Hochschule für Angewandte Wissenschaften Hamburg
Fakultät Design, Medien und Information
Department Medientechnik

KairosVR: Erstellung eines Assets zur Zeitmanipulation durch Gestensteuerung in Virtual Reality

Fakultät Design, Medien und Information
Department Medientechnik
der Hochschule für Angewandte Wissenschaften Hamburg

Bachelor-Thesis

Zur Erlangung des akademischen Grades
Bachelor of Science

vorgelegt von

Thomas Pommerening
Matrikelnummer: [redacted]

Erstprüfer: Prof. Dr. R. Greule

Zweitprüfer: Dipl. Ing. (FH) M. Kuhr

Hamburg, 17.01.2020

Eidesstattliche Erklärung

Hiermit versichere ich, dass ich die vorliegende Bachelor-Thesis mit dem Titel: KairosVR: Erstellung eines Assets zur Zeitmanipulation durch Gestensteuerung in Virtual Reality selbstständig und nur mit den angegebenen Hilfsmitteln verfasst habe. Alle Passagen, die ich wörtlich aus der Literatur oder aus anderen Quellen wie z.B. Internetseiten übernommen habe, habe ich deutlich als Zitat mit Angabe der Quelle kenntlich gemacht.

Datum: _____ Unterschrift: _____

Kurzzusammenfassung/Abstract

Diese Bachelorarbeit beschäftigt sich mit der Erstellung des Assets ‚KairosVR' für die Unity-Engine. Dieses Asset soll bekannte Spielmechaniken umsetzen, die sich mit Zeitmanipulation beschäftigen. Es soll schnell und einfach in ein Projekt eingefügt und benutzt werden können, um diese Mechaniken nutzbar zu machen. Bei Bedarf soll der Benutzer in der Lage sein können, Gestensteuerung in Virtual Reality einzusetzen, es aber nicht müssen.
Diese Arbeit analysiert zunächst bekannte Spielmechaniken und beschreibt anschließend die Umsetzung in Unity. Ergänzt wird der ganze Prozess durch die Anbindung an SteamVR und den Bau einer Testszene.

Inhaltsverzeichnis

Eidesstattliche Erklärung....2

Kurzzusammenfassung/Abstract....3

1 Grundlagen....6

1.1 Namensgebung....6

1.2 Unity....6

1.2.1 SteamVR Plugin Unity....7

1.3 Blender....7

1.4 Virtual Reality....7

1.4.1 Geschichte der Virtual Reality....7

1.4.2 Oculus Rift....8

1.4.3 HTC VIVE....9

2 Mechaniken in Videospielen....11

2.1 Prince of Persia: The Sands of Time....11

2.2 SUPERHOT....12

2.3 Thinking with Time Machine....12

2.4 Mechanik BulletTime....13

3 Umsetzung der Mechaniken in Unity....13

3.1 Grundmechanik....13

3.2 Prince of Persia: The Sands of Time....14

3.3 SUPERHOT....15

3.4 Thinking with Time Machine....16

3.5 Mechanik BulletTime....16

4 VR Gestensteuerung....16

4.1 Definition....16

4.2 Umsetzung der Gestensteuerung....16

4.3 Anbindung der Gestensteuerung an die Mechaniken....18

5 Bau der Testszene 18
5.1 Version 1 18
5.2 Probleme von Version 1 20
5.3 Version 2 33
5.4 Probleme von Version 2 37
5.5 Version 3 38
5.6 Probleme von Version 3 47
5.7 Version 4 49
6 Fazit/Zukunftsausblick 54
7 Abbildungsverzeichnis 55
8 Literaturverzeichnis 57

1 Grundlagen

1.1 Namensgebung

Kairos ist ein griechischer Gott. Im Gegensatz zu Chronos, welcher den unaufhaltsamen Fluss der Zeit symbolisiert, ist Kairos das Sinnbild für den idealen Moment. Eine lange Haarlocke hängt ihm ins Gesicht, die gepackt werden kann, sobald er an einem vorbeiläuft. Diese Locke gilt als Ursprung des Sprichwortes „Die Gelegenheit beim Schopfe packen“. Doch im Gegensatz zu seiner langen Locke ist Kairos Hinterkopf kahl. Somit ist es unmöglich ihn noch zu erwischen, wenn man nicht rechtzeitig zugegriffen hat. Der ideale Moment wird somit verpasst.

Abbildung 1 Kairos-Relief von Lysippos, Kopie in Trogir

Das Asset, das im Laufe dieser Bachelorarbeit entsteht, trägt den Namen KairosVR. Um den idealen Moment zu erschaffen und zu erkennen, ist die Fähigkeit der Zeitkontrolle durchaus sehr nützlich: Es gilt, nicht der verpassten Gelegenheit hinterherzulaufen, sondern sie sich zurückzuholen.

1.2 Unity

Die Erstellung und Nutzung von KairosVR findet in der Unity-Engine statt. Während des Studiums wurde einem die Unity-Engine vorgestellt und man hatte genug Zeit, sich mit ihr auseinanderzusetzen, da man sie bereits für viele eigene Projekte benutzt hat.

Unity ist eine kostenlose Echtzeit-Entwicklungsplattform oder auch Game-Engine von Unity Technologies. Ursprünglich wurde Unity Technologies unter dem Namen ‚Over The Edge‘ 2004 in Kopenhagen gegründet und wurde 2006 zum aktuellen ‚Unity Technologies‘ umbenannt. Unity selbst erschien am 8. Juni 2005 unter dem Namen Unity3D.

Unity kann genutzt werden, um Videospiele oder Anwendungen für Windows, OS X oder Linux zu entwickeln. Des Weiteren unterstützt Unity Videospielkonsolen wie die ‚Playstation

4‘ von Sony Interactive Entertainment oder die ‚Switch‘ von Nintendo. Es werden Mobilgeräte sowie Virtual Reality Devices wie beispielsweise die ‚HTC VIVE‘ von HTC und Valve unterstützt, aber auch Augmented Reality Devices wie die ‚HoloLens‘ von Microsoft. Programmiert wird in Unity in einer leicht abgewandelten Fassung der Programmiersprache C#, es ist somit objektorientiert. Unity nutzt als Entwicklungsumgebung im Normalfall ‚Visual Studio‘ von Microsoft, sofern es auf einem Windows Betriebssystem genutzt wird.

1.2.1 SteamVR Plugin Unity

Mit dem SteamVR Plugin brachte Valve eine umfangreiche Erweiterung für Unity auf den Markt. Dieses kostenlose Asset dient als Bindeglied zwischen Unity und VR. Im Asset enthalten ist eine unkomplizierte und direkte Verknüpfung zur Software SteamVR, welche kostenlos über Steam, die Spiele- und Softwareplattform von Valve, verfügbar ist. Zusätzlich werden 3D-Modelle der Controller oder deren alternative Darstellung in Form von Händen beigelegt, um das Orientieren in der virtuellen Realität zu erleichtern. Das Asset enthält die wichtigsten Grundskripte für die bequeme Bedienung in VR wie bspw. die Teleportation oder die Interaktion mit einem Gegenstand. Das neue verbesserte ActionSet-System von SteamVR ermöglicht das leichte Anbinden an alle möglichen VR-Headsets und Controller und bietet den Nutzern volle Kontrolle über die Tastenverteilung. Es können ganz einfach eigene ActionSets geschrieben und an Controller gebunden werden.

1.3 Blender

Blender ist eine kostenlose OpenSource 3D-Modelliersoftware von The Blender Foundation. The Blender Foundation ist eine in 2002 gegründete ‚independent public benefit‘- Organisation, welche die ‚free OpenSource 3D creation pipeline‘ Blender wartet und updated. Mit Blender können nicht nur 3D-Modelle erstellt, sondern auch geriggt werden. Animationen lassen sich ebenfalls erstellen. Die meisten 3D-Modelle in KairosVR sind durch eigene Hand in Blender entstanden. Bei den wenigen Ausnahmen handelt es sich entweder um Standartobjekte wie Würfel, die Unity eigens liefert, oder um Modelle wie die Handschuhe, die dem Nutzer durch Hinzufügen von ‚SteamVR‘ zur Verfügung stehen.

1.4 Virtual Reality

1.4.1 Geschichte der Virtual Reality

Die Idee der Virtual Reality (kurz VR) existiert schon seit dem 19. Jahrhundert. 1838 erstellte Charles Whatstone ein ‚Stereoscope‘ welches zeigte, dass das menschliche Gehirn mithilfe von zwei zweidimensionalen Bildern, die jeweils nur einem Auge gezeigt werden, ein

dreidimensionales Objekt mit Tiefe erzeugen kann. Dies kann als Grundstein der modernen VR-Technik gesehen werden.
Ein weiterer Vorgänger ist der ‚Link Trainer' von Ed Link aus dem Jahr 1929, welcher der erste Flugsimulator für Piloten im Training war. Auch wenn diese Erfindung keine visuelle Komponente besitzt, gilt sie dennoch als ein Meilenstein zur Erstellung noch immersiverer Maschinerie.
1968 wurde das erste ‚Head-mounted display' (kurz HMD) mit dem Namen ‚The Sword of Damocles' von Ivan Sutherland und Bob Sproull gebaut. Dieses HMD zeigte seinem Träger ein schachbrettartiges Muster durch ein ‚see-through' Display. Dieses Muster änderte seine Perspektive je nach Bewegung des Kopfes des Trägers. Doch das enorme Gewicht dieses HMDs konnte nur durch Unterstützung eines mechanischen Armes getragen werden und so endete es als nichts weiter als ein Laborexperiment.
Mitte der 1980er wurde der Begriff ‚virtual reality' von Jaron Lanier geprägt. Er versuchte den Mainstream für VR zu gewinnen, indem er durch das von ihm gegründete Unternehmen ‚VPL Research' die VR-Entwicklung unterstützte. Durch die ‚Virtual Programming Language' (kurz VPL) sollten mehr VR-Anwendungen entstehen und somit den Verkauf für VR-Brillen und Handschuhe durch sein Unternehmen ankurbeln. Doch die Firma scheiterte und ging Bankrott im Jahr 1990.
Im Jahr 1993 versuchte die Videospielefirma Sega mit einem eigenen Modell den VR-Markt zu erreichen. Die ‚Sega VR' sollte 1993 entwickelt werden und ursprünglich 1994 mit vier Spielen für die ‚Sega Genesis/Mega Drive'-Konsole erscheinen. Doch floppte die Brille schon als Prototyp. Eine weitere Firma, die am VR-Konzept scheiterte, war Nintendo. Der ‚Nintendo Virtual Boy' wurde 1995 auf den Markt gebracht, doch blieb er aufgrund seines monochromen Displays und der unkomfortablen Nutzposition erfolglos. Lange Zeit wurde es nun still um das Thema VR, dennoch gaben nicht alle das Thema auf.
Im Jahr 2012 startete eine Kickstarter Kampagne, die die ‚Moderne VR-Industrie' einleitete. Diese Kampagne wurde von der damals neuen Firma Oculus VR erstellt und brachte das Zehnfache des ursprünglichen Zieles ein, nämlich 2,5 Millionen Dollar statt den erhofften 250.000. Zwei Jahre später kaufte Facebook die Firma Oculus VR für 2,5 Milliarden Dollar.

1.4.2 Oculus Rift

Die Oculus Rift wurde 2012 von Oculus VR entwickelt und durch Kickstarter finanziert. Allerdings verzögerte sich das Erscheinen der Brille für die Verbraucher bis zum 28.03.2016. Bei der Oculus Rift handelt es sich um den ersten populären Ableger der VR-Brillen. Das ‚Head-Tracking' der Oculus Rift funktioniert über interne Sensoren und eine Infrarotkamera.

Bei den eingebauten Sensoren handelt es sich zum einen um ein sogenanntes 3-Achsen-Gyrometer: ein Gerät, welches Drehbewegungen messen kann, einige Beschleunigungssensoren und ein Magnetometer, welches die magnetische Flussdichte messen kann. Zusätzlich wurde in der Ursprungsfassung der Oculus Rift eine Infrarotkamera benötigt, die - wie die Oculus Rift selbst - an den Computer angeschlossen werden muss. Zusammen sorgen diese Sensoren für eine schnelle Erkennung der Blickrichtung bzw. des Blickwinkels, um ein möglichst authentisches Bild zu zeigen. Das Display der Oculus Rift bietet eine Auflösung von 1080 x 1200 Pixeln pro Auge und hat ein Sichtfeld von 110 Grad. Als Controller können Xbox-Controller oder die etwas später erschienenen Oculus Touch Controller verwendet werden. Inzwischen sind weitere Generationen der Oculus Rift erschienen, die sich auf unterschiedliche Kundengruppen spezialisiert haben. Diese neuen Varianten verzichten nun auf den externen Sensor und sind somit schneller überall einsatzbereit. Die günstigste Variante, die Oculus Go, wird nur per App auf dem Smartphone und einen Controller gesteuert. Jedoch besitzt die Oculus Go nur eine Bildwiederholungsrate von 60 Hz. Eine weitere Variante dieses HMDs ist die Oculus Quest. Die Oculus Quest ist wie die Oculus Go ohne PC, sondern per App zu benutzen. Sie unterstützt zwei Touch Controller der Oculus Rift und hat eine Bildwiederholungsrate von 72 Hz. Die letzte Variante, die Oculus Rift S, ist wiederum auf einen leistungsstarken PC angewiesen. Sie kommt mit einer Auflösung von 1280 x 1440 Pixeln und einer Bildwiederholungsrate von 80 Hz und ist somit das stärkste Model der Oculus VR-Brillen.

Abbildung 2 Oculus Rift Consumer Version 1

1.4.3 HTC VIVE

Die HTC VIVE ist eine Virtual Reality Brille, entwickelt von HTC in Kooperation mit Valve. Sie kommt mit zwei Sensoren, genannt Basis Stationen, und zwei Controllern. Bei dieser VR-Brille handelt es sich um das erste Device, welches das Konzept vom Roomscaling anwendet.

Die Basis Stationen decken einen Bereich von 2m x 1,5m bis hin zu 3,5m x 3,5m ab. Dieser Bereich wird bei Erstbenutzung noch einmal mit einem der Controller in der Hand umrundet, um sicherzustellen, dass keine Wände oder Schränke im Weg sind. Der so markierte Bereich wird dem Benutzer nun als eine Art Käfig dargestellt, falls er dem Rand nahekommt. Dies soll verhindern, dass der Benutzer während des Tragens des Headsets gegen Hindernisse stößt.
Die Controller liegen gut in der Hand und alle Tasten sind ohne Probleme zu erreichen, ohne extra nach ihnen suchen zu müssen. Sie sind kabellos und besitzen eine Akkulaufzeit von ca. sechs Stunden. Das Headset selbst besitzt eine Auflösung von 1080 x 1200 Pixeln pro Auge mit einer Bildwiederholungsrate von 90 Hz. Die Displays bieten zusammen ein Sichtfeld von 110 Grad. In späteren Modellen der HTC VIVE wie der ‚VIVE Pro', der ‚HTC Cosmos' oder dem ‚Index Headset' von Valve sind wesentlich hochwertigere Displays verbaut, was den ‚Fliegengittereffekt', den Nutzer der originalen HTC VIVE kennen, minimieren soll. Des Weiteren besitzen die neuen Modelle ein größeres Sichtfeld, was für mehr Immersion in Spielen sorgen soll.

Abbildung 3 HTC VIVE: (Links) Base Stations, (Mitte) Headset und (Rechts) Controllers

2 Mechaniken in Videospielen

2.1 Prince of Persia: The Sands of Time

Prince of Persia: The Sands of Time ist ein Spiel entwickelt und gepublisht von ‚Ubisoft' aus dem Jahr 2003. Am 18. November 2010 wurde die Trilogie, bestehend aus ‚Prince of Persia: The Sands of Time', ‚Warrior Within' und ‚The Two Thrones' in der ‚Classic HD'-Fassung für die Playstation3 erneut veröffentlicht.

Der Hauptcharakter erhält in diesem ThirdPerson Action-/Kletterspiel eine spezielle Waffe, mit der sich die Zeit kontrollieren lässt. Bei dieser Waffe handelt es sich um den ‚Dolch der Zeit'. Der namensgebende ‚Sand der Zeit' wird in diesem Dolch gesammelt und kann verbraucht werden, um die Zeit zu manipulieren. Sollte der Spieler weiteren Sand benötigen, kann dieser durch Ausschalten der im Spiel vorkommenden Gegnern gesammelt werden. Mit voranschreitender Spieldauer bekommt der Dolch der Zeit immer mehr Fähigkeiten, die sowohl Kämpfe als auch Kletterpassagen beeinflussen können.

Die erste Fähigkeit, die der Spieler erhält, ist das Zurückspulen der Zeit. Dabei wird alles - inklusive des Spielers - bis zu ca. 20 Sekunden zurückgesetzt. Wie weit die Zeit zurückgesetzt wird, kontrolliert der Spieler durch die Dauer des Drückens des dafür vorgesehenen Knopfes. Der Spieler kann dies nutzen, um misslungene Sprünge in den Kletterpassagen rückgängig zu machen. Außerdem lassen sich im Kampf Fehlentscheidungen wie das Erleiden von Schaden und der darauffolgende Charaktertod korrigieren. Wie weit die Zeit zurückgespult werden kann, ist in der UI (User Interface) durch eine Kurve gekennzeichnet, die sich bis zu einem Kreis vervollständigt.

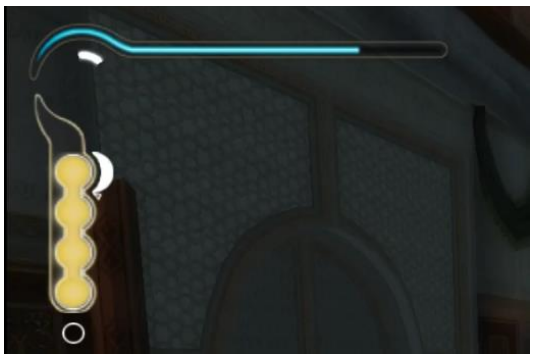


Abbildung 4 Prince of Persia: The Sands of Time UI

Eine weitere Fähigkeit, die während des Spielverlaufes öfters automatisch aktiviert wird, ist die Vision. Durch die Vision sieht der Spieler die Ereignisse, die er in der Zukunft erleben wird.

Dadurch erhält er Hinweise, wie die Kletterpassagen zu bestehen sind oder wie bestimmte Gegner überlistet werden können.

Die dritte Fähigkeit des Dolches der Zeit besteht darin, dass man die Welt verlangsamen kann. In einer Art Zeitlupe wird für die Dauer des Indikators (sprich ca. 20 Sekunden) alles verlangsamt. Dies kann genutzt werden, um besser Fallen ausweichen zu können oder im Kampf für mehr Zeit zum Überlegen zu sorgen.

Die vierte Fähigkeit des Dolches lässt Gegner in eine Art Stasis verfallen. Dadurch kann der Gegner für kurze Zeit nicht mehr handeln und ist ein einfaches Ziel.

Die fünfte und letzte Fähigkeit lässt den Spieler selbst blitzschnell werden, was einem Vorspulen der Zeit ähnelt. Während diese Fähigkeit aktiv ist, können die Gegner nicht mehr vernünftig auf die Angriffe des Spielers reagieren und fallen ihm ohne große Gegenwehr zum Opfer.

2.2 SUPERHOT

Bei SUPERHOT handelt es sich um ein Indiegame. Es wurde entwickelt und gepublisht vom ‚SUPERHOT Team' im Jahr 2016. Dieser mehrfach ausgezeichnete Singleplayer-First-Person-Shooter benutzt eine besondere Spielmechanik, denn bei SUPERHOT vergeht die Zeit effektiv nur, wenn sich der Spieler bewegt. Beim Umschauen vergeht die Zeit nur minimal und man kann sich einen Überblick verschaffen, um die ‚Rätsel', die dieses Spiel einem bietet, zu lösen. Am 25. Mai 2017 erschien ‚SUPERHOT VR', ebenfalls vom SUPERHOT Team entwickelt und gepublisht. In SUPERHOT VR wird die Mechanik von SUPERHOT in die virtuelle Realität gebracht. Der Spieler kann sich in einer Art Zeitlupe mit dem Headset umschauen, doch wenn er die Controller bewegt, schreitet gleichermaßen die Zeit voran.

2.3 Thinking with Time Machine

Thinking with Time Machine ist eine Modifikation (kurz Mod). Sie wurde von Stridemann entwickelt und vom ‚SignHead Studio' gepublished. Diese Mod basiert auf dem Spiel ‚Portal 2' von Valve. Portal 2 ist ein Rätselspiel, in welchem der Spieler eine einzigartige Waffe, die ‚Portalgun', erhält. Mit diesem Gerät ist der Spieler in der Lage, zwei Portale zu schießen, wobei das erste den Eingang und das zweite den Ausgang darstellt. Auf diese Weise können Gegenstände verschoben und Energiestrahlen umgelenkt werden und der Spielcharakter kann an sonst unerreichbare Orte gelangen. Ohne dieses Gerät wäre es dem Spieler unmöglich, die Rätsel in Portal 2 zu lösen.

Die Mod Thinking with Time Machine gibt dem Spieler ein weiteres Gerät, um weitere Rätsel zu lösen, eine Art Zeitmaschine. Mithilfe dieser Zeitmaschine ist es dem Spieler möglich, Aufnahmen von sich erstellen. Diese Aufnahmen können dann abgespielt werden, wodurch ein weiterer Spielercharakter auftaucht, welcher der Aufnahme folgt und mit physischen Objekten interagieren kann.

2.4 Mechanik BulletTime

In einigen Spielen gibt es eine Mechanik, die es dem Spieler erlaubt, die eigene Zeitwahrnehmung zu verlangsamen, um besser zielen zu können. Ein Beispiel hierfür ist die ‚Max Payne'-Reihe mit ihrem Seitwärtssprung. Bis zu dem Zeitpunkt, an dem der Spieler bei diesem Sprung den Boden erreicht, ist er in der Lage, die verlangsamten Gegner ins Visier zu nehmen. Ein weiteres Beispiel ist die ‚Fallout'-Reihe, in der mit einer ‚V.A.T.S.' genannten Mechanik die Zeit verlangsamt wird. Während V.A.T.S. aktiviert ist, können gegnerische Körperteile anvisiert und anschließend automatisch beschossen werden.

3 Umsetzung der Mechaniken in Unity

3.1 Grundmechanik

Nach einigem Herumtesten mit Timescale (Time.timescale), einer der Unity-Engine eigenen Variablen für globale Zeitkontrolle, war festzustellen, dass diese Variable sehr stark auf Kosten der Bilder pro Sekunde/Frames Per Second (kurz FPS) arbeitet. Aufgrund dieses Problems wird in KairosVR eine eigene Variante dieser Variable benutzt. Insgesamt werden zwei Zeitvariablen genutzt: eine globale Variante und eine lokale Variante. Die globale Zeitvariable ist gespeichert und auslesbar in der Klasse ‚TimeManager'. Bei dieser Klasse handelt es sich um einen Singleton, eine Art statisches Objekt, welches jederzeit aufgerufen werden kann, ohne es extra vorher zu referenzieren. Singletons haben den Vorteil, dass nur ein Objekt mit ihnen existiert. Somit bleibt es bei einem einzelnen Eintrag von ‚globalTimeScale'. Die lokale Zeitvariable ‚localTimeScale' ist in der Klasse ‚TimeBase' gespeichert, die für alle darauf zugreifenden Klassen als Bedingung gilt. In ‚TimeBase' wird eine dritte Variable aus der Globalen und der Lokalen Variable errechnet, die Variable ‚totalTimeScale'. Die Errechnung dieser Variable hängt von der Auswahl des Nutzers ab. Es kann entweder ein multiplizierendes oder addierendes Verfahren gewählt werden.

Die Klasse ‚TimeBase' ist eine bedingte Komponente für die Klasse ‚TimeAnimation', welche sich um die Umsetzung der Animationsgeschwindigkeit des Objektes kümmert. Eine weitere Klasse, die ‚TimeBase' als Bedingung hat, ist ‚TimeRigid', welche sich um die Kontrolle des

```
public static TimeManager instance;

private void Awake()
{
    if (instance == null)
    {
        instance = this;
    }
    else
    {
        Destroy(this.gameObject);
    }
}
```

Abbildung 5 Singletoncode vom TimeManager-Skript

‚Rigidbodys' des Objektes kümmert und somit dessen Force/Kraft an die Variablen anpasst. Ein ‚Rigidbody' ist eine Unity Komponente, welche an Objekte gebunden wird, die mit der Physik in Unity interagieren sollen (Bsp. Gravitation). Die dritte Klasse, die ‚TimeBase' benötigt, ist die ‚TimeNavAgent'-Klasse, welche auf Unitys künstliche Intelligenz zugreift (Unity.AI).

```
[RequireComponent(typeof(Animator))]
[RequireComponent(typeof(TimeBase))]
public class TimeAnimation : MonoBehaviour
{
    Animator anim;
    TimeBase tbase;
```

Abbildung 6 RequireComponent auf dem TimeAnimation-Skript

3.2 Prince of Persia: The Sands of Time

Prince of Persia: The Sands of Time bietet von allen analysierten Spielen die meisten Mechaniken zur Zeitmanipulation. Zunächst müssen wir die umsetzbaren Mechaniken herausfiltern. Die Vision zum Beispiel ist nicht geeignet, um sie als Asset umzusetzen, weil sie stark kontextabhängig und eher eine Videosequenz im Spiel ist. Die Fähigkeit des Anhaltens der Gegner durch eine Art Stase ist durch die Grundmechanik des ‚TimeBase'-Skriptes leicht umzusetzen. Um die Stase zu nutzen, muss nur die lokale Zeitvariable des Objektes auf 0 oder 0,1 gesetzt werden, sofern ein multiplizierendes Verfahren gewählt wurde. Mit einem addierenden Verfahren ist dies leider nicht ohne Einschränkungen möglich. Alternativ zum direkten Setzen der Variablen kann auch die Methode ‚SetLocalTemporary()' vom ‚TimeBase'-Skript aufgerufen werden. Diese Methode wird mit zwei Parametern ausgeführt: der

gewünschten neuen lokalen Zeitvariable und der Dauer dieses Effektes, bis die ursprüngliche lokale Zeitvariable wieder aktiviert wird. Die Funktion kann auch mit einem weiteren dritten Parameter ausgeführt werden, welcher die lokale Zeitvariable am Ende der Funktion neu auf den gewählten Parameter setzt.

Die Fähigkeit der Auslösung der Zeitlupe kann auch durch die Methode ‚SetLocalTemporary()' genutzt werden, doch bietet es sich eher an, die globale Zeitvariable zu nutzen, da sich ja alles in Zeitlupe bewegen soll. Dies kann durch eine ähnliche Funktion im ‚TimeManager'-Skript ausgeführt werden, dem ‚SetGlobalTemporary()'. Diese Methode funktioniert wie ihr lokales Äquivalent und braucht zum Starten zwei Parameter: die neue Zeitvariable und deren Dauer.

Um die Fähigkeit des Zeitzurückspulens umzusetzen wird der ‚struct' (Struktur) ‚PointInTime' benötigt. Dieser Datenspeicher sammelt die Daten der aktuellen Position, der Rotation sowie der momentanen ‚velocity' (Geschwindigkeit) und der ‚angularVelocity' (Winkelgeschwindigkeit). Die Klasse ‚Rewinder' wird auf das Objekt gelegt, welches zurückgespult werden soll. Ist der ‚totalTimeScale' von ‚TimeBase' größer als 0, werden die Daten in einer Liste an ‚PointsInTime' gespeichert. Um den Speicher des Nutzers nicht zu sehr zu belasten kann eine maximale Anzahl an zu speichernden Sekunden eingetragen werden, der sogenannte ‚secondsBuffer'. Sollten mehr ‚PointsInTime' eingetragen werden als der Buffer zulässt, werden die Daten gelöscht, die am frühesten eingetragen wurden. Sollte die ‚totalTimeScale' Variable des ‚TimeBase'-Skriptes nun unter 0 fallen, wird nicht mehr aufgenommen, sondern die letzten Aktionen rückwärts abgespielt. Die Daten werden nun Frame für Frame übertragen und anschließend gelöscht. Um unnötige Schritte zu überspringen, wird die ‚velocity' sowie die ‚angularVelocity' erst beim Beenden des Zurückspulens auf das Objekt angewandt.

3.3 SUPERHOT

Durch die zuvor gelegte Grundstruktur ist der Aufbau der SUPERHOT-Mechanik nicht allzu kompliziert. Um ein ähnliches Spiel mit KairosVR zu erstellen, braucht jedes Objekt (jeder Gegner/jede Waffe) ein ‚TimeBase'-Skript mit seinen Erweiterungen. Im nächsten Schritt wird ein globaler Controller hinzugefügt, der ‚InputHandler'. Der ‚InputHandler' liest die Spielereingaben aus und passt je nach Bewegung die globale Zeitvariable an. Im ‚InputHandler' kann eingestellt werden, welche Eingaben für das Skalieren berücksichtigt werden und welche nicht.

3.4 Thinking with Time Machine

Ähnlich wie die Zurückspulmechanik in Prince of Persia: The Sands of Time kann auch die Aufnahmefunktion von Thinking with Time Machine durch das ‚Rewinder'-Skript gelöst werden. In diesen Fall müsste die ‚Rewind()'-Funktion nur einmal umgeschrieben werden, sodass sie nicht die Ereignisse der ‚PointsInTime' umgekehrt sondern in der tatsächlich aufgenommenen Reihenfolge abspielt. Dies ist eine einfache Änderung in der ‚Rewind'-Schleife.

3.5 Mechanik BulletTime

Die BulletTime Mechanik wird wie die Stase-Fähigkeit aus ‚Prince of Persia: The Sands of Time' durch Aufrufen der Methode ‚SetLocalTemporary()' oder ‚SetGlobalTemporary()' ausgeführt. Durch diese Methoden kann die Zeitvariable für Spielobjekte temporär verändert werden.

4 VR Gestensteuerung

4.1 Definition

Unter Gestensteuerung versteht man im Allgemeinen die Steuerung von Elementen ohne das Betätigen einer Taste. Dies geschieht in den meisten Fällen über Kameras oder Bewegungssensoren, die dann von einer Software ausgewertet werden. Eine weitere Art der Gestensteuerung besteht darin die Position und Rotation von speziellen Controllern zu erkennen und durch die Änderung dieser Werte eine Geste zu erfassen.

4.2 Umsetzung der Gestensteuerung

In KairosVR werden ein paar Gestensteuerungen genutzt, beispielsweise die Erkennung der Controllerrotation mitsamt der Bewegungsrichtung.

Sollte die Aktivierungstaste gedrückt gehalten werden, wird die Ursprungsrotation erfasst und gespeichert. Im Beispielfall handelt es sich bei dieser Taste um die ‚Grip'-Taste des Controllers, die sich direkt bei der Griffposition des Controllers befindet und somit jederzeit ohne Probleme erreichbar ist. Sollte die aktuelle Rotation des Controllers im Vergleich zum Startpunkt mehr als 30 Grad (variabler Wert) überschritten werden, wird errechnet, in welche Richtung sich der Controller gedreht hat. Dort gibt es im Standartfall nur zwei Möglichkeiten: mit oder gegen den Uhrzeigersinn. Dieses Verfahren in VR umzusetzen ist nicht ganz einfach, da die Rotation je nach Richtung andere Grundparameter für ‚oben' besitzt. In einer Richtung liegt der Wert bei 180 Grad, in der anderen bei 360 Grad. Somit kann im einfachen Sinne nicht mühelos nach

einem ‚größer' oder ‚kleiner' als der Ursprungsrotation gefragt werden. Um ein einheitliches Ergebnis zu erzielen, muss der Wert zunächst genormt werden. Als Erstes wird der aktuelle Rotationswert der Y-Koordinate von dem ursprünglichen Rotationswert der Y-Koordinate abgezogen. Dieser Wert wird anschließend mit 360 summiert, um einen positiven Wert zu erhalten. Damit dieser Wert den Maximalwert einer Rotationskoordinate nicht überschreitet, wird er mit dem Modulo Operators im Bereich der 360 Grad gehalten. Sollte der gewählte Wert nun kleiner als 180 Grad sein, hat sich der Controller nach links gedreht. Sollte er hingegen größer als 180 Grad sein, wurde der Controller in die entgegengesetzte Richtung gedreht, also nach rechts.

```
if (Quaternion.Angle(rotationStart, actualRot) > 30)
{
    //if rotated left = -1 if right 1 (+360 % 360 normalizes the value)
    float rotateDirection = (((rotationStart.eulerAngles.y - actualRot.eulerAngles.y) + 360f) % 360f) > 180.0f ? 1 : -1;

    if (!PointerHandler.instance.selectedObject)
    {
        TimeManager.instance.globalTimeScale += Time.deltaTime * rotateDirection;
    }
    else if (PointerHandler.instance.selectedObject.GetComponent<TimeBase>())
    {
        PointerHandler.instance.selectedObject.GetComponent<TimeBase>().localTimeScale += Time.deltaTime * rotateDirection;
    }
```

Abbildung 7 GetControllerRotation-Skript Rotationserkennungslösung

Eine weitere Gestensteuerung in KairosVR ist das Erfassen der Entfernung, in welcher sich der Controller seit Beginn der Aktivierung bewegt hat. Anders ausgedrückt wird erfasst, ob der Nutzer eine gewollte Bewegung in eine Richtung gemacht hat oder ob es sich nur um ein leichtes Zittern oder ähnliche kleine Bewegungen handelt. Umgesetzt wird das Ganze durch die Erfassung der Position zu Beginn des Drückens der Aktivierungstaste. Die Aktivierungstaste ist wie bei der Funktion, die die Änderung in der Rotation des Controllers erkennt, im Standartfall auf die ‚Grip'-Taste gelegt. Nachdem die Startposition erfasst wurde, kontrolliert die Funktion, ob die Entfernung zwischen der aktuellen Position und der alten Position einen gewissen Schwellenwert überschreitet. Sollte dies der Fall sein, löst dieses Ereignis die weitere gewünschte Funktion aus.

```
//Stop Time
actualPos = (usePose) ? pose.lastLocalPosition : hand.transform.localPosition;

if(Vector3.Distance(actualPos,positionStart) > 0.3f)
{
    if (!PointerHandler.instance.selectedObject)
    {
        TimeManager.instance.globalTimeScale = 0;
        StartCoroutine(StayStopped());
    }
    else if (PointerHandler.instance.selectedObject.GetComponent<TimeBase>())
    {
        PointerHandler.instance.selectedObject.GetComponent<TimeBase>().localTimeScale = 0;
    }
```

Abbildung 8 GetControllerRotation Skript Stopp durch Distanzunterschied

4.3 Anbindung der Gestensteuerung an die Mechaniken

Für die Gestensteuerung, die die Rotation des Controllers erkennt, wird in KairosVR das Skalieren der Zeitvariablen genutzt. Dies gibt dem Nutzer die Möglichkeit, durch Controllerrotation beim Gedrückthalten der Grip-Tasten die Variablen ‚globalTimeScale' bzw. ‚lokalTimeScale' zu beeinflussen. Wird der Controller im Uhrzeigersinn gedreht, vergeht die Zeit schneller und die Variable wird somit hochgesetzt. Wird der Controller hingegen gegen den Uhrzeigersinn gedreht, wird die Zeit verlangsamt bzw. zurückgedreht, was einem Verringern der Variablen gleichzusetzen ist.

Eine weitere Funktion, die durch eine Geste ausgelöst werden kann, ist das Stoppen der Zeit bzw. das Stoppen der Zeit des ausgewählten Objektes. Diese Geste kann ausgelöst werden, indem der Nutzer den Controller nach dem Drücken der dazugehörigen Taste in eine beliebige Richtung zieht. Sollte der Controller nun genug Strecke zurückgelegt haben, wird der Effekt ausgelöst. Der Effekt des Stillstandes wird dadurch erzeugt, dass die Zeit-Variable auf 0 gesetzt wird.

5 Bau der Testszene

5.1 Version 1

Die erste VR-Testszene beinhaltete schon alle Grundmechaniken.
Die Variablen ‚globalTimeScale' und ‚localTimeScale' sind auf einer Wand dargestellt und können so direkt abgelesen werden. Zusätzlich ist das aktuell ausgewählte Objekt auf dieser

Wand vermerkt, um dem Spieler alle wichtigen Informationen zu geben.

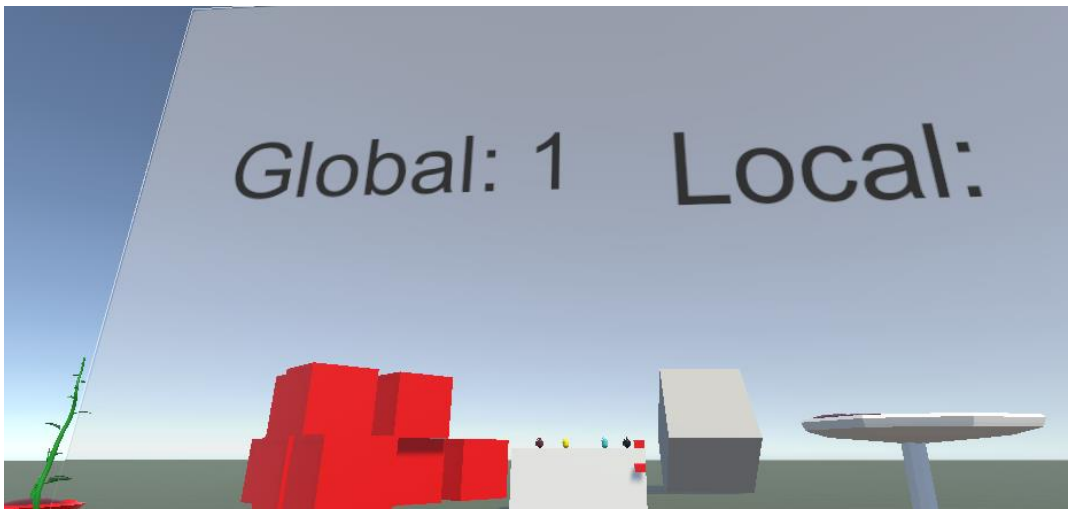


Abbildung 9 Version 1 Riesige UI im Raum

Es wurde ein kleiner Tisch vor den Spieler gesetzt, auf welchem er Granaten finden kann. Eine davon ist eine klassische Granate, die durch die Explosion physische Objekte beeinflussen kann. Eine andere Granate mit dem Namen ‚SpeedUp' kann nach dem Werfen allen Objekten in ihrem Explosionsradius einen Zeitbonus geben. In der Standartkonfiguration steigert dieser Bonus die lokale Zeitvariable um den Wert 0,5. Eine weitere Granate auf dem Tisch trägt den Namen ‚SlowDown' und löst bei der Explosion den gegenteiligen Effekt der ‚SpeedUp'-Granate auf die Objekte aus. Sie verringert die lokale Zeitvariable um 0,5. Die letzte Granate, die sich auf dem Tisch befindet, ist die sogenannte ‚Stop'-Granate. Diese Granate sorgt dafür, dass alle Objekte in ihrem Explosionsradius zum Stillstand kommen. Dies tut sie dadurch, dass sie die lokale Zeitvariable auf den Wert 0 setzt. Sollten dem Spieler die Testgranaten ausgehen, kann er die Taste am rechten Tischrand betätigen, um jede Granate ein weiteres Mal zu spawnen. Dies kann der Spieler beliebig oft wiederholen.

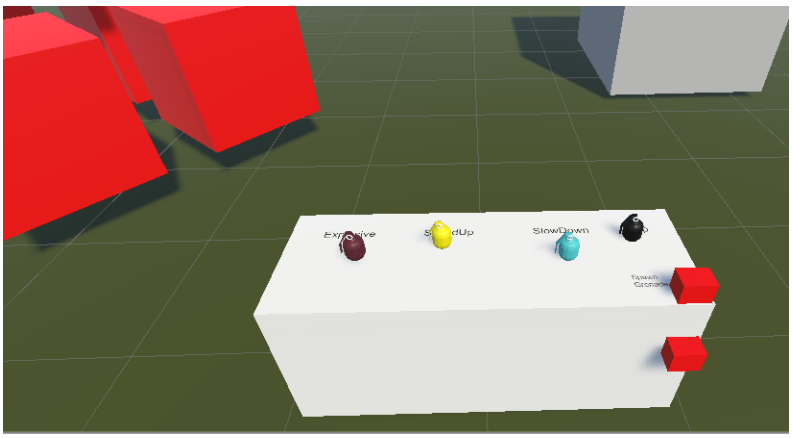


Abbildung 10 Version 1 Granatentisch

Zwischen dem Granatentisch und der Zeittafel befindet sich ein rotierender Würfel, an welchem die Granaten ausprobiert werden können. Zur Linken des Spielers befindet sich ein Haufen von Würfeln. Sollte der Spieler die Würfel durch beispielsweise eine Explosion von ihrer Ursprungsposition verschieben, können sie für ein paar Sekunden zurückgesetzt werden. Dies geschieht erneut durch das Setzen der ‚globalTimeScale'-Variable auf einen Wert unter 0.
Auch links vom Spieler, aber etwas weiter hinten platziert, befinden sich zwei Topfpflanzen. Diese wachsen, erblühen und wachsen erneut in einer Endlosschleife und können in ihrem Wachstum durch die Zeitvariablen beeinflusst werden. Zusätzlich kann durch Drücken der Touchpad-Taste des linken Controllers nach unten die Animation angehalten werden. Nun kann durch Drücken der linken beziehungsweise rechten Touchpad-Taste die aktuelle Animation durchgescrollt werden.

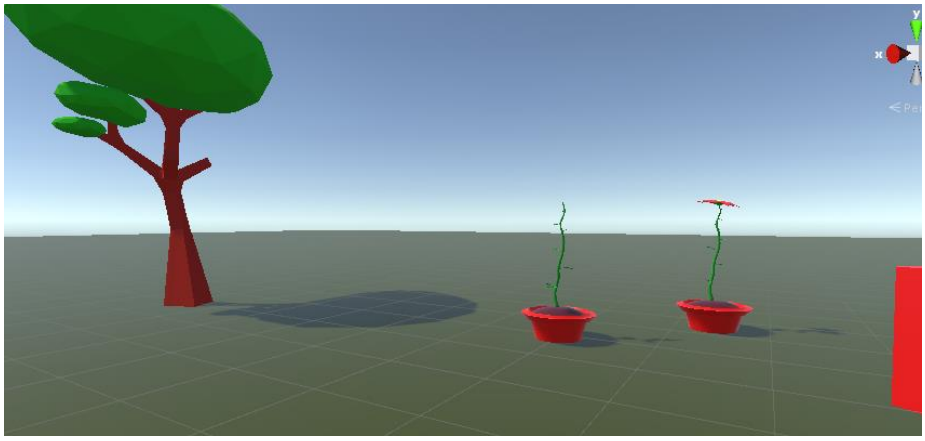


Abbildung 11 Version 1 low-poly-tree und Topfpflanzen

In der Szene hinter dem Spieler platziert befindet sich ein ‚Low-Poly'-Baum. Durch Ändern der Zeitvariablen kann dieser Baum wachsen beziehungsweise schrumpfen. Der Baum ist mit ‚BlendShapes' ausgestattet, die dadurch manipuliert werden.
Mehr im Hintergrund sieht man zur Linken des Spielers ein Männchen hin- und herlaufen. Dieses Männchen hat eine ‚NavMeshAgent-Componnent' und kann ebenfalls durch das Verändern der Zeitvariablen verlangsamt und beschleunigt werden.
Rechts vom Spieler befindet sich ein runder Tisch mit einer kleinen Landschaft. Auf dieser Landschaft läuft ebenfalls ein kleines Testmännchen hin und her. Beide Männchen erfüllen denselben Zweck, dennoch ist es fast unmöglich, das kleine Männchen zur Linken des Spielers vernünftig anzuvisieren.

5.2 Probleme von Version 1

Bei dieser Szene gab es einige Probleme. Zum Beispiel wirkte alles viel zu gestaucht. Die einzelnen Stationen waren einfach zu nahe aneinander und beeinflussten sich somit leider

gegenseitig. Ein weiteres Problem lag in der Programmierung der Physik zum Zeitpunkt des Erstellens dieser Testszene. Viele physische Spielereien waren zu diesem Zeitpunkt nicht ganz durchdacht. Beispielsweise hielten Objekte, deren effektiver Zeitwert 0 ist, nicht in der Luft an. Ein weiteres Problem war die überdimensionierte Wand, an der die Daten zu den aktuellen Zeitvariablen standen. Die Kommunikation vom Spiel zum Spieler über die möglichen Eingaben, die der Spieler tätigen kann, war ebenfalls problembehaftet. Eingaben wie zum Beispiel ‚TimeOn' - eine Steam_VR_Action, die betätigt werden muss, um das Skalieren der Zeitvariablen zu aktivieren - waren nur schwer ersichtlich.

Um dem Spieler das Ablesen der Zeitanzeige zu erleichtern, wurde der Versuch unternommen, die Anzeige am Controller zu befestigen. Dies stellte sich aber auch schon kurze Zeit später, beziehungsweise nach einer kurzen Testphase, als nicht praktikabel heraus. Denn lesbar waren die Anzeigen eigentlich nur aus einem Winkel.

Abbildung 12 Version1-2 HandUI aus unterschiedlichen Winkeln

Als Alternative zur kleinen Zeitvariablenanzeige gibt es ein etwas größeres Fenster, welches durch Gedrückthalten der ‚Trigger'-Taste des zeitkontrollierenden Controllers angezeigt werden kann. In diesem neuen Fenster werden die globale Zeitvariable, die lokale Zeitvariable sowie das ausgewählte ‚GameObject' angezeigt.

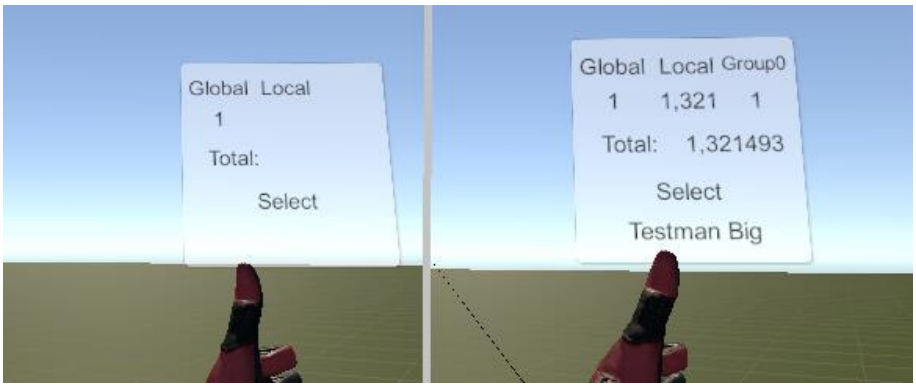


Abbildung 13 Neue HandUI durch TriggerPress | Links nichts ausgewählt / Rechts Objekt ausgewählt

Um die ‚Rigidbody'-Eigenschaften mit der Zeitkomponente zu versehen, wird das ‚TimeRigid'-Skript auf dem Objekt benötigt. Dieses Skript braucht wie eigentlich alle ‚Time'-Skripte das ‚TimeBase'-Skript, um auf die Variable ‚totalTimeScale' zugreifen zu können. Das ‚TimeRigid'-Skript beeinflusst den ‚Rigidbody' des ‚GameObjects', indem es die die ursprüngliche ‚velocity' (Geschwindigkeit) des ‚GameObjects' mit der ‚totalTimeScale'-Variable multipliziert und zurück an die ‚velocity' des ‚Rigidbodys' übergibt. Dasselbe Verfahren wird auf die ‚angularVelocity' angewandt, um auch das Drehmoment des ‚GameObjects' vernünftig zu verändern. Die Gravitation des ‚GameObjects' wird in dieser Fassung des ‚TimeRigid'-Skriptes noch in der ‚Start()'-Funktion von Unity deaktiviert und eigenständig zum ‚Rigidbody' des ‚GameObjects' hinzugerechnet. Die gesamte Berechnung findet in der Methode ‚FixedUpdate()' von Unity statt. ‚FixedUpdate()' ist für die Berechnung der physischen Elemente in Unity zuständig und sollte somit auch für genau diese Zwecke genutzt werden, was in diesem Fall auch getan wurde.

```
void FixedUpdate()
{
    if (affectGravity)
    {
        rb.AddForce(Physics.gravity * tbase.totalTimeScale);

    }
    Vector3 velocity = rb.velocity * tbase.totalTimeScale;
    Vector3 anguVelo = rb.angularVelocity * tbase.totalTimeScale;
    rb.velocity = velocity;
    rb.angularVelocity = anguVelo;
}
```

Abbildung 14 TimeRigid Version 1

Es stellte sich heraus, dass das eigene Hinzufügen der Gravitationskraft überflüssig beziehungsweise falsch war. Deshalb wurde die Berechnung der aktuellen

Gravitationsbeschleunigung aus dem Skript genommen und die normale Methode von Unity genutzt, um diese Beschleunigung nach unten zu erzeugen. Das Einfließen der Zeitvariablen ‚totalTimeScale' in die aktuelle Physikberechnung des ‚GameObjects' passierte nämlich schon durch die Veränderung der aktuellen ‚velocity' des Objektes.

```
void FixedUpdate()
{

    velocity = rb.velocity * tbase.totalTimeScale;
    anguVelo = rb.angularVelocity * tbase.totalTimeScale;
    rb.velocity = velocity;
    rb.angularVelocity = anguVelo;

}
```

Abbildung 15 TimeRigid Version1.1

Ein zusätzliches Problem der alten Version des ‚TimeRigid'-Skriptes bestand darin, dass sich die ‚velocity' zu oft mit sich selbst überschrieb. Das führte dazu, dass das ‚GameObject' bei einem ‚totalTimeScale'-Wert von unter 1 viel schneller an Tempo verlor. Dies passiert, weil es den Wert mit jedem ‚FixedUpdate()'-Aufruf neu nachskaliert. Ist nun beispielsweise der ‚totalTimeScale'-Wert auf 0,5, halbiert sich beim ersten Aufruf die ‚velocity'. Beim zweiten Aufruf wird die ‚velocity' erneut halbiert und beträgt somit nur noch ein Viertel ihrer ursprünglichen Größe. Das gleiche gilt auch für ‚totalTimeScale'-Werte über 1. In diesen Fall fängt die Physik sehr schnell an verrückt zu spielen. Durch das ständige Anpassen der Gravitationskraft werden die ‚GameObjects' mit immer stärker werdender Kraft auf die ‚GameObjects' unter ihnen gedrückt, bis diese wiederum anfangen weggeschleudert zu werden. Um dieses Problem zu beheben, musste das ‚TimeBase'-Skript angepasst werden.

```
private void Update()
{
    oldTimeScale = totalTimeScale;

    if (handling == TimeHandling.Additve)
    {
        totalTimeScale = localTimeScale + TimeManager.instance.globalTimeScale;
    }
    else if(handling == TimeHandling.Multiply)
    {
        totalTimeScale = localTimeScale * TimeManager.instance.globalTimeScale;
    }
    if(oldTimeScale != totalTimeScale)
    {
        StartCoroutine(ChangedValue());
    }
}
IEnumerator ChangedValue()
{
    changedScale = true;
    yield return null;
    changedScale = false;
}
```

Abbildung 16 TimeBase Skript Update + ChangedValue Coroutine

Durch das nun angepasste ‚TimeBase'-Skript kann von allen anderen Skripten eingelesen werden, ob es zu einer Veränderung der ‚totalTimeScale'-Variablen kam. Am Anfang der ‚Update'-Funktion wird der alte Wert von ‚totalTimeScale' in einer anderen Variablen abgespeichert. Danach errechnet das Skript den aktuellen Wert von ‚totalTimeScale', um ihn dann im letzten Schritt mit dem alten Wert zu vergleichen. Sollten diese Variablen nun nicht identisch sein, wird eine ‚Coroutine'-Funktion aufgerufen. Diese ‚Coroutine'-Funktion setzt die ‚Boolean'-Variable ‚changedScale' zunächst auf ‚true' und anschließenden einen ‚Frame' später wieder auf ‚false'. Die Variable ‚changedScale' ist von allen anderen Klassen frei einlesbar, aber nicht überschreibbar, um ihren Wahrheitsgehalt zu garantieren. Die neue Version vom ‚TimeBase'-Skript erlaubt es nun, das ‚TimeRigid'-Skript ebenfalls zu verbessern. In der verbesserten Version vom ‚TimeRidig'-Skript wird in der ‚FixedUpdate'-Funktion zunächst nach der Variablen ‚changedScale' gefragt, um zu ermitteln, ob eine Anpassung der ‚velocity' und ‚angularVelocity' nötig ist. Um nun den Effekt zu behalten, der bei einem Zeitstillstand auftreten soll, wird abgefragt, ob die Zeitvariable ‚totalTimeScale' bei 0 liegt. Ist dies der Fall, wird die ‚velocity' sowie die ‚angularVelocity' des ‚GameObjects' auf 0 gesetzt.

```
void FixedUpdate()
{
    if (tbase.totalTimeScale == 0)
    {
        rb.AddForce(-Physics.gravity);
        rb.velocity = Vector3.zero;
        rb.angularVelocity = Vector3.zero;
    }

    if (tbase.changedScale)
    {
        velocity = rb.velocity * tbase.totalTimeScale;
        anguVelo = rb.angularVelocity * tbase.totalTimeScale;
        rb.velocity = velocity;
        rb.angularVelocity = anguVelo;
    }
```

Abbildung 17 TimeRigid FixedUpdate Version2 ChangedScale

Da die Gravitation bei den ‚GameObjetcs' nun leider dennoch einen kleinen Einfluss hat, obwohl die ‚velocity' eigentlich bei 0 liegen müsste, muss eine Gegenkraft auf das ‚GameObject' ausgeübt werden. Diese Kraft ist schlicht gesagt das Gegenteil der Gravitationskraft.

```
if (tbase.totalTimeScale == 0)
{
    rb.AddForce(-Physics.gravity * rb.mass);
    rb.velocity = Vector3.zero;
    rb.angularVelocity = Vector3.zero;
}
```

Abbildung 18 TimeRigid AntiGravity bei Zeitskalierung 0

In dieser Version des Skriptes stellte sich nun aber ein weiteres Problem dem eigentlichen Ziel entgegen. Nach einigem Testen mit verschiedenen Startwerten der Zeitskalierung stellte sich heraus, dass die Gravitation von Unity leider doch unabhängig von der Zeitvariable agierte, da ja keine direkte Änderung vorlag. Allgemein stellte sich heraus, dass neu hinzugefügte Kräfte nicht vernünftig mit dem ‚TimeRigid'-Skript interagierten. Um dieses Problem zu lösen, entstand die statische Klasse ‚RigidTools'. Dadurch, dass es sich bei ‚RigidTools' um eine statische Klasse mit statischen Methoden handelt, kann ‚RigidTools' von überall ohne Referenz aufgerufen werden. Dieses Skript dient der korrekten Anwendung von ‚AddForce()', der ‚Rigidbody'-Komponente von Unity. Statt wie gewohnt die Methode vom ‚Rigidbody' zu verwenden, werden nun alle ‚AddForce()'-Aufrufe über das ‚RigidTools'-Skript mit der passenden Zeitvariable angepasst und dann an den ‚Rigidbody' weitergeleitet. Sollte es sich bei

dem zu bearbeitenden Objekt nicht um eines mit einer Zeitvariable handeln, führt ‚RigidTools' den normalen ‚AddForce()' Befehl aus.

```
public static class RigidTools
{
    public static void AddForce(Rigidbody rb, Vector3 force)
    {
        if(rb.GetComponent<TimeBase>())
            rb.AddForce( force * rb.GetComponent<TimeBase>().totalTimeScale);
        else
            rb.AddForce( force * rb.GetComponent<TimeBase>().totalTimeScale);
    }
    public static void AddExplosiveForce(Rigidbody rb, float force, Vector3 exploPosition, float exploRadius)
```

Abbildung 19 RigidTools Auschnitt AddForce()

Mit dem ‚RigidTools'-Skript müssen nun die anderen Skripte überarbeitet werden. Die normale Granate gibt nun die Kraft der Explosion über ‚RigidTools' an die ‚GameObjects' in der Nähe weiter. So auch das Explosionsskript, welches über den Controller ausgeführt werden kann, um die großen roten Würfel in Bewegung zu bringen. Das Wichtigste, was an die neuen Gegebenheiten angepasst werden muss, ist jedoch das ‚TimeRigid'-Skript. In der ‚Start()'-Funktion des Skriptes wird nun geschaut, ob der ‚Rigidbody' des ‚GameObjects' Gravitation benutzt oder nicht. Sollte dies der Fall sein, wird dies skriptintern vermerkt und die Gravitation, die Unity standarttechnisch mitliefert, ausgeschaltet. In der ‚FixedUpdate()'-Funktion wird nun geschaut, ob Gravitation genutzt werden soll oder nicht. Sollte dies der Fall sein und die ‚totalTimeScale'-Variable größer als 0 sein, wird die Gravitationskraft mit der Masse des ‚GameObjects' an ‚RigidTools' übergeben und anschließend an das Objekt weitergegeben. Da die Gravitation nun nur noch durch das Skript selbst auf das ‚GameObject' weitergegeben wird, können die gravitationsrelevanten Codezeilen bei einem ‚totalTimeScale'-Wert von 0 nun gelöscht werden.

```
        if (rb.useGravity)
        {
            gravity = true;
            rb.useGravity = false;
        }
    }


    void FixedUpdate()
    {
        if (rewindable)...

        if (gravity && tbase.totalTimeScale > 0)
        {
            RigidTools.AddForce(rb, Physics.gravity * rb.mass);
        }
```

Abbildung 20 TimeRigid Manuelle Gravitationskraft

Um für eine noch individuellere Erfahrung für die Zeitsteuerung in KairosVR zu sorgen, wurde dem Asset folgendes Feature hinzugefügt: Zeitgruppen. Zeitgruppen können wie eine zweite globale Zeitvariable gesehen werden, dennoch sind sie es nicht ganz. Zeitgruppen sind wie die globale Zeitvariable, die jedoch lokal agiert. Die Zeitvariable der Zeitgruppen ist gespeichert und auslesbar im ‚TimeGroupHandler'-Skript. Bei diesem Skript handelt es sich erneut um einen ‚Singleton', um ein für eine permanente Verfügbarkeit der Daten zu sorgen. Die Gruppen sind als ein ‚enum' unterhalb des ‚TimeBase'-Skriptes unter dem Namen ‚TimeGroup' gespeichert und können dort nach Belieben umbenannt werden. Bei Bedarf ist das Erweitern dieses ‚enums' kein Problem, da sich die Anzahl der einlesbaren Zeitvariablen sofort aktualisiert.

```
public enum TimeGroup
{
    Group0,
    Group1,
    Group2,
    Group3
}
```

Abbildung 21 Enum TimeGroup

Das ‚TimeGroupHandler'-Skript selbst dient nur als Auslese- beziehungsweise als Aktualisierpunkt der Gruppenzeitvariablen. Als Zusatzfunktion bietet diese Klasse für den ‚Inspector' noch eine Erweiterung des ‚ContextMenus', welche alle Gruppenzeitvariablen zurück auf 1 setzen kann.

```
public class TimeGroupHandler : MonoBehaviour
{
    public static TimeGroupHandler instance { get; set; }

    public float[] groupScale;

    private void Start()...

    [ContextMenu("ResetClocks")]
    public void Reset()
    {
        for (int i = 0; i < groupScale.Length; i++)
        {
            groupScale[i] = 1f;
        }

    }
}
```

Abbildung 22 TimeGroupHandler BasicSkript

Um für eine gute Übersicht über die einzelnen Gruppen und ihren aktuellen Wert zu sorgen, gibt es ein ‚Editor'-Skript. Das ‚Editor'-Skript ‚TimeGroupHandlerEditor' erbt wie alle ‚Editor'-Skripte von ‚Editor' und überschreibt die Methode ‚OnInspectorGUI()'. Zunächst referenziert das Skript auf das ‚target'-Skript und sucht sich die relevante Datenstruktur durch die ‚FindProperty()'-Methode. In diesem Fall handelt es sich bei der gewünschten Variablen um ein ‚float-Array' mit dem Namen ‚groupScale'. In diesem ‚Array' werden die Zeitvariablen der Zeitgruppen gespeichert. Um repetitiven ‚Code' zu verhindern, wird danach eine ‚for'-Schleife ausgeführt, die über jedes mögliche Element des ‚TimeGroup enums' iteriert. Zunächst wird ein Label mit dem aktuellen ‚TimeGroup'-Namen erstellt und mit einen ‚FloatField' ausgestattet. Dieses ‚FloatField' zeigt den aktuellen Wert von ‚groupScale' an und kann wie gewohnt verändert werden. Anschließend wird der eingetragene Wert wieder auf die referenzierende Variable übertragen. Das Ganze wird von einer horizontalen Gruppe umspannt, um für mehr Lesbarkeit im Editor zu sorgen. Tatsächlich wird durch diese Methode die Größe des ‚float Arrays' festgelegt.

```
using System;
using UnityEngine;
using UnityEditor;

namespace KairosVR
{
    [CustomEditor(typeof(TimeGroupHandler))]
    public class TimeGroupHandlerEditor : Editor
    {
        //Layout
        float labelWidth = 150f;

        public override void OnInspectorGUI()
        {
            //base.OnInspectorGUI();

            var serializedObject = new SerializedObject(target);
            var myArrayProperty = serializedObject.FindProperty("groupScale");


            for (int i = 0; i < Enum.GetValues(typeof(TimeGroup)).Length; i++)
            {
                GUILayout.BeginHorizontal();
                GUILayout.Label(Enum.GetName(typeof(TimeGroup), i), GUILayout.Width(labelWidth));
                var myElement = myArrayProperty.GetArrayElementAtIndex(i);
                myElement.floatValue  = EditorGUILayout.FloatField(myElement.floatValue);
                serializedObject.ApplyModifiedProperties();
                GUILayout.EndHorizontal();

            }

        }
    }
}
```

Abbildung 23 TimeGroupHandlerEditor Skript

Um dem Nutzer von KairosVR noch eine weitere individuelle Zeitsteuerung zu ermöglichen, wurden die Zeitgruppen im nächsten Schritt um eine Kleinigkeit erweitert. Eine ‚Checkbox' ist rechts neben dem Eingabefeld der Zeitvariable der jeweiligen Zeitgruppe platziert worden. Der aktuelle Zustand der ‚Checkbox' bestimmt ob die Zeitgruppe durch die Zeitvariable ‚globalTimeScale' beeinflusst wird oder nicht. Der Wert dieses ‚booleans' wird ebenfalls im ‚TimeGroupHandler'-Skript gespeichert und von ‚TimeBase' abgefragt.

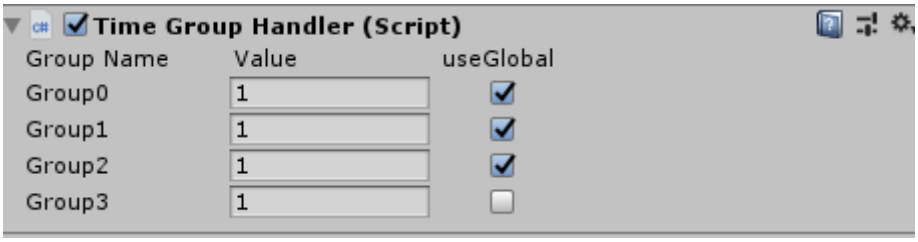


Abbildung 24 TimeGroupHandler Inspektor mit Checkbox

Das Abkoppeln von der globalen Zeitvariable kann sinnvoll sein, wenn der Nutzer beispielsweise einzelne Räume in sein Projekt einbaut, die individuell gelöst, sprich nicht durch die zuvor getätigte Zeitmanipulation beeinflusst werden sollen. Die neue Berechnung der ‚totalTimeScale'-Variable im ‚TimeBase'-Skript fragt nun innerhalb der Funktion nach der Variablen ‚useGlobal' vom ‚TimeGroupHandler'-Skript. Sollte ‚useGlobal' ‚true' sein, wird wie zuvor ‚localTimeScale' mit dem ‚groupScale' und dem ‚globalTimeScale' verrechnet. Sollte sie auf ‚false' stehen, wird anstatt mit ‚globalTimeScale' mit dem neutralen Element der Berechnungsart gerechnet. Dies ist im Fall des additiven Verfahrens ‚0' und im multiplikativen Verfahren ‚1'.

```
else if(handling == TimeHandling.Multiply)
{
    totalTimeScale = localTimeScale * TimeGroupHandler.instance.groupScale[(int)group] *
        (TimeGroupHandler.instance.useGlobal[(int)group] ? TimeManager.instance.globalTimeScale : 1);
}
```

Abbildung 25 TimeBase Berechnung anhand von TimeGroups

Das Verhalten der Zeitgruppen wird in der zweiten Version der Testszene beispielsweise beim ‚Gate' benutzt.

Um für ein weiteres Element der neuen Testszene zu sorgen, wurde eine ‚Handfeuerwaffe' auf dem ‚Granaten'-Tisch platziert.

Abbildung 26 VR Handfeuerwaffe

Die abgefeuerten ‚Bullets' haben dieselbe Zeitvariable und ‚TimeGroup' wie die ‚Handfeuerwaffe', die sie abfeuert. Diese wird bei dem Abfeuern beziehungsweise dem ‚Spawnen' des ‚Bullets' gesetzt. Der Punkt der Instanziierung ist dabei etwas vor der ‚Handfeuerwaffe' positioniert, um ein Kollidieren mit dieser zu vermeiden. Wie alle ‚Interactable-GameObjects' deaktiviert auch die ‚Handfeuerwaffe' das ‚LaserPointer'-Skript. Dies geschieht, um zu verhindern, dass es zu Verwirrung bei der Benutzung der ‚Interactable-Gameobjects' kommt. Des Weiteren sollte eine Taste nicht doppelt belegt nutzbar sein. Um für eine gute graphische Darstellung des Haltens dieser ‚Handfeuerwaffe' zu sorgen, wird das ‚SkeletonPoser'-System von ‚SteamVR' genutzt. Gleichzeitig sorgt das Nutzen des ‚SkeletonPoser'-Systems für eine korrekte Haltung der ‚Handfeuerwaffe'.

```
void Update()
{
    if (interactable.attachedToHand)
    {
        if (trigger.stateDown && canShoot)
        {
            Shoot();
        }
        if (interactable.attachedToHand.GetComponent<SteamVR_LaserPointer>())
        {
            lastPointer = interactable.attachedToHand.GetComponent<SteamVR_LaserPointer>();
            lastPointer.pointer.gameObject.SetActive(false);
            lastPointer.enabled = false;
        }
        else
        {
            lastPointer = null;
        }
    }
    else
    {
        if (lastPointer)
        {
            lastPointer.enabled = true;
            lastPointer.pointer.gameObject.SetActive(true);
        }
```

Abbildung 27 VRGun Skript V1 Update

Beim Erstellen der ‚Handfeuerwaffe' und der ‚Bullets' fiel ein großer Fehler in der Physikberechnung des ‚TimeRigid'-Skriptes beziehungsweise des ‚RigidTools'-Skriptes auf. Sollte die Zeit einmal auf 0 gesetzt werden, werden sämtliche Objekte sich danach nicht an ihre vorherige ‚velocity' erinnern, da diese in ‚FixedUpdate()' ständig überschrieben wird. Um dies zu verhindern, wurde das ‚TimeRigid'-Skript mit ein paar weiteren Variablen ausgestattet, den Variablen ‚realVelocity' und ‚realAnguVelo'. Diese Variablen speichern, wie ihr Name schon sagt, die reale ‚velocity' bzw. ‚angularVelocity' des ‚rigidbodies' des jeweiligen ‚GameObjects'. In der ‚FixedUpdate()'-Funktion des ‚TimeRigid'-Skriptes wird nun die ‚velocity' anhand der ‚realVelocity' multipliziert mit dem ‚totalTimeScale' ermittelt, statt wie zuvor, wo die ‚velocity' direkt mit dem ‚totalTimeScale'-Wert multipliziert wurde. Dasselbe gilt auch für die ‚angularVelocity'.

Doch ging dieses Prinzip leider nicht so einfach in Unity auf. Nach einigen Versuchen die echte ‚velocity' zu berechnen entstand eine Änderung in der statischen Klasse ‚RigidTools'. Nach einiger Recherche wurde klar, wie sich anhand der ‚AddForce()'-Methode die ‚velocity' errechnet. Je nach ‚ForceMode' gibt es eine unterschiedliche Berechnung. Im Falle von ‚ForceMode.Force' wird die neu hinzuzufügende ‚force' im Zusammenspiel mit ‚fixedDeltaTime' durch die Masse des Objektes geteilt und auf die ‚velocity' addiert. ‚ForceMode.Acceleration' hat eine ähnliche Berechnung, nur wird die Masse hierfür nicht benötigt. Bei ‚ForceMode.Impulse' gibt es keinen Zeitwert, der berücksichtigt wird, somit errechnet sich diese Änderung der ‚velocity' durch Teilen der ‚force' durch die Masse und wird anschließend zur ‚velocity' hinzuaddiert. Im letzten Fall ‚ForceMode.VelocityChange' wird ganz simpel die ‚force' mit der ‚velocity' addiert.

```
if (rb.GetComponent<TimeRigid>())
{
    TimeRigid tr = rb.GetComponent<TimeRigid>();


    switch (forceMode)
    {
        case ForceMode.Force:
            tr.forcesToAdd.Add(force * Time.fixedDeltaTime / rb.mass);
            break;
        case ForceMode.Acceleration:
            tr.forcesToAdd.Add(force * Time.fixedDeltaTime);
            break;
        case ForceMode.Impulse:
            tr.forcesToAdd.Add(force / rb.mass);
            break;
        case ForceMode.VelocityChange:
            tr.forcesToAdd.Add(force);
            break;
    }
```

Abbildung 28 RigidTools ForceMode berechnung

Ein weiterer Versuch das Problem mit der Physik zu lösen, bestand darin, eine Liste zu erstellen, die alle einwirkenden Kräfte in Form von ‚Vector3' aufnimmt und bei einem ‚totalTimeScale' von größer als 0 dem ‚GameObject' wieder hinzufügt. Dies stellte sich aber auch als Fehler heraus, da die Berechnung der Anfangswerte nicht korrekt von statten ging. Das ständige Hinzufügen der Gravitationsbeschleunigung wurde hiermit leider auch nicht gelöst.

Um die echte ‚velocity' zu berechnen wird mehr benötigt als die ‚AddForce()' Methoden des ‚Rigidbodies' anzupassen. Um eine wirklich korrekte Darstellung zu bekommen, müssten im nächsten Schritt sämtliche ‚Collisions' der ‚Collider' angepasst werden. Es müsste somit eine komplett eigene ‚Physics-Engine' geschrieben werden. Dies würde zum einen den Rahmen dieser Bachelorarbeit sprengen und zum anderen die Nutzung des Assets sehr erschweren. Was bringt einem ein Asset, das nicht mehr vernünftig mit dem Rest von Unity agiert. Um aber dennoch den Effekt der ‚BulletTime' oder ein ähnliches Verhalten der ‚Bullets' wie in ‚SUPERHOT' zu erreichen, wurde dem ‚TimeRigid'-Skript eine alternative Berechnung hinzugefügt. Sollte der ‚public Boolean experimantel' auf ‚true' gesetzt werden, wird die ursprüngliche Berechnung ignoriert und das alternative Verfahren benutzt. Dieses Verfahren geht davon aus, dass die ‚trueVelocity' von außerhalb gesetzt wird und passt die ‚velocity' des ‚GameObjects' entsprechend an.

```
void FixedUpdate()
{
    if (experimental)
    {
        if(tbase.totalTimeScale >= 0)
        {
            rb.velocity = trueVelo * tbase.totalTimeScale;
            rb.angularVelocity = trueAnguVelo * tbase.totalTimeScale;
        }
        else
        {
            rb.velocity = Vector3.zero;
            rb.angularVelocity = Vector3.zero;
        }
    }
```

Abbildung 29 TimeRigid experimental Physics

5.3 Version 2

In der zweiten Version der Testszene wurde eine Wand im Sichtfeld des Nutzers platziert. Auf dieser Wand sieht man zum einen die Gestensteuerung visuell erklärt. Sollte die entsprechende Taste zur Zeitkontrolle gedrückt gehalten werden, zeigt sich an der Hand der ‚magische' Kreis, der auch an der Wand zu sehen ist und die gezeigten Gesten können angewandt werden. Unter der Gestenerklärung befindet sich eine Orientierungshilfe was den momentanen Zeitzustand der ‚GameObjects' angeht. Sollte ein ‚GameObject' einen ‚totalTimeScale'-Wert von mehr als 1 haben, wird das ‚GameObject' gelb gefärbt dargestellt. Sollte der Wert kleiner als 1 aber größer als 0 sein, wird es cyanfarben dargestellt. Bei einem ‚totalTimeScale'-Wert von genau 0 wird das entsprechende ‚Material' in schwarz gefärbt. Wenn das ‚GameObject' einen Wert von niedriger als 0, also einen Wert im negativen Bereich hat, wird es grau gefärbt. In diesem Zustand wird dann wie zuvor auch der ‚Rewind'-Effekt aufgerufen.

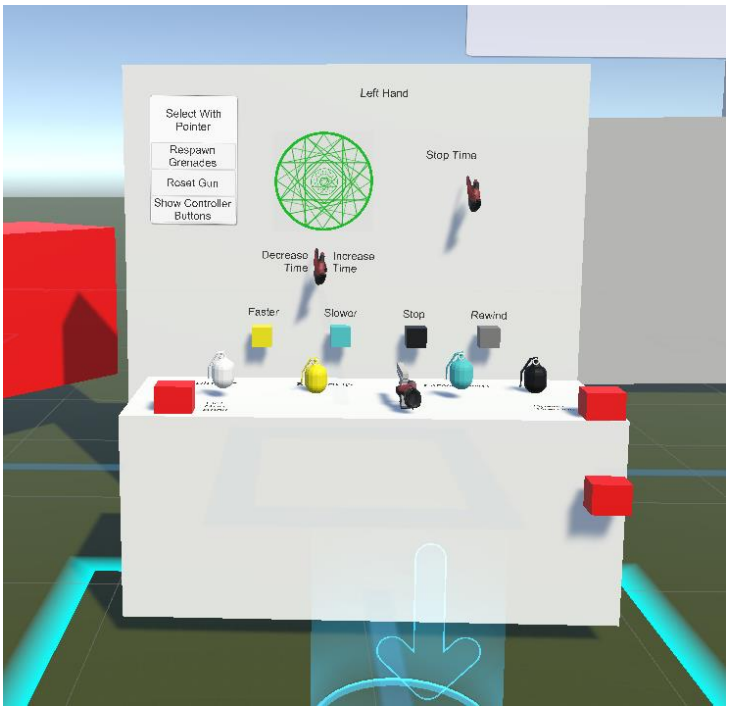


Abbildung 30 Version 2 Startpunkt

Ebenfalls an der Wand befindet sich ein ‚Panel' mit ‚Buttons' von ‚UnityUI', die mit dem ‚LaserPointer' ausgewählt werden können. Die ‚Buttons' erfüllen ähnliche Funktionen wie die drückbaren Tasten auf dem Tisch vor der Wand. So ‚spawned' ein ‚Button' die ‚Granaten' neu und eine andere Taste setzt die ‚Handfeuerwaffe' wieder auf ihre ursprüngliche Position. Tests haben gezeigt, dass die ‚Handfeuerwaffe' leider oft außerhalb der Nutzerreichweite fallen konnte. Zu Beginn der Testszene werden für einen kurzen Zeitraum mithilfe der ‚SteamVR'-Funktion ‚ControllerShowButtonsHint' die relevanten Tasten gezeigt. Da nicht alle Nutzer direkt auf ihre Hände schauen, wenn sie die Szene starten, ruft der unterste ‚Button' diese Darstellung erneut auf. Links vom Starttisch befinden sich die ‚rewindable'-Würfel. Da sie diesmal mit dem ‚TimeScaleColorChanger'-Skript ausgestattet sind, ist gut zu sehen, in welchem Zustand der Zeitvariablen sich die individuellen Würfel befinden.

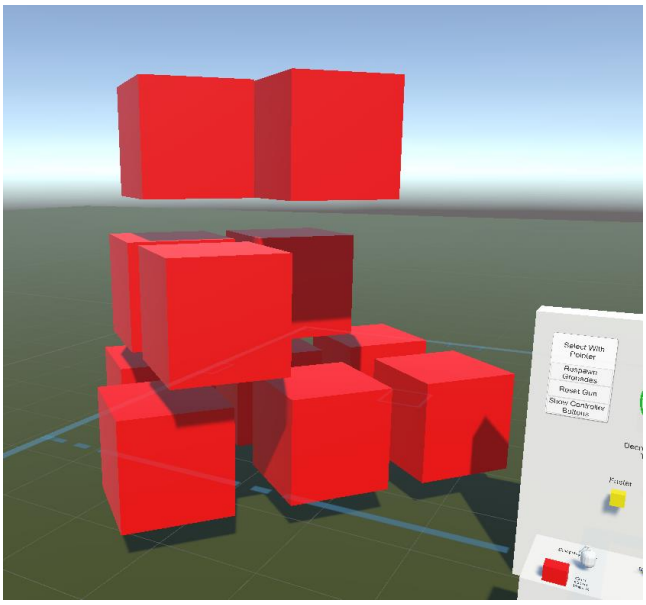


Abbildung 31 Version 2 Explosive Cubes

Rechts vom Starttisch befindet sich der rotierende Würfel. Über dem Würfel wurde ein kleines ‚UI'-Element gesetzt, welches den aktuellen ‚totalTimeScale' des Würfels anzeigt, um das anfängliche Herumprobieren mit der Variablen zu erleichtern. Hinter dem Nutzer befindet sich das wie zuvor erwähnte ‚Gate', welches aus Würfeln besteht, die in einer Zeitgruppe sind, in der sie unabhängig von der ‚globalTimeScale'-Variable agieren. Zwischen dem ‚Gate' und den ‚rewindable'-Würfeln befinden sich der ‚low-poly-tree' und die Topfpflanzen, welche die Manipulation von ‚Blendshapes' und die Animationsgeschwindigkeit zeigen sollen. Zwischen den Pflanzen und den ‚rewindable'-Würfeln befindet sich etwas weiter entfernt das ‚Testman-GameObject'. Dieses ‚GameObject' trägt das ‚NavMeshAgent'-Skript von Unity sowie das ‚TimeNavAgent'-Skript von ‚KairosVR'. An diesem ‚GameObject' soll die Manipulierung der Variablen vom ‚NavMeshAgent' gezeigt werden.

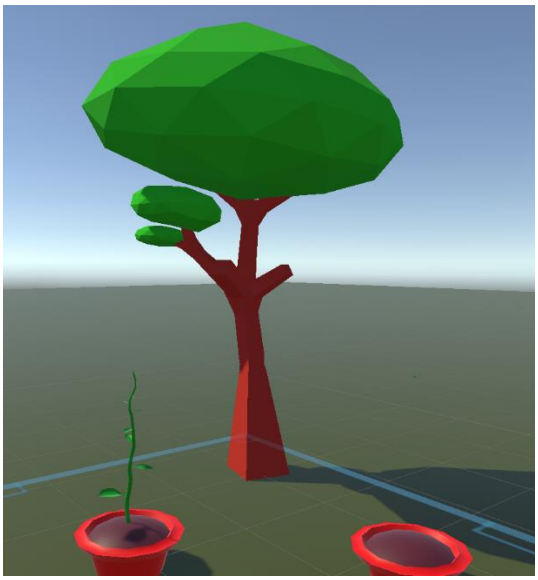


Abbildung 32 Version 2 TimeAnimation und TimeBlendshapes

Die größte Neuerung in dieser Version ist der ‚Schießstand'. Der ‚Schießstand' befindet sich rechts vom Startpunkt in einen kleinen eigenen Bereich der Testszene. Dort findet der Nutzer eine Wand mit einer Zielscheibe sowie eine ‚Handfeuerwaffe'. Sollte der Nutzer nun mit der Waffe auf das Ziel schießen, erhält er je nach Trefferzone unterschiedliche Punkte. Sollte der Nutzer zehn ‚Bullets' verschießen, startet die zweite Runde mit einer sich leicht bewegenden Zielscheibe. Nach zehn weiteren Schüssen beginnt die dritte und letzte Runde, in der sich die Zielscheibe nicht nur in zwei, sondern in allen drei Dimensionen bewegen kann. Diese Zielscheiben sind natürlich auch durch Zeitmanipulation beeinflussbar. Der aktuelle Punktestand sowie die durchschnittlichen Punkte pro ‚Bullet' werden dem Nutzer an der Zielscheibenwand jederzeit angezeigt. Sollte am Ende der dritten Runde eine neue Höchstpunktzahl erreicht werden, wird diese an der Wand rechts vom Nutzer eingetragen. Wenn die ‚Handfeuerwaffe' einmal verloren geht oder wenn der Nutzer eine weitere Runde spielen möchte, kann die ‚Reset'-Taste rechts vom Nutzer gedrückt werden. Sobald diese Taste gedrückt wird, wird die ‚Handfeuerwaffe' wieder auf die Startposition und die Zielscheiben wieder auf Runde eins zurückgesetzt. Die innere Logik des ‚Schießstandes' wird vom ‚ShootingRange'-Skript sowie dem ‚ShootingTarget'- beziehungsweise ‚ShootingTargetMove'-Skript gesteuert.

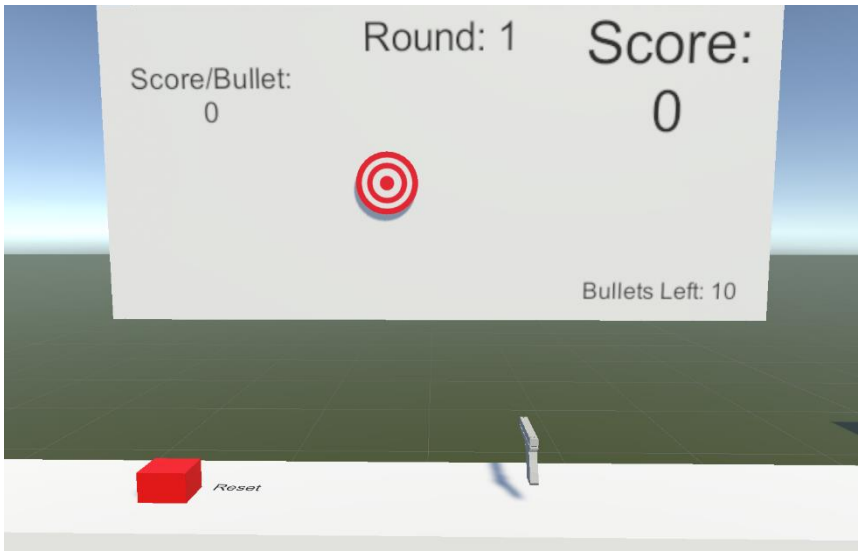


Abbildung 33 Version 2 Schießstand

5.4 Probleme von Version 2

Das Hauptproblem der zweiten Version der Testszene war immer noch, dass alle Elemente zu nahe zueinander positioniert sind. Somit ist der Nutzer schnell überfordert, was die vielen Möglichkeiten dieser Anwendung angeht. Um dies zu beheben, werden in der dritten Version der Testszene die einzelnen Elemente thematisch in Räume eingeteilt, ähnlich wie schon der Schießstand in der zweiten Version.

Ein weiteres Problem dieser Version bestand darin, dass sie oft manuell neugestartet werden musste, um bestimmte Funktionen vernünftig als Nutzer testen zu können. Der erste Ansatz zur Behebung dieses Problems war schon das Einführen der Tasten auf dem Tisch und im ‚User Interface' an der Wand. Diese Tasten müssen nun nur noch an das Zurücksetzen der relevanten Elemente gekoppelt werden.

Ein weiteres Problem bestand in dem Zeigen der relevanten Tasten für die Steuerung in dieser Version der Testszene. Zwei Sekunden reichen den meisten Menschen leider nicht aus, um sie beim Starten der Testszene richtig wahrzunehmen. Gegebenenfalls hilft eine Zone, in der die Hinweise dauerhaft angezeigt werden, solange sich der Nutzer in dieser befindet.

Eine weitere Möglichkeit zum Verbessern der Orientierung ist das Hinzufügen des ‚TimeScaleColorChanger'-Skriptes auf mehr ‚GameObjects' wie beispielsweise den Blumentopf der wachsenden Pflanze. Um dies möglichst einfach umzusetzen, wird das ‚TimeScaleColorChanger'-Skript auf das ‚GameObject' mit dem umzufärbenden Material gelegt. Das Skript ermittelt dann in der ‚Start()'-Funktion, um welchen ‚Renderer' es sich bei dem ‚GameObject' handelt und sucht sich dementsprechend das ‚Material' heraus. Das Skript

sucht sich nun außerdem das ‚TimeBase'-Skript aus einem möglichen ‚Parent-GameObject' heraus, um die Farbänderung akkurat durchzuführen.

```
public class TimeScaleColorChanger : MonoBehaviour
{
    Material material;
    TimeBase tb;

    void Awake()
    {
        if (GetComponent<MeshRenderer>())
            material = GetComponent<MeshRenderer>().materials[0];
        else
            material = GetComponent<SkinnedMeshRenderer>().materials[0];
        if (GetComponent<TimeBase>())
            tb = GetComponent<TimeBase>();
        else
            tb = GetComponentInParent<TimeBase>();
    }
```

```
    void Update()
    {
        if(tb.totalTimeScale > 1)
        {
            material.color = Color.yellow;
        }
        else if(tb.totalTimeScale == 1)
        {
            material.color = Color.green;
        }
        else if(tb.totalTimeScale > 0)
        {
            material.color = Color.cyan;
        }
        else if(tb.totalTimeScale == 0)
        {
            material.color = Color.black;
        }
        else
        {
            material.color = Color.grey;
        }
    }
}
```

Abbildung 34 TimeChangeColorChanger - links Awake() - rechts Update()

5.5 Version 3

Die dritte Version der Testszene ist in drei Zonen unterteilt:

Die Startzone ist der Ort, an dem der Nutzer startet. Hier befindet sich, ähnlich wie in der zweiten Version, vor dem Nutzer eine Wand mit Informationen zur Gestensteuerung. Die Leiste oberhalb dieser Informationen zeigt dem Nutzer den Farbcode, den die meisten ‚GameObjects' in dieser Szene benutzen. Im Vergleich zur alten Version gibt es nun auch eine grüne Farbänderung im ‚TimeScaleColorChanger'-Skript. Grüne ‚GameObjects' sind bisher unbeeinflusst von der Zeitmanipulation und haben somit einen ‚totalTimeScale'-Wert von genau 1.

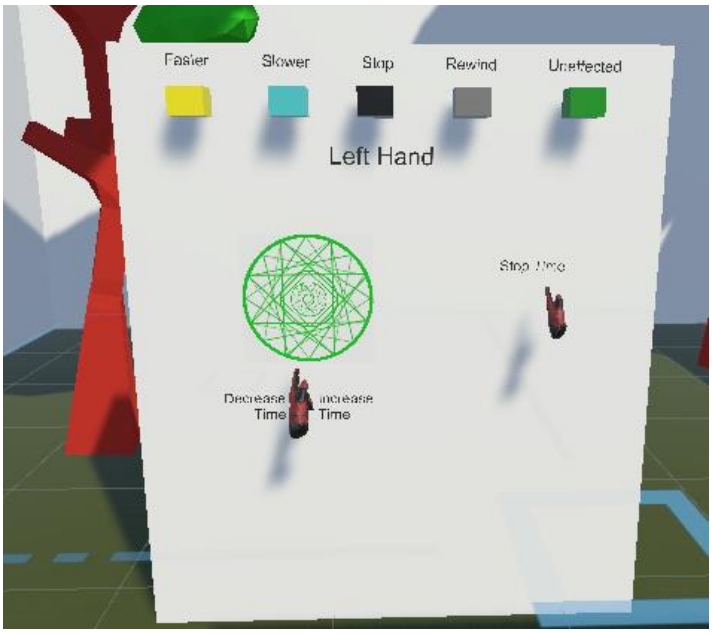


Abbildung 35 Version 3 Startwand mit Tutorial

Hinter dieser Startwand sind ein paar ‚low-poly-trees' platziert um die Option des ‚ShapeKey/Blendshape'-Manipulierens zu demonstrieren. Rechts vom Nutzer befindet sich der rotierende Würfel. Dieser Würfel ist mit dem ‚TimeScaleColorChange'-Skript ausgestattet und hat über sich ein ‚UI'-Element, welches seinen aktuellen ‚totalTimeScale'-Wert anzeigt. Wie schon in der zweiten Version der Testszene soll einem dieser Würfel für die anfängliche Orientierung in der Szene helfen. In diesen ersten Raum verteilt befinden sich noch zwei wachsende Pflanzen, welche nun einen Topf haben, der ihren momentanen Zeitzustand anhand des Farbcodes anzeigt.

Hinter dem Nutzer und somit gegenüber der Erklärungswand befindet sich der Eingang zur mittleren Zone. Auf dem Weg zur mittleren Zone passiert der Nutzer zwei ‚Gates'. Diese ‚Gates' bestehen aus Würfeln, deren lokaler Zeitwert beim Start der Szene auf 0 gesetzt wird. Sie können mithilfe der Controller geschoben und ihre Zeitwerte einzeln wie gewohnt manipuliert werden. Beim Verlassen der ersten Zone werden die Controllerhinweise deaktiviert, die in dieser Zone dauerhaft aktiv bleiben. Sollte der Nutzer diese Hinweise erneut angezeigt bekommen wollen, kann er jederzeit wieder hierhin zurückkehren.

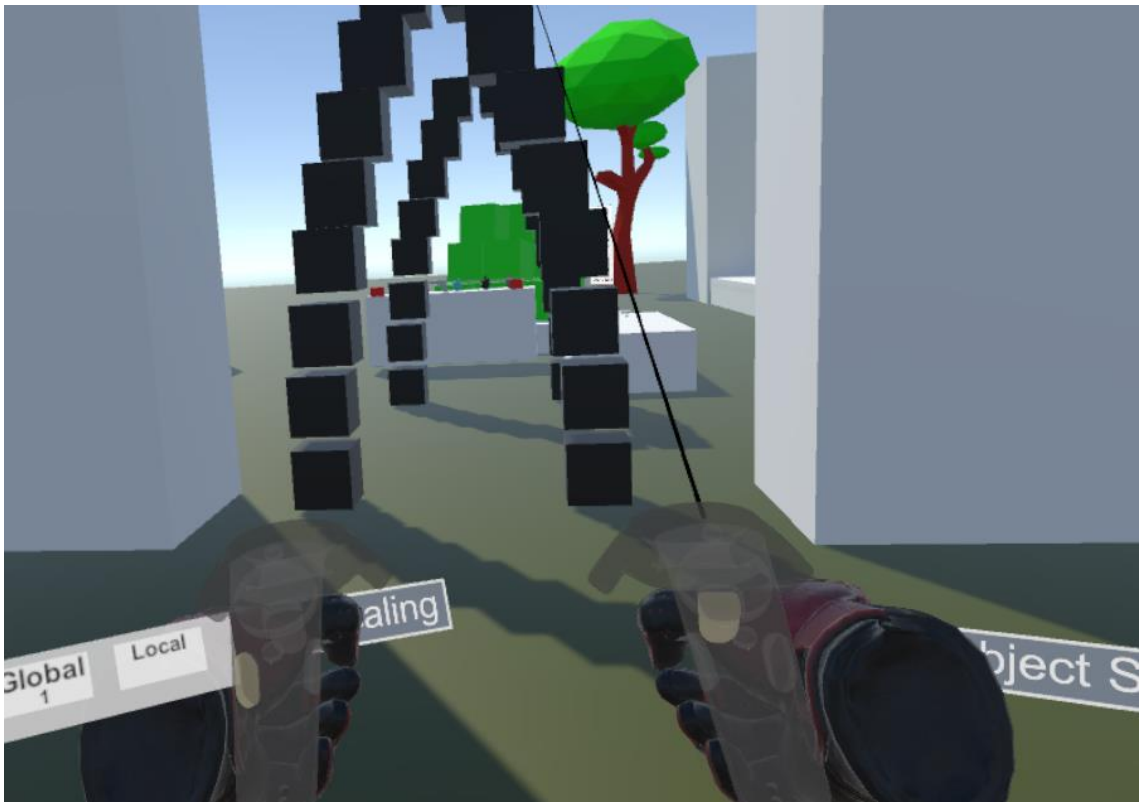


Abbildung 36 Version 3 Gate und Controllerhinweise

Hinter den ‚Gates' findet der Nutzer wie in der zweiten Version den Tisch mit den ‚Granaten'. Dieses Mal ist auf dem Tisch aber noch eine weitere ‚Granate' platziert, eine ‚Granate' in grau. Der Farbcodierung entsprechend sorgt diese ‚Granate' beim Explodieren dafür, dass die ‚GameObjects' im Explosionsradius, falls möglich, in den ‚rewind'-Status verfallen. Der ‚rewind'-Status entspricht einem ‚totalTimeScale'-Wert von ‚-1'.
Ähnlich wie in der zweiten Version der Testszene befindet sich auf dem Tisch eine ‚Handfeuerwaffe'. Diese ‚Handfeuerwaffe' ist nun durch eine Hochzieh- beziehungsweise Herunterziehbewegung auf dem ‚Touchpad' des Controllers beeinflussbar.

```
private void TouchScroll(Hand holder)
{
    if(holder == Player.instance.rightHand)
    {
        if (touchpad.GetAxis(SteamVR_Input_Sources.RightHand).y != 0)
        {
            if (touchpad.GetAxisDelta(SteamVR_Input_Sources.RightHand).y > 0.01f)
            {
                scrollIndex++;
            }
            else if (touchpad.GetAxisDelta(SteamVR_Input_Sources.RightHand).y < -0.01f)
            {
                scrollIndex--;
            }
            scrollIndex = Mathf.Clamp(scrollIndex, -30, 30);
            tType = KairosTools.ChangeColor(topMat, tType, scrollIndex);
        }
    }
```

Abbildung 37 TouchScoll vom VRGun Skript

Das Scrollen ändert die Art der ‚Bullets', welche die ‚Handfeuerwaffe' abfeuert. Das obere Ende der ‚Handfeuerwaffe' wird durch das Scrollen umgefärbt, um dem Nutzer die aktuelle ‚Bullet'-Art anzuzeigen. Die Verfärbung folgt demselben Farbcode wie auch die von ‚TimeScaleColorChange' beeinflussten ‚GameObjects'. Wird nach unten ‚gescrollt', verfärbt sich die ‚Handfeuerwaffe' zunächst türkis, was einer Verlangsamungs-‚Bullet' entspricht. In der aktuellen Version des Skripts können die ‚SlowDown-Bullets' beziehungsweise ‚Granaten' den Zeitwert nicht mehr auf unter 0 setzen. Sie reduzieren den Wert auf ein Minimum von 0,1. Wird noch weiter nach unten gescrollt, wird der obere Teil der ‚Handfeuerwaffe' schwarz. Diese Kugeln haben die Eigenschaft, die getroffenen ‚GameObjects' zu stoppen, sprich ihren ‚totalTimeScale' auf 0 zu setzen. Die dritte und letzte Stufe des Herunterscrollens entspricht dem ‚rewind'-Effekt und ist somit grau gefärbt. Der ‚totalTimeScale' wird in diesem Fall auf -1 gesetzt. Sollte hingegen nach oben gescrollt werden, färbt sich die Handfeuerwaffe' gelb und beschleunigt die getroffenen ‚GameObjects'. Um das Ändern des ‚TimeTypes' durch das ‚scrollen' zu verallgemeinern, wird es durch die Methode ‚ChangeColor()' der statischen Klasse ‚KairosTools' übernommen.

```
public static TimeType ChangeColor(Material mat, TimeType tType, int index)
{
    if (index > 15)
    {
        tType = TimeType.SpeedUp;
        mat.color = Color.yellow;
    }
    else if (index > -1)
    {
        tType = TimeType.Normal;
        mat.color = Color.white;
    }
    else if (index > -15)
    {
        tType = TimeType.SlowDown;
        mat.color = Color.cyan;
    }
    else if (index > -25)
    {
        tType = TimeType.Stop;
        mat.color = Color.black;
    }
    else
    {
        tType = TimeType.Rewind;
        mat.color = Color.gray;
    }
    return tType;
}

 public enum TimeType
 {
     Normal,
     SpeedUp,
     SlowDown,
     Stop,
     Rewind
 }
```

Abbildung 38 KairosTools ChangeColor() + TimeTypes

Hinter dem ‚Granaten'-Tisch befinden sich die ‚rewindable Cubes', welche im Vergleich zur ersten und zweiten Version etwas kleiner sind. Sie sind nun auf einer Art Podest platziert, da sich herausstellte, dass das ‚TeleportArea'-Skript von ‚SteamVR' einen kleinen Fehler aufweist: Durch das Aktivieren der ‚teleport'-Taste werden das ‚Mesh' und der ‚Collider' des ‚GameObjects', welches zuvor mit dem ‚SteamVR_TeleportArea'-Skript ausgestattet wurde, wieder aktiviert. Dadurch, dass sich dieser ‚Collider' ein Stück über der Basisebene befinden muss, damit diese Funktion genutzt werden kann, kollidieren somit die auf der Basisebene befindlichen ‚GameObjects' mit der ‚TeleportArea'. Durch diese Kollision werden zum einen die ‚OnCollisionEnter()'-Methoden der ‚GameObjects' aufgerufen, zum anderen werden die

‚GameObjects' damit leicht nach oben gedrückt. Das Podest, auf welchen nun die ‚rewindable Cubes' stehen, verhindert diesen Effekt dadurch, dass sich die besagten Würfel nun nicht mehr auf der Basisebene befinden.

Zur Rechten des ‚Granaten'-Tisches befindet sich eine kleine weitere Plattform. Auf dieser Plattform befinden sich erneut farbige Würfel mit Beschriftung. Diese Würfel geben dem Nutzer abermals Hinweise zum Farbcode, der in der Testszene genutzt wird. Zusätzlich liegt auf der Plattform eine weitere Waffe, die in der Testszene ausprobiert werden kann. Bei dieser Waffe handelt es sich um ein ‚Katana', welches ähnliche Funktionen wie die ‚Handfeuerwaffe' aufweist. Durch ‚scrollen' des ‚Touchpads' kann bei dieser Waffe ebenfalls durch die verschiedenen Modi hin und her geschaltet werden. Bei dem ‚Katana' ändert sich durch ‚scrollen' die Farbe der Klinge und dessen Funktion beim Kollidieren mit anderen ‚GameObjects'. So werden bei einer schwarzen Klinge getroffene ‚GameObjects' gestoppt und bei einer gelben Klinge beschleunigt.

Abbildung 39 Katana auf dem ColorCode-Tisch

Zwischen der Farbcodeplattform und dem ‚Granaten'-Tisch befindet sich ein schwebendes ‚UI'-Element. In dieser Version dieses ‚UserInterfaces' ist die Überschrift beziehungsweise der Ausführungshinweis „Select with Pointer" in blau gefärbt, um es von den restlichen ‚UI-Buttons' besser zu unterscheiden. Die Funktion ‚Respawn Grenades' ist wie in der zweiten Version umgesetzt, nur ist die neue ‚Granaten'-Variante ‚rewind' nun auch vertreten. Der ‚Reset Gun Button' wurde in ‚Reset Weapon' umbenannt, da er nun auch das ‚Katana' an seinen Startpunkt zurückführt. Des Weiteren wird nun beim Zurücksetzen der Waffen deren Geschwindigkeit ebenfalls zurückgesetzt, um das bizarre Verhalten, das bei sehr schnellen ‚GameObjects' auftreten konnte, zu verhindern. Der ‚Show Controller Buttons-Button' ruft erneut die Tastenbelegung über ‚SteamVR-ButtonHints' auf, um dem Nutzer diese Information bei Bedarf nochmals zu geben, ohne dass er zuerst in den ersten Raum zurückkehren muss. Als Erweiterung in dieser Version gibt es nun auch einen ‚Reset Time-Button'. Dieser ‚Button' setzt

die ‚globalTimeScale'-Variable sowie jede einzelne ‚localTimeScale'-Variable auf 1. Dies führt in den meisten Fällen zu einer Zurücksetzung auf den Startzeitwert der einzelnen Objekte. Ausgenommen von der Normalisierung der Zeitvariable sind in der Testszene allerdings die Würfel in der ‚Gate'-Formation. Diese Würfel haben beim Start der Testszene einen ‚localTimeScale'-Wert von 0, was dafür sorgt, dass sie für die erste Testphase gut positioniert bleiben und dem Nutzer einen ersten Eindruck von ‚GameObjects' in der Luft geben können. Sollte nun der ‚Reset Time- Button' ausgelöst werden, fangen diese Würfel an zu fallen, was den ersten Einsatz der ‚Rewind-Granate' interessant macht. Unterhalb des ‚Reset Time-Buttons' befindet sich der ‚Reset Cubes-Button'. Dieser ‚Button' setzt die Position und Rotation der Würfel in der Testszene wieder zurück. Somit werden die ‚Gate'-Würfel und der Würfelhaufen vor dem Nutzer wieder in die Startposition zurückgesetzt. Die ‚velocity' sowie die ‚angularVelocity' der Würfel werden in diesen Fall auch auf 0 zurückgesetzt, um einen frischen Start zu ermöglichen. Nach einigen Tests stellte sich heraus, dass die Würfel anfingen sich zu teleportieren, falls sie vom ‚Rewind'-Effekt betroffen wurden. Sie gingen somit wieder in ihre Position vor dem ‚Reset' zurück, was nicht so geplant war. Um dies zu verhindern, werden nun auch ihre gespeicherten ‚pointsInTime'-Werte beim ‚Reset' gelöscht.

```
public void ResetPosition()
{
    for (int i = 0; i < length; i++)
    {
        children[i].position = positions[i];
        if (children[i].GetComponent<Rigidbody>())
        {
            children[i].GetComponent<Rigidbody>().velocity = Vector3.zero;
            children[i].GetComponent<Rigidbody>().angularVelocity = Vector3.zero;
            if (children[i].GetComponent<TimeRigid>())
            {
                children[i].GetComponent<TimeRigid>().ResetRewind();
            }
        }
    }
}
```

Abbildung 40 ResetPosition() vom ResetPositions-Skript, das auf einem UI-Element liegt

Der letzte ‚Button' der Liste ist der ‚CleanUp Grenades-Button'. Dieser ‚Button' löscht alle herumliegenden ‚Granaten'. Bei einigen Tests stellte sich heraus, dass die ‚Granaten' durch die Explosion der anderen ‚Granaten' oftmals anfingen wegzufliegen oder wegzurollen. Dies ist am Anfang noch in Ordnung, doch durch ständiges ‚Spawnen' neuer ‚Granaten' sammelt sich in der Testszene schon sehr viel Müll an. Um ein schöneres Bild der Testszene zu erhalten, ist die Benutzung dieses ‚Buttons' unumgänglich.

Hinter dem ‚Katana'-Tisch befindet sich eine kleine abgegrenzte Laufstrecke. Auf dieser Laufstrecke läuft der ‚Testman' hin und her. Er ist mit dem ‚TimeScaleColorChanger'-Skript sowie dem ‚TimeAnimation'- und ‚TimeNavAgent'-Skript ausgestattet. Durch die Raumbegrenzung sollte der Pfad des ‚Testmans' für den Nutzer klar ersichtlich sein. Die Entfernung der Laufstrecke zum ‚Granaten'-Tisch ist ausreichend, um den ‚Testman' mit der ‚Handfeuerwaffe' gut treffen zu können. Durch Überarbeitung des ‚TimeNavAgent'-Skriptes ist das Stoppen des ‚Testmans' nun kein Problem mehr, da die alte ‚destination' zwischengespeichert wird und der ‚NavMeshAgent' beim Stopp deaktiviert wird, da dieser ein seltsames Verhalten aufwies, wenn der ‚speed' des ‚Agents' kleiner als 0 ist. Zum jetzigen Zeitpunkt ist das ‚TimeNavAgent'-Skript nicht direkt mit negativen ‚totalTimeScale'-Werten kompatibel. Für eine zukünftige Version kann dies aber in Betracht gezogen werden. Wenn eine ‚Rewind'-Funktion dennoch hinzugefügt werden soll, kann dies über das ‚TimeRigid'-Skript einfach gelöst werden.

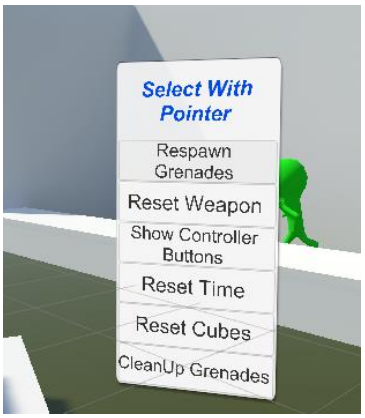


Abbildung 41 Version 3 ButtonUI

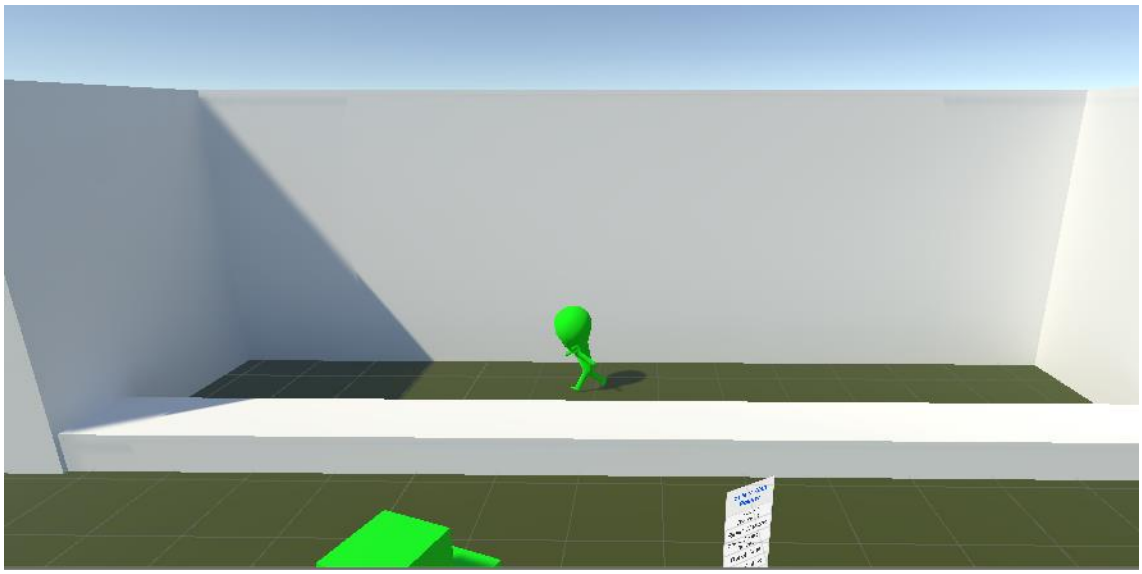


Abbildung 42 Version 3 Laufstrecke des TimeNavAgents

Links vom ‚Granaten'-Tisch geht es in die dritte und letzte Zone dieser Version der Testszene. In dieser Zone gibt es zwei Stationen: zum einen den Schießstand und zum anderen die Zeitzonen.

Der Schießstand ist ähnlich aufgebaut wie in der zweiten Version der Testszene. Allerdings wurde links vom Teleportationspunkt des Schießstandes eine Säule mit einer Kugel darauf platziert. Über der Kugel schwebt ein ‚UI'-Element, welches dem Nutzer durch die Nachricht ‚Select for GroupControl' sagt, was getan werden kann. Sollte die Kugel mit dem ‚LaserPointer' ausgewählt werden, wird statt des ‚localTimeScale'- oder ‚globalTimeScale'-Werts der ‚groupScale'-Wert angepasst. Im Falle des Schießstandes handelt es sich hierbei um die Gruppe ‚group1'. In ‚group1' befinden sich die ‚Schießstand'-Ziele sowie die ‚Handfeuerwaffe' beim Schießstand. ‚GameObjects' in dieser Gruppe sind nicht von der globalen Zeitvariable beeinflusst und können somit nur über die Kugel alle gleichzeitig beeinflusst werden.

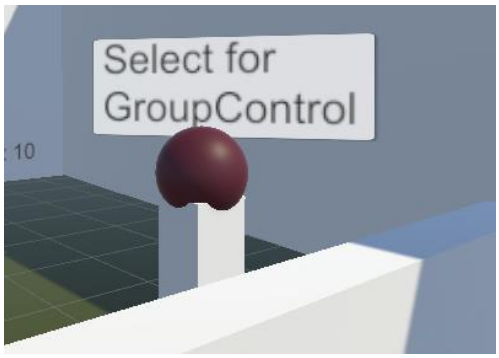


Abbildung 43 Version3 GroupScale-GameObject

Um beim Schießstand für ‚Gerechtigkeit' bei der Punkteverteilung zu sorgen, wird in der dritten Version der Testszene der Wert der Trefferzone mit dem aktuellen ‚totalTimeScale'-Wert der Zielscheibe multipliziert. Mit dieser neuen Berechnung sind verlangsamte Zielscheiben also weniger Punkte wert und schnellere Ziele mehr. Gestoppte Ziele haben nach dieser Rechnung keinen Punktewert mehr, was in zukünftigen Versionen vielleicht noch angepasst werden kann. Es gibt allerdings immer noch einen Weg, um sich das Treffen etwas zu erleichtern: Der Nutzer kann den ‚localTimeScale'-Wert der ‚Handfeuerwaffe' immer noch erhöhen, um so für schnellere ‚Bullets' zu sorgen. Neben der ‚Handfeuerwaffe' befindet sich wie gewohnt die ‚Reset'-Taste. Mit ihr wird die aktuelle Runde zurückgesetzt und die ‚Handfeuerwaffe' zurück an ihre Anfangsposition gebracht. Da sich links vom Schießstand in der dritten Version der Testszene keine Wand mehr befindet, wurde die ‚Highscore'-Tafel auf die Wand rechts vom Nutzer verschoben. Sie zeigt den höchsten Punktestand an, der in der aktuellen Sitzung erzielt wurde.

Gegenüber von dem Schießstand findet man die Zeitzonen. Ein kleines Podest mit einer Taste mit der Beschriftung ‚AutoGun On/Off' ist rechts von den eigentlichen Zonen platziert. Wird die Taste betätigt, schaltet sich die ‚AutoGun' an beziehungsweise ab. Die ‚AutoGun' ist in der linken Ecke der Zeitzonenwand platziert und feuert bei Aktivierung in regelmäßigen Abständen Projektile ab, welche in Richtung der Zeitzonen fliegen.

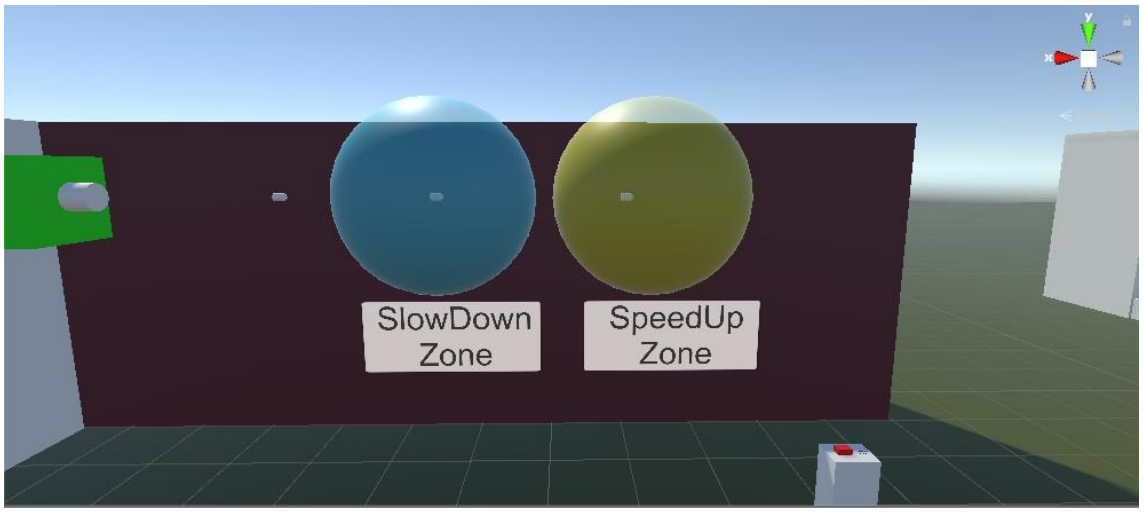


Abbildung 44 Version3 Autoschussanlage mit Zeitzonen

5.6 Probleme von Version 3

Ein Problem in der dritten Version der Testszene bestand in der ‚Rewind()'-Funktion der ‚Interactable-GameObjects', beispielsweise beim ‚Katana'. Sollte der Nutzer das ‚GameObject' während des ‚rewindens' aufheben, wird die Position und Rotation weiterhin aktualisiert, was zum Ignorieren der Nutzereingaben führt. Um dieses Problem zu lösen, wird, sobald ein ‚Interactable-GameObject' aufgenommen wird, dessen ‚PointsInTime'-Liste wieder auf einen leeren Stand gebracht.

```
if (tbase.totalTimeScale <= 0)
{
    if (interactable)
    {
        if(GetComponent<Valve.VR.InteractionSystem.Interactable>().attachedToHand == null)
        {
            rb.velocity = Vector3.zero;
            rb.angularVelocity = Vector3.zero;
        }
        else if(tbase.totalTimeScale < 0)
        {
            pointsInTime.Clear();
        }
    }
```

Abbildung 45 Leeren der Liste in TimeRigids FixedUpdate()

Ein Punkt, der in der dritten Version der Testszene verbessert werden musste, war, dass die ‚Gates' mit einem lokalen Zeitwert von 0 starteten. Dies sorgte bei einigen Testpersonen für Verwirrung. Um dies zu beheben wurde das erste ‚Gate' nun etwas modifiziert, um beim Starten der Testszene bei einem ‚totalTimeScale'-Wert von 1 nicht direkt in sich zusammen zu brechen.

Das zweite ‚Gate' wurde durch einen Turm ausgetauscht, um für mehr Stabilität des Gebildes zu sorgen und um ein besseres Versuchsobjekt für das ‚Katana' zu sein.

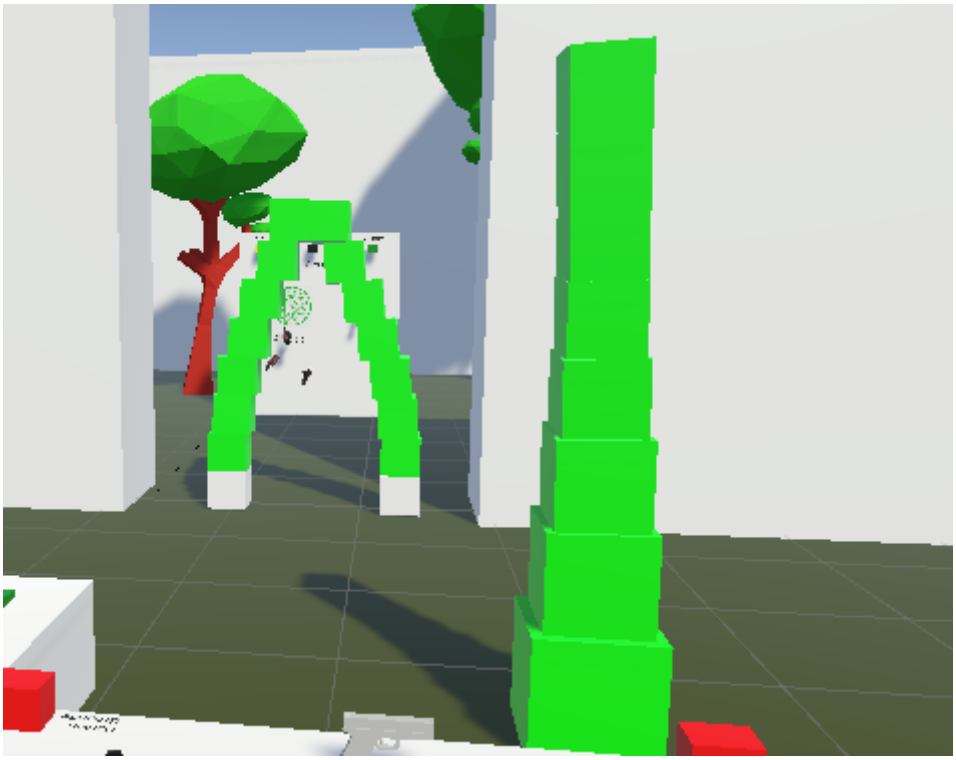


Abbildung 46 Version 3 Turm und Gate mit einem Timescale von 1

Ein weiteres Problem, das sich seit vielen Versionen zeigte, war das Fehlen der ‚Teleport'-Funktion des linken Controllers. In der ersten Version der Testszene war die entsprechende Taste des Controllers noch mit einer anderen Funktion besetzt, welche sich in späteren Versionen als irrelevant entpuppte. Durch eine einfache Änderung in der ‚SteamVR-BindingUI' war dieses Problem leicht zu lösen. Während der Behebung dieses Problems wurde noch die ‚Grip'-Funktion des linken Controllers angepasst. Sollte nun ein ‚GameObject' aufgehoben werden, wird nicht mehr parallel dazu die Zeitskalierung aktiviert. Des Weiteren wird das zusätzliche Fenster, welches sich durch die ‚Trigger'-Taste des linken Controllers öffnet, beim Halten von ‚GameObjects' nicht länger angezeigt. Somit kann nun auch mit der linken Hand normale Objektinteraktion betrieben werden.

```
void Update()
{
    if (!hand.GetComponentInChildren<Interactable>())
    {
        if (record.stateDown)
        {
            recording = true;

            rotationStart = (usePose) ? pose.lastLocalRotation: hand.transform.localRotation;
            positionStart = (usePose) ? pose.lastLocalPosition : hand.transform.localPosition;
            forwardPos = hand.transform.GetChild(1).forward;
            if (circle != null)
                circle.gameObject.SetActive(true);
```

Abbildung 47 GetControllerRotation deaktiviert sich beim Halten eines Interactables

Das letzte große Problem in dieser Testszene ist ein allgemein bekanntes: Dem Nutzer wird zu viel auf einmal zum Entdecken gegeben. Um dies zu beheben, folgt eine vierte und letzte Version der Testszene.

5.7 Version 4

In der letzten Version der Testszene wird dem Nutzer grundsätzlich alles in kleinen Portionen gezeigt. Er startet in dieser Szene in einem langgezogenen Raum. Vor ihm befindet sich eine ‚Autoschussanlage', welche ‚Bullets' durch einige ‚TimeZones' schießt. Die verschossenen ‚Bullets' besitzen das ‚TimeScaleColorChanger'-Skript und ändern somit ihre Farbe danach, in welcher ‚TimeZone' sie sich befinden. Hinter den ‚TimeZones' an der Wand befinden sich Elemente auf denen geschrieben steht, um was für eine Zone es sich jeweils handelt. Etwas weiter unterhalb der ‚TimeZones' ist die Reihe an Würfeln mit der Beschriftung für die Zeitfarbkodierung platziert, um den Nutzer noch einmal das aktuelle Geschehen zu erklären. Zunächst fliegen die ‚Bullets' durch eine ‚SlowDown'-Zone. Diese hat der Farbcodierung entsprechend einen türkisen Farbton, welcher leicht transparent ist, um das Innere der Zone zu zeigen. Anschließend fliegen die ‚Bullets' durch eine ‚SpeedUp'-Zone, um den gegenteiligen Effekt zu demonstrieren. Die ‚SpeedUp'-Zone ist demgemäß in einem gelbtransparenten Farbton dargestellt. Am Ende der ‚Schusslinie' befindet sich die schwarztransparente ‚Stop'-Zone. Sollten die ‚Bullets' mit ihr in Berührung geraten, werden sie sofort gestoppt und einige Zeit später entfernt, um einer Überflutung an ‚Bullets' entgegenzuwirken. Am Anfang dieser Testszene besitzt der Nutzer noch keine zeitmanipulierenden Fähigkeiten. Diese werden ihm erst beim weiteren Voranschreiten gegeben.

Abbildung 48 Version 4 Zeitzonen | Links Schussstart +slowDown / Rechts speedUp- + stop-Zone

Gegenüber des ‚TimeZone'-Demonstrationsbereiches befinden einige weitere ‚TimeZones'. Vor diesen sind einige Würfel platziert, die mit dem ‚Interactable'- sowie dem ‚Throwable'-Skript von ‚SteamVR' ausgestattet sind. Diese Würfel sind dazu gedacht, dem Nutzer die Möglichkeit zu geben, durch eigene Handlungen mit den ‚TimeZones' zu interagieren. So kann der Nutzer die Würfel hochheben und probieren, diese durch die ‚TimeZones' zu werfen. Sollte er eine der Zonen treffen, wird der Würfel dem ‚TimeScaleColorChanger'-Skript entsprechend gefärbt und durch die ‚TimeZones' selbst physikalisch modifiziert.

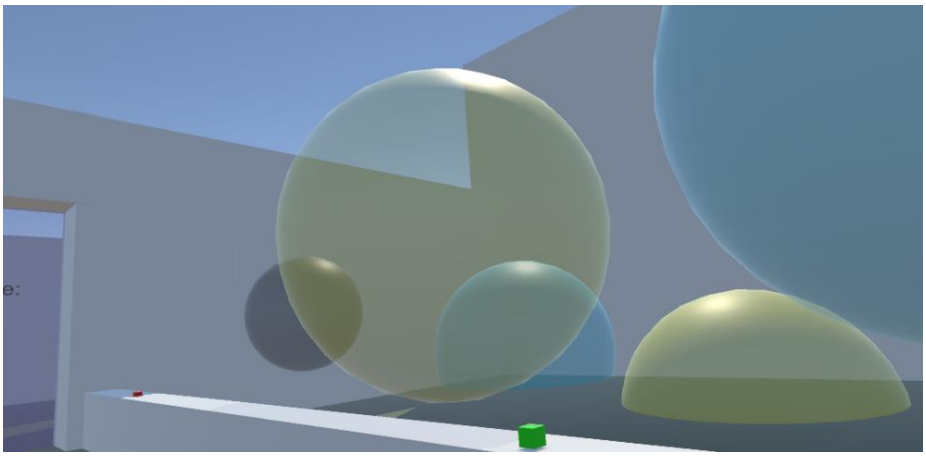


Abbildung 49 Version 4 Würfel durch Zeitzonen werfen

Folgt der Nutzer nun dem Gang weiter bis zu einer Art Tor, sieht er vor sich eine lilatransparente Wand. Durch diese Wand kann er sich hindurchbewegen, um den nächsten Raum zu erreichen. Sobald er diesen Raum betritt, wird ein ‚ControllerButtonHint'-Event für den rechten Controller aufgerufen.

Abbildung 50 Version 4 Barriere zum dritten Bereich

Dieses Tutorial zeigt dem Nutzer die Tastenbelegung der rechten Hand an. Somit lernt er nun ‚GameObjetcs' auszuwählen und die ‚Bullets' durch eine Scrollbewegung mit dem ‚TouchPad' zu wechseln. In diesem Bereich befindet sich der ‚Schießstand', der schon aus älteren Versionen der Testszene bekannt sein sollte. Hier kann der Nutzer zum ersten Mal mit der ‚Handfeuerwaffe' interagieren und die zeitmanipulierenden ‚Bullets' verstehen. Als Erweiterung des ‚Schießstandes' gibt es nun direkt vom Teleportpunkt aus die Farbkodierung zu sehen.

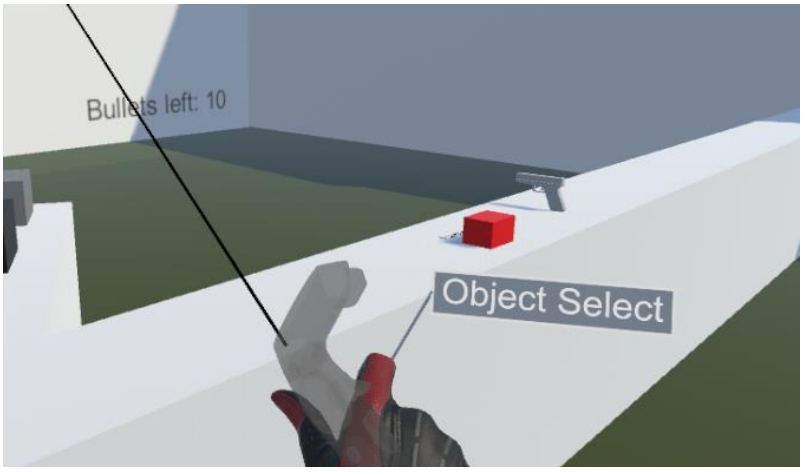
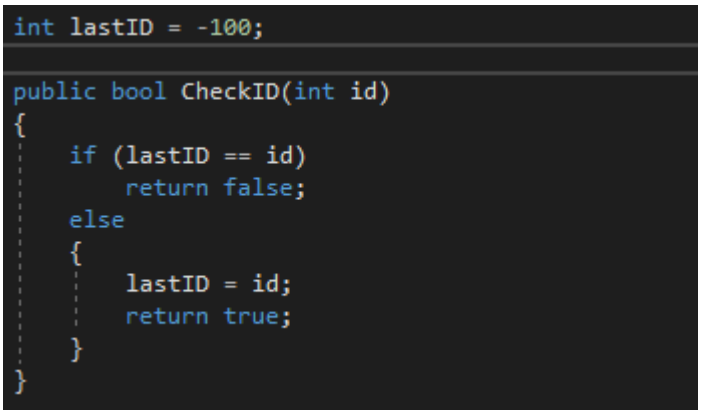


Abbildung 51 Version 4 Schießstand + ButtonHints

Zusätzlich zur Anzeige der Tastenbelegung sorgt das Eintreten durch die Barriere für einen zeitvariablen ‚Reset', um für ein frisches Erlebnis zu sorgen. Andernfalls könnte es dazu kommen, dass die Zeitvariablen in einem Raum so manipuliert wurden, dass zum Beispiel alle ‚GameObjects' gestoppt wurden, was im Falle der ‚TimeZones' zu einem sehr ernüchternden Erlebnis führen würde. Da die ‚Teleport'-Funktion von ‚SteamVR' jedes Mal die

‚OnTriggerEnter()‘-Funktion aufruft, die im Bereichsskript ‚TimeResetTrigger‘ enthalten ist, musste ein weiteres Skript erstellt werden, welches die Aufgabe hat zu überprüfen, ob der Bereich nun tatsächlich ein anderer ist. Das ‚TriggerZoneOrganizer‘-Skript überprüft mit seiner ‚public‘-Methode, ob die aktuelle ‚zoneID‘ dieselbe ist wie die ‚zoneID‘, die beim letzten ‚Teleport‘ genutzt wurde, und gibt dementsprechend ein ‚true‘ oder ‚false‘ aus. Der ‚lastID‘-Wert ist zu Beginn der Szene absichtlich auf einen unrealistischen Wert gesetzt worden, um die Fehleranfälligkeit so niedrig wie möglich zu halten.



```
int lastID = -100;

public bool CheckID(int id)
{
    if (lastID == id)
        return false;
    else
    {
        lastID = id;
        return true;
    }
}
```

Abbildung 52 TriggerZoneOrganizer-Skript CheckID()

Die ‚zoneID‘-Variable des ‚TimeResetTrigger’-Skripts ist ein ‚public int‘, um für einen einfachen Zugriff im Inspector zu sorgen. Zusätzlich kann eine ‚TimeGroup‘ ausgewählt werden, die ebenfalls zurückgesetzt werden soll. Wird nun die ‚OnTriggerEnter()‘-Methode des Skripts aufgerufen, wird auch die ‚CheckID()‘-Methode vom ‚parent‘ aufgerufen. Sollte es sich nun um eine neue Zone handeln, werden die Zeitvariablen auf 1 gesetzt und gegebenenfalls noch weitere Skripte aufgerufen. Für eine einfache ‚GameObject‘-Findung sind alle ‚Trigger‘-Zonen ‚children‘ des ‚GameObjects‘, welches das ‚TriggerZoneOrganizer‘-Skript trägt.

In der nächsten Zone, welche sich gegenüber vom ‚Schießstand‘ befindet, werden dem Nutzer beim ersten Betreten die zeitmanipulierenden Fähigkeiten gegeben. Diese befinden sich wie gewohnt auf dem linken Controller. Zusätzlich werden dem Nutzer alle relevanten Tasten durch ein ‚ButtonHint‘-Event erläutert. Direkt im Sichtfeld platziert befindet sich die ‚Tutorial‘-Wand, die schon in vorherigen Versionen benutzt wurde. Rechts von dieser Wand befindet sich der rotierende Würfel mit der ‚Value‘-Anzeige, dieses Mal gibt es allerdings noch einen Hinweis darauf, wie die lokalen Zeitwerte von Objekten manipuliert werden können. Des Weiteren befindet sich in diesem Raum eine Topfpflanze, die in ihrer Animationsgeschwindigkeit verändert werden kann.

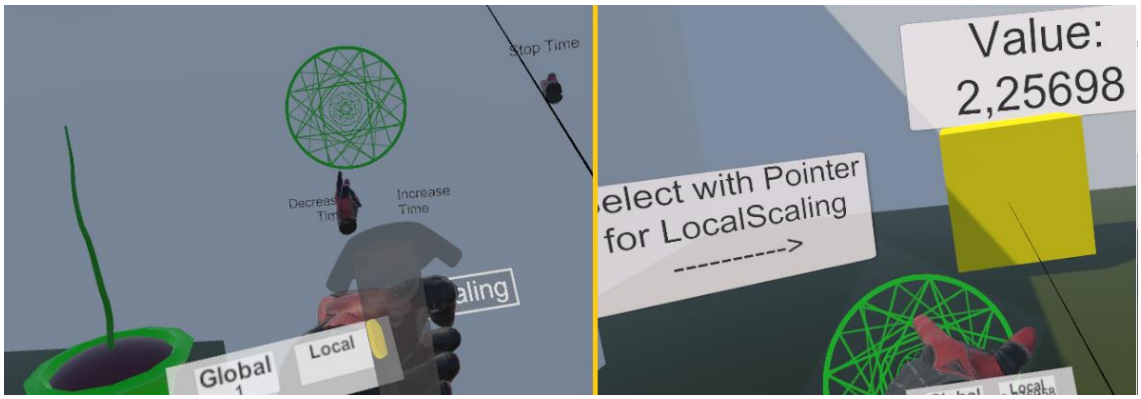


Abbildung 53 Version 4 Links ControllerHint / Rechts RotatingCube lokales Skalieren

Vorne links von der ‚Tutorial'-Wand befindet sich der Durchgang in den letzten Raum der Testszene. Wieder werden beim Betreten der neuen Zone die Zeitvariablen auf 1 zurückgesetzt und als Erweiterung auch die Würfel an ihre ursprüngliche Position zurückgebracht. In diesem letzten Bereich befinden sich erneut bekannte Objekte aus der dritten Version der Testszene. Hierzu zählen der Tisch mit den ‚Granaten' und der ‚Handfeuerwaffe' sowie die fliegende ‚ButtonUI' und ein weiterer Tisch mit dem Farbcode und dem ‚Katana'. Des Weiteren befindet sich links die Laufstrecke des ‚TimeNavAgents' und hinter dem Granatentisch der ‚rewindable'-Würfelhaufen. Der Turm und das ‚Gate' aus Würfeln sind ebenfalls erreichbar und im hinteren Bereich sind auch die ‚low-poly-trees' zu finden.
Der letzte Bereich soll eine Spielfläche der Funktionen sein, die der Nutzer schlussendlich erreicht, sobald er die anfänglichen Erklärungen und nach und nach verstanden hat.

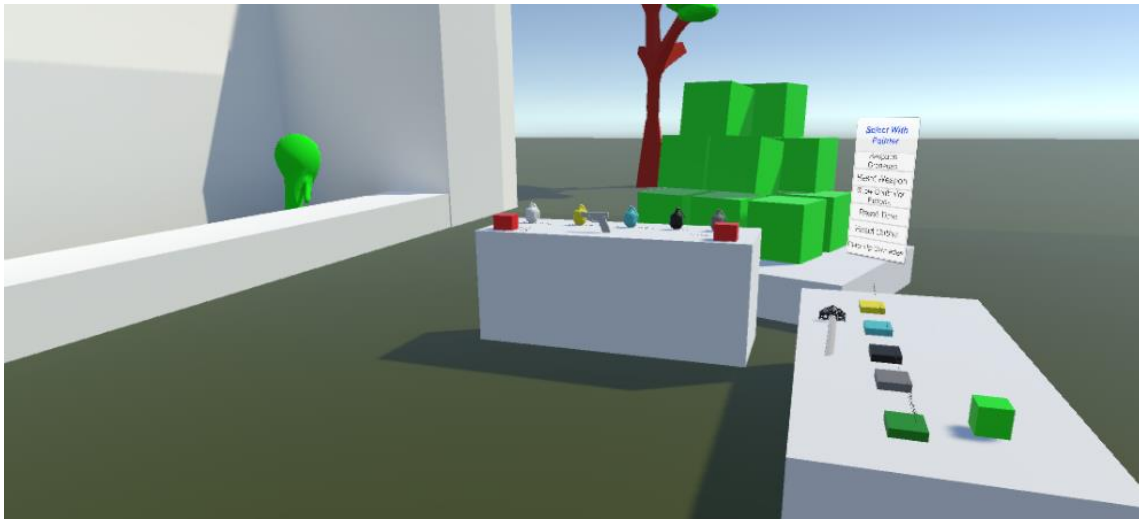


Abbildung 54 Version 4 Bekannte Elemente am Ende der Testszene

6 Fazit/Zukunftsausblick

‚KairosVR: Erstellung eines Assets zur Zeitmanipulation durch Gestensteuerung in Virtual Reality' ist ein Projekt, das durchaus als gelungen bezeichnet werden kann. Die zu Beginn der Arbeit herausgearbeiteten Grundmechaniken sind mit den Skripten des ‚Unity'-Projektes umsetzbar. Des Weiteren sind die Gestensteuerungen zwar simpel, aber dennoch funktional, auch wenn sich die Geste der Controllerrotation doch als unangenehmer herausstellte als anfangs angenommen.

Dieses Projekt besitzt noch einige weitere Schwächen, die in zukünftigen Versionen behoben werden können. Eines dieser Probleme ist die nicht ganz einfache Handhabung der Skripte für einen Nutzer, der noch nicht sehr viel Programmiererfahrung hat. Einige ‚Editor'-Skripte sind bereits im Projekt enthalten, dennoch sind es noch nicht ansatzweise genug, um für eine unkomplizierte und dennoch übersichtliche Einbindung in andere Projekte zu sorgen. Viele der Skripte könnten wahrscheinlich zusammengefasst werden, da viele Funktionen auf einmal von Nutzern genutzt würden. Beispielsweise könnten die Skripte erkennen, welche Komponenten wirklich genutzt werden und die Funktionalitäten der Skripte demensprechend freischalten.

Schlussendlich ist ‚KairosVR' ein Projekt mit vielen kleinen Fehlern, dennoch funktionieren die Grundmechaniken der Zeitmanipulation zuverlässig und machten den Testern Spaß.

7 Abbildungsverzeichnis

Abbildung 1 Kairos-Relief von Lysippos, Kopie in Trogir....6
Quelle: https://de.wikipedia.org/wiki/Kairos#/media/Datei:Kairos-Relief_von_Lysippos,_Kopie_in_Trogir.jpg
Abgerufen: 09-01.2020 (17:30)

Abbildung 2 Oculus Rift Consumer Version 1....9
Quelle: https://biareview.com/oculus-rift/
Abgerufen 07.01.2020 (16:05)

Abbildung 3 HTC VIVE: (Links) Base Stations, (Mitte) Headset und (Rechts) Controllers....10
https://www.vive.com/de/product/
Abgerufen: 09.01.2020 (17:29)

Abbildung 4 Prince of Persia: The Sands of Time UI....11
Eigener Screenshot von ©Ubisoft : Prince of Persia: The Sands of Time (Uplay-Fassung)

Abbildung 5 Singletoncode vom TimeManager-Skript....14
Abbildung 6 RequireComponent auf dem TimeAnimation-Skript....14
Abbildung 7 GetControllerRotation-Skript Rotationserkennungslösung....17
Abbildung 8 GetControllerRotation Skript Stopp durch Distanzunterschied....18
Abbildung 9 Version 1 Riesige UI im Raum....19
Abbildung 10 Version 1 Granatentisch....19
Abbildung 11 Version 1 low-poly-tree und Topfpflanzen....20
Abbildung 12 Version1-2 HandUI aus unterschiedlichen Winkeln....21
Abbildung 13 Neue HandUI durch TriggerPress | Links nichts ausgewählt / Rechts Objekt ausgewählt....22
Abbildung 14 TimeRigid Version 1....22
Abbildung 15 TimeRigid Version1.1....23
Abbildung 16 TimeBase Skript Update + ChangedValue Coroutine....24
Abbildung 17 TimeRigid FixedUpdate Version2 ChangedScale....25
Abbildung 18 TimeRigid AntiGravity bei Zeitskalierung 0....25
Abbildung 19 RigidTools Auschnitt AddForce()....26
Abbildung 20 TimeRigid Manuelle Gravitationskraft....26
Abbildung 21 Enum TimeGroup....27
Abbildung 22 TimeGroupHandler BasicSkript....27

Abbildung 23 TimeGroupHandlerEditor Skript 28
Abbildung 24 TimeGroupHandler Inspektor mit Checkbox 29
Abbildung 25 TimeBase Berechnung anhand von TimeGroups 29
Abbildung 26 VR Handfeuerwaffe 30
Abbildung 27 VRGun Skript V1 Update 30
Abbildung 28 RigidTools ForceMode Berechnung 32
Abbildung 29 TimeRigid Experimental Physics 33
Abbildung 30 Version 2 Startpunkt 34
Abbildung 31 Version 2 Explosive Cubes 35
Abbildung 32 Version 2 TimeAnimation und TimeBlendshapes 36
Abbildung 33 Version 2 Schießstand 37
Abbildung 34 TimeChangeColorChanger - links Awake() - rechts Update() 38
Abbildung 35 Version 3 Startwand mit Tutorial 39
Abbildung 36 Version 3 Gate und Controllerhinweise 40
Abbildung 37 TouchScoll vom VRGun Skript 41
Abbildung 38 KairosTools ChangeColor() + TimeTypes 42
Abbildung 39 Katana auf dem ColorCode-Tisch 43
Abbildung 40 ResetPosition() vom ResetPositions-SKript das auf einem UI-Element liegt 44
Abbildung 41 Version 3 ButtonUI 45
Abbildung 42 Version 3 Laufstrecke des TimeNavAgents 45
Abbildung 43 Version3 GroupScale-GameObject 46
Abbildung 44 Version3 Autoschussanlage mit Zeitzonen 47
Abbildung 45 Leeren der Liste in TimeRigids FixedUpdate() 47
Abbildung 46 Version 3 Turm und Gate mit einem Timescale von 1 48
Abbildung 47 GetControllerRotation deaktiviert sich beim Halten eines Interactables 49
Abbildung 48 Version 4 Zeitzonen | Links Schussstart +slowDown / Rechts speedUp- + stop-Zone 50
Abbildung 49 Version 4 Würfel werfen durch Zeitzonen 50
Abbildung 50 Version 4 Barriere zum dritten Bereich 51
Abbildung 51 Version 4 Schießstand + ButtonHints 51
Abbildung 52 TriggerZoneOrganizer-Skript CheckID() 52
Abbildung 53 Version 4 Links ControllerHint / Rechts RotatingCube lokales Skalieren 53
Abbildung 54 Version 4 Bekannte Elemente am Ende der Testszene 53

8 Literaturverzeichnis

Atsma, A. J. (2000-2017). *Theoi Project*. Retrieved 12 07, 2019, from https://www.theoi.com/Daimon/Kairos.html

HTC. (2011). *Vive*. Abgerufen am 06. 12. 2019 von https://www.vive.com/de/product/

Moor, T. D. (2019, 01 20). *onebonsai*. Retrieved 12 06, 2019, from https://lab.onebonsai.com/a-brief-history-of-virtual-reality-6b2a5f6f5c41

Scharlock, T. (19. 10 2015). *vrlife*. Abgerufen am 06. 12 2019 von https://vrlife.de/oculus-rift-story/

The Blender Foundation. (2019). *Blender*. Abgerufen am 06. 12. 2019 von https://www.blender.org/about/