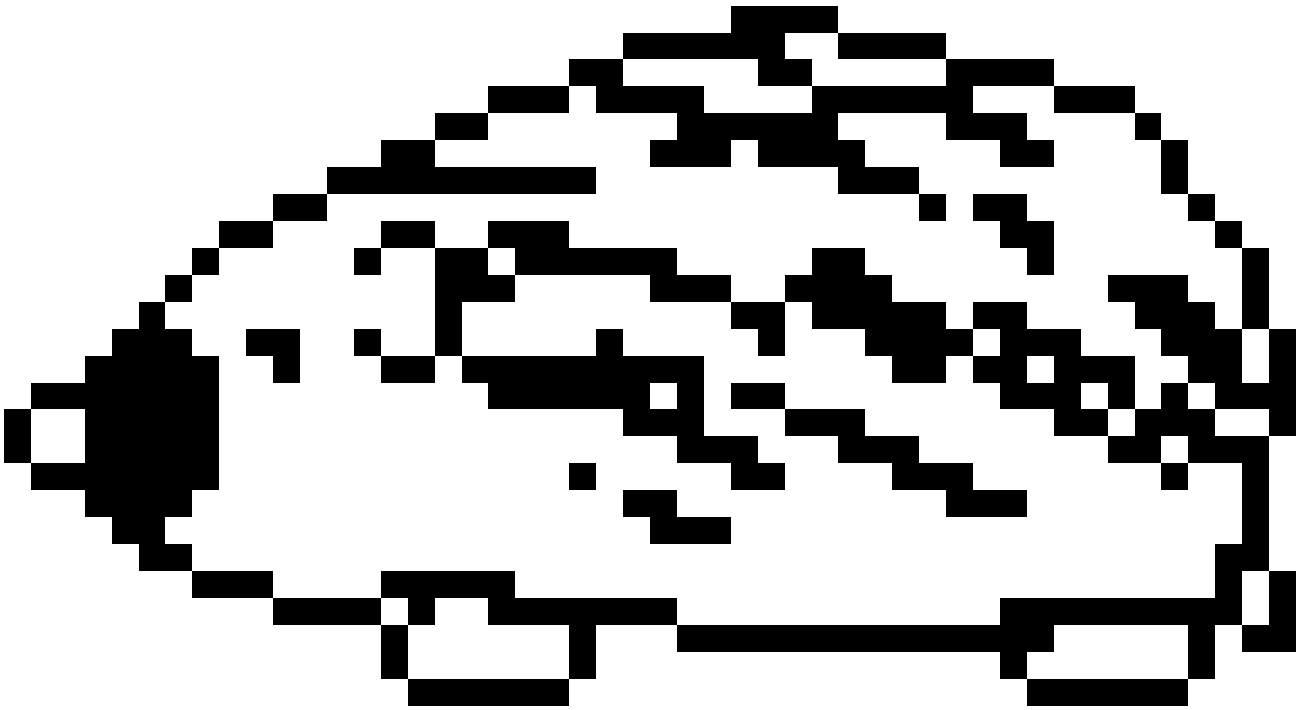
3. Milestone 2: 3 Räume = Mehr Content!

Nachdem wir einen ersten Prototyp Raum fertiggestellt haben und wir euch einen tieferen Einblick in die Erstellung eines Gegners gewährt haben, richten wir unser Ziel jetzt darauf, die restlichen Gegner fertig zu stellen und einen neuen Raumtypen ins Leben zu rufen, den Fallenraum.

Monsterraum

Zum Ende der letzten Zusammenfassung hatten wir einen sehr einfachen Braunbären, der immer auf den Spieler zurennt und ihn attackiert,
einen Selbstmordattentäter in der Form eines Lila Bären, der auf den Spieler zurennt und ab einer gewissen Entfernung zum Spieler einen Zünder mit einer Bombe aktiviert,
sowie einen auf zwei Beinen laufenden Hamster, der mit einem Blasrohr ausgestattet ist und zum Spieler eine gewisse Distanz behä und diesen regelmäßig mit Pfeilen beschiesst.

Geplant haben wir zusätzlich einen Wurm, der kurz an einem zufälligen Punkt auf dem Spielfeld erscheint, den Spieler mit einer verfolgenden Matschkugel beschiesst und daraufhin wieder verschwindet, um diesen Prozess zu wiederholen.
Des Weiteren haben wir einen ohne ziellos rumlaufenden Igel, der zur Selbstverteidigung regelmäßig 8 Pfeile um sich herum verschießt.
Zuletzt haben wir ein Schwein, das wutbesessen den Spieler verfolgt und sobald er dicht genug am Spieler ist, rammt er diesen mit einem mächtigen Sprint.

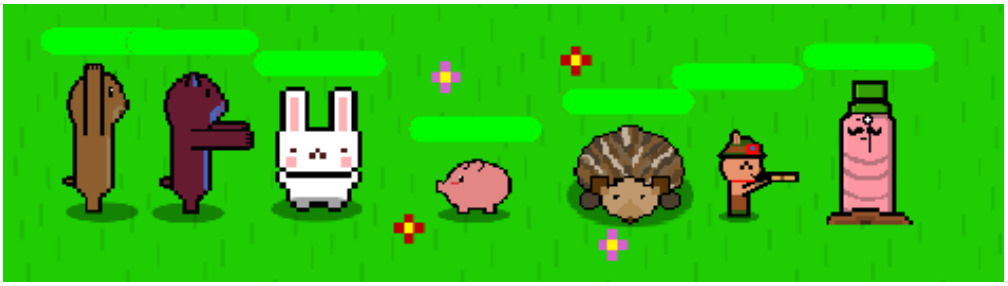


Wie wir so einen Gegner erstellen, haben wir bereits anhand des Hamsters gezeigt.

Die Schritte sind wie folgt:

1. Front – und Seitenskizze erstellen
2. Walkcycle (oben/unten/links/rechts) erstellen
3. Angriffsanimation erstellen
4. Animation in Unity übertragen
5. KI und Verhalten programmieren
6. Testing

Dies haben wir für jeden weiteren Gegnertypen gemacht. Ganz besonders wichtig waren die Größe und die Optik im Verhältnis zu de
bisherigen Gegnern beizubehalten.

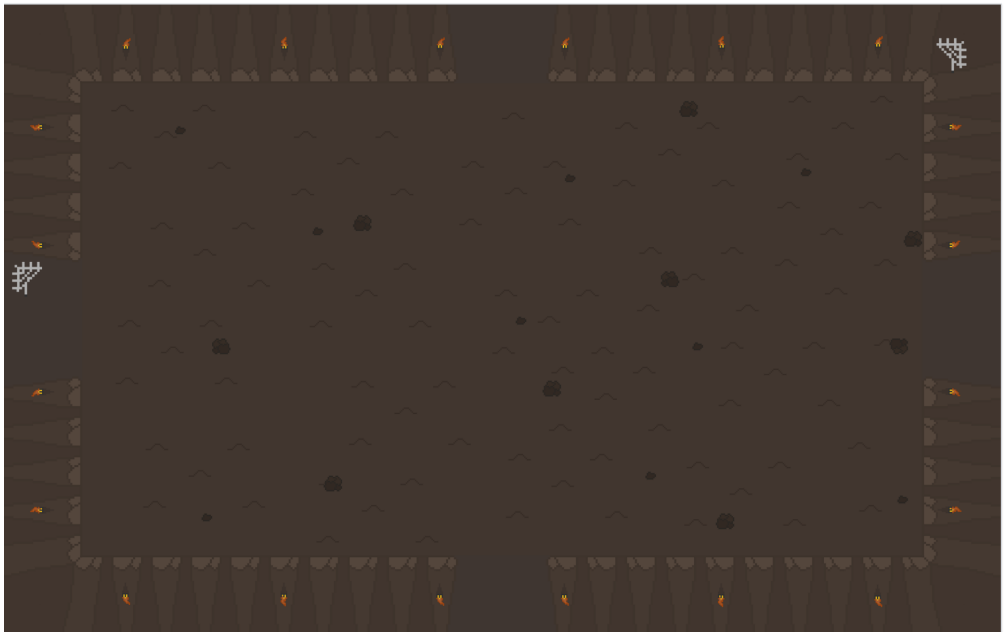


Mit dem Erstellen unserer letzten Gegnertypen, dem wild umherlaufenden Hasen, haben wir uns mit einer besonderen Sache beschäftigt.
Der Hase greift den Spieler nicht an, läuft aber ständig durch die Gegend, ist sehr schnell und hält sehr viele Treffer aus.
Um dies noch mehr zu erschweren, bleibt der Hase auch nur für eine bestimmte Zeit nach dem Betreten des Raumes.
Als Belohnung, wenn man es geschafft hat, ihn zu töten, erhält der Spieler zu 100% eine neue Waffe.
Diesen Waffendrop haben wir dann für jeden Gegnertypen implementiert, mit dem Unterschied, dass alle Gegner außer dem Hasen eine geringere Chance haben, beim Tod eine Waffe zu hinterlassen.
Zusätzlich hat der Spieler, um es fairer zu machen, zusätzlich die Chance, einen Lebenspunkt (Herz) bei Tötung eines Gegners zu erhalten.
Sollte der Zufall auf der Seite des Spielers sein und er wäre kurz davor, eine Waffe und ein Herz zugleich zu erhalten, müssen wir ihn leider enttäuschen, denn dann bekäme er nur das Herz.

Nachdem alle Gegner und Funktionen implementiert wurden, haben wir der Welt ein wenig mehr Leben gegeben und dem Spieler je ein wenig mehr Animationen.

Damit sind wir mit den Monsterräumen durch und wir gucken uns den nächsten speziellen Raum Typ an, den Fallenraum. Hierfür führen wir euch durch alle Fallen, die wir haben und gehen auf ein paar Details ein.

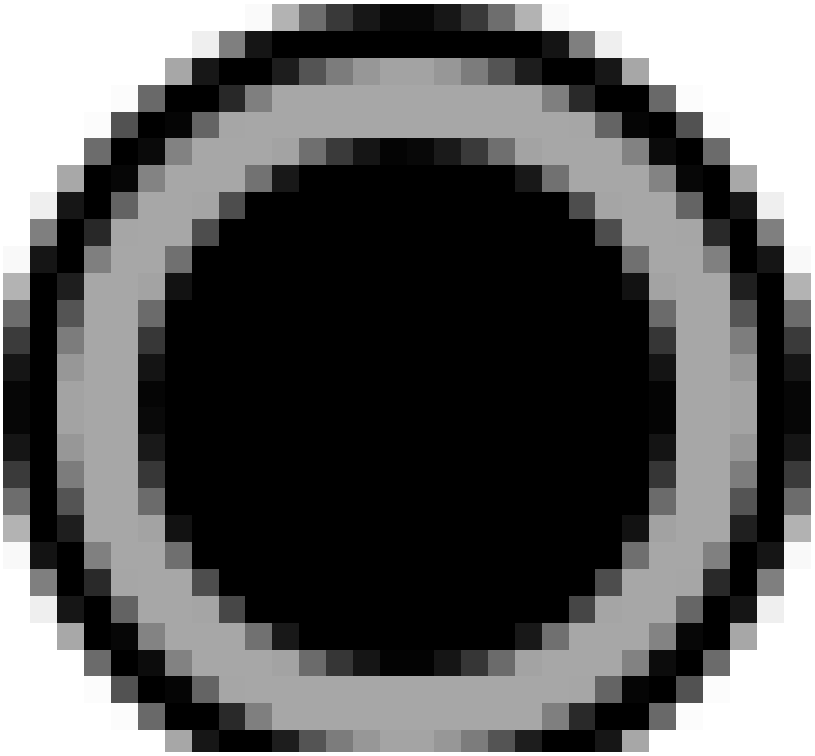
Zuallererst der Raum selbst.



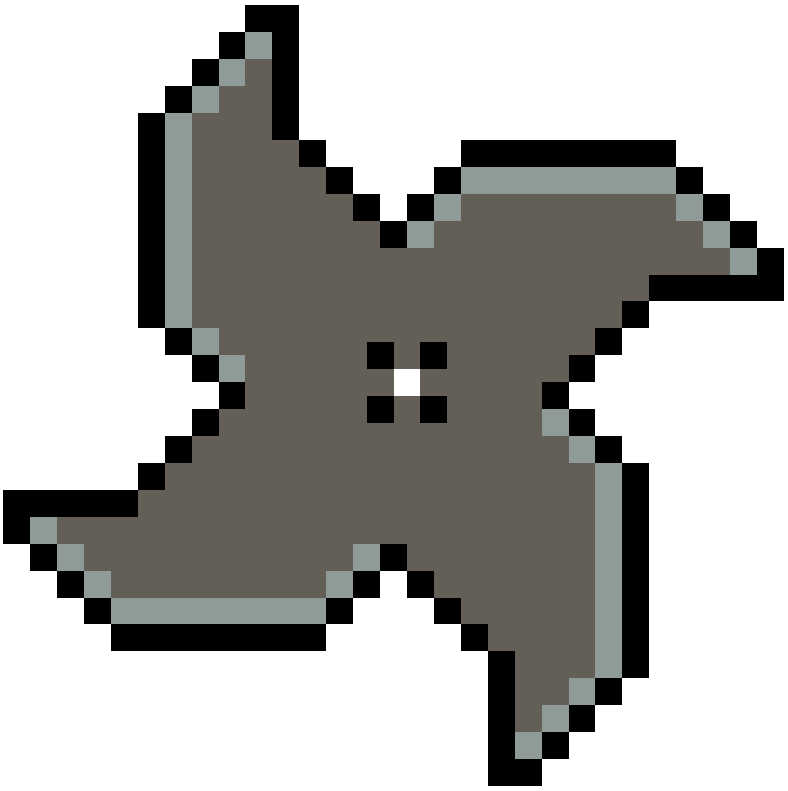
Hier haben wir uns für eine dunkle, schlecht beleuchtete Hölle entschieden. Dies soll erstmal im harten Kontrast zum grünen, farbenfrohen Monsterraum sein.
Außerdem erzeugen wir somit eine angespannte, düstere Atmosphäre.
Um das Level zu gestalten haben wir Trennwände und Abgründe entwickelt.
Die Trennwände agieren als eine einfache Wand. Bei den Abgründen hingegen kann man durchfallen und wird zum Anfang des Fallenlevels gebracht und verliert zusätzlich ein Leben.

Somit fehlen nur noch die Fallen, welche wir im Folgenden erläutern.

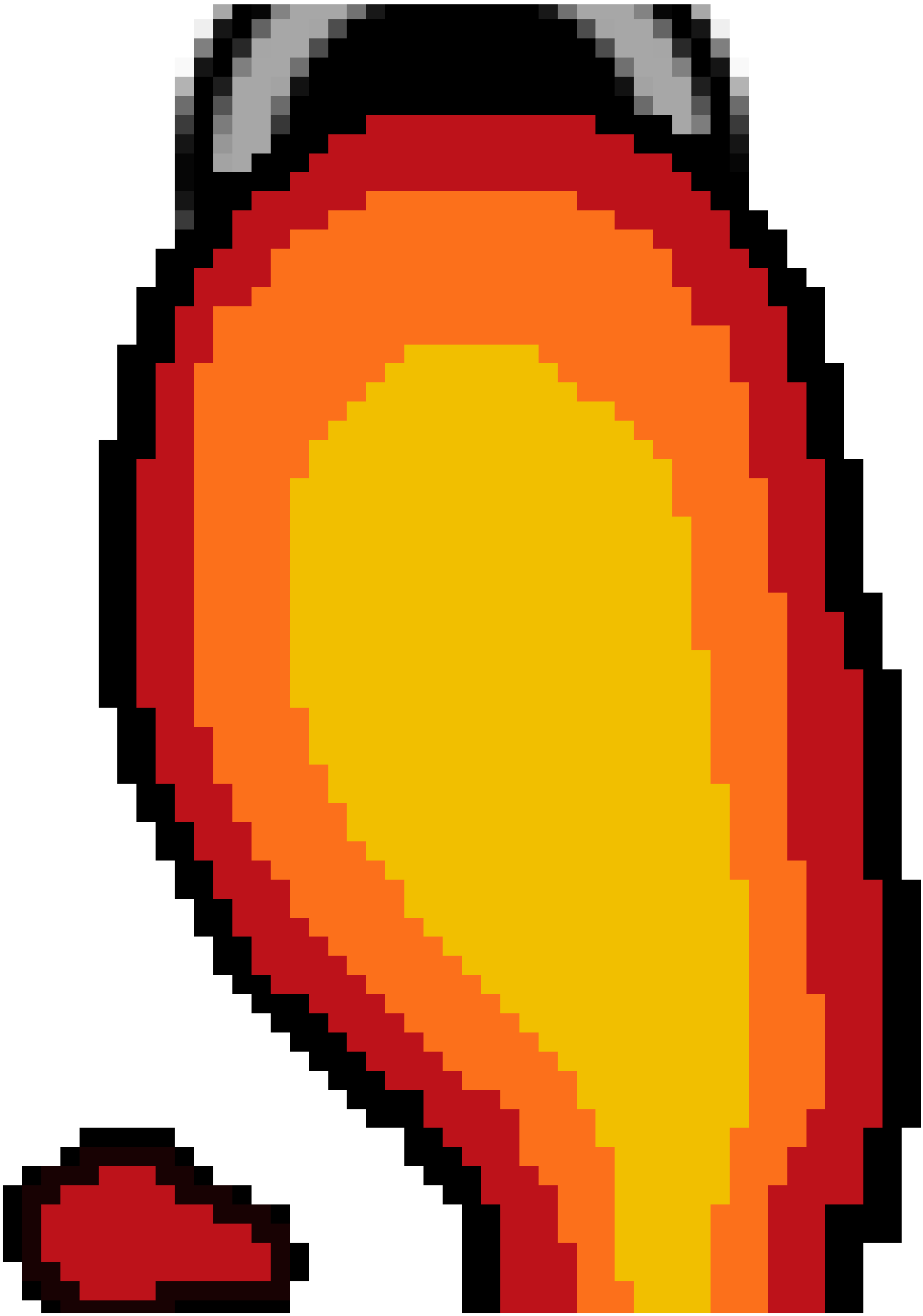
Pfeilenfalle:
Diese Falle schießt in einem gleichmäßigen Intervall Pfeile in einer geraden Linie. Bei Berührung bekommt der Spieler Schaden.

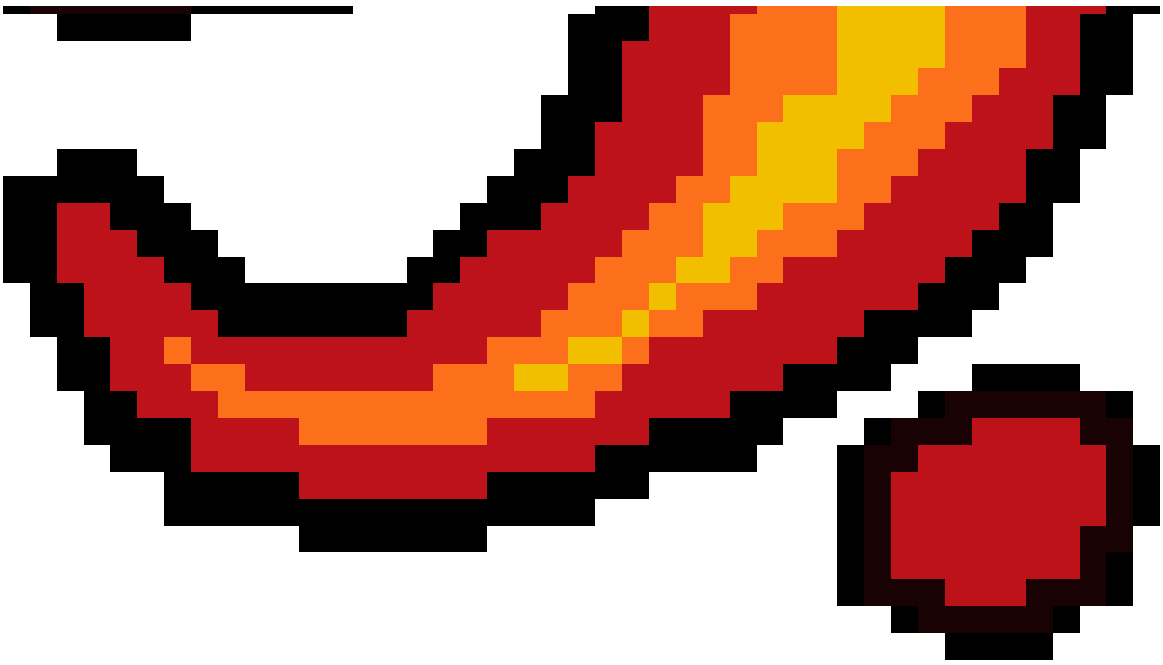


Shuriken:
Die Shuriken fliegen zwischen bestimmten Punkten hin und her und fügen dem Spieler bei Berührung Schaden zu.

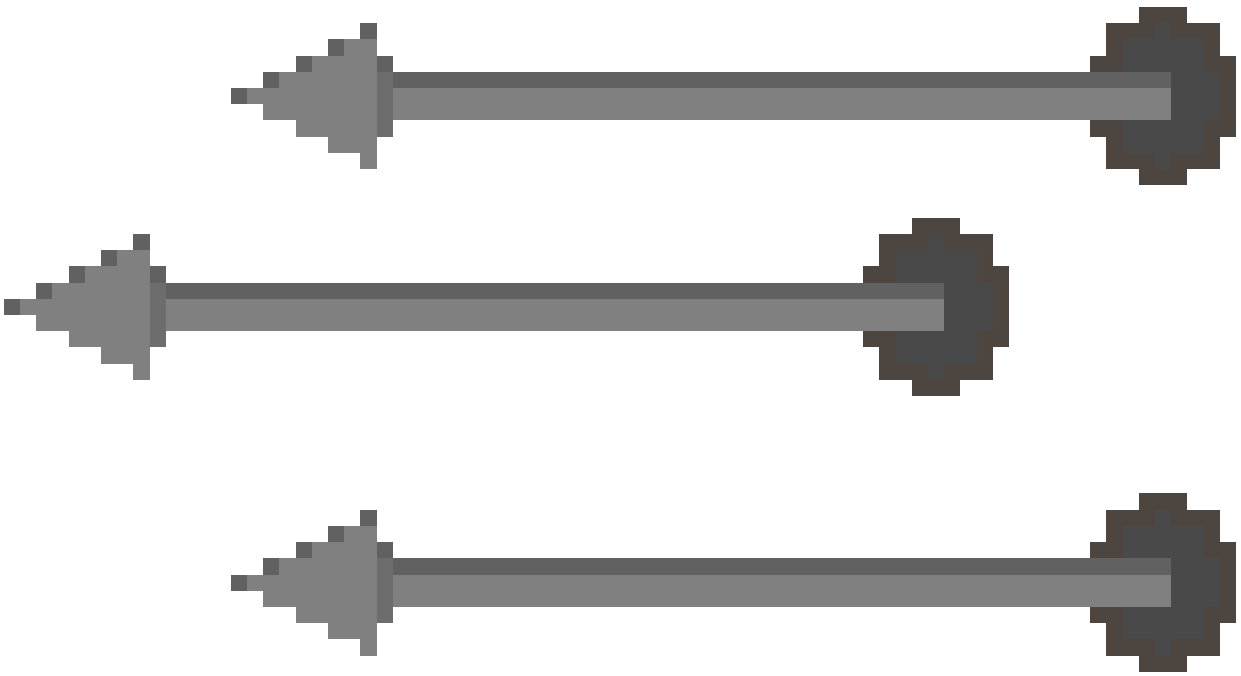


Flammenfalle:
Die Flamenfalle spuckt in einem kleinen Radius regelmäßig Feuer, das dem Spieler bei Berührung Schaden zufügen.

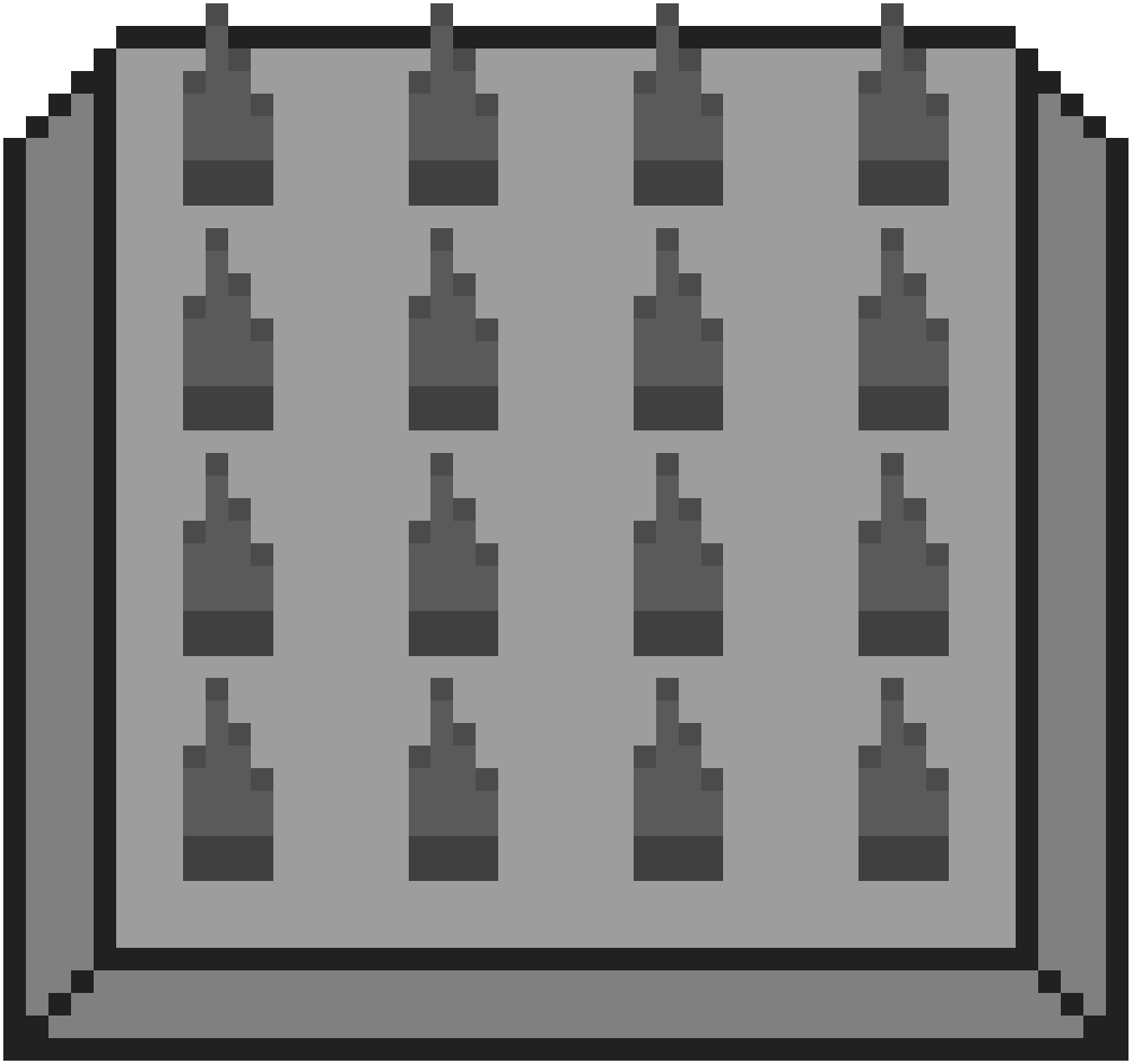




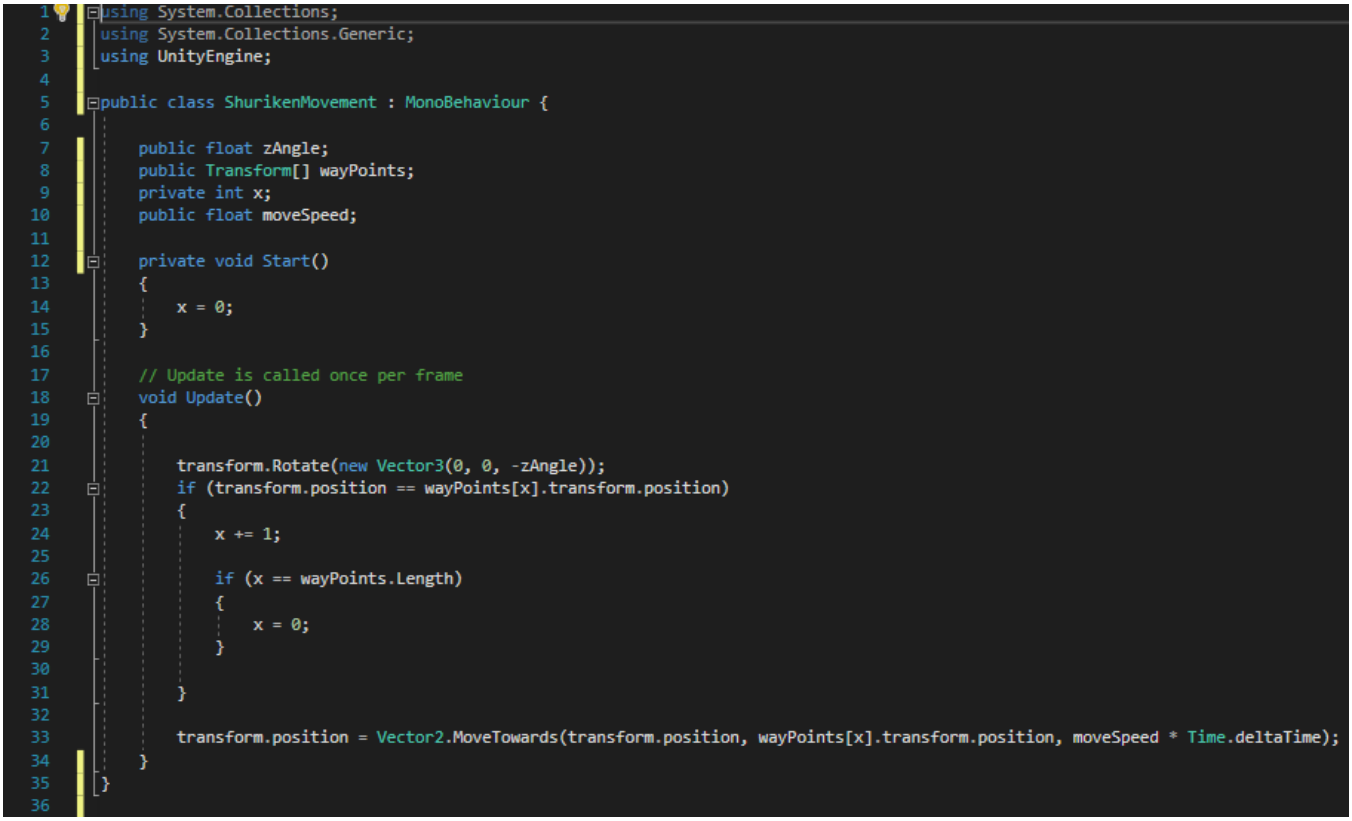
Speerefalle:
Bei der Speerefalle kommen beim Betreten des Bodens vor der Falle schlagartig Speere aus der Wand, die den Spieler wegstoßen und
ihm Schaden zufügen.



Stachelfalle:
Bei dieser Falle kommen Stacheln regelmäßig aus dem Boden. Sollte der Spieler diese berühren, erhält er Schaden.



Auch hier wollen wir euch kein Code vorenthalten. Also gehen wir im Folgenden auf das Shurikenskript ein.



```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class ShurikenMovement : MonoBehaviour {

    public float zAngle;
    public Transform[] wayPoints;
    private int x;
    public float moveSpeed;

    private void Start()
    {
        x = 0;
    }

    // Update is called once per frame
    void Update()
    {

        transform.Rotate(new Vector3(0, 0, -zAngle));
        if (transform.position == wayPoints[x].transform.position)
        {
            x += 1;

            if (x == wayPoints.Length)
            {
                x = 0;
            }

        }

        transform.position = Vector2.MoveTowards(transform.position, wayPoints[x].transform.position, moveSpeed * Time.deltaTime);
    }
}

```

Wir haben ein Array „wayPoints[]“ indem wir empty Gameobjecte speichern. Diese geben die Punkte mit den Koordinaten an, zu denen der Shuriken fliegen soll.
In der Update() in Zeile 21 lassen wir erstmal den Shuriken drehen.
In Zeile 22 überprüfen wir, ob die Position des Shurikens mit dem aktuellen Index des Arrays übereinstimmt.
Wenn das zutrifft, gehen wir zum nächsten Arrayindex. In der inneren If – Abfrage prüfen wir, ob unser Index am Ende angekomme ist.
Sollte dies der Fall sein, setzten wir ihn wieder zum Anfang zurück. Unabhängig davon lassen wir den Shuriken in Zeile 33 sich zu de einzelnen Zielpunkten bewegen.

Hier ein Bild eines fertigen Fallenraumes.
Zu sehen ist der Grundriss, die Abgründe und verschiedene Fallentypen.

Jetzt müssen wir unsere beiden Räume verbinden.
Genauer gesagt, müssen wir jetzt unseren Teleporter entwickeln, welcher uns von Raum A nach Raum B bringt, abhängig davon in welche Richtung wir Raum A verlassen wollen.

```
6   public class Teleport : MonoBehaviour {
7
8       public int levelToTeleport;
9       public int nextStart;
10      private MainCharacter player;
11
12      // Use this for initialization
13      void Start () {
14
15          player = GameObject.Find("MainCharacter").GetComponent<MainCharacter>();
16      }
17
18      private void OnTriggerEnter2D(Collider2D collision)
19      {
20          if(collision.tag == "Player")
21          {
22              PlayerPrefs.SetInt("PlayerHealth", player.getHealth());
23              PlayerPrefs.SetInt("PlayerStart", nextStart);
24              PlayerPrefs.SetInt("CurrentWeapon", player.getCurrentWeapon());
25              SceneManager.LoadScene(levelToTeleport);
26          }
27      }
28  }
```

Wenn der Teleporter Collider eine Collision mit dem Spieler feststellt speichern wir in welche Richtung er weiter gehene will, wie viel leben er hat und welche Waffe er gerade benutzt. Dann Laden wir das Level welches er gehen soll über den Unity SceneManager.

```
76          //start position new room
77          //up
78          if (PlayerPrefs.GetInt("PlayerStart") == 0)
79          {
80              transform.position = new Vector3(0, 4.0f, 0);
81              deactivateAllWeapons();
82              activateRightWeapon();
83          }
84          //right
85          else if (PlayerPrefs.GetInt("PlayerStart") == 1)
86          {
87              transform.position = new Vector3(7.5f, 0, 0);
88              deactivateAllWeapons();
89              activateRightWeapon();
90          }
91          //down
92          else if (PlayerPrefs.GetInt("PlayerStart") == 2)
93          {
94              transform.position = new Vector3(0, -4.0f, 0);
95              deactivateAllWeapons();
96              activateRightWeapon();
97          }
98          //left
99          else if (PlayerPrefs.GetInt("PlayerStart") == 3)
100         {
101             transform.position = new Vector3(-7.5f, 0, 0);
102             deactivateAllWeapons();
103             activateRightWeapon();
104         }
105
106         moveBlockTimer = 0;
107     }
```

In dem neuen Level prüfen wir, in welche Richtung er den alten Raum verlassen hat und setzten mit dem Wissen seine Startposition

Und so kommen wir, wenn wir in Raum A den rechten Ausgang nutzen, am linken Ausgang in Raum B an.
Damit ist dieser Milestone auch beendet, wir haben jetzt zwei Raumtypen und können zwischen diesen beiden hin und her wechseln und beide sind voll ausgestattet.

Ziel für den nächsten Milestone:
Im nächsten Milestone werden wir das Spiel designen. Darunter fallen eine Level-Struktur entwickeln, Level erstellen, diese zu verbinden und letztendlich das Spiel zu Balancen.