Lesson 04 – Collision

After we have theoretically prepared in lesson 3, we now want to program the collision query.

First we need a function *detectHit*, which takes a Vector3 *_position* and a node *_node* and returns a Boolean value at the end.

As we noted in the previous lesson, the collision should be recognized when the center of the ball is within a rectangle. Looking at the documentation of the Core API, we can find out that the Rectangle class practically knows an *isInside* method. *isInside* returns *true* if a point lies inside or on the edge of a rectangle. Before using the method, we need to save a copy of the scaling of the passed node and we need to invert *_position*. From the scaling, we create a new rectangle, to which we apply *isInside*. The boolean value is returned.

```
function detectHit(_position: ƒ.Vector3, _node: ƒ.Node): boolean {
    let sclRect: ƒ.Vector3 = _node.getComponent
        (ƒ.ComponentMesh).pivot.scaling.copy;
    let mtxInverse: ƒ.Matrix4x4 = ƒ.Matrix4x4.INVERSION
        (_node.cmpTransform.local);
    _position.transform(mtxInverse);
    let rect: ƒ.Rectangle = new ƒ.Rectangle
        (0, 0, sclRect.x, sclRect.y, ƒ.ORIGIN2D.CENTER);
    return rect.isInside(_position.toVector2());
}
```

Next we have to write a function that says what should happen in a collision. This function is called *processHit* and accepts a node *_node*. The function contains a switch-case that uses the name of the node as the key. In the event of a collision on the top or bottom wall, the x value of the ball speed must be inverted. If the ball hits a paddle, the y value must be reversed. For the walls on the left and right, the y value is currently also being changed. A point count or a change in speed if a paddle was hit can be integrated here later.

```
function processHit(_node: ƒ.Node): void {
        console.log("Reflect at ", _node.name);
        switch (_node.name) {
            case "WallTop":
            case "WallBottom":
                ballSpeed.y *= -1;
                break;
            case "WallRight":
            case "WallLeft":
            case "PaddleLeft":
            case "PaddleRight":
                ballSpeed.x *= -1;
                break;
            default:
                console.warn("Oh, no, I hit something unknown!!",
                        _node.name);
                break;
        }
    }
```

In the *update* function we create a boolean variable *hit*. With the help of a for loop, we can go through all pong child nodes. If the child node is not the ball, then the position of the ball and the child node should be passed to *detectHit* and the returned value saved in hit. If *hit* is true, *processHit* should be executed.

```
    function update(_event: Event): void {
        processInput();
        let hit: boolean = false;
        for (let node of pong.getChildren()) {
            if (node.name == "Ball")
                continue;

            hit = detectHit(mtxBall.translation, node);

            if (hit) {
                processHit(node);
                break;
            }
        }
        moveBall();
        viewport.draw();
    }
```

At the end, we can add a centre line to our playing field. For this we need a variable of the type *CanvasRenderingContext2D* and save the method call *canvas.getContext ("2d")* inside this variable. Now we can change the color, line style and some other things.

Here is the full code for this lesson:

```
namespace Pong {
    import ƒ = FudgeCore;

    interface KeyPressed {
        [code: string]: boolean;
    }

    interface Control {
        paddle: ƒ.Node;
        translation: ƒ.Vector3;
    }

    interface Controls {
        [code: string]: Control;
    }


    let viewport: ƒ.Viewport;

    let pong: ƒ.Node;
    let ball: ƒ.Node;
    let mtxBall: ƒ.Matrix4x4;
    let paddleLeft: ƒ.Node;
    let paddleRight: ƒ.Node;
    let keysPressed: KeyPressed = {};
    let ballSpeed: ƒ.Vector3 = new ƒ.Vector3(0.1, -0.1, 0);
    let paddleSpeedTranslation: number = 0.5;
    let controls: Controls;

    let crc2: CanvasRenderingContext2D;

    window.addEventListener("load", hndLoad);

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

```
        crc2 = canvas.getContext("2d");

        pong = createPong();
        controls = defineControls();
        mtxBall = ball.cmpTransform.local;

        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);

        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 {
        processInput();
        let hit: boolean = false;
        for (let node of pong.getChildren()) {
            if (node.name == "Ball")
                continue;

            hit = detectHit(mtxBall.translation, node);

            if (hit) {
                processHit(node);
                break;
            }
        }

        moveBall();

        viewport.draw();

        crc2.strokeStyle = "white";
        crc2.lineWidth = 4;
```

```
        crc2.setLineDash([10, 10]);
        crc2.moveTo(crc2.canvas.width / 2, 0);
        crc2.lineTo(crc2.canvas.width / 2, crc2.canvas.height);
        crc2.stroke();
    }

    function detectHit(_position: ƒ.Vector3, _node: ƒ.Node): boolean {
        let sclRect: ƒ.Vector3 = _node.getComponent
            (ƒ.ComponentMesh).pivot.scaling.copy;
        let mtxInverse: ƒ.Matrix4x4 = ƒ.Matrix4x4.INVERSION
            (_node.cmpTransform.local);
        _position.transform(mtxInverse);
        let rect: ƒ.Rectangle = new ƒ.Rectangle
            (0, 0, sclRect.x, sclRect.y, ƒ.ORIGIN2D.CENTER);
        return rect.isInside(_position.toVector2());
    }

    function processHit(_node: ƒ.Node): void {
        console.log("Reflect at ", _node.name);
        switch (_node.name) {
            case "WallTop":
            case "WallBottom":
                ballSpeed.y *= -1;
                break;
            case "WallRight":
            case "WallLeft":
            case "PaddleLeft":
            case "PaddleRight":
                ballSpeed.x *= -1;
                break;
            default:
                console.warn("Oh, no, I hit something unknown!!",
                        _node.name);
                break;
        }
    }

    function processInput(): void {
        for (let code in controls) {
            if (keysPressed[code]) {
                let control: Control = controls[code];
                let mtxPaddle: ƒ.Matrix4x4 =
                        control.paddle.cmpTransform.local;
                mtxPaddle.translation = ƒ.Vector3.SUM
```

```
                    (mtxPaddle.translation, control.translation);
            }
        }
    }

    function defineControls(): Controls {
                let controls: Controls = {};
        controls[ƒ.KEYBOARD_CODE.ARROW_UP] = { paddle: paddleRight,
            translation: ƒ.Vector3.Y(paddleSpeedTranslation)};

        controls[ƒ.KEYBOARD_CODE.ARROW_DOWN] = { paddle: paddleRight,
            translation: ƒ.Vector3.Y(-paddleSpeedTranslation)};

        controls[ƒ.KEYBOARD_CODE.W] = { paddle: paddleLeft,
            translation: ƒ.Vector3.Y(paddleSpeedTranslation)};

        controls[ƒ.KEYBOARD_CODE.S] = { paddle: paddleLeft,
            translation: ƒ.Vector3.Y(-paddleSpeedTranslation) };
        return controls;
    }

    function moveBall(): void {
        mtxBall.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();

        pong.appendChild(createNode("WallLeft", meshQuad, mtrSolidWhite,
```

```
            new ƒ.Vector2(-22, 0), new ƒ.Vector2(1, 30)));
        pong.appendChild(createNode("WallRight", meshQuad, mtrSolidWhite,
            new ƒ.Vector2(22, 0), new ƒ.Vector2(1, 30)));
        pong.appendChild(createNode("WallTop", meshQuad, mtrSolidWhite,
            new ƒ.Vector2(0, 15), new ƒ.Vector2(45, 1)));
        pong.appendChild(createNode("WallBottom", meshQuad, mtrSolidWhite,
            new ƒ.Vector2(0, -15), new ƒ.Vector2(45, 1)));

        ball = createNode("Ball", meshQuad, mtrSolidWhite,
            ƒ.Vector2.ZERO(), new ƒ.Vector2(1, 1));
        paddleLeft = createNode("PaddleLeft", meshQuad, mtrSolidWhite,
            new ƒ.Vector2(-20, 0), new ƒ.Vector2(1, 4));
        paddleRight = createNode("PaddleRight", meshQuad, mtrSolidWhite,
            new ƒ.Vector2(20, 0), new ƒ.Vector2(1, 4));

        pong.appendChild(ball);
        pong.appendChild(paddleLeft);
        pong.appendChild(paddleRight);
        return pong;
    }
    function createNode(_name: string, _mesh: ƒ.Mesh,_material: ƒ.Material,
      _translation: ƒ.Vector2, _scaling: ƒ.Vector2): ƒ.Node {
        let node: ƒ.Node = new ƒ.Node(_name);
        node.addComponent(new ƒ.ComponentTransform);
        node.addComponent(new ƒ.ComponentMaterial(_material));
        node.addComponent(new ƒ.ComponentMesh(_mesh));
        node.cmpTransform.local.translate(_translation.toVector3());
        node.getComponent(ƒ.ComponentMesh).pivot.scale
            (_scaling.toVector3());
        return node;
    }
}
```

If you want, you can now add a point display to the game.