshReflectionProbe : MonoBehaviour {
    public ReflectionProbe probe;

    void OnTriggerEnter()
    {
        probe.RenderProbe();
    }
}
```

Um das Skript aus Listing 7.1 zu nutzen, weisen Sie dieses einem *Trigger-Collider* zu. Nun brauchen Sie nur noch den *Reflection Probe* auf die Variable probe zu ziehen und schon wird das Bild der *Reflection Probes* aktualisiert, sobald sich ein Objekt mit einem Collider (und einem Rigidbody) in diesen hineinbewegt. Das Bild 7.25 stellt es noch einmal bildlich dar.

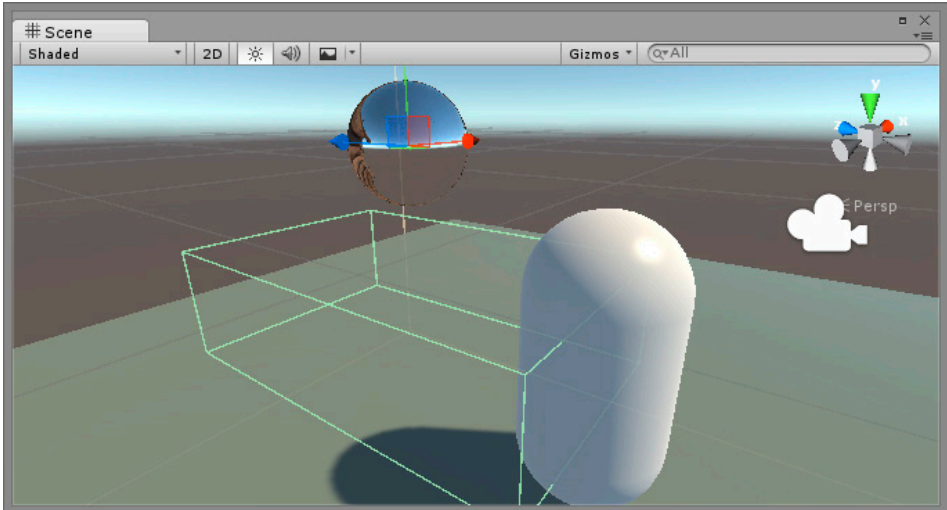


Bild 7.25 Per Trigger-Collider einen Reflection Probe aktualisieren

Wie eingangs erwähnt, fangen *Reflection Probes* die eigene Umgebung ein, um diese dann als Reflexionen auf anderen Materialien anzuwenden. Welche Umgebung hierbei eingefangen wird, wird durch eine gelbe Box gekennzeichnet.

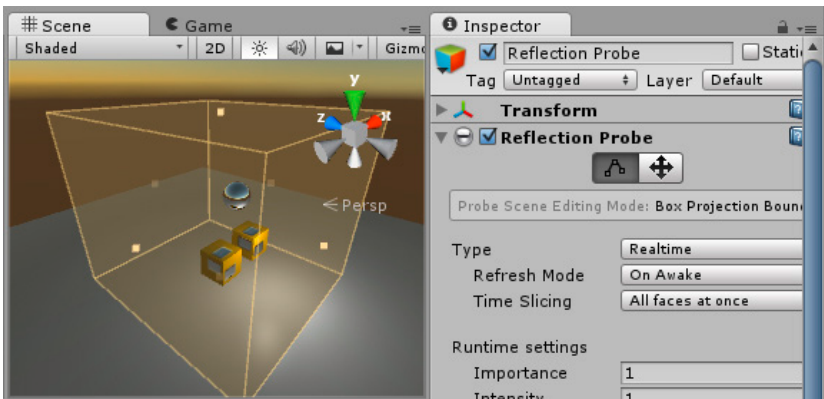


Bild 7.26 Reflection Probe-Box mit aktiviertem Size-Werkzeug

Mit dem *Size-Tool* können Sie nun diese Box modifizieren. Nutzen Sie hierfür die *Handle-Punkte* an den Flächen der Box (siehe Bild 7.26). Zum Aktivieren des *Size-Tools* drücken Sie den linken Funktions-Button im *Inspector* der *Reflection Probe*-Komponente. Neben dem *Size-Tool* finden Sie rechts daneben das *Origin-Tool* (gekennzeichnet durch das Pfeile-Kreuz). Mit diesem können Sie den *Reflection Probe* innerhalb der Box verschieben.

■ 7.12 Qualitätseinstellungen

Wie Sie bereits sicher gemerkt haben, bietet Unity Ihnen viele unterschiedliche Einstellungen bezüglich der Hochwertigkeit der grafischen Darstellung.

Die Grundeinstellungen hierfür können Sie in den *Quality Settings* vornehmen, die Sie über **EDIT/PROJECT SETTINGS/QUALITY** erreichen.

7.12.1 Quality Settings

In den *Quality Settings* haben Sie die Möglichkeit, unterschiedliche Qualitätsstufen zu definieren, die dann je nach Stufe performance-lastiger und hochwertiger bzw. sparsamer und rudimentärer aussehen.

Selektieren Sie hierfür in der dort zu findenden Matrix ein *Quality Level* und nehmen im unteren Bereich dann die Grundeinstellungen für diese Stufe in den Bereichen *Rendering*, *Shadow* und *Other* vor. Über den Button **ADD QUALITY LEVEL** können Sie zudem der Matrix beliebig viele weitere Levels zufügen und dann definieren.

Außerdem können Sie in der Matrix die *Quality Level* den verschiedenen Plattformen zuordnen bzw. festlegen, welche Qualitätsstufen dort zur Verfügung stehen sollen.

7.12.2 Qualitätsstufen per Code festlegen

Zum Nutzen dieser Qualitätsstufen in Spielen bietet Unity Ihnen die Klasse `QualitySettings` an, die verschiedene statische Methoden bereithält, um z.B. die *Quality Levels* abzufragen oder zu verändern.

Das Listing 7.2 zeigt Ihnen ein einfaches Skript, mit dem die Qualitätsstufen gesenkt oder angehoben werden können. Der Name der aktuellen Stufe wird dabei in einem Text-Objekt der GUI angezeigt. Neben diesem Text-Objekt brauchen Sie lediglich noch zwei Buttons Ihrer Szene zuzufügen, die dann die beiden Funktionen `IncreaseQuality` und `DecreaseQuality` aufrufen. Mehr zu diesem Thema erfahren Sie im Kapitel 14 „GUI“.

Listing 7.2 Einfaches Skript zum Ändern der Qualitätsstufen

```
using UnityEngine;
using System.Collections;
using UnityEngine.UI;

public class SetQualityLevel : MonoBehaviour {
    //Text-Objekt zur Anzeige des aktuellen Quality-Level-Namens
    public Text qualityText;
    //Array zum Speichern alle Quality-Level-Namen
    public string[] names;
    void Start() {
        //QualityLevel-Namen abfragen und Array zuweisen
        names = QualitySettings.names;
        //GUI aktualisieren
    }
}
```

```

    UpdateView();
}

public void IncreaseQuality()
{
    //Qualitätsstufe anheben
    QualitySettings.IncreaseLevel(true);
    //GUI aktualisieren
    UpdateView();
}

public void DecreaseQuality()
{
    //Qualitätsstufe senken
    QualitySettings.DecreaseLevel(true);
    //GUI aktualisieren
    UpdateView();
}
//Name des aktuellen Quality-Levels in der GUI aktualisieren
void UpdateView()
{
    //Quality-Level-Index abfragen, Name aus Array holen und GUI zuweisen
    qualityText.text = names[QualitySettings.GetQualityLevel()];
}
}

```

Mit einer einfach gehaltenen Oberfläche könnte das Skript in Verbindung mit drei UI-Elementen so aussehen wie in Bild 7.27.

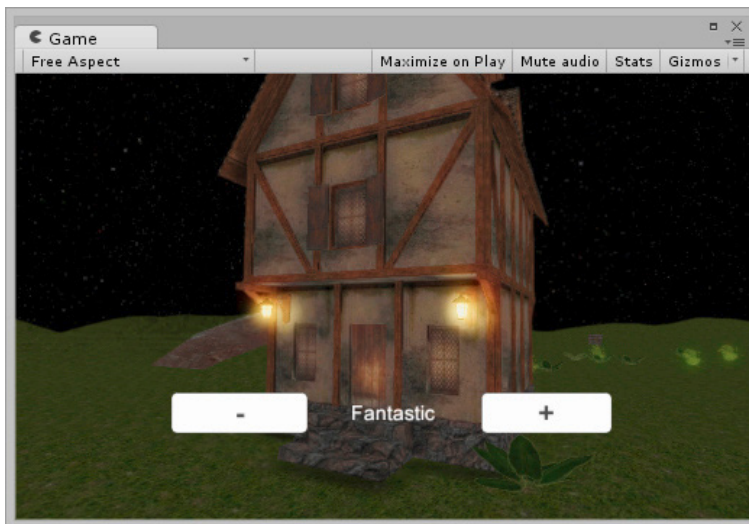


Bild 7.27 Einfache GUI zum Ändern der aktuellen Qualitätsstufe

Index

Symbole

2D 107, 143, 344, 393
2D-Button 10, 12
2D-Physik 247
3D-Koordinatensystem 107

A

A*-Algorithmus 429
Accelerometer 265
Accessoren 59
activeTerrain 335
Adaptive Reinhard 158
AddComponent 50, 86
AddExplosionForce 214
AddField 379
AddForce 213
AddForceAtPosition 214
Add Quality Level 207
AddRelativeForce 214
AddRelativeTorque 214
AddTorque 214
Advanced - Texture Type 134
Albedo (Standard Shader) 121
Allow Wet Mixing 281
Alpha Cutoff 120
Alpha-Kanal 134
Ambient GI 181
Ambient Intensity 181
Ambient Light 181
Ambient Occlusion 203
Ambient Source 181
Anchors 350
Anchor-Handle 351
Anchor Presets 350

Angular Drag 210
Animation 363, 389, 401
Animation-Clip 401, 406
Animation Events 427
Animation State 405, 567
Animation Types 399
Animation View 391
Animator 420
Animator Controller 404, 567
Antialiasing 158, 201
anyKey 257
Any State 408
Application 96
Apply Root Motion 396
Area Effector 2D 250
Area Light 185, 195
Array 39 ff., 58, 68
Asset 8 f., 15 f., 20 ff., 25, 29 ff.
Asset Store 31, 140, 326, 336
Attribut 87
Audio 269
AudioClip 276
Audio-Filter 278
AudioListener 157, 269
Audio Mixer 270, 278
AudioSource 270
Ausführungsreihenfolge 101
Avatar 399
Avatar Definition 401
Avatar Mask 409
Awake 80

B

Background 155
Backup 492

Baked GI 202
 Baken 195
 Baking 182
 Balancing 597
 Basisklasse 60, 74
 Batch 448
 Bedingte Operatoren 45
 BeginHorizontal 370
 BeginVertical 370
 Behaviour 76
 Beschleunigungssensor 265
 Betrag 109
 Bit-Feld 224
 Blending 410
 Blend Tree 408
 Blueprint 349
 Bone-Mapping 399
 Boo 73
 Box Collider 216
 Brush 321
 Build-In-Shader 118
 Button 359, 368
 Byte Order 324

C

C# 35, 73
 Camel Case 37
 Camera 155, 157
 CancellInvoke 92
 Canvas 344
 Canvas Scaler 347, 485
 Capsule 138
 Capsule Collider 216
 centerOfMass 212
 Character Controller 241 f., 245, 533
 Clear Flags 155
 Clipping Planes 156, 177
 Cluster 332
 Collider 86, 216
 Collision 223
 Collision2D 248 f.
 Collision Detection 211, 222
 CollisionEvent 304
 Component 9, 18, 27, 75, 81, 84
 Conditions 406
 Console 8, 22, 72, 99
 ConstantForce 216
 Constant Physical Size 347

Constant Pixel Size 347
 Constraints 211
 Continuous Baking 195
 Cookie 134
 Coroutine 90, 377
 Create from Grayscale 135
 CreateInstance 75
 Create Table 381
 Create Tree Collider 329
 CrossPlatformInput 263
 CSV 381
 Cube 138
 Cubemap 134, 189
 Culling Mask 155, 183
 Cursor 134, 260
 CursorMode 260
 Curves 392
 Cutout (Rendering Mode) 120
 Cylinder 138

D

Datenbank 377, 380
 Datenbanksprache 380
 Datentypen 38
 Debug 99
 Default Cursor 516
 Default Time 101
 Deferred Lighting 201
 deltaTime 78
 Depth 156, 169
 Destroy 74, 84, 87
 Detail Albedo x2 127
 Detail Mask 127
 Detail-Texturen 127
 Dictionary 69, 503
 diffuse Reflexion 130
 Directional Light 183
 Display-Orientierungen 265
 DontDestroyOnLoad 74, 98
 DontRequireReceiver 86
 Dope Sheet 392
 do-Schleife 49
 Drag 210
 Draw Call 448
 Ducking 284

E

Edit in Playmode 282
 Editor 100
 Editor Extensions 31, 104, 140
 Eigenschaftsmethode 59
 Eltern-Objekt 115
 Emission 125
 Empty GameObject 27, 114, 244
 enableEmission 303
 EndHorizontal 370
 EndVertical 370
 Enumeration 41
 Environment Lighting 181
 eulerAngles 116, 245
 Event 360, 404
 Event Camera 347
 Event-Delegates 358
 Event-Methoden 78
 EventSystem 344
 Event Trigger 364

F

Fade - Rendering Mode 120
 Farbe 130
 FBX 139, 398
 Field of View 156
 Filter Mode 145
 Find 83
 FindGameObjectsWithTag 83
 FindWithTag 80, 82
 First Person Controller 245
 fixedDeltaTime 209
 Fixed Joint 239
 Fixed Timestep 209 f., 221, 237
 FixedUpdate 79, 209, 213
 Flare Layer 157
 Flatten 322
 fontSize 366
 ForceMode 215
 foreach-Schleife 48
 for-Schleife 47
 Forward Friction 226
 Forward Rendering 199
 Frame 32, 78, 90
 Frustrum Culling 177

G

Game Controller 520
 Game-Design-Dokument 596
 GameObject 8 ff., 13, 15, 18, 26, 74 f., 80, 82 ff.,
 86, 96
 Game View 8, 12 f., 33
 Gamma-Korrektur 160
 Gamma-Rendering 160
 Gamma-Space 161
 Generate Colliders 425
 Generate root motion curves 396
 Generic - Animation Type 399
 gerichtete Reflexion 129
 GET 379
 .GetAxis 255
 GetButton 256
 GetButtonDown 245
 GetCollisionEvents 304
 GetComponent 85
 GetCurrentAnimatorStateInfo 424
 GetKey 257
 GetMouseButton 258
 GetMouseButtonDown 215
 GetQualityLevel 335
 GetTouch 263
 GetWorldPose 230
 Gizmo 11 f., 270
 Gizmo Display Toggles 15
 Graphical User Interface 343
 Grass Lighting 331
 Gravitation 210
 Gravity Scale 247
 Grayscale 123, 135, 188, 323
 GUI 134, 343
 GUI-Klasse 367
 GUILayout 157
 GUILayout-Klasse 370
 GUISkin 371
 GUIStyle 371

H

Hand-Tool 14
 Hard Shadows 186
 Has Exit Time 407
 Hash-Wert 424
 HDR-Rendering 157
 Heightmap 123, 135, 323 f.
 Hierarchy 8, 15 f., 18, 27, 33 f.

High Dynamic Range-Rendering 157
 Highscore 380
 Hinge Joint 239
 HTML 379
 Humanoid - Animation Type 399

I

Icon 18
 identity 117
 IEnumerator 91
 if-Anweisung 44
 Image 354, 362
 Image Effects 172
 Initialisierung 38
 Input 215, 255, 261
 InputField 360
 Input-Manager 251, 254, 267
 Input Settings 246
 Insert into 381
 insideUnitSphere 90
 Inspector 8, 59, 82
 Instantiate 117, 374
 Instanzieren 39, 50
 Instanzmember 56
 Interface 63
 Interpolate 211
 Intersection 304
 Inventarsystem 503
 Invoke 91
 InvokeRepeat 92
 IsInvoking 92
 Is Kinematic 210, 219, 266
 Is Trigger 220
 isWebPlayer 590

J

JavaScript 73
 Joints 239
 Joystick 254

K

Kamera 155
 KeyCode 257
 Keyframe 391
 KI 429, 576

Kind-Objekt 115
 Klasse 50, 60, 71, 76
 Klassendiagramm 74
 Klassenmember 56
 Klick-Sound 360
 Kollisionen 86, 216
 Kollisionserkennung 210, 216, 221
 Kompilierungsreihenfolge 100
 Komponente 17ff., 26, 50, 74f., 81, 84, 87
 Konsole 99
 Konstanten 41
 Künstliche Intelligenz 429, 564, 576

L

Label 367
 LateUpdate 81, 90
 Layer 15, 18, 29
 Layer-Based Collision Detection 223
 Layer Collision Matrix 223
 LayerMask 224, 542
 LDR-Rendering 157
 Lebensverwaltung 522
 Lens Flares 191
 Level Index 96, 98
 Level Of Detail 141
 Light 181
 Light Cookies 188
 Light Halos 190
 Lighting 195
 Lighting-Fenster 174, 181
 Lightmap 134, 195
 Lightmap Snapshot 196
 Lightmap Static 194
 Lightmapping 181, 186, 194
 Light Probe 196
 Light Probe Group 197
 Linear-Rendering 160
 Linear-Space 161
 linkshändiges Koordinatensystem 107
 List 68
 Lizenzmodelle 2
 localEulerAngles 116
 localPosition 116
 localRotation 116
 LOD 141
 LODGroup 142
 LookAt 116
 loop match 403
 Low Dynamic Range 157

M

magnitude 109
 Main Camera 155, 165, 270
 Main Maps 121
 Manual 9
 Mask 404
 Mass 210
 Masseschwerpunkt 212
 Mass place trees 329
 Material 117
 Mathf 56
 Mehrspieler-Games 267
 Mesh 111, 216 f.
 Mesh Collider 216 f., 223
 MeshFilter 112
 MeshRenderer 112, 187
 Metallic 121
 Metallic-Workflow 129, 131
 Methode 51
 Minimap 169
 Mipmap 145
 Missing (MonoBehaviour) 77
 MoCap 390
 Model Import Settings 140
 MonoBehaviour 51, 60, 74, 76, 78, 96
 MonoDevelop 2, 71 f., 77
 Mono-Framework 35
 Motion Capturing 390
 mousePosition 259
 Move 242
 Muscle 400
 mySQL 380
 mysql_query 381

N

nameHash 425
 Namespace 66, 74, 78
 Navigation 357
 Navigation Area 435
 NavigationMesh 430, 432
 NavMesh 432, 569
 NavMeshAgent 430, 565
 NavMeshObstacle 436, 570
 Negationsoperator 87
 Netzwerkspiele 268
 Noise 266
 Normale 112
 Normalenvektor 112, 187

Normalisieren 110
 normalized 110
 Normalmap 122, 134 f.
 null 84

O

Object 74
 Objektorientierte Programmiersprache 50
 Objektorientierte Programmierung 60, 67
 Occludee Static 179
 Occluder Static 179
 Occlusion 124
 Occlusion Area 179
 Occlusion Culling 177, 449
 Off-Mesh Link 431, 434, 437
 OnCollision 219 f.
 OnCollision2D 248 f.
 OnCollisionColliderHit 244
 OnDestroy 96
 OnGUI 80, 366
 OnLevelWasLoaded 98
 OnMouseDown 214
 OnMouseOver 215
 OnParticleCollision 298, 304, 313, 317
 OnTrigger2D 248
 Opaque - Rendering Mode 120
 Order in Layer 149, 150
 Orthogonal 12
 Orthographic 170
 Ortsvektor 109
 Other Settings 198
 out 57

P

Packing Tag 147
 Paint Details-Tool 330
 Paint Height-Tool 322
 Paint Texture-Tool 325
 Panel 355
 Parallax Scrolling 152
 Parametermodifizierer 57
 Parenting 16, 115, 163
 parsen 383
 Particle Effect Control 291
 ParticleSystem 289, 339
 Partikeleffekte 289
 Partikelsysteme 289, 340

Pathfinding 429, 564
 Perspective 24
 Perspektivisch 12
 PHP 377, 379 f.
 phpMyAdmin 381
 Physically Based Shading 119, 160
 Physic Effectors 250
 Physic Materials 226, 238
 Physics 2D 248
 Physics Settings 223
 Physik-Engine 79, 209, 216
 Pixelkoordinaten 259
 Pixelkoordinatensystem 259
 Pixel Lighting 199
 Place Tree-Tool 328
 Plane 138
 Platform Effector 2D 250
 PlayClipAtPoint 275, 317
 Play Controls 15
 Play Mode 12, 148
 PlayerPrefs 93 f., 96
 Point Effector 2D 250
 Point Light 184
 Polygon 111
 Polygonnetz 111
 POST 379
 Postprocessing-Effekte 172
 Precomputed Realtime GI 203
 Prefabs 84, 373
 Primitive Collider 211, 216 f., 222
 Primitives 138, 219
 Produktionsphasen 595
 Produktionsprozess 595
 Project Browser 8, 15 f., 18, 20 f., 24, 29 ff.,
 33
 Projection 156, 170
 Projector 192
 Projekt Browser 77
 Prozedurale Mesh-Generierung 141

Q

Quad 138
 Quality Level 207
 Quality Settings 207, 335
 Quaternion 90, 116, 245
 Quelltext 36
 Quest 548
 Questgeber 550

R

Raise/Lower-Tool 321
 Random 89
 Raw Image 355
 Raycast 58, 110, 222, 240
 RaycastHit 240
 Raycasting 240
 Rect-Handle 348 f.
 Rect-Tool 14, 348 f.
 RectTransform 76, 348
 ref 57
 Reflection Probe 204
 Reflection Probe Static 204
 Reflection Source 204
 Remote-Profiling 452
 Rename 77
 Rendering Mode (Standard Shader) 120
 Rendering Path 157, 198
 Rendering-Statistik 179, 447
 RenderSettings 174
 Render Texture 170
 RepeatButton 368
 RESET 114
 Reverb Zone 276
 RGB 134
 Richtungsvektor 109
 Rig 399
 Rigidbody 210, 266
 Root 115
 Root Motion 396, 403
 Rotate 116
 Rotate-Tool 14
 RotateTowards 245

S

Sandbox 377
 Scale-Tool 14
 Scale with Screen 347
 Scene Gizmo 11 f.
 Scene View 8, 11 ff., 18, 24, 33, 107, 216
 Schatten 186
 Schnittstellen 63
 Schriftgröße 366
 Screen 365
 ScreenPointToRay 166, 259
 Screen Space - Camera 346, 363
 Screen Space - Overlay 346
 Script Execution Order Settings 101

- ScriptableObject 75, 102, 419
 - Scripting Reference 3, 9, 72
 - Scrollbar 362
 - Secondary Maps 127
 - SELECT 382
 - Send Collision Messages 303
 - SendMessage 85
 - SendMessageOptions 85
 - SetActive 83
 - SetCursor 260
 - SetDestination 431
 - SetQualityLevel 335
 - Shader 117
 - Shader Calibration Scene 130
 - Shadow Type 183
 - Shuriken 289
 - Sichtbarkeit 55, 61
 - Sideways Friction 226
 - SimpleMove 242
 - Singleton 98
 - Size 156
 - SkinnedMeshRenderer 112
 - Skript 71, 76, 81, 100
 - Skybox 174
 - Slider 361
 - Smartphones 261
 - Smooth Height-Tool 323
 - Smoothness 122, 131, 133
 - Snapping 14
 - Snapping-Modus 349
 - Snap Settings 14
 - Snapshots 286
 - Soft Shadows 186
 - Sorting Layer 149, 150
 - Source-Code 36
 - Spawn-Point 536
 - Specular 121
 - Specular-Workflow 129, 132
 - Sphere 138
 - Sphere Collider 216
 - Spherical Harmonics 199
 - Spiegel 171, 204
 - Split 383
 - Split-Screen 168, 267
 - Spot Light 184
 - sprachübergreifende Zugriff 100
 - Spring Joint 239
 - Sprite 15, 134, 144, 352, 354, 394
 - Sprite Animation 299, 393
 - Sprite Atlas 145, 147
 - Sprite Editor 146, 352, 394
 - Sprite-Element 145
 - Sprite Packer 147
 - SpriteRenderer 149
 - Sprite-Shader 151
 - Sprite Sheet 144
 - SQL 380
 - sqrMagnitude 109
 - Stand-alone 93, 252
 - Stand-alone Input Module 344
 - Standard (Specular setup) 119
 - Standard Assets 100
 - Standard-Shader 118, 119
 - Start 80
 - State Machine 404
 - Static 18, 194
 - Static Collider 221
 - Stiffness 227
 - StringToHash 424
 - Subfenster 369
 - Sub-Quest 558
 - Sub-State Machine 411
 - Sun 174
 - Superglobals 379
 - Surface Effector 2D 250
 - Suspension Spring 226
 - switch-Anweisung 46
 - Syntax 36
 - Szene 25, 96
- ## T
- Tablets 261
 - Tabs 7
 - Tag 28, 82
 - Target Texture 157, 170 f.
 - Terrain 223, 319, 339
 - Terrain Collider 223, 320, 329
 - Terrain Settings 323 f.
 - Text 353, 362
 - TextArea 368
 - TextField 368
 - Texture 134
 - Texture Atlas 144
 - Texture Type 352, 394
 - Tiefpass-Filter 266
 - Tilling 128
 - Time 78, 209
 - Time Manager 209
 - Toggle 360
 - Toggle Group 360

- Tonemapping 158
- Touch 261
- TouchCount 262
- Touch Input Module 344
- Transform 14, 76, 79, 114
- Transform-Tools 10, 11, 13 ff., 33, 114, 348
- Transition 357, 406
- Translate 116, 266
- Translate-Tool 14
- Transparent - Rendering Mode 120
- Tree 328 f., 339
- Triangle 111, 136
- Trigger 220
- Trigger-Collider 220, 272

U

- uGUI 344
- UI Scale Modes 347
- UnityEngine 74, 78
- UnityEngine.UI 344
- UnityGUI 366
- UnityPackage 30
- UnityScript 74
- Update 78, 90
- Upgrading 25
- Use Gravity 210
- Use Light Probes 196
- UV Mapping 136

V

- var 39
- Vector3 108, 116
- Vektor 108
- Vererbung 60
- Vergleichsoperatoren 44
- Versiegeln 62
- Version Control System 492
- Versionsverwaltung 492
- Vertex Lighting 199, 331
- Vertex Lit 200

- Vertex-Snapping 14
- Vertices 111
- View Port Rect 156, 168
- Views 287
- virtuelle Achsen 251, 253
- virtuelle Tasten 253

W

- WaitForSeconds 91
- WASD-Tastensteuerung 245, 533
- Wasser 337
- Webplayer 2, 93, 377, 590
- Webplayer Template 100, 590
- Webspace 380
- Wegfindung 429
- Wet Mixing 281
- Wheel Collider 225
- Wheel Friction Curve 226, 237
- WheelHit 232
- while-Schleife 49
- Wind Zones 328, 336, 339
- World Space 347
- Wrap Mode 128
- WWWForm 379
- WWW-Klasse 377

X

- XML 381

Y

- yield 91, 377

Z

- Zeilenumbrüche 354
- Zufallswerte 89
- Zugriffsmodifizierer 64, 84