Einführung in die Informatik für Games Engineering

Tutorials

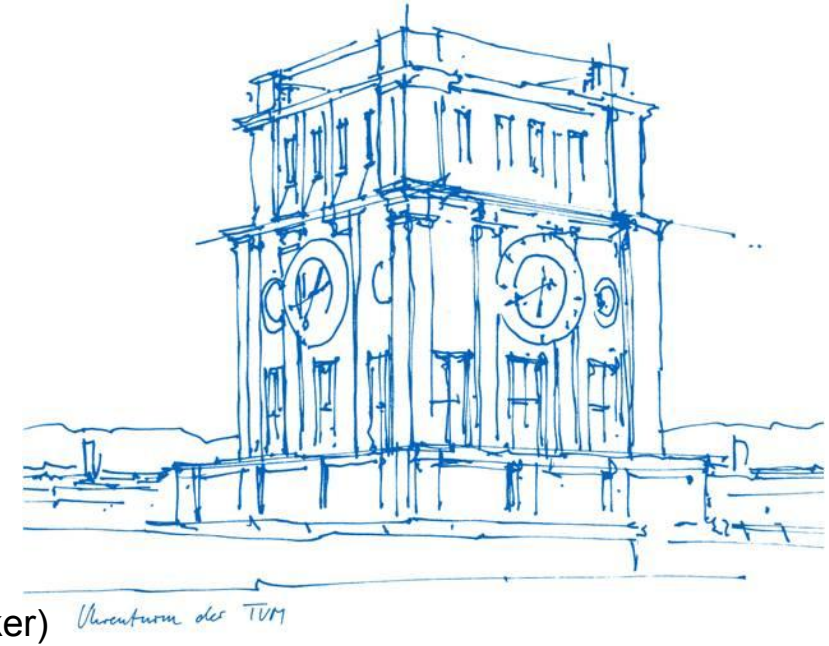
Gudrun Klinker (klinker@in.tum.de

Sven Liedtke (sven.liedtke@cit.tum.de)

Technical University of Munich

School of Computation, Information and Technology

Associate Professorship of Augmented Reality (Prof. Klinker)



Associate Professorship of Augmented Reality
School of Computation, Information and Technology
Technical University of Munich

What do you remember from last week?

Associate Professorship of Augmented Reality
School of Computation, Information and Technology
Technical University of Munich

What do you remember from last week?

- Coroutines
- Enums

Lessons Learned – Coroutines

Reminder:

- Normal functions execute everything at once
- Coroutines can pause (“yield”) at any time before continuing

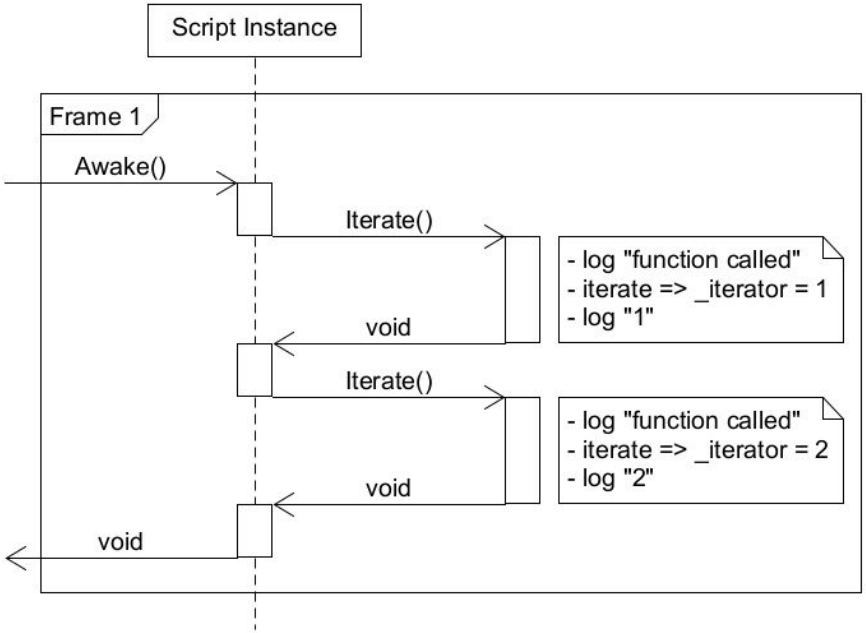
```
private void Awake()
{
    LogMessage();
}

private void LogMessage()
{
    Debug.Log("This is logged right after" +
              " this function has been called.");
}
```

```
private void Awake()
{
    StartCoroutine(LogMessage());
}

private IEnumerator LogMessage()
{
    yield return new WaitForSeconds(1);
    Debug.Log("This is logged 1 SECOND after" +
              " this function has been called.");
}
```

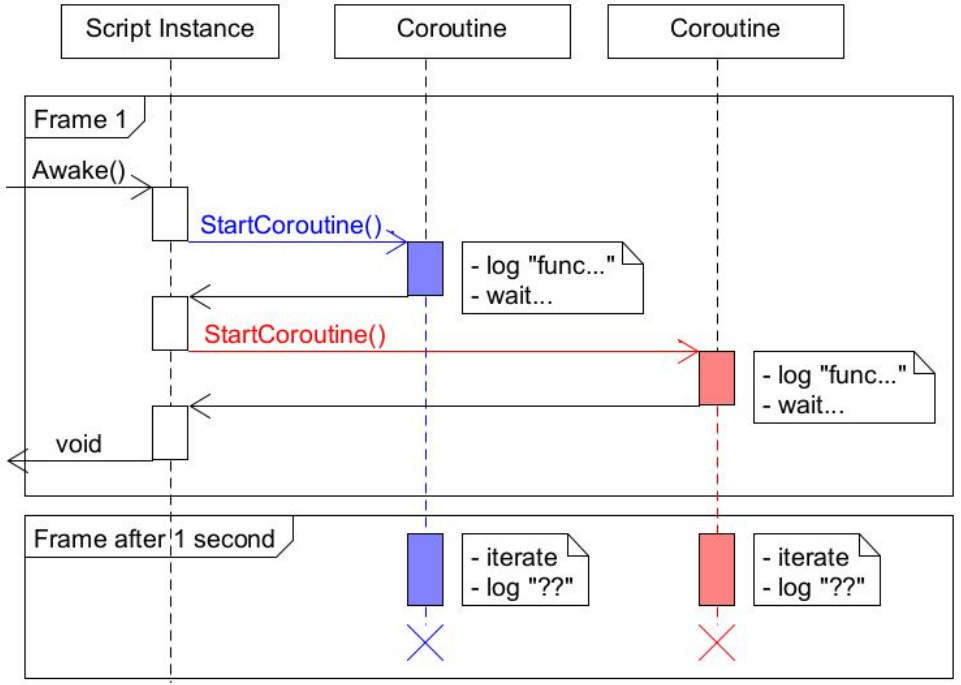
Lessons Learned – Coroutines

Visual example: Normal function call

```
private int _iterator;

private void Awake()
{
    Iterate();
    Iterate();
}

private void Iterate()
{
    Debug.Log("function called");
    _iterator++;
    Debug.Log(_iterator);
}
```



```
private int _iterator;

private void Awake()
{
    StartCoroutine(IterateAfterASec());
    StartCoroutine(IterateAfterASec());
}

private IEnumerator IterateAfterASec()
{
    Debug.Log("function called");
    yield return new WaitForSeconds(1);
    _iterator++;
    Debug.Log(_iterator);
}
```



Visual example: Coroutine call

- (graphic is heavily simplified)
- Notice how the coroutines' lifetimes last longer than the Awake's lifetime
- Also, they are not connected with their original caller after being called
- Beware of race conditions → unclear value of _iterator in each coroutine

Lessons Learned – Coroutines

- What happens if you start this coroutine from Update()?

```
private int _iterator;

private void Update()
{
    StartCoroutine(IterateAfterASec());
}

private IEnumerator IterateAfterASec()
{
    Debug.Log("function called");
    yield return new WaitForSeconds(1);
    _iterator++;
    Debug.Log(_iterator);
}
```

Lessons Learned – Coroutines

- What happens if you start this coroutine from Update()?

→ “function called” is logged every frame!

→ After 1 second: Logs the total frame count every frame, too!

→ There would be as many coroutines as FPS at the same time!

```
private int _iterator;

private void Update()
{
    StartCoroutine(IterateAfterASec());
}

private IEnumerator IterateAfterASec()
{
    Debug.Log("function called");
    yield return new WaitForSeconds(1);
    _iterator++;
    Debug.Log(_iterator);
}
```

Associate Professorship of Augmented Reality
School of Computation, Information and Technology
Technical University of Munich

Lessons Learned – Values and references

- Public variables can store primitive values or reference values
- Primitive values are, e.g. int, string, Vector3, etc… These types of variables have an initial value that you can also work with, e.g. 0, string.Empty, Vector3.zero
- Reference variables store references to objects, e.g. game objects, components, assets, etc... This reference MUST be set first to work with the object. Otherwise, the variable is empty (null) and there is no object to work with → error

```
public GameObject explosionPrefab; // <- needs to be set in the inspector!
public float speed;                // <- is 0
```

- Don’t use scene objects as template reference for instantiation! Use prefabs instead.

Additional Task (fre[illegible]k t[illegible]during tutorial or at home)

#1 Show your animations

Questions?

Today’s topics

- Evaluation
- Make it first person
- Adjust the features / revisit old topics

Evaluations

Please check your email with YOUR evaluation link

First Person View

Tasks #1

1) Discuss what to change if you want to fly with the view of a pilot

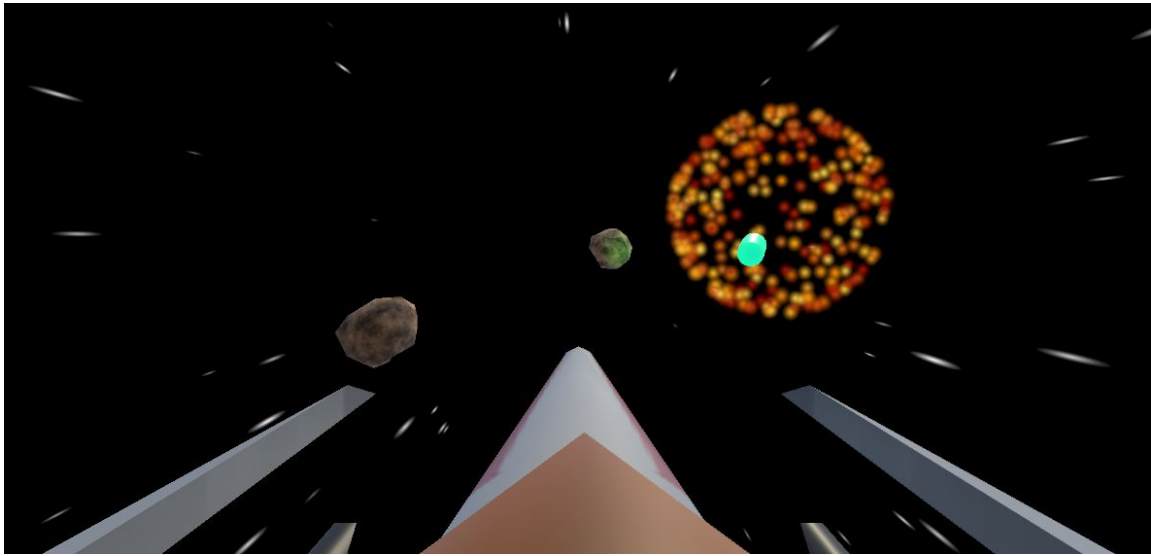
Ideas

- Cockpit
- Adopt Star/Dust Background
- Positions for asteroids
- Size/Speed
- Weapons
- Perspective

Our approach

1) Set the camera as child of the ship
2) Use WASD to move up/left/down/right
3) Shoot at mouse position

Use a duplicate of your level scene, so you don’t lose what is already working.



1) Camera

1) Set the camera projection to perspective
2) Drag the camera object into the ship object
3) Adjust its transform, so it’s right above the cockpit
4) Reduce the Near Clipping Plane of the Camera component, so we don’t see the ship model being cut in the game view

2) Movement

1) Duplicate your Player script and name it Player3D

2) Replace your y-direction with z-direction (Vector3.up → Vector3.back)

```
// transform.position += Vector3.right * amtToMoveX + Vector3.up * amtToMoveY;
transform.position += Vector3.right * amtToMoveX + Vector3.back * amtToMoveY;
```

What about position wrapping and clamping? Let's leave it for now...

3) Shooting

There are many ways how to do it. (LookAt(), mathematically, …). We use quaternions.

Approach:

1) Get world position of mouse
2) Calculate direction quaternion
3) Set rotation of projectiles accordingly when shooting
4) Make sure projectiles moves in the direction it is looking at

3) Shooting – Mouse position

1) Get the mouse position from the Input system. It is in Screen Space (pixel coordinates), though, so it must be converted first via ScreenToWorldPoint(..) function of the Camera object.

2) This function requires a Vector3 value with the pixel coordinates as x- and y-, and the depth as z-component. For the depth, we just try a value that fits best, e.g. 10.

```
// get world space position of mouse
var mousePos:Vector3 = Input.mousePosition;
mousePos.z = 10;
mousePos = Camera.main.ScreenToWorldPoint(mousePos);
```

3) Shooting – Direction quaternion

1) Calculate the direction from the weapon location and the mouse world position from the previous step

2) Generate a quaternion from that direction using Quaternion.LookRotation(..). This function needs to know the “up” direction, which is in our case Vector3.back

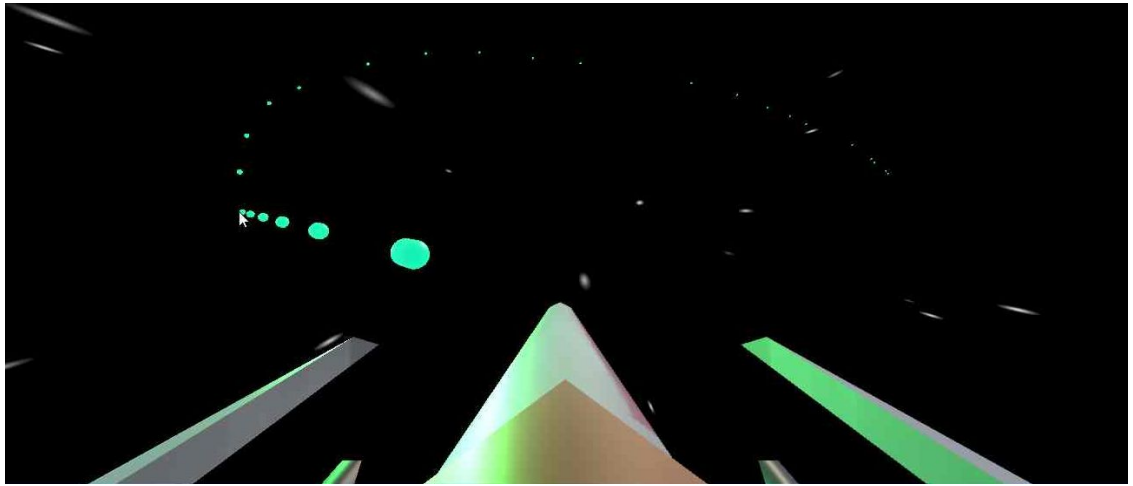
```
// calculate rotation by direction
var direction:Vector3 = (mousePos - weaponLocation.position).normalized;
var rotation:Quaternion = Quaternion.LookRotation(direction, Vector3.back);
```

Info: Why do we use quaternions? “They are compact, don't suffer from gimbal lock and can easily be interpolated. They are based on complex numbers and are not easy to understand intuitively.” https://docs.unity3d.com/ScriptReference/Quaternion.html

Gimbal lock example: https://en.wikipedia.org/wiki/Gimbal_lock#/media/File:Gimbal_Lock_Plane.gif

3) Shooting – Shoot and adjust projectile

1) When instantiating the projectile, use the quaternion we've just calculated as the rotation parameter
2) Inside the Projectile script, use the local up-direction of the projectile transform instead of the global up-direction to move the object
3) Test it out. What happens?



What happens?

- Nothing works anymore!
- Already implemented features need to be revisited and adjusted which can be stressful.
- Sadly such requirement changes like 2D⟶3D can occur later at work.

For now..

- … either take the challenge and adjust your project that it’s a 3D endless tunnel space shooter!
- … or take the opportunity to recap topics you haven’t fully understood yet.

Additional Tasks (free work time during tutorial or at home)

1) Take the challenge to make it 3D
2) Think of how to implement… :
    1) Movement boundaries
    2) Asteroid spawns
    3) ...