

[Spiele entwickeln mit Unity 5](#)

2D- und 3D-Games mit Unity und C# für Desktop, Web & Mobile. Für Unity 5.6

Bearbeitet von
Von: Carsten Seifert, und Jan Wislaug

3., aktualisierte und erweiterte Auflage 2017. Buch. 670 S. Softcover
ISBN 978 3 446 45197 1
Format (B x L): 18,5 x 24,6 cm
Gewicht: 1322 g

[Weitere Fachgebiete > EDV, Informatik > Programmiersprachen: Methoden > Spiele-
Programmierung, Rendering, Animation](#)

Zu [Inhaltsverzeichnis](#)

schnell und portofrei erhältlich bei

The logo for beck-shop.de features the text 'beck-shop.de' in a bold, red, sans-serif font. Above the 'i' in 'shop' are three red dots of varying sizes, arranged in a slight arc. Below the main text, the words 'DIE FACHBUCHHANDLUNG' are written in a smaller, red, all-caps, sans-serif font.

beck-shop.de
DIE FACHBUCHHANDLUNG

Die Online-Fachbuchhandlung [beck-shop.de](#) ist spezialisiert auf Fachbücher, insbesondere Recht, Steuern und Wirtschaft. Im Sortiment finden Sie alle Medien (Bücher, Zeitschriften, CDs, eBooks, etc.) aller Verlage. Ergänzt wird das Programm durch Services wie Neuerscheinungsdienst oder Zusammenstellungen von Büchern zu Sonderpreisen. Der Shop führt mehr als 8 Millionen Produkte.



Leseprobe

Carsten Seifert, Jan Wislaug

Spiele entwickeln mit Unity 5

2D- und 3D-Games mit Unity und C# für Desktop, Web & Mobile. Für
Unity 5.6

ISBN (Buch): 978-3-446-45197-1

ISBN (E-Book): 978-3-446-45368-5

Weitere Informationen oder Bestellungen unter

<http://www.hanser-fachbuch.de/978-3-446-45197-1>

sowie im Buchhandel.

Inhalt

Vorwort	XIX
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	5
2 Grundlagen	7
2.1 Installation	7
2.2 Oberfläche	7
2.2.1 Hauptmenü	9
2.2.2 Scene View	10
2.2.3 Game View	12
2.2.4 Toolbar	14
2.2.5 Hierarchy	16
2.2.6 Inspector	17
2.2.7 Project Browser	21
2.2.8 Console	23
2.3 Das Unity-Projekt	23
2.3.1 Neues Projekt anlegen	24
2.3.2 Bestehendes Projekt öffnen	25
2.3.3 Projektdateien	26
2.3.4 Szene	26
2.3.5 Game Objects	27
2.3.6 Tags	29
2.3.7 Layer	30
2.3.8 Assets	31
2.3.9 Frames	34
2.4 Das erste Übungsprojekt	34

3	C# und Unity	37
3.1	Die Sprache C#	37
3.2	Syntax	38
3.3	Kommentare	39
3.4	Variablen	39
	3.4.1 Namenskonventionen	39
	3.4.2 Datentypen	40
	3.4.3 Schlüsselwort var	41
	3.4.4 Datenfelder/Array	41
3.5	Konstanten	43
	3.5.1 Enumeration	43
3.6	Typkonvertierung	44
3.7	Rechnen	44
3.8	Verzweigungen	45
	3.8.1 if-Anweisungen	46
	3.8.2 switch-Anweisung	48
3.9	Schleifen	49
	3.9.1 for-Schleife	49
	3.9.2 Foreach-Schleife	50
	3.9.3 while-Schleife	50
	3.9.4 do-Schleife	51
3.10	Klassen	51
	3.10.1 Komponenten per Code zuweisen	52
	3.10.2 Instanziierung von Nichtkomponenten	52
	3.10.3 Werttypen und Referenztypen	54
	3.10.4 Überladene Methoden	55
3.11	Der Konstruktor	55
	3.11.1 Konstruktoren in Unity	56
3.12	Lokale und globale Variablen	56
	3.12.1 Namensverwechslung verhindern mit this	56
3.13	Zugriff und Sichtbarkeit	57
3.14	Statische Klassen und Klassenmember	57
3.15	Parametermodifizierer out/ref	58
3.16	Array-Übergabe mit params	59
3.17	Eigenschaften und Eigenschaftsmethoden	60
3.18	Vererbung	61
	3.18.1 Basisklasse und abgeleitete Klassen	62
	3.18.2 Vererbung und die Sichtbarkeit	62
	3.18.3 Geerbte Methode überschreiben	63
	3.18.4 Zugriff auf die Basisklasse	63
	3.18.5 Klassen versiegeln	64
3.19	Polymorphie	64
3.20	Schnittstellen	65
	3.20.1 Schnittstelle definieren	65

3.20.2	Schnittstellen implementieren	65
3.20.3	Zugriff über eine Schnittstelle	66
3.21	Namespaces	67
3.21.1	Eigene Namespaces definieren	68
3.22	Generische Klassen und Methoden	69
3.22.1	List	69
3.22.2	Dictionary	70
4	Skript-Programmierung	73
4.1	MonoDevelop	73
4.1.1	Hilfe in MonoDevelop	74
4.1.2	Syntaxfehler	74
4.2	Nutzbare Programmiersprachen	75
4.2.1	Warum C#?	76
4.3	Unitys Vererbungsstruktur	76
4.3.1	Object	77
4.3.2	GameObject	77
4.3.3	ScriptableObject	77
4.3.4	Component	77
4.3.5	Transform	78
4.3.6	Behaviour	78
4.3.7	MonoBehaviour	78
4.4	Skripte erstellen	78
4.4.1	Skripte umbenennen	79
4.5	Das Skript-Grundgerüst	80
4.6	Unitys Event-Methoden	80
4.6.1	Update	81
4.6.2	FixedUpdate	81
4.6.3	Awake	82
4.6.4	Start	82
4.6.5	OnGUI	82
4.6.6	LateUpdate	83
4.6.7	Aufruf-Reihenfolge	83
4.7	Komponentenprogrammierung	84
4.7.1	Auf GameObjects zugreifen	84
4.7.2	GameObjects aktivieren und deaktivieren	86
4.7.3	GameObjects zerstören	86
4.7.4	GameObjects erstellen	86
4.7.5	Auf Components zugreifen	87
4.7.6	Components hinzufügen	89
4.7.7	Components entfernen	89
4.7.8	Components aktivieren und deaktivieren	89
4.7.9	Attribute	90
4.8	Zufallswerte	91

4.9	Parallel Code ausführen	92
4.9.1	WaitForSeconds	93
4.10	Verzögerte und wiederholende Funktionsaufrufe mit Invoke	94
4.10.1	Invoke	94
4.10.2	InvokeRepeating, IsInvoking und CancelInvoke	94
4.11	Daten speichern und laden	95
4.11.1	PlayerPrefs-Voreinstellungen	95
4.11.2	Daten speichern	96
4.11.3	Daten laden	97
4.11.4	Key überprüfen	97
4.11.5	Löschen	97
4.11.6	Save	97
4.12	Szeneübergreifende Daten	98
4.12.1	Werteübergabe mit PlayerPrefs	98
4.12.2	Zerstörung unterbinden	100
4.13	Debug-Klasse	102
4.14	Kompilierungsreihenfolge	102
4.14.1	Programmsprachen mischen und der sprachübergreifende Zugriff	103
4.15	Ausführungsreihenfolge	103
4.16	Plattformabhängig Code kompilieren	104
4.17	Eigene Assets mit ScriptableObject	105
4.17.1	Neue ScriptableObject-Subklasse erstellen	105
4.17.2	Instanzen eines ScriptableObjects erstellen	106
5	Objekte in der zweiten und dritten Dimension	109
5.1	Das 3D-Koordinatensystem	109
5.2	Vektoren	110
5.2.1	Ort, Winkel und Länge	111
5.2.2	Normalisieren	112
5.3	Das Mesh	113
5.3.1	Normalenvektor	114
5.3.2	MeshFilter und MeshRenderer	115
5.4	Transform	117
5.4.1	Kontextmenü der Transform-Komponente	117
5.4.2	Objekthierarchien	118
5.4.3	Scripting mit Transform	119
5.4.4	Quaternion	119
5.5	Shader und Materials	120
5.5.1	Der Standard-Shader	121
5.5.2	Texturen	138
5.5.3	UV Mapping	141
5.6	3D-Modelle einer Szene zufügen	142
5.6.1	Primitives	142
5.6.2	3D-Modelle importieren	144

5.6.3	In Unity modellieren	145
5.6.4	Prozedurale Mesh-Generierung	146
5.6.5	Level Of Detail	146
5.7	2D in Unity	148
5.7.1	Sprites	149
5.7.2	SpriteRenderer	154
5.7.3	Parallax Scrolling	157
6	Kameras, die Augen des Spielers	161
6.1	Die Kamera	161
6.1.1	Komponenten eines Kamera-Objektes	163
6.1.2	HDR – High Dynamic Range-Rendering	163
6.1.3	Linearer- und Gamma-Farbraum	166
6.2	Kamerasteuerung	169
6.2.1	Statische Kamera	169
6.2.2	Parenting-Kamera	170
6.2.3	Kamera-Skripte	170
6.3	ScreenPointToRay	172
6.4	Mehrere Kameras	173
6.4.1	Kamerawechsel	173
6.4.2	Split-Screen	174
6.4.3	Einfache Minimap	175
6.4.4	Render Texture	177
6.5	Image Effects	179
6.5.1	Beispiel: Haus bei Nacht	179
6.6	Skybox	181
6.6.1	Mehrere Skyboxen gleichzeitig einsetzen	182
6.6.2	Skybox selber erstellen	183
6.7	Occlusion Culling	184
6.7.1	Occluder Static und Occludee Static	186
6.7.2	Occlusion Culling erstellen	186
7	Licht und Schatten	189
7.1	Environment Lighting	189
7.2	Lichtarten	191
7.2.1	Directional Light	192
7.2.2	Point Light	193
7.2.3	Spot Light	194
7.2.4	Area Light	195
7.3	Schatten	196
7.3.1	Einfluss des MeshRenderers auf Schatten	197
7.4	Light Cookies	198
7.4.1	Import Settings eines Light Cookies	198
7.4.2	Light Cookies und Point Lights	199

7.5	Light Halos	200
7.5.1	Unabhängige Halos	201
7.6	Lens Flares	201
7.6.1	Eigene Lens Flares	202
7.7	Projector	202
7.7.1	Standard Projectors	202
7.8	Lightmapping	204
7.8.1	Light Probes	207
7.9	Rendering Paths	209
7.9.1	Forward Rendering	210
7.9.2	Vertex Lit	211
7.9.3	Deferred Lighting	212
7.10	Global Illumination	213
7.10.1	Baked GI	214
7.10.2	Realtime Lighting	215
7.10.3	Lightmapping Settings	216
7.11	Light Explorer	217
7.12	Reflexionen (Spiegelungen)	218
7.12.1	Reflection Probes	219
7.13	Qualitätseinstellungen	222
7.13.1	Quality Settings	222
7.13.2	Qualitätsstufen per Code festlegen	222
8	Physik in Unity	225
8.1	Physikberechnung	225
8.2	Rigidbodyes	226
8.2.1	Rigidbodyes kennenlernen	227
8.2.2	Masseschwerpunkt	228
8.2.3	Kräfte und Drehmomente zufügen	229
8.3	Kollisionen	232
8.3.1	Collider	232
8.3.2	Trigger	236
8.3.3	Static Collider	238
8.3.4	Kollisionen mit schnellen Objekten	238
8.3.5	Terrain Collider	239
8.3.6	Layer-basierende Kollisionserkennung	239
8.3.7	Mit Layer-Masken arbeiten	240
8.4	Wheel Collider	242
8.4.1	Wheel Friction Curve	243
8.4.2	Entwicklung einer Fahrzeugsteuerung	245
8.4.3	Autokonfiguration	252
8.4.4	Fahrzeugstabilität	254
8.5	Physic Materials	254
8.6	Joints	255
8.6.1	Fixed Joint	255

8.6.2	Spring Joint	256
8.6.3	Hinge Joint	256
8.7	Raycasting	256
8.8	Character Controller	258
8.8.1	SimpleMove	258
8.8.2	Move	259
8.8.3	Kräfte zufügen	260
8.8.4	Einfacher First Person Controller	261
8.9	2D-Physik	263
8.9.1	OnCollision2D- und OnTrigger2D-Methoden	265
8.9.2	2D Physic Effectors	266
9	Maus, Tastatur, Touch	269
9.1	Virtuelle Achsen und Tasten	269
9.1.1	Der Input-Manager	269
9.1.2	Virtuelle Achsen	271
9.1.3	Virtuelle Tasten	271
9.1.4	Steuern mit Mauseingaben	272
9.1.5	Joystick-Inputs	272
9.1.6	Anlegen neuer Inputs	273
9.2	Achsen- und Tasteneingaben auswerten	273
9.2.1	GetAxis	273
9.2.2	GetButton	274
9.3	Tastatureingaben auswerten	275
9.3.1	GetKey	275
9.3.2	anyKey	275
9.4	Mauseingaben auswerten	276
9.4.1	GetMouseButton	276
9.4.2	Mauseingaben auf Objekten per Event	277
9.4.3	mousePosition	277
9.4.4	Mauszeiger ändern	278
9.5	Touch-Eingaben auswerten	280
9.5.1	Der Touch-Typ	280
9.5.2	Input.touches	281
9.5.3	TouchCount	281
9.5.4	GetTouch	281
9.5.5	CrossPlatformInput	282
9.6	Beschleunigungssensor auswerten	283
9.6.1	Input.acceleration	284
9.6.2	Tiefpass-Filter	285
9.7	Steuerungen bei Mehrspieler-Games	286
9.7.1	Split-Screen-Steuerung	286
9.7.2	Netzwerkspiele	287

10	Audio	289
10.1	AudioListener	289
10.2	AudioSource	290
	10.2.1 Durch Mauern hören verhindern	292
	10.2.2 Sound starten und stoppen	294
	10.2.3 Temporäre AudioSource	295
10.3	AudioClip	296
	10.3.1 Länge ermitteln	296
10.4	Reverb Zone	296
10.5	Filter	298
10.6	Audio Mixer	298
	10.6.1 Das Audio Mixer-Fenster	298
	10.6.2 Audiosignalwege	302
	10.6.3 Mit Snapshots arbeiten	306
	10.6.4 Views erstellen	307
	10.6.5 Parameter per Skript bearbeiten	307
11	Partikeleffekte mit Shuriken	311
11.1	Editor-Fenster	312
11.2	Particle Effect Control	313
11.3	Numerische Parametervarianten	313
11.4	Farbparameter-Varianten	314
11.5	Default-Modul	314
11.6	Effekt-Module	316
	11.6.1 Emission	316
	11.6.2 Shape	316
	11.6.3 Velocity over Lifetime	318
	11.6.4 Limit Velocity over Lifetime	318
	11.6.5 Inherit Velocity	319
	11.6.6 Force over Lifetime	319
	11.6.7 Color over Lifetime	319
	11.6.8 Color by Speed	320
	11.6.9 Size over Lifetime	320
	11.6.10 Size by Speed	320
	11.6.11 Rotation over Lifetime	320
	11.6.12 Rotation by Speed	321
	11.6.13 External Forces	321
	11.6.14 Noise	321
	11.6.15 Collision	322
	11.6.16 Triggers	323
	11.6.17 Sub Emitter	325
	11.6.18 Texture-Sheet-Animation	325
	11.6.19 Lights	326
	11.6.20 Trails	326
	11.6.21 Renderer	327

11.7	Partikelemission starten, stoppen und unterbrechen	329
11.7.1	Play	330
11.7.2	Stop	330
11.7.3	Pause	330
11.7.4	enableEmission	330
11.8	OnParticleCollision	331
11.8.1	GetCollisionEvents	331
11.9	Feuer erstellen	332
11.9.1	Materials erstellen	332
11.9.2	Feuer-Partikelsystem	333
11.9.3	Rauch-Partikelsystem	336
11.10	Wassertropfen erstellen	340
11.10.1	Tropfen-Material erstellen	340
11.10.2	Wassertropfen-Partikelsystem	341
11.10.3	Kollisionspartikelsystem	343
11.10.4	Kollisionsound	345
12	Landschaften gestalten	347
12.1	Was Terrains können und wo die Grenzen liegen	348
12.2	Terrainhöhe verändern	348
12.2.1	Pinsel	349
12.2.2	Oberflächen anheben und senken	349
12.2.3	Plateaus und Schluchten erstellen	350
12.2.4	Oberflächen weicher machen	351
12.2.5	Heightmaps	351
12.3	Terrain texturieren	353
12.3.1	Textur-Pinsel	354
12.3.2	Texturen verwalten	354
12.4	Bäume und Sträucher	356
12.4.1	Bedienung des Place Tree-Tools	357
12.4.2	Wälder erstellen	357
12.4.3	Mit Bäumen kollidieren	357
12.5	Gräser und Details hinzufügen	358
12.5.1	Detail-Meshs	359
12.5.2	Gräser	360
12.5.3	Quelldaten nachladen	360
12.6	Terrain-Einstellungen	361
12.6.1	Base Terrain	361
12.6.2	Resolution	361
12.6.3	Tree & Details Objects	362
12.6.4	Wind Settings	362
12.6.5	Zur Laufzeit Terrain-Eigenschaften verändern	363
12.7	Der Weg zum perfekten Terrain	364
12.8	Gewässer	365

13	Wind Zones	367
13.1	Spherical vs. Directional	368
13.2	Wind Zone - Eigenschaften	369
13.3	Frische Brise	370
13.4	Turbine	370
14	GUI	371
14.1	Das UI-System uGUI	372
14.1.1	Canvas	372
14.1.2	RectTransform	376
14.1.3	UI-Sprite Import	380
14.1.4	Grafische Controls	381
14.1.5	Interaktive Controls	385
14.1.6	Controls designen	392
14.1.7	Animationen in uGUI	393
14.1.8	Event Trigger	394
14.2	Screen-Klasse	395
14.2.1	Schriftgröße dem Bildschirm anpassen	395
14.3	OnGUI-Programmierung	396
14.3.1	GUI	397
14.3.2	GUILayout	399
14.3.3	GUIStyle und GUISkin	400
15	Prefabs	403
15.1	Prefabs erstellen und nutzen	403
15.2	Prefab-Instanzen erzeugen	403
15.2.1	Instanzen per Code erstellen	404
15.2.2	Instanzen weiter bearbeiten	405
15.3	Prefabs ersetzen und zurücksetzen	405
15.4	Prefab-Verbindungen auflösen	406
16	Internet und Datenbanken	407
16.1	Die WWW-Klasse	407
16.1.1	Rückgabewert-Formate	408
16.1.2	Parameter übergeben	409
16.2	Datenbank-Kommunikation	410
16.2.1	Daten in einer Datenbank speichern	410
16.2.2	Daten von einer Datenbank abfragen	411
16.2.3	Rückgabewerte parsen	413
16.2.4	Datenhaltung in eigenen Datentypen	414
16.2.5	HighscoreCommunication.cs	416
16.2.6	Datenbankverbindung in PHP	417

17	Animationen	419
17.1	Allgemeiner Animation-Workflow	420
17.2	Animationen erstellen	420
17.2.1	Animation View	421
17.2.2	Curves vs. Dope Sheet	422
17.2.3	Animationsaufnahme	422
17.2.4	Beispiel Fallgatter-Animation	427
17.3	Animationen importieren	428
17.3.1	Rig	429
17.3.2	Animationen	431
17.4	Animationen einbinden	434
17.4.1	Animator Controller	435
17.4.2	Animator-Komponente	450
17.4.3	Beispiel Fallgatter: Animator Controller	451
17.5	Controller-Skripte	453
17.5.1	Parameter des Animator Controllers setzen	454
17.5.2	Animation States abfragen	454
17.5.3	Beispiel Fallgatter Controller-Skript	455
17.6	Animation Events	457
17.7	Das „alte“ Animationssystem	458
18	Künstliche Intelligenz	461
18.1	NavMeshAgent	462
18.1.1	Eigenschaften der Navigationskomponente	463
18.1.2	Zielpunkt zuweisen	464
18.1.3	Pfadsuche unterbrechen und fortsetzen	464
18.2	Navigation-Fenster	465
18.2.1	Agents Tab	466
18.2.2	Object Tab	467
18.2.3	Bake Tab	467
18.2.4	Areas Tab	468
18.3	NavMeshObstacle	469
18.4	Off-Mesh Link	470
18.4.1	Automatische Off-Mesh Links	470
18.4.2	Manuelle Off-Mesh Links	471
18.5	Point & Click-Steuerung für Maus und Touch	472
19	Fehlersuche und Performance	475
19.1	Fehlersuche	475
19.1.1	Breakpoints	476
19.1.2	Variablen beobachten	477
19.1.3	Console Tab nutzen	478
19.1.4	GUI- und GUILayout nutzen	478
19.1.5	Fehlersuche bei mobilen Plattformen	479

19.2	Performance	481
19.2.1	Rendering-Statistik	482
19.2.2	Batching-Verfahren	483
19.2.3	Analyse mit dem Profiler	484
19.2.4	Echtzeit-Analyse auf Endgeräten	486
20	Spiele erstellen und publizieren	489
20.1	Der Build-Prozess	489
20.1.1	Szenen des Spiels	490
20.1.2	Plattformen	491
20.1.3	Notwendige SDKs	491
20.1.4	Plattformspezifische Optionen	492
20.1.5	Developer Builds	492
20.2	Publizieren	493
20.2.1	App	494
20.2.2	Browser-Game	494
20.2.3	Desktop-Anwendung	495
21	Erstes Beispiel-Game: 2D-Touch-Game	497
21.1	Projekt und Szene	497
21.1.1	Die Kamera	499
21.1.2	Texturen importieren und Sprites definieren	500
21.2	Gespenster und Hintergrund	502
21.2.1	Gespenster animieren	505
21.2.2	Gespenster laufen lassen	509
21.2.3	Gespenster-Prefab erstellen	511
21.3	Der GameController	512
21.3.1	Der Spawner	512
21.3.2	Level-Anzeige	514
21.3.3	Der Input-Controller	515
21.3.4	Game Over-UI	517
21.3.5	Hintergrundmusik	524
21.4	Punkte zählen	525
21.5	Spielende	526
21.6	Spiel erstellen	527
22	Zweites Beispiel-Game: 3D Dungeon Crawler	529
22.1	Level-Design	530
22.1.1	Modellimport	531
22.1.2	Materials konfigurieren	532
22.1.3	Prefabs erstellen	533
22.1.4	Dungeon erstellen	535
22.1.5	Dekoration erstellen	540
22.2	Inventarsystem erstellen	542
22.2.1	Verwaltungslogik	542

22.2.2	Oberfläche des Inventarsystems	550
22.2.3	Inventar-Items	553
22.3	Game Controller	560
22.4	Spieler erstellen	560
22.4.1	Lebensverwaltung	562
22.4.2	Spielersteuerung	573
22.4.3	Wurfstein entwickeln	581
22.4.4	Lautstärke steuern	587
22.5	Quest erstellen	588
22.5.1	Erfahrungspunkte verwalten	588
22.5.2	Questgeber erstellen	590
22.5.3	Sub-Quest erstellen	599
22.6	Gegner erstellen	604
22.6.1	Model-, Rig- und Animationsimport	604
22.6.2	Komponenten und Prefab konfigurieren	605
22.6.3	Animator Controller erstellen	607
22.6.4	NavMesh erstellen	609
22.6.5	Umgebung und Feinde erkennen	610
22.6.6	Gesundheitszustand verwalten	613
22.6.7	Künstliche Intelligenz entwickeln	617
22.7	Eröffnungsszene	626
22.7.1	Szene erstellen	626
22.7.2	Startmenü-Logik erstellen	627
22.7.3	Menü-GUI erstellen	629
22.8	WebGL-Anpassungen	631
22.8.1	WebGL-Input ändern	631
22.8.2	Quit-Methode in WebGL abfangen	632
22.9	Finale Einstellungen	633
22.10	So könnte es weitergehen	636
23	Der Produktionsprozess in der Spieleentwicklung	637
23.1	Die Produktionsphasen	637
23.1.1	Ideen- und Konzeptionsphase	638
23.1.2	Planungsphase	638
23.1.3	Entwicklungsphase	638
23.1.4	Testphase	639
23.1.5	Veröffentlichung und Postproduktion	639
23.2	Das Game-Design-Dokument	639
24	Schlusswort	641
	Index	643

Vorwort

Für viele von uns sind Computerspiele und Handysgames heutzutage allgegenwärtige Wegbegleiter. Egal ob auf dem Smartphone, dem Tablet, installiert auf dem heimischen PC oder direkt aufgerufen im Browser werden sie von uns täglich genutzt. 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 mittlerweile viele, teilweise sogar staatlich anerkannte, Studiengänge existieren, die sich dem Entwickeln von Computerspielen widmen.

In diesem Buch möchten wir Ihnen Unity, eine weit verbreitete Entwicklungsumgebung für Computerspiele und auch andere Anwendungen, näherbringen. Wir erklären und erläutern ausführlich, wie Sie mit diesem Werkzeug Spiele entwickeln können. Dabei richtet sich das Buch sowohl an Einsteiger und Umsteiger als auch an Spieleentwickler, die mit Unity nun richtig durchstarten möchten.

Als ursprünglicher Autor dieses Buches möchte ich, Carsten Seifert, mich an dieser Stelle 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.

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 bei der Arbeit an den ersten beiden Auflagen des Buches jederzeit mit Rat und Tat zur Seite standen.

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.

Ich, Jan Wislaug, möchte mich als überarbeitender Autor vor allem bei Sylvia Hasselbach vom Hanser Verlag bedanken. Sie stand mir bei der Arbeit an der dritten Auflage immer freundlich mit Rat und Tat beiseite.

Darüber hinaus danke ich Carsten Seifert für die gute Zusammenarbeit und dafür, dass sich sein Werk dank der guten Struktur so toll überarbeiten ließ.

Zusammen möchten wir beide Will Goldstone und Unity Technologies danken, die uns die bis dato aktuellsten Beta-Versionen zur Verfügung gestellt haben.

Nicht zuletzt danken wir auch der gesamten Community, die zum einen Carsten Seifert auf seinem Blog und seinen sozialen Kanälen begleitet hat und zum anderen Jan Wislaug bei seiner Überarbeitung auf die neue Unity-Version unterstützte.

Carsten Seifert und Jan Wislaug

im Juli 2017

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 Environment Lighting

Unity besitzt eine globale Beleuchtung namens *Environment Lighting* (oder ehemals auch *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/SETTINGS** erreichen. Im Bereich „Environment“ (siehe Bild 7.1) finden Sie die Parameter des *Environment Lighting*:

- **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).
- **Intensity Multiplier** bestimmt die Lichtstärke des *Diffuse Lighting*. Sehen Sie diesen Wert als Multiplikator.
- **Ambient Mode** legt fest, wie die globale Beleuchtung berechnet werden soll. Im Modus *Realtime* sind Änderungen der Parameter möglich und wirken sich aus. Bei *Baked* wird die Beleuchtung mit in die *Lightmaps* eingerechnet. Änderungen zur Laufzeit sind dann nicht mehr möglich. Dieser Parameter ist nur editierbar wenn in den Bereichen *Realtime* und *Mixed Lighting* jeweils die beiden Checkboxes aktiv sind.

Wie genau das *Diffuse Lighting* berechnet wird, ob zum Beispiel zur Laufzeit oder aber vorab beim *Baken* der *Lightmaps* wird weiter unten in den Einstellungen über die *Global Illumination* geregelt. Mehr dazu lesen Sie in Abschnitt 7.10, „Global Illumination“.

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

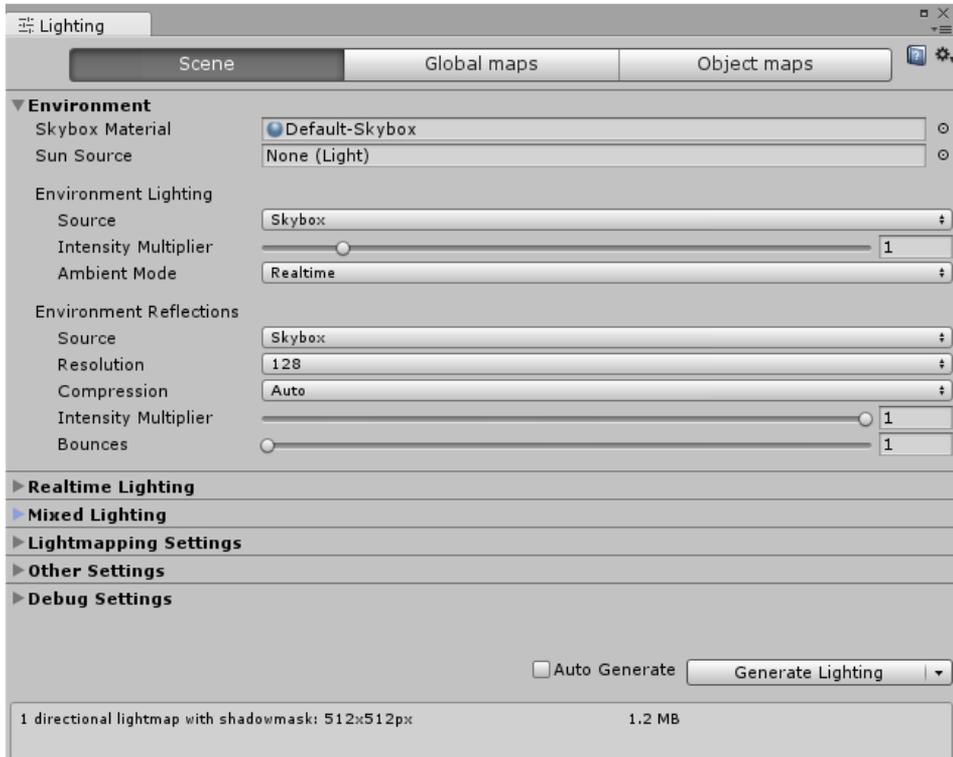


Bild 7.1 Lighting-Fenster

■ 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 wir noch eingehen werden. 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.
- **Color** legt die Lichtfarbe fest,
- **Range** setzt die Reichweite der Lichtquelle fest (sichtbar, falls die Lichtquelle diesen Parameter unterstützt)
- **Mode** 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.
- **Intensity** definiert die Lichtstärke.
- **Indirect Multiplier** bestimmt die Helligkeit des Lichts, das von angestrahlten Flächen zurückgeworfen wird, auch indirekte Beleuchtung genannt. Mehr zu diesem Parameter erfahren Sie in Abschnitt 7.10, „Global Illumination“.
- **Shadow Type** legt die Art des Schattens fest. Allgemein stehen *No Shadows*, *Hard Shadows* und *Soft Shadows* zur Verfügung. Auf die Details werden wir gleich noch eingehen.
- **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. Der zugehörige Parameter **Cookie Size** steuert dabei die Größe dieser Lichtschablone.
- **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 7.9 „Rendering Paths“) Echtzeitschatten erzeugen.

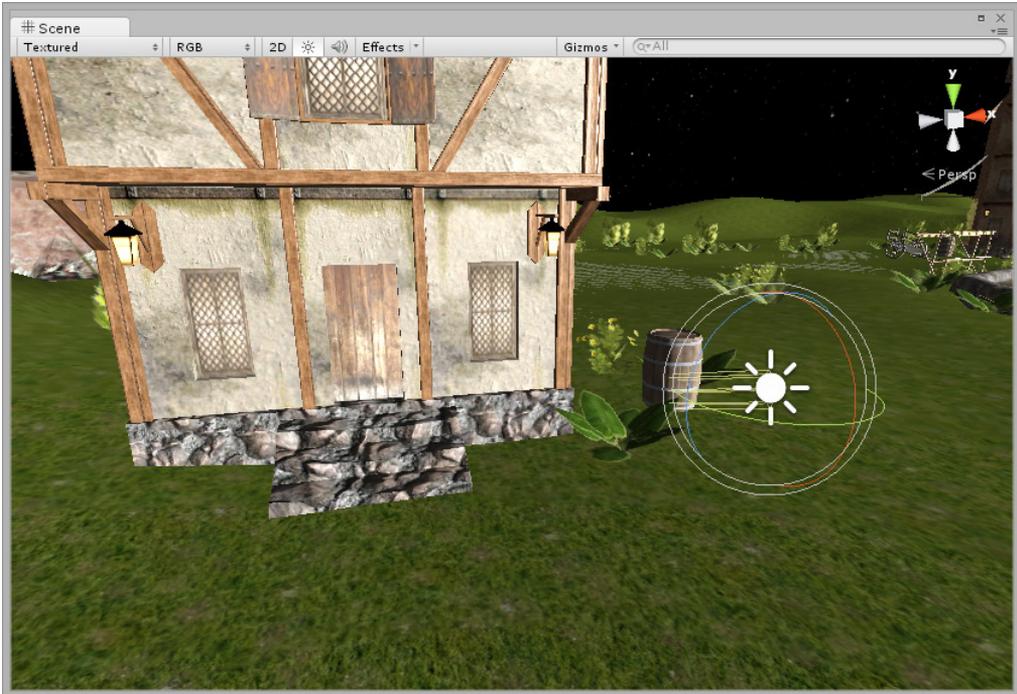


Bild 7.2 Directional Light

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.

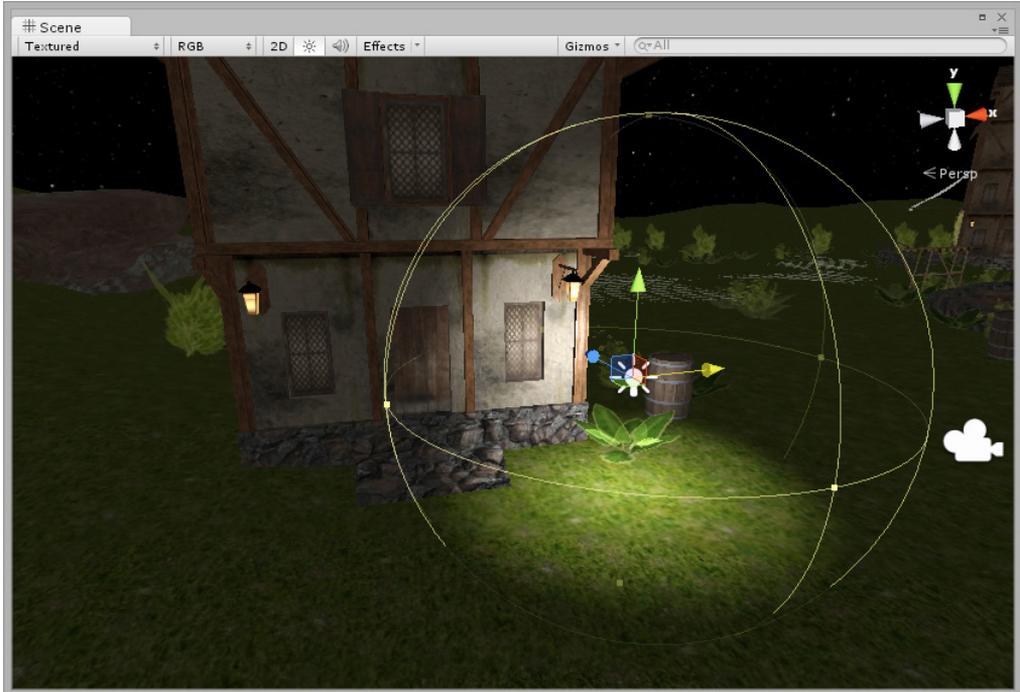


Bild 7.3 Point Light

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.

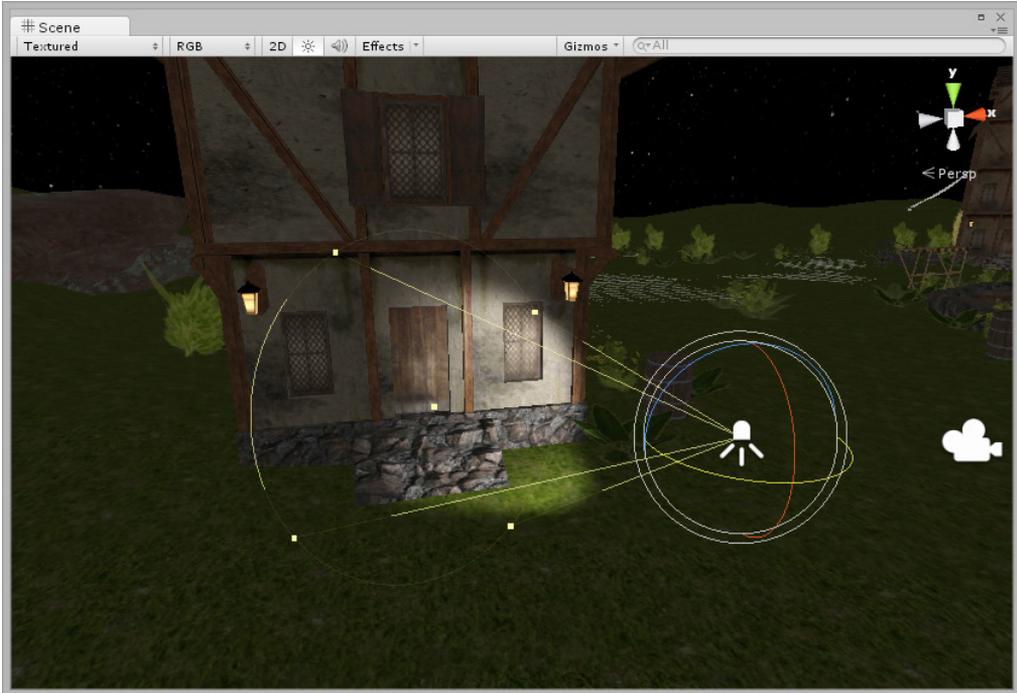


Bild 7.4 Spot Light

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 *Baked Global Illumination* aktiviert ist (mehr dazu finden Sie in Abschnitt 7.10).

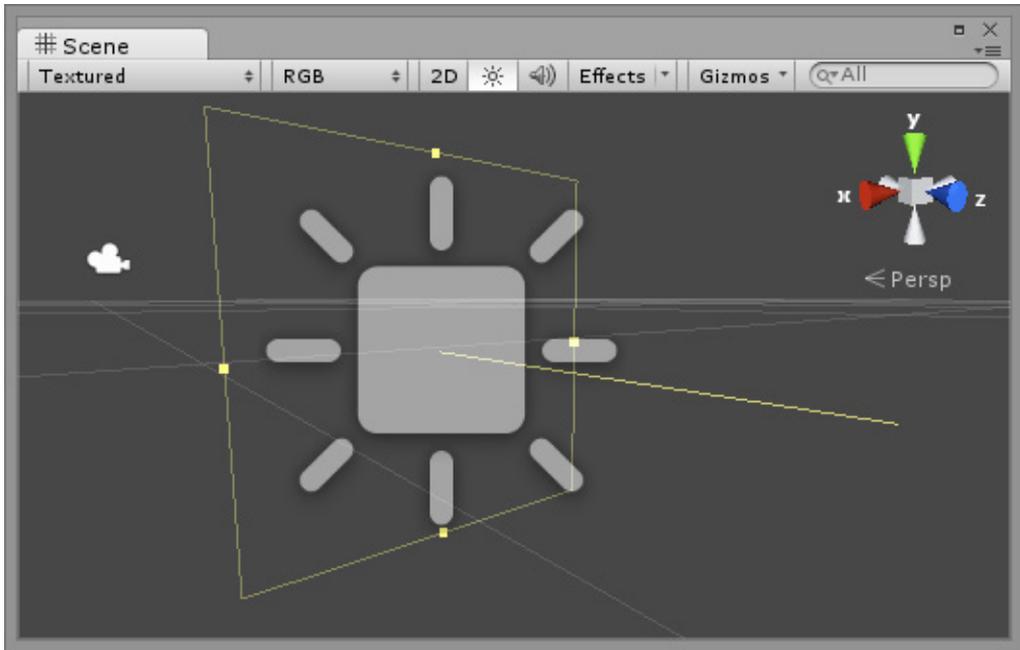


Bild 7.5 Area Light

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.

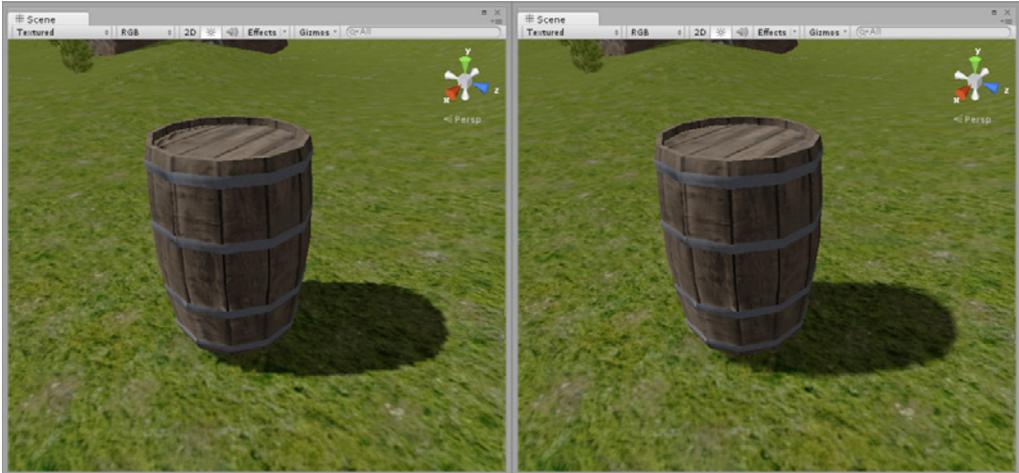


Bild 7.6 Vergleich: Hard Shadows vs. Soft Shadows

Neben der Schattenart bieten die verschiedenen 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.
- **Normal Bias** hat Einfluss darauf, ab wann die schattenwerfende Oberfläche etwas geschrumpft wird. Dabei wird das Objekt selbst nicht beeinflusst. Dieser Wert ist nützlich, um Artefakte zu verhindern, die durch eventuelle Selbstbeschattung entstehen.
- **Near Plane** hat mit dem Abstand von Lichtquelle und Objekt zu tun. Dieser Wert kontrolliert, ab welcher Nähe Schatten erzeugt wird.
- **Baked Shadow Angle** legt fest, wie stark eine nachträgliche Weichzeichnung auf die Kanten von gebackenem Licht angewendet werden soll. Bei Lichtern, die den *Mode Baked* haben, ist nur dieser Parameter einstellbar.

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. 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 „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*.

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.

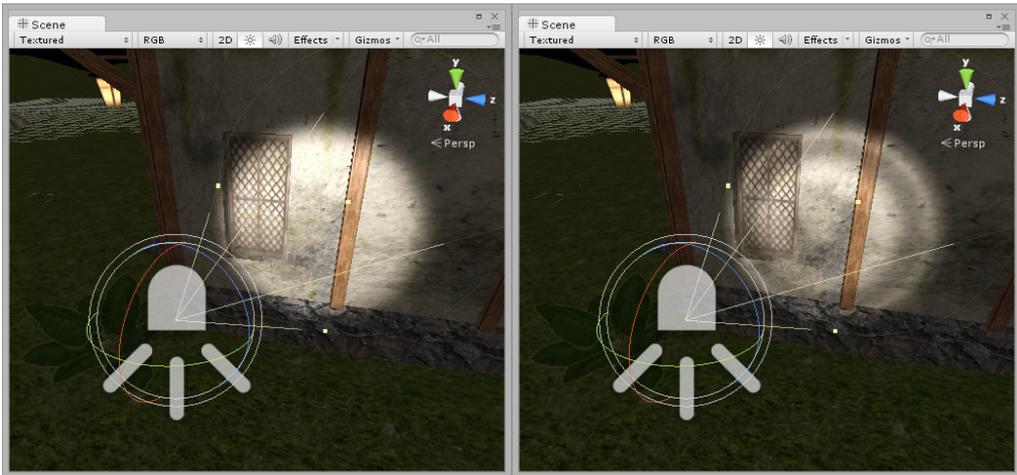


Bild 7.8 Spot Light mit einem taschenlampenähnlichen Light Cookie

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 Motives 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.

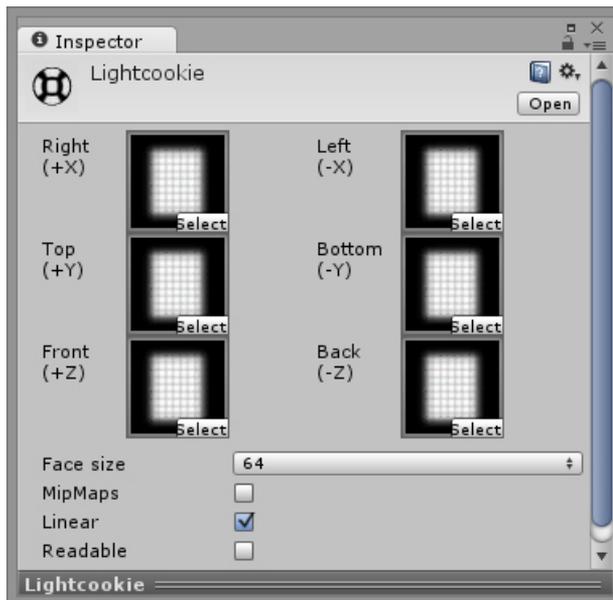


Bild 7.9 Beispiel eines Cubemaps Light Cookie

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

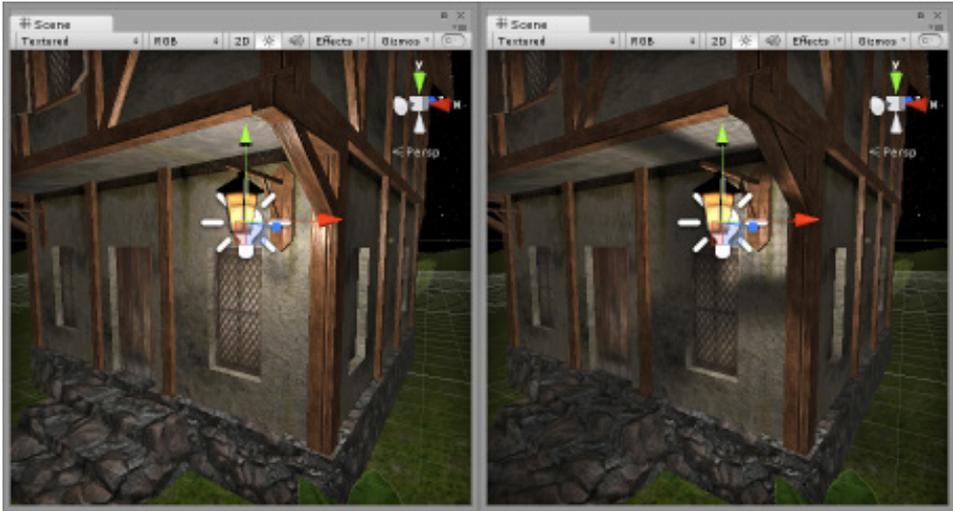


Bild 7.10 Einsatz eines Light Cookies bei einem Point Light

■ 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.

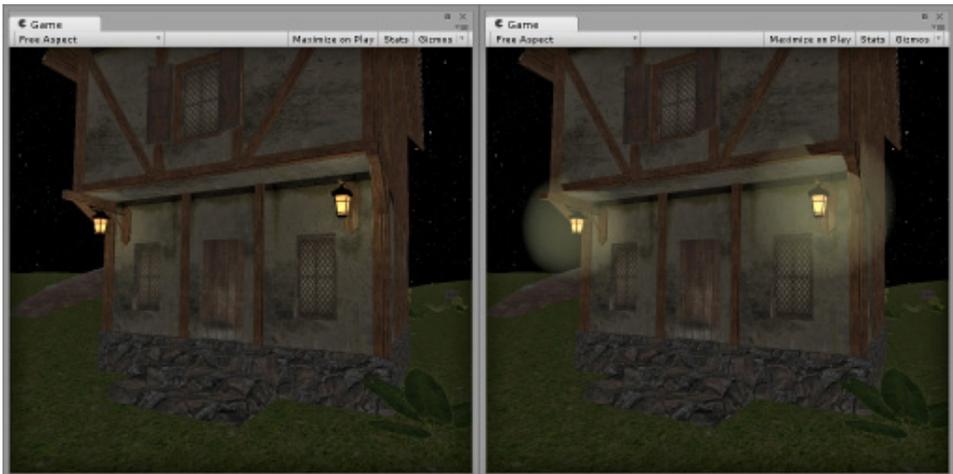


Bild 7.11 Light Halos

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. 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.

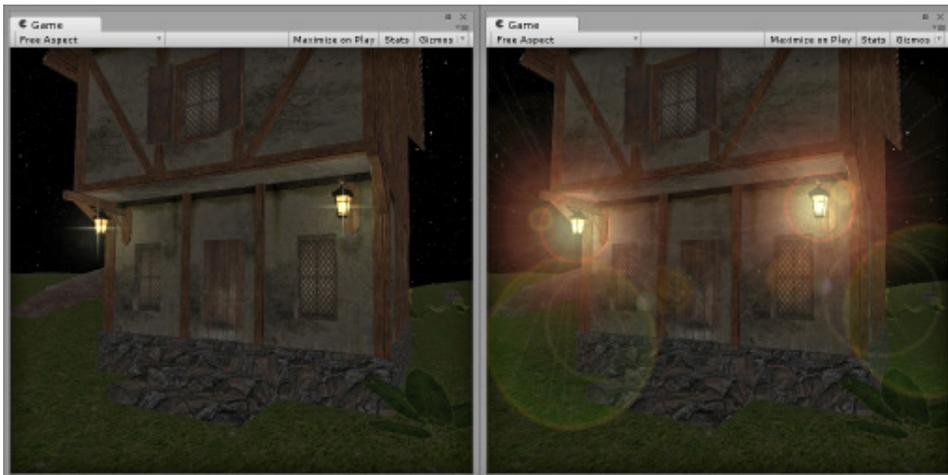


Bild 7.12 Lens Flares „Small Flare“ und „50 mm Zoom“

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*-Texturen 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 Projektionskegel 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. Platzieren Sie diesen *Projector* über einem Objekt und strahlen Sie auf diesen hinab, können Sie damit einen Schatten simulieren.

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.

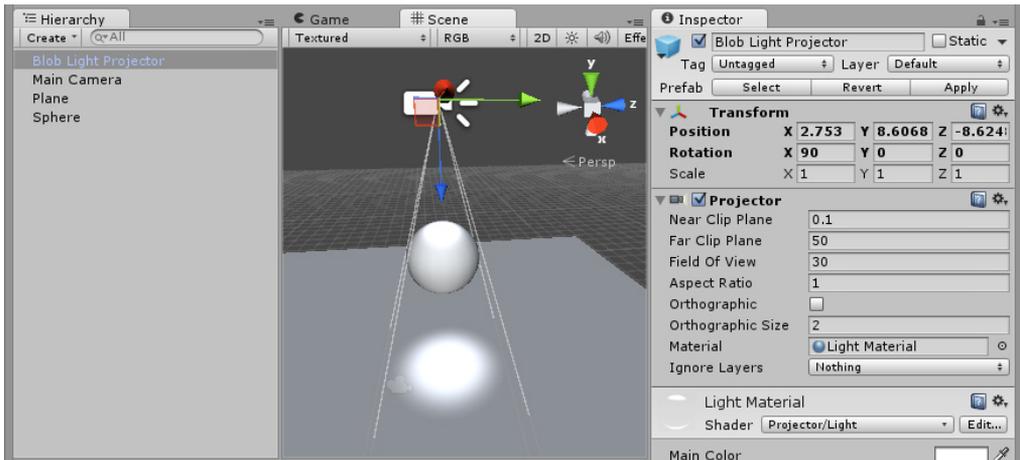


Bild 7.13 Blob Light Projector

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.

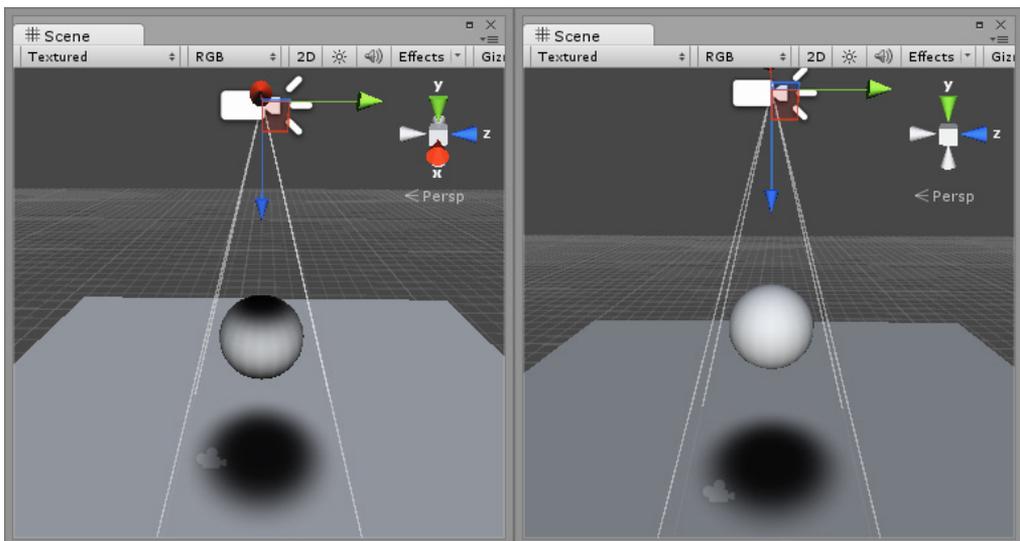


Bild 7.14 Blob Shadow Projector

■ 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 *Baked* gestellt ist. Allerdings bietet Unity auch eine Möglichkeit, bewegliche Objekte vom *Lightmapping* profitieren zu lassen. Hierbei werden sogenannte *Light Probes* eingesetzt, die wir aber noch in Abschnitt 7.8.1 „Light Probes“ behandeln werden. Mit dem Modus *Mixed* ist es ebenfalls möglich, statische und bewegte Objekte miteinander zu kombinieren, was die Lichtberechnung angeht. Mehr dazu finden Sie in Abschnitt 7.10, „Global Illumination“.

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.

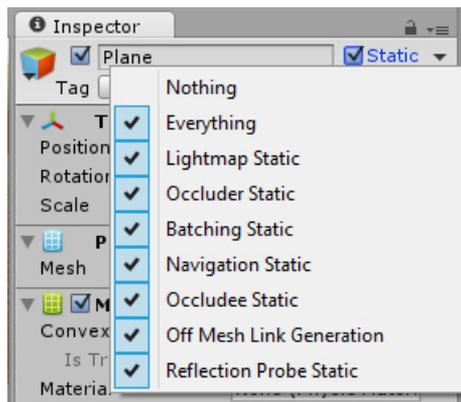


Bild 7.15 „Lightmap Static“-Option im Static-Menü

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 *Auto Generate* deaktiviert, wird das *Baken* nur noch manuell über den daneben befindlichen Knopf **GENERATE LIGHTING** 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 „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.

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.

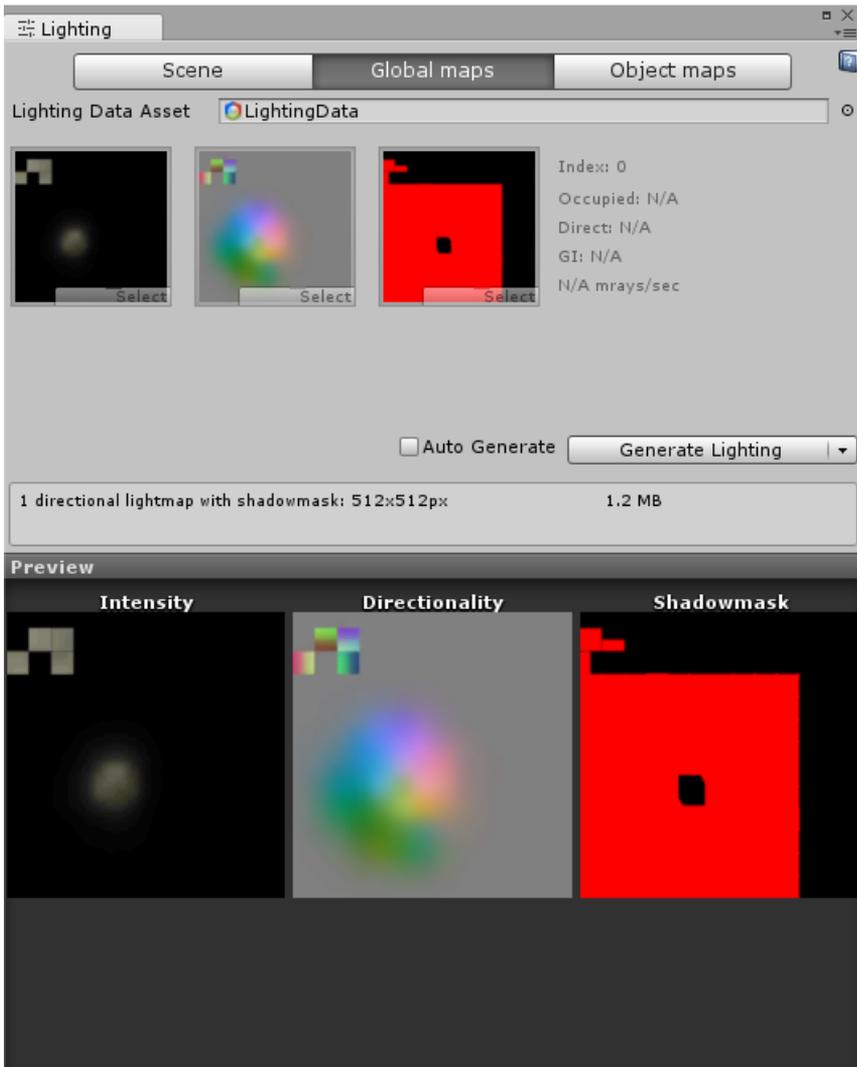


Bild 7.16 Lighting-Fenster mit Lightmaps-Bereich

Beachten Sie zudem auf dem *Scene*-Reiter des *Lighting*-Fensters die Funktion *Baked GI*. Sie berechnet die *Global Illumination* (siehe Abschnitt 7.10 „Global Illumination“), also die indirekte Beleuchtung beim *Lightmapping*. Ist diese deaktiviert, können Sie auch keine *Lightmaps* erstellen.

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.

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 „Objekte in der zweiten und dritten Dimension“). Hinzu kommen eine statische Plane (*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.

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.

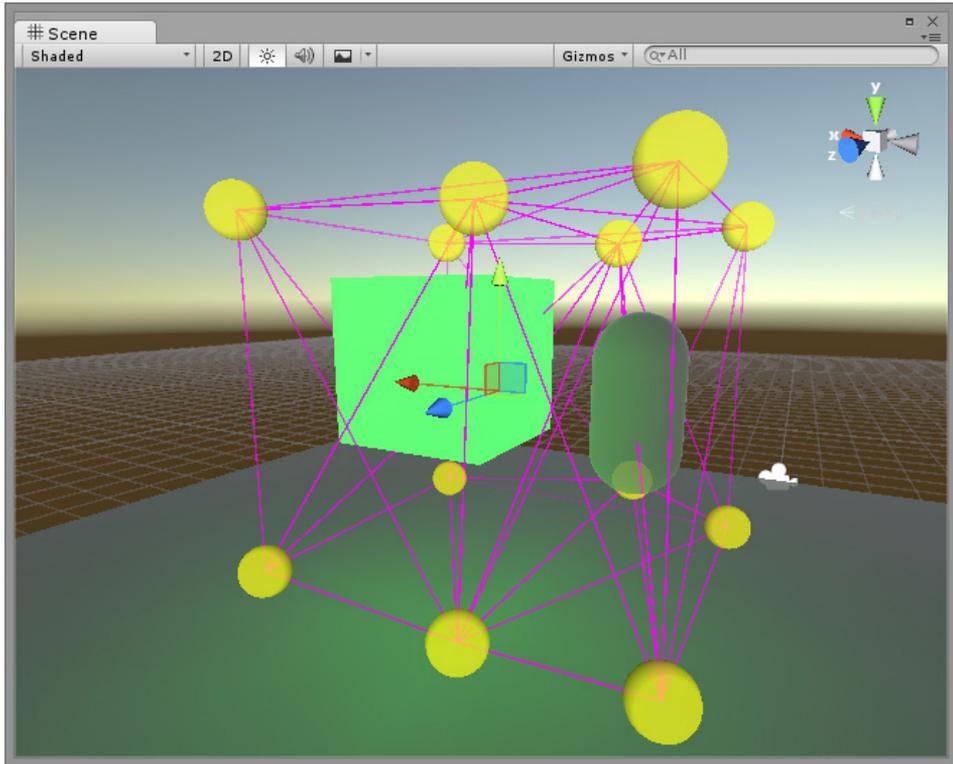


Bild 7.17 Im 3D-Raster angeordnete Light Probes

Am Anfang ist es sinnvoll, *Light Probes* zu positionieren, dass sie wie in Bild 7.17 in einem gleichmäßigem 3D-Gitter 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.

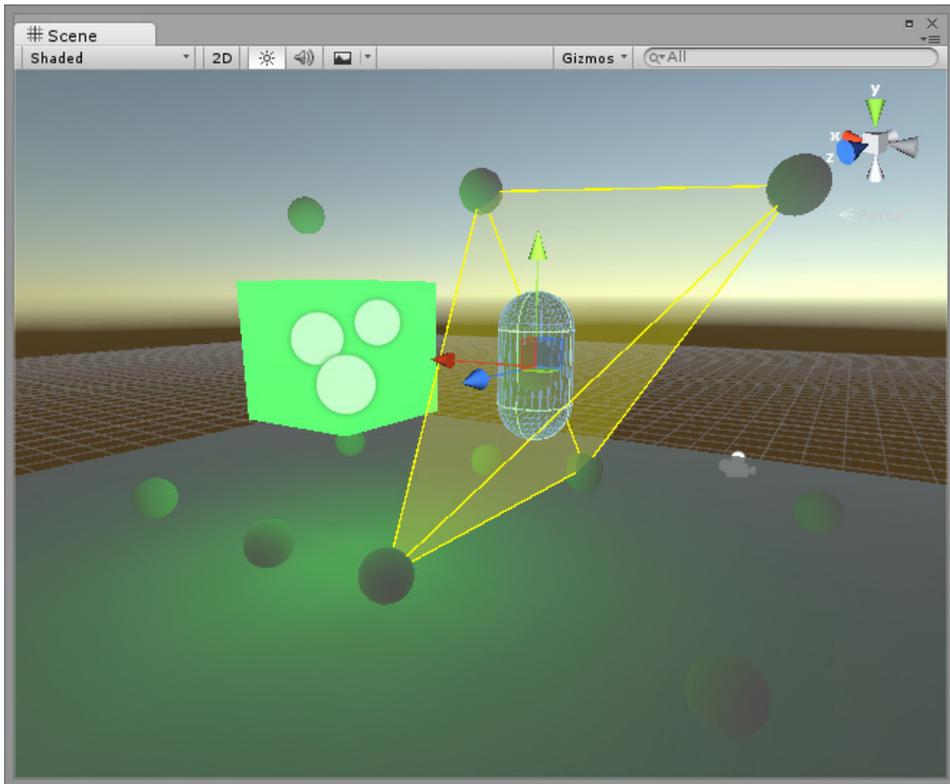


Bild 7.18 Mit Light Probes ausgeleuchtetes Objekt

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 wir im Folgenden noch weiter vorstellen werden. 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 wichtigsten.

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.

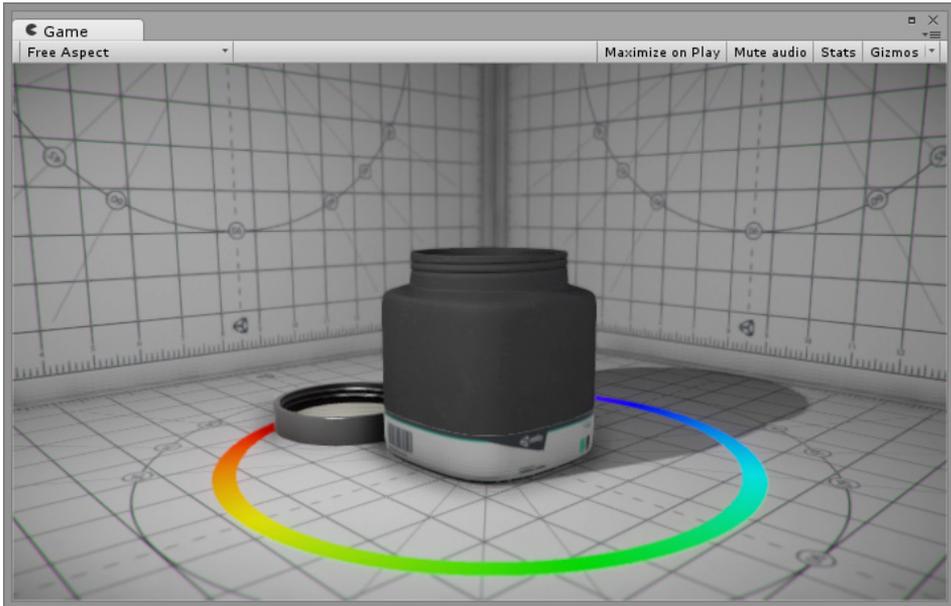


Bild 7.19 Mit Forward Rendering dargestellte Szene

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 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.

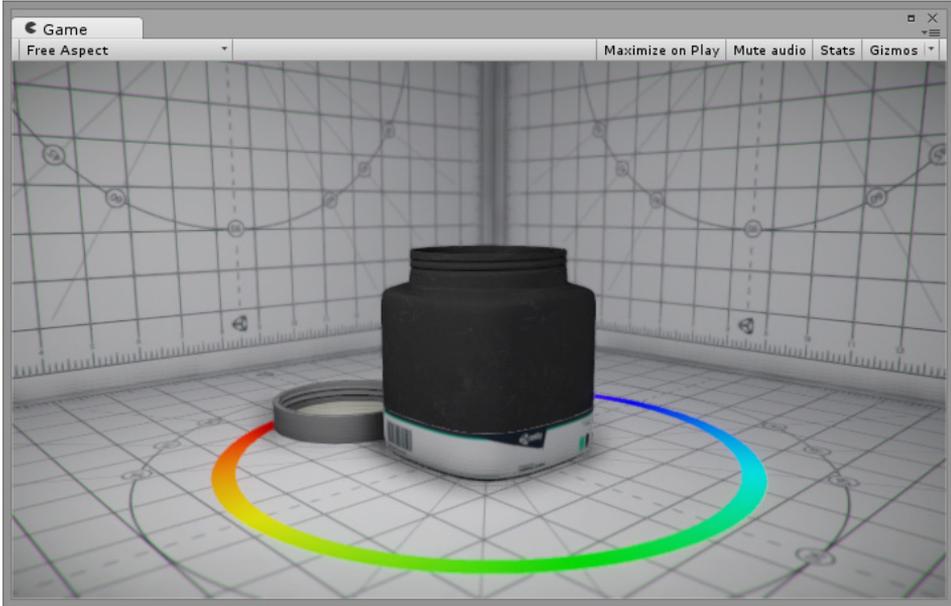


Bild 7.20 Gerenderte Szene mit Vertex Lit

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.

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.

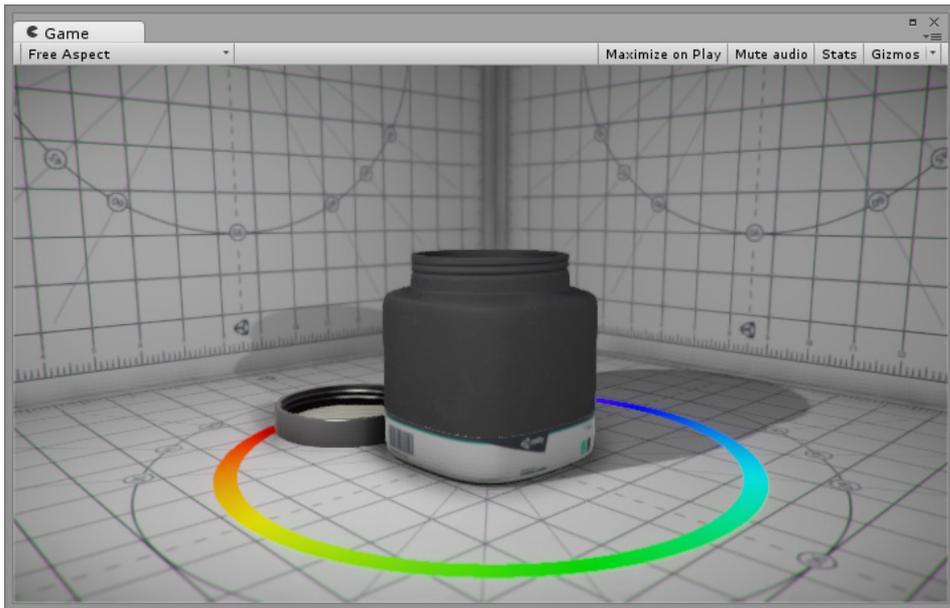


Bild 7.21 Gerenderte Szene mit Deferred Lighting

■ 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 *Indirect Multiplier*-Parameter, wie stark das Licht von einer Fläche abgestrahlt wird und die Umgebung erhellt. Bild 7.22 zeigt Ihnen, wie durch die reflektierende (indirekte) Beleuchtung eines Spotlights auch der Dachkasten und die Seitenbalken eines Holzhauses angeleuchtet werden.

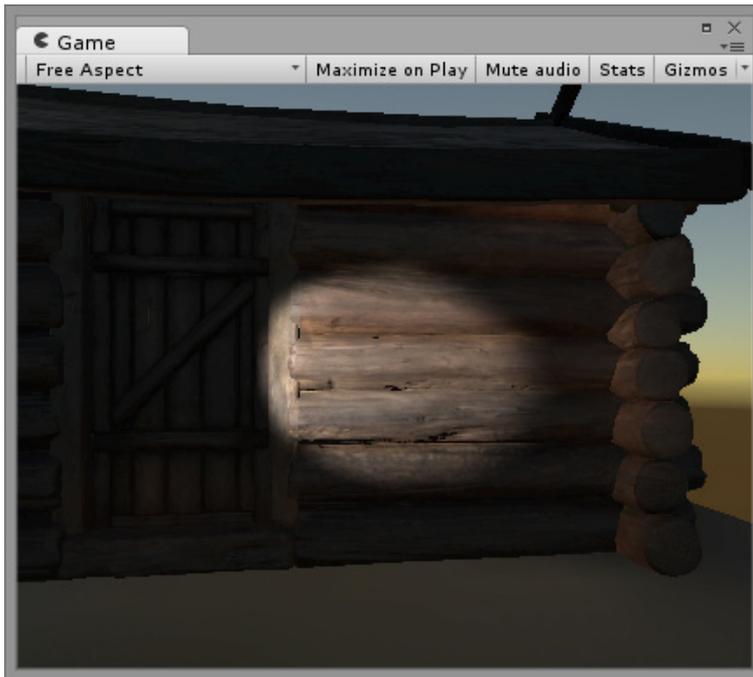


Bild 7.22 Indirekte Beleuchtung durch ein Spotlight

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 *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 *Baked* gestellt sein. Allerdings werden bei diesem Verfahren nur statische Objekte berücksichtigt, die mit *Static* oder *Lightmap Static* gekennzeichnet wurden.

Im *Lighting-Fenster* (*WINDOW/LIGHTING/SETTINGS*) gibt es für *Baked GI* einen extra Bereich, in dem Sie für dieses Verfahren einige Grundeinstellungen vornehmen und auch die komplette Funktion deaktivieren können. Der Bereich heißt **Mixed Lighting**. Sie können dort über die Checkbox den ganzen Bereich und damit das *Baked GI* deaktivieren. Ist der Bereich aktiviert, haben Sie über den Parameter *Lighting Mode* die Möglichkeit festzulegen, in welchem Modus die Berechnung stattfinden soll. Dieses Feature ist mit Unity 5.6 neu hinzugekommen. Folgende Auswahlmöglichkeiten sind vorhanden:

- **Baked Indirect** bietet sich für PCs an, deren Performance im mittleren Bereich liegt, oder für High-End-Mobilgeräte. Hierbei wird nur Licht vorberechnet, keine Schatten. Das bedeutet, dass Schatten immer zur Laufzeit berechnet werden.

- **Distance Shadowmask** bietet sich für High-End-Geräte wie einen performanten PC oder neuste Konsolen an. In diesem Modus können statische Objekte auch Schatten auf dynamische Objekte werfen. Schatten von statischen und dynamischen Objekten werden zusammen berechnet und verarbeitet.
- **Shadowmask** bietet sich für weniger bis durchschnittlich performante PCs oder Mobilgeräte an. Dieser Modus berechnet Schatten von statischen Objekten inklusive deren Selbstbeschattung untereinander vor. Außerdem werden Schatten von dynamischen Objekten korrekt mit vorberechneten Schatten kombiniert. Schatten von statischen Objekten auf dynamischen können nur mithilfe von *Light Probes* erzielt werden.
- **Subtractiv** bietet sich für wenig performante Mobilgeräte an. Dieser Modus ist der am wenigsten rechenintensive. Damit ist er aber auch limitiert, was die Möglichkeiten angeht, Licht und Schatten zu verarbeiten. Hierbei wird die direkte Beleuchtung von vornherein in die *Lightmap* mit einbezogen, ohne Rücksicht auf dynamische Schatten oder andere Einflüsse. Änderungen des Lichts haben zur Laufzeit also keinerlei Einfluss. Einzig das „*main directional light*“, welches häufig die Sonne ist, ist in der Lage, Echtzeitschatten zu werfen.

Je nachdem, welche Voraussetzungen Sie haben, sollten Sie die verschiedenen Modi ausprobieren und jenen wählen, der den besten Kompromiss aus Leistung und Optik bietet.

Direkt mit diesem Bereich verwandt sind die meisten Parameter der **Lightmapping Settings** in Abschnitt 7.10.3.

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 Realtime Lighting

Das zweite *Global Illumination*-Verfahren ist das sogenannte *Precomputed Realtime GI*, also vorberechnetes Echtzeit-GI. Ab Unity 5.6 wird es nur noch *Realtime Global Illumination* genannt.

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 *Realtime Global Illumination* allerdings deaktivieren, können Sie dies im *Lighting-Settings*-Fenster machen. Dazu deaktivieren Sie einfach die Checkbox im Abschnitt **Realtime Lighting**. In Abschnitt 7.10.3 finden Sie einen Hinweis auf den Parameter *Indirect Resolution*, der nur editierbar ist, wenn das **Realtime**

Lighting-Modul aktiv ist. Dieser Parameter hat direkt mit der Berechnung des Echtzeit-Lichtes zu tun.

7.10.3 Lightmapping Settings

Wie Licht berechnet wird, ist ein sehr komplexes Thema und so waren die Einstellungsmöglichkeiten dazu in Unity schon immer sehr umfangreich und auf den ersten Blick nicht leicht durchschaubar. Auch in der aktuellen Version 5.6 hat sich hier wenig getan. Außerdem unterliegt dieses Thema ständiger Veränderung und Verbesserung, weshalb wir Ihnen hier nochmals die Unity-eigene Dokumentation ans Herz legen möchten: <https://docs.unity3d.com/Manual>

An dieser Stelle folgt nun aber ein kompakter Überblick über die zur Verfügung stehenden Parameter. Wichtig: Die meisten können Sie nur editieren, wenn der Bereich **Mixed Lighting** über die entsprechende Checkbox aktiviert ist.

- **Lightmapper** legt fest, welches System zur Lichtberechnung benutzt wird. Sie haben hier die Wahl zwischen dem „normalen“ System „**Enlighten**“ oder dem mit Unity 5.6 neu hinzugekommenen System „**Progressive**“. Letzteres hat eine andere Herangehensweise an die Berechnung und zeigt kontinuierliche Updates währenddessen an. Das bedeutet, dass Sie während der aufwendigen Lichtberechnung schon sehen, wie in etwa das Ergebnis aussehen wird. Mit fortschreitender Zeit wird das Ergebnis hier immer besser. Haben Sie **Enlighten** gewählt, müssen Sie den gesamten Berechnungsprozess abwarten, um ein Ergebnis zu sehen.
- **Indirect Resolution** legt die Auflösung und damit die Genauigkeit fest, mit der indirektes Licht berechnet wird. Höhere Werte bedeuten außerdem längere Rechenzeiten.
- **Lightmap Resolution** legt die Auflösung und damit die Genauigkeit fest, mit der direkte, globale Beleuchtung berechnet wird. Höhere Werte bedeuten auch hier längere Rechenzeiten.
- **Lightmap Padding** kontrolliert den Abstand zwischen vorberechneten *Lightmap-Patches* (mehrere kleine *Lightmaps* innerhalb einer großen Textur).
- **Lightmap Size** legt fest, wie groß eine *Lightmap* maximal sein darf. Die Angabe ist in Pixeln.
- **Compress Lightmaps** ist standardmäßig aktiviert und sorgt dafür, dass die berechneten *Lightmaps* komprimiert werden, um weniger Speicherplatz zu benötigen. Dies kann jedoch auch zu ungewünschten Artefakten führen.
- **Ambient Occlusion** kontrolliert den Lichteinfluss in verdeckten Stellen wie z.B. Innenecken oder Selbstbeschattung bei nahe aneinander liegenden Objekten.
- **Max Distance** bezieht sich auf die *Ambient Occlusion* und legt fest, wie weit die Strahlen geschickt werden, die zu deren Berechnung vonnöten sind.
- **Indirect** und **Direct Contribution** kontrollieren jeweils den Kontrast der berechneten *Ambient Occlusion*; einmal für direktes und einmal für indirektes Licht.
- **Final Gather** ist ein Verfahren, das die Lichtberechnung im letzten Schritt noch einmal verbessern soll. Hierdurch können die Berechnungszeiten stark erhöht werden, das Ergebnis wird aber vor allem beim indirekten Licht besser sein.

- **RayCount** bezieht sich auf das *Final Gather*-System und kontrolliert, wie viele Strahlen zur Berechnung benutzt werden.
- **Denoising** legt fest, ob ein Entrauschungs-Filter über das Ergebnis der *Final Gather* Berechnung gelegt wird.
- **Directional Mode** bietet Ihnen die Optionen *Directional* und *Non-Directional*. Intern wird damit festgelegt, ob die berechneten *Lightmaps* (*baked* und *realtime*) Richtungs-Informationen von ihrer Haupt-Lichtquelle speichern oder nicht.

7.11 Light Explorer

Seit Unity 5.6 gibt es den sogenannten *Light Explorer*, den Sie unter dem Menüpunkt WINDOWS/LIGHTING/LIGHT EXPLORER finden. Dieses Fenster dient als Übersicht über verschiedene Daten, Objekte und Einstellungen, die Sie jetzt vereint in einem Editorfenster vorfinden.

Dies vereinfacht zum einen die Kontrolle über die von Ihnen getroffenen Einstellungen, zeigt damit viel schneller, wenn man mal etwas falsch eingestellt oder vergessen hat, und bietet eine komfortable Lösung, relevante Einstellungen an einem Platz statt verstreut über mehrere Orte zu finden.

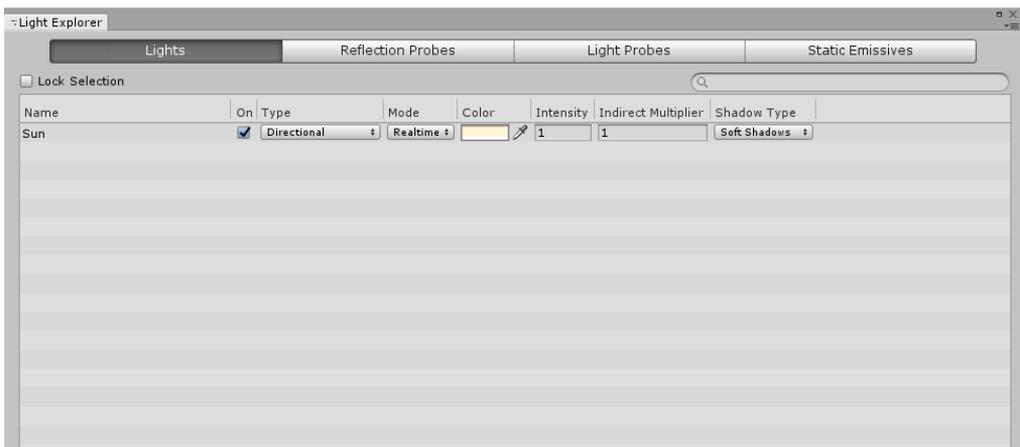


Bild 7.23 Der neue Light Explorer seit Unity 5.6

Der Übersichtlichkeit halber ist der *Light Explorer* in vier Bereiche *Lights*, *Reflection Probes*, *Light Probes* und *Static Emissives* aufgeteilt.

Je nach Bereich werden existierende Elemente aufgelistet und sind mit ihren Parametern, die Sie sonst im *Inspector* wiederfinden würden, aufgeführt.

■ 7.12 Reflexionen (Spiegelungen)

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

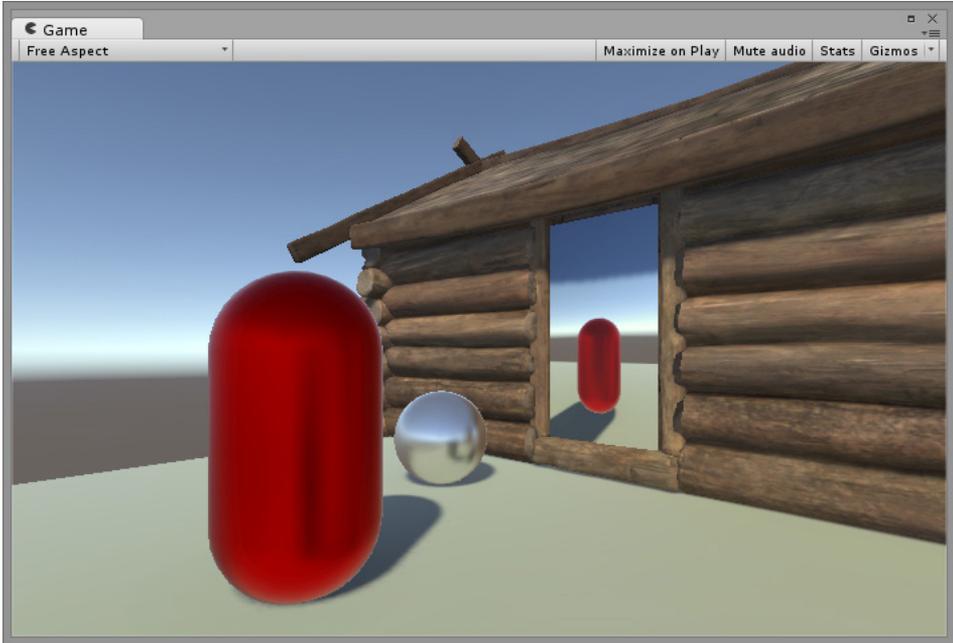


Bild 7.24 Szene mit einem Spiegel und anderen spiegelnden Objekten

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“ im Kapitel „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 weiter nichts 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 „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 *Reflections Probes* zum Einsatz, die wir im Folgenden etwas genauer erläutern.

7.12.1 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.

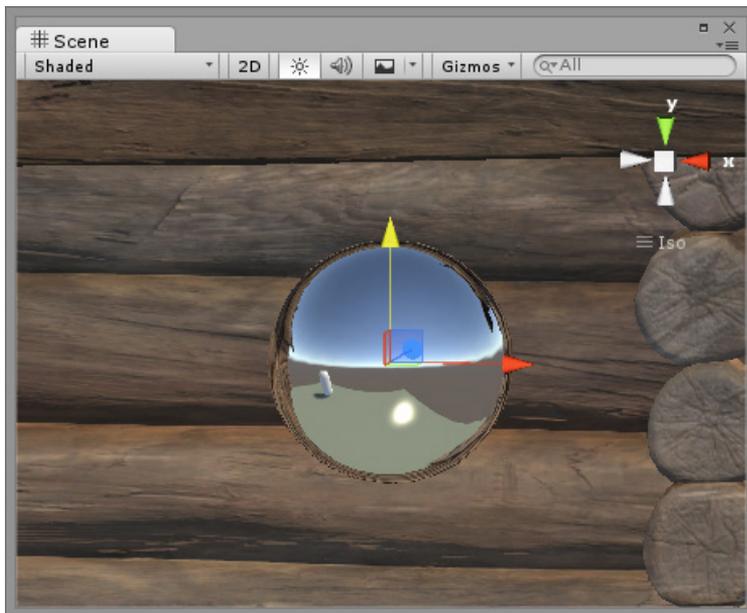


Bild 7.25 Reflection Probe

Reflection Probes sind kugelförmige Gebilde (siehe Bild 7.25), 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.

- **Baked** 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.

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. Bild 7.26 stellt es noch einmal bildlich dar.

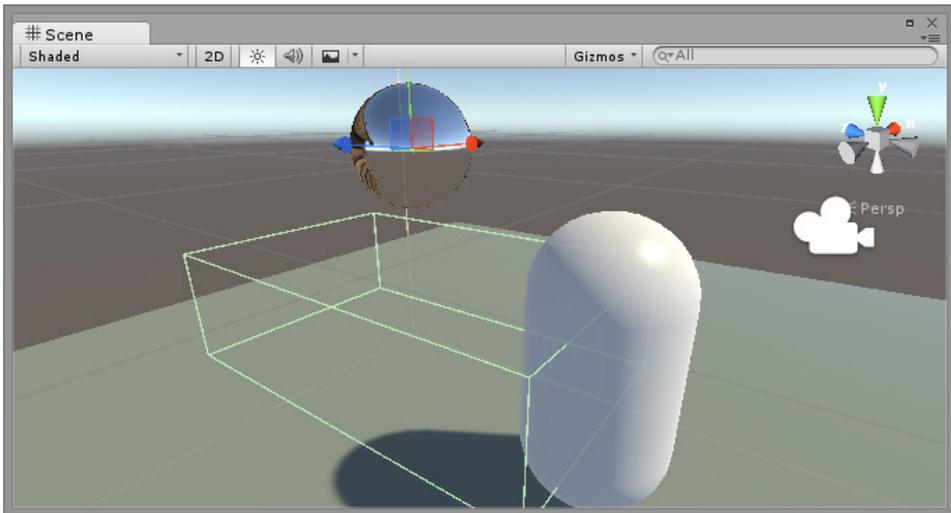


Bild 7.26 Per Trigger-Collider einen Reflection Probe aktualisieren

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.

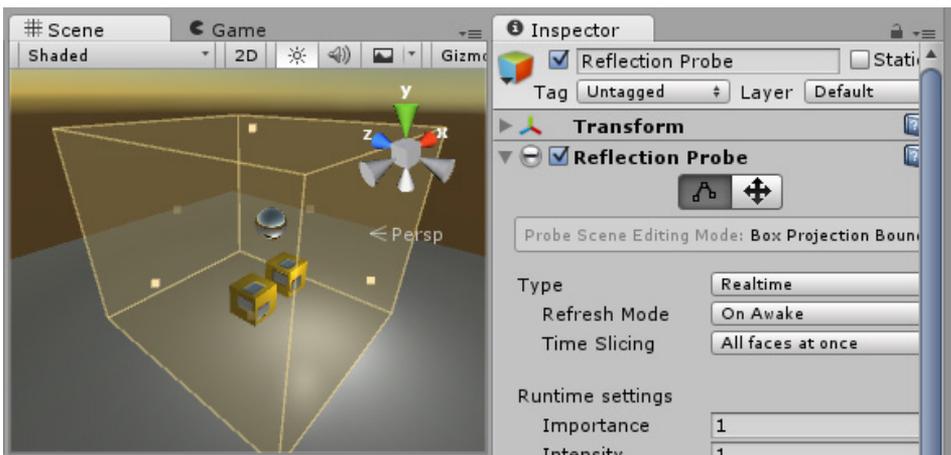


Bild 7.27 Reflection Probe-Box mit aktiviertem Size-Werkzeug

Mit dem *Size-Tool* können Sie nun diese Box modifizieren. Nutzen Sie hierfür die *Handle-Punkte* an den Flächen der Box (siehe Bild 7.27). 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.

■ 7.13 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.

7.13.1 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.

7.13.2 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.

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 „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 7.28.

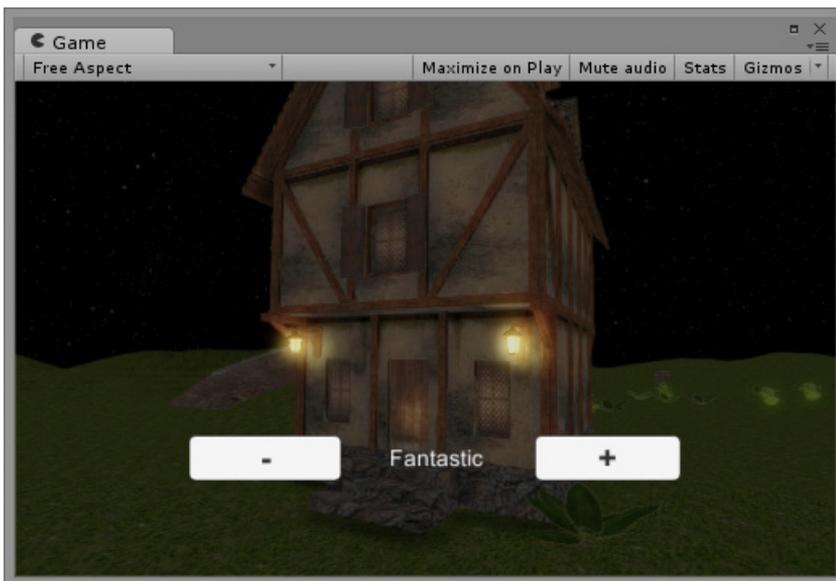


Bild 7.28 Einfache GUI zum Ändern der aktuellen Qualitätsstufe

Index

Symbole

2D 109, 148, 372, 423
2D-Button 10, 12
2D-Physik 263
3D-Koordinatensystem 109

A

A*-Algorithmus 461
Accelerometer 283
Accessoren 60
activeTerrain 363
Adaptive Reinhard 164
AddComponent 52, 89
AddExplosionForce 230
AddField 409
AddForce 229
AddForceAtPosition 230
Add Quality Level 222
AddRelativeForce 230
AddRelativeTorque 230
AddTorque 230
Advanced - Texture Type 139
Albedo 124
Allow Wet Mixing 301
Alpha Cutoff 123
Alpha-Kanal 138
Ambient Intensity 189
Ambient Light 189
Ambient Occlusion 215
Ambient Source 189
Anchor-Handle 379
Anchors 378
Angular Drag 226

Animation 393, 419, 431
Animation-Clip 431, 436
Animation Events 457f.
Animation State 436, 607
Animation Types 429
Animation View 421
Animator 450
Animator Controller 434, 607
Antialiasing 164, 212
anyKey 275
Any State 438
Application 98
Apply Root Motion 427
Area Effector 2D 266
Area Light 195, 205
Array 41 ff., 59 f., 69
Asset 8 f., 17, 20 ff., 26, 31 ff.
Asset Store 33, 145, 354, 365
Attribut 90
Audio 289
AudioClip 296
Audio-Filter 298
AudioListener 163, 289
Audio Mixer 290, 298
AudioSource 290
Ausführungsreihenfolge 103
Avatar 429
Avatar Definition 431
Avatar Mask 440
Awake 82

B

Background 161
 Backup 530
 Baked GI 214
 Baken 205
 Baking 191
 Balancing 639
 Basisklasse 62, 76
 Batch 483
 Bedingte Operatoren 47
 BeginHorizontal 400
 BeginVertical 400
 Behaviour 78
 Beschleunigungssensor 283
 Betrag 111
 Bit-Feld 241
 Blending 440
 Blend Tree 438
 Blueprint 377
 Bone-Mapping 429
 Boo 75
 Box Collider 232
 Brush 349
 Build-In-Shader 120
 Button 387, 397
 Byte Order 352

C

C# 37, 75
 Camel Case 39
 Camera 161, 163
 CancellInvoke 94
 Canvas 372
 Canvas Scaler 375, 523
 Capsule 143
 Capsule Collider 232
 centerOfMass 228
 Character Controller 258, 263, 573
 Clear Flags 161
 Clipping Planes 162, 185
 Cluster 360
 Collider 88, 232
 Collision 236
 Collision2D 265
 Collision Detection 227, 238
 CollisionEvent 331
 Component 18, 27, 77, 84, 87

Conditions 436
 Console 8, 23, 74, 102
 ConstantForce 232
 Constant Physical Size 375
 Constant Pixel Size 375
 Constraints 227
 Continuous Baking 205
 Cookie 139
 Coroutine 92, 407
 Create from Grayscale 140
 CreateInstance 77
 CREATE TABLE 411
 Create Tree Collider 357
 CrossPlatformInput 282
 CSV 412
 Cube 143
 Cubemap 199
 Culling Mask 161, 191
 Cursor 138, 278
 CursorMode 279
 Curves 422
 Cutout - Rendering Mode 123
 Cylinder 143

D

Datenbank 407, 410
 Datenbanksprache 410
 Datentypen 40
 Debug 102
 Default Cursor 555
 Default Time 104
 Deferred Lighting 212
 Delete 396
 deltaTime 81
 Depth 162, 176
 Destroy 77, 86, 89
 Detail Albedo x2 131
 Detail Mask 130
 Detail-Texturen 131
 Dictionary 70, 542
 diffuse Reflexion 134
 Directional Light 192
 DontDestroyOnLoad 77, 100 f.
 DontRequireReceiver 88
 Dope Sheet 422
 do-Schleife 51
 Drag 226
 Draw Call 483
 Ducking 305

E

Edit in Playmode 302
Editor 102
Editor Extensions 33, 106, 145
Eigenschaftsmethode 60
Eltern-Objekt 118
Emission 129
Empty GameObject 28, 117, 252
enableEmission 330
EndHorizontal 400
EndVertical 400
Enumeration 43
Environment Lighting 189
eulerAngles 119, 261
Event 389, 434
Event Camera 375
Event-Delegates 387
Event-Methoden 80
EventSystem 372
Event Trigger 394

F

Fade - Rendering Mode 123
Farbe 134
FBX 144, 428
Field of View 162
Filter Mode 150
Find 85f.
FindGameObjectsWithTag 85
FindWithTag 82, 85
First Person Controller 261
fixedDeltaTime 225
Fixed Joint 255
Fixed Timestep 225f., 238, 254
FixedUpdate 81, 225, 229
Flare Layer 163
Flatten 350
fontSize 396
ForceMode 231
foreach-Schleife 50
for-Schleife 49
Forward Friction 244
Forward Rendering 210
Frame 34, 81, 92
Frustrum Culling 185

G

Game Controller 560
Game-Design-Dokument 638
GameObject 8 ff., 17ff., 27ff., 34f., 76f., 82, 84ff., 89, 98
Game View 8, 12f., 15, 35
Gamma-Korrektur 166
Gamma-Rendering 166
Gamma-Space 168
Generate Colliders 455
Generate Root Motion Curves 426
Generic - Animation Type 429
gerichtete Reflexion 133
GET 409
GetAxis 273
GetButton 274
GetButtonDown 261
GetCollisionEvents 331
GetComponent 87
GetCurrentAnimatorStateInfo 454
GetKey 275
GetMouseButton 276f.
GetMouseButtonDown 231
GetQualityLevel 363
GetTouch 281
GetWorldPose 247
Gizmo 11ff., 290
Gizmo Display Toggles 15
Graphical User Interface 371
Grass Lighting 359
Gravitation 226
Gravity Scale 264
Grayscale 126, 139, 198, 351
GUI 371
GUI-Klasse 397
GUILayout 163
GUILayout-Klasse 399
GUISkin 400
GUIStyle 400

H

Hand-Tool 14
Hard Shadows 196
Has Exit Time 437
Hash-Wert 454
HDR-Rendering 164
Heightmap 126, 139, 351f.
Hierarchy 8, 11, 16ff., 34f.

High Dynamic Range-Rendering 164
 Highscore 410
 Hinge Joint 256
 HTML 409
 Humanoid - Animation Type 429

I

Icon 18
 identity 120
 IEnumerator 93
 if-Anweisung 46
 Image 383, 392
 Image Effects 179
 Immediate Mode GUI (IMGUI) 396
 Initialisierung 40
 Input 231, 273, 280
 InputField 389
 Input-Manager 269, 272, 286
 Input Settings 263
 INSERT INTO 411
 insideUnitSphere 92
 Inspector 17, 61, 84
 Instantiate 120, 404
 Instanzieren 41, 52
 Instanzmember 58
 Interface 65
 Interpolate 227
 Intersection 331
 Inventarsystem 542
 Invoke 94
 InvokeRepeat 94
 IsInvoking 94
 Is Kinematic 226, 236, 285
 Is Trigger 236

J

JavaScript 75
 Joints 255
 Joystick 272

K

Kamera 161
 KeyCode 275
 Keyframe 421
 KI 461, 617

Kind-Objekt 118
 Klasse 51, 61, 73, 78
 Klassendiagramm 76
 Klassenmember 57
 Klick-Sound 388
 Kollisionen 88, 232
 Kollisionserkennung 226, 232, 237
 Kompilierungsreihenfolge 102
 Komponente 17, 20, 27, 51f., 76, 78, 84, 87, 89
 Konsole 102
 Konstanten 43
 Künstliche Intelligenz 461, 604, 617

L

Label 397
 LateUpdate 83, 92
 Layer 16, 19, 30
 Layer-Based Collision Detection 239
 Layer Collision Matrix 239
 LayerMask 240, 582
 LDR-Rendering 164
 Lebensverwaltung 562
 Lens Flares 201
 Level Index 98, 100
 Level Of Detail 146
 Light 189
 Light Cookies 198
 Light Halos 200
 Lighting 205
 Lighting-Fenster 181, 189
 Lightmap 139, 205
 Lightmapping 189, 195, 204
 Lightmap Snapshot 205
 Lightmap Static 204
 Light Probe 207
 Light Probe Group 207
 Linear-Rendering 166
 Linear-Space 168
 linkshändiges Koordinatensystem 109
 List 69
 localEulerAngles 119
 localPosition 119
 localRotation 119
 LOD 146
 LODGroup 147
 LookAt 119
 loop match 433
 Low Dynamic Range 163

M

magnitude 111
Main Camera 161, 171, 290
Main Maps 124
Manual 9
Mask 434
Mass 226
Masseschwerpunkt 228
Mass Place Trees 357
Material 120
Mathf 58
Mehrspieler-Games 286
Mesh 113, 232
Mesh Collider 232, 234, 239
MeshFilter 115
MeshRenderer 115, 197
Metallic 124
Metallic-Workflow 133, 135
Methode 53
Minimap 175
Mipmap 150
Missing (MonoBehaviour) 79
MoCap 420
Model Import Settings 145
MonoBehaviour 52, 61, 76, 78, 80, 98
MonoDevelop 2, 73 f., 79
Mono-Framework 37
Motion Capturing 420
mousePosition 277
Move 259
Muscle 430
mySQL 410
mysql_query 411

N

nameHash 455
Namespace 67, 76, 80
Navigation 386
Navigation Area 468
NavigationMesh 462, 465
NavMesh 465, 609
NavMeshAgent 462, 606
NavMeshObstacle 469, 610
Negationsoperator 89
Netzwerkspiele 287
Noise 285
Normale 114

Normalenvektor 114, 197
Normalisieren 112
normalized 112
Normalmap 125, 138 f.
null 87

O

Object 76 f.
Objektorientierte Programmiersprache 51
Objektorientierte Programmierung 61, 69
Occludee Static 186
Occluder Static 186
Occlusion 127
Occlusion Area 187
Occlusion Culling 184, 484
Off-Mesh Link 463, 467, 470
OnCollision 236
OnCollision2D 265
OnControllerColliderHit 260
OnDestroy 98
OnGUI 82, 396
OnLevelWasLoaded 100
OnMouseDown 230
OnMouseOver 231
OnParticleCollision 322, 331, 341, 345
OnTrigger2D 265
Opaque - Rendering Mode 123
Order in Layer 155
Orientierung 283
Orthogonal 12
Orthographic 24, 176
Ortsvektor 111
Other Settings 209
out 59

P

Packing Tag 153
Paint Details-Tool 358
Paint Height-Tool 350
Paint Texture-Tool 353
Panel 384
Parallax Scrolling 157
Parametermodifizierer 58
Parenting 17, 118, 170
parsen 413
Particle Effect Control 313
ParticleSystem 311, 367

Partikeleffekte 311
 Partikelsysteme 311, 368
 Pathfinding 461, 604
 Perspective 24
 Perspektivisch 12
 PHP 407, 409 f.
 phpMyAdmin 411
 Physically Based Shading 122, 166
 Physic Effectors 266
 Physic Materials 242, 254
 Physics 2D 263
 Physics Settings 239
 Physik-Engine 81, 225, 232
 Pixelkoordinaten 277
 Pixel Lighting 210
 Place Tree-Tool 356
 Plane 143
 PlayClipAtPoint 295, 345
 Play Controls 15
 PlayerPrefs 95 f., 98
 Play Mode 12, 153
 Point Effector 2D 267
 Point Light 193
 Polygon 113
 Polygonnetz 113
 POST 409
 Postprocessing-Effekte 179
 Precomputed Realtime GI 215
 Prefabs 86, 403
 Primitive Collider 232, 234, 238
 Primitives 142, 235
 Produktionsphasen 637
 Produktionsprozess 637
 Project Browser 8, 10, 16, 18 ff., 25 f., 31 f. 34
 Projection 162, 176
 Projector 202
 Projekt Browser 79
 Pro Standard Assets 103
 Prozedurale Mesh-Generierung 146

Q

Quad 143
 Quality Level 222
 Quality Settings 222, 363
 Quaternion 92, 119, 261
 Quelltext 38
 Quest 588
 Questgeber 590

R

Raise/Lower-Tool 349
 Random 91
 Raw Image 384
 Raycast 59, 112, 239, 256
 RaycastHit 257
 Raycasting 256
 Rect-Handle 377
 Rect-Tool 14, 377
 RectTransform 78, 376
 ref 59
 Reflection Probe 219
 Reflection Probe Static 220
 Reflection Source 218
 Remote-Profiling 487
 Rename 79
 Rendering Mode 123
 Rendering Path 162, 209
 Rendering-Statistik 187, 482
 Render Mode 191, 373
 RenderSettings 181
 Render Texture 177, 218
 RepeatButton 397
 Reset 117
 Reverb Zone 296
 RGB 138
 Richtungsvektor 111
 Rig 429
 Rigidbody 226, 285
 Root 118
 Root Motion 426, 433
 Rotate 119
 Rotate-Tool 14
 RotateTowards 261

S

Sandbox 407
 Scale-Tool 14
 Scale with Screen 375
 Scene Gizmo 11 f.
 Scene View 8, 10 f., 13 f., 16, 18, 34 f., 109, 235
 Schatten 196
 Schnittstellen 65
 Schriftgröße 395
 Screen 395
 ScreenPointToRay 172, 278
 Screen Space - Camera 374, 393

- Screen Space – Overlay 373
 - ScriptableObject 77, 105, 449
 - Script Execution Order Settings 103
 - Scripting Reference 3, 9, 74
 - Scrollbar 391
 - Secondary Maps 131
 - SELECT 412
 - Send Collision Messages 331
 - SendMessage 88
 - SendMessageOptions 88
 - SetActive 86
 - SetCursor 279
 - SetDestination 464
 - SetQualityLevel 363
 - Shader 120
 - Shader Calibration Scene 134
 - Shadow Type 191
 - Shuriken 311
 - Sichtbarkeit 57, 62
 - Sideways Friction 244
 - SimpleMove 258
 - Singleton 101
 - Size 162
 - SkinnedMeshRenderer 115
 - Skript 73, 78, 84, 102
 - Skybox 181
 - Slider 390
 - Smartphones 280
 - Smooth Height-Tool 351
 - Smoothness 125, 136 f.
 - Snapping 14
 - Snapping-Modus 377
 - Snap Settings 15
 - Snapshots 306
 - Soft Shadows 196
 - Sorting Layer 155
 - Source-Code 38
 - Spawn-Point 576
 - Specular 124
 - Specular-Workflow 133, 136
 - Sphere 143
 - Sphere Collider 232
 - Spherical Harmonics 210
 - Spiegel 178, 218
 - Split 413
 - Split-Screen 174, 286
 - Spot Light 194
 - sprachübergreifende Zugriff 103
 - Spring Joint 256
 - Sprite 15, 138, 149, 380, 383, 424
 - Sprite Animation 325, 423
 - Sprite Atlas 150, 152
 - Sprite Editor 151, 380, 424
 - Sprite-Element 150
 - Sprite Packer 152
 - SpriteRenderer 154
 - Sprite-Shader 157
 - Sprite Sheet 150
 - SQL 410
 - sqrMagnitude 111
 - Stand-alone 270
 - Stand-alone Input Module 372
 - Standard Assets 103
 - Standard-Shader 121, 122
 - Standard (Specular setup) 122
 - Start 82
 - State Machine 435
 - Static 18, 204
 - Static Collider 238
 - Stiffness 244
 - StringToHash 454
 - Subfenster 399
 - Sub-Quest 599
 - Sub-State Machine 441
 - Sun 181
 - Superglobals 409
 - switch-Anweisung 48
 - Syntax 38
 - Szene 26, 98
- ## T
- Tablets 280
 - Tabs 7
 - Tag 29, 85
 - Target Texture 162, 177 f.
 - Terrain 239, 347, 367
 - Terrain Collider 239, 348, 357
 - Terrain Settings 351 f.
 - Text 382, 392
 - TextArea 398
 - TextField 398
 - Texture 138
 - Texture Atlas 150
 - Texture Type 380, 424
 - Tiefpass-Filter 285
 - Tilling 132
 - Time 81, 225
 - Time Manager 225
 - Toggle 389
 - Toggle Group 389

Tonemapping 164
 Touch 280
 TouchCount 281
 Touch Input Module 372
 Transform 17, 78, 81, 117
 Transform-Tools 10f., 14f., 34, 117, 377
 Transition 385, 436
 Translate 119, 285
 Translate-Tool 14
 Transparent – Rendering Mode 123
 Tree 357, 367
 Triangle 113, 141
 Trigger 236
 Trigger-Collider 236, 292

U

uGUI 372
 UI Scale Modes 375
 UnityEngine 76, 80
 UnityPackage 31f.
 UnityScript 76
 Update 81, 92
 Upgrading 25
 Use Gravity 226
 Use Light Probes 207
 UV Mapping 141

V

var 41
 Vector3 110, 119
 Vektor 110
 Vererbung 61
 Vergleichsoperatoren 46
 Versiegeln 64
 Version Control System 530
 Versionsverwaltung 530
 Vertex Lighting 210, 359
 Vertex Lit 211
 Vertex-Snapping 15
 Vertices 113

View Port Rect 162, 174
 Views 307
 Virtuelle Achsen 269, 271
 Virtuelle Tasten 271

W

WaitForSeconds 93
 WASD-Tastensteuerung 261, 573
 Wasser 365
 Water 365
 WebGL 407
 Web-Player 631
 Webspaces 410
 Wegfindung 461
 Wet Mixing 301
 Wheel Collider 242
 Wheel Friction Curve 243, 254
 WheelHit 249
 while-Schleife 50
 Wind Zones 356, 365, 367
 World Space 375
 Wrap Mode 132
 WWWForm 409
 WWW-Klasse 407

X

XML 412

Y

yield 93, 407

Z

Zeilenumbrüche 382
 Zufallswerte 91
 Zugriffsmodifizierer 65, 87