Dev Diary - Serious Games Projekt

Von Ella Pohl und Ruben Härle (Gruppe 33)

Link

Link zum Spiel

Aufgabe 1

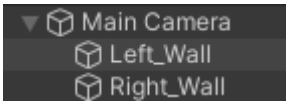
Zeitraum:

- 23.Mai – 30.Mai

Aufgaben:

Wir beide haben Unity installiert und angefangen die Entwicklungsumgebung zu erkunden. Dabei haben wir festgestellt, dass es verschiedene Templates für verschiedene Arten von Spielen gibt, zum Beispiel 3D Welten, oder 2D Runner Welten. Wir haben uns für eine plane 2D Welt entschieden, da dies am besten zu unserer Aufgabe gepasst hat. Dabei haben wir uns für unsere Erkundung stark an den bereitgestellten Links aus der Aufgabenstellung orientiert.

Als nächstes haben wir uns dazu entschieden ganz einfach anzufangen. Das heißt, wir haben als Bohrer ein Dreieck genommen und als Begrenzungen für die rechte und linke Seite zwei statische Rechtecke. Damit die Begrenzungen nicht aus dem Bild verschwinden, haben wir sie mit unserer Hauptkamera verbunden. So bewegen sich die Rechtecke immer dahin wo auch die Kamera hinbewegt wird.



Als nächstes haben wir die Kamera mit ihrer y-Achse mit einem Skript an die Drill gekoppelt, damit wir der Drill immer folgen können. Bis jetzt ist der x-Achsen Wert der Drill ein fester Wert, soll aber in späteren Aufgaben überschrieben werden können. Aktuell fällt die Drill einfach nach unten und die Kamera folgt ihr.

```
1 // Camera default following the drill only on y direction
2 camera.transform.position = new Vector3(0, transform.position.y -
  centerOffsetCamera, -10);
```

Unsere letzte Aufgabe für diese Woche war die Implementierung eines Menüs. Hierfür haben wir eine neue Szene angelegt in der wir einen Canvas angelegt haben in den wir Text und einen Button hinzugefügt haben, um das Spiel zu starten. Außerdem haben wir noch einen Platzhalter für einen Highscore hinzugefügt.

Probleme:

Zu Anfang sind unsere Bildbegrenzungen nicht auf Höhe des Bohrers bzw. der Kamera geblieben. Nach einiger Recherche konnten wir herausfinden, dass man in der Hierarchie Objekte untereinander anordnen kann, was dazu führte, dass die Begrenzungen auf Höhe der Kamera und Bohrers blieben, da die Koordinaten dann relativ zum Parent-Objekt gesetzt werden.

Ein anderes Hindernis war die Implementierung des Menüs. Zu Anfang versuchten wir alles in einer Szene zu implementieren. Dies erwies sich jedoch als sehr schwierig. Deshalb entschieden wir uns eine separat Szene zu verwenden, und dort eine Transition zu einer anderen Szene mit Hilfe eines Buttons zu implementieren.

```
public class Menu : MonoBehaviour
{
    [SerializeField] private TMP_Text textUI;

    public void Start(){
        textUI.text = "Highscore:"; // here we will add the value of the highscore
    }

    public void move_to_scene(string scene_name)
    {
        SceneManager.LoadScene(scene_name);
    }
}
```

Aufgabe 2

Zeitraum:

- 30.Mai – 6.Juni

Aufgaben:

Für die Steuerung des Bohrers haben wir einige mathematische Funktionen von C# verwendet. Die Kurvenrichtung wird durch einen positiven bzw. negativen Faktor von 1 bzw. -1 für rechts bzw. links gesteuert. Um eine schönere Kurvenbewegung zu bekommen interpolieren wir anhand aktueller Geschwindigkeit, Richtung und Rotationsgeschwindigkeit die neue Bewegungsrichtung. Dies wird nun in die Rotation des Bohrers übersetzt. Anschießend setzten wir die aktuelle Geschwindigkeit, da diese sich in unterschiedlichen Erdschichten ändern kann. Unser "One Button" ist die Space-Taste, welchen wir im Inspektor festgelegt haben.

```
if(Input.GetKeyDown(Right))
        { moveDirection = 1;
        }else if(Input.GetKeyUp(Right))
        { moveDirection = -1; }
// Linear Interpolation for smoother rotation
currentMoveDirection = Mathf.Lerp(
        currentMoveDirection, moveDirection, rotation_speed*Time.deltaTime);
// Rotation angle
transform.rotation = Quaternion.Euler(0, 0, default_angle*currentMoveDirection);
// Drill going locally down
RB.velocity = (transform.up * -1) * MoveSpeed;
```

Damit unsere Drill nicht rechts und links aus dem Bild laufen kann, haben wir jeweils einen *Box Collider 2D* an den Rändern angebracht. Nach oben kann der Bohrer nicht aus dem Bild laufen, da der Bewegungsradius immer so gewählt wird, dass der Bohrer keine 180° Wendung machen kann. Des weiteren kann die Kamera dem Bohrer nur in y-Achsen Richtung folgen.

Als Implementierung für Hindernisse haben wir uns für 3 Arten entschieden. Ein größeres unbewegliches, ein kleines Bewegliches und ein kleines unbewegliches. Das bewegliche Hindernis wandert dabei immer von rechts nach links oder umgekehrt. Das bewegliche Hindernis ist als einziges der drei Hindernisse nicht in der x-Achse beschränkt, weshalb wir über ein Skript den Bewegungsfreiraum definieren. Die Unbeweglichkeit wurde durch Check-boxen im Inspektor gesetzt.

```
// Bewegungsfreiraum bewegliches Hindernis
void Update () {
    if(transform.position.x >= 7.3){
        speed = -speed;
    } else if(transform.position.x <= -7.4){
        speed = -speed;
    }
    myBody.velocity = new Vector2 (speed, 0);
}
```

Berührt der Bohrer ein Hindernis ist er anfangs leicht rot und färbt sich bei einer zweiten Berührung noch stärker rot. Bei der dritten ist das Spiel vorbei und der Spieler gelangt zurück ins Hauptmenü. Nach einem Zusammenstoß wird das entsprechende Hindernis zerstört.

```
Color red_light = new Color(1f, 0.5f, 0.5f); // leicht rot
Color red_dark = new Color(1f, 0f, 0f); // stark rot
if (collision.gameObject.CompareTag("Obstacle")){
    counter += 1;
    // change color of drill after collision
    if (counter == 1) Drill.material.color = red_light;
    if (counter == 2) Drill.material.color = red_dark;
    Destroy(collision.gameObject); // destroy obstacle
    if (counter == 3) SceneManager.LoadScene("Menu");
    ...
}
```

Das Spiel wird außerdem beendet wenn der Spieler den Erdkern erreicht. In unserem Fall ist dies eine Tiefe von 600 Scorepunkten.

```
...
else if((erdkern - score) == 0){
    // go back to menu
    SceneManager.LoadScene("Menu");
}
```

Aufgabe 3

Zeitraum:

- 6.Juni – 13.Juni

Aufgaben:

Um den Score zu berechnen machen wir uns die y-Koordinaten zu nutze. Hierbei subtrahieren wir die aktuelle y-Position von der Start y-Position und erhalten so einen Score. Um einen ganzzahligen Score zu erhalten, mussten wir die Position runden und anschließen in einen Integer umwandeln.

```
// calculate score
current = (int)(transform.position.y + 0.5f);
score = (int)(start+0.5f) - current;
```

Wir haben für den Score ein UI Textfeld angelegt. Der Wert im Textfeld wird mit der Update-Funktion des Drill-Skriptes auf den aktuellen Wert gesetzt. Das Textfeld haben wir im Code als Serialized definiert, so können wir darauf im Inspektor zugreifen und das entsprechende Textfeld setzten.

```
[SerializeField] private TMP_Text textUI;
...
textUI.text = "Score:" + score;
```

Um den Score über mehrere Szenen also mehrere Spiele hinweg zu speichern, haben wir einen Game-Manager verwendet. In dem Game-Manager ist besonders wichtig, das dieser beim laden einer neuen Szene nicht vernichtet / gelöscht wird. Das erreichen wir mit Zeile 12 unten:

```
public class GameManager : MonoBehaviour
{
    public static GameManager Instance;
    public int highscore = 0;

    void Awake(){
        if(Instance == null){
            Instance = this;
        }else if(Instance != this){
            Destroy(this.gameObject);
        }
        DontDestroyOnLoad(this.gameObject);
```

```
    }
}
```

Um etwas Abwechslung in den Hintergrund zu bringen, haben wir uns für 5 verschiedene Hintergründe entschieden. Wechselt ein Hintergrund, so wechselt sich auch die Geschwindigkeit und Rotationswinkel des Bohrers, dafür hat jede Erdschicht einen *BoxColider 2D* mit der Eigenschaft `isTrigger=True` bekommen. Für das Setzen von einer der 5 Konfigurationen rufen wir in dem Drill-Skript die Funktion `OnTriggerEnter2D` auf. Jede Erdschicht hat dazu einen Tag bekommen, um sie eindeutig bestimmen zu können. Hierfür haben wir 5 Einstellungen getroffen:

```
private void OnTriggerEnter2D(Collider2D other) {
    if (other.gameObject.CompareTag("Erdschicht1")){
        Debug.Log("Erdschicht1");
        default_angle = 20;
        MoveSpeed = 8;
    }else if(other.gameObject.CompareTag("Erdschicht2")){
        Debug.Log("Erdschicht2");
        default_angle = 40;
        MoveSpeed = 10;
    }else if(other.gameObject.CompareTag("Erdschicht3")){
        Debug.Log("Erdschicht3");
        default_angle = 50;
        MoveSpeed = 10;
    }else if(other.gameObject.CompareTag("Erdschicht4")){
        Debug.Log("Erdschicht4");
        default_angle = 10;
        MoveSpeed = 13;
    }else if(other.gameObject.CompareTag("Erdschicht5")){
        Debug.Log("Erdschicht5");
        default_angle = 20;
        MoveSpeed = 14;
    }
}
```

Damit die Hindernisse unberechenbar bleiben, spawnen wir die Hindernisse an zufälligen Stellen und in zufälligen Arten. Die Hindernisse können dabei nur in x-Achsen Abschnitt -7,3 bis 7,3 erscheinen. Der y-Achsen Wert ist so beschränkt, dass er in einem kleinen Bereich unterhalb des Sichtfeldes der Kamera gesetzt wird.

```
// Zufälliges Hindernis (klein/groß/beweglich)
GameObject obst = Instantiate(
    hindernisse[Random.Range(0,hindernisse.Length)], this.transform) as
GameObject;
// Zufällige Position
obst.transform.localPosition = new Vector3(
    Random.Range(7.3f,-7.3f),Random.Range(firstSpawn,firstSpawn-12f),0);
```

Damit nicht zu viele Hindernisse pro Erdschicht auftauchen, haben wir die Anzahl in der Update Funktion des Obstacle-Skripts beschränkt:

```
void Update () {
    posi = player.position.y;
    if(((firstSpawn - 12f) > posi) && count <3){
        if(count == 0){
            firstSpawn = posi;
        }
        newObstacle(posi);
        count += 1;
        if(count >=2){
            count = 0;
        }
    }
}
```

Aufgabe 4

Zeitraum:

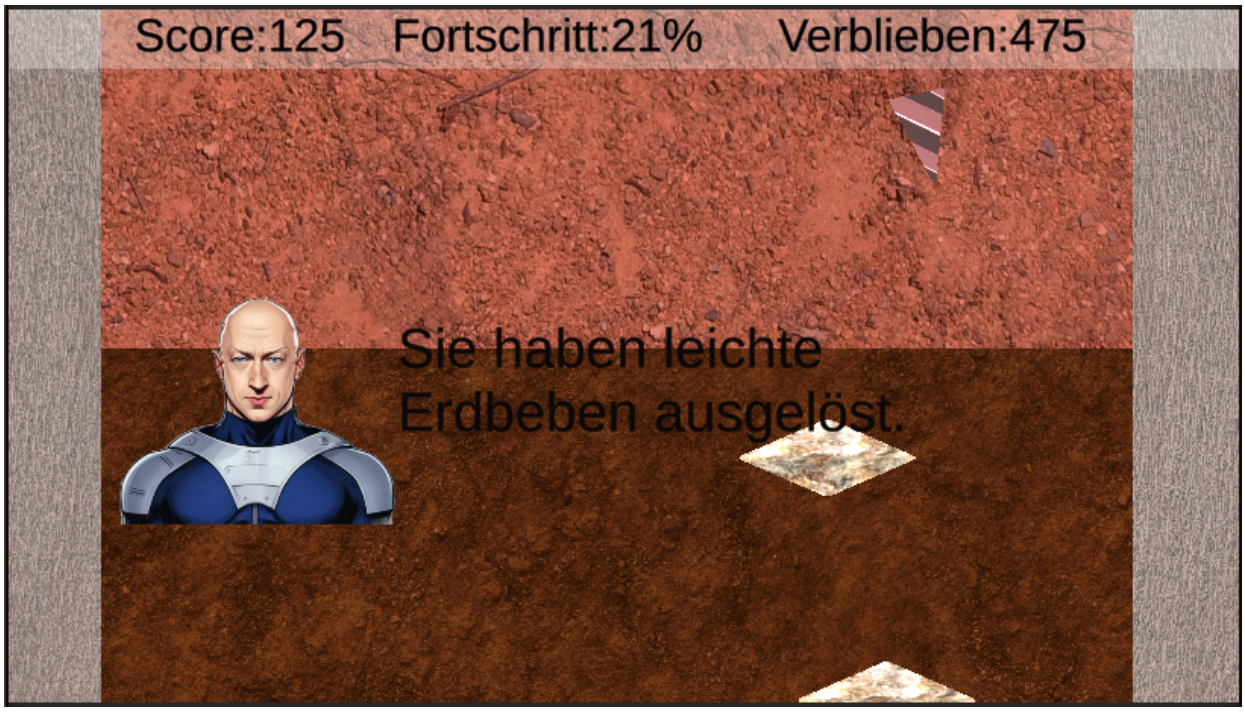
- 13.Juni – 20.Juni

Aufgaben:

In dieser Aufgabe haben wir 3 Meilensteine implementiert. Diese werden bei einem Score von 150, 300, 450 angezeigt. Im Bild unten ist ein Beispiel zu sehen. Während des Spiels spricht Zark in einer von AI generierten Stimme zusätzlich, das was auf dem Bild geschrieben steht. Das Spiel wird so lange wie die Tonspur ist angehalten:

```
float currentTime = 0;
float maxTime = audioS.clip.length;
while(currentTime<maxTime){
    currentTime += Time.unscaledDeltaTime;
    yield return null;
}
```

Im Bild ist zusehen, wie groß der Fortschritt bzw. der noch verbleibende Weg ist:



Unsere Formeln für *Fortschritts* und *Verbleibend* sind:

```
fortschritt =  System.Math.Round(((start - transform.position.y)/erdkern)*100,
1);
verblieben = erdkern - score;

textUI.text = "Score:" + score + "\t" + "Fortschritt:" + fortschritt +"%" +"\t" +
"Verblieben:" + verblieben;
```

Probleme:

Ein großes Problem war die Einbindung der Audiospur. Zu beginn der Einbindung war trotz einwandfreier mp3-Files kein klares Audio zu hören. Die Tonspur war so verzehrt, das nichts verstanden werden konnte. Letztendlich war das Problem, dass wir den Score Wert aus einem Float zu einem Integer umgewandelt haben. Bei der Konvertierung von Float zu Integer werden einfach die Nachkommastellen abgeschnitten bzw. gerundet, was dazu führt, dass sowohl $150,1$ und $150,2$ zu 150 werden und `score == 150` mehrfach `True` war. Deshalb wurde unser Meilenstein "mehrfach" aufgerufen, was dazu führte, dass die Audio verzehrt und unverständlich wurde. Durch eine Coroutine und ein boolischen Wert konnten wir das Problem lösen. Da dieser Vergleich in der Update-Funktion aufgerufen wird, war es wichtig, dass wir einen Frame warten (Zeile 32) und den Score um 1 erhöhen, bevor wir den boolischen Wert auf `True` setzten, der ein weiteres Vergleichen mit den Meilensteinen ermöglicht.

```
if(canPlayNext == true){
    if (score == 150){
        audioS.clip = audio[0];
        Meilenstein.text = "Sie haben leichte Erdbeben ausgelöst.";
        StartCoroutine(DelayedClearMeilensteinText());
    }
...
```

```
8       }else if((erdkern - score) == 0){
9           audioS.clip = audio[3];
10          Meilenstein.text = "Sie haben den Erdkern erreicht und die Erde
    zerstört";
11          StartCoroutine(DelayedClearMeilensteinText());
12          if(erdkern > GameManager.Instance.highscore){
13              // set score as Highscore if it is higher then the old one
14              GameManager.Instance.highscore = erdkern;
15          }
16          StartCoroutine(ReachedEnd());
17      }
18  }
19
20  IEnumerator DelayedClearMeilensteinText(){
21      canPlayNext = false;
22      float currentTime = 0;
23      float maxTime = audioS.clip.length;
24      score += 1;
25      while(currentTime<maxTime){ // alernative yield return new
    WaitForSecondsRealtime(5);
26          currentTime += Time.unscaledDeltaTime;
27          yield return null;
28      }
29      Meilenstein.text = "";
30      Zark.gameObject.SetActive(false);
31      Time.timeScale = 1;
32      yield return new WaitForSecondsRealtime(1);
33      canPlayNext = true;
34  }
```

Aufgabe 5

Zeitraum:

- 20.Juni – 04.Juli

Aufgaben:

Im Hauptmenü kann der aktuelle (lokale) Highscore gesehen werden (Zeile 7) über den beiden Menü-Punkten *Story* und *Endless*. In unserem Fall ist *Story* (zuvor *Start* genannt) der vorgegebene Modus aus den Aufgaben und *Endless* ist eines unserer Zusatz-Features. Wie der Name besagt, handelt es sich hierbei um eine Variante, in der der Spieler ins unendliche spielen kann, ohne Story Unterbrechungen, dies ist für diejenigen, die wirklich wissen wollen wie weit sie im Spiel überleben können. Dafür haben wir die Erdschichten und Hindernisse aus dem *Story* Modus verwendet.

Das andere Zusatz-Feature ist eine weitere Art von Hindernis, die oben schon genannt wurde. So haben wir nämlich 3 anstatt den vorgegebenen 2 Hindernissen. Des weiteren kann die Tonspur im Hauptmenü stumm geschaltet werden. Dazu haben wir einen Audio-Mixer, ein Toggle-Button und zwei Icons verwendet. Bei der Lautstärke in den Zeilen 19 und 23 ist zu beachten, dass die Einheit der

Werte in Dezibel angegeben werden muss. Dieses Feature ist sinnvoll, da nicht jeder Spieler ein Audio-Feedback erhalten möchte.

```
1  [SerializeField] private Image newImage;
2  [SerializeField] private Sprite spriteOn;
3  [SerializeField] private Sprite spriteOff;
4  [SerializeField] private AudioMixer audioMixer;
5
6  public void Start(){
7      textUI.text = "Highscore:" + GameManager.Instance.highscore;
8      float vol = 0;
9      audioMixer.GetFloat("masterVolume", out vol);
10     if(vol <= -30){
11         ChangeButtonImage(false);
12     }
13 }
14 ...
15 public void ChangeButtonImage(bool onOff)
16     {
17     if(onOff == true){
18         newImage.sprite = spriteOn;
19         audioMixer.SetFloat("masterVolume", -20);
20     }else{
21         newImage.sprite = spriteOff;
22         audioMixer.SetFloat("masterVolume", -80);
23     }
24 }
```

Zur Optimierung haben wir zwei Dinge getan einmal werden unsere Hindernisse frühstens zerstört wenn sie die Bildfläche nach oben verlassen. Zusätzlich haben wir die maximal dargestellten Hindernisse auf 6 beschränkt (Zeile 7). Dies haben wir durch eine `Queue` gelöst. In diese werden die Hindernis-Objekte angefügt und nach dem FIFO Prinzip auch wieder zerstört. Code Beispiel für die Hindernisse:

```
1  GameObject obst = Instantiate(...) as GameObject;
2  obst.transform.localPosition = new Vector3(...);
3  // An die Queue anfügen
4  obstQueue.Enqueue(obst);
5
6  // Wenn mehr als 6 Hindernisse auf dem Bild sind, ist das erste außerhalb des
   Bildes
7  if(obstQueue.Count > 6){
8      // erstes eingegangenes Objekt wird zerstört
9      Destroy(obstQueue.Dequeue());
10 }
```

Zum anderen zerstören wir auch unsere Erdschichten, die besonders im *Endless*-Mode zum Problem werden können. Dabei benutzen wir einen Counter, der bei der Instantiierung hoch gezählt wird. Nach spawnen einer neuen Schicht wird die Schicht mit dem Counter-Wert -2 zerstört.

```
public void newSchicht(float posit){
    if(counter == 0 ){
        schicht1 = Instantiate(erdschicht[Random.Range(0,erdschicht.Length)], this.transform) as GameObject;
        schicht1.transform.localPosition = new Vector3(0,posit+1f,0);
        counter = 1;
        Destroy(schicht2);
    }else if (counter == 1){
        schicht2 = Instantiate(erdschicht[Random.Range(0,erdschicht.Length)], this.transform) as GameObject;
        schicht2.transform.localPosition = new Vector3(0,posit+1f,0);
        counter = 2;
        Destroy(schicht3);
    }else if (counter == 2){
        schicht3 = Instantiate(erdschicht[Random.Range(0,erdschicht.Length)], this.transform) as GameObject;
        schicht3.transform.localPosition = new Vector3(0,posit+1f,0);
        counter = 0;
        Destroy(schicht1);
    }
}

```

Der Link zum Spiel kann oben im Dokument gefunden werden.