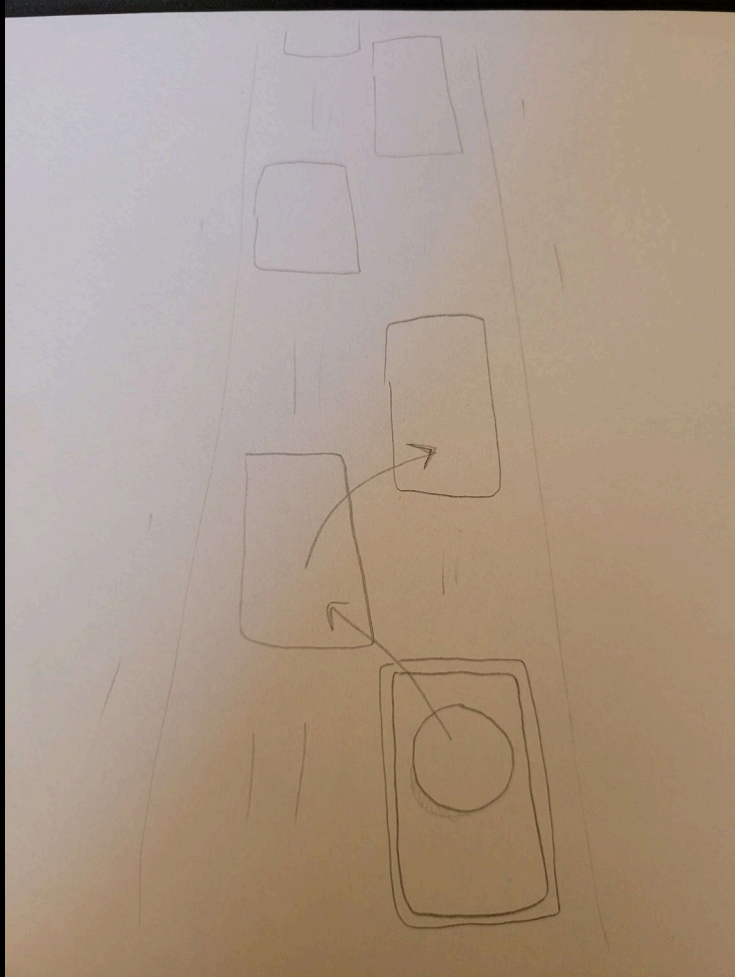




Bouncy Ball - Tiles Jumper

Prima Dokumentation
Rebecca Chemata OMB 6

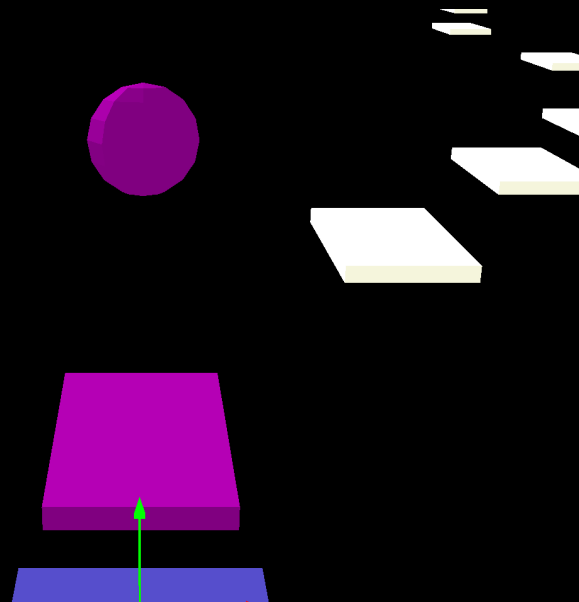
Steuerung



- Der Ball/Avatar wird von von Tile zu Tile bewegt
- Die Richtung, in die der Avatar sich bewegt, wird durch Mausbewegungen nach links oder rechts gesteuert.
- Ziel ist es einen hohen Score zu erzielen und dabei nicht von den Tiles zu fallen

1. Units and Positions

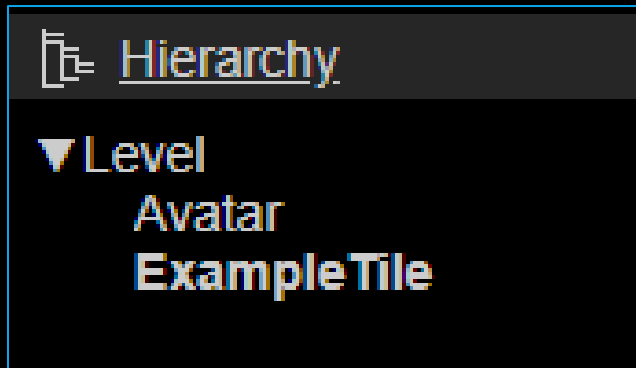
- 3D Game
- 0: Die Position des Balls auf dem ersten Tile
- 1: Das nächste Tile entweder auf der x- oder z-Achse entlang
- Dabei gibt die z-Achse die Tonhöhe an, der Abstand der Tiles die Tonlänge
- Der Ball bewegt sich dabei konstant in x-Richtung, nur die Sprungkraft wird währenddessen immer angepasst, damit das nächste Tile getroffen werden kann.



2. Hierarchie

Die Hierarchie ist sehr einfach gehalten:

- Das Basic Setup im Level-Graph mit Licht und Kamera
- Nodes: Avatar (Ball) und ExampleTile ohne children



- *Level*
 - *ComponentLight*
 - *ComponentLight*
 - *ComponentCamera*
- *Avatar*
 - *ComponentMesh*
 - *ComponentMaterial*
 - *ComponentTransform*
 - *ComponentRigidbody*
 - *ComponentAnimator*
 - *BallBouncer{}*
- *ExampleTile*
 - *ComponentMesh*
 - *ComponentMaterial*
 - *ComponentTransform*
 - *ComponentRigidbody*

3. Editor

- Der Editor wurde für das Basic Setup benutzt, also Camera und Licht für das Level
- Der Ball als Avatar Node im Editor erleichtert das Hinzufügen von Komponenten wie ComponentRigidbody, ComponentScripts etc.
- Das Example Tile diene einzig und allein dem Zweck, um das richtige Scaling für die restlichen Tiles zu finden
Das kann also genauso gut auch weggelassen werden
- Der Editor wurde außerdem auch für die Bouncing Ball Animation benutzt

4.ScriptComponents

- Es wurde ein ScriptComponent benutzt für den Avatar Node
 - > BallBouncer{}
- Wie der Name schon sagt, sorgt das Script dafür, dass der Ball in nem ständigen Loop hoch und runter springt
- Da dafür aber die zwei Variablen „rigidbodyAvatar“ und „isGrounded“ exportieren mussten, hätte man das genauso gut auch in der Main.ts ohne ScriptComponent regeln können

```
public update = (_event: Event): void => {  
  if (isGrounded) {  
    let material: f.ComponentMaterial = this.node.getComponent(f.ComponentMaterial);  
    let color: string = f.Random.default.getElement(["purple"]);  
    material.clrPrimary = f.Color.CSS(color);  
  
    rigidbodyAvatar.addVelocity(f.Vector3.Y(7));  
    isGrounded = false;  
  }  
}
```

Components Avatar				
▶	ComponentMesh			
▶	ComponentMaterial			
▼	ComponentTransform			
	active			
	mtxLocal	x	y	z
	translation	+ 0.000e+0	+ 0.600e+0	- 4.200e+0
	rotation	+ 0.000e+0	+ 0.000e+0	+ 0.000e+0
	scaling	+ 1.000e+0	+ 1.000e+0	+ 1.000e+0
▶	ComponentRigidbody			
▶	BallBouncer			
▶	ComponentAnimator			

5.Extend

- Es wurde eine Tiles Klasse benutzt, um beliebig viele Instanzen derselben Klasse zu generieren
- So konnte jedem Tile eine Notenhöhe, eine Notenlänge, eine Sprungkraft für den Ball, eine Frequenz, eine Position und eine Farbe zugeordnet werden
- Das erleichterte das generieren der Tiles und die Aktionen, die durch die Kollision mit dem Ball ausgeführt wurden erheblich

```
export class Tile extends f.Node{
    static meshTile: f.MeshCube = new f.MeshCube("Tile");
    static mtrTile: f.Material = new f.Material("Tile", f.ShaderFlat, new f.CoatRemissive());
    pitch: string;
    length: string;
    jumpforce: number;
    frequency: number;

    constructor(pitch: string, length: string, jumpforce:number, frequency:number, _position: f.Vector3, _color: f.Color){
        super("Tile");
        this.pitch = pitch;
        this.length = length;
        this.jumpforce = jumpforce;
        this.frequency = frequency;

        this.addComponent(new f.ComponentMesh(Tile.meshTile));

        let cmpMaterial: f.ComponentMaterial = new f.ComponentMaterial(Tile.mtrTile);
        cmpMaterial.clrPrimary = _color;
        this.addComponent(cmpMaterial);

        let cmpTransform: f.ComponentTransform = new f.ComponentTransform(f.Matrix4x4.TRANSLATION(_position));
        this.addComponent(cmpTransform);

        let cmpPicker: f.ComponentPick = new f.ComponentPick();
        cmpPicker.pick = f.PICK.RADIUS;
        this.addComponent(cmpPicker);

        let cpmRigidbody: f.ComponentRigidbody = new f.ComponentRigidbody(1, f.BODY_TYPE.STATIC, f.COLLIDER_TYPE.CUBE );
        this.addComponent(cpmRigidbody);
    }
}
```

6.Sounds

- Es wurde die Web Audio API benutzt, um Sinuswellen von bestimmten Frequenzen zu erzeugen, die von der Notenhöhe des Tiles abhängen
- So wird nach jeder Kollision des Balls mit einem Tile ein bestimmter Sinuston erzeugt

```
{  
  "pitch": "E",  
  "length": "1/4",  
  "jumpforce": -3,  
  "frequency": 329.63  
},
```

Note: E

Frequency: 329,63

```
function generateTone(frequency: number, duration: number) {  
  const oscillator = audioContext.createOscillator();  
  const gainNode = audioContext.createGain();  
  
  oscillator.type = 'sine';  
  oscillator.frequency.value = frequency;  
  
  oscillator.connect(gainNode);  
  gainNode.connect(audioContext.destination);  
  
  const releaseTime = 0.5;  
  const currentTime = audioContext.currentTime;  
  
  gainNode.gain.setValueAtTime(1, currentTime);  
  gainNode.gain.linearRampToValueAtTime(0, currentTime + releaseTime);  
  oscillator.start();  
  
  oscillator.stop(audioContext.currentTime + duration);  
}
```


7.VUI

- Mithilfe des VUI kann der User jederzeit seinen Score, die aktuelle Notenhöhe und die passende Frequenz des gerade berührten Teils sehen

Score: 2

Note: E

Frequency: 329,63

```
export class GameState extends f.Mutable {  
  public score: number;  
  public note: string;  
  public frequency: number;  
  
  constructor() {  
    super();  
    this.score = 0;  
    this.note = "";  
    this.frequency = this.frequency;  
  
    let vui: HTMLDivElement = document.querySelector("div#vui");  
    new fUi.Controller(this, vui);  
    this.addEventListener(f.EVENT.MUTATE, () => console.log(this));  
  }  
  protected reduceMutator(_mutator: f.Mutator): void {  
  }  
}
```

8.Event-System

- Es wurde ein Custom Event benutzt
- Die Funktion „avatarCollided()“ wird aufgerufen, sobald der Ball mit einem Tile kollidiert
- So kann die Gamestate, die Tilefarbe, die Animation, die Sprungkraft des Balls ,die richtige Frequenz etc. im richtigen Moment angepasst werden

```
function avatarCollided(): void {
    isGrounded = true;

    //generate Custom Event vor Collision
    let customEvent: CustomEvent = new CustomEvent(BOUNCYBALL.AVATAR_COLLIDES, { bubbles: true, detail: avatar.mtxWorld.translation });

    avatar.dispatchEvent(customEvent);

    //Update Gamestate
    score++;
    gamestate.score = score;
    gamestate.note = tileList[score].pitch;
    gamestate.frequency = tileList[score].frequency;

    let material: f.ComponentMaterial = tileList[score].getComponent(f.ComponentMaterial);
    let rigidbodyTile: f.ComponentRigidbody = tileList[score].getComponent(f.ComponentRigidbody);
    let animation: f.ComponentAnimator = avatar.getComponent(f.ComponentAnimator);

    //Play and Stop Animation
    animation.playmode = f.ANIMATION_PLAYMODE.LOOP;
    console.log(animation.time);
    setTimeout(() => {
        animation.playmode = f.ANIMATION_PLAYMODE.STOP;
    }, 200);

    //Adjust Jumpforce, Color and Tile TypeBody
    jumpforce = tileList[score].jumpforce;
    material.clrPrimary = f.Color.CSS("purple");
    rigidbodyTile.typeBody = f.BODY_TYPE.DYNAMIC;

    //Generate Tone Audio
    generateTone(tileList[score].frequency, 1);
}
```

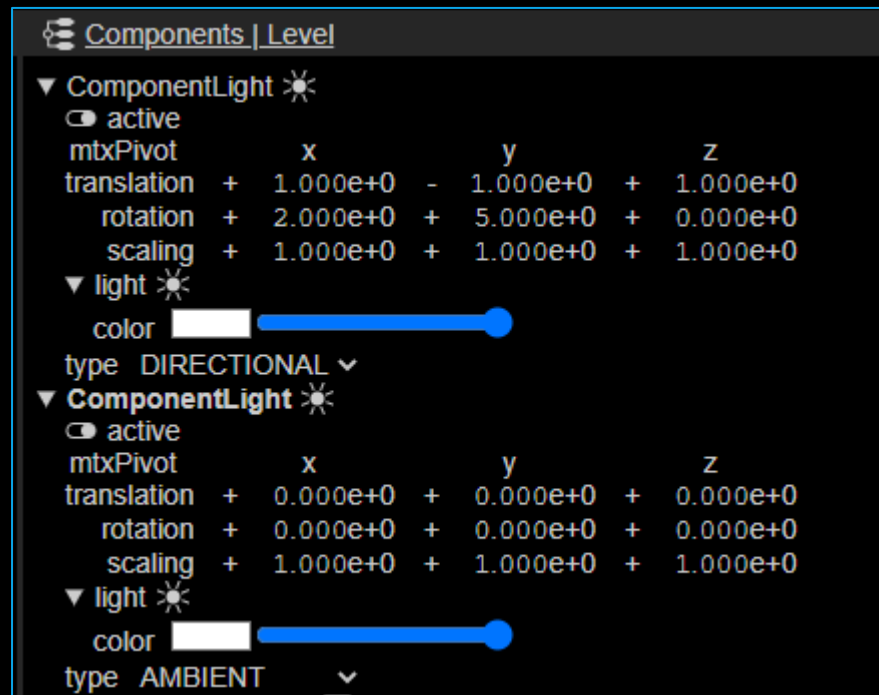
9.External Data

- In der config.json sind alle Tiles dokumentiert mit ihren Eigenschaften
- Dies erleichtert nicht nur den Aufbau des Levels, sondern auch das Benutzen der Eigenschaften bei jedem Collide

```
{
  "tiles": [
    {
      "pitch": "C",
      "length": "1/4",
      "jumpforce": -3,
      "frequency": 261.63
    },
    {
      "pitch": "D",
      "length": "1/4",
      "jumpforce": -3,
      "frequency": 293.66
    },
    {
      "pitch": "E",
      "length": "1/4",
      "jumpforce": -3,
      "frequency": 329.63
    },
    {
      "pitch": "F",
      "length": "1/4",
      "jumpforce": -9,
      "frequency": 349.23
    }
  ]
}
```

10.Light

- Wird benötigt, um die Komponenten zu sehen beim 3D Game
- Es wurde ein Ambient Light für das Umgebungslicht und ein Directional Light für „die Sonne“ benutzt



11.Physics

- Mithilfe von Physic Forces und Velocity wird der Ball via Mouse Bewegung gesteuert
- Die Schwerkraft erleichtert das bouncen
- Es wurde mit Kollisions gearbeitet

```
f.Physics.simulate();  
cameraMover();  
avatarPos = rigidbodyAvatar.node.mtxLocal.translation;  
rigidbodyAvatar.applyForce(f.Vector3.Z(jumpforce));
```

```
rigidbodyAvatar.addVelocity(f.Vector3.Y(7));|
```

```
rigidbodyAvatar.addEventListener(f.EVENT_PHYSICS.COLLISION_ENTER, avatarCollided);|
```

12.Animation

- Es wurde das Animation System von FudgeCore im Editor benutzt, um eine „Bouncing Ball Animation“ zu simulieren
- Das klappt mehr oder weniger, da der Ball selbst während des Sprungs rotiert und die Animation deshalb manchmal nicht passt



```
animation.playmode = f.ANIMATION_PLAYMODE.LOOP;  
console.log(animation.time);  
setTimeout(()=> {  
  animation.playmode = f.ANIMATION_PLAYMODE.STOP;  
}, 200);
```

Links:

- **Spiel:**

https://rebeccachemata13.github.io/PRIMA_SoSe_23/Bouncy_Ball_Tile_Jumper/index.html

- **Code :**

https://github.com/rebeccachemata13/PRIMA_SoSe_23/tree/main/Bouncy_Ball_Tile_Jumper