Lesson 02 - Ball Movement and Paddle Control

The explanations and code for this lesson build on the content of Lesson 01. It explains how the canvas, the ball and the paddles are generated.
Lesson 2 explains the animation of the ball and how to move the paddles using the keyboard.

An animation is nothing more than a series of individual images in which something changes every time. So in order to represent a movement of the ball or the paddles, we not only have to change the positions, but also tell the code to show the changes - so we need a loop that executes the code repeatedly.
In the documentation of the Core API we find an entry "Loop". After a short scroll we find out again that we can add an EventListener to *Loop* with *addEventListener* and pass a function that is executed again and again after the loop has started.
So we can add a function *update(_event: Event)* to our code and pass it to *Loop*. *update(_event: Event)* repeatedly calls *viewport.draw*, which means that the canvas image is always redrawn and the changes are shown.

```
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();

        ƒ.Loop.addEventListener(ƒ.EVENT.LOOP_FRAME, update);
        ƒ.Loop.start();
    }

    function update(_event: Event): void {
        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;
    }
}
```

Now we can start moving the ball. To do this, we always have to change its position with the help of *transformComponent*. It is best to create a variable called *ballSpeed* that represents the speed of the ball. For the sake of clarity, we carry out the position change in a function *moveBall*, which we call in turn in *update(_event: Event)*.

```
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");

    let ballSpeed: ƒ.Vector3 = new ƒ.Vector3(0.1, -0.1, 0);

    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();

        ƒ.Loop.addEventListener(ƒ.EVENT.LOOP_FRAME, update);
        ƒ.Loop.start();
    }

    function update(_event: Event): void {
        moveBall();
        viewport.draw();
    }

    function moveBall(): void {
        ball.cmpTransform.local.translate(ballSpeed);
    }

    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;
    }
}
```

If we now run the code in the browser, we can see the ball moving towards the lower right corner.

Figure 1: The ball moves into the lower right corner

We need to add a few more lines of code to control the paddles.
First, let's do some theoretical preliminary thinking about the control:
In order to be able to play on a computer for two, the left paddle should be controlled with the W and S keys and the right paddle with the up and down arrow keys. It is important that several keys can be pressed at the same time.
The keystroke is determined via two eventListeners, which are added to the document-object and call the *hndKeydown* function while the key is pressed and the *hndKeyup* function when the key is released.
For better readability and comprehensibility, it is recommended to create an associative array *KeysPressed* in order to access the boolean states of the four keys stored in it by strings. This requires an interface with which the array is typed.
In the end, we have to check in the *update function* with the help of if queries whether a certain key was pressed. If so, the paddle should move in the appropriate direction.

It is important to use four individual *if statements* instead of one *if-else statement* so that the code can respond when multiple keys are pressed at the same time.

```
namespace Pong {
    interface KeyPressed {
        [code: string]: boolean;
    }

    import ƒ = FudgeCore;
    ƒ.RenderManager.initialize();
    let keysPressed: KeyPressed = {};
    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");

    let ballSpeed: ƒ.Vector3 = new ƒ.Vector3(0.1, -0.1, 0);

    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);
```

```
        document.addEventListener("keydown", hndKeydown);
        document.addEventListener("keyup", hndKeyup);

        viewport.draw();

        ƒ.Loop.addEventListener(ƒ.EVENT.LOOP_FRAME, update);
        ƒ.Loop.start();
    }

    function update(_event: Event): void {
        if (keysPressed[ƒ.KEYBOARD_CODE.ARROW_UP])
          paddleRight.cmpTransform.local.translate(new ƒ.Vector3(0,0.3,0));

        if (keysPressed[ƒ.KEYBOARD_CODE.ARROW_DOWN])
          paddleRight.cmpTransform.local.translate(ƒ.Vector3.Y(-0.3));

        if (keysPressed[ƒ.KEYBOARD_CODE.W])
          paddleLeft.cmpTransform.local.translate(new ƒ.Vector3(0,0.3,0));

        if (keysPressed[ƒ.KEYBOARD_CODE.S])
            paddleLeft.cmpTransform.local.translate(ƒ.Vector3.Y(-0.3));

        moveBall();
        viewport.draw();
    }

    function moveBall(): void {
        ball.cmpTransform.local.translate(ballSpeed);
    }

    function hndKeyup(_event: KeyboardEvent): void {
        keysPressed[_event.code] = false;
    }
    function hndKeydown(_event: KeyboardEvent): void {
        keysPressed[_event.code] = true;
    }
```

```
    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;
    }
}
```

We can now act with the paddles and the ball moves through the picture. The next step is to ensure that the ball bounces off the paddles. Then we can already play with our game.

If you want, you can change the code so that the ball flies in a random direction at the start of the game instead of just to the bottom right.