

Miniprojekt Unity/VR

Maze VR Dokumentation

Teilnehmer: Robin Röcker (80193), Christian Schmidt(79557)

Datum: 12.2.2023

Inhaltsverzeichnis

1. Einleitung	3
2. Entwicklungsschritte	3
<i>Ideenphase</i>	<i>3</i>
<i>Ideeneinschränkung.....</i>	<i>3</i>
<i>Entscheidungsphase.....</i>	<i>3</i>
<i>Einarbeitung in Unity.....</i>	<i>3</i>
<i>Designphase</i>	<i>4</i>
<i>Funktionalität hinzufügen</i>	<i>4</i>
<i>Spielmechaniken implementieren</i>	<i>4</i>
<i>Finale Version.....</i>	<i>4</i>
3. Funktionale Skripte	5
<i>UIQuit.....</i>	<i>5</i>
<i>TimeBlock</i>	<i>6</i>
<i>TimeBlockUI.....</i>	<i>7</i>
<i>TimeMechanic.....</i>	<i>8</i>
<i>Klassendiagramm.....</i>	<i>9</i>
4. Spielmechaniken	10
<i>Einsammeln der „Diamonds“</i>	<i>10</i>
<i>Ablauf der Zeit</i>	<i>10</i>
<i>Gewinnen</i>	<i>10</i>
<i>Verlieren</i>	<i>10</i>
5. Ablaufdiagramm	11

1. Einleitung

Im Zuge der Vorlesung „Virtuelle Realität und Animation“, haben wir eine VR-Anwendung mit der Unity-Engine entwickelt. Für die Entwicklung der Anwendung werden sowohl externe als auch selbst erstellte Assets eingebunden und verwendet. Des Weiteren stellt die Anwendung verschiedene Spielmechaniken bereit, die Interaktionen mittels einer VR-Brille und den dazugehörigen Controllern ermöglicht.

In den folgenden Abschnitten wird das entwickelte Spiel „MazeVR“ genauer beschrieben. Dabei werden sowohl die jeweiligen Spielmechaniken als auch die durchlaufenen Entwicklungsschritte genau beschrieben. Außerdem werden die erstellten C#-Skripte erläutert und deren Zusammenspiel mit den Spielobjekten erklärt.

2. Entwicklungsschritte

Ideenphase

In dieser Phase haben wir uns mit der VR-Thematik auseinandergesetzt und uns ein Bild davon gemacht, was unter diesem Kontext verstanden wird. Wir haben uns darüber informiert, wie solche Anwendungen entwickelt werden und welche Technologien dazu benötigt werden. Ausgehend von diesen Recherchen, haben wir uns Gedanken über ein Projektthema gemacht. Es wurde schnell klar, dass das Projekt in die Richtung eines „Jump N Run“- Spiels gehen wird. So haben wir die möglichen Themen auf ein Parkour-Spiel und ein Labyrinth-Spiel eingegrenzt.

Ideeneinschränkung

Bei der Themenvorstellung hatten wir zunächst damit gerechnet, dass wir das Parkour-Spiel umsetzen wollen. Jedoch wurde uns schnell klar, dass dies nicht so einfach wie gedacht umzusetzen ist. Dabei haben Themen wie Collider sowie Teleportation eine erhebliche Rolle gespielt.

Da wir nun ein verständlicheres Bild vom Umfang einer VR-Anwendung hatten, haben wir uns dazu entschieden, noch Mal über die Option eines Themenwechsels zu sprechen.

Entscheidungsphase

Wie bereits angesprochen, haben wir uns über die Option eines Themenwechsels unterhalten. Dabei haben wir den Entschluss gefasst, das bisherige Thema eines Parkour-Spiels zu verwerfen und unsere andere Idee, das Labyrinth-Spiel, zu wählen. Somit haben wir zum Zeitpunkt der Zwischenpräsentation einen Stand vorzeigen können, der die grundlegenden Inhalte eines Labyrinth-Spiels beinhaltet. Dazu gehört zunächst die Steuerung über die Tastatur und Maus. Außerdem konnte bereits das Polygon City Pack vorgestellt werden, das die Landschaftlichen Elemente für das Labyrinth bereitstellt. Der Name des Spiels wurde auf „MazeVR“ festgelegt und soll ein Labyrinth mit einem Stadt-Theme beinhalten.

Einarbeitung in Unity

Die grundlegende Bedienung von Unity haben wir durch die Entscheidungsphase zwar erlernt, jedoch gilt es nun diese zu vertiefen, um so beispielsweise das Design sowie die Spielmechaniken einzubinden.

In Zuge dessen, haben wir uns beigebracht, wie verschiedene Objekte erstellt werden und ihnen ein C#-Skript zugewiesen wird. Des Weiteren haben wir gelernt, wie man Komponenten einem Objekt hinzufügt und diverse Eigenschaften ändert.

Designphase

In der Designphase haben wir das Labyrinth und das Spielmenü entworfen. Wie bereits erwähnt, haben wir für das Labyrinth das Asset-Packet Polygon City Pack verwendet. Dieses stellt unterschiedlichste Gebäude zur Verfügung, die wir verwendeten, um das Labyrinth zu gestalten. Im Grunde wurden diese so platziert, dass die Spielfigur nicht zwischen Ihnen, dem Labyrinth „entfliehen“ kann.

Für das Menü war ursprünglich ein Canvas-Element geplant, über das der Spieler das Spiel betreten und wieder verlassen kann. Im Verlauf der Entwicklung sind wir aber darauf aufmerksam geworden, dass dies zum einen nicht ohne Weiteres Umsetzbar ist und zum anderen nicht die Natur eines VR-Spiels unterstützt.

Da uns ebenfalls bewusst geworden ist, dass nur das Laufen durch ein Labyrinth etwas eintönig und langweilig sein kann, haben wir uns dazu entschieden, eine weitere Spielmechanik zu implementieren. Aus diesem Grund haben wir zusätzlich Diamanten in das Spiel eingefügt, die vom Spieler eingesammelt werden können. Außerdem ist das Ziel nun nicht mehr, das Ende des Labyrinths zu finden, sondern alle Diamanten innerhalb einer gewissen Zeit, in dem Labyrinth zu finden. Dabei wird die Zeit jedes Mal erhöht, wenn ein Diamant eingesammelt wurde.

Funktionalität hinzufügen

Da nun die Regeln des Spiels klar definiert sind, war es an der Zeit die Funktionalitäten in das Spiel zu integrieren. Dazu haben wir zunächst die Bewegungselemente hinzugefügt. Dafür haben wir das XR Interaction Toolkit verwendet. Es hat uns mit vordefinierten Skripten versorgt, die wir benötigten, um die Bewegungssteuerung mit der Oculus-VR-Brille zu realisieren.

Spielmechaniken implementieren

Nun da das Design und die grundlegende Steuerung funktioniert, haben wir angefangen die Spielmechaniken zu implementieren. Dazu haben wir Skripte geschrieben, die beispielsweise das Aufsammeln von Diamanten ermöglicht und dabei die verbleibende Zeit erhöht.

Da außerdem noch kein Menü vorhanden ist, da der ursprüngliche Ansatz mit einem Canvas-Objekt nicht funktioniert hat, haben wir uns dazu entschieden der Spielfigur eine Art Armband zu geben, die zum einen alle benötigten Spielinformationen enthält und zum anderen einen Button besitzt, über den das Spiel verlassen werden kann. Dazu haben wir, wenn man so will ein eigenes Asset entworfen. Es besteht aus einem Canvas und beinhaltet Bestandteile wie Texte, Buttons und Images. Dies haben wir dann der Linken Hand hinzugefügt, sodass diese Elemente jeder Zeit während des Spiels sichtbar sind.

Finale Version

Wir haben nun alle Funktionalitäten umgesetzt und haben dabei die Erkenntnis gemacht, dass wir vergessen haben eine Sieg-Option bereitzustellen. Bis lang war die Aufgabe, alle Diamanten in der vorgegebenen Zeit zu finden. Ist einem das nicht gelungen, war das Spiel zu ende. Aus diesem Grund haben wir die Funktion so erweitert, dass der Spieler über das Armband einsehen kann, wie viel Zeit ihm noch verbleibt, wie viele Diamanten er gefunden hat und wie viele ihm noch fehlen. Zusätzlich verfügt das Armband über einen Statustext, der den Spieler darauf aufmerksam macht, wann er gewonnen oder verloren hat.

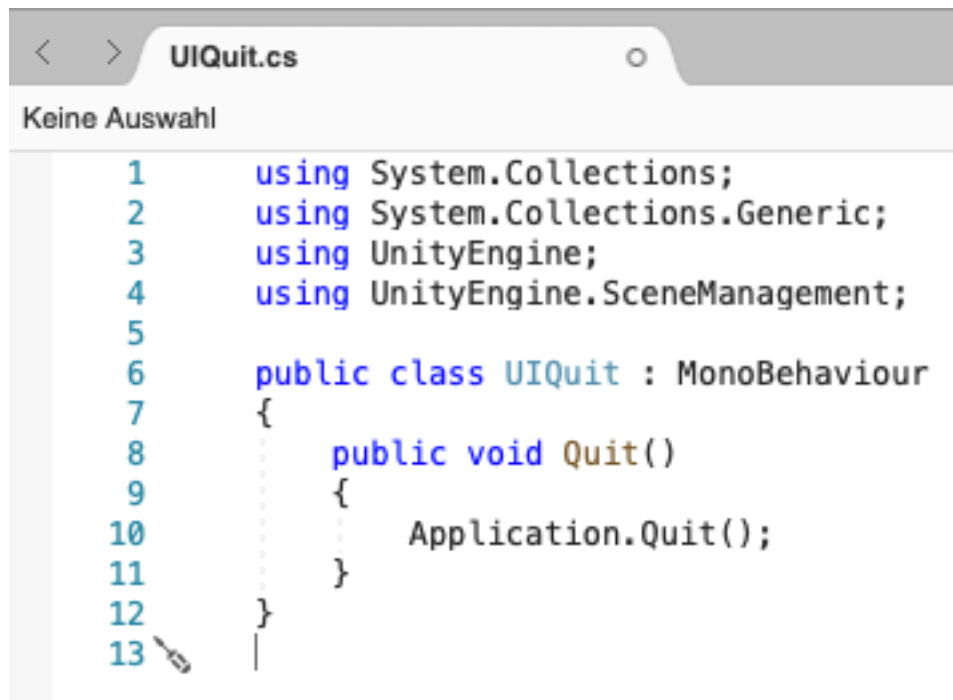
Mit dem Hinzufügen dieser Funktion, haben wir all unsere Anforderungen an das System umsetzen können und die Anwendung erfolgreich bauen und testen können.

3. Funktionale Skripte

Das Programm MazeVR nutzt drei Skripte. TimeBlock, TimeBlockUI und TimeMechanic.

UIQuit

Das Skript UIQuit enthält die Klasse UIQuit. Die Klasse besitzt eine Methode Quit. Diese Methode beendet das Spiel und wird durch das Drücken des Exit-Buttons ausgeführt.



The screenshot shows a code editor window titled "UIQuit.cs". The editor contains the following C# code:

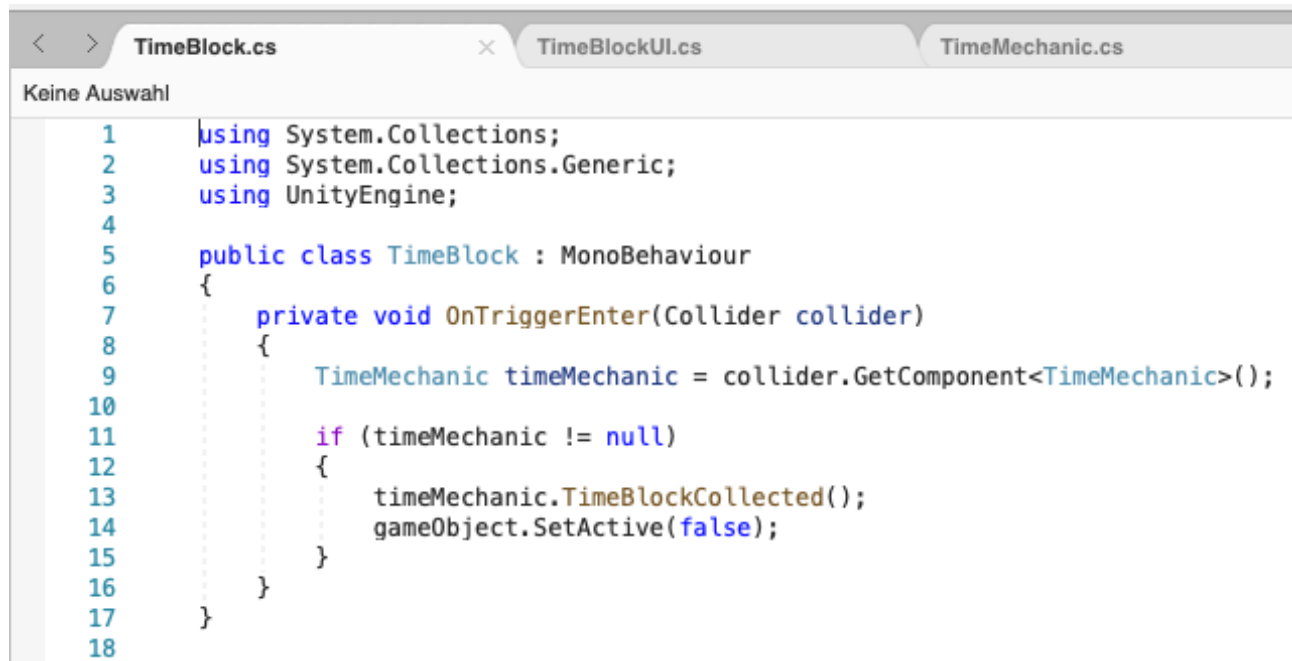
```
1 using System.Collections;
2 using System.Collections.Generic;
3 using UnityEngine;
4 using UnityEngine.SceneManagement;
5
6 public class UIQuit : MonoBehaviour
7 {
8     public void Quit()
9     {
10         Application.Quit();
11     }
12 }
13
```

The code is displayed with line numbers on the left. The editor interface includes a tab at the top, a status bar showing "Keine Auswahl", and a vertical scrollbar on the right.

TimeBlock

Das Skript TimeBlock enthält eine Klasse TimeBlock. Diese Klasse behandelt die Kollision des Players mit einem Diamond.

Sie beinhaltet eine Methode OnTriggerEnter. Diese Methode wird aufgerufen, wenn der Spieler mit einem Diamond kollidiert. Im ersten Schritt überprüfen wir, ob das Objekt, mit dem der Diamond kollidiert, der Spieler ist. Dies tun wir, indem wir überprüfen, ob die TimeMechanic Komponente von dem Objekt, das mit dem Diamond kollidiert, null ist. Falls die Komponente nicht null ist, handelt es sich bei dem Objekt um den Spieler. Im zweiten Schritt rufen wir die TimeBlockCollected Methode auf. Im letzten Schritt setzen wir gameObject.SetActive() auf false um den Diamond nach der Kollision auf inaktiv zu setzen.

The image shows a screenshot of a code editor with three tabs: 'TimeBlock.cs', 'TimeBlockUI.cs', and 'TimeMechanic.cs'. The 'TimeBlock.cs' tab is active, and the text 'Keine Auswahl' (No selection) is visible above the code. The code is written in C# and defines a 'TimeBlock' class that inherits from 'MonoBehaviour'. It includes three 'using' statements at the top: 'using System.Collections;', 'using System.Collections.Generic;', and 'using UnityEngine;'. The class contains a private method 'OnTriggerEnter' that takes a 'Collider' parameter. Inside this method, it gets a 'TimeMechanic' component from the collider. If the component is not null, it calls 'TimeBlockCollected()' on the 'TimeMechanic' object and then sets 'gameObject.SetActive(false)'. The code is numbered from 1 to 18 on the left side of the editor.

```
1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4
5  public class TimeBlock : MonoBehaviour
6  {
7      private void OnTriggerEnter(Collider collider)
8      {
9          TimeMechanic timeMechanic = collider.GetComponent<TimeMechanic>();
10
11          if (timeMechanic != null)
12          {
13              timeMechanic.TimeBlockCollected();
14              gameObject.SetActive(false);
15          }
16      }
17  }
18
```

TimeBlockUI

Das Skript TimeBlockUI beinhaltet eine Klasse TimeBlockUI. Diese Klasse behandelt zwei Themen. Das Anzeigen der Werte in der UI und das Auslösen von Gewinnen oder Verlieren.

In der Start Methode rufen wir die drei Text Komponenten ab und setzen sie jeweils auf die Properties timeText, collectionText und statusText. timeText beinhaltet die verbliebene Zeit. collectionText beinhaltet die Anzahl der eingesammelten Diamonds und statusText beinhaltet den Status des aktuellen Spieles d.h gewonnen / verloren.

Die Klasse besitzt eine Methode UpdateTimeBlockText in ihr wird der collectionText und timeText der jeweilige aktuelle Wert zugeordnet.

Falls der Spieler durch das Einsammeln des Diamonds nun 30 Diamonds besitzt, wird der statusText auf „You Won“ gesetzt und nach 5 Sekunden das Spiel beendet.

Des Weiteren wird überprüft, ob die NumberOfTimeBlocks null erreicht hat. Falls ja, ist die Spielzeit abgelaufen. In Folge dessen wird der StatusText auf „Game Over“ gesetzt und nach fünf Sekunden das Spiel beendet.



```
< > TimeBlock.cs TimeBlockUI.cs TimeMechanic.cs
TimeBlockUI ▶ UpdateTimeBlockText(TimeMechanic timeMechanic)
1 using System.Collections;
2 using System.Collections.Generic;
3 using System.Threading.Tasks;
4 using UnityEngine;
5 using TMPro;
6
7 public class TimeBlockUI : MonoBehaviour
8 {
9     public TextMeshProUGUI timeText;
10    public TextMeshProUGUI collectionText;
11    public TextMeshProUGUI statusText;
12
13    void Start()
14    {
15        timeText = GetComponent<TextMeshProUGUI>();
16        collectionText = GetComponent<TextMeshProUGUI>();
17        statusText = GetComponent<TextMeshProUGUI>();
18    }
19
20    public async void UpdateTimeBlockText(TimeMechanic timeMechanic)
21    {
22        collectionText.text = $"{timeMechanic.CollectionCount.ToString()} / 30";
23        timeText.text = timeMechanic.NumberOfTimeBlocks.ToString();
24
25        if (timeMechanic.CollectionCount == 30)
26        {
27            statusText.text = "You Won";
28            await Task.Delay(5000);
29            Application.Quit();
30            return;
31        }
32        if (timeMechanic.NumberOfTimeBlocks == 0)
33        {
34            statusText.text = "Game Over";
35            await Task.Delay(5000);
36            Application.Quit();
37            return;
38        }
39    }
40 }
41
```

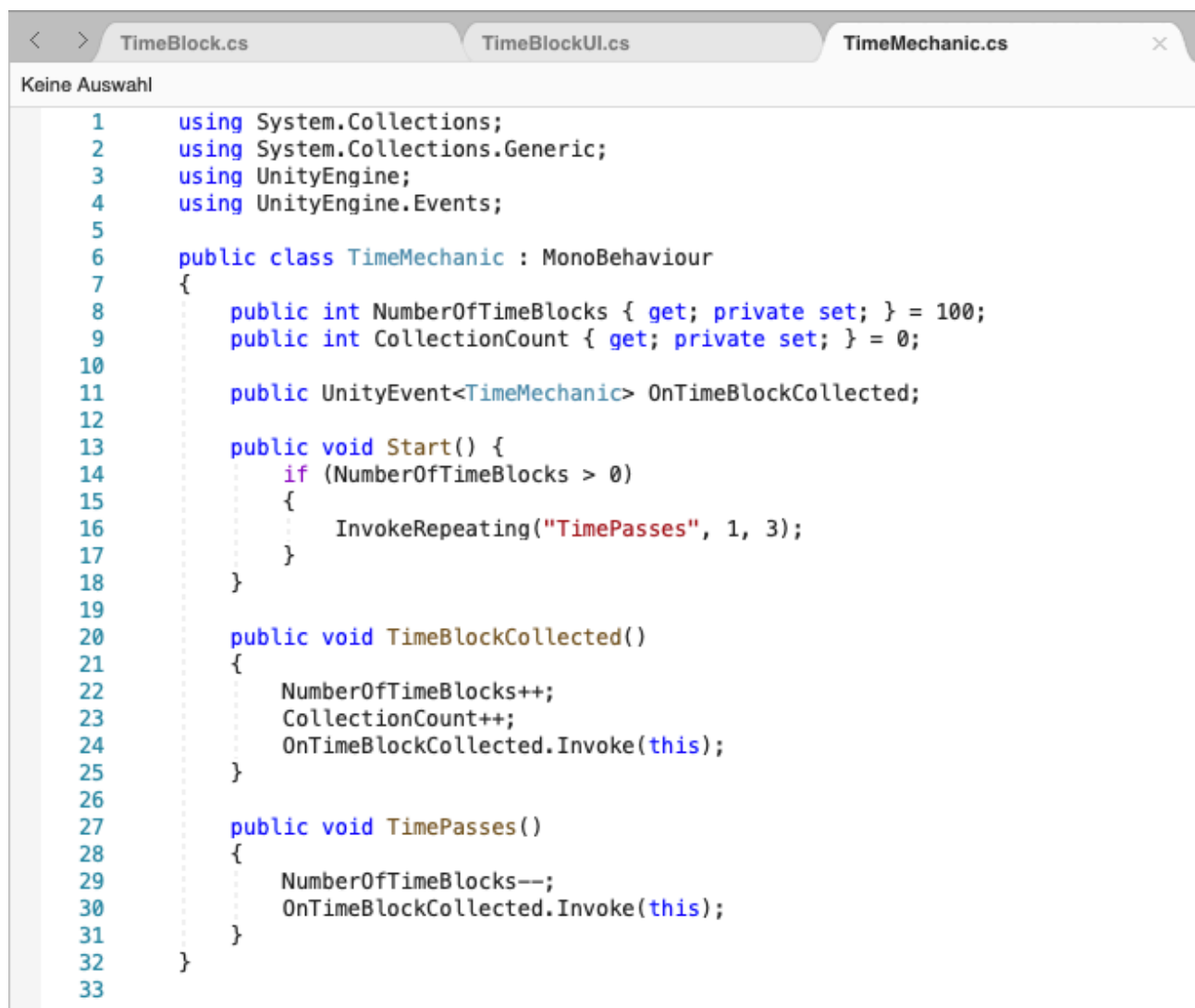
TimeMechanic

Das Skript TimeMechanic besitzt eine Klasse TimeMechanic. Sie besitzt drei Properties NumberOfTimeBlocks, CollectionCount und OnTimeBlockCollected.

NumberOfTimeBlocks beinhaltet die verbliebene Zeit und CollectionCount zählt die Anzahl der eingesammelten Diamonds. Bei OnTimeBlockCollected handelt es sich um ein UnityEvent.

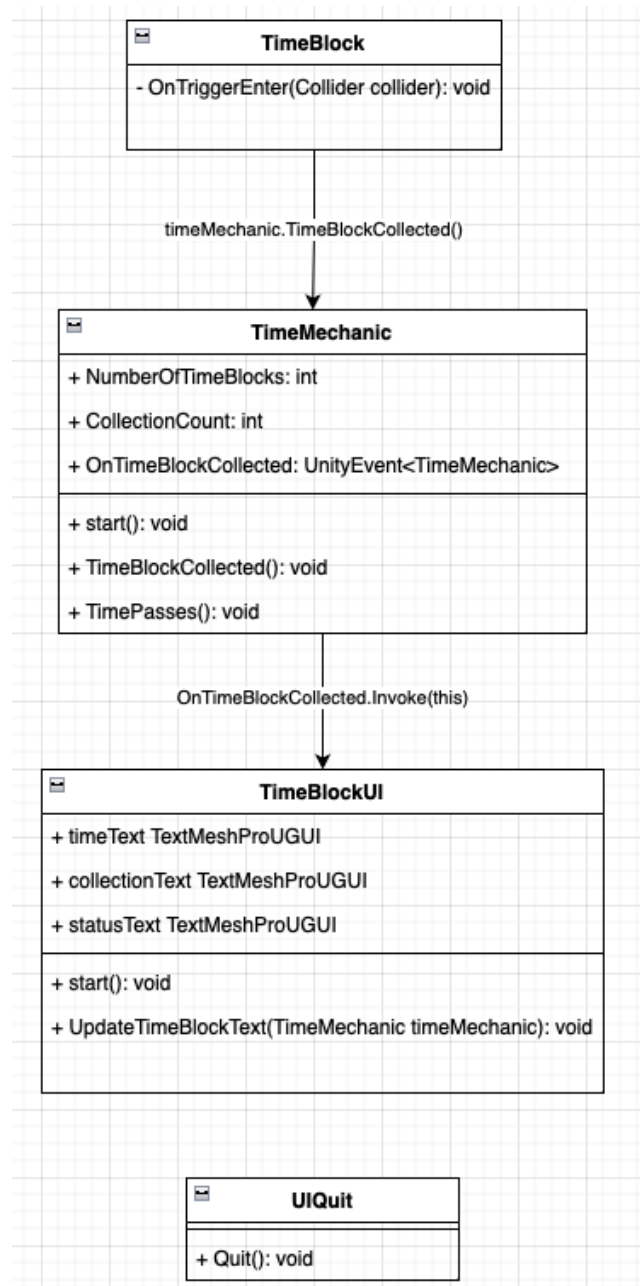
Die Methode TimeBlockCollected erhöht die Anzahl von CollectionCount und NumberOfTimeBlocks um eins. Außerdem triggert die Methode das Event OnTimeBlockCollected.

Die Methode TimePasses wird durch Start jede drei Sekunden aufgerufen. In ihr wird die NumberOfTimeBlocks um den Wert 1 erniedrigt und das Event OnTimeBlockCollected getriggert.



```
1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4  using UnityEngine.Events;
5
6  public class TimeMechanic : MonoBehaviour
7  {
8      public int NumberOfTimeBlocks { get; private set; } = 100;
9      public int CollectionCount { get; private set; } = 0;
10
11     public UnityEvent<TimeMechanic> OnTimeBlockCollected;
12
13     public void Start() {
14         if (NumberOfTimeBlocks > 0)
15         {
16             InvokeRepeating("TimePasses", 1, 3);
17         }
18     }
19
20     public void TimeBlockCollected()
21     {
22         NumberOfTimeBlocks++;
23         CollectionCount++;
24         OnTimeBlockCollected.Invoke(this);
25     }
26
27     public void TimePasses()
28     {
29         NumberOfTimeBlocks--;
30         OnTimeBlockCollected.Invoke(this);
31     }
32 }
33
```


Klassendiagramm



4. Spielmechaniken

Einsammeln der „Diamonds“

Der Spieler kann „Diamonds“ in dem Labyrinth einsammeln. Durch das Einsammeln eines „Diamonds“ erhöht sich der Zähler der „Diamonds“ um den Wert eins und der Zähler der verbleibenden Zeit erhöht sich um eine Sekunde.

Ablauf der Zeit

Zum Start des Spieles wird die Zeit auf 100 gesetzt. Jede drei Sekunden sinkt die Zeit um eins. Durch das Einsammeln von „Diamonds“ kann die Zeit erhöht werden.

Gewinnen

Der Spieler kann das Spiel gewinnen indem er alle 30 „Diamonds“ in der gegebenen Zeit einsammelt. Nach dem Gewinnen des Spiels wird dem Spieler angezeigt das er gewonnen hat und das Spiel beendet.

Verlieren

Der Spieler kann das Spiel verlieren, wenn er in der gegebenen Zeit es nicht schafft alle 30 „Diamonds“ einzusammeln.

Nach dem Verlieren des Spiels wird dem Spieler angezeigt das er verloren hat und das Spiel beendet.

5. Ablaufdiagramm

