



Praxissemesterbericht
Studiengang : Informatik

2D Jump and Run

von

Lea-Marie Kindermann, Jenny Jost

82789, 83272

Betreuer Professor: Prof. Dr. Carsten Lecon

Einreichungsdatum : 07. Januar 2024

Eidesstattliche Erklärung

Hiermit erkläre ich, **Lea-Marie Kindermann, Jenny Jost**, dass ich die vorliegenden Angaben in dieser Arbeit wahrheitsgetreu und selbständig verfasst habe.

Weiterhin versichere ich, keine anderen als die angegebenen Quellen und Hilfsmittel benutzt zu haben, dass alle Ausführungen, die anderen Schriften wörtlich oder sinngemäß entnommen wurden, kenntlich gemacht sind und dass die Arbeit in gleicher oder ähnlicher Fassung noch nicht Bestandteil einer Studien- oder Prüfungsleistung war.

Ort, Datum

Unterschrift (Student)

Kurzfassung

In dieser Ausarbeitung behandeln wir das im Verlauf dieses Semesters entwickelte 2D Jump-and-Run-Spiel. Das Projekt beinhaltet die Umsetzung von drei abwechslungsreichen Levels und einem Bosskampf, wobei unterschiedliche Mechaniken integriert wurden. Die erfolgreiche Implementierung von sammelbaren Sternen und Münzen sowie die Schaffung qualitativ hochwertiger Animationen für Spieler und Gegner tragen zur Verbesserung des Spielgefühls bei. Die Bewältigung von Herausforderungen, wie der präzisen Steuerung von Gegnern, der Synchronisierung von Animationen und der Gewährleistung von Flüssigkeit in verschiedenen Spielmechaniken, wird thematisiert. Ebenso wird auf die Anpassung an die Unity-Entwicklungsumgebung eingegangen, insbesondere bezüglich der Nutzung des Canvas und der damit verbundenen UI-Herausforderungen. Insgesamt zeigt die Entwicklung des Spiels eine erfolgreiche Umsetzung der ursprünglichen Ideen und sogar die Integration zusätzlicher Funktionen.

Inhaltsverzeichnis

Eidesstattliche Erklärung	i
Kurzfassung	ii
Inhaltsverzeichnis	iii
Abbildungsverzeichnis	vi
Quelltextverzeichnis	viii
Abkürzungsverzeichnis	ix
1 Einleitung	1
1.1 Spielidee	1
2 Grundlagen	2
2.1 Steuerung	2
2.1.1 Bewegung	2
2.1.2 Springen	2
2.1.3 Angreifen	2
2.1.4 Klettern	3
2.1.5 Pause	4
2.1.6 Levelbeendigung	5
2.1.7 Level freischalten	7

2.1.8	Spielerleben	7
2.1.9	Tutorial-Schilder	7
2.2	Spielwelt	8
2.2.1	Gegner	8
2.2.2	Levelmenü	12
2.2.3	Level 1	15
2.3	Level 2	16
2.3.1	Level 3	21
2.3.2	Hauptmenü	22
2.3.3	Nach-Tod-Optionen	23
2.3.4	Erfolgsanzeige	24
3	Implementierung	25
3.1	Spielersteuerung	25
3.2	Angriffsmuster des Endbosses	27
3.3	Levelmanager und der Spielstand	28
3.4	Checkpoint	30
3.5	TilesAlgorithm	31
3.6	Sammeln	32
3.7	Sound	34
3.8	Animation	36
4	Inbetriebnahme	37
5	Evaluierung	38
5.1	Herrausforderungen	38
5.2	Erreichte Ergebnisse	39

6 Quellen	40
6.1 Software	40
6.2 Assets	40
6.3 Musik	40

Abbildungsverzeichnis

2.1 Angriff	3
2.2 Klettern	4
2.3 Pausen UI	5
2.4 Ende von Level 1	6
2.5 Level beenden	6
2.6 Checkpoint	7
2.7 Slime	8
2.8 Fledermaus	9
2.9 Spinne	10
2.10 Zerstörtes Nest	11
2.11 Skelettmonster	12
2.12 Level Zugang	13
2.13 Levelübersicht	14
2.14 nicht freigeschaltet	15
2.15 Levelübersicht	16
2.16 Geheimgang 1	17
2.17 Hebel	17
2.18 Geheimgang	18
2.19 Dungean	19
2.20 Dungean Ausgang	20
2.21 Geheimer ausgang	20

2.22 Geheimgang Level 3	21
2.23 gebrochener Boden	22
2.24 Hauptmenü	23
2.25 Nach Tod Option	24
3.1 Hebel	35
3.2 Hebel	36

Listings

3.1	Player.cs Line 46-97	25
3.2	Angriffsmuster des Endbosses	27
3.3	Levelmanager und der Spielstand	28
3.4	Checkpoint	30
3.5	TilesAlgorithm.cs	31
3.6	Sammeln	32

Abkürzungsverzeichnis

1 Einleitung

In dieser Ausarbeitung behandeln wir das im Verlauf dieses Semesters entwickelte 2D Jump-and-Run-Spiel. Wir werden dabei auf die Spielidee, den Spielaufbau, die Steuerung, unsere verwendeten Quellen sowie gelegentlich auf die Implementierung eingehen. Darüber hinaus möchten wir auf die Herausforderungen eingehen, denen wir erfolgreich begegnet sind.

1.1 Spielidee

Die Spielidee bestand darin, jedes Level mit neuen Herausforderungen zu gestalten, um eine kontinuierliche Spannung für den Spieler zu gewährleisten. Dabei wurden Elemente wie Plattformen, Sammeln und Gegnerintegration angestrebt. Zusätzlich beabsichtigten wir, den Spieler durch das Einbinden von geheimen Gängen dazu zu animieren, die Level zu erkunden. Als Höhepunkt des Spiels war die Implementierung eines Boss-Gegners vorgesehen. Die gesamte Spielerfahrung sollte durch die Untermalung mit Musik und Soundeffekten weiter verstärkt werden.

2 Grundlagen

Das Kapitel "Grundlagen" beinhaltet die Spielanleitung. Dabei werden die Steuerung erläutert, der Aufbau der Spielwelt beschrieben, die verschiedenen Gegnertypen dargestellt und die Besonderheiten der einzelnen Level hervorgehoben.

2.1 Steuerung

2.1.1 Bewegung

Der Spieler kann die Spielfigur mithilfe der Tasten A und D oder den Pfeiltasten nach links und rechts bewegen.

2.1.2 Springen

Durch Betätigen der Leertaste wird die Spielfigur zum Springen gebracht.

2.1.3 Angreifen

Sofern der Spieler im Besitz einer Waffe ist, kann er durch Drücken der Enter-Taste angreifen.

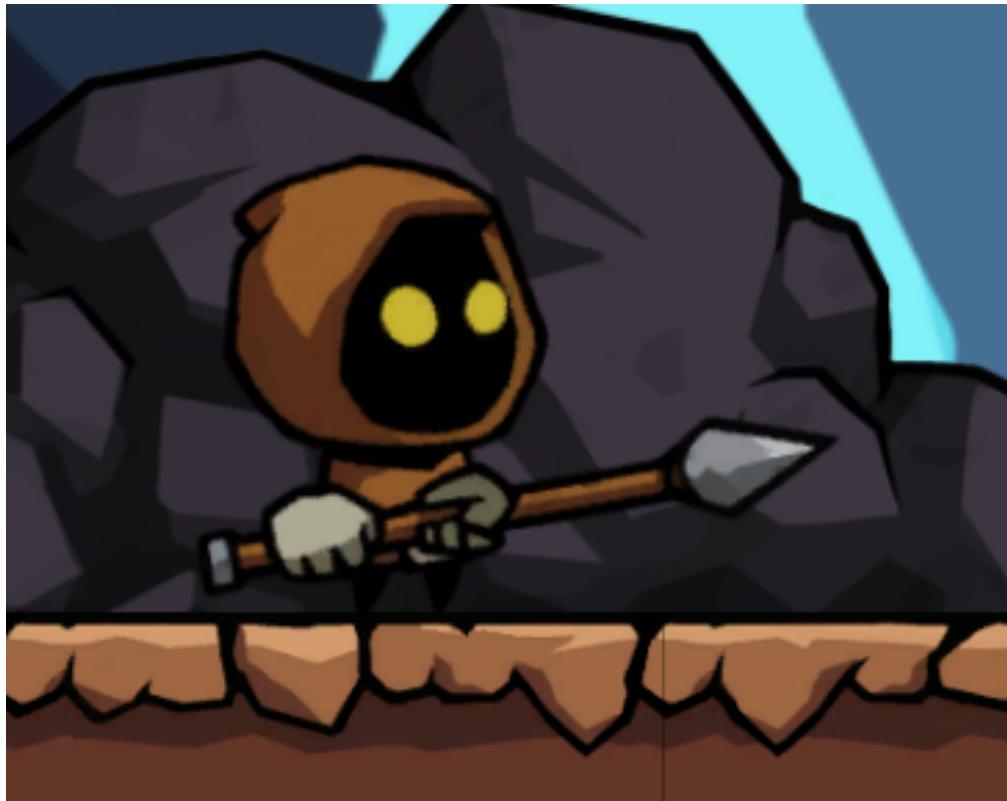


Abbildung 2.1: Angriff

2.1.4 Klettern

Steht der Spieler vor einer Leiter, kann er sich mithilfe der Tasten W und S oder den Pfeiltasten nach oben oder unten bewegen.

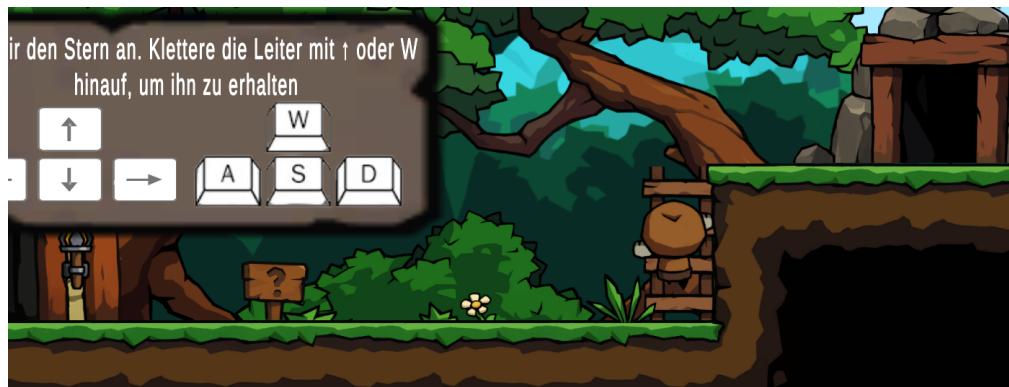


Abbildung 2.2: Klettern

2.1.5 Pause

Durch Drücken der Escape-Taste kann der Spieler jederzeit das Pause-Menü aufrufen. Dort hat er die Möglichkeit, durch Mausklick das Level neu zu starten, zum Level-Menü zu gelangen oder zum Hauptmenü zurückzukehren.



Abbildung 2.3: Pausen UI

2.1.6 Levelbeendigung

Erreicht der Spieler das Ende eines Levels, wird er automatisch weitergeleitet. Dort erhält er eine Übersicht über die gesammelten Münzen und Sterne. Der Spieler kann diesen Bildschirm jederzeit mit der Enter-Taste verlassen und zum Level-Menü zurückkehren.



Abbildung 2.4: Ende von Level 1



Abbildung 2.5: Level beenden

2.1.7 Level freischalten

Die Level müssen in richtiger Reihenfolge gespielt werden. Im Level-Menü wird der nachfolgende Level erst freigeschaltet, wenn der vorherige erfolgreich abgeschlossen wurde. Dabei ist das Sammeln von Sternen oder Münzen nicht zwingend erforderlich. Das bloße Erreichen des Levelendes genügt.

2.1.8 Spielerleben

Zu Beginn jedes Levels verfügt der Spieler über eine volle Lebensanzeige. Schaden kann durch Stacheln oder Angriffe von Gegnern verursacht werden. Bei zu viel Schaden oder einem Sturz stirbt der Spieler. Der Spieler kann entweder das Level von vorne beginnen oder, sofern ein Checkpoint zuvor erreicht wurde, von dort aus starten. Der Checkpoint stellt die Lebensanzeige des Spielers wieder her.



Abbildung 2.6: Checkpoint

2.1.9 Tutorial-Schilder

In den Levels sind kleine Schilder mit Erklärungen verteilt. Beim Passieren eines solchen Schildes erscheint eine Textbox mit Erklärungen. Im Verlauf des Spiels nimmt die Anzahl der Schilder ab, damit der Spieler selbstständig dazulernt.

2.2 Spielwelt

Die Spielwelt besteht aus drei Levels. Das Levelmenü ermöglicht den Zugang zu den verschiedenen Levels, wobei sich der Spieler frei im Menü bewegen kann. Jedes Level spielt in einer unterschiedlichen Umgebung und führt neue Mechaniken ein, um die Spannung aufrechtzuerhalten.

2.2.1 Gegner

Slime

Die Slime sind die ersten Gegner, die im Spiel vorgestellt werden. Es handelt sich dabei um grüne Monster, die sich in einem bestimmten Radius hin und her bewegen. Sie zielen nicht direkt auf den Spieler, aber beim Berühren erleidet der Spieler Schaden. Um die Slime zu besiegen, muss der Spieler auf sie springen.

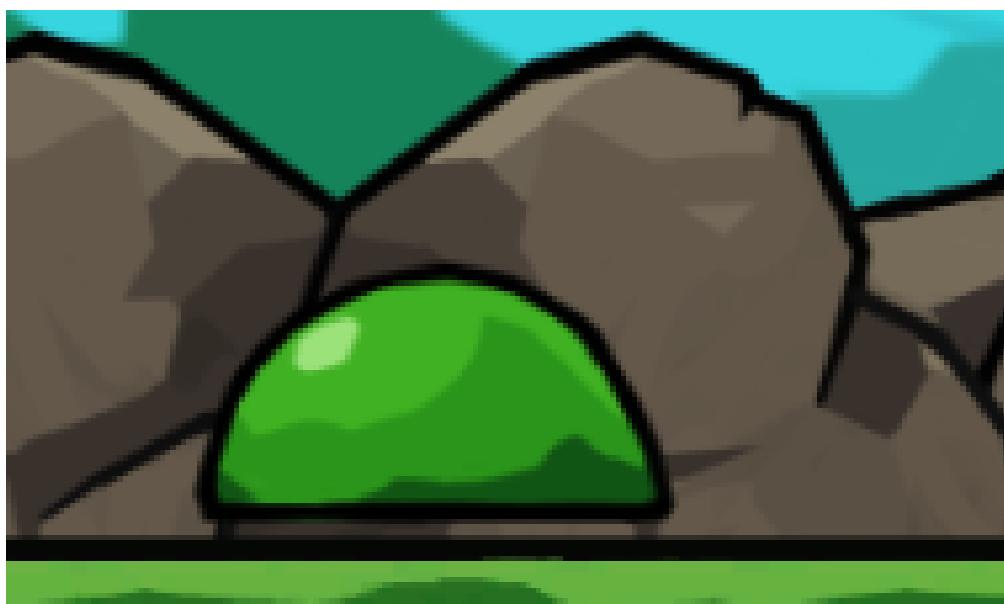


Abbildung 2.7: Slime

Fledermaus

Fledermäuse schlafen an der Decke. Wenn sich der Spieler ihnen nähert, erwachen sie und greifen den Spieler aktiv an. Dabei verfolgen sie den Spieler bis zu einem bestimmten Radius. Im zweiten Level erhält der Spieler einen Speer, den er verwenden muss, um die Fledermaus anzugreifen und zu eliminieren.



Abbildung 2.8: Fledermaus

Spinne

Spinnen bewohnen ihre Nester. Sobald sich der Spieler dem Nest nähert, schlüpfen die Spinnen heraus. Die Anzahl der Spinnen kann dabei variieren. Der Spieler hat die Möglichkeit, die Nester zu zerstören. Dabei verfolgen die Spinnen den Spieler bis zu einem bestimmten Radius.

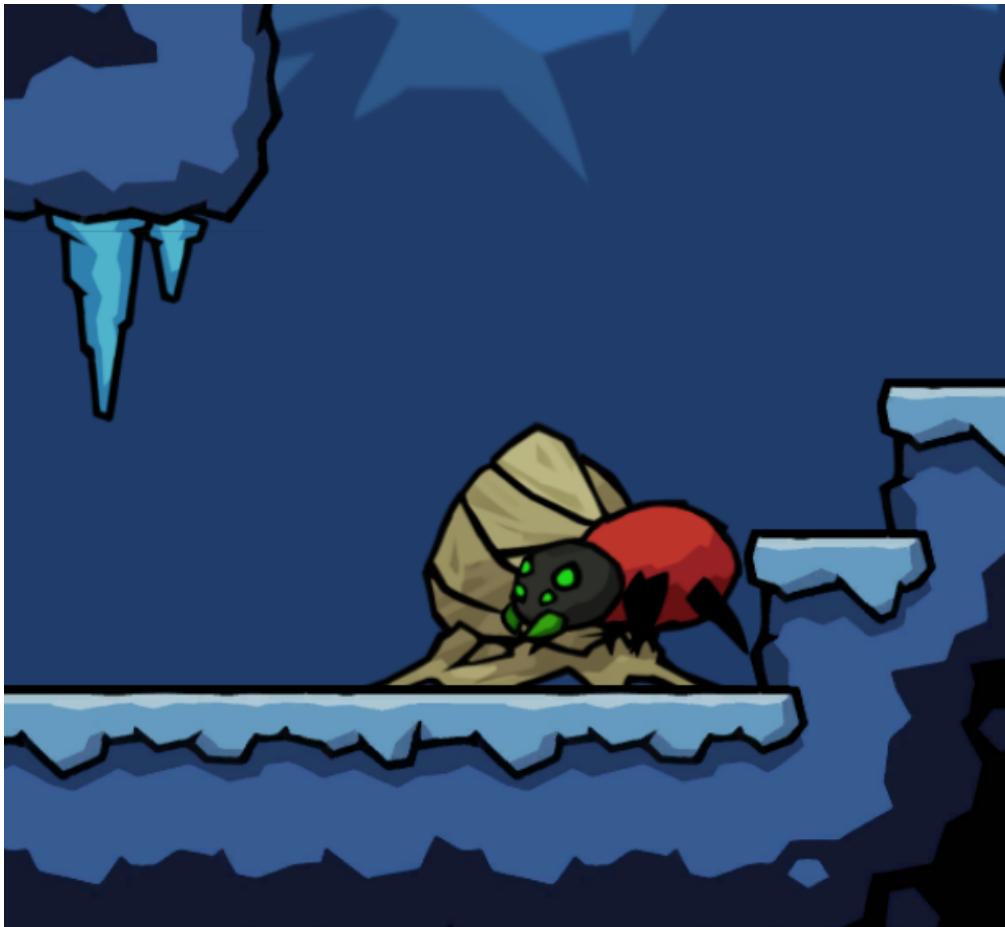


Abbildung 2.9: Spinne



Abbildung 2.10: Zerstörtes Nest

Skelettmonster

Das Skelettmonster fungiert als Boss in diesem Spiel. Es verfügt über zwei Angriffs-muster. Zum einen kann es auf den Spieler zuspringen, in diesem Fall muss der Spieler sich schnell entfernen. Zum anderen kann der Boss willkürlich nach links und rechts schlagen. Nach jedem Angriff ist der Boss für kurze Zeit verwundbar. In dieser Zeit muss der Spieler ihn mit seinem Speer angreifen.



Abbildung 2.11: Skelettmonster

2.2.2 Levelmenü

Das Spiel umfasst, wie zuvor erläutert, drei Levels, die durch das Levelmenü miteinander verknüpft sind. Die Zugänge zu den Levels sind durch Lagerfeuer und Banner klar markiert.

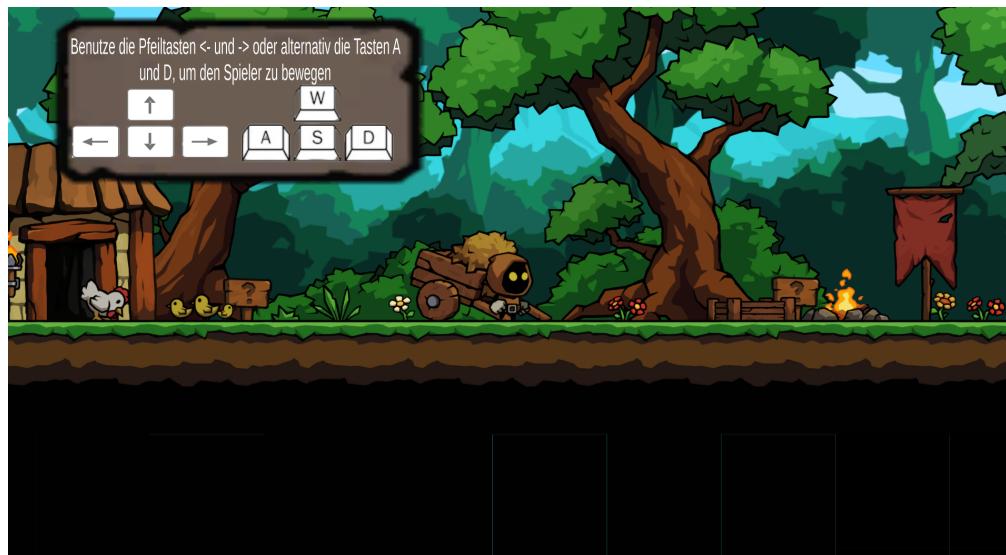


Abbildung 2.12: Level Zugang

Eine Levelübersicht wird aktiviert, wenn sich der Spieler in der Nähe des Lagerfeuers aufhält. Durch Drücken der Eingabetaste während der Übersicht kann das Level gestartet werden. In dieser Übersicht werden dem Spieler die fehlenden Sterne präsentiert, wobei in jedem Level drei Sterne platziert sind. Anhand der Platzierung der Sterne kann der Spieler erkennen, welche der drei Sterne noch fehlen. Zudem wird die Anzahl der im vorherigen Versuch gesammelten Münzen angezeigt. Nach erfolgreichem Abschluss eines Levels wird ein Bestätigungszeichen in Form eines grünen Hakens angezeigt.

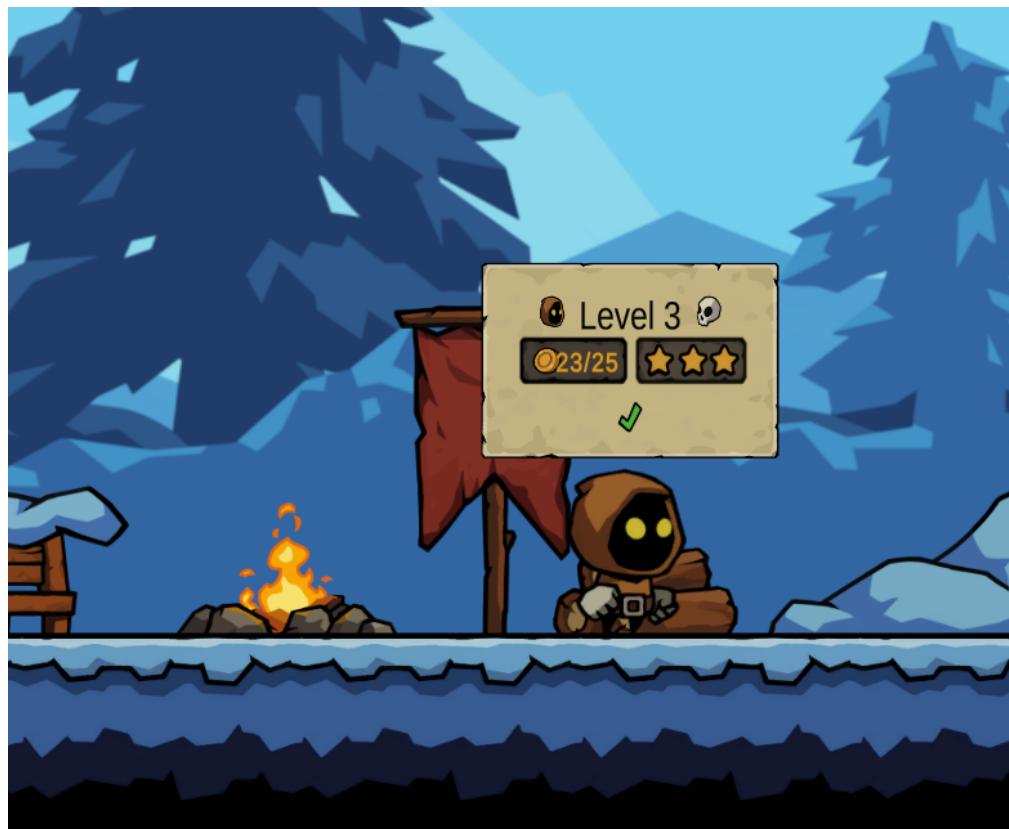


Abbildung 2.13: Levelübersicht

Sollte das Level noch nicht freigeschaltet sein, wird die Übersicht durch ein Schloss symbolisiert. An den Eingang von Level 2 ist ein Speer platziert, um dem Spieler anzuseigen, dass er in diesem Level eine Waffe zur Verfügung hat.

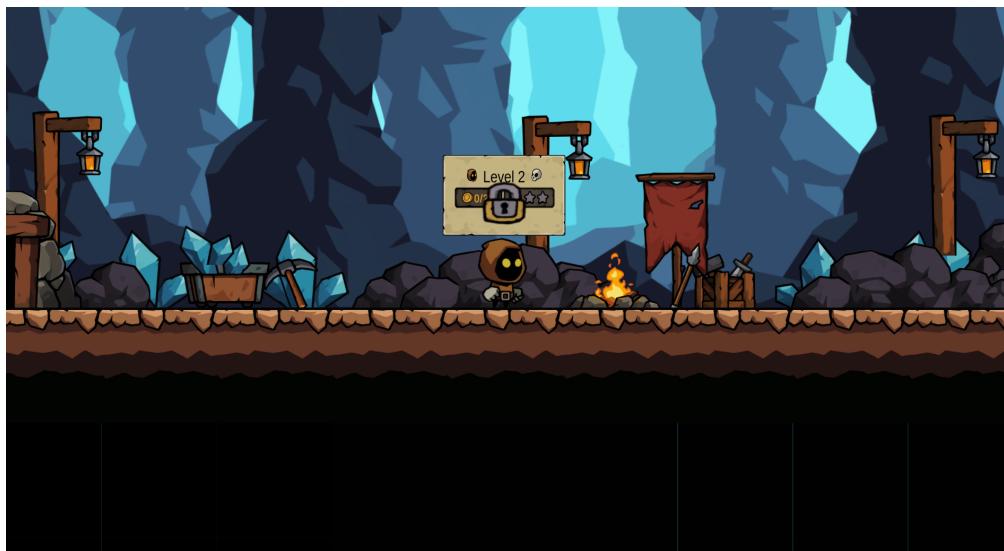


Abbildung 2.14: nicht freigeschaltet

2.2.3 Level 1

Im ersten Level werden dem Spieler die grundlegenden Mechaniken des Spiels vermittelt. Hierzu sind zahlreiche Schilder im Level verteilt, welche die Steuerung erläutern. In diesem Abschnitt verfügt der Spieler noch nicht über eine Waffe. Zudem begegnen ihm Slimes als erste feindliche Charaktere. Das Setting des ersten Levels ist ein Wald, das am Eingang einer Höhle endet, da das zweite Level dort stattfindet. Das Level beinhaltet leichte Plattforming-Elemente, wobei der Spieler in diesem Abschnitt noch nicht abstürzen kann. Verletzungen sind lediglich durch Stacheln möglich. Darüber hinaus sind Leitern im Level platziert, die der Spieler erklimmen muss. Die Konzeption dieses Levels zielt darauf ab, ein fesselndes Tutorial-Level zu präsentieren, das dem Spieler die Spielmechaniken näherbringt. Hierbei wird als unterstützendes Element eine fröhliche Musik eingespielt, um einen sanften Einstieg zu gewährleisten.



Abbildung 2.15: Levelübersicht

2.3 Level 2

Das zweite Level präsentiert sich in einer mystischen Höhle, begleitet von passender, geheimnisvoller Musik. Hierbei wird erstmals ein Speer als Waffe eingeführt. Mithilfe dieser Waffe müssen die Fledermäuse bekämpft werden, während auch die Slimes weiterhin im Level präsent sind und durch Sprünge überwunden werden. Die Plattforming-Elemente in diesem Level sind anspruchsvoller gestaltet. Zudem sind drei geheime Pfade strategisch im Level platziert, um den Spieler zur Erkundung zu motivieren.

Der erste Geheimgang befindet sich bereits zu Beginn des Levels. Der Spieler muss unmittelbar nach links springen, um einen Gang zu erreichen und den ersten Stern zu sammeln.

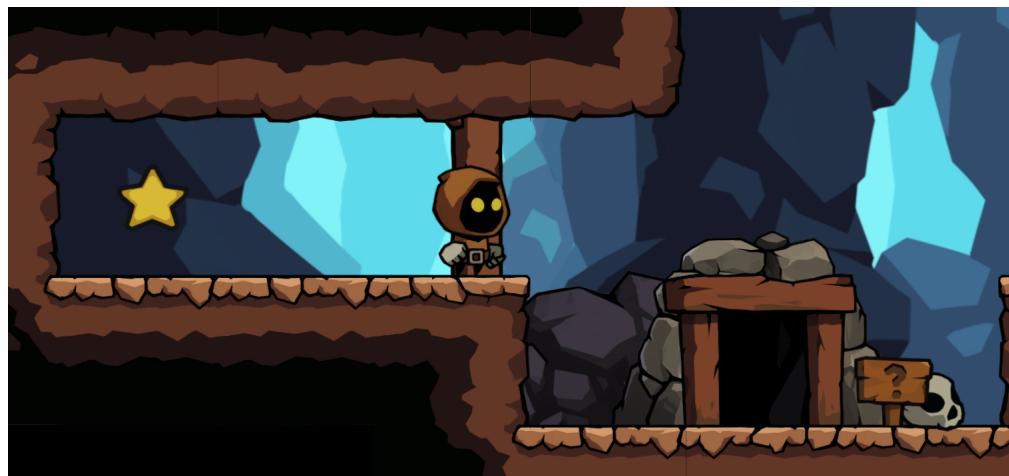


Abbildung 2.16: Geheimgang 1

Im weiteren Verlauf muss der Spieler einen Hebel betätigen, um Plattformen erscheinen zu lassen, die für die Fortbewegung im Level erforderlich sind.

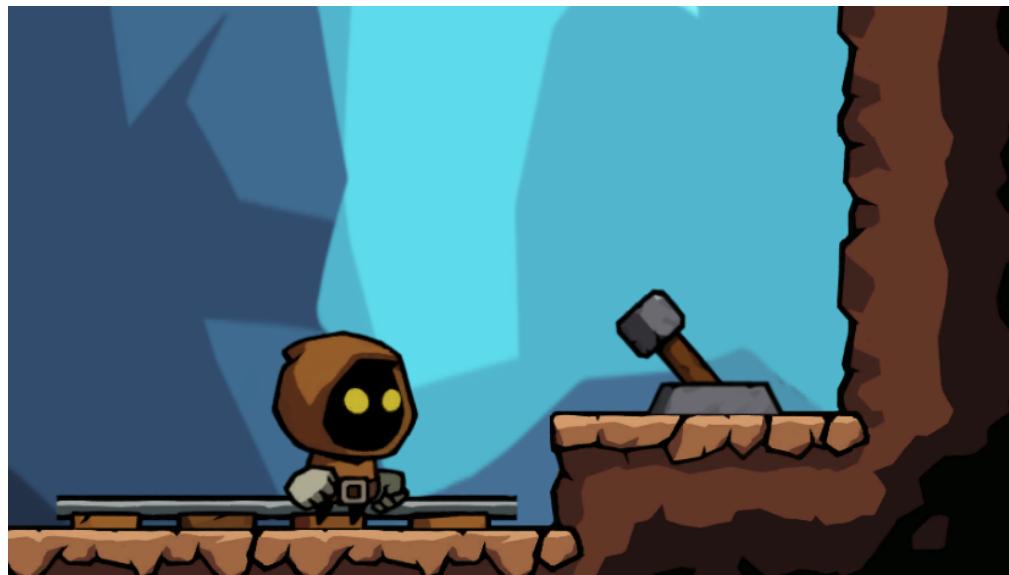


Abbildung 2.17: Hebel

In einem großflächigen Areal kann der Spieler bis ganz nach unten rechts springen,

um den zweiten Stern zu erhalten, da sich dort ein weiterer geheimer Gang befindet. Anschließend muss der Spieler wieder nach oben springen.

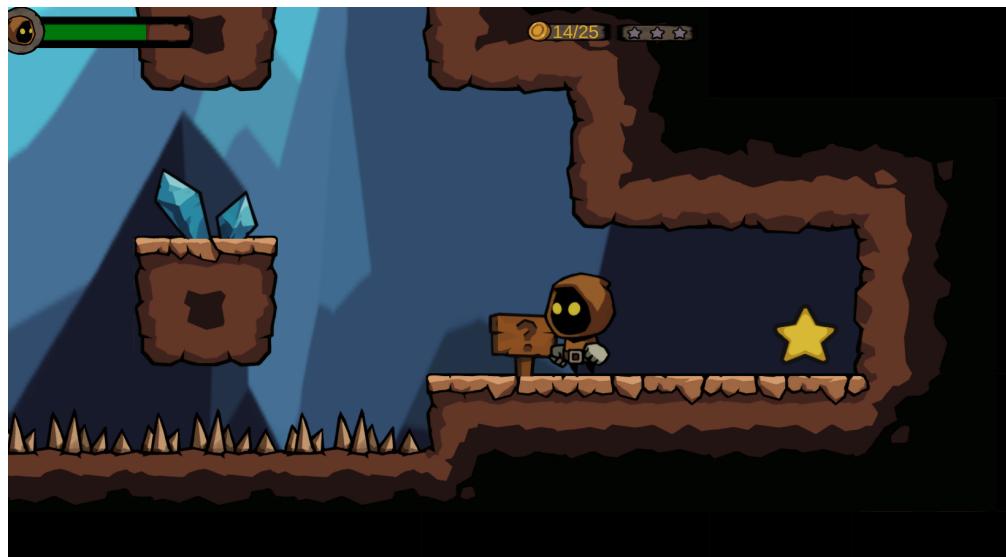


Abbildung 2.18: Geheimgang

Das Highlight des Levels ist der Dungeon. Nach Aktivierung des Checkpoints wird der Spieler automatisch dorthin teleportiert. Dort muss er sich einer Horde von Gegnern stellen, die nach und nach spawnen. Am Ende des Dungeons erscheint der Ausgang. Es gibt eine Abkürzung: Wenn der Gegner von der Klippe springt, befindet sich auf der linken Seite ein geheimer Gang, der den Spieler direkt nach draußen führt.

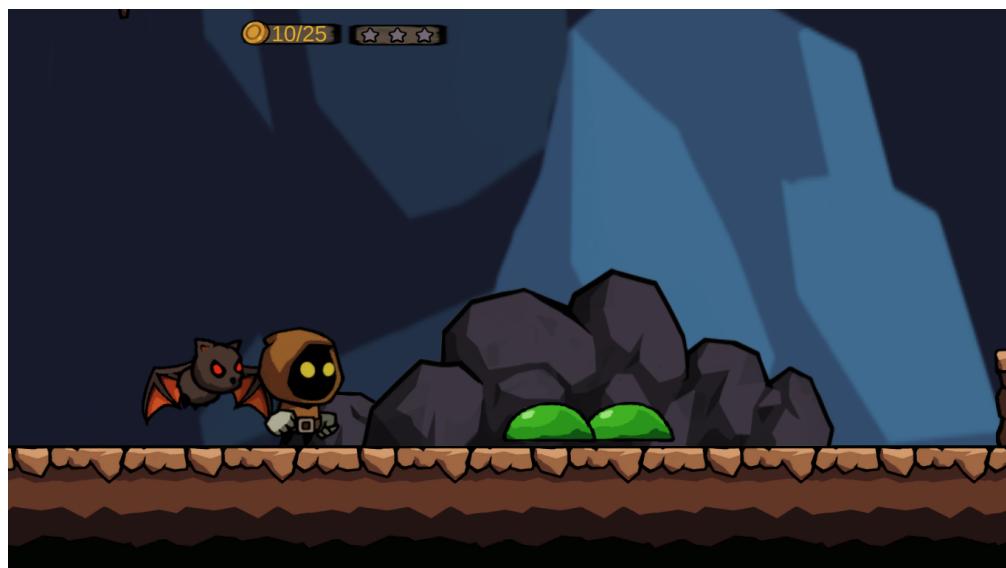


Abbildung 2.19: Dungean

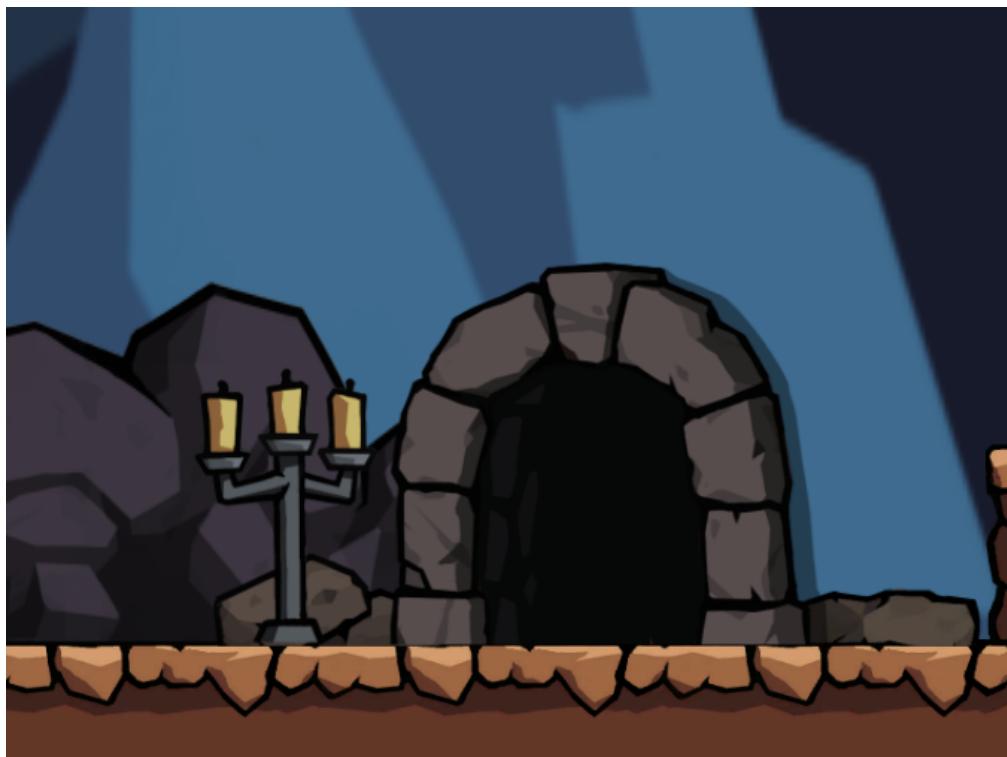


Abbildung 2.20: Dungeon Ausgang



Abbildung 2.21: Geheimer ausgang

2.3.1 Level 3

Im dritten und abschließenden Level findet das Spielgeschehen in einer Schneelandschaft statt, wobei der Höhepunkt in einem Schloss in Form des Bosskampfes gegen das Skelettmonster erreicht wird. Dieses Level wird durch einen besonders anspruchsvollen Schwierigkeitsgrad charakterisiert. Die Auswahl an Gegnern umfasst Spinnen und Slimes. Ein zusätzliches Element ist die Präsenz eines versteckten geheimen Gangs, der dem Spieler weitere Herausforderungen bietet.

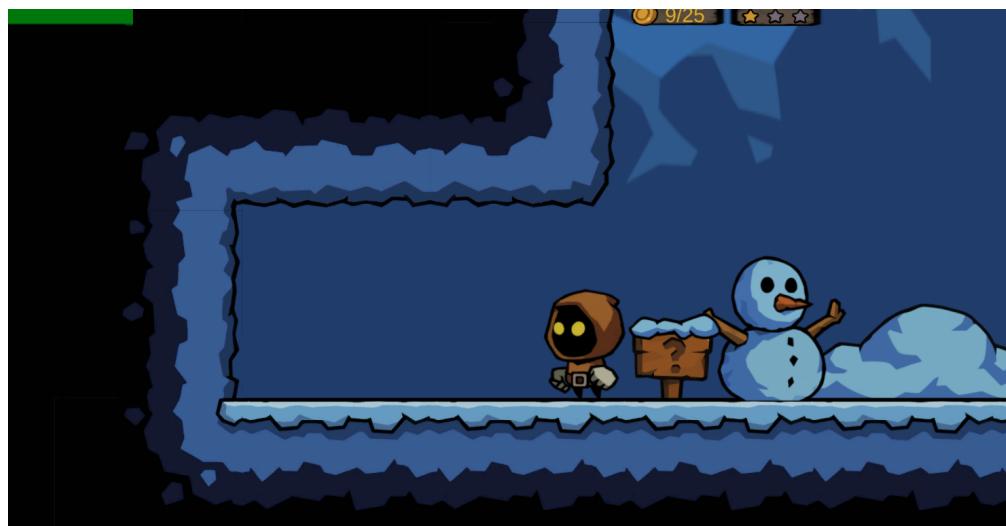


Abbildung 2.22: Geheimgang Level 3

Die Plattforming-Elemente in diesem Level werden anspruchsvoller, wobei einige Blöcke in einem festgelegten 3-Sekunden-Muster verschwinden. Der Spieler muss präzise getimte Sprünge ausführen, um auf den nächsten Block zu gelangen(auf 2). Zusätzlich kann es vorkommen, dass der Boden unter den Füßen des Spielers nachgibt.

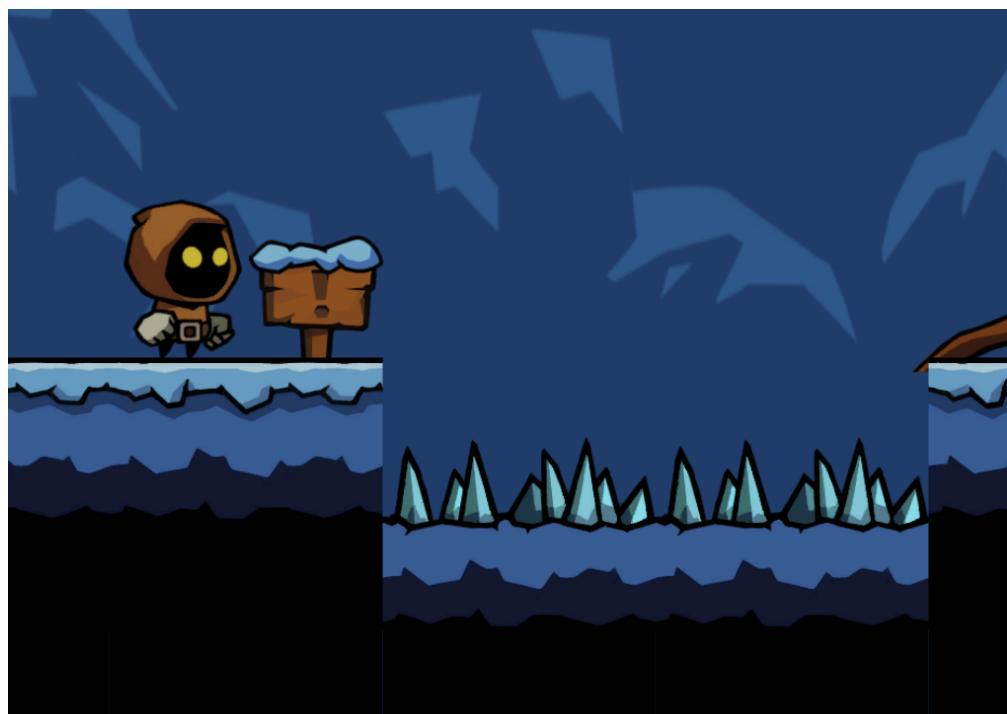


Abbildung 2.23: gebrochener Boden

Der Höhepunkt dieses Levels und des gesamten Spiels manifestiert sich im Bosskampf gegen das Skelettmonster im Schloss. Um die dramatische Intensität zu steigern, erfolgt eine thematisch angepasste musikalische Untermalung.

2.3.2 Hauptmenü

Beim Spielstart wird zunächst das Hauptmenü präsentiert. In diesem Menü hat der Spieler die Möglichkeit, das Spiel zu beginnen, zu beenden oder die Einstellungen zu öffnen. Die Navigation erfolgt dabei mithilfe der Maus. Innerhalb der Einstellungen besteht für den Spieler die Option, seinen Spielstand zu löschen.



Abbildung 2.24: Hauptmenü

2.3.3 Nach-Tod-Optionen

Nach dem Tod des Spielers wird eine Benutzeroberfläche angezeigt, die dem Spieler die Option bietet, das aktuelle Level erneut zu versuchen oder zurück zum Levelmenü zu gelangen.



Abbildung 2.25: Nach Tod Option

2.3.4 Erfolgsanzeige

Nach erfolgreichem Abschluss eines Levels wird eine Belohnungs-Benutzeroberfläche präsentiert, die dem Spieler anzeigt, welche Ressourcen er gesammelt hat. Diese Anzeige wird von einem erfreulichen Klang begleitet. Mithilfe der Sterneplatzierung in der Benutzeroberfläche kann der Spieler erkennen, welche Sterne ihm noch fehlen.

3 Implementierung

Im nachfolgenden Abschnitt erfolgt eine teilweise Erläuterung des Codes des Spiels, wobei ausschließlich auf essenzielle Stellen eingegangen wird. Diese Darstellung beschränkt sich auf einen repräsentativen Ausschnitt und konzentriert sich auf bedeutende Code-Passagen.

3.1 Spielersteuerung

```
30         }
31         if (direction > 0)
32     {
33             transform.eulerAngles = rotation;
34             transform.Translate(Vector2.right * speed * direction
35                         * Time.deltaTime);
36     }
37     if (isGrounded == false)
38     {
39         anim.SetBool("isJumping", true);
40     }
41     else
42     {
43         anim.SetBool("isJumping", false);
44     }
45
46     if (Input.GetKeyDown(KeyCode.Space) && isGrounded)
47     {
48         rb.AddForce(Vector2.up * jumpHeight,
49                     ForceMode2D.Impulse);
50         isGrounded = false;
51     }
52 }
```

Quelltext 3.1: Player.cs | Line 46-97

Das vorliegende Unity-Skript dient der Steuerung eines 2D-Spielers und bildet die Grundlage für die Interaktionen innerhalb des Spiels. Die Update-Methode wird verwendet, um kontinuierlich die Eingaben des Spielers zu überwachen und entsprechende Aktionen auszuführen. Ein wichtiger Aspekt des Skripts ist die Lebensüberprüfung des Spielers, die sicherstellt, dass Aktionen nur ausgeführt werden, solange der Spieler noch lebt.

Die Bewegungssteuerung des Spielers wird durch die Erfassung der horizontalen Eingabe gesteuert, wobei die Geschwindigkeit und Richtung der Bewegung entsprechend angepasst werden. Die Animation des Spielers wird dynamisch aktualisiert, um zwischen Lauf- und Ruhezuständen zu wechseln, je nachdem, ob der Spieler sich bewegt oder nicht.

Eine weitere wichtige Funktion des Skripts ist die Angriffssteuerung, die es dem Spieler ermöglicht, Angriffe auszuführen, wenn die Eingabetaste gedrückt wird. Dies löst eine Angriffsanimation aus und aktiviert eine Trefferbox für potenzielle Kollisionen mit anderen Objekten oder Gegnern im Spiel.

Das Skript implementiert auch ein Sprungsystem, das es dem Spieler ermöglicht, sich durch Drücken der Leertaste vom Boden abzustoßen. Dieser Sprungmecha-

nismus berücksichtigt die aktuelle Bodenhaftung des Spielers, um sicherzustellen, dass Sprünge nur ausgeführt werden, wenn der Spieler sich auf festem Untergrund befindet.

3.2 Angriffsmuster des Endbosses

```

1             IEnumerator AttackPattern()
2     {
3         while (true) // Dauerhaft wiederholen
4     {
5
6         if (bossHP.IsDead()) {
7             isDead = true;
8             yield break;
9         }
10
11        yield return new WaitForSeconds(attackCooldown);
12
13        yield return StartCoroutine(Dash());
14
15        yield return StartCoroutine(Attack());
16
17        bossAnimator.SetTrigger("stun");
18        weaponHitbox.enabled = false;
19        yield return new WaitForSeconds(2f);
20    }
21 }
```

Quelltext 3.2: Angriffsmuster des Endbosses

Das vorliegende Skript implementiert das Angriffsmuster des Endbosses des Spiels. Die AttackPattern()-Coroutine wird dauerhaft wiederholt, um die Aktionen des Bosses zu steuern. Vor jedem Angriff wird überprüft, ob der Boss noch am Leben ist. Falls nicht, wird die Coroutine sofort beendet, um unnötige Berechnungen zu vermeiden.

Die Angriffsabfolge des Bosses besteht aus verschiedenen Phasen. Der Kampf beginnt, sobald der Spieler in den Bereich des Bosses tritt. Zunächst wartet der Boss eine festgelegte Zeit zwischen den Angriffen, um dem Spieler die Gelegenheit zur Reaktion zu geben. Danach wird der Dash ausgeführt, um die Distanz zwischen dem Boss und dem Spieler zu schließen. Anschließend folgt der eigentliche Angriff, in welcher der Boss äusrastet und wild nach links und rechts um sich schlägt.

Nach jedem Angriff wird der Boss für eine kurze Zeit betäubt, da er sich verausgabt hat. Währenddessen ist die Waffen-Hitbox des Bosses deaktiviert, um zu verhindern, dass der Spieler Schaden erleidet.

Die Überprüfung am Anfang der while-Schleife versichert, dass der Boss aufhört anzugreifen wenn er besiegt wurde.

Die Verwendung von Coroutines ermöglicht eine geordnete Abfolge der Angriffe, wobei auf den Abschluss einzelner Aktionen gewartet wird, bevor die nächste ausgeführt wird. Dies trägt zur klaren und strukturierten Steuerung des Bossverhaltens bei und ermöglicht eine präzise Kontrolle über den Ablauf des Kampfes.

3.3 Levelmanager und der Spielstand

```
1      using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4
5  public class LevelManager : MonoBehaviour
6  {
7
8      // Level-IDs
9      public enum LevelID
10     {
11         Level1,
12         Level2,
13         Level3
14     }
15
16     // Funktion zum Speichern des abgeschlossenen Levels
17     public static void SaveCompletedLevel(LevelID levelID)
18     {
19         string key = "CompletedLevel_" + levelID.ToString();
20         PlayerPrefs.SetInt(key, 1);
21         PlayerPrefs.Save();
22     }
23
24     // Funktion zum Überprüfen, ob ein Level abgeschlossen ist
25     public static bool IsLevelCompleted(LevelID levelID)
26     {
27         string key = "CompletedLevel_" + levelID.ToString();
28         return PlayerPrefs.HasKey(key) && PlayerPrefs.GetInt(key)
29             == 1;
30     }
31
32     // Funktion zum Speichern der gesammelten Münzen in einem
33     // Level
34     public static void SaveCollectedCoins(LevelID levelID, int
35         collectedCoins)
36     {
37         string key = "CollectedCoins_" + levelID.ToString();
```

```
35         int currentCollectedCoins = PlayerPrefs.GetInt(key, 0);
36         currentCollectedCoins += collectedCoins;
37         PlayerPrefs.SetInt(key, collectedCoins);
38         PlayerPrefs.Save();
39     }
40
41
42
43     // Funktion zum Überprüfen, ob alle Münzen in einem Level
44     // gesammelt wurden
45     public static bool AreAllCoinsCollected(LevelID levelID)
46     {
47         string key = "CollectedCoins_" + levelID.ToString();
48         int collectedCoins = PlayerPrefs.GetInt(key, 0);
49         return collectedCoins == 25;
50     }
51
52     // Funktion zum Überprüfen, wie viele Münzen in einem Level
53     // gesammelt wurden
54     public static int GetCollectedCoins(LevelID levelID)
55     {
56         string key = "CollectedCoins_" + levelID.ToString();
57         int collectedCoins = PlayerPrefs.GetInt(key, 0);
58         return collectedCoins;
59     }
60
61     public static void SaveCollectedStar(LevelID levelID, int
62                                         starNumber) {
63         string key = "CollectedStar_" + levelID.ToString() +
64                     starNumber.ToString();
65         PlayerPrefs.SetInt(key, 1);
66         PlayerPrefs.Save();
67     }
68
69     public static bool[] GetCollectedStars(LevelID levelID){
70         bool[] starArry = new bool[3];
71         for (int i = 0; i < starArry.Length; i++){
72             string key = "CollectedStar_" + levelID.ToString() +
73                         i.ToString();
74             int star = PlayerPrefs.GetInt(key, 0);
75
76             if( star == 1){
77                 starArry[i] = true;
78             }
79             else {
80                 starArry[i] = false;
81             }
82         }
83         return starArry;
84     }
85 }
```

```
79         }
80     return starArry;
81 }
82 }
83 }
84 }
```

Quelltext 3.3: Levelmanager und der Spielstand

Das LevelManager-Skript übernimmt eine zentrale Rolle in der Verwaltung des Spielerfortschritts innerhalb des Spiels. Es ermöglicht das strukturierte Speichern und Abrufen von Informationen zu abgeschlossenen Level, gesammelten Münzen und Sternen. Die Funktionalitäten des Skripts dienen dazu, einen umfassenden Überblick über den Fortschritt des Spielers zu gewährleisten und spielen eine wichtige Rolle in der Spielererfahrung.

Die LevelID-Enumeration definiert dabei die verschiedenen Level im Spiel, was die Referenzierung und Handhabung innerhalb des Skripts vereinfacht. Durch die Methode SaveCompletedLevel() wird der Abschluss eines Levels gespeichert, was essentiell für die Spielprogression und den Zugang zu weiteren Level ist.

Die Funktionen zur Speicherung und Überprüfung von gesammelten Münzen sowie Sternen bieten die Möglichkeit, den Fortschritt des Spielers in Bezug auf optionale Sammelobjekte zu verfolgen. Die Methoden wie SaveCollectedCoins(), AreAllCoinsCollected() und GetCollectedCoins() ermöglichen eine differenzierte Handhabung von Münzen, während SaveCollectedStar() und GetCollectedStars() dies für Sterne tun.

Die Wichtigkeit des LevelManager-Skripts liegt in seiner Fähigkeit, den Spielerfortschritt zu speichern, zu überprüfen und abzurufen. Dies ermöglicht nicht nur die korrekte Steuerung des Spielverlaufs, sondern auch die Implementierung von Belohnungssystemen und optionalen Herausforderungen. Die klare Strukturierung und Verwaltung von Spielinformationen durch das Skript trägt somit entscheidend zur Gestaltung einer unterhaltsamen und motivierenden Spielererfahrung bei.

3.4 Checkpoint

```
1             private void OnTriggerEnter2D(Collider2D other)
2 {
3     if (other.CompareTag("Player") && !activated)
4     {
5         Debug.Log("Checkpoint aktiviert!");
6         // Heile den Spieler
7         PlayerHealth playerHealth =
player.GetComponent<PlayerHealth>();
```

```

8         playerHealth.SetFullHealth();
9
10        // Speichere die Position des Checkpoints als
11        // Respawn-Position
12        respawnPosition = transform.position;
13
14        // Setze den Checkpoint als aktiviert
15        activated = true;
16
17    }

```

Quelltext 3.4: Checkpoint

Das vorliegende Skript implementiert die Funktionalität des Checkpoints. Bei Kollision mit dem Spieler und sofern der Checkpoint noch nicht aktiviert wurde, wird der Checkpoint aktiviert. Dies beinhaltet das Wiederherstellen der Gesundheit des Spielers, das Speichern der Checkpoint-Position als Respawn-Position und das Markieren des Checkpoints als aktiviert. Diese Funktion ermöglicht es dem Spieler, seinen Fortschritt im Spiel zu sichern und Rückschläge zu vermeiden, was zur Verbesserung der Spielererfahrung und des Spielflusses beiträgt.

3.5 TilesAlgorithm

```

1             IEnumerator ActivateObjects()
2
3     {
4         while (true) // Dauerhaft wiederholen
5         {
6             foreach (GameObject obj in parkourObjects)
7             {
8                 obj.SetActive(false); // Deaktiviere alle Objekte
9                     zu Beginn
10            }
11
12            currentIndex = 0; // Setze den Index zurück
13
14            while (currentIndex < parkourObjects.Length)
15            {
16                // Aktiviere das aktuelle Objekt
17                parkourObjects[currentIndex].SetActive(true);
18
19                // Warte für die Dauer, die die Box aktiv bleiben
20                soll
21                yield return new
22                    WaitForSeconds(boxActiveDuration);
23
24            }
25
26        }
27
28    }

```

```

20          // Deaktiviere das aktuelle Objekt
21          parkourObjects[currentIndex].SetActive(false);
22
23          // Inkrementiere den Index für das nächste Objekt
24          currentIndex++;
25
26          // Warte für die Verzögerung zwischen den Boxen,
27          // wenn es noch eine gibt
28          if (currentIndex < parkourObjects.Length)
29          {
30              yield return new
31                  WaitForSeconds(timeBetweenBoxes);
32          }
33      }

```

Quelltext 3.5: TilesAlgorithm.cs

Dieser Skriptausschnitt implementiert eine Coroutine, die eine Gruppe von Game-Objects nacheinander aktiviert und deaktiviert. Diese Funktion wird verwendet, um Plattformen zu einem herausfordernden Parkour zu machen. Die Coroutine läuft dauerhaft und aktiviert jedes GameObject für eine bestimmte Zeit, bevor es deaktiviert wird. Durch diese Funktionalität konnten dynamische und herausfordernde Umgebungen geschaffen werden, die das Spielerlebnis bereichern und die Levelgestaltung vereinfachen.

3.6 Sammeln

```

1             using System.Collections;
2 using System.Collections.Generic;
3 using UnityEngine;
4
5 public class collect : MonoBehaviour
6 {
7     public CollectManager managerCoin;
8     public CollectStarManager managerStar;
9     public AudioSource coinAudio;
10    public AudioSource starAudio;
11
12
13    void Update()
14    {
15
16    }
17

```

```
18     private void OnTriggerEnter2D(Collider2D other)
19     {
20         if (other.CompareTag("star1"))
21         {
22             starAudio.Play();
23             managerStar.AddStar( 0 );
24             Destroy(other.gameObject);
25         }
26         else if (other.CompareTag("star2"))
27         {
28             starAudio.Play();
29             managerStar.AddStar( 1 );
30             Destroy(other.gameObject);
31         }
32         else if (other.CompareTag("star3"))
33         {
34             starAudio.Play();
35             managerStar.AddStar( 2 );
36             Destroy(other.gameObject);
37         }
38         else if (other.CompareTag("coin"))
39         {
40             coinAudio.Play();
41             managerCoin.AddCoin();
42             Destroy(other.gameObject);
43         }
44     }
45 }
```

Quelltext 3.6: Sammeln

Das vorliegende Skript steuert das Verhalten von dem Sammeln der Münzen und Sterne des Spiels und bildet somit auch eine der Grundlagen des Spieleanlaufs. Wenn der Collider mit einem Objekt kollidiert, das mit den Tags "star1", "star2" oder "star3" markiert ist, wird ein Sound abgespielt, und der Manager für Sterne wird aufgerufen, um dem Spieler den entsprechenden Stern hinzuzufügen. Das kollidierte Objekt wird anschließend zerstört. Falls das kollidierte Objekt mit dem Tag "coin" markiert ist, wird ebenfalls ein Sound abgespielt, der Münzen-Manager wird aufgerufen, um dem Spieler eine Münze hinzuzufügen, und das kollidierte Objekt wird zerstört. Diese Funktionalität ermöglicht es dem Spieler, Sterne und Münzen im Spiel zu sammeln und die Spielmechanik zu unterstützen.

3.7 Sound

Im Spiel wurden verschiedene Sounds implementiert. Auf dem Bild ist der Soundplayer eines Slimes zu sehen. Um sicherzustellen, dass der Sound nur hörbar ist, wenn sich der Spieler in der Nähe befindet, wurde die Einstellung auf 3D-Sound festgelegt. Mithilfe der linearen Einstellung des Radius kann bestimmt werden, wann der Sound hörbar ist. Die Soundauslösung für andere Objekte erfolgt teilweise durch Skripte.

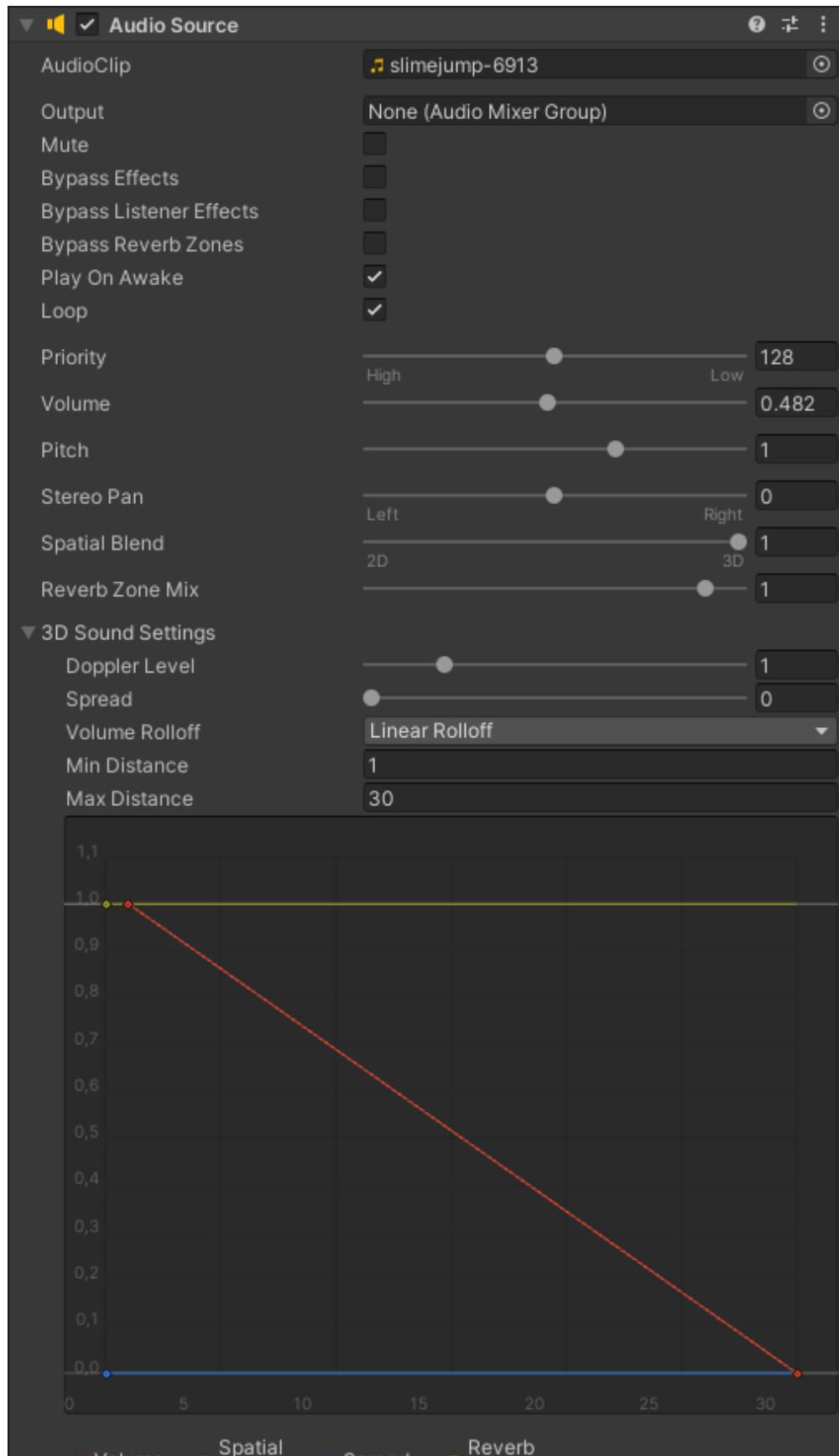


Abbildung 3.1: Hebel

3.8 Animation

Die Animationen für den Spieler und die Gegner werden durch einen Animationsplayer gesteuert. Auf dem Bild ist der Animationsplayer des Spielers zu sehen.

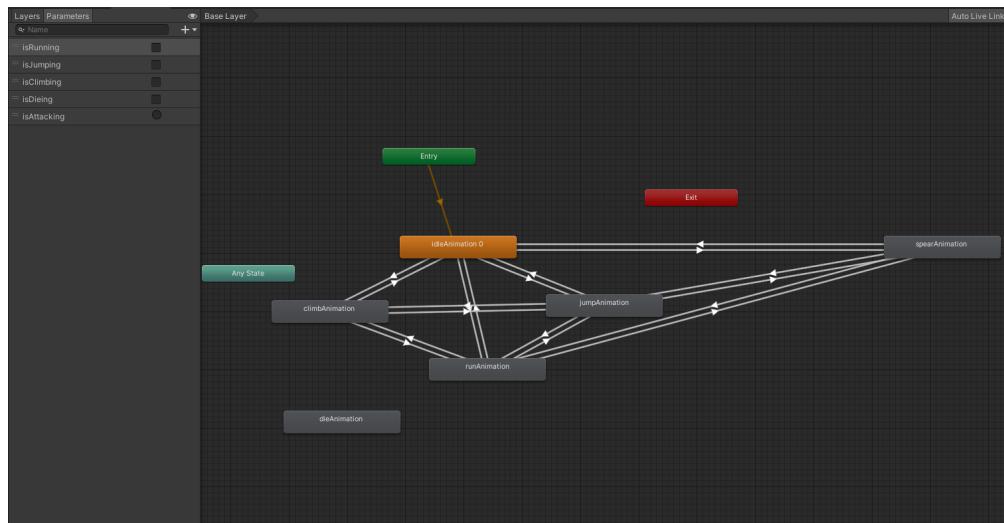


Abbildung 3.2: Hebel

4 Inbetriebnahme

Der Quellcode für das Spiel ist auf GitHub verfügbar und kann über den folgenden Link eingesehen werden.

<https://github.com/LeaMarieKindermann/Spieleprogrammierung>

Die Abgabe ist im Branch Äbgabe" hinterlegt.

Zusätzlich steht die ausführbare Datei (.exe) auf Google Drive zur Verfügung, welche vom Spieler einfach gestartet werden kann:

<https://drive.google.com/drive/folders/12l5Uj8EeFmnLL1I452tbvbxWBptW9aN0>

5 Evaluierung

In diesem Abschnitt beabsichtigen wir, unsere erzielten Ergebnisse zu evaluieren. Hierbei werden wir auf die erreichten Ergebnisse eingehen und gleichzeitig Herausforderungen vorstellen, die wir erfolgreich bewältigt haben.

5.1 Herausforderungen

Im Verlauf der Entwicklungsphase des Spiels traten diverse Herausforderungen auf, die in diesem Abschnitt näher erörtert werden. Die Integration der Gegner erwies sich als signifikante Aufgabe und brachte vielfältige Schwierigkeiten mit sich. Die Programmierung der Gegner, um sie zu vordefinierten Zeitpunkten auszulösen und den Spieler zu verfolgen, gestaltete sich als anspruchsvoll. Insbesondere führte die Animation beim Angreifen anfangs zu zeitlichen Verzögerungen, die überwunden werden mussten. Eine weitere Herausforderung bestand darin, sicherzustellen, dass die Hitbox während der Schlaganimation des Spielers korrekt verschoben wurde. In diesem Zusammenhang wurde eine zusätzliche Hitbox als Lösung implementiert, die während des Schlags aktiviert und anschließend deaktiviert wird.

Die Gewährleistung von Spannung und Abwechslung in jedem Level erforderte die geschickte Integration unterschiedlichster Mechaniken. Insbesondere stellte das Kletter-Feature im ersten Level eine Herausforderung dar, da der Spieler nur dann klettern sollte, wenn er die Leiter berührt. Obwohl dieses Ziel erreicht wurde, weist das Ergebnis eine gewisse Unflüssigkeit auf. Mechaniken wie das rhythmische Verschwinden von Blöcken und das Nachgeben des Bodens unter den Füßen beanspruchten ebenso einen erheblichen Zeitraum.

Eine zusätzliche betont anspruchsvolle Hürde stellte sich im Umgang mit dem Canvas von Unity dar. Insbesondere gestaltete sich der Einsatz zu Beginn als herausfordernd. Nach erfolgreicher Einarbeitung ergab sich die Schwierigkeit, dass sich die Benutzeroberfläche im Canvas wiederholt verschob. Diese Problematik konnte durch gezielte Verankerungsmaßnahmen erfolgreich gelöst werden.

5.2 Erreichte Ergebnisse

Das Spiel umfasst erfolgreich die geplante Struktur mit drei Levels und einem Bosskampf. Jedes Level ist abwechslungsreich gestaltet und nutzt unterschiedliche Spielmechaniken. Die Implementierung der sammelbaren Sterne und Münzen wurde erfolgreich realisiert. Sowohl die Gegner als auch der Spieler verfügen über qualitativ hochwertige Animationen. Die akustische Gestaltung durch Musik und Soundeffekte trägt effektiv zur Verbesserung des Spielgefühls bei. Besonders gelungen ist die Integration kleiner Details, wie die unterschiedlichen Klangfarben der Schritte des Spielers je nach Beschaffenheit des Bodens.

Die Gegner wurden erfolgreich in das Spiel integriert, begleitet von einer sinnvollen Lebensleiste für den Spieler. Die Hintergrundgestaltung ist detailreich und liebevoll ausgeführt. Der Bosskampf stellt den Höhepunkt des Spiels dar, und die Steigerung in den verschiedenen Levels wurde erfolgreich umgesetzt.

Die Umsetzung unserer ursprünglichen Spielidee verlief erfolgreich, wobei wir sogar mehr erreichen konnten als anfänglich geplant.

6 Quellen

6.1 Software

Unity Musik schneiden: <https://cliffeo.com/de>

6.2 Assets

<https://assetstore.unity.com/packages/2d/environments/2d-platformer-tileset-173155>

6.3 Musik

Schlag: <https://pixabay.com/de/sound-effects/thump-105302/> Level beendet: <https://pixabay.com/de/sound-effects/winbrass-39632/> Slime: <https://pixabay.com/de/sound-effects/slimejump-6913/>
Spinne: <https://pixabay.com/de/sound-effects/zombie-growl-3-6863/> Hebel: <https://pixabay.com/de/sound-effects/trigger-36813/> Feldermaus: <https://pixabay.com/de/sound-effects/flapping-39306/>
Hüpfen: <https://pixabay.com/de/sound-effects/cartoon-jump-6462/> Leiter: <https://pixabay.com/de/sound-effects/ladder-82581/> Münzen: <https://pixabay.com/de/sound-effects/collectcoin-6075/>
Sterne: <https://pixabay.com/de/sound-effects/fairy-dust-shimmer-1-175611/> Schritte Gras: <https://pixabay.com/de/sound-effects/footsteps-grass-1-6810/> Schritte Schnee: <https://pixabay.com/de/sound-effects/snow-walking-sound-6009/> Schritte Stein: <https://pixabay.com/de/sound-effects/steps-in-corridor-104456/>

Hauptmenü: <https://www.youtube.com/watch?v=foC7UgnkfGM> Level1 und Levelmenü: <https://www.youtube.com/watch?v=wUrUHFtWLy0> Boss: <https://www.youtube.com/watch?v=SYTS2>
Level3: <https://www.youtube.com/watch?v=XGcYPyZRyTY> Level2: <https://www.youtube.com/watch?v=f>