Računarska grafika 2

13M111RG2

4 Unity

Uvod u Unity

- 2005, samo za OS X
  - David Helgason, Joachim Ante, Nicholas Francis
- Cilj: “demokratizacija” proizvodnje igara (pristupačno svima)
  - skup alata
  - jednostavna manipulacija sadržajem
  - brzo pravljenje prototipova
- Danas pokriva 21+ platformu
- C#, JS, Boo
- Razne vrste licenci

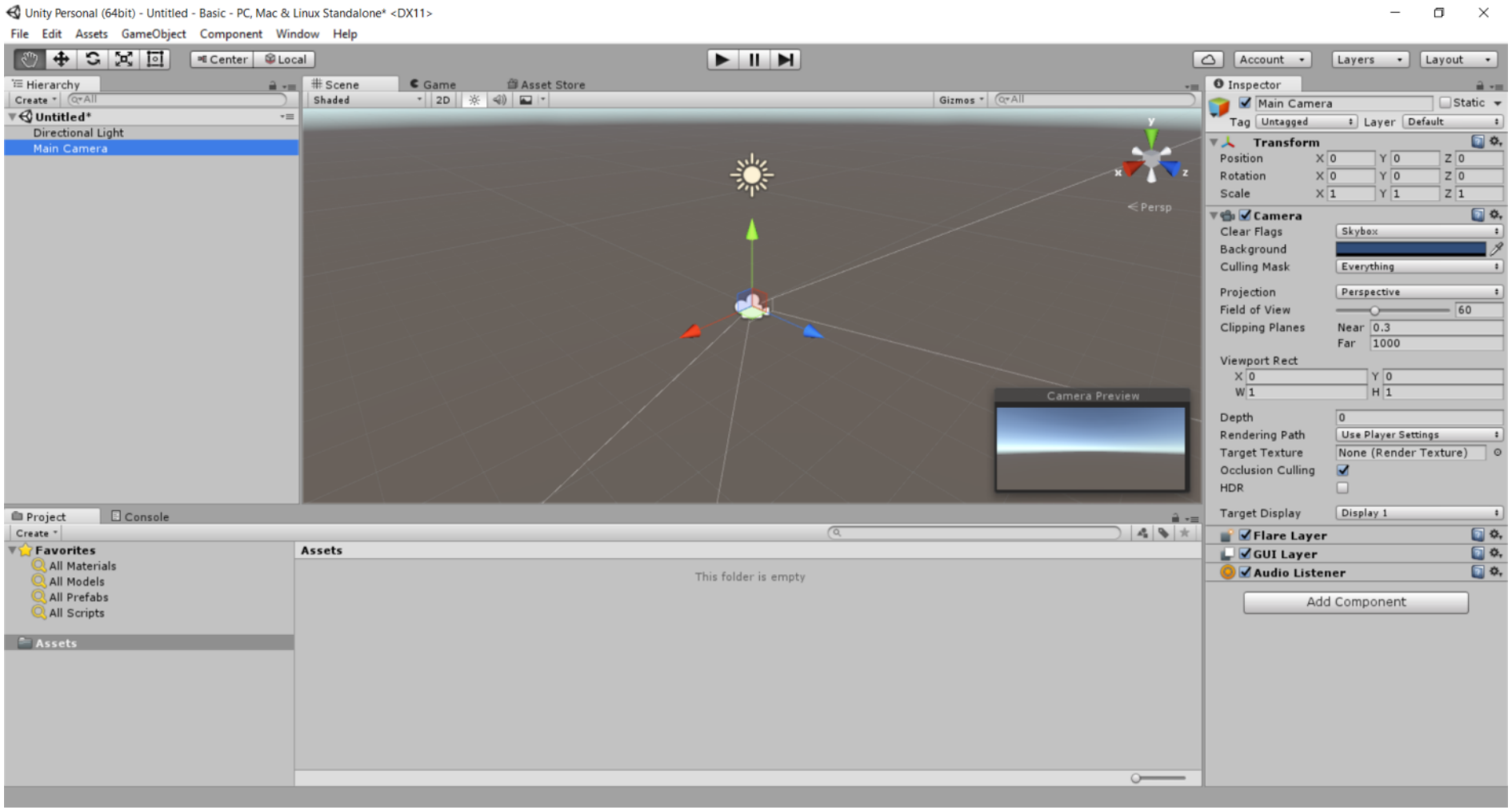
Uvod u Unity

- Editor
  - “drag’n’drop” pristup
  - više prozora

Scene	**Game**
Hierarchy	**Project**
Console	**Inspector**

  - sastavljanje scene
    - hijerarhijski uređena kolekcija instanci tipa `GameObject`
    - neophodna instanca tipa `Camera`
  - Player
    - “pokretanje” aktuelne scene
    - ne mora da se pravi izvršna datoteka

Uvod u Unity

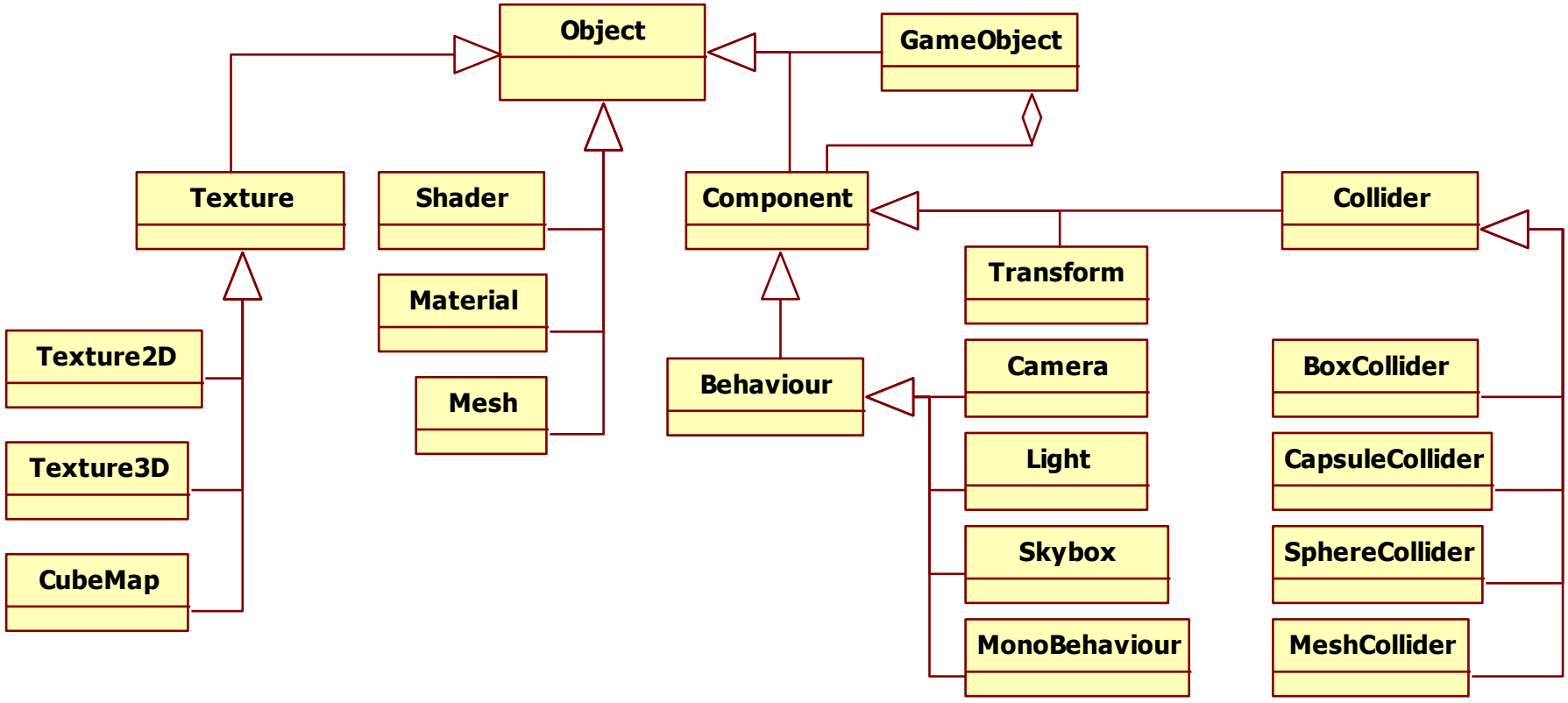

Unity Personal (64bit) - Untitled - Basic - PC, Mac & Linux Standalone* <DX11>
File Edit Assets GameObject Component Window Help
Center Local
Account Layers Layout
Hierarchy
Create
Untitled*
Directional Light
Main Camera
Scene Game Asset Store
Shaded 2D Gizmos
Persp
Camera Preview
Inspector
Main Camera
Static
Tag Untagged Layer Default
Transform
Position X 0 Y 0 Z 0
Rotation X 0 Y 0 Z 0
Scale X 1 Y 1 Z 1
Camera
Clear Flags Skybox
Background
Culling Mask Everything
Projection Perspective
Field of View 60
Clipping Planes Near 0.3 Far 1000
Viewport Rect
X 0 Y 0 W 1 H 1
Depth 0
Rendering Path Use Player Settings
Target Texture None (Render Texture)
Occlusion Culling
HDR
Target Display Display 1
Flare Layer
GUI Layer
Audio Listener
Add Component
Project Console
Create
Favorites
All Materials
All Models
All Prefabs
All Scripts
Assets
Assets
This folder is empty

Unity princip organizovanja scene

- Scena se sastoji od instanci `GameObject`
  - hijerarhijska struktura stabla
  - transformacija koju trpi roditelj se prenosi na potomke
- `GameObject` sadrži komponente
  - pozicija i orijentacija (`Transform`)
  - ponašanje (`Behaviour`)
  - crtanje (`Renderer`)
  - kolizija (`Collider`)
  - …
- `MonoBehaviour`
  - programski definisano ponašanje
  - korutine za simuliranje konkurentnog izvršavanja skripti

Unity hijerarhija klasa

- 200+ klasa



Camera

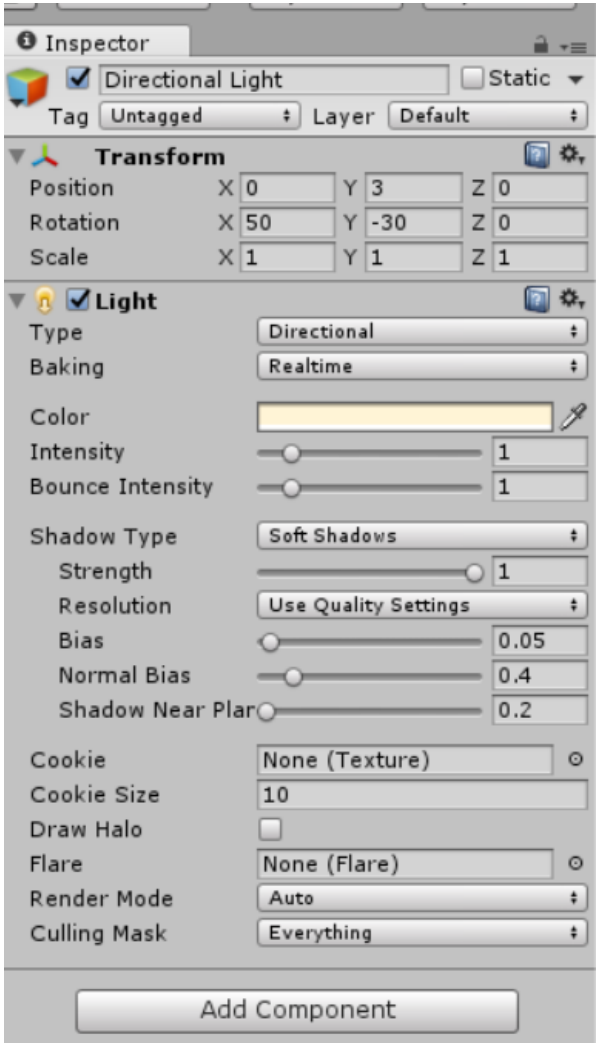
- Dodavanje kamere u scenu
- Stvara se GameObject sa komponentama
  - Transform
  - Camera
  - GUI Layer
  - Flare Layer
  - Audio Listener
- Inspector
  - Za kameru
  - ... i za druge ugrađene tipove
  - korisnik piše inspector za svoje tipove

Camera

- Clear Flags:
  - Skybox, Solid color, Depth only, Don’t clear
- Culling Mask:
  - Nothing, Everything, izabrani slojevi
- Projection:
  - Perspective, Orthographic
- Viewport
- Depth
  - redosled crtanja kada ima više kamera
- Rendering Path:
  - Player settings, Forward, Deferred
- Target Texture

Light

- Type:
  - Spot, Directional, Point, Area (baked only)
- Baking:
  - Realtime, Baked, Mixed
- Shadow type:
  - Soft shadows, Hard shadows, No shadows
- Cookie – za projekcione teksture
- Render Mode
  - Auto, Important, Not Important
- Culling Mask
  - isto kao i kod kamere



Programski definisano ponašanje

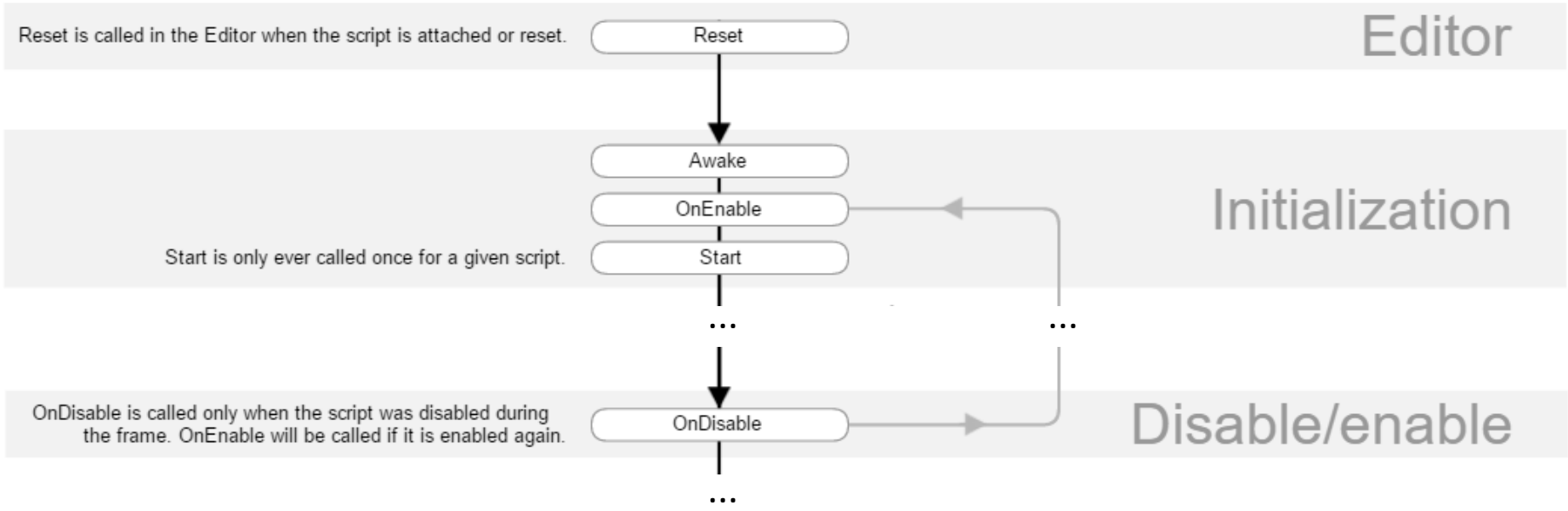
- Komponenta `MonoBehaviour`
- Komponenta se dodeljuje `GameObject`-u
  - definiše ponašanje objekta
- Svaka “skripta” mora nasledi ovu klasu
  - nema polimorfne metode, koristi se refleksija za dohvatanje metoda
- Objekat može imati proizvoljan broj skripti
  - mogu biti aktivne ili neaktivne
- Sistem poziva metode čime obaveštava o događajima
- Za primanje događaja se takođe mogu registrovati metode ne-MonoBehaviour klasa

Događaji (nekompletna lista)

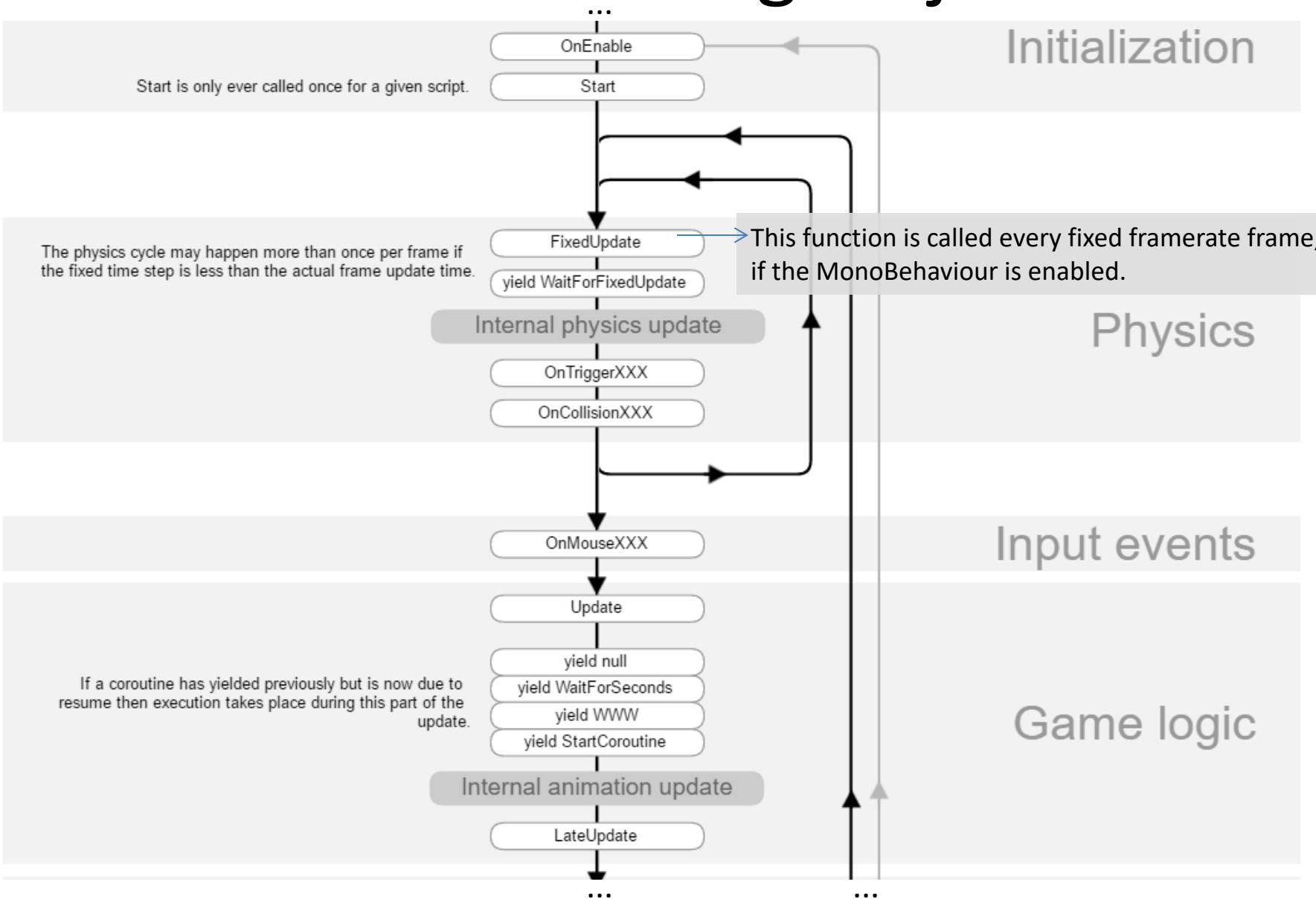
- Inicijalizacija
  - Reset (samo u Editoru)
  - Awake
  - OnEnable
  - ~~OnLevelWasLoaded~~
  - Start
- Ažuriranje
  - Update
  - LateUpdate
  - FixedUpdate
- Crtanje
  - OnWillRenderObject
  - OnPreRender
  - OnRenderObject
  - OnPostRender
- Oslobađanje
  - OnApplicationQuit
  - OnDisable
  - OnDestroy

Sekvenca događaja

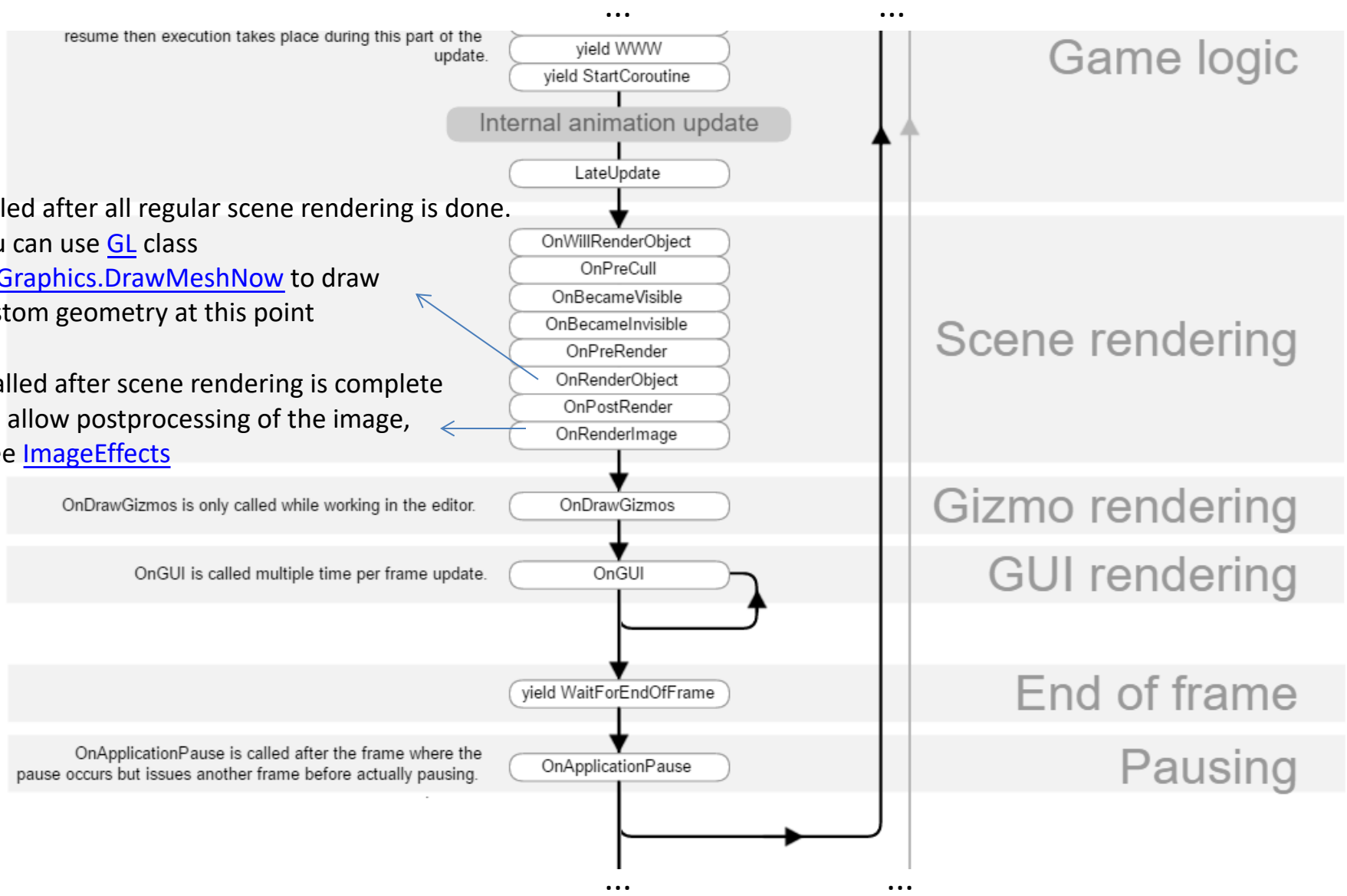
MonoBehaviour metode (i ostatak Unity sistema)
ne mogu biti pozvane iz konstruktora



Sekvenca događaja



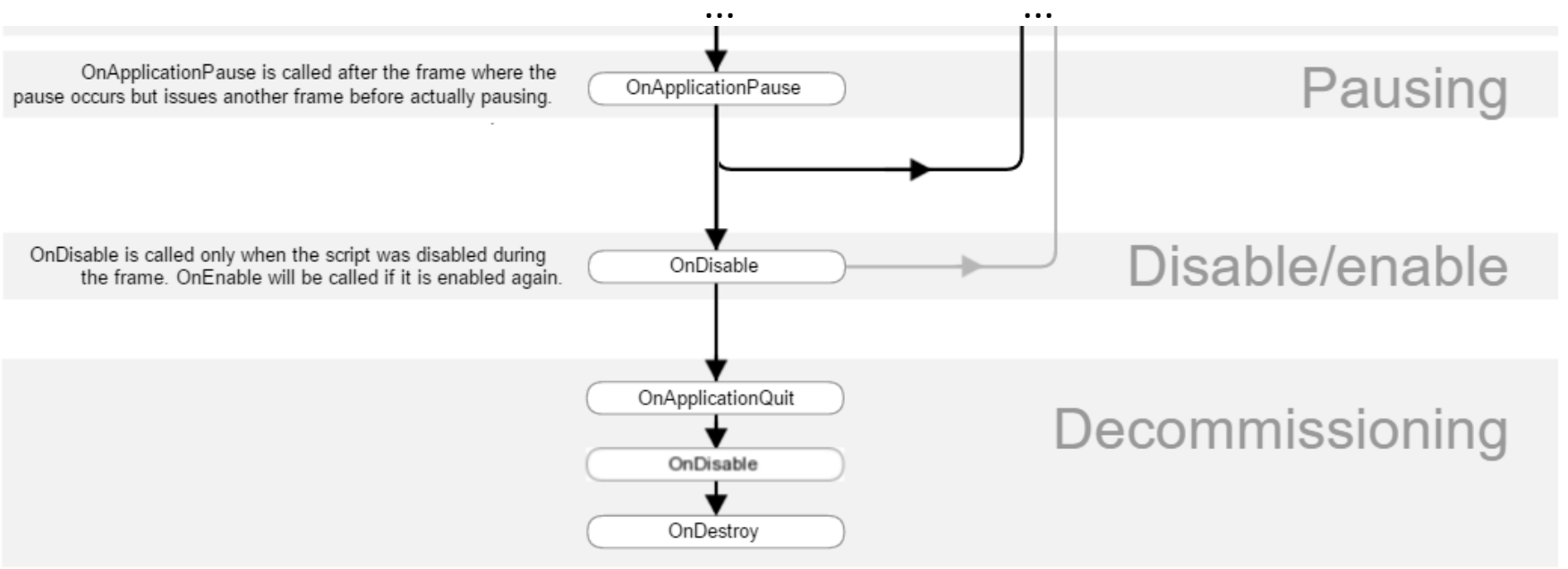
Sekvenca događaja



Called after all regular scene rendering is done. You can use GL class or Graphics.DrawMeshNow to draw custom geometry at this point

Called after scene rendering is complete to allow postprocessing of the image, see ImageEffects

Sekvenca događaja



Raspored fajlova

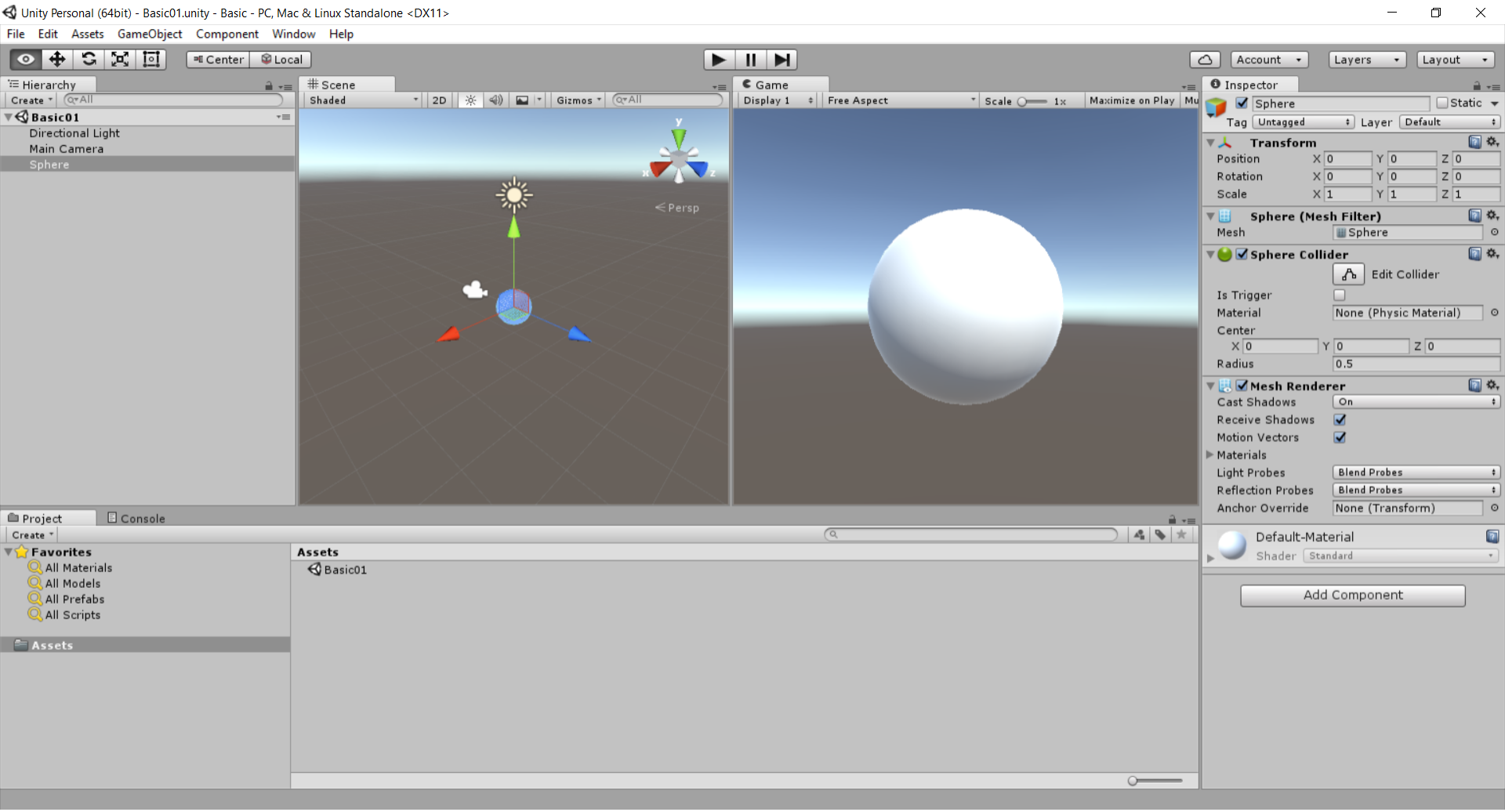
- Ne postoji “projektni” fajl – ceo projekat je u jednom dir.
- Assets – sav sadržaj (izvorni kod, teksture, ...)
- Library – keš
  - Ponekad (prelazak na novu verziju) dolazi do greške
  - i neophodno je obrisati ovaj dir.
- ProjectSettings
- Temp – privremeni *build* fajlovi

Raspored fajlova

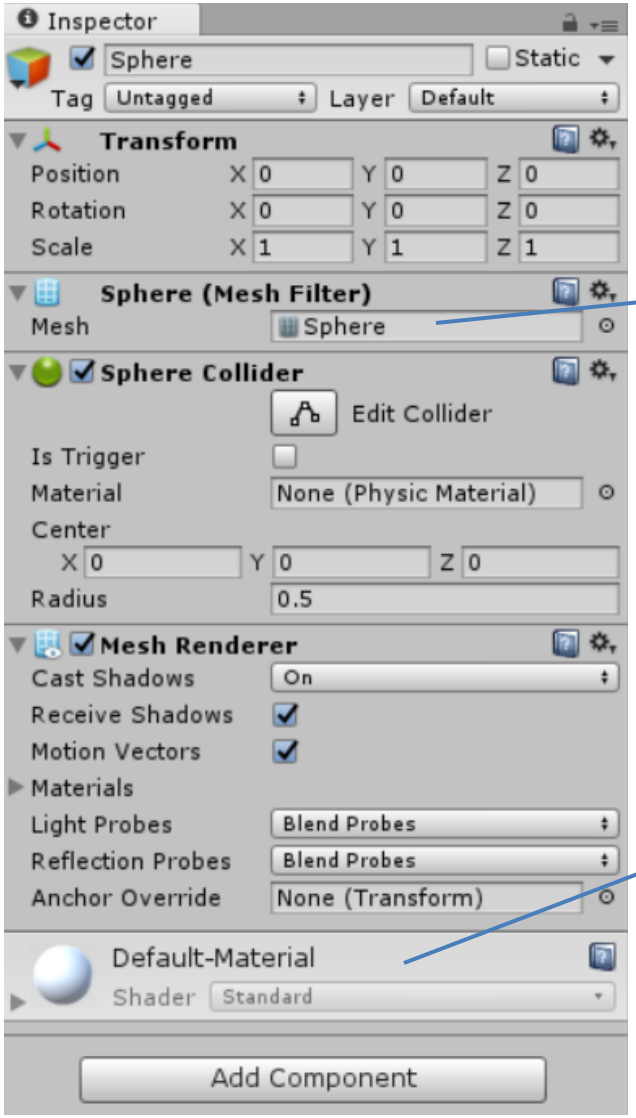
- Assets/…/Editor/
  - Sadržaj koji može biti upotrebljen samo u editoru
  - Ostatak koda (van Editor dir.) nema pristupa ovom sadržaju
- Assets/.../Resources/
  - Sadržaj koji će biti ugrađen u izvršni fajl
  - Može se dohvatiti sa Resources.Load

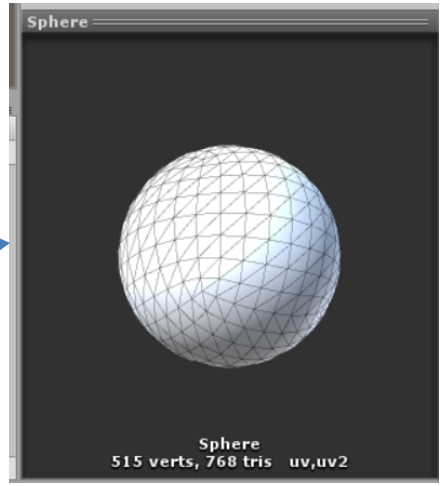
```
using UnityEngine;
using System.Collections;

public class ExampleClass : MonoBehaviour {
  void Start() {
    GameObject go = GameObject.CreatePrimitive(PrimitiveType.Plane);
    Renderer rend = go.GetComponent<Renderer>();
    rend.material.mainTexture = Resources.Load("glass") as Texture;
  }
}
```

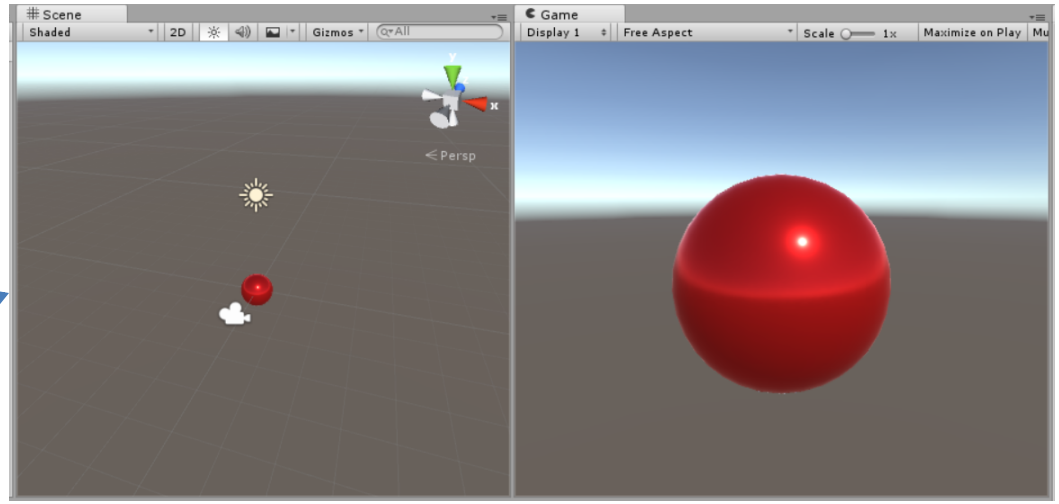
Rad sa objektima



Rad sa objektima - inspektor



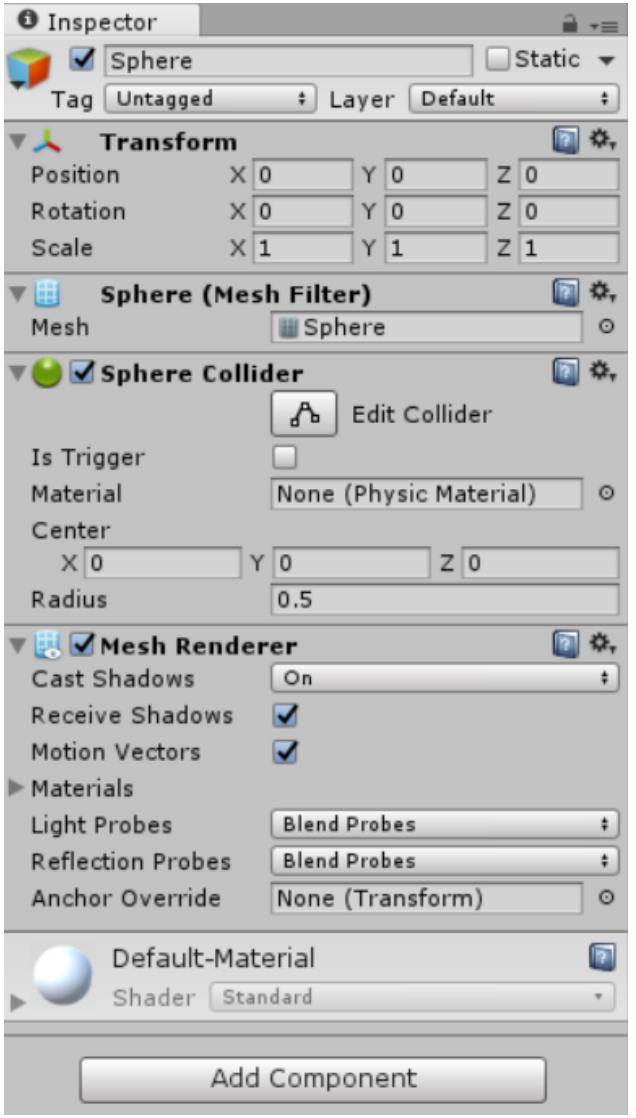




Rad sa objektima
dodavanje ponašanja

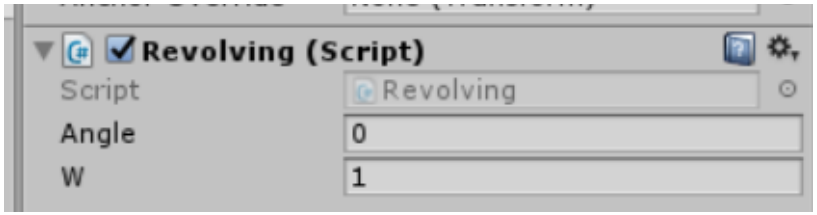
- Add Component
  - C# - daje se ime komponente
  - Generiše se kod za klasu datog imena, izvedenu iz MonoBehaviour, sa praznim telom metoda Start() i Update()

- Može i drag&drop sa postojećom C# skriptom



Rad sa objektima
Dodavanje ponašanja

```
using UnityEngine;
using System.Collections;
public class Revolving : MonoBehaviour
{
    [SerializeField]
    private float angle = 0;
    [SerializeField]
    private float w = 1;

    private Vector3 ComputePosition()
    {
        return new Vector3(Mathf.Cos(angle), 0, Mathf.Sin(angle));
    }

    // Use this for initialization
    void Start ()  { }

    // Update is called once per frame
    void Update ()
    {
        angle += w * Time.deltaTime;
        gameObject.transform.position = ComputePosition();
    }
}
```



Rad sa objektima
Dodavanje ponašanja

Serijalizacija polja:

- čuvanje vrednosti atributa prilikom snimanja scene
- restauracija prilikom učitavanja scene

Serijalizacija se automatski vrši pre ulaska u Play mod
Deserijalizacija se automatski vrši nakon izlaska iz Play moda

Sva **javna** polja `MonoBehaviour` objekata: podrazumevano serijalizuju

Sva ne javna polja koja su anotirana atributom `SerializeField`
se serijalizuju

Instanciranje objekata

- Svaki novi `GameObject` se automatski dodaje u aktivnu (tekuću) scenu

```
using UnityEngine;
using System.Collections;

public class Instantiate : MonoBehaviour
{
	public Material material;
	// Use this for initialization
	void Start ()
	{
		GameObject sphere = GameObject.CreatePrimitive(PrimitiveType.Sphere);
		sphere.name = "Instantiated sphere";
		sphere.AddComponent<Revolving>();
		sphere.GetComponent<MeshRenderer>().material = material;
	}
}
```

Stvaranje hijerarhije objekata

- Svaki `GameObject` podraz. ima komponentu **`Transform`**
- Hijerarhija se ostvaruje povezivanjem `Transform` komponenti relacijom potomak -> roditelj
- Zbog hijerarhijskog odnosa, postoji podatak o poziciji u k.s. sveta i podatak o lokalnoj poziciji

Stvaranje hijerarhije objekata

```
using UnityEngine;
using System.Collections;

public class ParentRevolving : MonoBehaviour {
  public float Angle { get; set; }
  public float W { get; set; }
  public float Radius { get; set; }

  private Vector3 ComputePosition()      { return new Vector3(
        Radius*Mathf.Cos(Angle), 0,      Radius*Mathf.Sin(Angle));        }

  void Awake() { Radius = 1; W = 1; }
  void Update () {
    Angle += W * Time.deltaTime;
    Vector3 parentPosition = Vector3.zero;
    if (gameObject.transform.parent != null)
        parentPosition = gameObject.transform.parent.position;
    gameObject.transform.position = parentPosition + ComputePosition();
  }
}
```

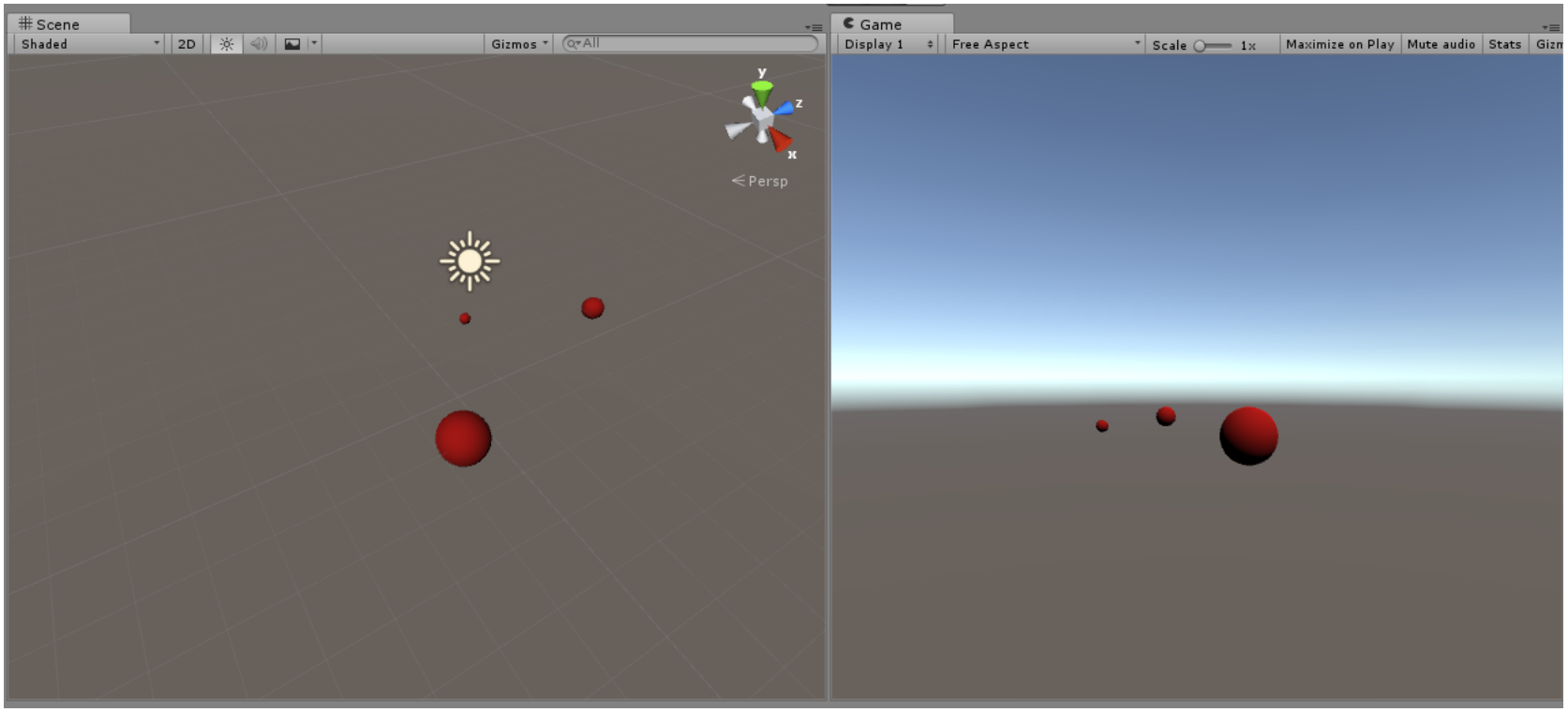
Stvaranje hijerarhije objekata

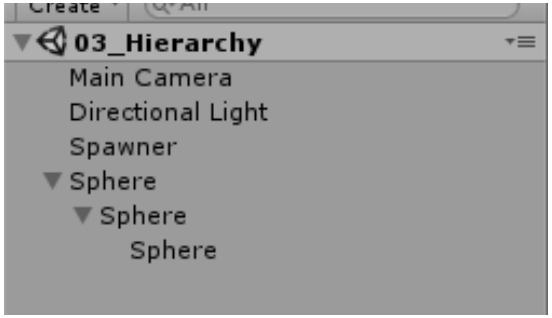
```
using UnityEngine;
using System.Collections;

public class InstantiateHierarchy : MonoBehaviour {
  public Material material;
  void Start () {
    GameObject mainSphere = GameObject.CreatePrimitive(PrimitiveType.Sphere);
    mainSphere.GetComponent<MeshRenderer>().material = material;
    mainSphere.AddComponent<ParentRevolving>();

    GameObject secondarySphere =
      GameObject.CreatePrimitive(PrimitiveType.Sphere);
    secondarySphere.GetComponent<MeshRenderer>().material = material;
    secondarySphere.transform.parent = mainSphere.transform;
    secondarySphere.AddComponent<ParentRevolving>();
    secondarySphere.GetComponent<ParentRevolving>().Radius = 5;
    secondarySphere.GetComponent<ParentRevolving>().W = 2.5f;
    secondarySphere.transform.localScale = new Vector3(0.5f, 0.5f, 0.5f);
```

Stvaranje hijerarhije objekata

```
    GameObject tertiarySphere =
      GameObject.CreatePrimitive(PrimitiveType.Sphere);
    tertiarySphere.GetComponent<MeshRenderer>().material = material;
    tertiarySphere.transform.parent = secondarySphere.transform;
    tertiarySphere.AddComponent<ParentRevolving>();
    tertiarySphere.GetComponent<ParentRevolving>().Radius = 3;
    tertiarySphere.GetComponent<ParentRevolving>().W = 4;
    tertiarySphere.transform.localScale = new Vector3(0.5f, 0.5f, 0.5f);
    }
}
```

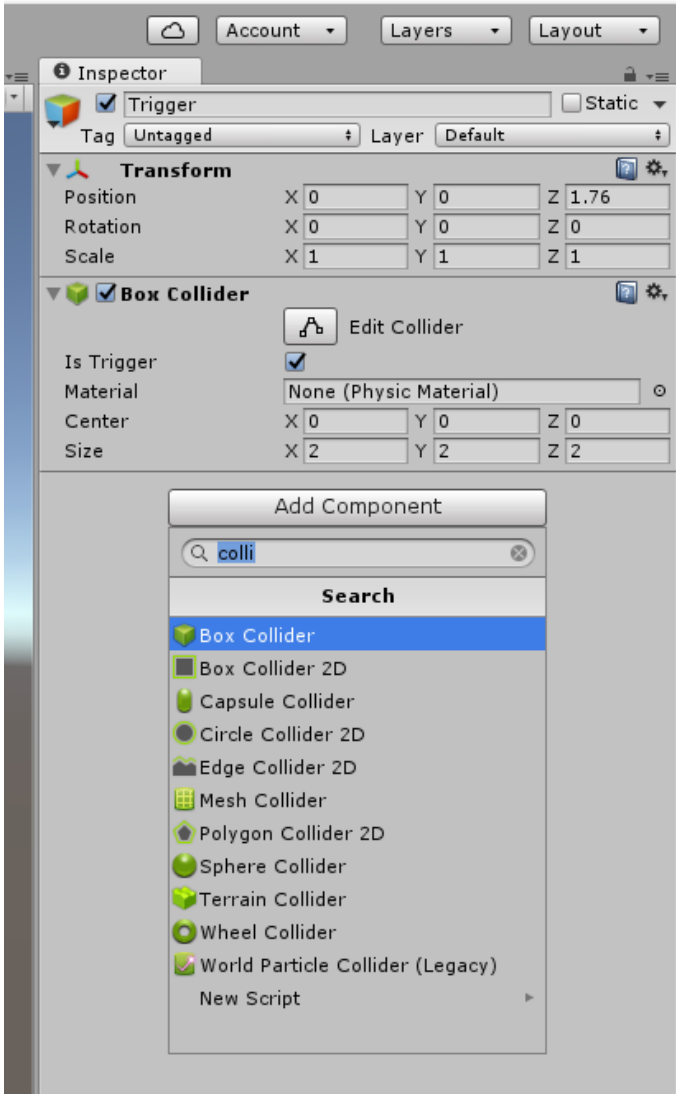
Stvaranje hijerarhije objekata





Kolizija – detekcija sudara

- Komponenta tipa Collider
  - BoxCollider (BoxCollider2D)
  - CapsuleCollider (CircleCollider2D)
  - EdgeCollider2D
  - MeshCollider
  - PolygonCollider2D
  - SphereCollider
  - TerrainCollider
  - WheelCollider



Kolizija – detekcija preseka

- Detekcija da je neki objekat sa komponentom tipa `Collider` u prostornom preseku sa zadatim `Collider-om`
- Jednostavan način detekcije ulaska, zadržavanja i izlaska iz zone
- Potrebno izabrati “Is Trigger”
  - Tada na dati `Collider` ne deluje simulirana fizika
- Potrebno objektu dodati MonoBehaviour skriptu:
  - `OnTriggerEnter`, `OnTriggerStay`, `OnTriggerExit`
- Najmanje jedan od objekata u koliziji mora imati RigidBody komponentu
- Kolizije se šalju čak i isključenim MonoBehaviour skriptama da bi se omogućilo “buđenje” u slučaju kolizije

Kolizija – detekcija preseka

```
using UnityEngine;

public class ChangeW : MonoBehaviour {

    void OnTriggerEnter(Collider other) {
          ParentRevolving pr = other.gameObject.GetComponent<ParentRevolving>();
                    if (pr != null)
                    pr.W = (5 * (Random.value + 0.5f))*(Random.value<0.5f?-1:1);
    }

    void OnTriggerStay(Collider other) {
          if (oldScale == null)
                    oldScale = other.gameObject.transform.localScale;
          other.gameObject.transform.localScale *= 0.95f;
    }

    void OnTriggerExit(Collider other) {
          if (oldScale != null) {
                    other.gameObject.transform.localScale = oldScale.Value;
                    oldScale = null;
          }
    }
}
```

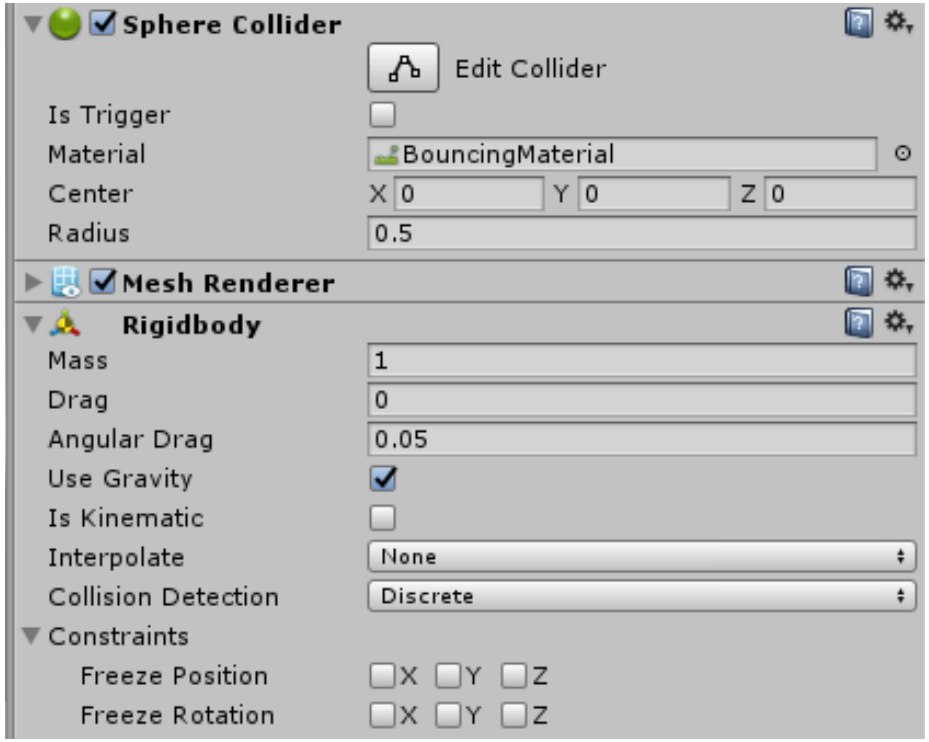
Kolizija krutih tela

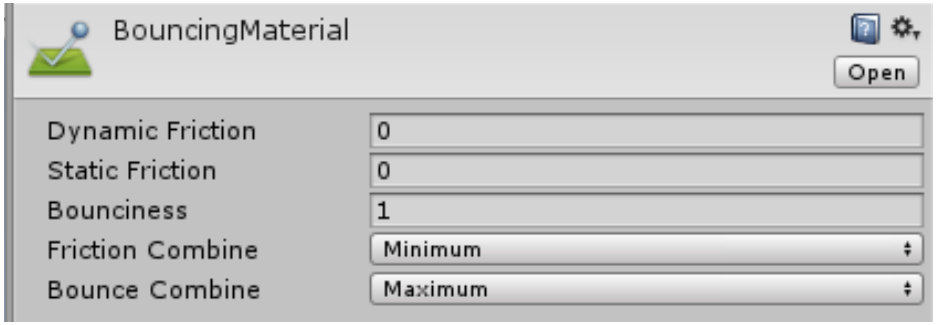
- Unity ima ugrađen mehanizam za simuliranje kretanja sistema objekata primenom zakona klasične mehanike
- Kruto telo → komponenta Rigidbody
  - brzina, masa, inercija
  - gravitacija
  - mogućnost redukcije broja stepeni slobode

```
void Start () {
  Rigidbody rb =
    gameObject.GetComponent<Rigidbody>();
  rb.velocity = new Vector3(5, 0, 0);
}
```

- Detekcija sudara između tela → komponenta `Collider`
  - Opis fizičkih svojstava materijala pri sudaru → `PhysicMaterial`
  - Trenje, elastičnost
  - Način kombinovanja svojstava materijala pri sudaru

Kolizija krutih tela

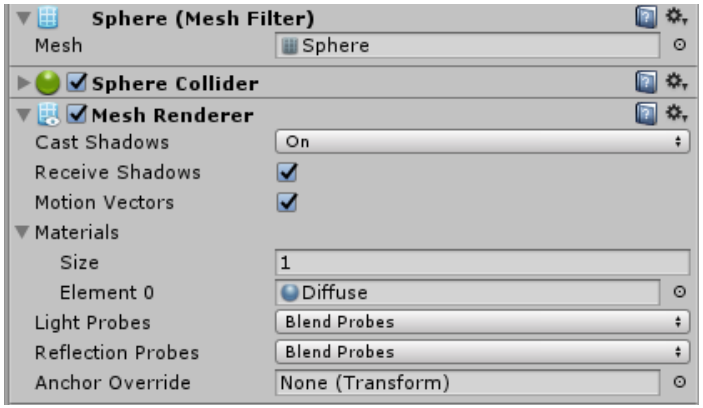




Kolizija krutih tela

- Reagovanje na koliziju
  - `OnCollisionEnter`, `OnCollisionStay`, `OnCollisionExit`
- Parametar je objekat tipa `Collision`
  - `Collider` sa kojim se dogodio sudar
  - skup kontaktnih tačaka prilikom kolizije
  - impuls, relativna brzina
  - ... drugi podaci relevantni za sudar i reagovanje na sudar

Proceduralno formiranje mreže trouglova



- `Mesh`
  - reprezentacija mreže trouglova
  - niz temena, niz indeksa (tri uzastopna)
  - ograničen broj atributa:
    - normala, 4 teksturne koordinate, boja, tangenta
  - Redosled dodavanja: prvo temena, posle indeksi
- Komponenta `MeshFilter`
  - `public Mesh mesh;`
  - `public Mesh sharedMesh;` [preporučeno **samo za čitanje**!]
- Komponenta `MeshRenderer`
  - prikazivanje mreže trouglova

Izmena postojeće mreže

```
using UnityEngine;
using System.Collections;

public class MeshSpikes : MonoBehaviour {

  void Start () {

    Mesh m = GetComponent<MeshFilter>().sharedMesh;
    Vector3[] vert = m.vertices;
    Vector3[] norm = m.normals;

    for (int i = 0; i < vert.Length / 10; i++)  {
      int index = (int)(Random.value * (vert.Length - 1));
      vert[index] += norm[index] * 0.5f;
    }

    GetComponent<MeshFilter>().mesh.vertices = vert;
  }

}
```

Stvaranje nove mreže

```
using UnityEngine;
using System.Collections;

public class ProceduralCube : MonoBehaviour {

  void Start () {

    Vector3[] vert = new Vector3[24];
    vert[0] = new Vector3(-1, -1, -1);
    vert[1] = new Vector3(1, -1, -1);
    vert[2] = new Vector3(1, 1, -1);
    vert[3] = new Vector3(-1, 1, -1);

    vert[4] = new Vector3(1, -1, -1);
    vert[5] = new Vector3(1, -1, 1);
    vert[6] = new Vector3(1, 1, 1);
    vert[7] = new Vector3(1, 1, -1);

    vert[8] = new Vector3(1, -1, 1);
    vert[9] = new Vector3(-1, -1, 1);
    vert[10] = new Vector3(-1, 1, 1);
    vert[11] = new Vector3(1, 1, 1);
```

Stvaranje nove mreže

```
vert[12] = new Vector3(-1, -1, 1);
vert[13] = new Vector3(-1, -1, -1);
vert[14] = new Vector3(-1, 1, -1);
vert[15] = new Vector3(-1, 1, 1);

vert[16] = new Vector3(-1, 1, -1);
vert[17] = new Vector3(1, 1, -1);
vert[18] = new Vector3(1, 1, 1);
vert[19] = new Vector3(-1, 1, 1);

vert[20] = new Vector3(-1, -1, 1);
vert[21] = new Vector3(1, -1, 1);
vert[22] = new Vector3(1, -1, -1);
vert[23] = new Vector3(-1, -1, -1);

int[] tris = {
          0, 2, 1, 0, 3, 2, 4, 6, 5, 4, 7, 6, 8, 10, 9, 8, 11, 10,
          12, 14, 13, 12, 15, 14, 16, 18, 17, 16, 19, 18,
          20, 22, 21, 20, 23, 22
          };
```

Stvaranje nove mreže

```
    Mesh mesh = new Mesh();
    mesh.vertices = vert;
    mesh.triangles = tris;

    mesh.RecalculateNormals();

    gameObject.AddComponent<MeshFilter>();
    MeshFilter mf = GetComponent<MeshFilter>();
    mf.mesh = mesh;

    gameObject.AddComponent<MeshRenderer>();
    Material diffuse = Resources.Load<Material>("Diffuse");
    GetComponent<MeshRenderer>().material = diffuse;
  }
}
```

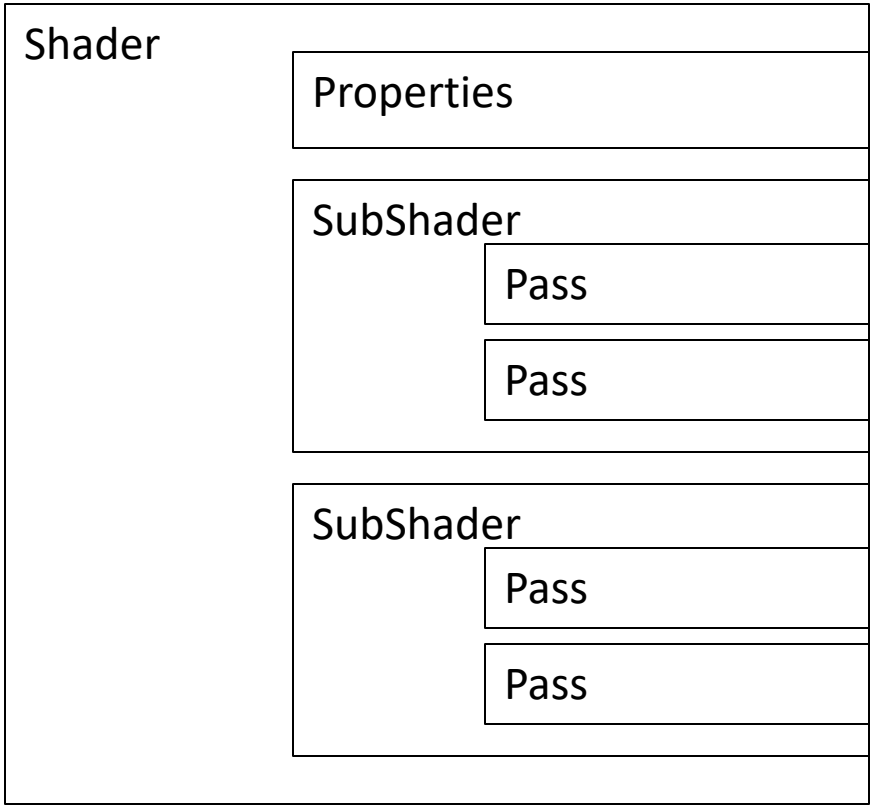
Material

- Klasa `Material`
  - apstrakcija materijala koji se primenjuje prilikom crtanja površi modela
  - suštinski, služi kao omotač za program za senčenje
    - postavljanje uniformnih vrednosti
    - postavljanje tekstura
- Jedan GameObject može imati više materijala
  - svi se, redom, primenjuju za crtanje mreže trouglova

Shader

- Klasa koja predstavlja apstrakciju programa za senčenje (VS+FS)
- Omotačka klasa, postupak prevođenja i aktiviranja potpuno sakriven od korisnika
- Dodeljuje se materijalu
  - ulazne podatke prikazuje inspektor materijala
- Može da se dodeli i kameri (`SetReplacementShader`) kada kamera ignoriše sve druge programe za senčenje
- Koristi se jezik Cg
  - NVidia, cross-platform (OpenGL, DirectX)
  - Specifičnosti zbog izlaganja ulaznih podataka prema Unity editoru

Shader – struktura programa



Shader
Surface shader

- Nije nova vrsta programa za senčenje
- Unity rešenje za programere koji ne pišu svoje VS + FS
- VS+FS su predefinisani
- SS popunjava strukturu SurfaceOutput

```
struct SurfaceOutputStandardSpecular {
        fixed3 Albedo; // diffuse color
        fixed3 Specular; // specular color
        fixed3 Normal; // tangent space normal, if written
        half3 Emission;
        half Smoothness; // 0=rough, 1=smooth
        half Occlusion; // occlusion (default 1)
        fixed Alpha; // alpha for transparencies
    };
```

Shader - sintaksa

```
Shader "name" {
   [Properties] Subshaders [Fallback] [CustomEditor] }
```

- https://docs.unity3d.com/Manual/SL-Shader.html

- Properties - podaci (teksture, boja, ...) vidljivi u inspektoru mat.
  - mogu podešavati i fiks. delove rend. pipel. – na primer Blend
  - vrednosti se čuvaju (serijalizuju) od strane Unity Editora
- Subshaders + Fallback
  - lista shader programa
  - najmanje 1
  - primenjuje se prvi koji može da radi na datom hardveru
- CustomEditor
  - naziv klase koju treba koristiti u Unity Editoru za prikazivanje ovog shader-a

Shader – sintaksa
Properties

```
Properties { Property [Property ...] }


name ("display name", Range (min, max)) = number
name ("display name", Float) = number
name ("display name", Int) = number

name ("display name", Color) = (number,number,number,number)
name ("display name", Vector) = (number,number,number,number)

name ("display name", 2D) = "defaulttexture" {}
name ("display name", Cube) = "defaulttexture" {}
name ("display name", 3D) = "defaulttexture" {}
```

```
"" "white" "black"
"gray" "bump" "red"
```

Shader – sintaksa
SubShader

```
Subshader { [Tags] [CommonState] Passdef [Passdef ...] }
```

- Definiše:
  - oznake (tags)
  - zajedničko stanje za sve prolaze (sintaksa ista kao za pojedinačne prolaze)
  - prolaze

Shader – sintaksa
Pass

```
Pass { [Name and Tags] [State] }
```

- Stanja:
  - Cull (Back | Front | Off)
  - ZTest (Less | Greater | LEqual | GEqual | Equal | NotEqual | Always)
  - ZWrite (On | Off)
  - Offset *OffsetFactor*, *OffsetUnits*
  - Blend *srcBlendMode dstBlendMode*, *aSr*cBlendMode *aDstBlendMode*
  - ColorMask (RGB | A | 0 | any combination of R, G, B, A)
- Vrste prolaza
  - "regularni"
  - UsePass – koristi postojeći prolaz nekog drugog prog. za senčenje
  - GrabPass – tekući sadržaj bafera slike čuva u teksturu dostupnu narednim prolazima

Shader – sintaksa
Primer

```
Shader "RG2/SimpleShader"
{
    Properties
    {
        _MainTex ("Texture", 2D) = "white" {}
    }
    SubShader
    {
        Tags { "RenderType"="Opaque" }
        LOD 100
        Pass
        {
            CGPROGRAM
            #pragma ...
            #include ...
                ...
            }
            ENDCG
        }
    }
}
```