JAVASCRIPT GAMES: PROGRAMMING FUNDAMENTALS

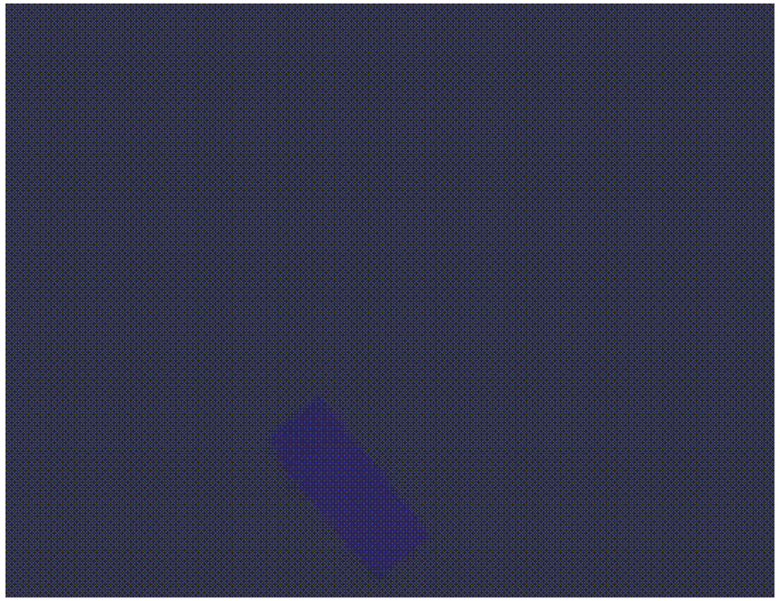
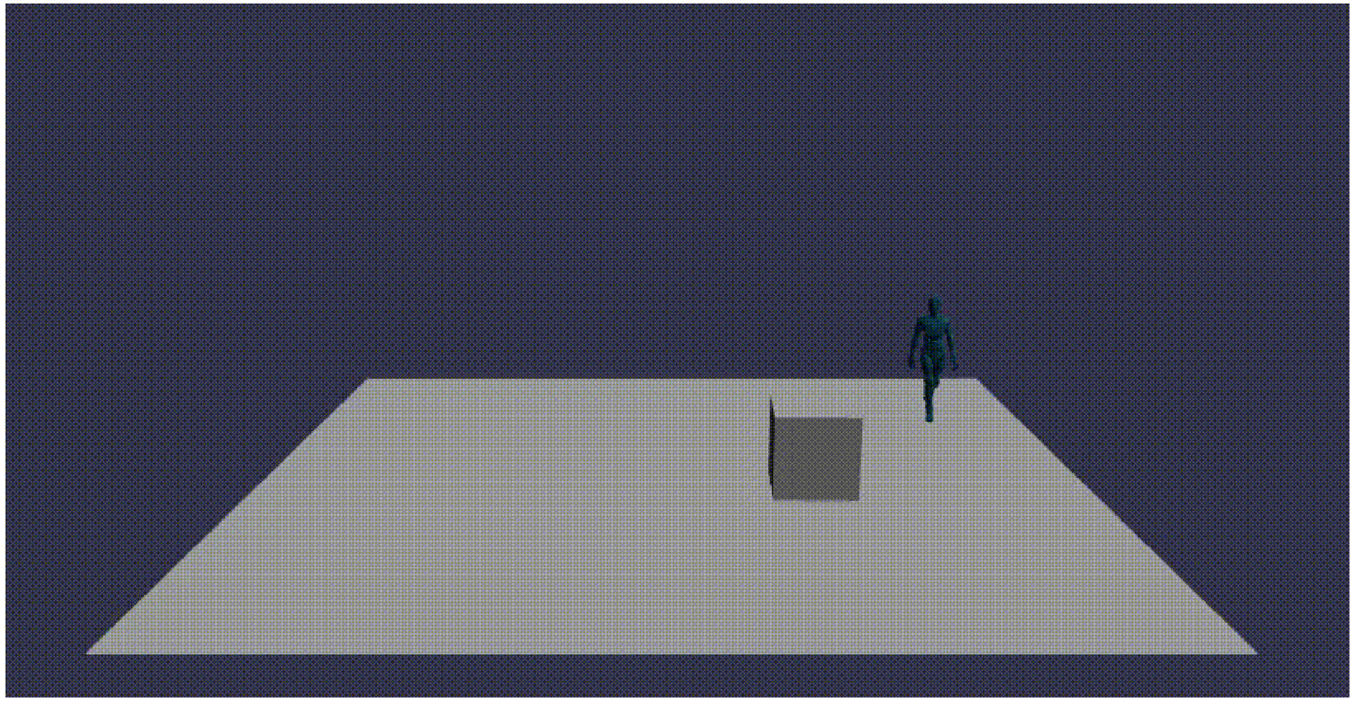
WEEK 10: BABYLONJS: ELEMENT 3

Contact:

Paisley
Alan.Williams@uws.ac.uk

Dumfries
Rebecca.Redden@uws.ac.uk

Module Coordinator
Derek.Turner@uws.ac.uk

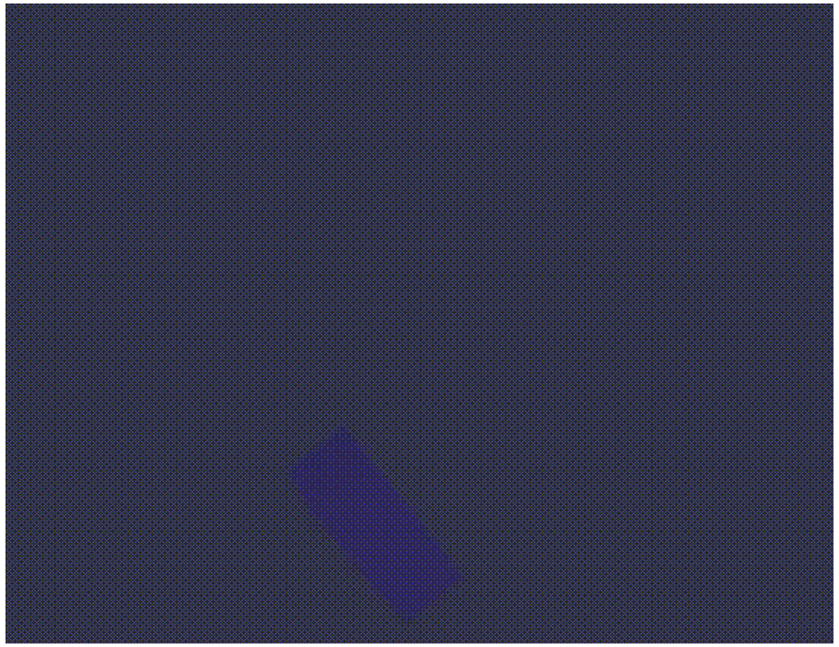



ASSESSMENT REMINDER

ELEMENT 3

ELEMENT 3

- **Remember each element is worth 15%.**
- **Element 3**
- "Demonstration of a movable player mesh incorporating player functional animation. Interaction between player and other objects with a physics in applied."
  - The element should demonstrate the follow:
    - A moveable player mesh.
    - Player animation (simple walking is fine).
    - Player interaction & physics (collision).

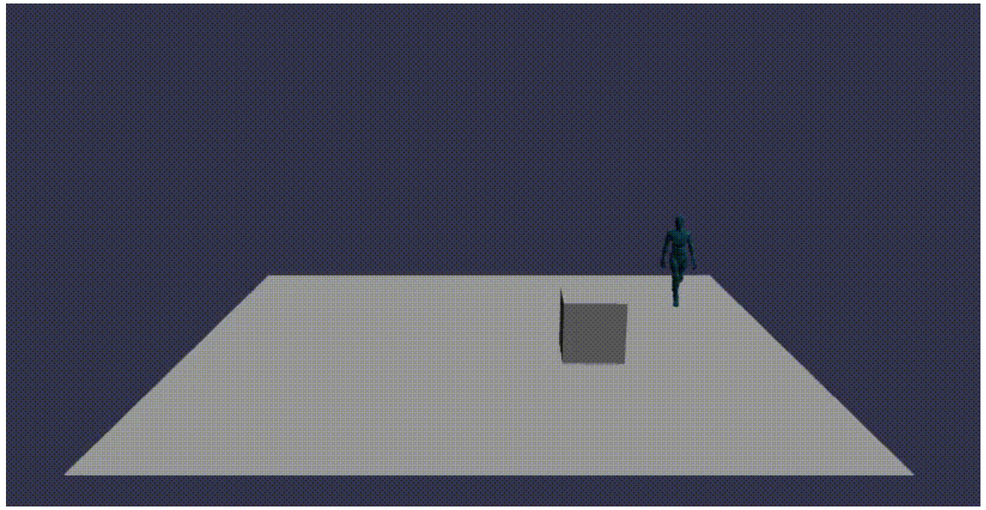


ELEMENT 3

- As usual… be creative! Create a small scene highlighting each of these.
- Player animation may be difficult to understand so I will give you a solution using a character we have for the demonstration.

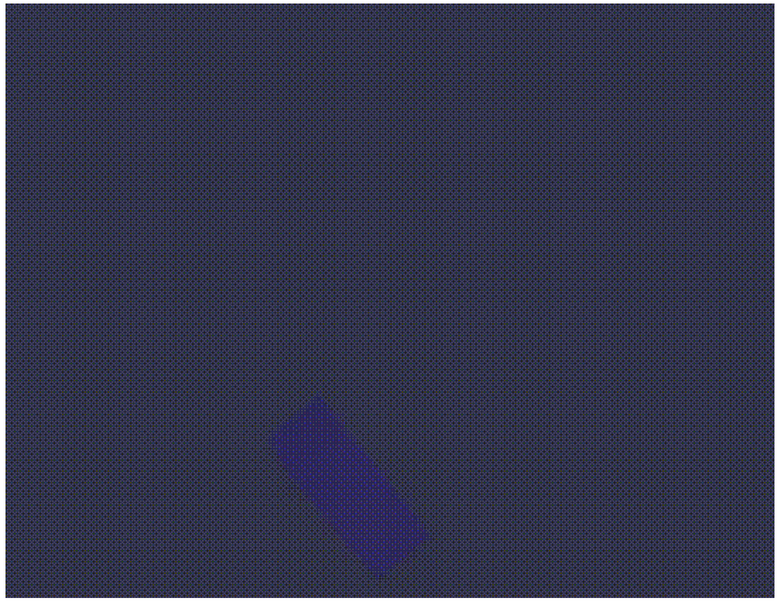
Reminders:

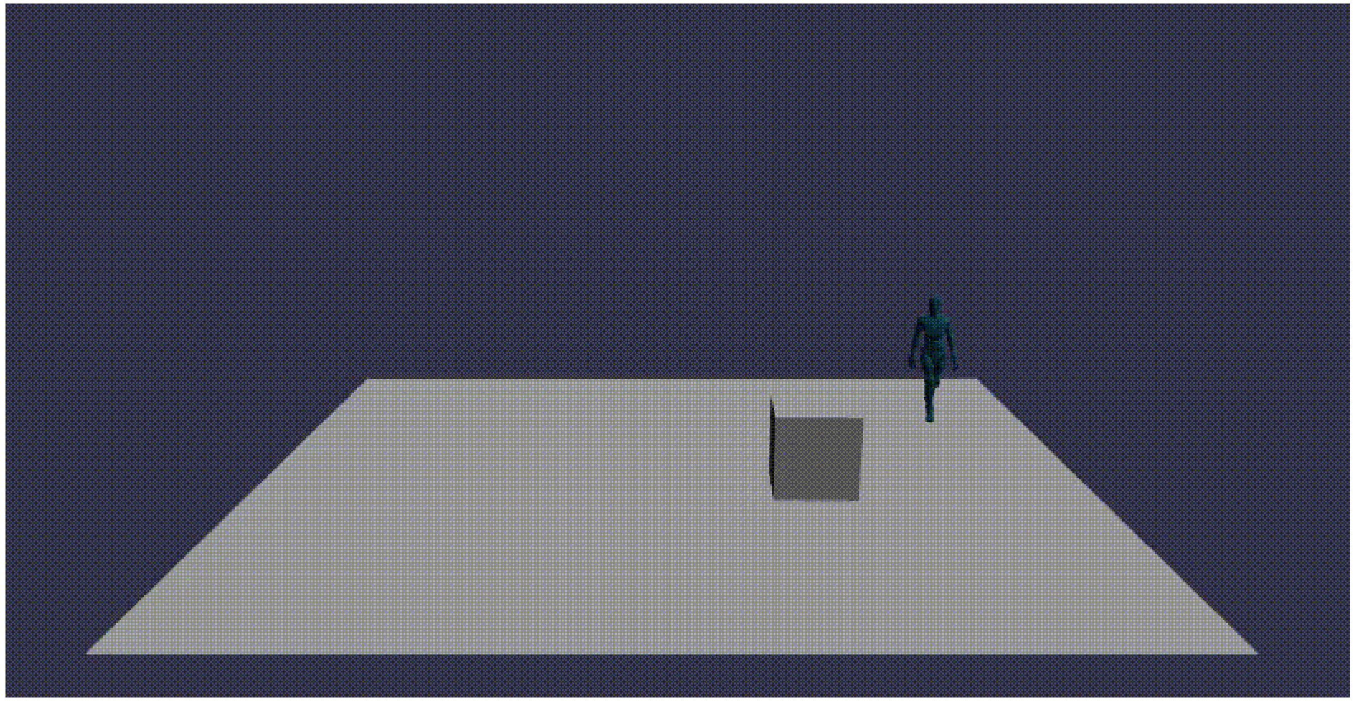
- Pay careful attention as we are taking from JavaScript to TypeScript.
- Be careful to how we work compared to how they set out code.
- Expand upon the tutorial where you can.
- You are free to use any resources either provided by BabylonJS, online, or of your own doing.



LAST WEEK

THIS WEEK

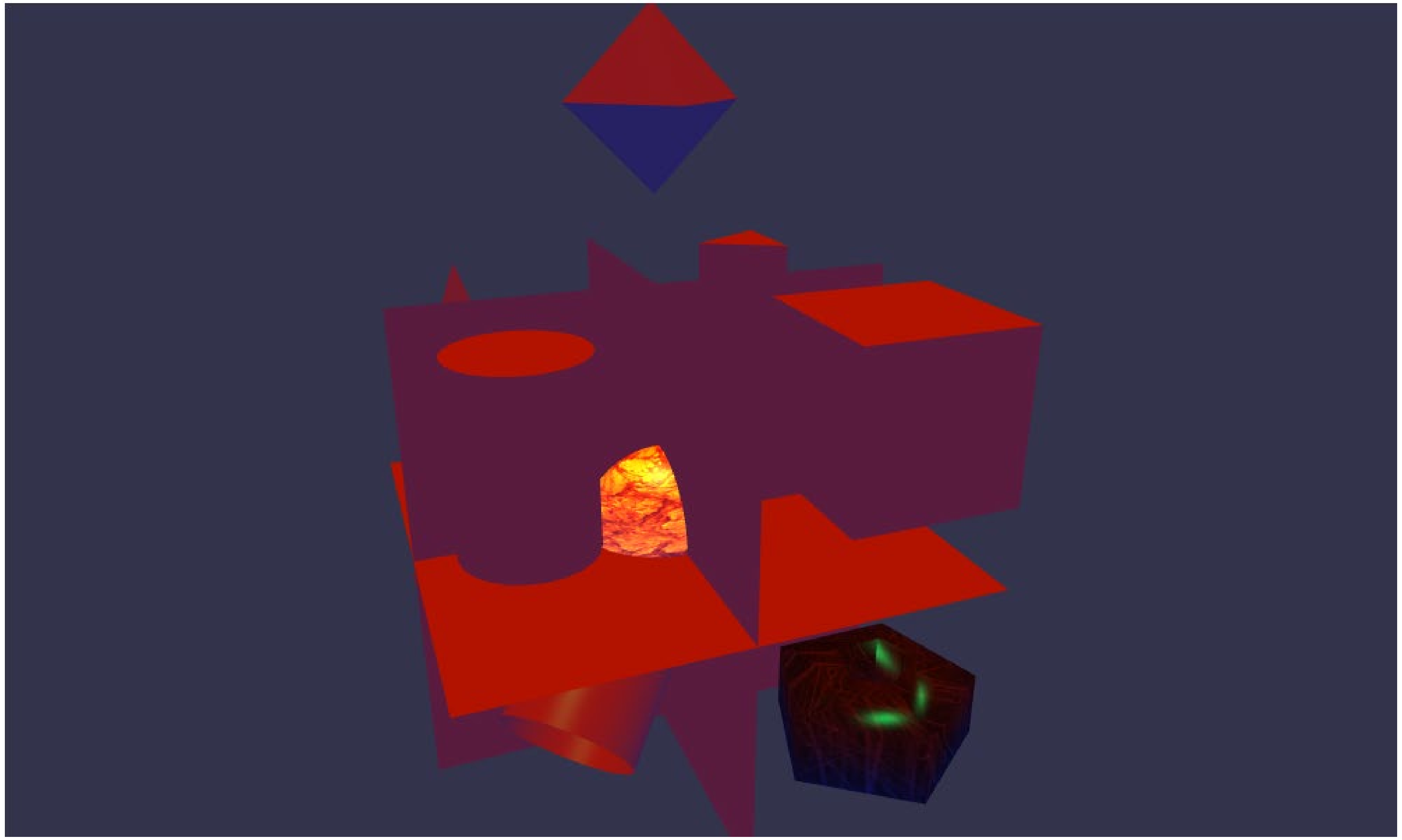




BABYLONJS

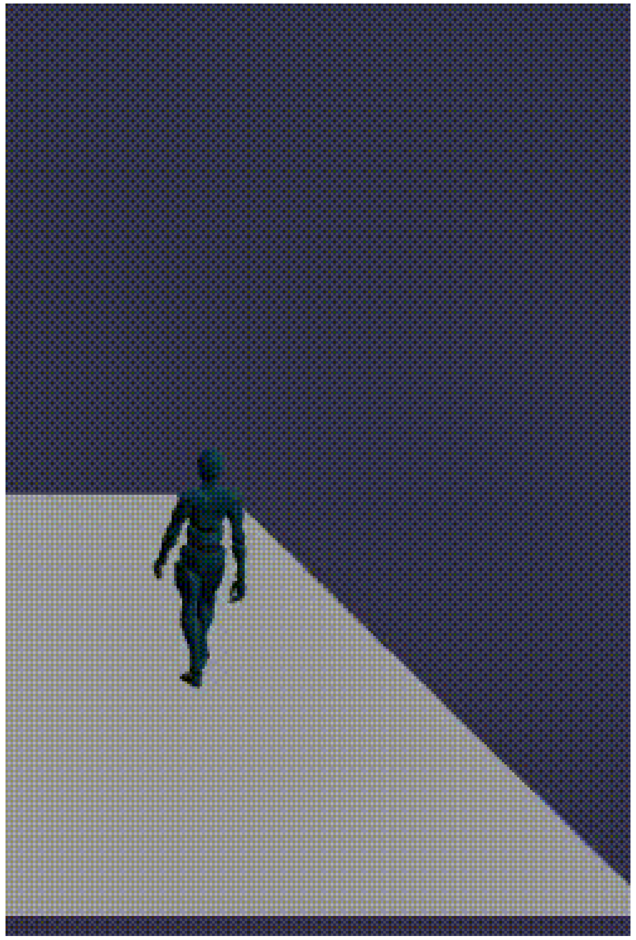
CONTINUED DEVELOPMENT

WEEK 1



PROCESS FOR ELEMENT 3 LEARNING

- Previous weeks we added internal BabylonJS meshes!
- **Element 3**
  - We will go through the process of dealing with:
    - Importing external meshes (outside BabylonJS).
    - Appropriate file formats to use.
    - Applying player movement.
    - General animation.
    - Applying animation to the player movement.
    - Applying physics and player interaction (collision).
- This element **can be** the most frustrating to set up as there is so much involved.
- Follow what I do today and refer to the GitHub repository.
- We'll cover a theory-based approach and go through demonstrations to get these applied to your projects for moving forward!



IMPORTING MESHES

```
import @babylonjs.loaders
```

- To import meshes in Babylon, we must import the BabylonJS Loaders library.
- We already done this on the initial setup. We should be declaring at the top.
- This allows the importing of support meshes from local files.
- BabylonJS Loaders support the following file types:
  - .glTF
  - .glb
  - .obj
  - .stl
- .babylon is a common file type for importing meshes too (we'll go over this).
- How we add this?
  - Well… BabylonJS likes to confuse you.

- Options to importing meshes;
  - SceneLoader.Append
  - SceneLoader.Load
  - SceneLoader.ImportMesh
  - SceneLoader.ImportMeshAsync
  - SceneLoader.LoadAssetContainer
  - SceneLoader.ImportAnimations
  - SceneLoader.AppendAsync
- You can explore this more here.
  - https://doc.babylonjs.com/features/featuresDeepDive/importers
- I will show you our method to do this.

EXTERNAL ASSETS

- Loading external assets can be confusing at first.
- Once you have the process… you are good to go.
- **Be careful with mesh materials/ texture preservation.**
- Some file formats will carry over the materials.
- .GLB can hold animations, as can .glTF.
- .FBX can be used too (another loader library/ extension must be used).
- Useful information from BabylonJS about 'Using External Assets'.
  - https://www.youtube.com/watch?v=3D6BtdMnnQI

IMPORTING A MESH

```
BABYLON.SceneLoader.ImportMesh(meshNames, rootUrl, sceneURLFileName, scene, onSuccess, onProgress, onError, message:,
pluginExtension);
```

- We will concentrate on using the **ImportMesh**.
- The most effective and efficient method to support importing external assets.

meshNames	A filter. Inserting “ ” loads all meshes.
rootUrl	Indication to the folder where your assets are being held.
sceneURLFileName	The data file or string name of file including extension at end.
onSuccess	A callback function when import succeeds.
onProgress	A callback function with progress event for each file being loaded.
onError	A callback function to handle any errors.
pluginExtension	Extension used to determine the plugin, returning function to apply anything to object.

IMPORTING A MESH

```
BABYLON.SceneLoader.ImportMesh("", "assets/models/", "isometric_buildings_watermelon_house.glb", scene, function(meshes) {
    const building = meshes[0];
    building.scaling = new BABYLON.Vector3(0.02, 0.02, 0.02);
});
```

- This seems to be a more efficient, constant working way.
- This allows us to effectively load in the assets, adding any attributes.
- The method allows for this to applied after the loading of the asset has happened.
- When the mesh is imported, it will scale it effectively.
- For this example:
  - https://sketchfab.com/3d-models/isometric-buildings-watermelon-house-90eed0a06042446eb3a633c8e2b19784#download

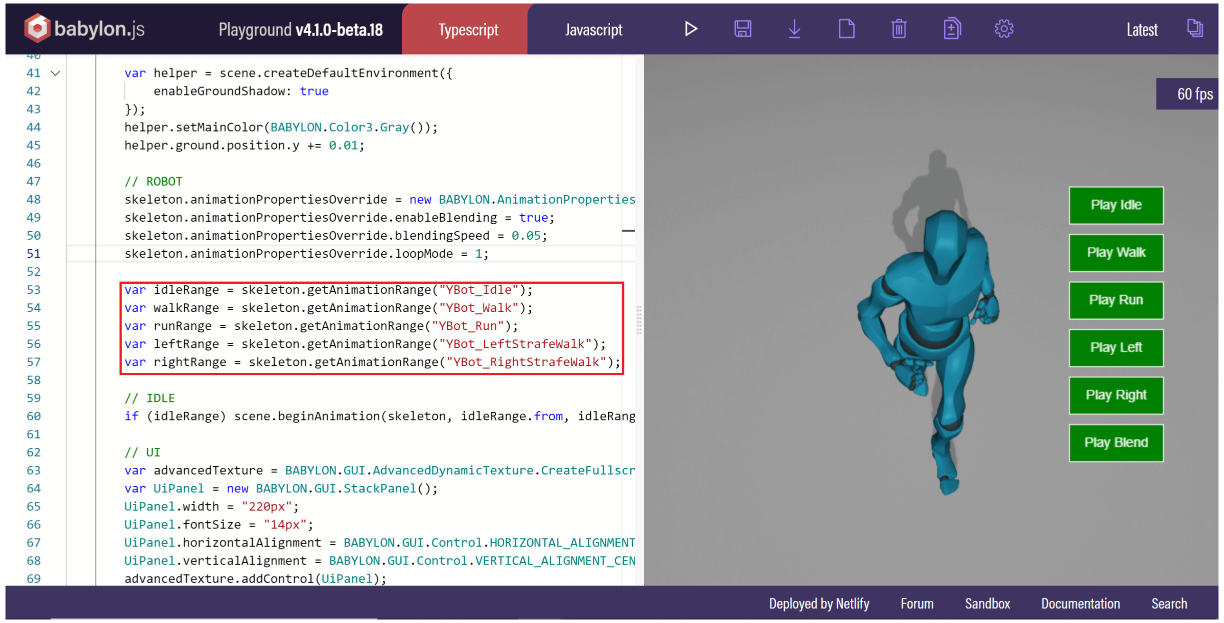
APPLYING SCALING, POSITIONING, ROTATION

```
BABYLON.SceneLoader.ImportMesh("", "assets/models/", "isometric_buildings_watermelon_house.glb", scene, function(meshes) {
    const building = meshes[0];
    building.scaling = new BABYLON.Vector3(0.02, 0.02, 0.02);
});
```

- `meshes[0]` is applying to the base skeleton of the model, annoying us to manipulate the mesh.
- Applying scaling, position, rotation, or any Vector3 based object, there are **two** methods this can be done.
- Method one.
  - ‘`building.scaling = new BABYLON.Vector3(0.02, 0.02, 0.02);`’
- Method two.
  - Individually assigning.
  - ‘`building.scaling.x = 1; building.scaling.y = 1;`’ and so on.
- Method one is a more clean and efficient coding way.
- We’ve saw this in previous element demonstrations.

EXTERNAL ASSET RECOMMENDATIONS

- TurboSquid
  - https://www.turbosquid.com
- SketchFab
  - https://sketchfab.com
- Quaternius
  - https://quaternius.com
- Mixamo
  - https://www.mixamo.com/#/
- These are some recommendations we have used in the past.
- There is support with Blender, Maya, 3ds Max.
  - https://doc.babylonjs.com/features/featuresDeepDive/Exporters
  - https://playground.babylonjs.com/#BCU1XR#0



```
var helper = scene.createDefaultEnvironment({
    enableGroundShadow: true
});
helper.setMainColor(BABYLON.Color3.Gray());
helper.ground.position.y += 0.01;

// ROBOT
skeleton.animationPropertiesOverride = new BABYLON.AnimationProperties
skeleton.animationPropertiesOverride.enableBlending = true;
skeleton.animationPropertiesOverride.blendingSpeed = 0.05;
skeleton.animationPropertiesOverride.loopMode = 1;

var idleRange = skeleton.getAnimationRange("YBot_Idle");
var walkRange = skeleton.getAnimationRange("YBot_Walk");
var runRange = skeleton.getAnimationRange("YBot_Run");
var leftRange = skeleton.getAnimationRange("YBot_LeftStrafeWalk");
var rightRange = skeleton.getAnimationRange("YBot_RightStrafeWalk");

// IDLE
if (idleRange) scene.beginAnimation(skeleton, idleRange.from, idleRang

// UI
var advancedTexture = BABYLON.GUI.AdvancedDynamicTexture.CreateFullscr
var UiPanel = new BABYLON.GUI.StackPanel();
UiPanel.width = "220px";
UiPanel.fontSize = "14px";
UiPanel.horizontalAlignment = BABYLON.GUI.Control.HORIZONTAL_ALIGNMENT
UiPanel.verticalAlignment = BABYLON.GUI.Control.VERTICAL_ALIGNMENT_CEN
advancedTexture.addControl(UiPanel);
```

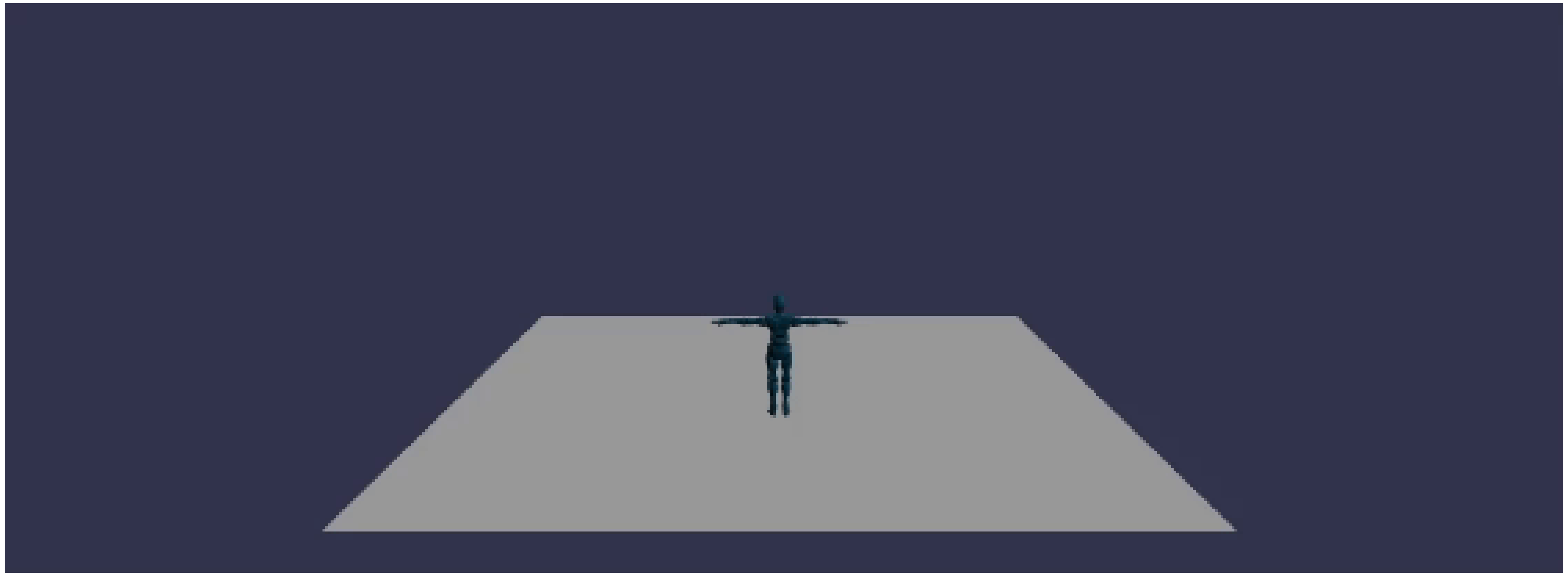
MIXAMO

- Highly popular process with Mixamo, Blender and BabylonJS to support bringing external assets into your project.
- There is a detailed guide on the documentation that goes through this.
- Recommend looking at this in your own time.
  - https://doc.babylonjs.com/features/featuresDeepDive/Exporters/Mixamo_to_Babylon

MESH/ CHARACTER/ ANIMATION WORKFLOW

- The how to of Character Animation for BabylonJS.
  - Use Blender to create the model.
  - Export the model as .FBX.
  - Load the model in Mixamo to rig it and add animations.
  - Download animated models as .FBX files.
  - Use Blender to combine the animated models into one.
  - Export as .GLB or .BABYLON(animation as Animation group).
  - Load the .GLB/ .BABYLON in a BabylonJS Playground and control the model movement via keyboard.
- More details on process:
  - https://doc.babylonjs.com/features/featuresDeepDive/animation/animatedCharacter

PLAYER MOVEMENT



ACTION MANAGER

- To execute player movement in a 3D space we will use Events.
- More specifically… the **Action Manager**.
- This will allow us to apply movement or interactions to specific meshes, objects, and beyond.

```
mesh.actionManager = new BABYLON.ActionManager(scene);
scene.actionManager = new BABYLON.ActionManager(scene);
```

- Attach to the **mesh** or **scene** in general. Then you must **register actions**.

https://playground.babylonjs.com/#9RUHH#5

REGISTERING ACTIONS

https://playground.babylonjs.com/#9RUHH#5

What actions do we need for player movement?

- Well first… we must create an empty object of where we will apply the inputs and create our actionManager.

```
let inputMap = {};
scene.actionManager = new BABYLON.ActionManager(scene);
```

- We are going to then register **two** action triggers.
  - **OnKeyDown** & **OnKeyUp**.
  - Additionally you can have **OnEveryFrameTrigger**.
- These can only be executed for scenes.

```
scene.actionManager.registerAction(new BABYLON.ExecuteCodeAction(BABYLON.ActionManager.OnKeyDownTrigger, function (evt) {
    inputMap[evt.sourceEvent.key] = evt.sourceEvent.type == “keydown”;
}));

scene.actionManager.registerAction(new BABYLON.ExecuteCodeAction(BABYLON.ActionManager.OnKeyUpTrigger, function (evt) {
    inputMap[evt.sourceEvent.key] = evt.sourceEvent.type == “keydown”;
}));
```

ADDITIONAL ACTIONS

```
scene.actionManager.registerAction(new BABYLON.ExecuteCodeAction(BABYLON.ActionManager.OnKeyDownTrigger, function (evt) {
     inputMap[evt.sourceEvent.key] = evt.sourceEvent.type == “keydown”;
}));
```

- Most actions have a **propertyPath** property.
- Direct values such as position can be used or position.x.
- Additional properties can be executed with the Action Manager.
- Useful playground resource: https://www.babylonjs-playground.com/#15EY4F#15

Available Actions (BABYLON.xxx)
SwitchBooleanAction
SetValueAction*
IncrementValueAction*
PlayAnimationAction
StopAnimationAction
DoNothingAction

Available Actions (BABYLON.xxx)
CombineAction
ExecuteCodeAction
SetParentAction
PlaySoundAction
StopSoundAction
InterpolateValueAction

More information can be found here:

https://doc.babylonjs.com/features/featuresDeepDive/events/actions

https://playground.babylonjs.com/#9RUHH#5

*** you will see these appear today when implementing player movement.**

PLAYER MOVEMENT

- We want to manipulate our player we import to move.
- We'll do this with [W, A, S, D] and arrow keys.
- Inputs such as "ArrowUp" refer to the JavaScript Key Event values (Event.Key/Event.Code).
  - https://www.freecodecamp.org/news/javascript-keycode-list-keypress-event-key-codes/
  - Space would be defined as "Space".
- This is **simple** player movement, but you could go beyond.
- I will detail this step-by-step in the lab sheet.
- Great thing about our setup is that we can combine importing mesh and movement into one function.

```
// Game/Render loop
scene.onBeforeRenderObservable.add(()=>{
    var keydown = false;
    if(inputMap["w"] || inputMap["ArrowUp"]){
        newMeshes[0].position.z+=0.1
        newMeshes[0].rotation.y = 0
        keydown=true;
    }
    if(inputMap["a"] || inputMap["ArrowLeft"]){
        newMeshes[0].position.x-=0.1
        newMeshes[0].rotation.y = 3*Math.PI/2
        keydown=true;
    }
    if(inputMap["s"] || inputMap["ArrowDown"]){
        newMeshes[0].position.z-=0.1
        newMeshes[0].rotation.y = 2*Math.PI/2
        keydown=true;
    }
    if(inputMap["d"] || inputMap["ArrowRight"]){
        newMeshes[0].position.x+=0.1
        newMeshes[0].rotation.y = Math.PI/2
        keydown=true;
    }
})
```

PLAYER MOVEMENT

```
BABYLON.SceneLoader.ImportMesh("", "assets/models/",
"isometric_buildings_watermelon_house.glb", scene,
function(meshes) {
    const building = meshes[0];
    building.scaling = new BABYLON.Vector3(0.02, 0.02, 0.02);
});
```

- Previously, we assigned **`let mesh = newMeshes[0];`**.
- Using this, we can assign **`mesh.position`** or **`mesh.rotation`**.
- This allows us to manipulate anything with the mesh (position, rotation, scale).

```
// Game/Render loop
scene.onBeforeRenderObservable.add(()=>{
    var keydown = false;
    if(inputMap["w"] || inputMap["ArrowUp"]){
        mesh.position.z+=0.1
        mesh.rotation.y = 0
        keydown=true;
    }
    if(inputMap["a"] || inputMap["ArrowLeft"]){
        mesh.position.x-=0.1
        mesh.rotation.y = 3*Math.PI/2
        keydown=true;
    }
    if(inputMap["s"] || inputMap["ArrowDown"]){
        mesh.position.z-=0.1
        mesh.rotation.y = 2*Math.PI/2
        keydown=true;
    }
    if(inputMap["d"] || inputMap["ArrowRight"]){
        mesh.position.x+=0.1
        mesh.rotation.y = Math.PI/2
        keydown=true;
    }
})
```

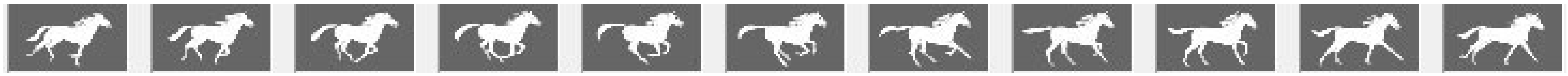
PLAYER MOVEMENT

scene.onBeforeRenderObservable.add()

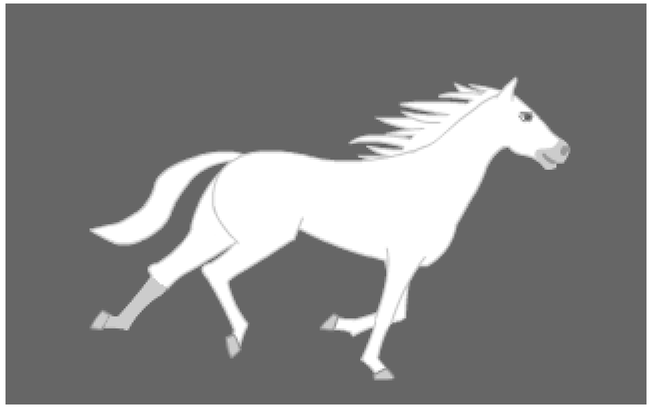
- This is the rendering loop (similar to Update() in Unity).
- Executed every frame, updating necessary elements.
- We will use this for character input.
- This allows for;
  - Input of certain keys.
  - Execution of certain animations.
  - Executed within the ImportMesh function we create.
- We are going to concentrate on movement.
- Visit the Playground link to explore animations more.
  - We will look more in the demonstration.
  - https://doc.babylonjs.com/features/featuresDeepDive/animation/animatedCharacter
  - https://playground.babylonjs.com/#AHQEIB#17

https://www.babylonjs-playground.com/#15EY4F#15

ANIMATION



- Animations make your game come alive!
- Animations takes three elements into account;
  - The action wanted.
  - The timing.
  - The number of frames.
- You need to consider how long you want the sequence to be.
- You need to consider how smooth it needs to be.
- Smoother movement = more frames will be required.



ANIMATION TERMINOLOGY

- When exploring the BabylonJS documentation, you need to be aware of the terminology they may use. These will have the given meaning within the How_To about animating.

*like a play or film script but applies to one property of a performer. This consists of:

- The property to be changed (position, rotation etc.)
- The rate of change of the property in FPS (Frames Per Second).
- Looping Conditions
- Key values of the property at key frames.

Terminology	Definition
Performer	Item that can be animated (mesh, light etc.)
Frame	Animation frame, not a rendered frame of the scene
Animation	*see more info
Scripted Performer	The performer plus all the animation to be undertaken by the performer.
Performance	Scripted performer and actions done by performer following the script (animatable object).
Clip	The viewable result of a performance.
Cartoon	A series of clips played at time intervals.

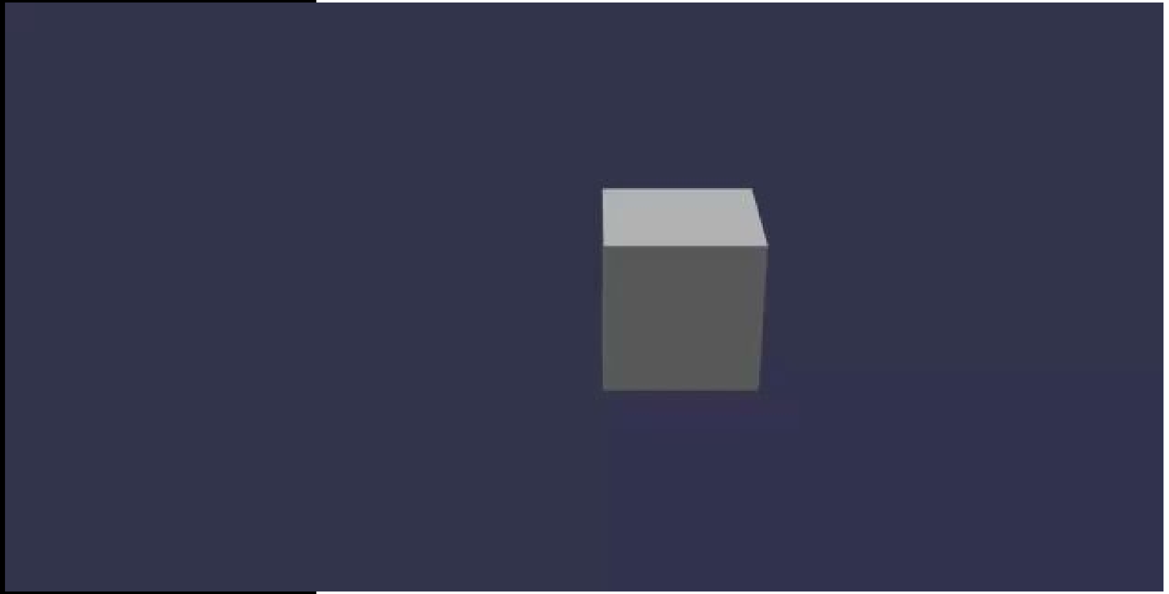
https://doc.babylonjs.com/features/featuresDeepDive/animation/animation_introduction

DESIGNING ANIMATIONS

```
const box = BABYLON.MeshBuilder.CreateBox("box", scene);
    box.position.x = 2;
    const frameRate = 10;
    const xSlide = new BABYLON.Animation("xSlide", "position.x",
frameRate, BABYLON.Animation.ANIMATIONTYPE_FLOAT,
BABYLON.Animation.ANIMATIONLOOPMODE_CYCLE);

    const keyFrames = [];
    keyFrames.push({
        frame: 0,
        value: 2
    });
    keyFrames.push({
        frame: frameRate,
        value: -2
    });
    keyFrames.push({
        frame: 2 * frameRate,
        value: 2
    });

    xSlide.setKeys(keyFrames);
    box.animations.push(xSlide);
    scene.beginAnimation(box, 0, 2 * frameRate, true);
```



ANIMATION METHOD

```
const myAnim = new BABYLON.Animation(name, property, frames_per_second,
                    property_type, loop_mode)
```

```
const xSlide = new BABYLON.Animation("xSlide", "position.x", frameRate,
BABYLON.Animation.ANIMATIONTYPE_FLOAT, BABYLON.Animation.ANIMATIONLOOPMODE_CYCLE);
```

- Name
  - String, name of the animation.
- Property
  - String, a property of the object that the animation will be applied to.
  - This includes position.x for example as shown before.
- FPS (Frames Per Second)
  - Number, the number of animation frames per second (independent of the scene rendering frames per second).

ANIMATION METHOD

```
const myAnim = new BABYLON.Animation(name, property, frames_per_second,
property_type, loop_mode)
```

```
const xSlide = new BABYLON.Animation("xSlide", "position.x", frameRate,
BABYLON.Animation.ANIMATIONTYPE_FLOAT, BABYLON.Animation.ANIMATIONLOOPMODE_CYCLE);
```

- Property Type
  - Number, the property type of the property parameter. This can be set with the following constraints.
    - **BABYLON.Animation.ANIMATIONTYPE_COLOR3**
    - **BABYLON.Animation.ANIMATIONTYPE_FLOAT**
    - **BABYLON.Animation.ANIMATIONTYPE_MATRIX**
    - **BABYLON.Animation.ANIMATIONTYPE_QUATERNION**
    - **BABYLON.Animation.ANIMATIONTYPE_VECTOR2**
    - **BABYLON.Animation.ANIMATIONTYPE_VECTOR3**

- Loop Mode (optional)
  - Number, this can be set using the following parameters.
    - **BABYLON.Animation.ANIMATIONLOOPMODE_CYCLE**
      - Restart the animation from initial value.
    - **BABYLON.Animation.ANIMATIONLOOPMODE_CONSTANT**
      - Pause the animation at the final value.
    - **BABYLON.Animation.ANIMATIONLOOPMODE_RELATIVE**
      - Repeat the animation incrementing using key value gradients. This may mean

SET KEY FRAMES

```
myAnim.setKeys(myKeys);
```

- myKeys (keyFrames) refers to the array of objects created.
- Each object has two properties:
  - Frame.
    - The frame number.
  - Value.
    - For the property being changed.

```
const keyFrames = [];
    keyFrames.push({
        frame: 0,
        value: 2
    });
    keyFrames.push({
        frame: frameRate,
        value: -2
    });
    keyFrames.push({
        frame: 2 * frameRate,
        value: 2
    });
```

```
xSlide.setKeys(keyFrames);
```

SETTING & TRIGGERING THE ANIMATION

- To run the animation, you must assign (push) the animations onto the mesh.
- Once this is done, you need to trigger the animation.
- scene.beginAnimation can be given a variable and supported methods can be used.
- Supporting methods include:
  - pause()
  - restart()
  - stop()
  - reset()

```
mesh.animations.push(myAnim)
```

```
box.animations.push(xSlide);
```

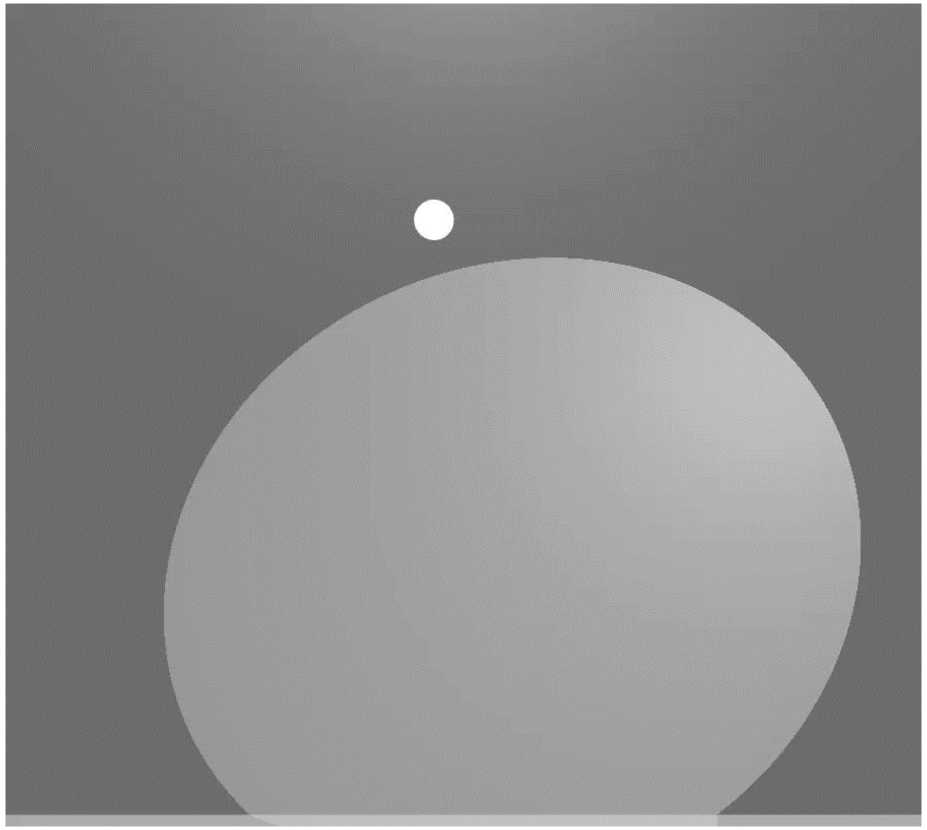
```
scene.beginAnimation(target, from, to, loop);
```

```
scene.beginAnimation(box, 0, 2 * frameRate, true);
```

Parameter	Output
target	Object to be animated.
from	The frame the animation starts at.
to	The frame the animation ends at.
loop	True or false. Repeat the animation.

ANIMATIONS

- Sequence animations can be achieved in BabylonJS.
- This involves combining several clips to form a sequence.
  - https://doc.babylonjs.com/features/featuresDeepDive/animation/sequenceAnimations
- Experiment with sequence animations in the lab.
- You may consider involving animations to your level to enhance the environment design in areas or in general.
- The method in which we implement the animation for our character today… will be slightly different to these methods so far.

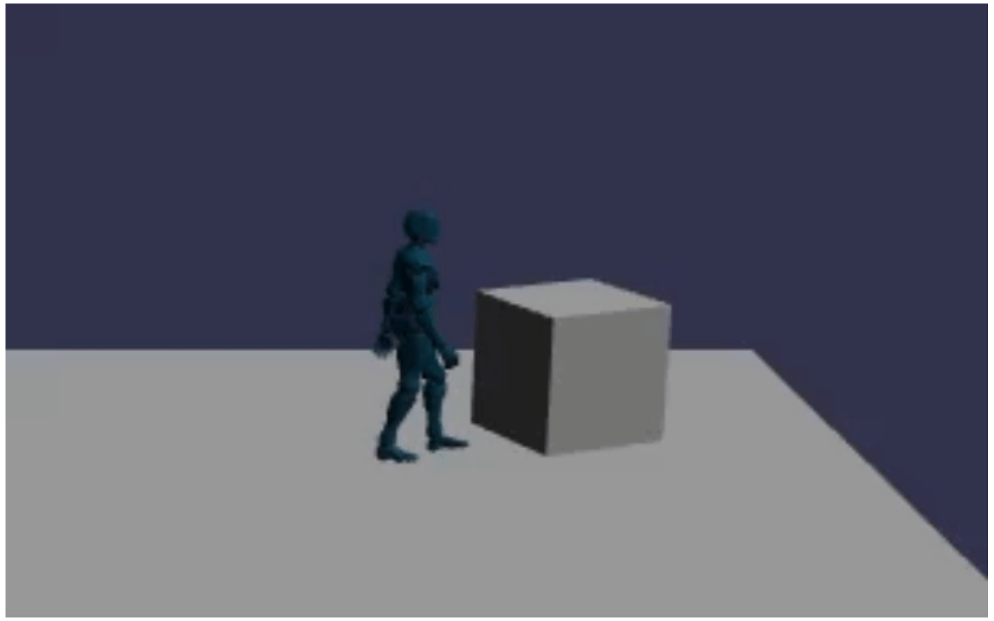


USEFUL PLAYGROUND EXAMPLES

- Some Playground animation examples.
- Good at showcasing different processes.
  - https://playground.babylonjs.com/#7V0Y1I
  - https://playground.babylonjs.com/#BCU1XR#0
  - https://playground.babylonjs.com/#AHQEIB#17
  - https://playground.babylonjs.com/#Z6SWJU#5
  - https://playground.babylonjs.com/#2L26P1#8
  - https://www.babylonjs-playground.com/#XZ0TH6

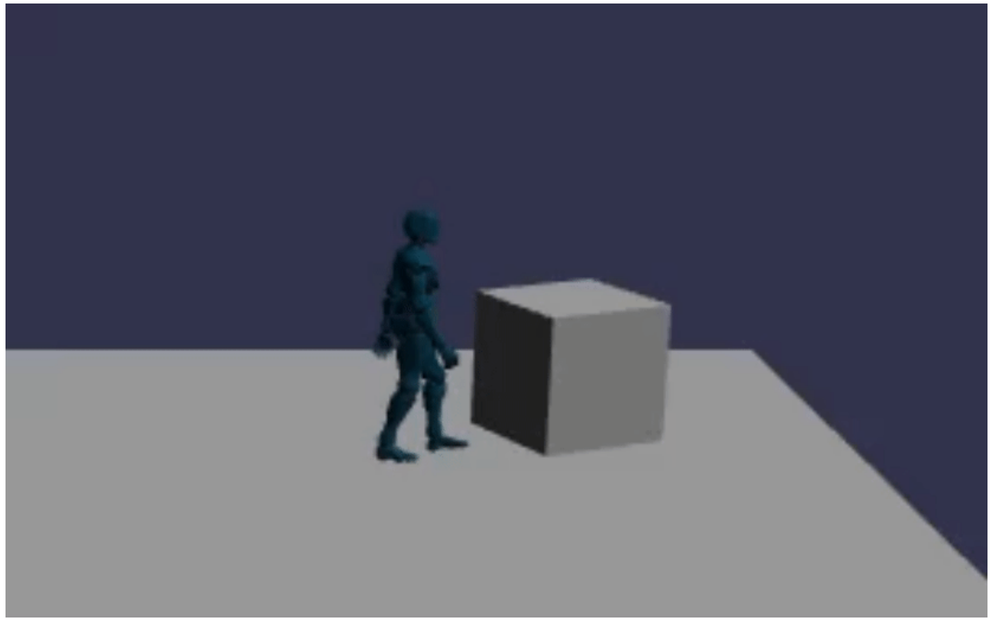
PHYSICS

- BabylonJS has a collision system that can detect when objects collide, and trigger programmed responses.
- BabylonJS generally uses 3rd party physics engines to add physical interaction with objects.
- Physics in BabylonJS is what effectively makes your collision work.
- All interactable objects **must have** a physics imposter.



PHYSICS

- BabylonJS used to have three alternative physics engines:
  - Ammo.js
  - Cannon.js
  - Oimo.js
- We can look at Cannon.js and collide our model with this simple box.
- But recently in v2 of Physics in BabylonJS, they have updated to using a plugin for the **Havok physics engine**.
  - https://doc.babylonjs.com/features/featuresDeepDive/physics/havokPlugin
- A guide to apply to your project will be given in the lab sheet/ demonstration.



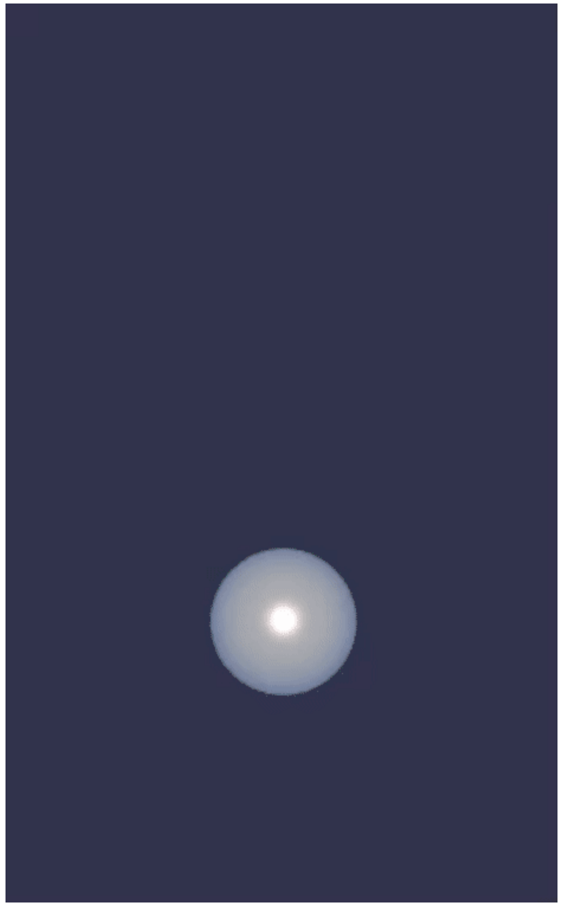
COLLISION

scene.onBeforeRenderObservable.add()

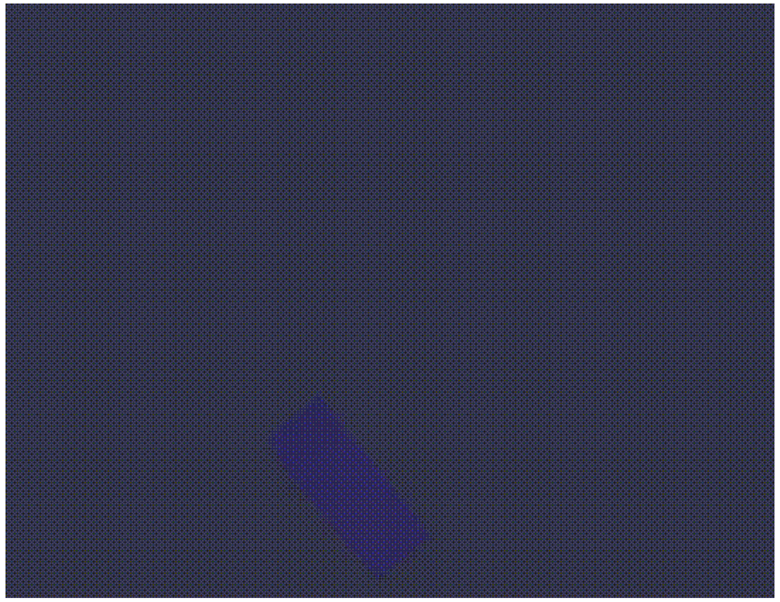
- Any collision-based code will go in this.

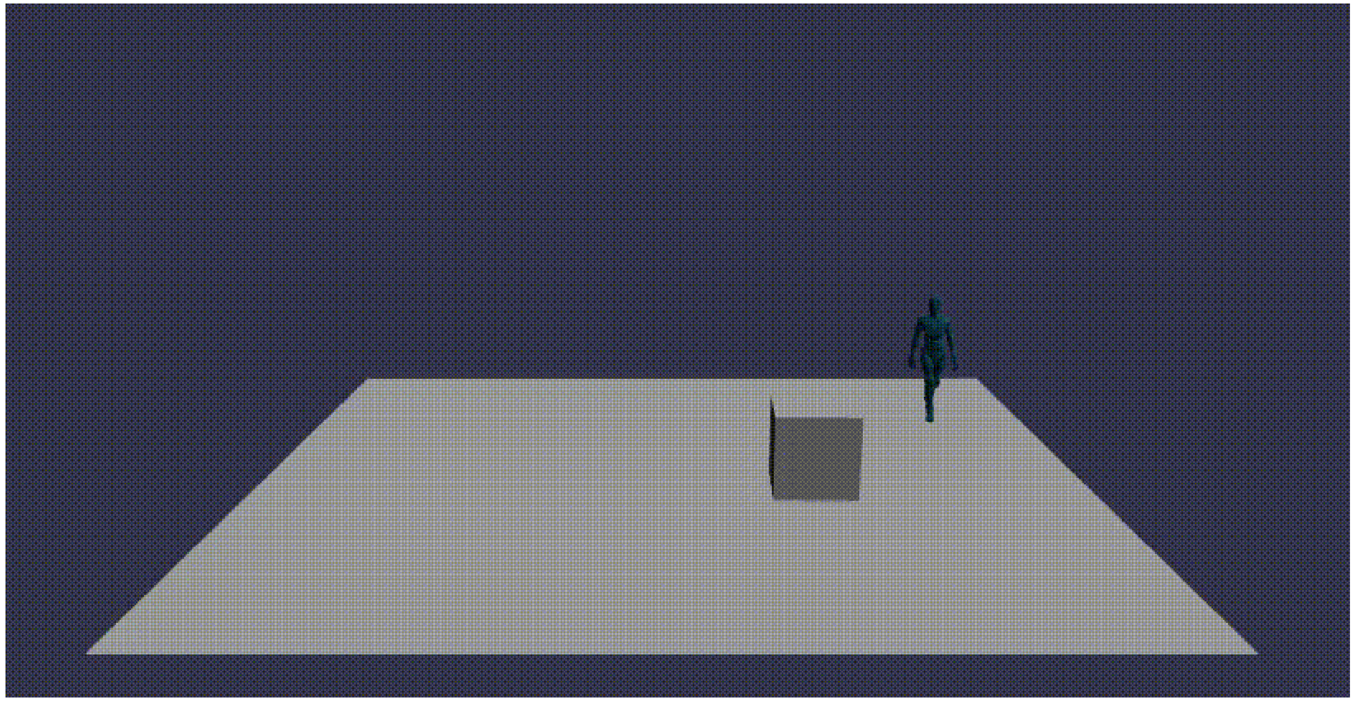
```
mesh1.intersectsMesh(mesh2);
```

- Checks if two meshes are colliding.
- Condition statement and have it output certain elements.
  - This is not needed for Element 3.
  - If you demonstration applied physics, then you are fine.
- This could extend in your element 4 & 5 allowing for a collectibles system/ scoring system in your elements.
- We'll look at this more with the incorporate of Babylon GUI next week.



YOUR TASK TODAY

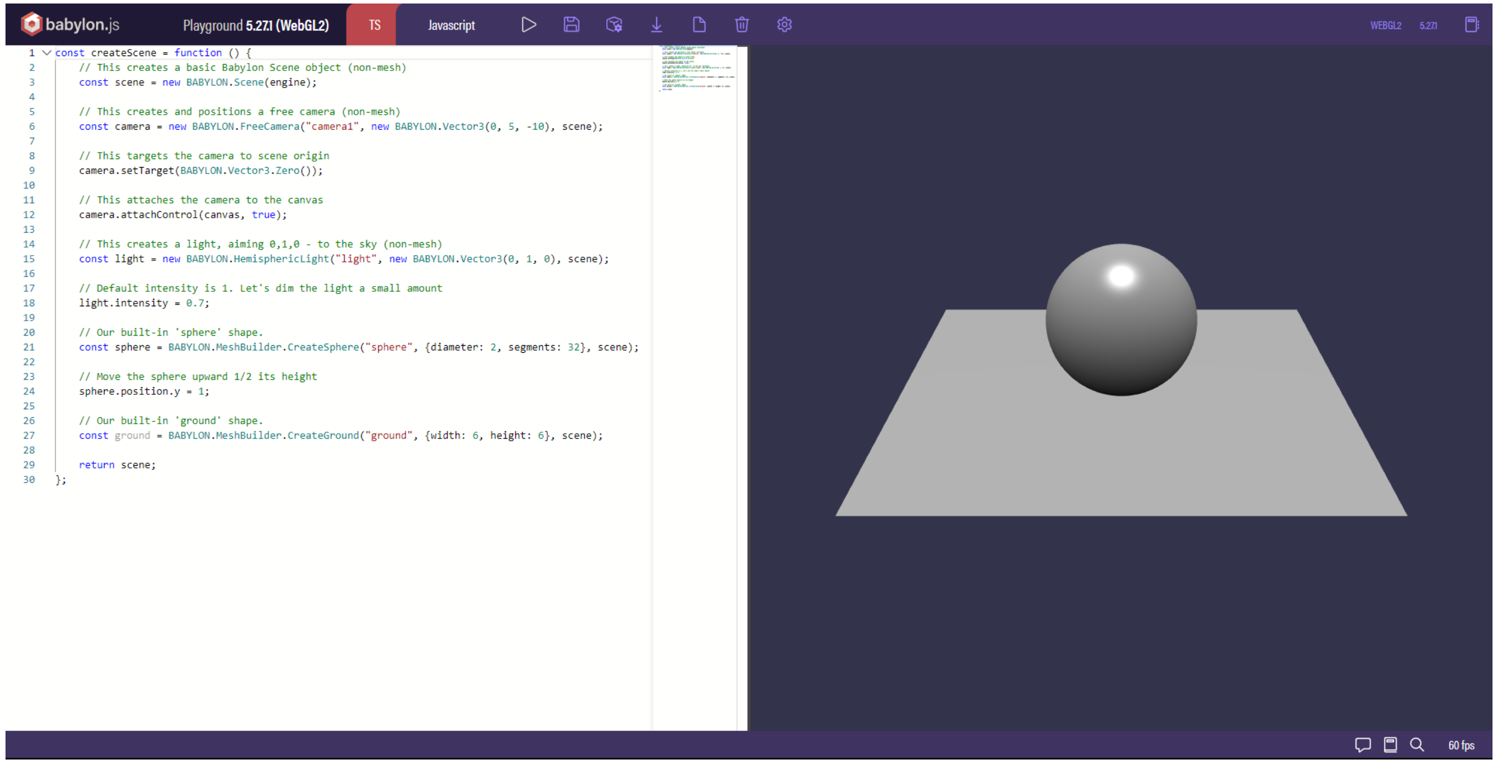




BABYLONJS PLAYGROUND

https://playground.babylonjs.com/

- The lab sheet today has many references to the BabylonJS documentation and playground.
- It is important that you understand how to transfer code from these to your own project.
- Get used to this and practice beyond the demonstration today.



```
const createScene = function () {
    // This creates a basic Babylon Scene object (non-mesh)
    const scene = new BABYLON.Scene(engine);

    // This creates and positions a free camera (non-mesh)
    const camera = new BABYLON.FreeCamera("camera1", new BABYLON.Vector3(0, 5, -10), scene);

    // This targets the camera to scene origin
    camera.setTarget(BABYLON.Vector3.Zero());

    // This attaches the camera to the canvas
    camera.attachControl(canvas, true);

    // This creates a light, aiming 0,1,0 - to the sky (non-mesh)
    const light = new BABYLON.HemisphericLight("light", new BABYLON.Vector3(0, 1, 0), scene);

    // Default intensity is 1. Let's dim the light a small amount
    light.intensity = 0.7;

    // Our built-in 'sphere' shape.
    const sphere = BABYLON.MeshBuilder.CreateSphere("sphere", {diameter: 2, segments: 32}, scene);

    // Move the sphere upward 1/2 its height
    sphere.position.y = 1;

    // Our built-in 'ground' shape.
    const ground = BABYLON.MeshBuilder.CreateGround("ground", {width: 6, height: 6}, scene);

    return scene;
};
```

REMINDER

- Understanding how you translate your code from the documentation or playground is **important**.
- Let’s look at this example.

CREATE A BOX MESH

```
const box = BABYLON.MeshBuilder.CreateBox("box", options, scene);
//scene is optional and defaults to the current scene
```

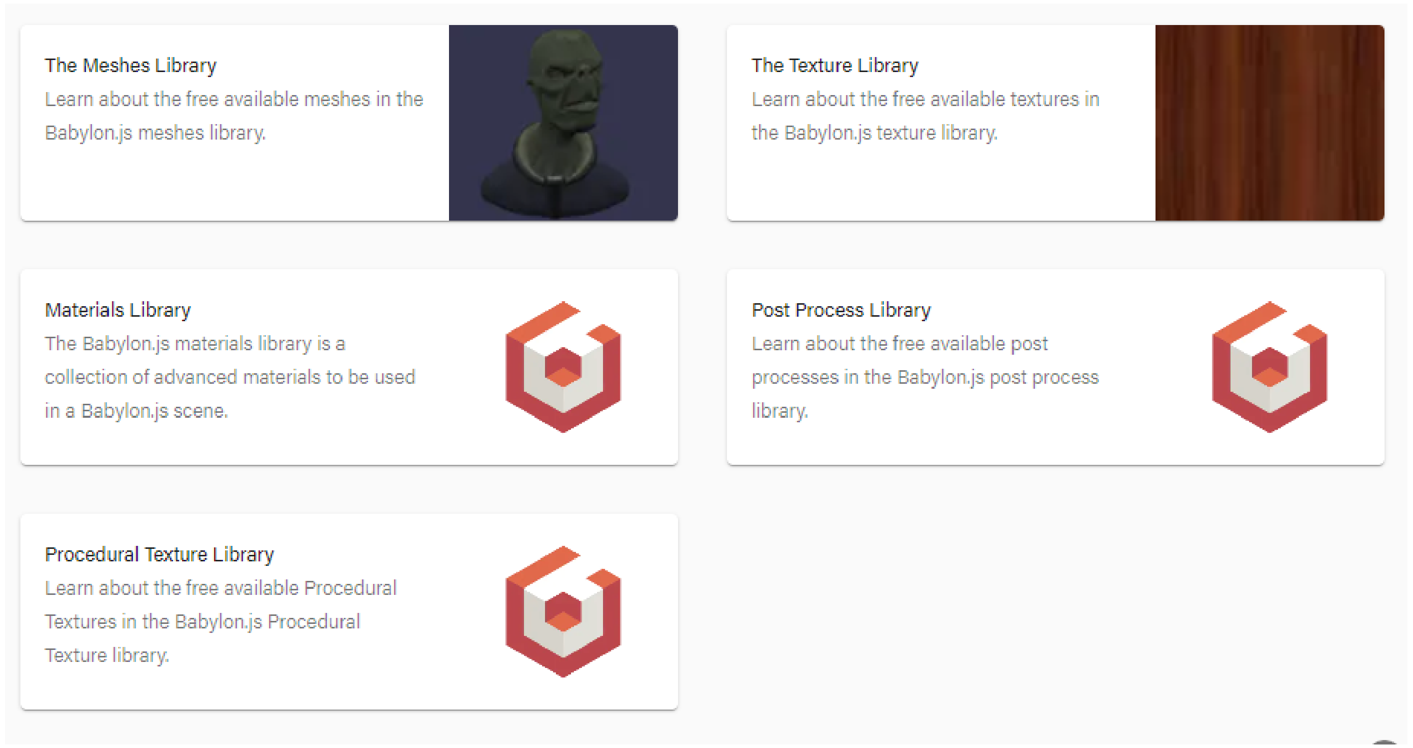
CONVERTING TO OWN

```
const box = MeshBuilder.CreateBox("box", options, scene);
```

We don’t need the BABYLON text due to our project setup!

BABYLONJS RESOURCES

- The official documentation has various resources available to you.
- Free to use, and easily accessible.
- Can be found at:
  - https://doc.babylonjs.com/toolsAndResources/assetLibraries
- You can source online too.



The Meshes Library
Learn about the free available meshes in the Babylon.js meshes library.

The Texture Library
Learn about the free available textures in the Babylon.js texture library.

Materials Library
The Babylon.js materials library is a collection of advanced materials to be used in a Babylon.js scene.

Post Process Library
Learn about the free available post processes in the Babylon.js post process library.

Procedural Texture Library
Learn about the free available Procedural Textures in the Babylon.js Procedural Texture library.

ONLINE REFERENCES

- https://doc.babylonjs.com/
- https://playground.babylonjs.com/
- https://doc.babylonjs.com/features/featuresDeepDive/physics/havokPlugin
- https://doc.babylonjs.com/features/featuresDeepDive/events
- https://doc.babylonjs.com/features/featuresDeepDive/animation
- https://doc.babylonjs.com/features/featuresDeepDive/Exporters
- https://doc.babylonjs.com/features/featuresDeepDive/input