Lesson 01 – Ball and paddle

This lesson explains how to create a level to build a base for the game. The level of the pong game includes the ball and the paddle.

To create the level, it is good to get a picture of the final result in advance. This makes it easier to determine the later positions. With Pong it should look something like this later:

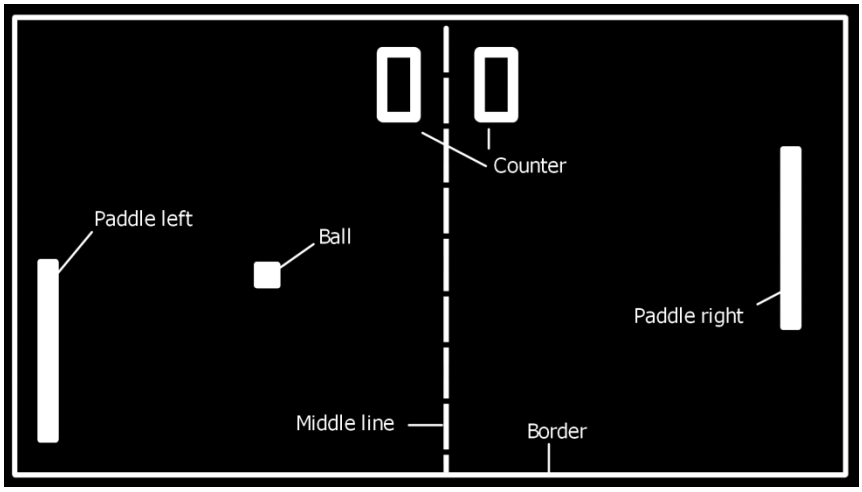

Figure 1: Example representation of Pong

The scene tree for this lesson will be structured as follows:

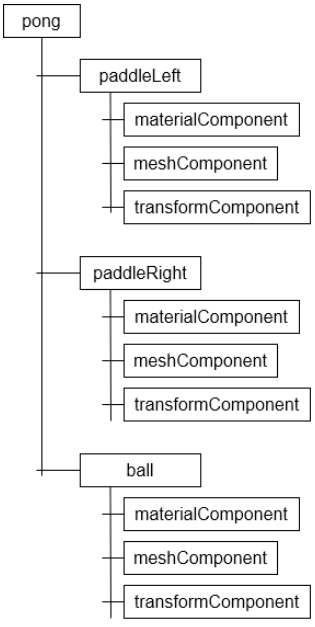

Figure 2: Scene tree

The pong node represents the level. The paddle and the ball are part of the level. *paddleLeft*, *paddleRight* and *ball* each have a *transformComponent*, a *meshComponent* and a *materialComponent*, which can be used to change the appearance and position. How these components are attached to the nodes is explained later in this lesson.

Now that we have a picture of the structure, we can start to implement the code.
We create a new project in VS Code and create the files *Main.html* and *Main.ts*. *Main.ts* is converted to a js file when saved.
First we have to prepare everything so that the level is displayed.
In *Main.ts*, all the code is written in a namespace to protect the variable names from being mixed up. Then the code is imported from *FudgeCore*, a viewport is created and an *eventListener* is attached to the browser window, which executes the code in the function *hndLoad (_event: Event)* as soon as the page has been loaded.
This function creates a canvas and initializes the *RenderManager*. In addition, a camera is created, moved to the appropriate location and thus initializes the viewport. Finally, the viewport is given the command to display everything.
Written in lines of code, it looks like this:

```
namespace Pong {
    import ƒ = FudgeCore;
    ƒ.RenderManager.initialize();
    let viewport: ƒ.Viewport;
    window.addEventListener("load", hndLoad);

    function hndLoad(_event: Event): void {
        const canvas: HTMLCanvasElement = document.querySelector("canvas");
        let cmpCamera: ƒ.ComponentCamera = new ƒ.ComponentCamera();
        cmpCamera.pivot.translateZ(42);
        cmpCamera.pivot.lookAt(ƒ.Vector3.ZERO());

        viewport = new ƒ.Viewport();
        viewport.initialize("Viewport", pong, cmpCamera, canvas);
        viewport.draw();
    }
}
```

In *Main.html* we have to write the following code:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <title>Pong</title>
    <script src="../Fudge/FudgeCore.js"></script>
    <script src="Main.js"></script>
</head>
<body>
    <h1>Pong</h1>
    <canvas width="600" height="400"></canvas>
</body>
</html>
```

It is important to ensure that the links in the two `<script>`-tags lead to the correct *Main.js* and *Main.html* documents.

The code can now be executed in the browser, but you will only see a black canvas with the heading Pong:

Figure 3: Display in the browser

We want to change this now.

We create the four nodes *pong*, *ball*, *paddleLeft* and *paddleRight*. We are also extending the *hndLoad (_event: Event)* function with the line `pong=createPong();`

With this function we want to keep *hndLoad (_event: Event)* clear and outsource everything that has to do with the creation of the game. *createNode ()* returns a node. In *createPong()* a variable *mtrSolidWhite* is created for the material and a variable *meshQuad* for a cube-shaped mesh.

Now we have to think about how to assign mesh and material to the nodes. If we call up the documentation of the Core API and take a closer look at the Node class, we see that Node knows the *addComponent* method. With this method we can add different components to a node. If we follow the link under *Parameters*, we learn what we can attach to a node - including meshes and materials.

We can add the material, the mesh and a transform component to the nodes *ball*, *paddleLeft* and *paddleRight* with the help of *addComponent*. With the Transform component, the paddles are scaled to the correct size and moved to the right and left side of the field.

In order to get the scene tree we want (see Figure 2), we have to subordinate the ball and the paddles to the node pong. With the API documentation, we can find out that Node also knows the *appendChild* method. At the end we can define *ball*, *paddleLeft* and *paddleRight* as children of *pong* and return *pong*.:

```
namespace Pong {
    import ƒ = FudgeCore;
    ƒ.RenderManager.initialize();
    window.addEventListener("load", hndLoad);
    let viewport: ƒ.Viewport;

    let ball: ƒ.Node = new ƒ.Node("Ball");
    let paddleLeft: ƒ.Node = new ƒ.Node("PaddleLeft");
    let paddleRight: ƒ.Node = new ƒ.Node("PaddleRight");

    function hndLoad(_event: Event): void {
        const canvas: HTMLCanvasElement = document.querySelector("canvas");

        let pong: ƒ.Node = createPong();

        let cmpCamera: ƒ.ComponentCamera = new ƒ.ComponentCamera();
        cmpCamera.pivot.translateZ(42);
        cmpCamera.pivot.lookAt(ƒ.Vector3.ZERO());
```

```
        paddleRight.cmpTransform.local.translateX(20);
        paddleLeft.cmpTransform.local.translateX(-20);
        paddleLeft.getComponent(ƒ.ComponentMesh).pivot.scaleY(4);
        paddleRight.getComponent(ƒ.ComponentMesh).pivot.scaleY(4);

        viewport = new ƒ.Viewport();
        viewport.initialize("Viewport", pong, cmpCamera, canvas);

        viewport.draw();
    }

    function createPong(): ƒ.Node {
        let pong: ƒ.Node = new ƒ.Node("Pong");

        let mtrSolidWhite: ƒ.Material = new ƒ.Material("SolidWhite",
            ƒ.ShaderUniColor, new ƒ.CoatColored(ƒ.Color.CSS("WHITE")));
        let meshQuad: ƒ.MeshQuad = new ƒ.MeshQuad();

        ball.addComponent(new ƒ.ComponentMesh(meshQuad));
        paddleLeft.addComponent(new ƒ.ComponentMesh(meshQuad));
        paddleRight.addComponent(new ƒ.ComponentMesh(meshQuad));

        ball.addComponent(new ƒ.ComponentMaterial(mtrSolidWhite));
        paddleLeft.addComponent(new ƒ.ComponentMaterial(mtrSolidWhite));
        paddleRight.addComponent(new ƒ.ComponentMaterial(mtrSolidWhite));

        ball.addComponent(new ƒ.ComponentTransform());
        paddleLeft.addComponent(new ƒ.ComponentTransform());
        paddleRight.addComponent(new ƒ.ComponentTransform());

        pong.appendChild(ball);
        pong.appendChild(paddleLeft);
        pong.appendChild(paddleRight);
        return pong;
    }
}
```

The browser now shows us the following image when executing the code:

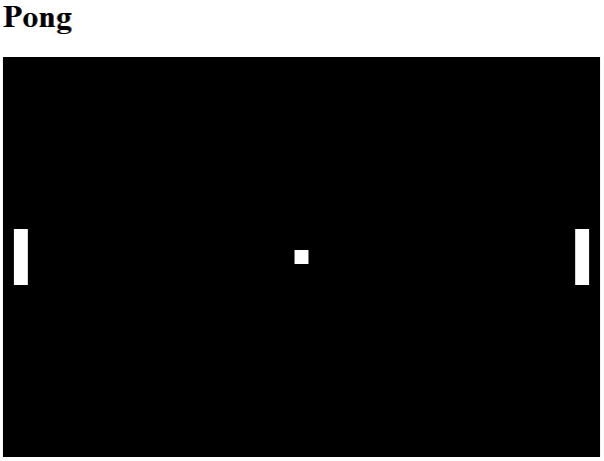

Figure 4: Display in the browser at the end of the lesson

We have thus laid the foundation for our game and created all the important components of the game - the ball and the two paddles.
Lesson 02 covers the movement of the ball and the control of the clubs.