



Thesis zum Erlangen des Grades Bachelor of Science
im Studiengang
Medieninformatik (MIB)

**Konzeption und Entwicklung eines
Singleplayer Adventure Games zum
Thema multidimensionale Realitäten**

Erstbetreuer : Prof. Jirka Dell’Oro-Friedl

Zweitbetreuer : Prof. Dr. Gabriel Rausch

Vorgelegt am : 28. Februar 2023

Vorgelegt von : Nick Philipp Häcker
Matrikelnummer: 262144
Gaußstraße 82a, 70193 Stuttgart
nick.philipp.haecker@hs-furtwangen.de

Abstract

“*A Fraction of Time (Arbeitstitel)*” basiert auf der Idee, den Spieler verschiedene parallele Zeitlinien von sich selbst erzeugen zu lassen. Der Spieler muss in diesen verschiedenen Zeitlinien, mithilfe diverser Zeitlinien von sich selbst, Rätsel lösen und mit seinen Zeitlinien zusammenarbeiten. Grundlage dieser Weiterentwicklung ist der digitale Prototyp, der bereits im Gamedesign Workshop im Wintersemester 2021/2022 entwickelt wurde.

Die Zielsetzung dieser Abschlussarbeit ist es, das bisher implementierte System auf technischer Seite so weit zu verbessern, dass es für den Spieler offensichtlich fehlerfrei funktioniert. Außerdem soll ein Rahmen geschaffen werden, in dem es möglich ist, Erweiterungen zu integrieren. Hinzu werden weitere Inhalte, wie weitere Levels und Spielwelten konzipiert und umgesetzt. Der Spieler soll dabei eine Variation an Spielwelten und Rätsel erhalten, welche er lösen muss, um das jeweilige Level abzuschließen. Hierbei soll ihm auch das Spielprinzip so vermittelt werden, dass er durch das Verständnis der Mechanik jedes Rätsel lösen kann. Validiert werden die Inhalte durch User-Tests, bei denen das Verständnis der Spielmechanik auf inhaltlicher und technischer Ebene abgefragt wird.

Gender-Hinweis

Zur besseren Lesbarkeit wird in dieser Hausarbeit das generische Maskulinum verwendet. Die in dieser Arbeit verwendeten Personenbezeichnungen beziehen sich – sofern nicht anders kenntlich gemacht – auf alle Geschlechter.

Inhaltsverzeichnis

Abstract	I
Inhaltsverzeichnis	V
Abbildungsverzeichnis	XIII
Quellcodeverzeichnis	XIX
Abkürzungsverzeichnis	XXI
1 Einleitung	1
1.1 Motivation und Aufgabenstellung	1
1.2 Struktur der Arbeit	1
2 Grundlagen	3
2.1 Forschungsstand zu Adventure-Games	3
2.2 Forschungsstand Replay-Systeme in Videospielen	4
2.3 Determinismus	4
3 Konzeption	7
3.1 Genre	7
3.1.1 Steampunk	7
3.2 Story	8
3.3 Spielmechanik	8
3.4 Spielablauf	10
3.4.1 Spielablauf des Spiels	10
3.4.2 Level Ablauf	12

3.4.3	Labor Ablauf	13
3.5	Zeitkontinuum	14
3.5.1	Zeitlinienkonvergenz	15
3.5.2	Graph	15
3.5.3	Zeitlinie	15
3.5.4	Timer	18
3.5.5	Avatar	19
3.5.6	Bewegungen und Interaktionen	19
3.5.7	Einflüsse auf die Spielwelt	19
3.6	Belohnungen	20
3.7	Labor	20
3.8	Aufnahmen des Chronologen	20
3.8.1	Platzhalter	21
3.8.2	Geister	21
3.9	Gebrauchsgegenstände	21
3.9.1	Steuereinheit	21
3.9.2	Datenleser	22
3.9.3	Imitator	22
3.9.4	Pager	22
3.10	Leveldesign	23
3.10.1	Interaktionsobjekte	23
3.10.2	Rätsel	24
3.11	Informationen für den Spieler	45
3.11.1	Informationen über die Spielwelt	46
3.11.2	Informationen über Szeneobjekte	46
3.11.3	Informationen über das User-Interface	46

3.11.4 Informationen über den derzeitigen Storystand	46
3.11.5 Informationen über die Maschine	47
3.12 Dialoge, die im Spiel vorkommen	47
3.12.1 Eingeworfene Dialoge	47
3.12.2 Story relevante Dialoge	48
3.13 Sounddesign	48
3.13.1 Hintergrund Musik	48
3.13.2 Umgebungsgeräusche	49
3.13.3 Charakter Geräusche	49
3.13.4 Dialoge	49
4 Visuelles Design des Prototyps	51
4.1 Moodboard	51
4.2 Art-Stil	52
4.3 Aussehen des Chronologen	52
4.3.1 Nicht mehr vom Spieler gesteuerte Chronologen	54
4.4 User Interface	55
4.4.1 Graph	56
4.4.2 Timer	57
4.4.3 Zeitkontinuum	57
4.4.4 Charakter Übersicht	58
4.4.5 Steuerhinweis	58
4.4.6 Dialoge	58
4.4.7 Systemfehler und Benachrichtigungen	58
4.4.8 Tragen von Objekten	59
4.4.9 Datenleser	60
4.4.10 Einstellungsmenü im Datenleser	60

4.4.11	Bildschirmanzeige zum Abschluss des Levels	61
4.4.12	Bildschirmanzeige bei Auftreten einer Zeitlinienkonvergenz	61
4.5	Führung durch das Level	62
4.5.1	Partikelsysteme	62
4.5.2	Kletterkanten	63
4.5.3	Tooltips	64
4.6	Interaktionsgegenstände	67
4.6.1	Druckplatte	67
4.6.2	Hebel	68
4.6.3	Ventil	69
4.6.4	Stützen	71
4.6.5	Portal Steuereinheit	72
4.6.6	Lore	74
4.7	Gebrauchsgegenstände	74
4.7.1	Steuereinheit	75
4.7.2	Datenleser	77
4.7.3	Pager	78
4.7.4	Imitator	79
4.8	Menü	81
4.8.1	Pause	81
4.8.2	Start	81
4.9	Leveldesign	83
4.9.1	Tutorial	83
4.9.2	Labor	85
4.9.3	Labor des Chronologen	86
5	Umsetzung des Prototyps	89

5.1	Verwendete Technologien	89
5.1.1	Unity Version 2021.3.16f1	89
5.1.2	Blender 3.3.1	89
5.1.3	Stable Diffusion	90
5.1.4	Mixamo	90
5.1.5	Envato Elements	90
5.1.6	Cinemachine 2.8.9.	90
5.1.7	Filmbox Exporter 4.1.3.	90
5.1.8	TextMeshPro 3.0.6.	91
5.1.9	Universal Render Pipeline 12.1.8.	91
5.1.10	Unity Prefab Assets	91
5.2	Ausgangssituation	92
5.2.1	Übernommene Aspekte	92
5.2.2	Aspekte zum Überarbeiten	100
5.3	Aufbau des Prototyps	105
5.3.1	Vorstellung der Spiellogik	105
5.3.2	Vorstellung der Spielszenen	108
5.3.3	Aufbau der Szenelogik	111
5.3.4	Vorstellung der einzelnen Komponenten	114
5.3.5	Vorstellung des Algorithmus, der entscheidet, wann gesplittet und wann gemergt wird	125
5.3.6	Vorstellung des Splits	128
5.3.7	Vorstellung der Methode OnTakeTimeline	130
5.3.8	Vorstellung des Merges	131
5.3.9	Vorstellung des HandleTimelineManagers	135
5.3.10	Vorstellung der Erzeugung und Reproduktion von Interaktionen	137

5.3.11 Fehlverhalten der Programmlogik	143
5.4 Technische Einbindung von Sounds	149
6 Evaluation	151
6.1 Zielsetzung	151
6.2 Planung und Durchführung	151
6.3 Auswertung der Tests	152
6.3.1 Umgesetztes Feedback	152
6.3.2 Anstehenden Verbesserungen und Optimierungen	153
6.3.3 Feedback, das im Ausblick mit einem neuen System angepasst werden sollte	154
6.3.4 Allgemeines Feedback	155
7 Fazit	157
7.1 Zusammenfassung	157
7.2 Ausblick	158
Literaturverzeichnis	161
Eidesstattliche Erklärung	165
Anhänge	167
Anhang A Anhang	167
A.1 Konzeption	167
A.1.1 Gamedesign Dokument	167
A.1.2 Mockups	167
A.1.3 UML Diagramme	167
A.2 Entwickelter Prototyp	167
A.2.1 Alter Prototyp	167
A.2.2 Neuer Prototyp	168
A.3 Durchgeführte Nutzertests	168

A.3.1 Leitfaden der Nutzertests	168
A.3.2 Ergebnisse der Nutzertests	168
A.4 Präsentation	168

Abbildungsverzeichnis

Abbildung 1:	Aktivitätsdiagramm des gesamten Spiels, (Quelle: eigene Darstellung), (In groß im Anhang A 1.3.1)	11
Abbildung 2:	Aktivitätsdiagramm eines Levels, (Quelle: eigene Darstellung) . . .	12
Abbildung 3:	Aktivitätsdiagramm des Labors, (Quelle: eigene Darstellung) . .	13
Abbildung 4:	Neu entstandene Zeitlinie nach Split, (Quelle: eigene Darstellung)	16
Abbildung 5:	derzeitige Zeitlinie, (Quelle: eigene Darstellung)	16
Abbildung 6:	Entstehender Baum (Graph) mit Verzweigungen (Zeitlinien), (Quelle: eigene Darstellung)	17
Abbildung 7:	Entstandener Baum mit Knotenpunkten, (Quelle: eigene Darstellung)	17
Abbildung 8:	verschiedene Anreihungen an Zeitlinien, nummeriert, (Quelle: eigene Darstellung)	26
Abbildung 9:	Rätsel im ersten Tutorial Raum, (Quelle: eigene Darstellung) . .	27
Abbildung 10:	Lösung zu Rätsel im ersten Tutorial Raum, (Quelle: eigene Darstellung)	27
Abbildung 11:	Element 1 im zweiten Tutorial Raum, (Quelle: eigene Darstellung)	28
Abbildung 12:	Graph zu Element 1 im zweiten Raum, (Quelle: eigene Darstellung)	28
Abbildung 13:	Element 2 im zweiten Tutorial Raum, (Quelle: eigene Darstellung)	29
Abbildung 14:	Lösung Element 2 im zweiten Tutorial Raum, (Quelle: eigene Darstellung)	29
Abbildung 15:	Lösung Variante 1 im zweiten Tutorial Raum, (Quelle: eigene Darstellung)	29

Abbildung 16: Lösung Variante 2 im zweiten Tutorial Raum, (Quelle: eigene Darstellung)	30
Abbildung 17: Element 1 im dritten Tutorial Raum, (Quelle: eigene Darstellung)	30
Abbildung 18: Element 2 im dritten Tutorial Raum, (Quelle: eigene Darstellung)	31
Abbildung 19: Element 1 in der Stromkammer, (Quelle: eigene Darstellung)	32
Abbildung 20: Element 2 in der Stromkammer, (Quelle: eigene Darstellung)	32
Abbildung 21: Element 3 in der Stromkammer, (Quelle: eigene Darstellung)	33
Abbildung 22: Element 4 in der Stromkammer, (Quelle: eigene Darstellung)	34
Abbildung 23: Lösungsweg gesamt Element 2 bis 4, (Quelle: eigene Darstellung)	34
Abbildung 24: Element 5 in der Stromkammer, (Quelle: eigene Darstellung)	35
Abbildung 25: Element 6 in der Stromkammer, (Quelle: eigene Darstellung)	36
Abbildung 26: Element 7 in der Stromkammer, (Quelle: eigene Darstellung)	37
Abbildung 27: Element 8 in der Stromkammer, (Quelle: eigene Darstellung)	38
Abbildung 28: Element 9 in der Stromkammer, (Quelle: eigene Darstellung)	39
Abbildung 29: Lösungsweg gesamt Element 8 bis 9, (Quelle: eigene Darstellung)	40
Abbildung 30: Element 10 in der Stromkammer, (Quelle: eigene Darstellung)	40
Abbildung 31: Element 1 im Laboreingang, (Quelle: eigene Darstellung)	41
Abbildung 32: Element 2 im Laboreingang, (Quelle: eigene Darstellung)	42
Abbildung 33: Element 3 im Laboreingang, (Quelle: eigene Darstellung)	43
Abbildung 34: Element 4 im Laboreingang, (Quelle: eigene Darstellung)	44
Abbildung 35: Lösungsweg gesamt Element 1 bis 4, (Quelle: eigene Darstellung)	44
Abbildung 36: Moodboard des Spiels, (Quellen: (von oben nach unten und von links nach rechts) sathak (2019), SaferDan (2013), Plays (o. D.), A44 (o. D.), Games (k.D.), Gamestar & Schemes (2016)) Schick (2017), Plane (o. D.), Sykoo (2018), Games (o. D.) Key (o. D.)	51
Abbildung 37: Spielobjekt des Chronologen, (Quelle: eigene Darstellung)	52

Abbildung 38: Spielobjekt des Chronologen, gelbe Weste, (Quelle: eigene Darstellung)	53
Abbildung 39: 3D Modell des Chronologen, blaue Weste, (Quelle: eigene Darstellung)	53
Abbildung 40: 3D Modell Platzhalter, blaue Weste, (Quelle: eigene Darstellung)	54
Abbildung 41: Platzhalter des Chronologen, (Quelle: eigene Darstellung)	55
Abbildung 42: Mockup Bildschirm des Spielers, (Quelle: eigene Darstellung)	55
Abbildung 43: Mockup Zeitlinienübersicht im UI, (Quelle: eigene Darstellung)	56
Abbildung 44: Mockup geöffnete Zeitlinienübersicht, (Quelle: eigene Darstellung)	56
Abbildung 45: Beispiel wie Chronologe Objekt trägt, (Quelle: GameTube (2014))	59
Abbildung 46: User-Interface beim Tragen eines Spielobjektes, (Quelle: eigene Darstellung)	59
Abbildung 47: Datenleser Collage, (Quelle: eigene Darstellung)	60
Abbildung 48: Indiz für einen neuen Eintrag, (Quelle: eigene Darstellung)	60
Abbildung 49: Einstellungsmenü im Datenleser, (Quelle: eigene Darstellung)	60
Abbildung 50: Endbildschirm eines erfolgreich gemeisterten Levels, (Quelle: eigene Darstellung)	61
Abbildung 51: Endbildschirm bei einer Zeitlinienkonvergenz, (Quelle: eigene Darstellung)	62
Abbildung 52: Partikelsystem Druckplatte, (Quelle: eigene Darstellung)	63
Abbildung 53: Kantenmarkierung an Spielweltobjekten, (Quelle: eigene Darstellung)	63
Abbildung 54: Hebel mit Tooltip, (Quelle: eigene Darstellung)	64
Abbildung 55: Treppe mit Tooltip, (Quelle: eigene Darstellung)	65
Abbildung 56: Treppe mit Tooltip und Aufforderung, (Quelle: eigene Darstellung)	65
Abbildung 57: Hebel mit Tooltip und Aufforderung, (Quelle: eigene Darstellung)	66
Abbildung 58: Kante mit Tooltip und Aufforderung, (Quelle: eigene Darstellung)	66
Abbildung 59: Druckplatte, (Quelle: eigene Darstellung)	67

Abbildung 60: Hebel, (Quelle: eigene Darstellung)	68
Abbildung 61: Mit Hebel wurde interagiert, (Quelle: eigene Darstellung)	68
Abbildung 62: Hebel mit Lampeninterface, (Quelle: eigene Darstellung)	69
Abbildung 63: Ventil Version 1, (Quelle: eigene Darstellung)	69
Abbildung 64: Ventil Version 2, (Quelle: eigene Darstellung)	70
Abbildung 65: Ventil Version 3, (Quelle: eigene Darstellung)	70
Abbildung 66: Zeichnung einer einfachen Stütze mit Zahnrädern, (Quelle: eigene Darstellung)	71
Abbildung 67: Zeichnung einer Reihe an Stützen angeschlossen an Mechanismus, (Quelle: eigene Darstellung)	71
Abbildung 68: tragbares Treppenelement, (Quelle: eigene Darstellung)	72
Abbildung 69: Computer des Chronologen, erstes Konzept, (Quelle: Stable Diffusion)	73
Abbildung 70: Portal der Maschine, erstes Konzept, (Quelle: Stable Diffusion) .	74
Abbildung 71: Lore, (Quelle: eigene Darstellung)	74
Abbildung 72: Steuereinheit, (Quelle: eigene Darstellung)	75
Abbildung 73: gesplitteter Zeitlinien Status, (Quelle: eigene Darstellung)	76
Abbildung 74: aktueller Zeitlinien Status, (Quelle: eigene Darstellung)	76
Abbildung 75: Zeitlinienstatus zum Zurückmergen, (Quelle: eigene Darstellung)	77
Abbildung 76: Vorderseite Datenleser, Konzeptzeichnung, (Quelle: Stable Diffusion)	78
Abbildung 77: Rückseite Datenleser, Konzeptzeichnung, (Quelle: Stable Diffusion)	78
Abbildung 78: Pager, (Quelle: Stable Diffusion)	79
Abbildung 79: Imitator, (Quelle: eigene Darstellung)	79
Abbildung 80: Pausemenü, (Quelle: eigene Darstellung)	81
Abbildung 81: Startmenü, (Quelle: eigene Darstellung)	82
Abbildung 82: Toneinstellungen, (Quelle: eigene Darstellung)	82

Abbildung 83: Kameraeinstellungen, (Quelle: eigene Darstellung)	83
Abbildung 84: Collage Tutorial, (Quelle: eigene Darstellung)	84
Abbildung 85: Collage Labor, (Quelle: eigene Darstellung)	85
Abbildung 86: Collage Labor des Chronologen, (Quelle: Plane (o. D.) (rechts oben), Stable Diffusion (rest))	87
Abbildung 87: Schaubild zu Codeausschnitt „Ausschnitt aus Timeline aus dem alten Prototyp“, (Quelle: eigene Darstellung)	93
Abbildung 88: Spiellogik des Prototyps, (Quelle: eigene Darstellung)	105
Abbildung 89: Spiellogik im Klassendiagramm eingezeichnet, (Quelle: eigene Darstellung), (In groß im Anhang A 1.3.2)	106
Abbildung 90: Szenen Ordner in Unity, (Quelle: eigene Darstellung)	109
Abbildung 91: Szenen Ordner im Detail, (Quelle: eigene Darstellung)	109
Abbildung 92: Szenehierarchie, (Quelle: eigene Darstellung)	110
Abbildung 93: Eintritts- und Austritts-Szenehierarchie, (Quelle: eigene Darstellung)	111
Abbildung 94: Szenelogik ohne Kernmechanik, (Quelle: eigene Darstellung), (Vollständig im Anhang A 1.3.2)	112
Abbildung 95: Szenelogik des Prototyps, (Quelle: eigene Darstellung), (In groß im Anhang A 1.3.2)	113
Abbildung 96: Klassendiagramm aus GraphController.cs, (Quelle: eigene Darstellung)	117
Abbildung 97: Aktivitätsdiagramm SaveTimeline, (Quelle: eigene Darstellung) .	118
Abbildung 98: Aktivitätsdiagramm LoadTimeline, (Quelle: eigene Darstellung)	118
Abbildung 99: Klassendiagramm der TimeController.cs, (Quelle: eigene Darstellung)	120
Abbildung 100: Klassendiagramm der Recordable.cs, (Quelle: eigene Darstellung)	121
Abbildung 101: Klassendiagramm des Executable.cs, (Quelle: eigene Darstellung)	122
Abbildung 102: Klassendiagramm der CharacterController.cs, (Quelle: eigene Darstellung)	123

Abbildung 103: Klassendiagramm der GroundController.cs, (Quelle: eigene Darstellung)	124
Abbildung 104: Aktivitätsdiagramm wie überprüft wird, ob ein Split oder Merge Prozess startet, (Quelle: eigene Darstellung)	125
Abbildung 105: Aktivitätsdiagramm eines Splits, (Quelle: eigene Darstellung) . . .	128
Abbildung 106: Klassendiagramm eines Splits mit der OnTakeTimeline Methode in der CharacterController.cs, (Quelle: eigene Darstellung), (In groß im Anhang A 1.3.2)	129
Abbildung 107: Aktivitätsdiagramm der TakeCurrentTimeline Methode, (Quelle: eigene Darstellung)	130
Abbildung 108: Aktivitätsdiagramm eines Merges, (Quelle: eigene Darstellung), (In groß im Anhang A 1.3.2)	131
Abbildung 109: Klassendiagramm eines Merges ohne OnTakeTimeline Methode in CharacterController.cs, (Quelle: eigene Darstellung), (In groß im Anhang A 1.3.2)	132
Abbildung 110: Klassendiagramm der grundlegenden Struktur bei Aufnahme und Reproduktion, (Quelle: eigene Darstellung), (In groß im Anhang A 1.3.2)	138
Abbildung 111: Klassendiagramm einer SoundTrigger.cs Komponente, (Quelle: eigene Darstellung)	149
Abbildung 112: Klassendiagramm der AudioSourceController.cs Komponenten, (Quelle: eigene Darstellung), (In groß im Anhang A 1.3.2) . . .	150

Quellcodeverzeichnis

Codeausschnitt 5.1: Ausschnitt aus GraphController.cs aus dem alten Prototyp	92
Codeausschnitt 5.2: Ausschnitt aus Timeline.cs aus dem alten Prototyp	93
Codeausschnitt 5.3: HandleMerge Methode aus dem alten Prototyp	94
Codeausschnitt 5.4: HandleMerge Methode aus dem alten Prototyp	95
Codeausschnitt 5.5: Zentrale Logik der Mechanik aus dem Gamedesign Workshop	96
Codeausschnitt 5.6: Rekonstruktionsmethode aus Shadow.cs	97
Codeausschnitt 5.7: Aufnahme von Bewegungen, Ausschnitt aus MovementAbility.cs	99
Codeausschnitt 5.8: Interaktionsklasse	100
Codeausschnitt 5.9: HandleCharacterSelection in der PlayerController.cs	101
Codeausschnitt 5.10: Alte HandleSplit und HandleMerge Methode aus PlayerController.cs	102
Codeausschnitt 5.11: HandleSplit und HandleMerge Methode aus GraphController.cs	103
Codeausschnitt 5.12: GameConfig.cs des Prototyps	106
Codeausschnitt 5.13: SceneData.ts aus Prototyp	107
Codeausschnitt 5.14: CharacterData.ts des Prototyps	114
Codeausschnitt 5.15: Ausschnitt aus Timeline.ts dieses Prototyps	114
Codeausschnitt 5.16: Ausschnitt aus TimeController.ts dieses Prototyps	120
Codeausschnitt 5.17: Ausschnitt aus Character.ts dieses Prototyps	122
Codeausschnitt 5.18: Split/Merge Überprüfung aus GraphController.ts	125
Codeausschnitt 5.19: OnRemoveOldPlaceholder aus GraphController.ts	133
Codeausschnitt 5.20: OnTimelinesToHandle aus GraphController.ts	134
Codeausschnitt 5.21: HandleTimelineManager aus GraphController.ts	135
Codeausschnitt 5.22: Information.cs des Prototyps	139
Codeausschnitt 5.23: Ausschnitt aus InteractionAbility.cs	140
Codeausschnitt 5.24: Ausschnitt aus ReconstructionInteractionAbility.cs	142
Codeausschnitt 5.25: Ausschnitt aus CharacterController.cs	144
Codeausschnitt 5.26: Ausschnitt aus CharacterController.cs	145
Codeausschnitt 5.27: Ausschnitt aus Moveable.cs	147
Codeausschnitt 5.28: Ausschnitt aus MoveableAnimationController.cs	148

Abkürzungsverzeichnis

3D dreidimensional

CTA Concurrent Think Aloud

DOTS Data-Oriented Technology Stack

EA Electronic Arts

ESC Escape

FBX Filmbox

GUI Grafische-User-Interfaces

HDRP High-Definition Render Pipeline

HFU Hochschule Furtwangen University

HUD Head-Up-Display

ID Identifikationsnummer

LTS Long Term Support

UI User Interface

URP Universal Render Pipeline

1. Einleitung

Das grundlegende Konzept dieser Bachelor-Thesis wurde im Rahmen des Gamedesign-Workshops an der Hochschule Furtwangen University (HFU) im Wintersemester 2021/2022 entworfen. Durch das gute Feedback am Tag der Medien 2021 und das Interesse an einer echten spielfähigen Version wurde beschlossen, das Konzept weiterzuentwickeln. Inhalte aus dem im Anhang beiliegenden Gamedesign-Dokument wurden übernommen, einige wurden verbessert und andere verworfen. Der Titel des Projektes heißt „*A Fraction of Time*“.

1.1. Motivation und Aufgabenstellung

Das Ziel dieser Bachelorarbeit ist es, den bisherigen digitalen und spielbaren Prototypen aus dem Gamedesign Workshop 2021/2022 auf technischer Ebene neu zu entwickeln. Das System soll zum einen fehlerfrei funktionieren und zum anderen um weitere Rätsel, Dialoge, Interaktionsobjekte und neue Szenarien erweiterbar sein. Zusätzlich soll eine Einführung des Spielers durch ein Tutorial erfolgen.

Die Leitfragen, an denen sich die Entwicklung und Konzeption des Prototyps orientiert hat, lauten:

- Wie kann das Spielkonzept von „*A Fraction of Time*“ als leicht verständliches Tutorial dargestellt werden?
- Wie komplex können Rätsel aufgebaut werden, sodass der Spieler maximal herausgefordert wird?

1.2. Struktur der Arbeit

Diese Arbeit gibt zunächst in „Kapitel 2: Grundlagen“ einen Überblick über die theoretischen Grundlagen des Determinismus, der für die spätere Umsetzung relevant ist. Dabei wird ebenfalls auf Spieldatei eingegangen, die von der Art ähnlich sind. Darauf folgend wird in „Kapitel 3: Konzeption“ auf die grundlegende Konzeption des Spiels eingegangen. Dabei werden Themen angesprochen, die entweder nur für das User Interface (UI) Design wichtig sind oder lediglich für die spätere Umsetzung. In „Kapitel

4: Visuelles Design des Prototyps" wird auf die gestalterischen Aspekte des Prototyps eingegangen. In "Kapitel 5: Umsetzung des Prototyps" wird die Umsetzung des Systems erklärt, das anschließend in "Kapitel 6: Evaluation" mit geführten User-Tests validiert wird. Abschließend werden ein Ausblick und ein Fazit über diese Arbeit und der dahinter stehenden Entwicklung gegeben.

2. Grundlagen

Im folgenden Kapitel werden die für das Spiel wichtigen grundlegenden Elemente beschrieben. Zunächst erfolgt eine erklärende Einführung in das Genre der Adventure-Games, die für die Konzeption dieses Spieles relevant ist. Im Anschluss wird darauf eingegangen, welche Spiele aus dem Adventure oder Action-Adventure-Genre im Single- oder Multiplayer-Modus einen Bezug auf das hier beschriebene Spiel besitzen. Darauffolgend wird auf die für das Spiel wichtige Spielmechaniken eingegangen. Die Betrachtung von Replay-Systemen und der damit einhergehende Determinismus der Physik-Engine ist dabei der wichtigste Bezugspunkt.

2.1. Forschungsstand zu Adventure-Games

Folgendes Zitat beschreibt den Aufbau von Adventure-Games:

“Adventure games focus on puzzle solving within a narrative framework, generally with few or no action elements [...]” (Bronstring (2012))

Auf dem Spielemarkt befinden sich viele Kooperative-Spiele, die entweder im Offline oder Online-Multiplayer gespielt werden können. Darunter fallen Titel wie “*A Way Out*” aus dem Hause “*Electronic Arts (EA)*” (Arts (2017)) oder “*It Takes Two*”, welches ebenfalls aus dem Hause “*EA*” stammt und von dem Entwicklerstudio “*Hazelight*” entwickelt wurde (vgl. Arts (2022)). Bekannte Spiele, welche kooperativ im Singleplayer wirken, sind Spiele “*The last of us*” oder der “*Uncharted*”- Reihe, bei denen an bestimmten Stellen der dargebotenen Rätsel NPC zur Seite springen und dem Spieler helfen (vgl. LLC (2022)) (vgl. Naughty Dog (2021)). Um auf die zeitliche Komponente einzugehen, befassen sich Spiele wie “*Quantumbreak*” von “*Xbox Games Studio*” (vgl. Entertainment (2016)) oder “*Bioshok Infinitie*” (vgl. Games (2013)) von “*Irrational Games*” mit geschichtlich festgelegten Zeitreisen bzw. Zeitsprünge in die Vergangenheit. Das bedeutet, dass Reisen in der Zeit als Kernpunkt der Geschichte anzusehen ist und passiert auf lineare Weise. Es hat keine Auswirkung auf das Spielgeschehen. Hingegen beschäftigt sich das Spiel “*Life is Strange*” (vgl. *Life is Strange Remastered / SQUARE ENIX (o. D.)*) damit, Zeitpunkte in der Vergangenheit zu ändern, um zukünftige Ereignisse zu beeinflussen.

2.2. Forschungsstand Replay-Systeme in Videospielen

Replay oder Wiederholungssysteme gibt es hauptsächlich bei Ego-Shootern und Rennspielen. Bei Ego-Shootern wird dieses System häufig bei der Wiederholung des letzten "Kills" in der Runde verwendet. Dabei werden die gespeicherten Inputs der Spieler neu in der Game-Engine ausgewertet, wodurch der Eindruck entsteht, eine Kamera würde das Gesehene aufnehmen (vgl. Project (2021)). Eine solche Kill-Cam ist in dem Spiel "*Call of Duty: Black Ops 2*" von "Activision" (vgl. Activision Publishing (2021)) integriert. Allerdings sind dies nur die Inputs der Spieler, die in einer deterministischen Engine ausgewertet wurden. Etwas Ähnliches gibt es auch bei Rennspielen, bei denen ausgewählte Rennen nochmals angesehen werden können. In "*Forza Horizon 4*" von den Entwicklerstudios "*Playground Games*" und "*Turn 10 Studios*" ist ein Replay-System eingebaut. Der Spieler kann einzelne Fahrten aufnehmen und diese über die Wiederholfunktion anschauen (vgl. Games (2018)). Dabei werden auch wieder die Inputs des Spielers gespeichert und beim Betrachten der Wiederholung erneut in der Sequenz ausgespielt. Wie bereits erwähnt ist es wichtig, dass die Engine deterministisch ist, damit das Endergebnis der nachgeholten Simulation identisch ist zu dem, was der Spieler zur Laufzeit des Rennens erlebt hat. Das Spiel "*The Talos Principle*" vom Entwicklungsteam "*Croteam*" ist ebenfalls ein Rätselspiel, bei dem der Spieler bestimmte Bewegungen und Interaktionen aufnehmen kann. Die aufgenommene Bewegung hat ebenfalls einen Einfluss in die Spielwelt wie auch der Spieler. Der Spieler kann diese Aufnahmen allerdings nicht verschachteln, wie es in dieser Bachelorarbeit der Fall sein wird (vgl. Croteam (2014)).

2.3. Determinismus

Determinismus in Computerspielen bezieht sich auf die Tatsache, dass jede Aktion des Spielers, die in einem bestimmten Zustand des Spiels ausgeführt wird, immer das gleiche Ergebnis hat. Das bedeutet, dass das Spielverhalten vorhersehbar und reproduzierbar ist, da jede Aktion des Spielers die gleiche Auswirkung auf den Zustand des Spiels hat (vgl. Sun (2019)). Wenn der Protagonist im Kontext des Spiels von einer Plattform springt und auf dem Boden landet, so sollte es immer denselben Sprung und denselben Fall geben und er sollte an derselben Stelle aufkommen. Diese Gegebenheit ist wichtig, um ein Replay-System einzubauen, welches den Kern dieses Spiels ausmacht. Deshalb ist es wichtig zu wissen, ob und auf welche Weisen die Engine Unity deterministisch ist. Unity ist grundlegend nicht deterministisch. Das liegt daran, dass die Prozesse, die im Hintergrund der Engine laufen, nicht immer gleich ausgeführt werden können, sei es plötzliches Schwanken der Bildrate oder der ungenauen Berechnung der Gravitation. Deshalb ist es notwendig, die Auswirkungen, die es auf den

Spielercharakter geben kann, bei der Aufnahme zu berücksichtigen. Es ist allerdings möglich, das System so umzubauen, damit es hinreichend deterministisch ist. In dieser Arbeit wird versucht, ein hinreichend deterministisches System einzubauen. Da Unity grundlegend nicht deterministisch ist, kann ein Replay-System, wie wir es von Rennspielen oder Ego-Shootern kennen, nicht eingebaut werden. Das Neuinterpretieren der Spieler-Eingaben könnte zu einem anderen Ergebnis führen, als es bei der Aufnahme der Fall war.

Seit März 2022 ist ein erster Experimental-Release eines neuen Technologie-Stacks von Unity erschienen, welcher größere skalierte Simulationen und Performance Scaling unterstützt. Dieser Technologie Stack wird Data-Oriented Technology Stack (DOTS) genannt und wird vermutlich in nächster Zeit vollständig erscheinen. Da dieser Stack noch nicht vollständig erschienenen ist, und die Gefahr zu groß ist, dass der Prototyp durch Softwarefehler von DOTS nicht spielbar wird, wurde darauf verzichtet, diesen Technologie Stack zu integrieren. Für die weitere Entwicklung dieses Technologie Stacks ist es aber denkbar, ihn einzubinden (vgl. Technologies (o. D.a)). Dieser Aspekt wird in “Kapitel 7.2: Ausblick” dieser Thesis erneut aufgegriffen.

3. Konzeption

In diesem Abschnitt wird das allgemeine Spielkonzept des Spiels, sowie die damit einhergehende Story des Spiels vorgestellt. Danach wird auf alle wichtigen Aspekte des Spiels eingegangen. In diesem Abschnitt werden alle grundlegenden Konzeptionen für das Spiel aufgelistet.

3.1. Genre

Das Setting dieses Spiels ist dem Steampunk unterzuordnen.

Das Steampunk-Genre wurde ausgewählt, weil es die passende Kombination aus Ästhetik und Technologie bietet. Die mechanische Ästhetik des Steampunks passt zu den Maschinen, die im Spiel zum Einsatz kommen. Die futuristische Technologie, die auf Dampfmaschinen und Mechanik basiert, verleiht dem Spiel einen einzigartigen Charme und eine besondere Atmosphäre. Die Verwendung von Kupfer, Messing, Leder und anderen Materialien, die an das Viktorianische Zeitalter erinnern, runden das Steampunk-Erlebnis ab und lässt den Spieler in eine andere Welt eintauchen.

Zum allgemeinen Design wird im "Kapitel 4: Visuelles Design des Prototyps" näher eingegangen.

3.1.1. Steampunk

Steampunk ist ein Kunstwort aus "Steam" (Dampf) und "Punk" und bezeichnet ein Literatur- und Kunstgenre, das im 19. Jahrhundert angesiedelt ist und die Technik und Erfindungen dieser Zeit mit moderner Technologie kombiniert. Es zeichnet sich durch eine Kombination aus Nostalgie und Hightech sowie Dampf- oder Zahnrad betriebener Technik und viktorianischen Kleidungsstilen aus. Steampunk hat sich zu einer kulturellen Bewegung, Mode und Kunst entwickelt und wird in der Literatur, Musik, Comics, Gesellschaft und Videospielen umgesetzt (vgl. Daniel (2022)).

3.2. Story

Der Spieler verfolgt die Geschichte eines Chronologen, der eine Maschine erfindet, um mit deren Technologie in andere Zeitlinien zu reisen. Dadurch möchte er Auswirkungen von anderen getroffenen Entscheidungen in anderen Zeitlinien erforschen und diese sichtbar machen. Die Maschine hat jedoch eine Fehlfunktion, anstatt dass er in andere Zeitlinien reisen kann und wieder zurück, wird er in eine alternative Zeitlinie gezogen und kommt von dort nicht mehr zurück. Über seine Erfindung kann er allerdings andere Zeitlinien in die Zeitlinie, in der er sich befindet, ziehen und diese haben dadurch einen Einfluss auf die Zeitlinie, in der er sich befindet. Da seine Maschine nicht wie vorgesehen funktioniert, muss er nun einen Weg finden, um sie zu reparieren. Dazu muss er durch verschiedene andere Zeitlinien reisen um Hinweise zu finden, wie er in seine eigene Zeitlinie zurückfinden kann.

Seine Motive ändern sich jedoch, als er feststellt, dass alle alternativen Zeitlinien von einer zukünftigen Version seiner selbst besucht wurden, die den Zeitlinien wichtige Teile entnommen und sie in Dystopien verwandelt hat. Bald findet er sich dabei wieder, sein zukünftiges Ich zu jagen, andere Zeitlinien zu reparieren, während er versucht einen Weg zurück in seine ursprüngliche Zeitlinie zu finden.

Der Protagonist wird auf seiner Reise verschiedene alternative Zeitstrände besuchen und ihre Geschichte kennenlernen, um damit seine Ziele zu erreichen. Während seiner Reisen wird er entdecken, wie die Bewohner der anderen Zeitlinien von der Existenz von parallelen Zeitlinien erfahren haben, genauer gesagt glauben zu wissen, was sich in anderen Zeitlinien abspielt. Die Geschichten der Zeitstrände werden hauptsächlich durch die Umgebung und durch kleine Objekte erzählt, die gesammelt werden können.

Die Geschichte des Spiels wird in fünf Kapitel geteilt. Die ersten fünf Level, die der Spieler absolvieren muss, bilden das erste Kapitel. Nach Abschließen des ersten Kapitels kann der Spieler in das Labor des Chronologen zurückreisen, um von dort seine Expeditionen und Herausforderungen zu starten.

3.3. Spielmechanik

Grundsätzlich ist das Spiel ein Rätsel-Spiel, das einen Bezug zur Zeit im Allgemeinen und zu einem fiktionalen "Zeitkontinuum" hat. Das "Zeitkontinuum" regelt Existenz und Abläufe von verschiedenen "Zeitlinien". Diese "Zeitlinien" können vom Spieler erzeugt werden, um auf diese Weise Türen für andere "Zeitlinien" zu öffnen. Diese "Zeitlinien" bewegen sich in einer hoch-zählenden Zeit. Das heißt jede "Zeitlinie" wurde zu einem bestimmten Zeitpunkt des Spiels erstellt und endet auch zu dem Zeitpunkt, an dem man von einer neu erstellten "Zeitlinie" wieder auf die "Grund-

zeitlinie" zurückwechselt. Der Chronologe, der Protagonist in diesem Spiel, bewegt sich dabei als Fixpunkt in diesem "Zeitkontinuum" auf seiner "Zeitlinie" voran, diese ist auch die "Grundzeitlinie". Alle weiteren Erstellten "Zeitlinien" werden entweder in Referenz zu dieser "Grundzeitlinie" gesetzt oder werden an andere neu erstellten "Zeitlinien" untergegliedert. Im "Kapitel 5: Umsetzung des Prototyps" erhält man einen genaueren Überblick darüber.

Der Spieler hat nun die Möglichkeit, durch diese Gegebenheit des "Zeitkontinuums", andere Ichs aus anderen "Zeitlinien" zu sich herzuholen und diese in sein "Zeitkontinuum" zu integrieren. Das bedeutet, er springt in die Person eines anderen Ichs und bewegt sich in dem Körper des anderen Ichs in der Spielwelt umher. Jede Aktion oder Bewegung, die ausgeführt wird, hat immer einen Einfluss auf danach entstehende "Zeitlinien". Deshalb kann er unter anderem eine verschlossene Tür öffnen, die später für die "Grundzeitlinie" geöffnet wird. Da jede "Zeitlinie" einen bestimmten Start-Zeitpunkt besitzt, kann nach einem Wechsel auf die Basis "Zeitlinie" nun zu dem entsprechenden Zeitpunkt des Kontinuums diese aufgenommene "Zeitlinie" ausgeführt werden. Während diese "Zeitlinie" in der Welt, in Form des entsprechenden Ichs des Chronologen, umhergeht, hat diese "Zeitlinie" eine Auswirkung auf den ursprünglichen Chronologen. Wenn diese "Zeitlinie" nun eine Druckplatte oder einen Hebel betätigt, so wird für den Chronologen eine Tür geöffnet, die vorher noch verschlossen war. Er kann durch die geöffnete Tür durchschreiten und hat somit eines von vielen Rätseln, auf die er stoßen wird, gelöst.

Einen Sprung von einer abgespaltenen, neu erzeugten "Zeitlinie" auf den referenzierten Ursprung kann man als Sprung in die Vergangenheit sehen, da der Spieler nun wieder auf einen zurückliegenden Punkt in der Zeit, auf den letzten Zeitwert, den diese "Zeitlinie" erlebt hat, zurückgesetzt wird. Er erfährt dadurch, welche Auswirkungen die erzeugte "Zeitlinie" auf ihn hat.

Der Spieler hat die Möglichkeit ab einem bestimmten Zeitpunkt der Geschichte in sein Labor zurückzureisen. Er kann von dort aus bereits absolvierte Level erneut starten. Nach Abschließen eines jeden Levels erhält der Spieler eine Belohnung. Diese Belohnungen können neu freigeschalteten Ichs aus anderen Zeitsträngen sein, oder Gegenstände, die der Spieler im weiteren Verlauf des Spiels einsetzen kann, um sich das Spiel leichter zu gestalten. Jedes Level wird ab einem bestimmten Zeitpunkt der Geschichte auch dazu dienen, dass der Spieler mit einer größeren Anzahl an verfügbaren Ichs andere Wege gehen kann, um dadurch weitere Level freizuschalten oder weitere Gegenstände zu finden. Beides wird im "Kapitel 3.6: Belohnungen" aufgegriffen und beschrieben. Das Erzeugen einer "Zeitlinie" wird im Sprachgebrauch dieser Thesis als "*Splitting*" bezeichnet, dieser Begriff wird noch einmal im gleichnamigen

Abschnitt erklärt. Gleiches gilt für den Begriff “*Merging*”, der für das Zurückspringen von einer erzeugten “*Zeitlinie*” auf seiner Referenzierten “*Zeitlinie*” verwendet wird.

3.4. Spielablauf

Die Spielabläufe des Spiels können in drei verschiedene Abschnitte unterteilt werden. Zum einen den Spielablauf des gesamten Spiels, den Ablauf eines einzelnen Levels und anschließend den Ablauf des Labors.

3.4.1. Spielablauf des Spiels

Sobald das Spiel gestartet ist und das Startmenü geladen wurde, kann der Spieler zwischen vier verschiedenen Optionen auswählen. Zunächst kann er aus dem Hauptmenü heraus Einstellungen bezüglich der sichtbaren UI tätigen, die Grafik, der Ton-Einstellung sowie der Kamera-Einstellung. Er kann das Spiel hier auch wieder beenden. Sofern der Spieler bereits einen Spielstand besitzt, kann er an diesem weiterspielen und würde entweder in seinem Labor beginnen, oder er startet das letzte freigeschaltete Level. Hat er das Spiel zum ersten Mal gestartet, so beginnt der Spieler im Tutorial. Zu Beginn des Tutorials kann der Spieler entscheiden, ob er das Tutorial Level als ein Tutorial spielen möchte, oder ob er das Level ohne den geführten Dialog meistern will. Mit geführtem Dialog bekommt der Spieler Hinweise über Objekte, die er anfangs suchen und betätigen muss. Zudem wird dem Spieler mitgeteilt, wie der Chronologe seine Erkenntnisse bezüglich der Mechanik des Spiels äußert. Dadurch wird dem Spieler erklärt, was bei den Prozessen des “*Splittings*” und des “*Mergings*” genau passiert und was diese Prozesse für eine Wirkung besitzen. Zudem erklärt der Chronologe so auch die Bedeutung der UI, die der Spieler zur Laufzeit sehen kann. Der Spieler kann durch die Auswahl auch ohne diese Dialoge spielen. So erhält er nur über erhaltene Informationspakete Informationen über das Geschehen des Spiels. Diese Informationen werden auch bei einem geführten Tutorial an den Spieler übergeben. Nach Abschließen des Tutorials erhält der Spieler ein Überblick über den Graphen, eine Bündelung aller entstandenen Zeitlinien zur Laufzeit des Levels, und welches Level er jetzt als Nächstes starten kann. Daraufhin sieht der Spieler in Form einer Zeichensequenz, wie der Chronologe durch ein Zeitportal schreitet und in eine weitere Zeitlinie übergeht. Bevor er nun das zweite Level starten kann, sieht der Spieler eine Zeichensequenz, wie der Chronologe die neue Spielwelt betritt und die Umgebung erkundet. Anschließend startet das Level für den Spieler. Die ersten fünf Level, inbegriffen das Tutorial, sind im selben Muster, wie es beim Tutorial beschrieben wurde aufgebaut. Im fünften Level erhält der Spieler Informationen, auf welche Weise er in sein Labor zurückkreisen kann. Allerdings befindet sich dieses nicht da, wo es vor dem Start

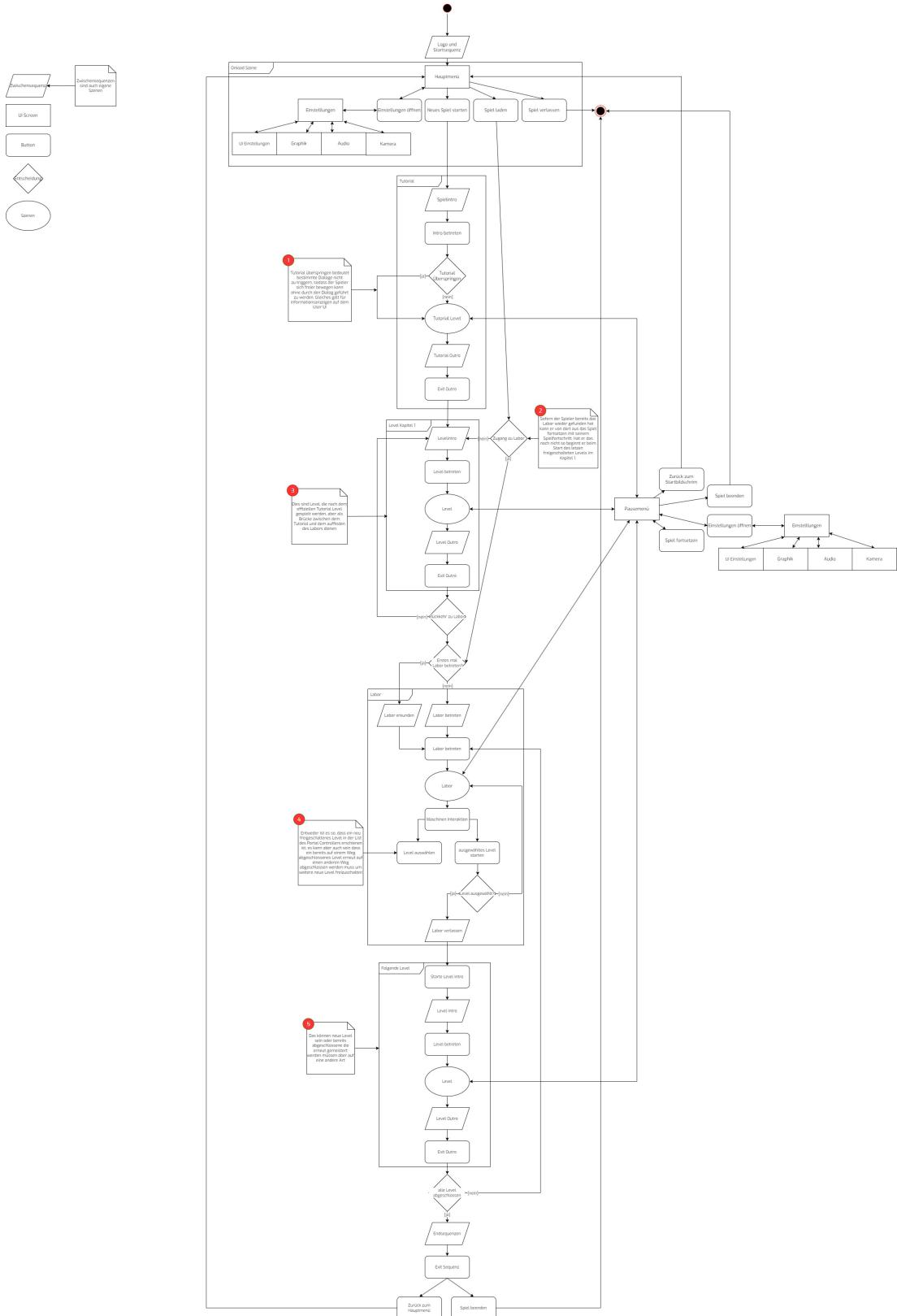


Abbildung 1.: Aktivitätsdiagramm des gesamten Spiels, (Quelle: eigene Darstellung),
(In groß im Anhang A 1.3.1)

der Maschine war, sondern in einem Zwischenraum der unterschiedlichen Zeitlinien des Kontinuums. Aus dem Labor aus kann der Spieler nun die bereits abgeschlossenen Level erneut spielen und kann diese auf demselben Weg abschließen wie bislang, oder er kann durch das Besitzen weiterer verfügbaren Zeitlinien das Level auch durch einen anderen Weg freischalten, oder bislang unerreichbare Gegenstände einsammeln. Dadurch können neue Level freigeschaltet werden, um in der Spielhandlung voranzuschreiten. Über den Computer des Chronologen im Labor kann der Spieler seine Auswahl treffen und reist fortan in ausgewählte Zeitlinien. Auf diese Weise kann der Spieler nun von Spielwelt zu Spielwelt reisen, bis er das Ende der Story erreicht. Nach Beendigung des Levels kann er das gesamte Spiel neu starten oder an dem Spielstand nach dem letzten Level fortführen.

3.4.2. Level Ablauf

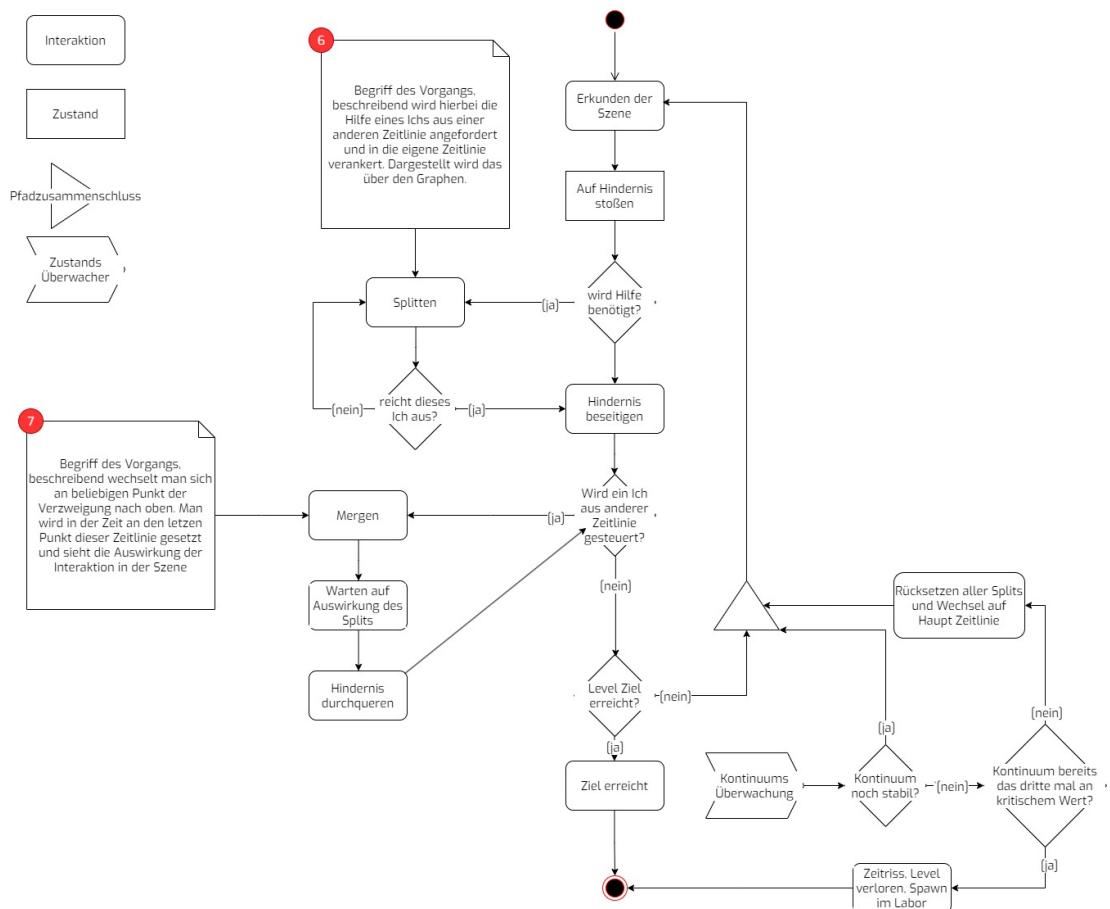


Abbildung 2.: Aktivitätsdiagramm eines Levels, (Quelle: eigene Darstellung)

Sobald das Level geladen ist, wird der Spieler auf den Startpunkt des Levels gesetzt. Von dort aus steuert der Spieler den Chronologen in der Spielwelt, bis er auf ein Hindernis oder ein Rätsel trifft. Ab diesem Zeitpunkt muss der Spieler ein Ich aus einer anderen Zeitlinie über die Steuereinheit auswählen. Dadurch wird die Zeitlinie

dieses Ichs an die Zeitlinie des Chronologen gehängt und der Spieler kann dieses Ich nun steuern. Auf diese Weise kann der Spieler den Mechanismus suchen und finden, welcher das Rätsel löst oder das Hindernis beseitigt. Bspw. muss er auf ein Gerüst klettern, auf welchem sich eine Druckplatte befindet, die eine verschlossene Tür öffnet. Sobald er sich auf die Druckplatte gestellt hat, wird genau zu diesem Zeitpunkt das Hindernis beseitigt. Dann kann er über die Steuereinheit den Chronologen der Hauptzeitlinie auswählen und sieht jetzt, wie die Tür durch die Druckplatte geöffnet wird. Solange das gesamte Level noch nicht absolviert ist und weitere Rätsel bzw. Hindernisse in den Weg gestellt werden, wird der Spieler diese, auf die eben beschriebenen Weise lösen. Es ist möglich, dass der Spieler sein "Jump 'n' Run" Fähigkeiten beweisen muss, allerdings ist das nicht immer der Fall. Der Spieler wird primär damit beschäftigt sein, wie er seine Charaktere in die richtige Reihenfolge "*Splitten*" muss, um die entsprechenden Rätsel zu lösen. Dabei hat er anfangs keine Zeitbegrenzung. Sein einziges Limit wird die Stabilität des Kontinuums sein, das er durch entsprechendes "*Splitting*" und "*Merging*" seiner diversen Chronologen beschädigen wird. Es wird allerdings auch Rätsel geben, an denen der Spieler seine Chronologen auf bestimmten Plattformen zu stellen hat, durch welche bestimmte Muster nachgebildet werden. Fällt die Stabilität des Kontinuums durch zu häufiges und zu langes Splitten dreimal an einen Grenzwert im roten Bereich, so entsteht eine Zeitlinienkonvergenz, durch welchen der Chronologe in der Zeit verloren geht und der Spieler das Level verliert. Nachdem der Spieler das Level neu gestartet hat, erscheint der Chronologe wieder in seinem Labor und der Spieler kann das Level erneut starten. Die Begrifflichkeiten "*Splitting*" und "*Merging*" werden in ihrem entsprechenden Kapitel angesprochen.

3.4.3. Labor Ablauf

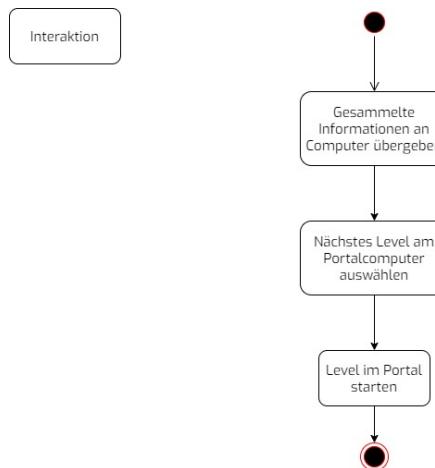


Abbildung 3.: Aktivitätsdiagramm des Labors, (Quelle: eigene Darstellung)

Nach dem Abschließen des ersten Kapitels, erhält der Spieler einen Hinweis, wie der

Chronologe in sein Labor zurückreisen kann. Mit diesen Informationen kann der Spieler zum ersten Mal in sein Labor zurückreisen und von dort aus seine Reisen in andere Zeitlinien starten. Das Labor dient dem Spieler künftig als Hub. Beim erstmaligen Betreten des Labors wird der Chronologe das Labor nach Schäden absuchen. Er stellt dabei fest, dass alles noch so aussieht, wie beim Start der Maschine. Als Erstes überträgt er seine gesammelten Notizen der ersten Level in seinen Computer. Über den Computer wird der Spieler noch mehr Informationen über die Welten erhalten. Der Computer kann aus den erhaltenen Informationen Rückschlüsse über die Welt herführen, die der Spieler auslesen und verwenden kann. Nachdem der Chronologe seine gesammelten Notizen an seinen Computer übertragen hat, kann der Spieler an der Steuereinheit der Maschine im Labor positioniert und das nächste Level auswählt. Im Anschluss kann er das Portal betreten und starten. Auf diese Weise wird der Spieler in weitere Level der verschiedenen Zeitlinien wechseln können. Einige Level sind dabei jeweils in derselben Zeitlinie, sie können manchmal auch vor oder nach dem zuvor gespielten Level in der Zeithierarchie stattfinden. Nach jedem erfolgreich abgeschlossenen Level geht der Chronologe zuerst an seinen Computer und trägt seine Funde in sein System ein. Sobald der Spieler ein Level aufgrund eines zerstörten Zeitkontinuums verliert, wird er wieder im Labor landen, allerdings so als ob sein Versuch niemals stattgefunden hätte.

3.5. Zeitkontinuum

Das Zeitkontinuum ist die überwachende Instanz des gesamten Spiels. Sie misst Auswirkungen und Einflüsse aller parallelen Zeitlinien dieser Spielwelt. Solange die Zeitlinien in sich geschlossen bleiben und nicht versucht wird, miteinander Kontakt aufzunehmen, kann das Kontinuum nicht gefährdet werden. Anders ist es jedoch, wenn versucht wird, die Barriere der Zeitlinien zu durchbrechen und zu erforschen, wie andere Szenarien aussehen würden, wenn andere Entscheidungen getroffen worden wären. Ein dadurch entstehendes Wanken und Beschädigen des Kontinuums hat zur Folge, dass Zeitlinien nicht nur für Einflüsse von anderen Zeitlinien anfällig werden, sondern zusätzlich auch noch für Einflüsse aus der Zukunft oder der Vergangenheit. Durch die Wirkung anderer Zeitlinien steigt die Gefahr der Entstehung von Paradoxen. Das Zeitkontinuum kann eine begrenzte Anzahl von Paradoxen durch ihre Stärke kompensieren. Zu viele Paradoxen sollten sich in den verschiedenen Zeitlinien allerdings nicht bilden, sonst kann es passieren, dass das Kontinuum zerfällt und es zu Konvergenzen der Zeit kommt. Da das Kontinuum ein allumfassendes Konstrukt ist, wird es für den Gebrauch dieses Spiels die Zeitlinie des Chronologen betrachten, der dem Bestreben nachgeht, parallele Zeitlinien zu erforschen. Diese Zeitlinie gilt von nun als Hauptbezugspunkt des Kontinuums. Allerdings bezieht es seine Auswirkungen auf das

Gebiet aller anderen Zeitlinien ebenfalls mit ein, wie die Auswirkungen auf die Zeitlinie des Chronologen. Das Kontinuum überwacht dabei das Entstehen von Paradoxen, die der Chronologe durch das Nutzen seiner Steuereinheit versehentlich erzeugen kann. Zudem misst es, ab welchem Zeitpunkt das Parallelisieren von Zeitlinien auf einer Zeitlinie zu Konvergenzen in der Zeit führen kann.

3.5.1. Zeitlinienkonvergenz

Die Zeitlinienkonvergenz ist eine Katastrophe, die entsteht, wenn die Stabilität des Kontinuums zu häufig auf einen gekennzeichneten Tiefpunkt gerät. Dadurch kann das Kontinuum die verschiedenen Zeitlinien und die Zeit nicht mehr ordnen und alle fallen auf eine Zeitlinie und auf einen Zeitpunkt. Dadurch wird das ganze Kontinuum zerstört. Sobald eine Zeitlinienkonvergenz entsteht, verliert der Spieler das aktuelle Level und wird entweder zurück in das Labor versetzt oder er muss das zuvor gestartete Level neu starten.

3.5.2. Graph

Der Graph ist ein Gerüst in dieser Konzeption und dient als Unterstützung bzw. Visualisierung des Zeitkontinuums, das auf die Zeitlinie des Chronologen gerichtet ist. Er dient visuell der Darstellung der "erzeugten" Zeitlinien, die in abhängiger Verästlung an die Bezugszeitlinie des Kontinuums gehängt wird. Dabei sind diese Zeitlinien im Geflecht eines gesamten Kontinuums nicht "neu" sondern lediglich in diese Zeitlinie integriert, sodass der Chronologe aus dieser Zeitlinie in der aktuellen Zeitlinie interagieren kann. Auf technischer Seite dient der Graph der Strukturierung der Zeitlinien. Durch ihn kann man feststellen, wann und in Bezug auf welche andere Zeitlinie man eine integrierte Zeitlinie nutzen möchte. Den entstehenden Graphen kann der Spieler jederzeit betrachten und inspizieren. Durch eine Zeitskala kann der Spieler sehen, zu welchem Zeitpunkt des Spiels er welche Zeitlinie ausgewählt hat und wie lange diese Zeitlinie integriert wurde. Einen endgültigen Überblick über den Graphen erhält der Spieler nach erfolgreichem Absolvieren eines Levels des Spiels. Dieser Graph kann sich von dem dargestellten Graphen, den der Spieler zur Laufzeit einsehen kann, unterscheiden, da er bereits laufende Zeitlinien unterbrechen oder abbrechen kann. Dazu in Abschnitt "*Zeitlinien löschen*" mehr.

3.5.3. Zeitlinie

Eine Zeitlinie beschreibt den Verlauf eines Lebens einer Person und die Leben der Personen, die in genau dieser Umgebung leben. Jede Entscheidung, die dabei getroffen

wird, hat einen Einfluss auf jedes Leben aller Personen in dieser Zeitlinie. Im Kontext der Konzeption dieses Spiels bildet die Zeitlinie das Leben eines Ichs des Chronologen ab, welches der Spieler später steuern wird. Man kann festhalten, dass der Chronologe vorhat, in das Leben eines anderen Ichs einzutauchen, um zu erfahren, wie das Leben wäre, wenn andere Entscheidungen getroffen worden wären. Dabei sähe der Chronologe ebenfalls eine andere Welt, die andere Ereignisse in der Vergangenheit erlebt hat und auch andere Ereignisse in der Zukunft erleben wird. Im Kontext dieses Spiels werden diese Zeitlinien in der Zeitlinie des Chronologen verankert. Dadurch entsteht der Effekt, dass Zeitlinien und ihre Ichs der Chronologen interaktiv mit der Zeitlinie des Haupt-Chronologen zusammenarbeiten können. Infolgedessen kann durch diesen Ansatz und dieses Verständnis auf der technischen Entwicklungs-Seite eine Abhängigkeit zwischen Zeitlinien erzeugen, auf welchen die gesteuerten Ichs des Chronologen abgebildet werden. Jede Zeitlinie bleibt in sich geschlossen und eigenständig, allerdings kann durch ein Konstrukt wie den Graphen eine Abhängigkeit voneinander erzeugt werden, welche für eine spätere Umsetzung notwendig ist.

3.5.3.1. Splitting

“Splitting” bedeutet in dem Kontext dieses Spiel das Erzeugen einer neuen Zeitlinie, auf der ein Ich des Chronologen lebt und gesteuert werden kann. Diese Zeitlinie wird an die bestehende Zeitlinie als Sublinie angehängt, wodurch eine Abhängigkeit zwischen den Zeitlinien entsteht.



Abbildung 4.: Neu entstandene Zeitlinie nach Split, (Quelle: eigene Darstellung)

Diese neu erzeugte Zeitlinie wird nun an die bisherige Zeitlinie angehängt. Dabei ist diese Zeitlinie keinesfalls neu, sondern lediglich die bislang bereits bestehende Zeitlinie eines Ichs des Chronologen, die in die Zeitlinie des Chronologen gelegt wird. Diese Zeitlinie besteht so lange, wie der Spieler das Ich des Chronologen steuert. Durch mehrmaliges Splitten ergibt sich solch ein Schaubild: Wie in “Abbildung 6” zu



Abbildung 5.: derzeitige Zeitlinie, (Quelle: eigene Darstellung)

sehen ist. Durch das Konstrukt des Graphen ist es nun möglich, diese angegliederten Zeitlinien auf technischer und visueller Ebene darzustellen und zu nutzen. Nach einem “Split-Vorgang” werden die Bewegungen und Interaktionen des Ichs des Chronologen

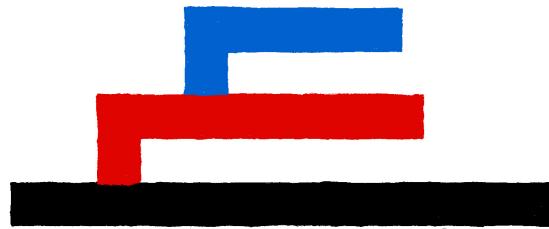


Abbildung 6.: Entstehender Baum (Graph) mit Verzweigungen (Zeitlinien), (Quelle: eigene Darstellung)

aufgenommen, sodass diese nach einem “*Merge-Vorgang*” abgespielt werden können und einen Einfluss auf die Spielwelt haben. Jede Bewegung und Interaktion bekommen dabei einen eindeutigen Zeitpunkt, um sie später reproduzieren zu können.

3.5.3.2. Merging

“*Merging*” beschreibt den Vorgang des Zurückspringens von einer neu erstellten Zeitlinie auf eine bereits existierende. Daraus folgend werden die aufgenommenen Bewegungen und Interaktionen anhand ihres Zeitpunktes genau reproduziert. Nach jedem

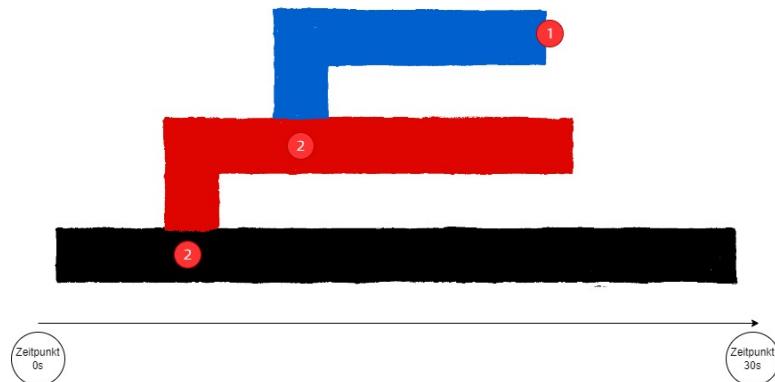


Abbildung 7.: Entstandener Baum mit Knotenpunkten, (Quelle: eigene Darstellung)

“*Split-Vorgang*” wird eine “neue” Zeitlinie an die bislang bestehende angesetzt. Daraufhin entsteht an dem Zeitpunkt, an dem die Zeitlinie ihre Verbindung erhält, ein Knotenpunkt (vgl. Abbildung 7, Markierung 2). Wenn der Spieler sich nach einem “*Split*” an der Markierung 1 in “Abbildung 7” befindet, so kann er für einen, “*Merge-Vorgang*” das Ich aus der roten oder der schwarzen Zeitlinie auswählen. Aus der Sicht der Spielzeit wird der Spieler jetzt an den Punkt zurückgesetzt, an denen einer der Knotenpunkte (Markierung mit der Zahl zwei) liegt. Wählt er beispielhaft den Knotenpunkt auf der schwarzen Zeitlinie aus, so wird zum selben Zeitpunkt die rote Zeitlinie aktiv und das Ich der roten Zeitlinie erscheint in der Szene. Nach einer bestimmten Zeit wird die blaue Zeitlinie aktiv und das Ich dieser Zeitlinie wird ebenfalls erscheinen. Diese Knotenpunkte entstehen nur bei einer Verankerung einer weiteren

Zeitlinie an die derzeitige.

3.5.3.3. Paradoxen

Paradoxen können auftreten, indem der Spieler einer aufgezeichneten und bereits in der Welt verankerten Zeitlinie die Zustände der Spielwelt verändert. Das kann zum Beispiel dann passieren, wenn dem "Ich" dieser verankerten Zeitlinie Gegenstände in den Weg gestellt werden, gegen die dieser Charakter nun stoßen wird. Aufgrund der Tatsache, dass dieser Zustand bei der Aufnahme der Zeitlinie nicht gegeben war, entsteht nun ein Paradoxon, das dazu führt, dass die aufgenommene Zeitlinie und die an ihr angehängten Zeitlinien nun aus dem Kontinuum entfernt werden und dem Kontinuum Schaden zuführen. Diese Schäden spiegeln sich in der Veränderung der Welt wider, die in "Kapitel 3.5.7: Einflüsse auf die Spielwelt" näher beschrieben wird. Durch zu viele entstehende Paradoxen kann es auch dazu kommen, dass es zu einer Zeitlinienkonvergenz kommt, durch welche der Spieler das Level erneut starten muss. Ein weiteres Beispiel ist z. B., dass der Spieler während der "*Splitphase*" einen Hebel betätigt, der bislang noch inaktiv war. Nachdem er durch das "*Merging*" an einen Knotenpunkt zurückgesprungen ist, hat er nun die Möglichkeit diesen Hebel ebenfalls zu betätigen. Tut der Spieler dies, bevor das Ich des aufgenommenen Charakters an den Hebel gelangt und hält ihn aktiv, bis dieser ihn aktivieren würde, so würde ein Paradoxon entstehen. Der Zustand des Hebels wurde nun geändert und führt dazu, dass ein inaktiver Hebel nicht mehr aktiviert werden kann. Das folgende Paradoxon würde die Gefahr einer auftretenden Zeitlinienkonvergenz vergrößern.

3.5.3.4. Zeitlinien löschen

Bereits eingegliederte Zeitlinien können nicht erneut eingegliedert werden, solange sie bereits eingegliedert sind. Das bedeutet, würde der Spieler sehen, dass er zu kurz auf einer Druckplatte stehen geblieben ist, so müsste er die Zeitlinie löschen und damit in Kauf nehmen, dass das Kontinuum einen Schaden davontragen werde, um das Ich der Zeitlinie erneut und für eine längere Zeit auf die Druckplatte zu stellen. Gleichwohl kann er das auch machen, wenn er mit der Aufnahme der Zeitlinie nicht zufrieden ist und mehrmals an einer Stelle gestürzt ist und einen Parcours nicht erfolgreich meistern konnte.

3.5.4. Timer

Der Timer zeigt dem Spieler an, wie lange er bereits in dem Level spielt. Er dient als Orientierung, ab welchem Zeitpunkt bis zu welchem Zeitpunkt der Spieler mit einem

Ich aus einer anderen Zeitlinie spielt. So kann er abschätzen, wann die Zeitlinie seines gespeicherten Ichs nicht mehr in der Welt verfügbar ist und wann dementsprechend der Charakter der Zeitlinie nicht mehr bspw. auf einer Druckplatte stehen wird. Der Timer ist diezählende Komponente des Spiels, über die das Zeitkontinuum wacht. Durch den Timer kann nun auch bestimmt werden, wann eine Zeitlinie aus dem Kontinuum an die Zeitlinie des Chronologen angegliedert wird und wann der Spieler durch das "Merging" an einen Knotenpunkt zurückgesprungen ist, also gewissermaßen wieder in die Vergangenheit gereist ist.

3.5.5. Avatar

Der Spieler sieht im Spiel über die Steuereinheit die Namen seiner Ichs aus anderen Zeitlinien. Über eine Ansicht der verschiedenen verfügbaren Zeitlinien erhält er ebenfalls nur die Ansicht des Ichs aus dieser Zeitlinie. Ein Avatar ist in dem Kontext des Spiels der eigentliche Chronologe aus der Zeitlinie des Kontinuums und seine abgewandelten Persönlichkeiten aus den erforschten Zeitlinien. Der Spieler steuert jeweils das Selbst des Chronologen in all seinen Facetten.

3.5.6. Bewegungen und Interaktionen

Der Spieler wird durch den Chronologen die Spielwelt erkunden. Dabei kann der Chronologe mit Gegenständen wie z.B. Hebeln interagieren. Er kann Gegenstände tragen, er kann an bestimmten Stellen auf Gegenstände klettern und er kann sie verschieben. Zudem ist es ihm möglich auf Objekte zu stehen und diese dadurch zu aktivieren. Außerdem kann er, über kleinere Abgründe hinwegspringen und rennen. All diese Bewegungs- und Interaktionsmöglichkeiten werden auf der Zeitlinie des ausgewählten Chronologen gespeichert und mit einem exakten Zeitpunkt versehen, damit diese chronologisch wieder rekonstruiert werden können. Steuert der Spieler den Haupt-Chronologen, so werden seine Bewegungen und Interaktionen nicht aufgenommen.

3.5.7. Einflüsse auf die Spielwelt

Ein instabiles Zeitkontinuum hat direkte Auswirkungen auf die Spielwelt, in der sich der Spieler mit dem Chronologen befindet. Durch die Gegebenheit, dass es viele verschiedene Zeitlinien gibt, die parallel verlaufen, existieren auch viele verschiedene Version des Raumes bzw. des Ortes, an dem sich der Spieler jetzt gerade in dem Moment befindet. Durch ein instabiles Zeitkontinuum sind die Grenzen der in sich geschlossenen Zeitlinien brüchig und können auf andere Zeitlinien übergehen. Das hat zur Folge, dass Gegenstände oder gar Wandelemente plötzlich in der Spielwelt auftauchen und

manche Bereiche des Levels nicht mehr betreten werden können, da an dieser Stelle plötzlich eine Wand erscheint. Es kann auch dazu führen, dass Elemente aus der Spielwelt verschwinden und plötzlich in einer anderen Zeitlinie in einer anderen Umgebung wieder auftauchen. Der Spieler muss also Sorge tragen, dass durch sein Handeln die Spielwelt nicht beschädigt wird. Durch eine Beschädigung würde der Spieler ein Level verlieren oder ihm würden unüberwindbare Hindernisse in den Weg gestellt werden. Er muss warten, bis sich das Kontinuum wieder erholt hat. Der Spieler wird das Erholen des Zeitkontinuums anhand einer Anzeige auf seinem UI sehen. Dies wird im “Kapitel 4.4.3: Zeitkontinuum” visualisiert.

Unabhängig von der Instabilität des Kontinuums kann sich die Welt aus der Perspektive der anderen Ichs des Chronologen auch verändern. Jedes Ich ist in seiner Zeitlinie unterschiedlich zu anderen Zeitlinien. Das bedeutet auch, dass diese Ichs bestimmte Dinge auf eine andere Art wahrnehmen oder andere Dinge sehen können. Zum Beispiel können sich bestimmte Dekorationsobjekte unterscheiden oder Teile der Wände sind verändert. So wird sich das Leveledesign aus Sicht der Ichs des Chronologen teilweise unterscheiden. Die Kernelemente der Rätsel bleiben jedoch aus jeder Sicht der Ichs identisch. So wird einer Verwirrung des Spielers vorgebeugt.

3.6. Belohnungen

Nach Abschließen eines Levels, erhält der Spieler das Ich des Chronologen aus dieser Zeitlinie zur Auswahl als Belohnung. Damit hat er für die kommenden und bereits absolvierten Rätsel eine Auswahlmöglichkeit mehr, seine Ichs anzurufen. Das ermöglicht dem Spieler, die Level über ein anderes Ziel abzuschließen.

3.7. Labor

Nachdem der Spieler nach Abschließen des ersten Kapitels der Story das Labor wieder entdeckt hat, kann er von dort aus bereits abgeschlossene Level erneut starten und zu neuen Levels reisen. Außerdem kann der Chronologe seine Erkenntnisse aus den abgeschlossenen Level hier abspeichern und erfährt dadurch mehr über die Spielwelten, in welche er reist.

3.8. Aufnahmen des Chronologen

Im Kontext dieses Spiels unterscheiden wir zwischen zwei Arten von “None-Playable” Charakteren, die ein Abbild des Chronologen darstellen, aber nicht vom Spieler gesteuert werden. Es wird im Sprachgebrauch von Platzhaltern und Geistern gesprochen.

3.8.1. Platzhalter

Sobald der Spieler in ein anderes Ich des Chronologen springt, wird an der letzten Stelle, an dem sich der bisherige Spielercharakter befand, ein Platzhalter gesetzt. Dieser Platzhalter dient dazu, zu zeigen, dass sich dieses Ich des Chronologen in der Zeit noch nicht fortbewegt hat, genauer gesagt zu diesen Zeitpunkten, die der Spieler nach dem „*Splitting*“ erlebt, noch nicht erlebt hat. Der Chronologe, der vor dem „*Mergeprozess*“ noch ein Platzhalter war, wird nach dem „*Mergeprozess*“ diese kommenden Zeitpunkte erleben. Er besitzt eine durchscheinend transparente Gestalt, welche dem Spieler vermittelt, er würde zum einen an dieser Stelle wieder beginnen und zum anderen auch anzeigen, dass dies kein paralleler Chronologe ist.

3.8.2. Geister

Nach einem „*Merge-Prozess*“ wird der Chronologe der gespielten Zeitlinie in der Spielwelt verankert. Er besitzt das Aussehen des Ichs des Chronologen, das der Spieler bereits beim Spielen des Ichs gesehen hat. Dies zeigt an, dass das Ich des Chronologen, den der Spieler gesteuert hat, die aufgenommenen Interaktionen und Bewegungen ausführt. Er ist die kooperative Komponente des Spiels. Auf das Verhalten dieses Ichs kann der Spieler nun reagieren, um die laufenden Rätsel zu lösen. Alle 3D Modelle des Chronologen, die nicht transparent sind, haben einen direkten Einfluss auf die Spielwelt und das Kontinuum.

3.9. Gebrauchsgegenstände

Zu Beginn des Spiels besitzt der Chronologe zwei Gebrauchsgegenstände, die auf die anderen Ichs des Chronologen übertragen werden können. Durch diese kann der Spieler seine im Labor befindliche Maschine ferngesteuert bedienen und er kann seine notierten Erkenntnisse und Notizen über die Welt notieren.

3.9.1. Steuereinheit

Seine Erfindung im Labor besitzt eine Steuereinheit. Sie ist Teil seines Computers, der ebenfalls im Labor steht. Um seine Maschine auch außerhalb des Levels zu bedienen, hat er zusätzlich eine mobile Steuereinheit entwickelt. Durch diese kann der Spieler die verschiedenen Versionen des Chronologen überblicken und auswählen. Er erhält dadurch eine Übersicht über die Namen und den derzeitigen Status in der Szene der Ichs.

3.9.2. Datenleser

Über den Datenleser kann der Chronologe seine Erkenntnisse und Notizen sammeln. Der Spieler kann diesen Datenleser öffnen, um über ihn Informationen über die Welt und die einzelnen Zeitlinien zu bekommen. Zusätzlich werden wichtige Abläufe und Informationen des Spiels darüber einsehbar. Der Spieler kann ihn, als Journal nutzen, um Wissen über das Spiel zu sammeln.

Im Laufe der Handlung des Spiels und beim wachsenden Erkundungsfortschritt der verschiedenen Zeitlinien erhält der Chronologe weitere Gegenstände, die ihm, als Hilfsmittel zur Lösung der Rätsel dienen. Durch den Erhalt von neuen Gebrauchsgegenständen steigt die Anforderung des Spiels an den Spieler. Er muss diese Gegenstände nun zielführend einsetzen, um die Rätsel lösen zu können. Er kann dadurch seinen Spielfluss verbessern, allerdings wird seine Umgebung auch herausfordernder (vgl. (Schell 2020, S. 290ff)).

3.9.3. Imitator

Der Chronologe findet im Laufe des Spiels die Auswirkungen der Steuereinheit auf den Ablauf immer besser kennen. Daher beschließt er, mit der Hilfe und den Informationen aus den Ichs der anderen Zeitlinien einen Imitator zu bauen, der begrenzt nutzbar einen Platzhalter auf eine Druckplatte oder Ähnliches stellt. Dadurch muss der Spieler keinen „*Split*“ ausführen und auf der Druckplatte warten, bis sein Ich nach einem „*Merge*“ über den daraus resultierenden Freiraum gegangen ist. Wie bereits erwähnt, führen viele „*Splits*“ und „*Merges*“ zu einer Instabilität des Kontinuums. Dieser Imitator soll das verhindern, er kann aber nicht so oft genutzt werden. Er soll lediglich an den Stellen nützlich sein, an denen es zu einer Knappheit der verfügbaren Ichs in der Steuereinheit kommt oder die Gefahr besteht, das Level zu verlieren.

Durch den Imitator kann der Spieler seine Auswirkungen auf das Kontinuum schonen. Zusätzlich kann er durch den ihn genau wissen, wie lang eine Kopie seiner selbst an diesem Punkt steht.

3.9.4. Pager

Einen Pager findet der Spieler auch zu einem bestimmten Zeitpunkt des Spiels gegen Ende der Story. Zu diesem Zeitpunkt sind die Rätsel komplexer geworden und der Spieler muss seine „*Splits*“ und „*Merges*“ nun genau abstimmen, damit zum richtigen Zeitpunkt die richtigen Auswirkungen auf die Spielwelt ausgeführt werden können. Ein Pager hilft ihm, seinem zu steuernden Chronologen nach einem „*Merge*“ Bescheid zu geben, wann bspw. ein Hebel umgelegt wird. Der Spieler erhält dabei eine Nachricht,

die wie folgt aussehen kann: “[Chronologe Ich Variante] legt jetzt Hebel um”. Durch diese erhaltene Nachricht weiß er nun, wann der Hebel umgelegt wird, und kann darauf reagieren. Sofern der derzeitig gesteuerte Chronologe nicht der ursprüngliche Chronologe ist, kann der Spieler nun auch weiteren zu steuernden Chronologen eine Nachricht senden. Jede Ich-Variante des Chronologen kann also sowohl Nachrichten empfangen als auch Nachrichten versenden. Der ursprüngliche Chronologe kann jedoch nur Nachrichten empfangen, um darauf zu reagieren.

3.10. Leveldesign

Ein großer Teil des Spiels beruht auf dem Leveldesign der einzelnen Schauplätze, in denen die verschiedenen Rätsel integriert sind. Im Folgenden werden die bisher umgesetzten Rätsel und ihre Lösungen aufgezählt und vorgestellt. Zudem wird es einen Ausblick darauf geben, welche weiteren Rätsel bereits konzipiert und noch nicht gelöst wurden. Des Weiteren wird beschrieben, welche weiteren Rätsel für das Spiel möglich wären.

In einem weiteren Teil werden die Objekte, mit denen interagiert werden kann, beschrieben. Diese werden im Folgenden aufgelistet. Im Abschnitt 4 dieser Thesis werden diese Objekte visuell dargestellt.

3.10.1. Interaktionsobjekte

In den Spielwelten des Spiels gibt es verschiedene Objekte, mit denen der Chronologe interagieren kann. Im Folgenden werden diese Objekte konzeptionell vorgestellt.

3.10.1.1. Druckplatte

Druckplatten befinden sich auf dem Boden und können durch das Körpergewicht des Chronologen ausgelöst werden. Nachdem Druckplatten ausgelöst wurden, aktivieren diese weiteren Objekte wie Türen, Kräne oder Plattformen an der Wand. Verlässt der Spieler die Druckplatte, so deaktiviert sie sich und ihre verknüpften Objekte. Eine Druckplatte bleibt nur aktiv, sobald ein Ich des Chronologen auf ihr steht.

3.10.1.2. Hebel

Hebel können sich auf dem Boden oder an Wänden befinden. Der Spieler kann ebenfalls mit diesem interagieren. Der Chronologe drückt den Hebel-Stab auf seine aktive Position. Diese muss der Chronologe halten. Solange der Hebel auf der Aktivposition des Hebels steht, ist der Hebel wirksam. Dadurch können Lampen angestellt, Tore

geöffnet oder Plattformen aktiviert werden. Entfernt sich der Spieler aber zu weit vom Hebel, deaktiviert sich dieser und seine verknüpften Objekte werden ebenfalls deaktiviert. Zum Beispiel schließt sich eine Tür oder ausgefahrenen Plattformen fahren wieder ein.

3.10.1.3. Ventil

Ventile befinden sich an der Wand oder an Rohren. Sie dienen dazu, um entweder Rohrflüsse zu aktivieren oder um bewegliche Plattformen zu verschieben. Durch einmalige Interaktion mit einem Ventil ist sowohl der Rohrfluss geöffnet oder geschlossen als auch die Plattform an ihrem entsprechenden Zielort.

3.10.1.4. Kletterkanten

Der Chronologe kann auf höher gelegene Objekte klettern. Diese Objekte, wie Plattformen oder Stützen, besitzen eine greifbare Kante, an der sich der Chronologe hochziehen kann.

3.10.1.5. Tragbare Objekte

Dem Chronologen ist es möglich, bestimmte Objekte zu tragen. Dadurch kann er Klettermöglichkeiten für einen anderen Spieler erzeugen, oder für sich selbst Vorrangstypen bauen, um über Abgründe hinwegzuspringen.

3.10.1.6. Portal Steuereinheit

Im Labor befindet sich eine stationäre Steuereinheit seiner Maschine. Diese dient dazu, das nächste Ziel oder das nächste Level in einer Zeitlinie auszuwählen. Aktiviert er nun die Maschine in einem geschaffenen Portal, so wird er an das Ziel befördert.

3.10.1.7. Objekte zum Schieben

Der Chronologe kann nicht nur bestimmte Objekte tragen, sondern auch schieben. Dadurch kann er etwa Wege freiräumen oder vorteilhaftere Positionen schaffen, von denen der Chronologe auf höher gelegene Objekte klettern kann.

3.10.2. Rätsel

Die Rätsel der einzelnen Spielwelten können variieren. Es gibt Abschnitte, in denen durch Betätigen einzelner Abschnitte andere Abschnitte freigelegt werden. Es kann

aber auch sein, dass durch das Anwenden eines “*Split*”- und “*Merge*”-Vorganges der richtige Weg ans Ziel gefunden werden muss. Oder aber es ist notwendig, zu bestimmten Zeitpunkten, mit den richtigen Ichs des Chronologen an bestimmten Stellen zu stehen, um Mechanismen auszulösen. Anschließend wird das Rätseldesign vorgestellt. Dabei wird zwischen bereits umgesetzten und konzipierten Rätseln unterschieden.

Um die Rätsel zu konzipieren, wurde zunächst überlegt, welche Interaktionsgegenstände weitere Weltgegenstände aktivieren oder deaktivieren können. Diese zu aktivierenden Gegenstände sollten ebenso wie die Interaktionsgegenstände in das Setting der Spielwelt passen. Nachdem eine Auswahl an geeigneten Gegenständen getroffen war, wurden die einzelnen Räume der Spielwelt gestaltet. Die eingebauten Rätsel sollen nun sukzessiv den Weg zu einem Zwischenziel oder dem gesamten Ziel freischalten, sodass der Spieler mit dem Haupt-Chronologen immer näher an das Ziel gelangt. Dabei wurde versucht, die Schwierigkeit in Form von verschieden gleichzeitigen Aktivitäten zu steigern. Zum einen müssen immer mehr Ichs des Chronologen verwendet werden, um zum anderen müssen immer mehr Objekte bewegt oder aktiviert werden. Dabei stößt der Spieler jederzeit auf bereits erlernte Mechanismen, die er einsetzen und mit neuen Mechanismen der Rätsel kombinieren muss.

Um die Schwierigkeit steigern zu können, wurde anhand von Skizzen überlegt, welche Anreihung an “*Splits*” und “*Merges*” der Spieler tätigen kann, um bestimmte Abschnitte zu lösen. Diese Skizzen wurden in der Konzeption auf maximal vier weitere Ichs Chronologen angelegt. In “Abbildung 8” ist ein Überlegungskonzept für den zweiten Raum angefertigt worden, durch welchen die Rätsel entstanden sind. Es beschreibt den dabei entstehenden Graphen.

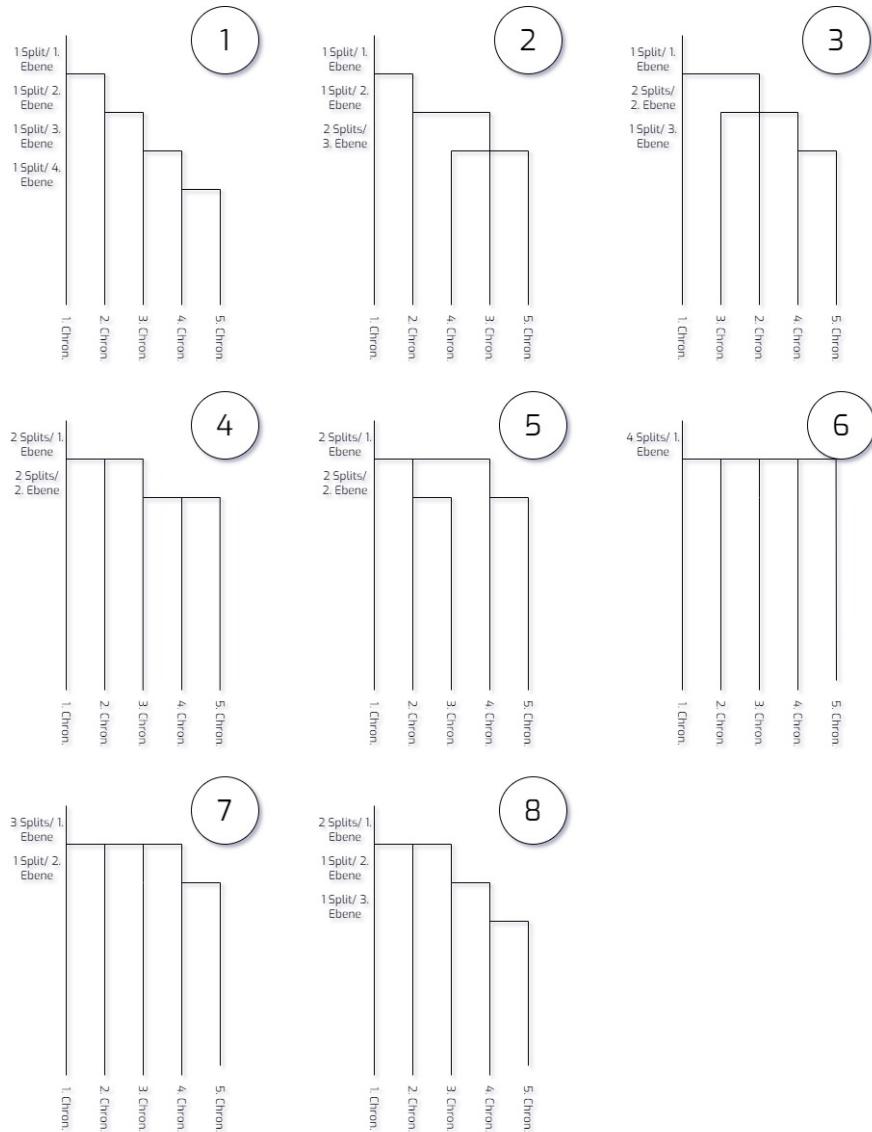


Abbildung 8.: verschiedene Anreihungen an Zeitlinien, nummeriert, (Quelle: eigene Darstellung)

3.10.2.1. Umgesetzte Rätsel

Im Folgenden werden nun die umgesetzten Rätsel mit ihrer jeweiligen Lösungsmethode vorgestellt.

Im Tutorial hat der Spieler zwei weitere Ichs des Chronologen zur Auswahl. Diese drei verschiedenen Zeitlinien haben eine jeweils eindeutige Farbe.

Tutorial, Raum 1: Zu Beginn trifft der Spieler auf eine verschlossene Tür (siehe Abbildung 9). Über die Druckplatte auf dem Gerüst kann der Spieler die Tür öffnen. Dazu muss er auf das Gerüst klettern, was ohne Probleme funktioniert. Um final mit dem Chronologen durch die Tür zu gelangen, muss der Spieler ein weiteres Ich des



Abbildung 9.: Rätsel im ersten Tutorial Raum, (Quelle: eigene Darstellung)

Chronologen zur Hilfe nehmen, der daraus resultierende Graph in “Abbildung 10” hat eine solche Form:



Abbildung 10.: Lösung zu Rätsel im ersten Tutorial Raum, (Quelle: eigene Darstellung)

Tutorial, Raum 2, Element 1: Durch eine kleine Öffnung in der Wand kann der Spieler die eingefahrenen Plattformen an der Wand aktivieren. Er muss allerdings am Hebel stehen bleiben, damit dieser nicht wieder einfährt und die Plattformen einfahren lässt (siehe “Abbildung 11”).

Der Spieler benötigt also erneut eine Ich-Version des Chronisten, die den Hebel aktiv hält. Daraus ergibt der Graph aus “Abbildung 10”

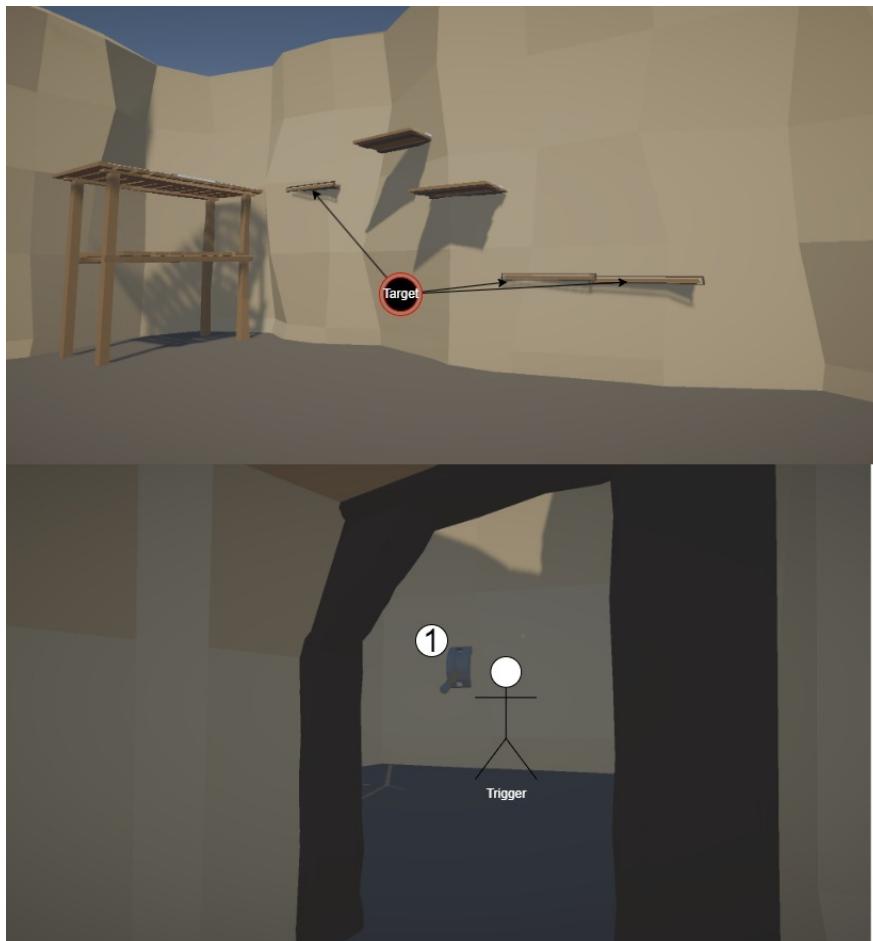


Abbildung 11.: Element 1 im zweiten Tutorial Raum, (Quelle: eigene Darstellung)



Abbildung 12.: Graph zu Element 1 im zweiten Raum, (Quelle: eigene Darstellung)

Tutorial, Raum 2, Element 2: Nachdem der Spieler auf das Gerüst geklettert ist, kann er über die Druckplatte die Zieltür öffnen. Durch diese Tür gelangt der Spieler in den nächsten Raum (siehe „Abbildung 13“). Daraus ergibt sich folgender Graph in „Abbildung 14“.

Betrachtet man die Elemente des zweiten Raumes als Ganzes, so kann man ihn auf 2 Wege lösen. Zunächst „splittet“ der Spieler entweder in das blaue oder rote Ich des Chronologen. Anschließend „splittet“ er sich in das verbleibende Ich des Chronologen. Mit dem aktuellen Ich betätigt der Spieler den Hebel. Hat der Spieler den Hebel lange genug umgelegt, „mergt“ er auf das Ich vom ersten „Split“ zurück. Mit diesem klettert er die Plattformen an der Wand hoch und aktiviert die Druckplatte. Anschließend „mergt“ der Spieler auf den ursprünglichen Chronologen zurück. Mit diesem kann er,

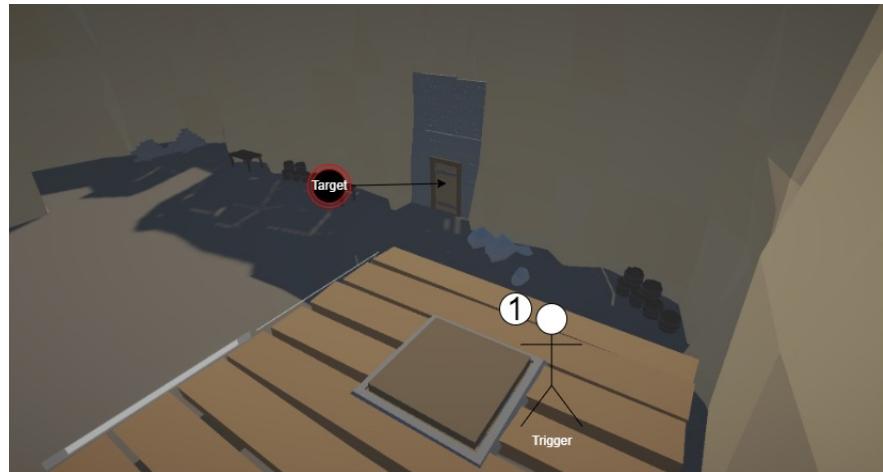


Abbildung 13.: Element 2 im zweiten Tutorial Raum, (Quelle: eigene Darstellung)



Abbildung 14.: Lösung Element 2 im zweiten Tutorial Raum, (Quelle: eigene Darstellung)

sobald das Ich aus dem ersten “*Split*” auf der Druckplatte steht, durch die Tür in den nächsten Raum. Daraus resultiert der Graph aus “Abbildung 15”.

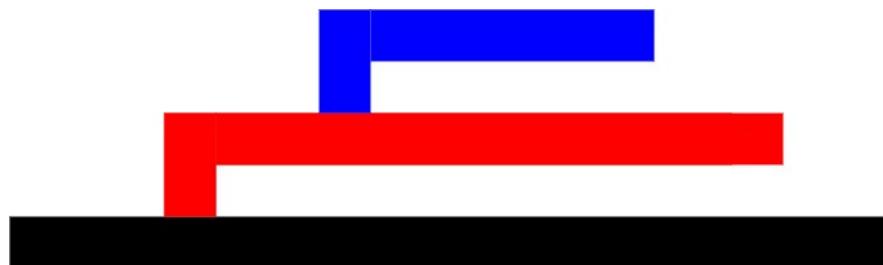


Abbildung 15.: Lösung Variante 1 im zweiten Tutorial Raum, (Quelle: eigene Darstellung)

Eine andere Möglichkeit wäre es, anstatt des Zweiten “*Splits*” vom Roten auf das blaue Ich, ein “*Merge*” zurück auf den Chronologen. Anschließend “*splittet*” der Spieler auf das verbleibende ich. Nach dem Ersten “*Split*” aktiviert der Spieler den Hebel, nach dem Zweiten “*Split*” nachdem er zurück “*gemergt*” hat, klettert der Spieler auf das Gerüst und öffnet die Tür. Hat der Spieler lange genug gewartet, so kann er auf den Chronologen zurück “*mergen*” um in den nächsten Raum zu gelangen. Der dargestellte Graph ist in “Abbildung 16” ersichtlich.

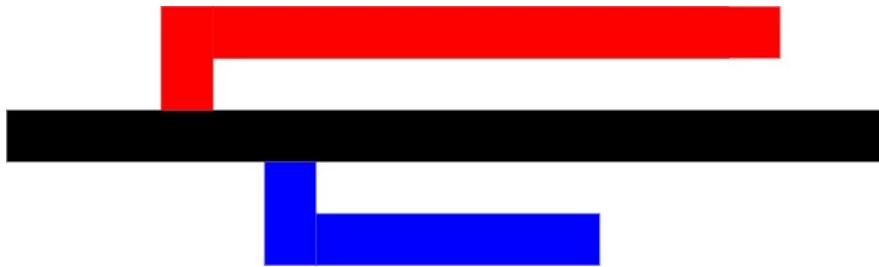


Abbildung 16.: Lösung Variante 2 im zweiten Tutorial Raum, (Quelle: eigene Darstellung)

Tutorial, Raum 3, Element 1: Über die Druckplatte, mit der Kennnummer 1 in „Abbildung 17“ markiert, kann der Spieler die Plattformen an der Wand aktivieren. Über die neu ausgefahrenen Plattformen kann der Spieler mithilfe der statischen Plattformen auf das Gerüst am Ende des Raumes klettern. Für diesen betrachteten Einzelfall ergibt sich derselbe Graph, wie er bereits in „Abbildung 14“ dargestellt wird.



Abbildung 17.: Element 1 im dritten Tutorial Raum, (Quelle: eigene Darstellung)

Tutorial, Raum 3, Element 2: Damit der Spieler das Level beenden kann, muss er zuletzt die hochgefahren Brücke überwinden. Über die Druckplatte auf dem Gerüst, auf das er nach erfolgreichem Abschließen von Element 1 kommt, kann der Spieler die Brücke herunterfahren (siehe „Abbildung 18“). Dazu benötigt er eine weitere Zeitlinie als Hilfsmittel. Daraus ergibt sich ein Graph, der aussehen kann, wie in „Abbildung 12 oder 14“.

Es ist möglich, diesen dritten Raum ebenfalls als Ganzes zu betrachten. Dabei kann sich der Spieler mit dem Chronologen direkt vor der Brücke positionieren oder an der Stelle, bevor er auf das erste Gerüst in „Abbildung 17“ springt. Nun „splittet“ der Spieler in eines der möglichen Ichs aus einer anderen Zeitlinie. Er wiederholt den Vorgang erneut und geht nun mit diesem zweiten Ich auf die Druckplatte. Dort wartet er so lange, wie er glaubt, dass er über die Plattformen auf das Gerüst bräuchte.

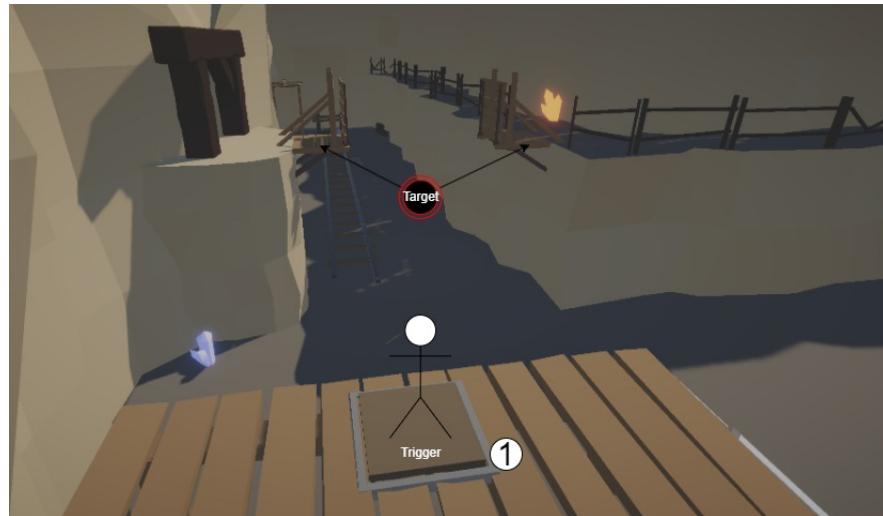


Abbildung 18.: Element 2 im dritten Tutorial Raum, (Quelle: eigene Darstellung)

Nachdem er gewartet hat, „*mergt*“ er auf das als erstes „*abgesplitteten*“ Ich des Chronologen zurück. Mit diesem kann er, sobald die Plattformen aktiviert wurde, über die Plattform auf das Gerüst klettern und die Druckplatte für die Brücke aktivieren. Der Spieler „*mergt*“ nun jetzt auf den Chronologen zurück und kann über die aktivierte Brücke springen. Zudem kann er das Tutorial beenden. Aus diesem beschriebenen Ablauf ergibt sich folgender Graph, der bereits in „Abbildung 15“ dargestellt wurde.

Hierbei ist es ebenfalls möglich, den Raum im Ganzen auf eine zweite Art zu lösen. Nachdem der Spieler das erste Mal auf sein Ich des Chronologen „*gesplittet*“ ist und anschließend die Druckplatte von Element 1 aktiv hält, kann er wieder zurück auf den Chronologen „*mergen*“. Anschließend kann der Spieler auf das verbleibende Ich des Chronologen „*splitten*“. Sobald das bereits „*gesplittete*“ Ich auf der Druckplatte steht, kann der Spieler mit dem derzeitig „*gesplitteten*“ Ich über die Plattformen auf die Druckplatte in Element 2 steigen. Nach einem „*Merge*“ zurück auf den Chronologen kann der Spieler nun über die Brücke gehen. Daraus ergibt sich der Graph, der bereits in „Abbildung 16“ dargestellt wurde.

Im zweiten Level dieser Prototypen hat der Spieler zwei weitere Ichs des Chronologen zur Auswahl. Das zweite Level wird Labor genannt und hat einen fertig bestehenden Raum. Dieser wird im Kontext dieser Aufzählung Stromkammer genannt.

Labor, Stromkammer, Element 1: Folgt der Spieler den Stromkabeln in die Stromkammer, so findet er sich in einer kleinen Kammer wieder, in die er zunächst eingesperrt wird. Über eine Druckplatte kann der Spieler Plattformen an der Wand aktivieren, durch die der Spieler aus der Kammer in die große Grotte gelangt (vgl. Abbildung 19). Dazu benötigt er ein Ich des Chronologen, in das der Spieler „*split-*

“tet” um im Anschluss auf den Chronologen zurück zu “mergen”. Daraus ergibt sich wiederum ein Graph, der ähnlich zu “Abbildung 12 oder 14” aussehen kann.

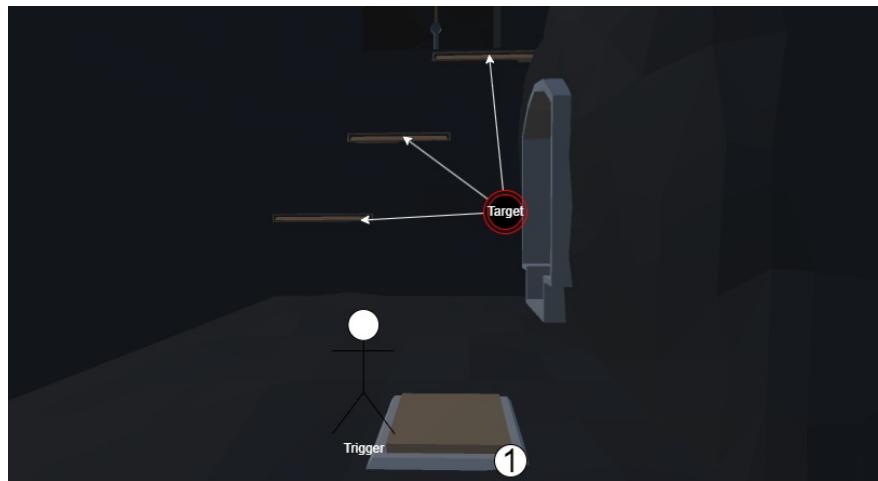


Abbildung 19.: Element 1 in der Stromkammer, (Quelle: eigene Darstellung)

Labor, Stromkammer, Element 2:

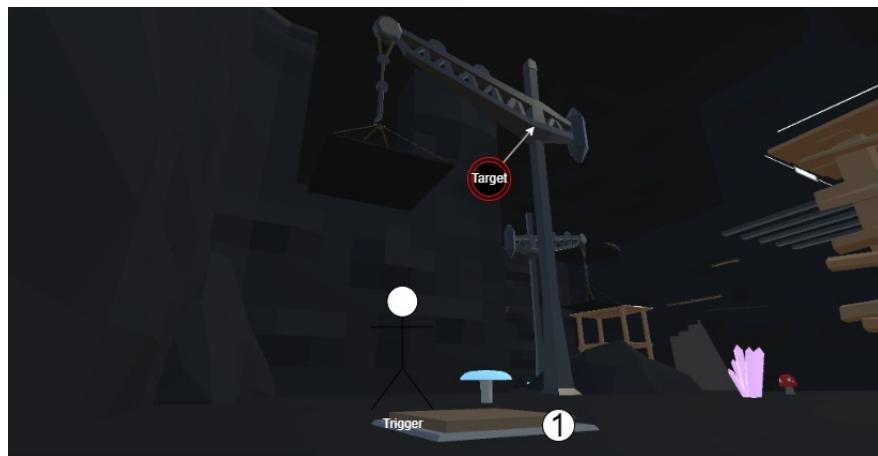


Abbildung 20.: Element 2 in der Stromkammer, (Quelle: eigene Darstellung)

Sobald der Spieler aus dem Eingangsraum herausgefunden hat, kann er mithilfe des Krans erneut wieder in die Kammer gehen. Außerdem kann der Spieler über den ersten Kran mit dem zweiten Kran in Kombination auf das Gerüst am Ende der Grotte gelangen (siehe “Abbildung 20”). Um das zu erreichen, benötigt er allerdings mehrere Komponenten als ausschließlich den Kran. Über die Druckplatte kann der Spieler den Kran aktivieren. Daraus erfolgt derselbe Graph, der ähnlich wie “Abbildung 12 oder 14” aussehen kann.

Labor, Stromkammer, Element 3:

Abbildung 21.: Element 3 in der Stromkammer, (Quelle: eigene Darstellung)

Die Druckplatte auf der Kranplattform aktiviert weitere Plattformen an der Wand. Diese Plattformen sind wichtig, um letztlich an den Stromansteller zu gelangen. Um hierbei nicht mit dem Chronologen die Druckplatte zu aktivieren, wird wiederum ein weiteres Ich des Chronologen benötigt.

Im Folgenden wird kein Abbild des Graphen dargestellt, da die beiden Kräne zu einem Mechanismus gehören, welcher nur im Ganzen gelöst werden kann. Später wird darauf separat eingegangen.

Labor, Stromkammer, Element 4:

Durch die Druckplatte in Labor, Stromkammer, Element 3:, werden an der Wand nun die Plattformen aktiv und der Spieler kann auf eine neu erschienene Druckplatte steigen. Diese Druckplatte aktiviert zum einen den zweiten Kran und zum anderen Druckplatten an der Wand neben dem Hochstand, das ein Zwischenziel darstellt (siehe "Abbildung 22"). Um auf diese Druckplatte zu gelangen, benötigt der Spieler ein weiteres Ich des Chronologen, um mit diesem den zweiten Kran zu aktivieren. Über den zweiten Kran gelangt der Spieler nun auf diesen Hochstand.

Im Anschluss werden die Elemente Labor, Stromkammer, Element 2: bis Labor, Stromkammer, Element 4: aus der Stromkammer benutzt, um den daraus resultierenden Graphen als ein Ganzes zu erklären.

Wie erwähnt muss der Spieler zum einen den Kran in Bewegung setzen, damit er an die Schnittstelle der zwei Kräne gelangt. Der zweite Kran muss nun auch aktiviert



Abbildung 22.: Element 4 in der Stromkammer, (Quelle: eigene Darstellung)

werden, das geschieht zum einen durch die Druckplatte auf dem ersten Kran aus Element Labor, Stromkammer, Element 3: und zum anderen durch die frei werdende Druckplatte in Element Labor, Stromkammer, Element 4:. Der zweite Kran fährt nach der Aktivierung zur Schnittstelle der zwei Kräne und anschließend wieder zurück zum Hochstand. Der Spieler bräuchte für diesen Zwischenschritt insgesamt drei weitere Ichs des Chronologen. Der Graph, der diesen Weg ebnen lässt, sieht folgendermaßen aus. Er wird in "Abbildung 23" gezeigt.

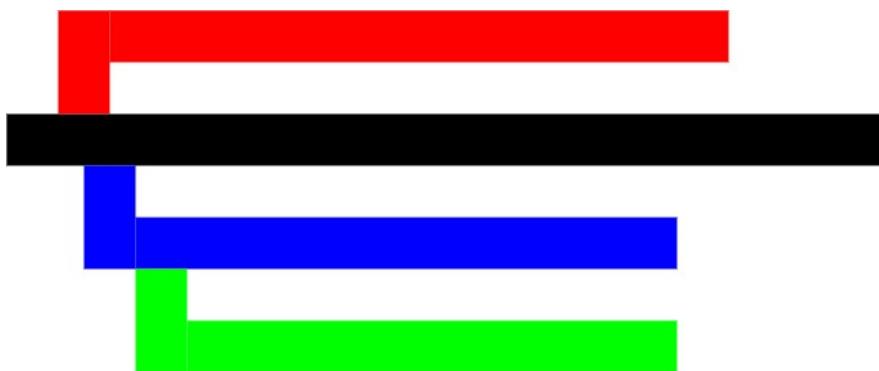


Abbildung 23.: Lösungsweg gesamt Element 2 bis 4, (Quelle: eigene Darstellung)

Die rote Zeitlinie repräsentiert das Ich des Chronologen, welches auf die Druckplatte steigt, welche den Kran in Bewegung setzt. Er muss die Druckplatte so lange aktiv halten, bis der Spieler vom ersten auf den zweiten Kran gesprungen ist. Der Spieler "*mergt*" in den Chronologen zurück, um auf den zweiten Kran zu gelangen.

Anschließend „splittet“ der Spieler in die blaue Zeitlinie, von dort aus in die Grüne. Mit dem Chronologen der grünen Zeitlinie stellt sich der Spieler auf die Druckplatte der Kranplattform. Diese öffnet die Plattformen an der Wand von „Abbildung 21“. Nachdem er lange genug gewartet hat, „mergt“ der Spieler zurück auf den Chronologen der blauen Zeitlinie. Mit diesem Ich des Chronologen steigt der Spieler nun auf die geöffnete Druckplatte aus „Abbildung 21“. Dadurch setzt sich der zweite Kran in Bewegung und fährt zur Schnittstelle und zurück. Anschließend „mergt“ der Spieler auf den eigentlichen Chronologen zurück und springt über den aktivierte zweiten Kran auf das Gerüst um das Zwischenziel zu erreichen.

Labor, Stromkammer, Element 5:

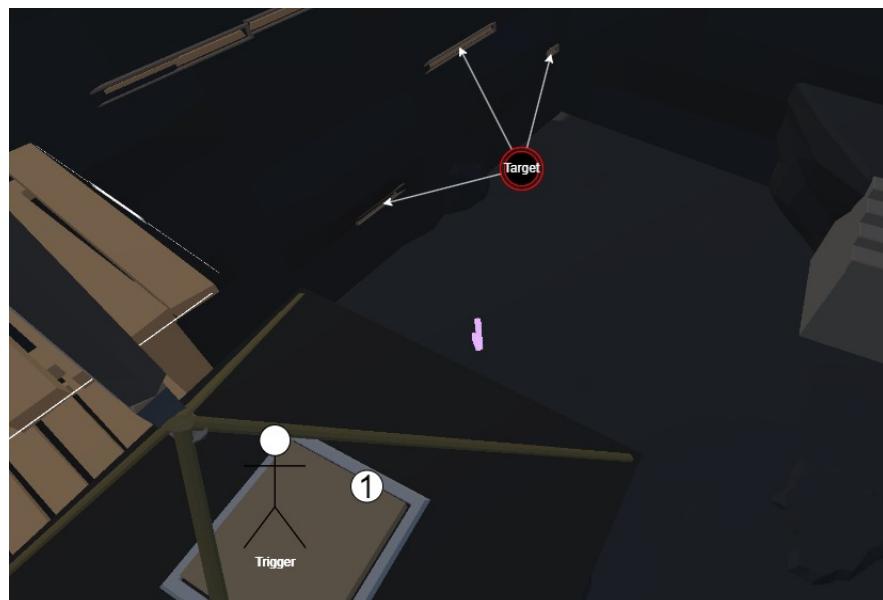


Abbildung 24.: Element 5 in der Stromkammer, (Quelle: eigene Darstellung)

Wie in „Abbildung 22“ bereits gezeigt, öffnet die Druckplatte auf dem zweiten Kran die unteren drei Plattformen, die in „Abbildung 24“ gekennzeichnet sind. Eine der ausfahrbaren Plattformen enthält eine Druckplatte, welche das letzte Stück des Weges aktiviert. Hierfür wird wieder ein Ich des Chronologen benötigt. Möglich ist es auch, dass erzeugten Zeitlinien von oben noch aktiv sind, dann wären die Plattformen bereits ausgefahren. Wenn man nicht von diesem Fall ausgeht, sieht der Graph aus wie in „Abbildung 12“ oder 14 bereits dargestellt.

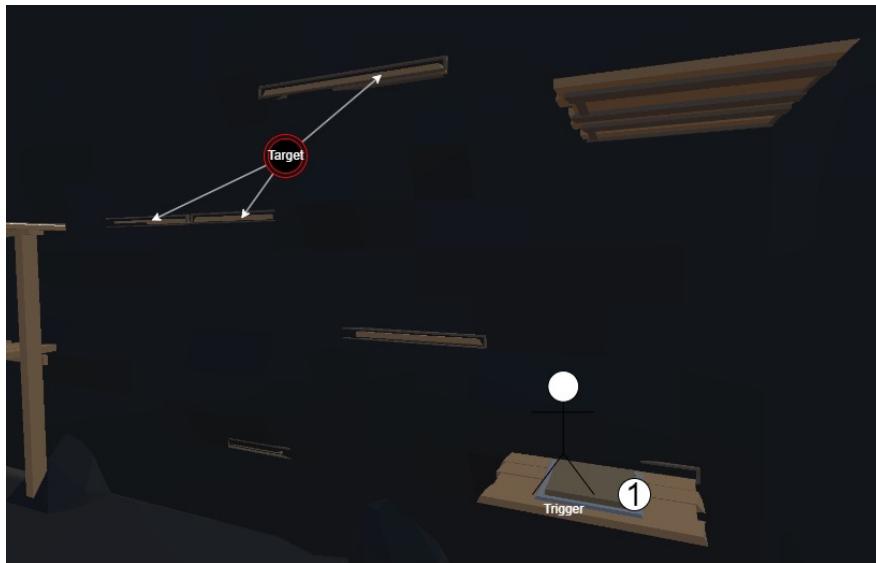
Labor, Stromkammer, Element 6:

Abbildung 25.: Element 6 in der Stromkammer, (Quelle: eigene Darstellung)

Die Druckplatte aus „Abbildung 25“ öffnet nun die letzten Plattformen, um in den Raum zu gelangen, indem der Strom angeschlossen werden kann. Hierfür wird ebenfalls ein weiteres Ich des Chronologen benötigt.

Fasst man die Schritte aus Element Labor, Stromkammer, Element 5: und Labor, Stromkammer, Element 6: zusammen, so würde dies einem Graphen aus Abbildung 14 oder 15 gleichen. Der Spieler muss in ein Ich des Chronologen „*splitten*“ um über die Druckplatte auf der Kranplattform die Plattformen an der Wand aktivieren. Das ist in „Abbildung 15 und 16“ jeweils die blaue Zeitlinie. Anschließend „*mergt*“ der Spieler wie in „Abbildung 16“ zurück auf den Chronologen und „*splittet*“ in ein weiteres Ich. Denkbar ist es auch, dass der Spieler wie in „Abbildung 15“ auf die rote Zeitlinie zurück „*mergt*“ um anschließend die geöffnete Druckplatte zu aktivieren. Nachdem diese Plattform eine Zeit aktiv war, „*mergt*“ der Spieler zurück auf den Chronologen und gelangt so in den Raum, in dem er den Strom einschalten kann.

Labor, Stromkammer, Element 7:

Abbildung 26.: Element 7 in der Stromkammer, (Quelle: eigene Darstellung)

Angekommen in der Stromschaltzentrale, muss der Spieler über die 2 Stromschalter den Strom für das Labor anstellen. Hierzu muss der Spieler zunächst in ein weiteres Ich des Chronologen „*splitten*“ damit nach einem folgenden „*Merge*“ beide Hebel der Aktivatoren betätigt werden können. Ein Graph, der diese Interaktion beschreibt, ist in „Abbildung 12 und 14“ zu finden. Nachdem der Strom angestellt wurde, gelangt der Spieler wieder in die Grotte zurück.

Nun muss der Spieler den anderen Ausgang öffnen.

Labor, Stromkammer, Element 8:

Abbildung 27.: Element 8 in der Stromkammer, (Quelle: eigene Darstellung)

Auf dem Felsen an der Wand des Raumes, in dem der Spieler zuvor den Strom aktiviert hat, befindet sich ein Hebel, der das Tor zum Ausgang der Grotte öffnet. Dieser Hebel benötigt Strom. Um auf den Felsen zu kommen, benötigt der Spieler einen Teil der Treppe, um von diesem auf den bestehenden Teil der Treppe zu springen. Das fehlende Stück befindet sich in der Eingangskammer, aus der der Spieler am Anfang über die Plattformen herausgeklettert ist. Zunächst muss der Spieler wieder in diese Kammer. Die Kammer kann über einen zweiten Weg geöffnet werden. Das Portal kann über die in "Abbildung 28" gekennzeichneten Hebel geöffnet werden.

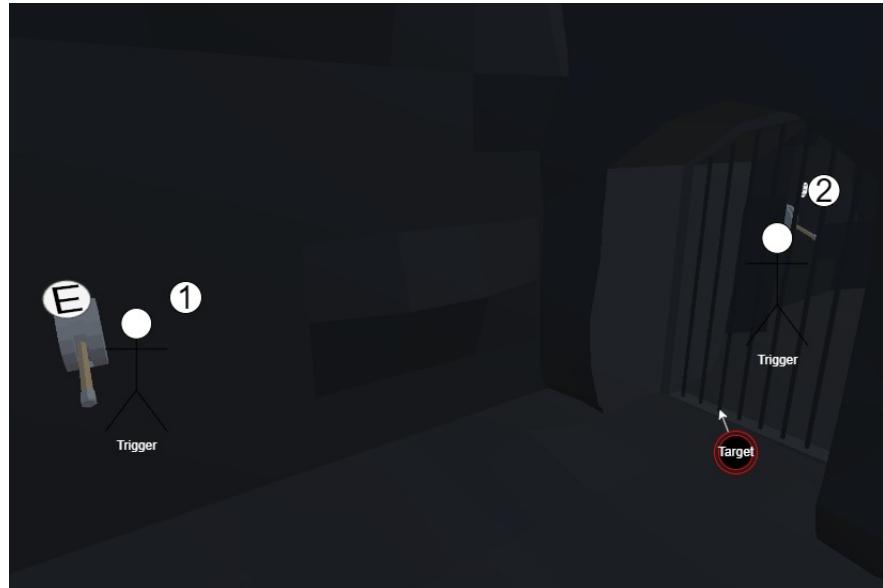
Labor, Stromkammer, Element 9:

Abbildung 28.: Element 9 in der Stromkammer, (Quelle: eigene Darstellung)

Der Spieler muss nun mit einem Ich des Chronologen in die Kammer zurück, um den Hebel in der Kammer zu bedienen. Dafür benötigt der Spieler erneut ein Ich des Chronologen, da der Kran hierfür aktiviert werden muss. Vom Hochstand, der neben dem Portal am Ausgang steht, kann der Spieler auf die Plattform des Kranes springen. Der Kran fährt eine Runde von der Öffnung an der Wand zum Hochstand und wieder zurück, ehe er weiter zur Schnittstelle mit dem zweiten Kran fährt. So kann der Spieler nun mit einer „gesplitteten“ Version des Chronologen in die Kammer gelangen. Auf der Außenseite der Kammer benötigt der Spieler ebenfalls ein weiteres Ich. Sobald die beiden Hebel gleichzeitig aktiviert wurden, öffnet sich das Portal und der Spieler kann die darin liegende Treppe heraustragen. Diese muss er, wie in „Abbildung 27“ gezeigt, vor die bestehende Treppe legen, damit der Spieler auf den Felsen springen kann. Der dabei entstehende Graph wird in „Abbildung 29“ gezeigt.

Das grüne Ich des Chronologen aktiviert den Kran, sodass das blaue Ich über den Kran in die Kammer gelangt. In der Kammer angelangt, aktiviert der Spieler den inneren Hebel. Nach einem „Merge“ zurück auf den Chronologen, „splittet“ er in das rote Ich und öffnet mit ihm gleichzeitig mit dem blauen Ich den zweiten Hebel und damit das Portal. Anschließend „mergt“ er auf den Chronologen zurück, um durch das offene Portal in die Kammer zu gelangen. Anschließend trägt er die Treppe an die in „Abbildung 27“ gekennzeichnete Stelle.

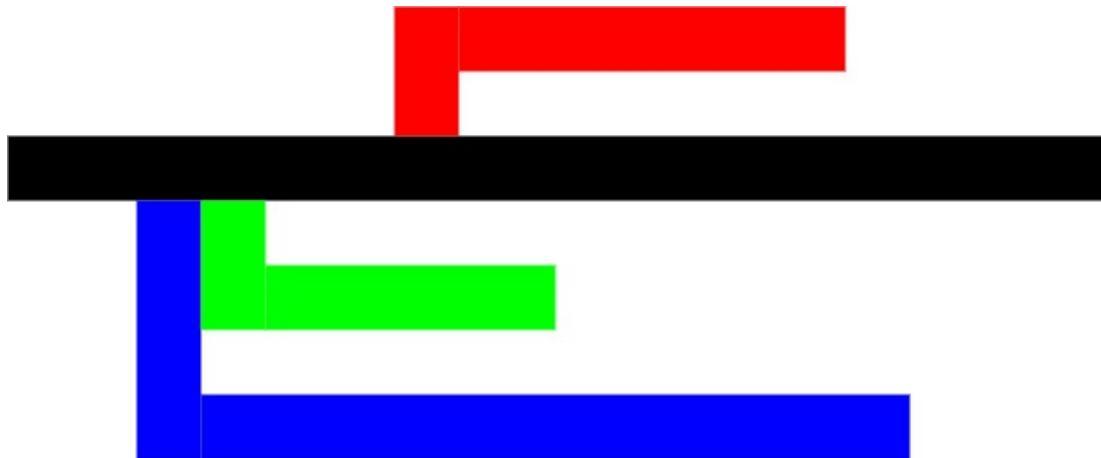


Abbildung 29.: Lösungsweg gesamt Element 8 bis 9, (Quelle: eigene Darstellung)

Labor, Stromkammer, Element 10:



Abbildung 30.: Element 10 in der Stromkammer, (Quelle: eigene Darstellung)

Im letzten Schritt muss der Spieler über die Treppe auf den Felsen springen und kann durch den Hebel das Portal am Ausgang öffnen. Der Hebel besitzt nun Strom und das Portal kann geöffnet werden. So gelangt der Spieler zurück zum Eingang des Labors.

3.10.2.2. Konzipierte Rätsel

Im Anschluss folgen nun die konzipierten Rätsel, diese kommenden Rätsel werden allesamt im zweiten Level des Prototyps zu finden sein. Den ersten Raum, um den

es gleich gehen wird, wird Laboreingang genannt und wurde in der Szene umgesetzt, allerdings besitzt dieser Raum noch keine Interaktivität.

Zuerst wird auf die einzelnen Elemente des Laboreingangs eingegangen. Anschließend wird die Lösung mitsamt einem daraus entstehenden Graphen gezeigt.

Labor, Laboreingang, Element 1:



Abbildung 31.: Element 1 im Laboreingang, (Quelle: eigene Darstellung)

Wie in "Abbildung 31" besitzt der Eingang eine bewegbare Plattform, die sich zwischen

der Rückwand des Einganges und dem anliegenden Begrenzer bewegen kann. Zudem ist es möglich, diese Plattform auch auf die Höhe des Begrenzers anzuheben. Der Spieler hat dazu zwei Ventile zur Auswahl. Mit dem einen Ventil kann der Spieler die Plattform an die Wand oder wieder zurückfahren. Mit dem anderen Ventil kann er die Plattform auf die Höhe des Begrenzers oder wieder auf den Boden fahren lassen. Das ermöglicht einem „gesplitteten“ Ich des Chronologen die Stützen an der Wand vor die Hebel zu stellen.

Durch die bewegende Plattform kann der Spieler jetzt Stützen aufeinander zusetzen. Sobald der Chronologe ein Objekt trägt, kann er allerdings nicht springen.

Labor, Laboreingang, Element 2:

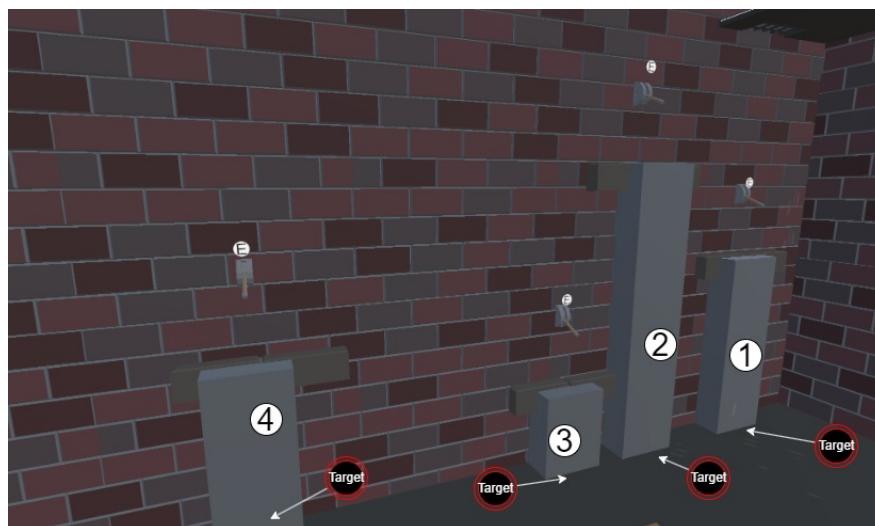


Abbildung 32.: Element 2 im Laboreingang, (Quelle: eigene Darstellung)

Wie eben dargestellt und in „Abbildung 32“ zu sehen, sind an einer der Seiten des Einganges Hebel an der Wand befestigt. Jeweils unter den einzelnen Hebelen ist ein kleiner Vorsprung, der dem Spieler anzeigen soll, an welche Stelle die Stütze gestellt werden muss. Der Spieler hat die Möglichkeit, auf die Stützen 4 und 2 zu klettern. Durch die Hilfe zweier weiterer Chronologen kann der Spieler nun mit einem Ich die Stützen in der richtigen Reihenfolge aufzustellen, damit ein weiteres Ich zu den Hebelen klettern kann.

Labor, Laboreingang, Element 3:

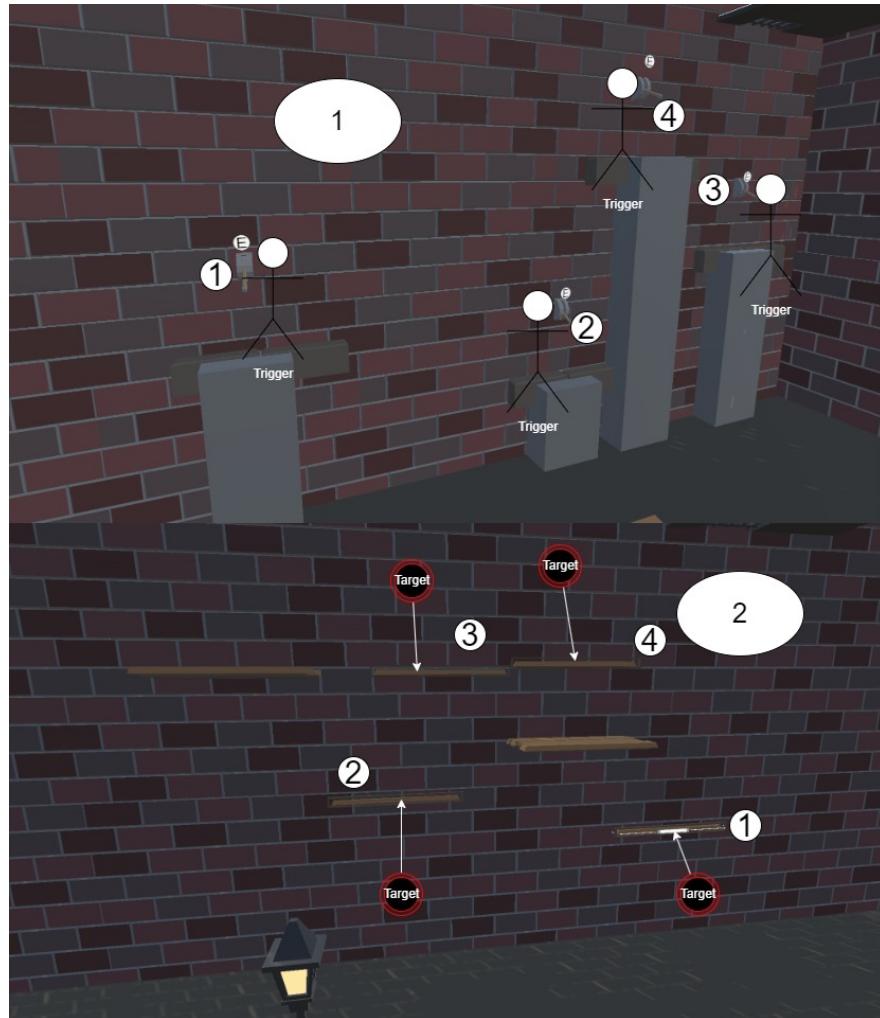


Abbildung 33.: Element 3 im Laboreingang, (Quelle: eigene Darstellung)

Die Hebel, in „Abbildung 33“, an der Wand mit der Markierung 1 öffnen jeweils eine Plattform an der gegenüberliegenden Wand, welche mit einer 2 markiert ist. Ein Ich des Chronologen muss nun, wie in Element 2 angesprochen, die Hebel in der richtigen Reihenfolge betätigen. Dadurch kann ein weiteres Ich auf der gegenüberliegenden Seite über die ausgefahrenen Plattformen in die nächste Etage des Einganges klettern kann.

Labor, Laboreingang, Element 4: Das Ich, das über die Hebel die Plattformen auf der gegenüberliegenden Wand betätigt, kann ebenfalls auf die zweite Etage des Eingangs klettern. Auf der oberen Etage befinden sich zwei von vier Hebelen, die aktiviert werden müssen, damit der Spieler in das Labor gelangen kann. Außerdem befinden sich jeweils neben dem Eingang in diesen Eingangsräum zwei weitere Hebel, die der Spieler aktivieren muss. Über der schweren Eisentür befinden sich vier Lampen,

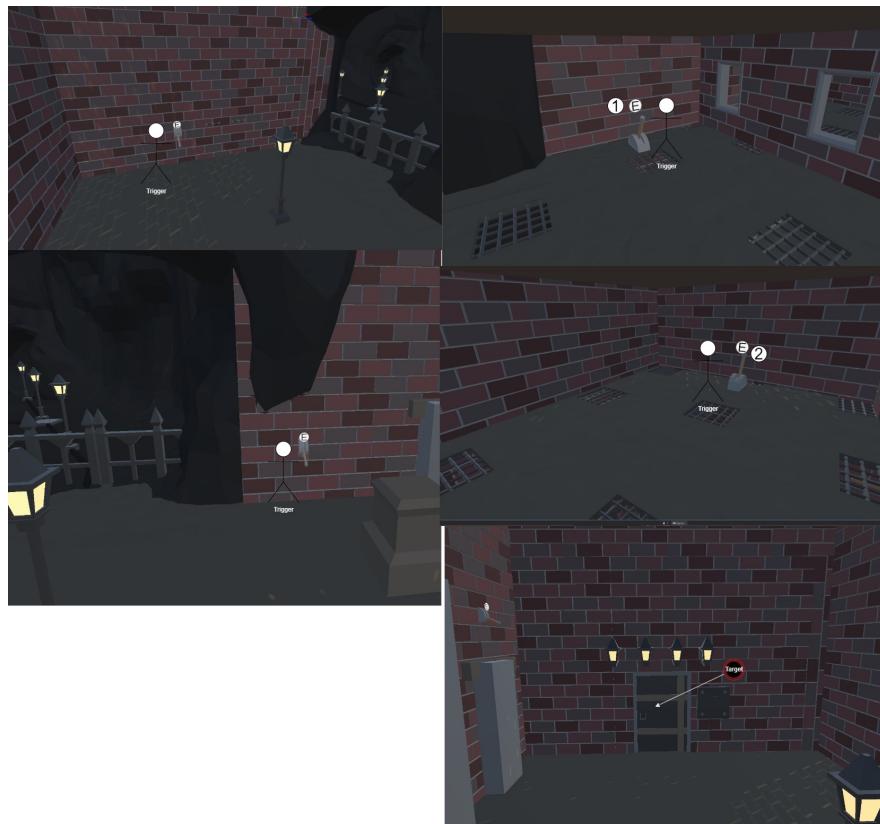


Abbildung 34.: Element 4 im Laboreingang, (Quelle: eigene Darstellung)

die die Aktivität der vier Hebel anzeigt (Licht an/ aktiv – Licht aus/ inaktiv). Wird ein Hebel betätigt, so leuchtet die jeweilige Lampe. Sobald alle vier Hebel gleichzeitig aktiviert wurden, kann der Spieler durch die Tür gehen. Die Hebel müssen allerdings aktiviert bleiben, bewegt sich ein „gesplittetes“ Ich vom Hebel weg, so schließt sich die Tür (vgl. „Abbildung 34“).

Da diese vier Elemente miteinander in einer abgestimmten Reihenfolge betätigt werden müssen, ergibt sich ein in „Abbildung 35“ abgebildeter Graph.

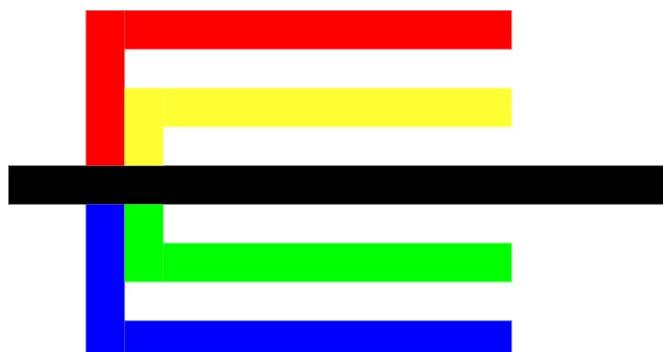


Abbildung 35.: Lösungsweg gesamt Element 1 bis 4, (Quelle: eigene Darstellung)

Zuerst muss der Spieler in das rote Ich des Chronologen "*splitten*". Durch diesen kann der Spieler die am Boden liegende Plattform bewegen. Der Spieler muss jetzt für einen späteren Zeitpunkt die Plattform jeweils so positionieren, dass er die Stützen für ein weiteres Ich richtig positionieren kann. Bevor der Spieler "*zurückmergt*" betätigt er den Hebel, der neben den zwei Ventilen zu finden ist. Dadurch wird eine der vier Lampen aktiviert. Anschließend "*mergt*" der Spieler zurück auf den Chronologen. Von da aus "*splittet*" er auf das gelbe Ich. Mit diesem stellt der Spieler nun einem weiteren Ich die Stützen an die Hebel, damit er diese später aktivieren kann. Der Spieler muss zunächst die Stützen so anordnen, wie sie in "Abbildung 32" dargestellt sind. Anschließend muss der Spieler die Stütze 4 neben Stütze 3 stellen, damit der dort positionierte Hebel umgelegt werden kann. Danach stellt der Spieler die Säule 3 zwischen Säule 1 und 2. Die Stütze 4 stellt er auf diese. Mithilfe der Plattform und der Stütze kann der Spieler nun den Hebel bei Stütze 1 betätigen. Dadurch wird die erste Lampe aktiviert. Anschließend kann er auf Stütze 2 klettern. Zum Schluss klettert er in die zweite Etage und betätigt den Hebel, der den Zugang zum Labor öffnet. Das Ich, das die Stützen positioniert, betätigt nun den Hebel, der links neben dem der Stütze 4 zu finden ist. Dadurch wird die zweite Lampe aktiviert. Anschließend muss der Spieler auf den Chronologen zurück "*mergen*". Von dort aus "*splittet*" er in das grüne Ich, um nun über die Stützen an die Hebel zu gelangen. Mit diesen Hebelen aktiviert der Spieler nun mit einem weiteren Ich an der gegenüberliegenden Wand die Plattformen und damit den Weg. Am Schluss klettert er, wie eben beschrieben, in die zweite Etage und betätigt den Hebel. Jetzt wird die dritte Lampe aktiviert und der Spieler benötigt nur noch ein weiteres Ich. Hierfür "*mergt*" der Spieler zurück auf den Chronologen und "*splittet*" in das verbleibende blaue Ich. Mit diesem klettert er nun über die, sich öffnenden, Plattformen auf die zweite Etage. Dort aktiviert er den verbleibenden vierten Hebel. Nach einem "*Merge*" auf den Chronologen muss der Spieler nur noch warten und kann, nachdem alle Hebel aktiviert wurden, in das Labor eintreten.

3.11. Informationen für den Spieler

Ein wichtiger Bestandteil des Spiels ist es, dass der Spieler Informationen über das aktuelle Geschehen und das bereits Geschehene in der Welt erhält. Hierbei wird es so dargestellt, dass der Chronologe Notizen des Gesehenen und Geschehenen macht. Diese Notizen kann der Spieler über den Datenleser zu jeder Zeit wieder aufrufen. Es wird als ein Journal mit mehreren Unterkategorien dargestellt werden. Im Folgenden werden die einzelnen Unterkategorien, über die der Spieler Informationen erhält, aufgezählt.

3.11.1. Informationen über die Spielwelt

Der Chronologe macht sich Notizen zu den einzelnen Spielwelten in den verschiedenen Zeitlinien. Dabei beschreibt er das Aussehen des Ortes, in dem er sich befindet und was mit diesem passiert sein könnte. Der Spieler wird vereinzelt auch auf die Menschen dieser Zeitlinie stoßen. Diese erzählen ihm mehr über den Ort und die Geschehnisse. Durch das Reisen in eine neue Zeitlinie erhält der Spieler ein weiteres Ich des Chronologen zur Auswahl. Dieses Ich des Chronologen kann, sobald der Spieler dieses nutzt, weitere Hintergrundinformationen erhalten. Dieses Ich lebt in der derzeitigen Zeitlinie und kann daher aus der Geschichte der Zeitlinie sprechen. Dabei ist es auch für den Spieler interessant zu erfahren, welche Entscheidungen in dieser Zeitlinie getroffen wurden. Diese erhält der Chronologe ebenfalls über andere Menschen dieser Zeitlinie und dem Ich des Chronologen.

3.11.2. Informationen über Szeneobjekte

Sobald der Spieler das erste Mal mit einem Interaktionsobjekt der Spielwelt interagiert hat, macht er sich Notizen über das Beobachtete. Er notiert sich die Objekte, die z. B. mit einer Druckplatte zusammenhängen und dass sie nur aktiv ist, solange er auf ihr stehen bleibt. Gleichwohl werden hierbei auch wichtige Informationen über die Steuerung dieser Objekte notiert. So erfährt der Spieler, mit welchen Tasten er diese Objekte bedienen kann.

3.11.3. Informationen über das User-Interface

Die User-Interfaces, die der Spieler während des Spiels sieht, sind zum Teil die Oberflächen der Gerätschaften des Chronologen. Der Spieler erhält darüber über den Dialog Informationen. Die wichtigsten Informationen aus dem Dialog werden im Datenleser des Chronologen gespeichert, sodass der Spieler sie jederzeit nachlesen kann. Zusätzlich werden die Head-Up-Display (HUD) Informationen, die der Spieler in der Spielwelt sieht, als Notiz in den Datenleser eingetragen.

3.11.4. Informationen über den derzeitigen Storystand

Das Spiel besteht aus einer Geschichte, die das Handeln des Chronologen bestimmt. Den aktuellen Stand dieser Geschichte kann der Spieler im Datenleser nachlesen. Dabei notiert der Chronologen auch eigene Gedankengänge im Story-Journal. Diese helfen dem Spieler einzuschätzen, wie sich der Chronologe fühlt und auch wie man den derzeitigen Stand der Spielwelt verstehen muss. Wichtige Schlüsseldialoge werden

dabei auch in den Datenleser übernommen. Der Spieler kann sich diese jederzeit erneut durchlesen.

3.11.5. Informationen über die Maschine

Der Chronologe war einige Jahre damit beschäftigt, diese Maschine zu entwerfen und zu entwickeln. Die Maschine besitzt verschiedene Kontrolleinheiten, wie den Graphen im UI, oder die Überwachungsfunktion der Stabilität des Kontinuums. Die Erklärungen zum UI des Spielers erfährt der Spieler ebenfalls über den Datenleser. Die Informationen umfassen das gesamte UI des Spielers und deren Hintergründe.

3.12. Dialoge, die im Spiel vorkommen

Ein weiterer wichtiger Bestandteil als Informationsquelle für den Spieler ist der Dialog des Chronologen und seiner Ichs. Der Dialog ist dabei in verschiedene Kategorien eingeteilt. Zum einen gibt der Chronologe Bemerkungen von sich, die zum Teil Hinweise auf Rätsel für den Spieler sind oder Erkenntnisse darstellen und den Spieler auf bestimmte Informationen hinweist wie eine sichtbare Kletterkante. Zum anderen gibt es Dialoge, die in einer fertigen Umsetzung Zwischensequenzen darstellen, die entweder vom Spieler gesteuert werden müssen, oder der Spieler sie nur überspringen kann. Dabei ist aber die Interaktion des Spielers mit der Welt blockiert und muss sich diese anschauen.

3.12.1. Eingeworfene Dialoge

Diese nicht zu überspringenden Dialoge sind eingeworfene Dialoge des Chronologen. Generell ist es so, dass Dialoge dazu führen, dass der Spieler zunächst keine weitere Interaktionsmöglichkeit hat, als den Dialog weiterzuführen. Eingeworfene Dialoge laufen parallel zum Spielgeschehen ab und können dem Spieler Hinweise vermitteln, z. B. zu den Zuständen von bestimmten Objekten. Nach längerem Ausprobieren, wie der Spieler die Rätsellemente anordnen muss, kann der Chronologe Hinweise einbringen.

3.12.1.1. Hinweise über Zustände

Beim erstmaligen Interagieren mit einem Objekt äußert der Chronologe sein Wissen, welche Handlung eine Interaktion bewirken würde. So ist unter anderem ein Hebel nur dann aktiviert, wenn man bei ihm stehen bleibt. Entfernt sich der Spieler, so wirft der Chronologe ein, dass sich der Hebel nach kurzer Zeit deaktivieren wird und man mehr Zeit einplanen muss. Sofern der Chronologe sehen kann, welches Objekt

sich nach einer Interaktion bewegt, teilt er es dem Spieler mit. Diese Option kann über das Einstellungsmenü deaktiviert werden. So kann der Spieler sich ganz auf das Ausprobieren konzentrieren.

3.12.1.2. Hinweise über Rätsel

Benötigt der Spieler für bestimmte Abschnitte zu lang, so kann der Chronologe Hinweise geben. Er hilft dem Spieler dadurch, die richtige Reihenfolge der zuerst zu aktivierenden Elemente zu finden. Er sagt dabei etwa Folgendes: „Ich muss diesen Kran bewegen, vielleicht ist hier etwas, das den Kran bewegt“.

3.12.2. Story relevante Dialoge

Der Spieler erfährt durch Dialoge und Textsequenzen vor und nach den Levels die Geschichte des Spiels. Diese Dialoge im Spiel sind für den Spieler überspringbar. Allerdings kann der Spieler während eines relevanten Dialogs keine Bewegungen ausführen. Der Sprachdialog ist zumeist eine Unterhaltung mit dem Spieler oder mit sich selbst. Der Chronologe unterhält sich ebenso mit den Ichs aus den anderen Zeitlinien, da er auf diese Weise ebenfalls über das Leben in den anderen Zeitlinien Informationen sammeln kann.

3.13. Sounddesign

Das Sounddesign des Spiels unterstützt den visuellen Eindruck des Spielers durch ein auditives Feedback. Dabei erzeugen unter anderem der Chronologe und die Objekte, mit denen der Spieler interagieren kann, Geräusche. Dieses auditive Feedback hat ein höheres immersives Wirken auf den Spieler.

Das entsprechende Sounddesign wird im Folgenden in unterschiedliche Kategorien aufgeteilt. Dabei sind atmosphärische und unterstützende Soundelemente zu unterscheiden.

3.13.1. Hintergrund Musik

Die Hintergrundmusik des Spiels dient als atmosphärische Unterstützung des Spiels. Sie gibt einen Rahmen für das Visuelle und beschreibt durch ihre Töne den aufkommenden Ausdruck. So wird etwa eine Höhle mit einem eher düsteren; tropfenden; ein Steinbruch mit einem hallenden und rauen Ton unterlegt.

Es wird hierbei zwischen dem Spielhintergrund und dem Pause-Menü unterschieden.

3.13.1.1. Pause Menü

Das Pause-Menü wird immer mit derselben Melodie unterlegt, dass dem Spieler Ruhe und Sicherheit vermitteln soll.

3.13.1.2. Spiel Hintergrund

Die Hintergrundmusik der Level ist durch die Charakteristik des Levels definiert. Sie dient der atmosphärischen Unterstützung des aktuellen Levels. Durch den atmosphärischen auditiven Eindruck fühlt sich der Spieler in die Welt integriert.

3.13.2. Umgebungsgeräusche

Alle Interaktionsobjekte der Spielwelten geben ein Geräusch von sich, nachdem diese verschoben, betätigt oder aktiviert wurden. So erklingt z. B. beim Aktivieren einer Druckplatte ein schleifendes Geräusch, da die Druckplatte durch ihre Öffnung geschoben wurde und auf Holz und Metall stößt. Des Weiteren macht z. B. das Klicken auf Buttons im Menü einen entsprechenden Ton.

3.13.3. Charakter Geräusche

Der Chronologe ist kein stiller Teilnehmer des Spiels. Sobald der Chronologe über Abgründe hinwegspringen muss, strengt ihn das an und er gibt einen angestrengten Laut von sich. Jeder Aktivität des Chronologen erzeugt seinen eigenen Laut. Zum Beispiel hört der Spieler die Fußstapfen, wenn der Chronologe sich bewegt oder aber ein raschelndes Geräusch der Kleidung, sobald der Chronologe klettert oder einen Hebel umlegt.

3.13.4. Dialoge

Der Spieler kann den Dialog des Spiels über seinen Untertitel verfolgen. Zusätzlich zum Untertitel gibt es einen gesprochenen Dialog. Der Chronologe hat dabei eine tiefe und staubige Stimme. Der Spieler kann den Untertitel des Dialogs über das Einstellungsmenü ausschalten.

4. Visuelles Design des Prototyps

In diesem Kapitel wird auf das visuelle Design des Spiels konzeptionell und in der Prototypisierung eingegangen. Es beinhaltet sowohl Grafische-User-Interfaces (GUI) als auch die Vorgabe für den Art-Stil.

4.1. Moodboard

Um einen ersten visuellen Eindruck für dieses Spiel gewinnen zu können, wurde das in "Abbildung 36" abgebildete Moodboard entworfen. Es zeigt erste visuelle Ideen, wie man sich einen Geist in der Szene vorstellen kann und wie Steampunk in Low-Poly integriert werden kann.

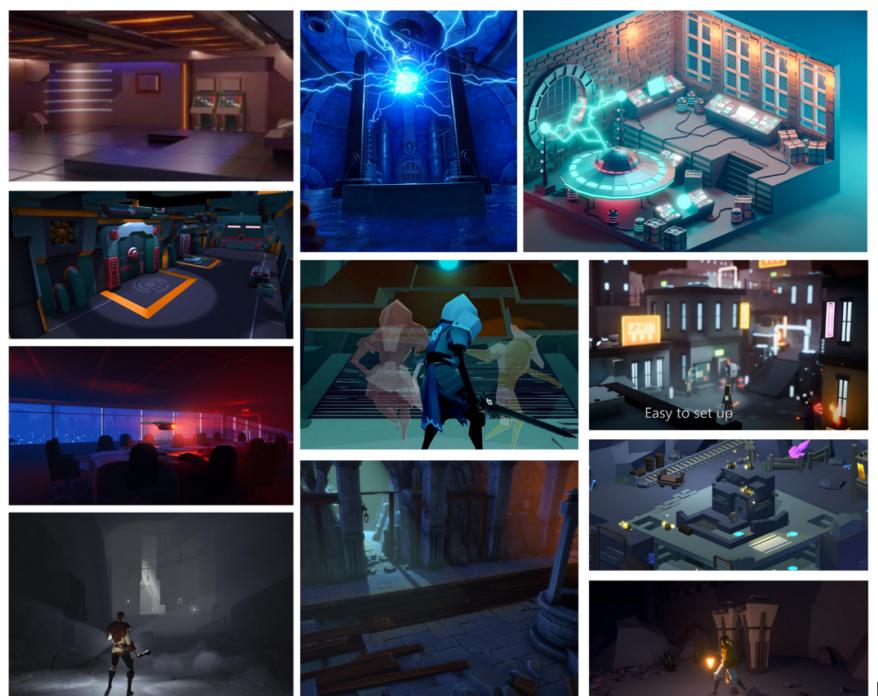


Abbildung 36.: Moodboard des Spiels, (Quellen: (von oben nach unten und von links nach rechts) sathak (2019), SaferDan (2013), Plays (o. D.), A44 (o. D.), Games (k.D.), Gamestar & Schemes (2016)) Schick (2017), Plane (o. D.), Sykoo (2018), Games (o. D.) Key (o. D.)

4.2. Art-Stil

Wie bereits in “Kapitel 4.1: Moodboard” angesprochen wurde, wird der Art-Stil des Spiels “Low-Poly” sein. Denkbar wäre es auch in einen texturierten Low-Poly Stil zu wechseln, um mehr Details in die Spielwelt integrieren zu können. Dazu im “Kapitel 7.2: Ausblick” mehr. Da das Spiel dem Steam-Punk Genre folgt, wird die Welt maschinell und viktorianisch aussehen.

Um aufseiten der Schriftart einen deckenden Eindruck zum Art-Stil zu schaffen, wurde die Schriftart Exo ausgewählt (herunterladbar unter Gama & Mientjes (o. D.)). Die Schriftart passt in das visuelle Gesamtbild des Spiels.

4.3. Aussehen des Chronologen

Der Chronologe ist durch sein Aussehen und sein Verhalten ebenfalls in das Spielgenre und den Art-Stil integriert. Er wirkt wie ein vor sich hin tüftelnder Wissenschaftler, der eher wenig aus seinem Labor kommt. Er trägt eine Brille mit großen eckig-milchigen Brillengläsern. Dazu trägt er eine Weste mit einem Hemd darunter. Zu seiner Weste mit Hemd trägt er eine ledrige, festsitzende Hose. Außerdem trägt er in die Jahre gekommene Lederschuhe (vgl. “Abbildung 37”).



Abbildung 37.: Spielobjekt des Chronologen, (Quelle: eigene Darstellung)

Im Kontext dieses Prototyps besitzen die Ich-Versionen des Chronologen unterschiedlich farbige Westen. Zum Beispiel hat der Chronologe aus den Zeitlinien des Tutorials eine blaue oder gelbe Weste. Diese Versionen des Chronologen kann man in

"Abbildung 39" oder "Abbildung 38" sehen. Durch die farblichen Merkmale ist der Chronologe für den Spieler unterscheidbar. Beim Lösen der Rätsel ist es wichtig zu wissen, welches Ich des Chronologen gerade an den jeweiligen Elementen der Rätsel steht. Diese farbliche Unterscheidung wird im Laufe dieses Kapitel in Abschnitt User Interface im Unterabschnitt 4.4.4.



Abbildung 38.: Spielobjekt des Chronologen, gelbe Weste, (Quelle: eigene Darstellung)



Abbildung 39.: 3D Modell des Chronologen, blaue Weste, (Quelle: eigene Darstellung)

Im Kontext der allgemeinen Konzeption besitzen die Chronologen, außer der Farbe, weitere Unterscheidungsmerkmale, die aber nicht im Prototyp umgesetzt wurden. So tragen die Chronologen unterschiedliche Kleidungen und Accessoires. Zudem sind die Körper der Chronologen unterschiedlich entwickelt und besitzen dadurch auch eine unterschiedliche Größe. Es kann aber auch zu Arbeitsunfällen oder dergleichen

gekommen sein, wodurch körperliche Veränderungen erfolgten. Diese Unterschiede in der Geschichte der einzelnen Chronologen spiegelt sich auch auf das Äußere des dreidimensional (3D) Modells wider.

Der Spieler erhält dadurch für jede Zeitlinie ein unterscheidbares Aussehen des Chronologen. Dadurch kann er im Verlauf des Spiels die einzelnen Aufnahmen der jeweiligen Chronologen unterscheiden und zuordnen.

4.3.1. Nicht mehr vom Spieler gesteuerte Chronologen

Wie bereits in Aufnahmen des Chronologen beschrieben, wird der Platzhalter des Chronologen transparent in der Szene dargestellt. Wie in den Abbildungen 40 und 41 zu sehen, haben die 3D-Körper des Chronologen transparente Materialien. Dadurch wird der Eindruck erweckt, dass dieses Ich im Moment der Spielzeit noch nicht existiert und zeitlich gesehen in der Vergangenheit an dieser Stelle steht.

Das aufgenommene Ich des Chronologen, das in den Abbildungen 37, 38 und 39, wird als solches in der Szene dargestellt. Ein nicht transparentes Aussehen des Chronologen bewirkt hierbei den Eindruck, dass das Ich des Chronologen aus der entsprechenden Zeitlinie einen Einfluss auf die Spielwelt besitzt. Der Spieler kann sich nun anhand dieses Ichs orientieren und weiß, dass dieses Ich gerade in der Szene aktiv ist und zu diesem Zeitpunkt nicht nur in der Vergangenheit existiert.

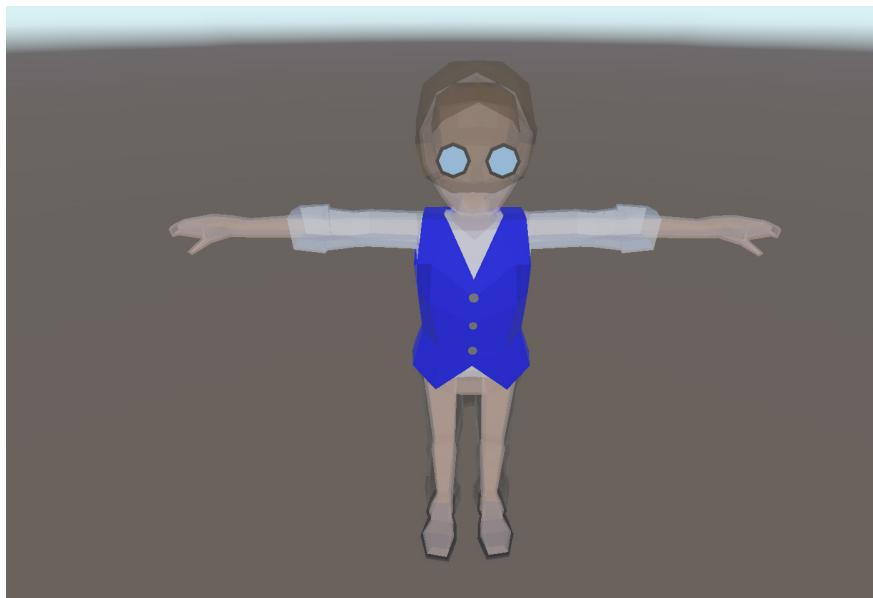


Abbildung 40.: 3D Modell Platzhalter, blaue Weste, (Quelle: eigene Darstellung)



Abbildung 41.: Platzhalter des Chronologen, (Quelle: eigene Darstellung)

4.4. User Interface

Im Folgenden werden die einzelnen Elemente der User-Interfaces vorgestellt und durch Mockups prototypisch visualisiert. Es handelt sich hierbei zum einen um das gesamte UI, das der Spieler während des Spiels zu sehen bekommt und zum anderen um das UI des Datenlesers, welches der Chronologe immer zur Hand hat.

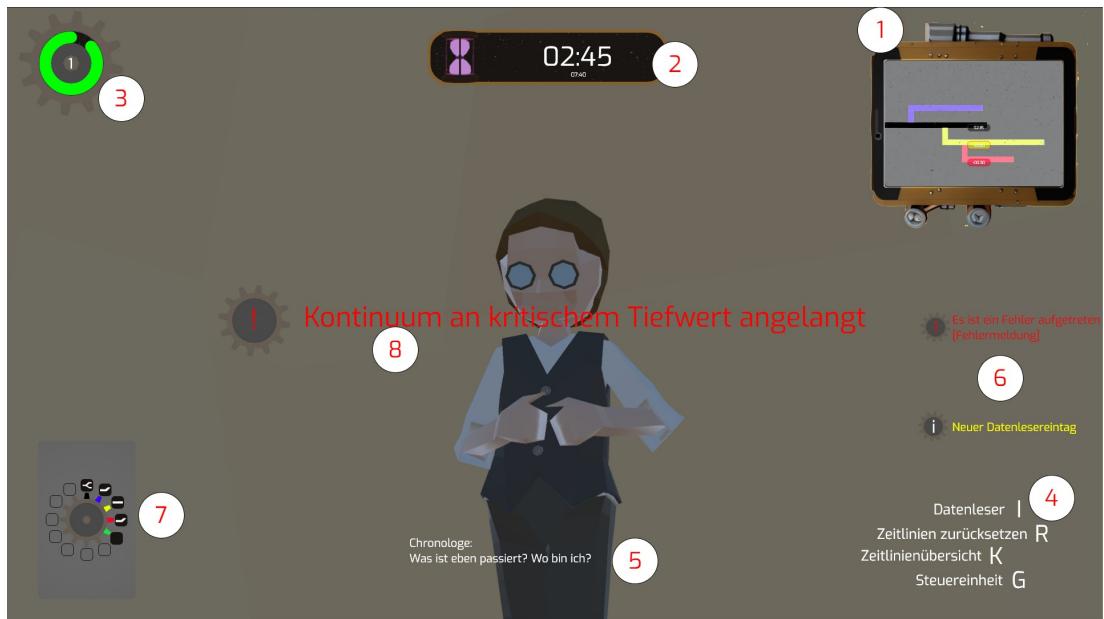


Abbildung 42.: Mockup Bildschirm des Spielers, (Quelle: eigene Darstellung)

Das Gesamte in Spiellevel UI ist als Mockup in "Abbildung 42" zu sehen und wird im Folgenden erklärt.

4.4.1. Graph

Wie in “Abbildung 42” zu sehen befindet sich der Graph (Markierung 1) oben rechts des Bildschirms. Er bildet die Zeitlinien, die der Spieler während des Spiels erstellt hat, ab.



Abbildung 43.: Mockup Zeitlinienübersicht im UI, (Quelle: eigene Darstellung)

Wie in “Abbildung 43” zu sehen, ist jede Zeitlinie mit der Farbe des Chronologen Ichs eingefärbt. Jede Zeitlinie, die nicht die aktive ist, hat dabei ein graues Overlay. Die aktuelle Zeitlinie (vgl. Markierung 1) ist in ihrer normalen Farbe dargestellt. Zusätzlich erhält jede Zeitlinie eine kleine Anzeige, auf dem die Spielzeit der Zeitlinie steht. Für parallel laufende Zeitlinien gibt es das auch, hier wird angezeigt, wie lange diese Zeitlinie noch aktiv ist (vgl. Markierung 2). Sobald keine Zeitlinie parallel läuft, wird die Anzeige nur auf der aktuellen Zeitlinie angezeigt (vgl. Markierungen 3 bis 5).

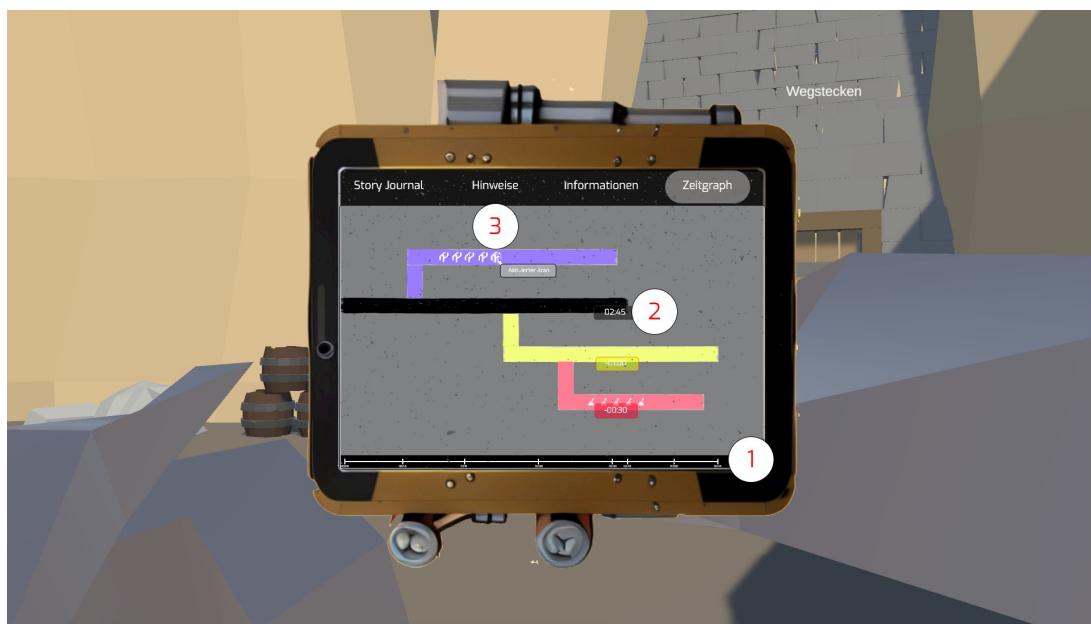


Abbildung 44.: Mockup geöffnete Zeitlinienübersicht, (Quelle: eigene Darstellung)

Sobald der Spieler sich eine Übersicht über alle erstellten Zeitlinien machen möchte, kann er den in “Abbildung 44” abgebildeten Graph sehen. Diese Ansicht enthält zum

einen eine Zeit-Skala (vgl. Markierung 1) über welche er die Zeitpunkte der Zeitlinie sehen kann. Die gezeigten Zeitlinien enthalten ebenso die Zeitanzeigen wie sie bereits beschrieben wurden (vgl. Markierung 2). Zusätzlich enthalten sie Abbildungen der Interaktion in Form von Icons, über die der Spieler hovern kann. Sie zeigen dem Spieler, was er zu diesem Zeitpunkt gemacht hat (vgl. Markierung 3).

4.4.2. Timer

In "Abbildung 42" wird bei der Markierung 2 der Timer des Spiels angezeigt. Er enthält zum einen die Spielzeit in der Szene des Spielers (große Schrift), als auch die gesamte Spielzeit, die der Spieler im Level verbracht hat (kleine Schrift). Der Inhalt der großen Schrift ist der Referenzwert für das System. Die kleine Schrift ist dagegen eine Referenz dafür, wie lange der Spieler insgesamt gebraucht hat, das Level abzuschließen.

4.4.3. Zeitkontinuum

Die Stabilität des Kontinuums wird in "Abbildung 42" an der Markierung links oben angezeigt. Die Stabilität des Kontinuums wird über die Anzeige des Kreisdiagramms angezeigt. Ist das Kontinuum stabil, so ist das Diagramm grün gefärbt. Wird das Kontinuum schwächer und der Kreis ist nur noch zur Hälfte gefüllt, so färbt es sich gelb. Ist das Kontinuum an einem kritischen Punkt, verfärbt es sich rot. Das ist für den Spieler der Hinweis darauf, auf den Chronologen zurück zu "mergen" und alle erstellten Zeitlinien zu löschen.

Der Wert in der Mitte des Zahnrades definiert die Anzahl an parallel laufenden Zeitlinien, welche durch das Auslösen von Paradoxen zerstört werden können.

Das Zahnrad, das den Hintergrund der Anzeige darstellt, dreht sich in die Richtung des wachsenden oder sinkenden Diagramm mit. Ist der Wert bei 2, so dreht sich das Zahnrad im Uhrzeigersinn und der Wert des Diagramms sinkt. Ist der Wert bei 1 und das Diagramm ist nicht ganz gefüllt, so dreht es sich gegen den Uhrzeigersinn, bis das Diagramm gefüllt ist. Ist das Diagramm gefüllt, so bleibt das Zahnrad stehen.

Sobald eine Zeitlinienkonvergenz droht, bekommt der Spieler die Meldung aus Markierung 8 eingeblendet. Im Anschluss wird er auf die Zeitlinie des Chronologen zurückgesetzt und alle seine angegliederten Zeitlinien werden gelöscht.

4.4.4. Charakter Übersicht

In "Abbildung 42" wird an der Markierung 7 die Übersicht über die verfügbaren Ichs aus den verfügbaren Zeitlinien angezeigt. Die Zacken des Zahnrades sind wie auf der Steuereinheit nach den unterschiedlichen Ichs gefärbt. Jede Zacke erhält an ihrem Ende einen kleinen Bildschirm, auf dem die Relation der verschiedenen Ichs aus den Zeitlinien abgebildet sind. Sie beziehen sich dabei immer auf das aktuell ausgewählte und gespielte Ich des Chronologen.

4.4.5. Steuerhinweis

Bei der Markierung 4 in "Abbildung 42" werden die grundlegenden Steuerhinweise, ausgeschlossen die Bewegungen, des Spiels für den Spieler angezeigt.

4.4.6. Dialoge

Der Untertitel des Sprachdialogs wird in "Abbildung 42" bei der Markierung 5 angezeigt. Er enthält den gesprochenen Text und den Namen des Sprechers.

4.4.7. Systemfehler und Benachrichtigungen

Der Spieler erhält bei der Markierung 6 in "Abbildung 42" die Information, dass es einen neuen Eintrag im Datenleser gab (gelber Text), sowie die Information darüber, dass er im Moment bestimmte Aktionen nicht ausführen kann oder ein Paradoxon an einer anderen Zeitlinie geschaffen hat (rote Schrift).

4.4.8. Tragen von Objekten



Abbildung 45.: Beispiel wie Chronologe Objekt trägt, (Quelle: GameTube (2014))

Sobald der Spieler ein Objekt trägt, erhält er wie in "Abbildung 46" zu sehen eine Anweisung, wie er das Objekt wieder hinlegen kann. Das Objekt trägt er dabei, wie es in "Abbildung 45" beim Spiel "The Talos Principle" zu sehen ist.

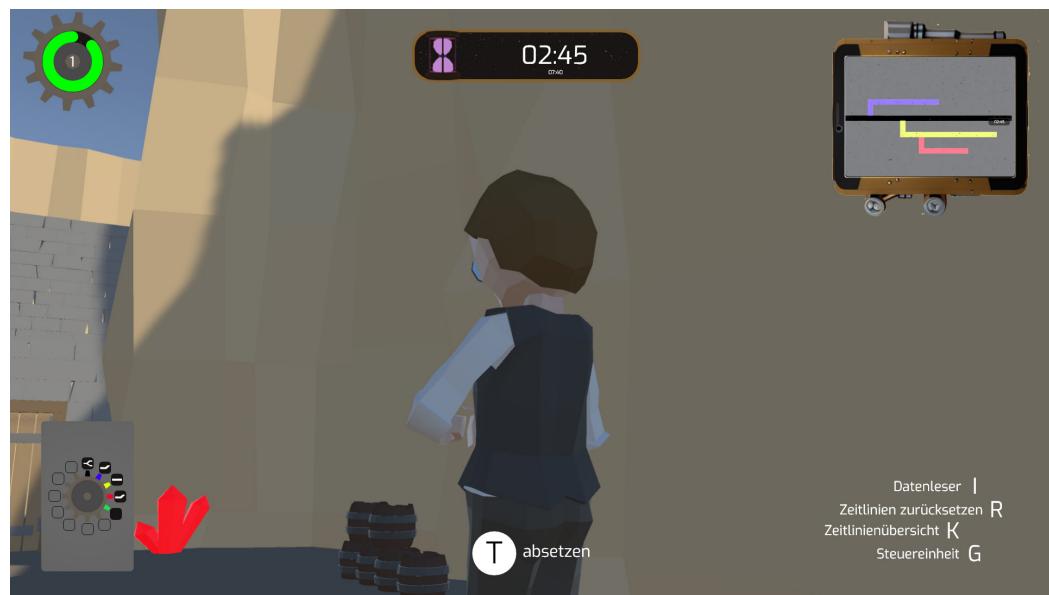


Abbildung 46.: User-Interface beim Tragen eines Spielobjektes, (Quelle: eigene Darstellung)

4.4.9. Datenleser



Abbildung 47.: Datenleser Collage, (Quelle: eigene Darstellung)

“Abbildung 47” zeigt eine Collage, wie das UI des Datenlesers auszusehen hat. Der Spieler kann darüber das Story-Journal einsehen, in das der Chronologe seine Notizen eingefügt hat. Zudem kann der Spieler erhaltene Informationen und Notizen nachlesen.

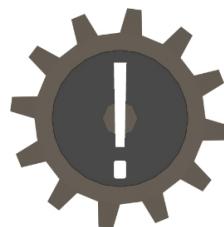


Abbildung 48.: Indiz für einen neuen Eintrag, (Quelle: eigene Darstellung)

Einträge, die das Symbol aus “Abbildung 48” enthalten, sind neu dazu gekommen und wurden noch nicht gelesen. Außerdem kann der Spieler die Liste der Informationen nach einer vorgefertigten Auswahl sortieren, etwa nach neuen Informationen oder einem bestimmten Typ Information.

4.4.10. Einstellungsmenü im Datenleser



Abbildung 49.: Einstellungsmenü im Datenleser, (Quelle: eigene Darstellung)

“Abbildung 49” zeigt das Pausemenü, das über den Datenleser dargestellt wird. Über ihn kann der Spieler, wie über das Startmenü, Spieleinstellungen tätigen. Die Einstellungen, die in der Collage abgebildet sind, sind die, die später einmal in diesem Spiel angedacht sind.

4.4.11. Bildschirmanzeige zum Abschluss des Levels



Abbildung 50.: Endbildschirm eines erfolgreich gemeisterten Levels, (Quelle: eigene Darstellung)

Sobald der Spieler das Ende des Levels erreicht, öffnet sich das in "Abbildung 50" gezeigte UI. Dabei sieht der Spieler eine Übersicht darüber, wie lange er für das Level gebraucht hat. Außerdem werden ihm seine erstellten Zeitlinien gezeigt, welche alle ein Fenster besitzen, auf dem die Spieldauer dieser Zeitlinie gezeigt wird.

4.4.12. Bildschirmanzeige bei Auftreten einer Zeitlinienkonvergenz

Falls der Spieler eine Zeitlinienkonvergenz auslöst, erfolgt folgender in "Abbildung 51" gezeigtes UI. Der Spieler kann das Level neu starten oder das Spiel beenden.

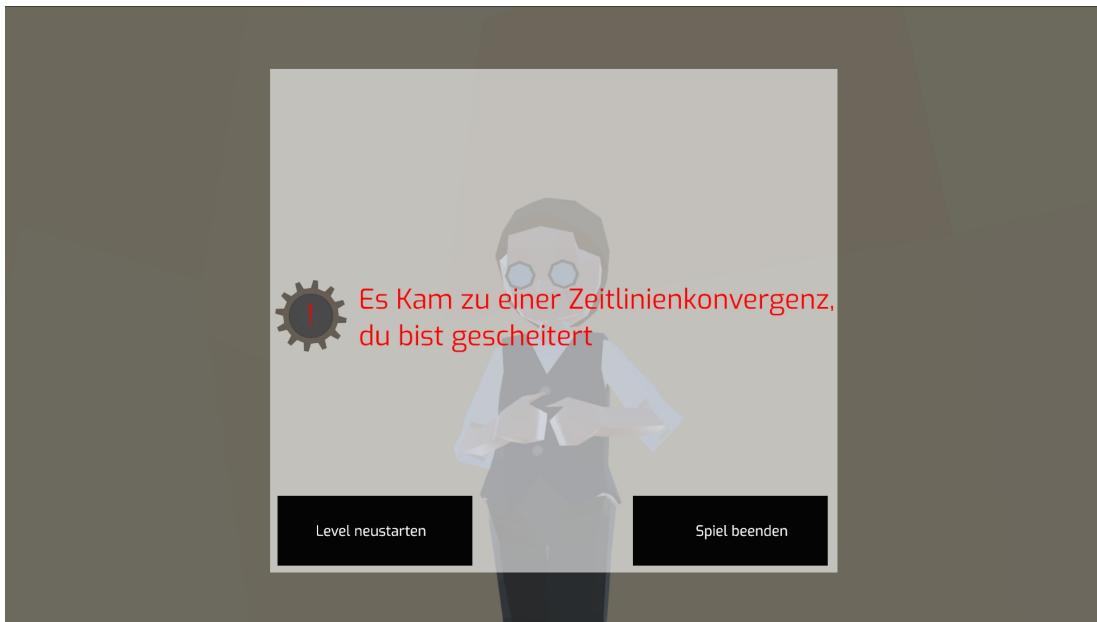


Abbildung 51.: Endbildschirm bei einer Zeitlinienkonvergenz, (Quelle: eigene Darstellung)

4.5. Führung durch das Level

Der Spieler erhält durch unterschiedliche Mechanismen eine Führung durch das Level. Dazu gehören unter anderem Partikelsysteme, Kletterkanten und Tooltips.

4.5.1. Partikelsysteme

Dem Spieler soll an bestimmten Stellen, wie an einer Druckplatte, vermittelt werden, dass er auf diese stehen soll, um mit seinem Gewicht einen Mechanismus auszulösen.

Wie in "Abbildung 52" zu sehen, wird, sobald der Chronologe nahe an einer Druckplatte steht, das Partikelsystem der Druckplatte aktiviert. Es zeigt durch seinen Fluss auf die Druckplatte dem Spieler an, dass er auf diese Druckplatte gehen muss, um sie durch seinen Körper zu aktivieren.



Abbildung 52.: Partikelsystem Druckplatte, (Quelle: eigene Darstellung)

4.5.2. Kletterkanten

Ein Bestandteil der Bewegungsmöglichkeiten des Spielers ist es, auf Gerüste oder Stützen zu klettern. Hierbei benötigt der Spieler auf zwei Wegen einen Hinweis darauf, wo und wie er auf Objekte klettern kann. Der Spieler kann durch ein Kabel, das an der Kante des entsprechenden Objektes liegt, erkennen, dass er auf das Objekt klettern kann. Das Kabel wird durch eine Schraffierung an der Kante unterstützt.



Abbildung 53.: Kantenmarkierung an Spielweltobjekten, (Quelle: eigene Darstellung)

Wie in “Abbildung 53” zu sehen ist, markiert die weiße Linie die Schraffierung der Kletterkante. Die Kante ist abgenutzt, weil bereits einige Menschen vor dem Chrono-

logen auf dieses Gerüst geklettert sind. Die schwarze Markierung auf der Abbildung stellt ein Kabel dar, das von der Kante als Sicherung befestigt ist. Über dieses Kabel kann man sich hochziehen, ohne abzurutschen.

Der zweite Weg wird im Folgenden “Kapitel 4.5.3: Tooltips” behandelt.

4.5.3. Tooltips

In Bezug auf die Kletterkante aus “Kapitel 4.5.2: Kletterkanten” erhält der Spieler eine zusätzliche visuelle Aufforderung, wie er mit der Kante zu interagieren hat. Diese Aufforderungen werden im Kontext dieser Arbeit Tooltips genannt. Diese Tooltips beinhalten einen Keycode der Tastatur, durch welchen der Spieler den Chronologen auf das Gerüst klettern lassen kann. Im Fall der Kletterkante ist es die Leertaste.

Weitere Gegenstände, mit denen der Chronologe interagieren kann, enthalten ein Tooltip. Zum Beispiel wird ein Tooltip angezeigt, sobald sich der Spieler einem Hebel nähert. Dies ist in “Abbildung 54” und in “Abbildung 55” zu sehen. Bei der Treppe handelt es sich hierbei um ein tragbares Objekt. Der Spieler muss dieses Treppenbruchstück an einem anderen Ort platzieren.

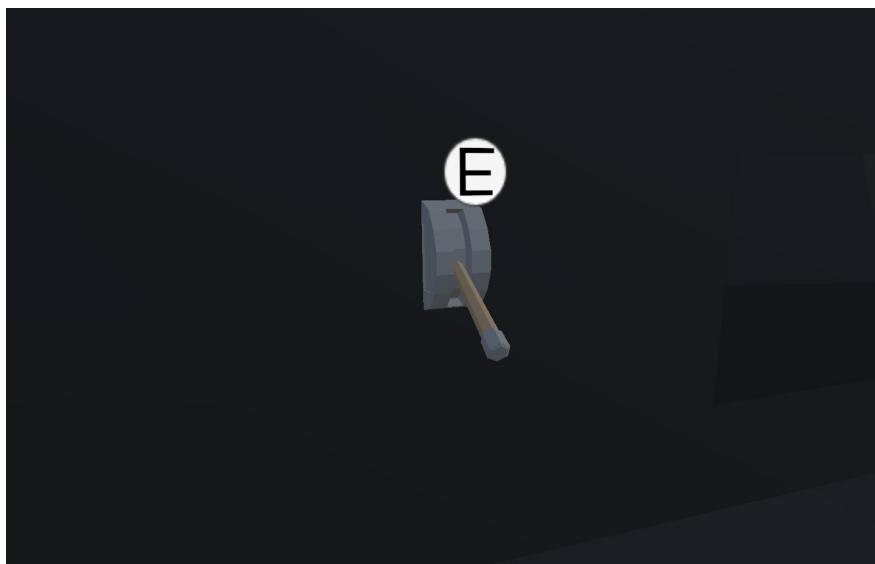


Abbildung 54.: Hebel mit Tooltip, (Quelle: eigene Darstellung)

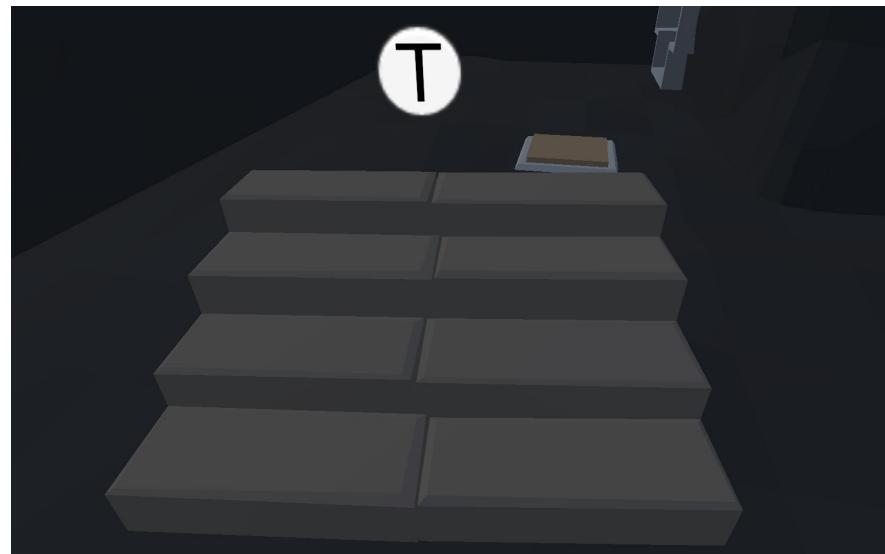


Abbildung 55.: Treppe mit Tooltip, (Quelle: eigene Darstellung)

Damit der Spieler ein besseres Verständnis dafür erhält, welche Interaktion er durch das Betätigen der Taste auslöst, wird der Name der Interaktion neben den Keycode positioniert. Diese Änderungen werden zu dem bislang implementierten Stand ergänzt und in den folgenden Abbildungen 56, 57 und 58 dargestellt.

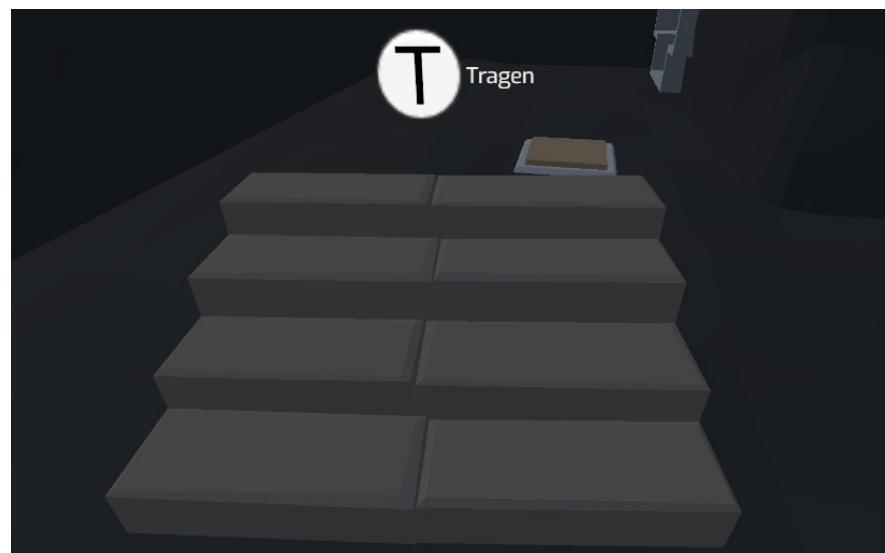


Abbildung 56.: Treppe mit Tooltip und Aufforderung, (Quelle: eigene Darstellung)

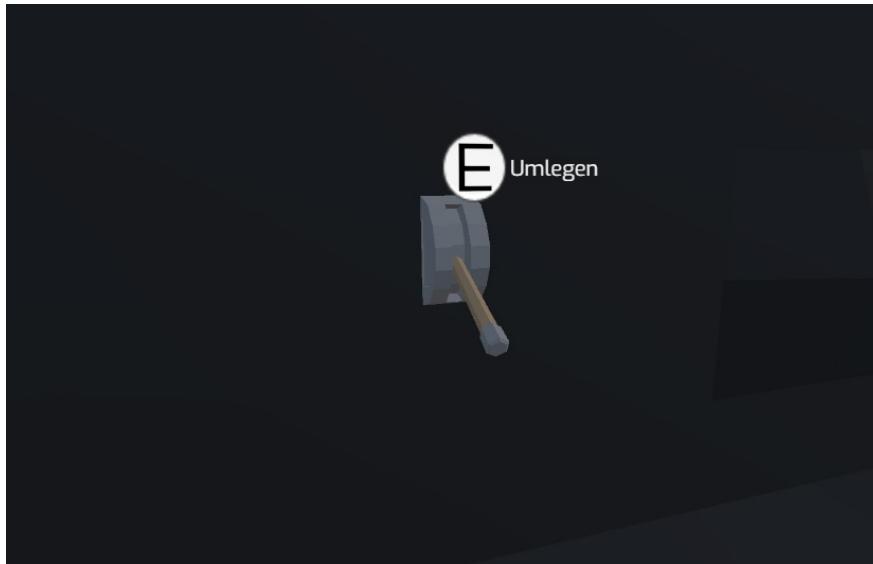


Abbildung 57.: Hebel mit Tooltip und Aufforderung, (Quelle: eigene Darstellung)



Abbildung 58.: Kante mit Tooltip und Aufforderung, (Quelle: eigene Darstellung)

Dadurch versteht der Spieler schneller und besser, welche Interaktion er bewirkt. Er muss nicht zuerst im Datenleser nachlesen, welche Auswirkung seine Interaktion haben wird, sondern diese sind beim Betrachten des Tooltips bereits ersichtlich.

4.6. Interaktionsgegenstände

Im folgenden Kapitel werden verschiedene Objekte dargestellt, mit denen der Chronologe interagieren kann. In der Aufzählung werden zum einen die bislang bestehenden Objekte aufgezählt und zum anderen konzeptionelle Anpassungen, die die Objekte ergänzen, vorgestellt.

4.6.1. Druckplatte

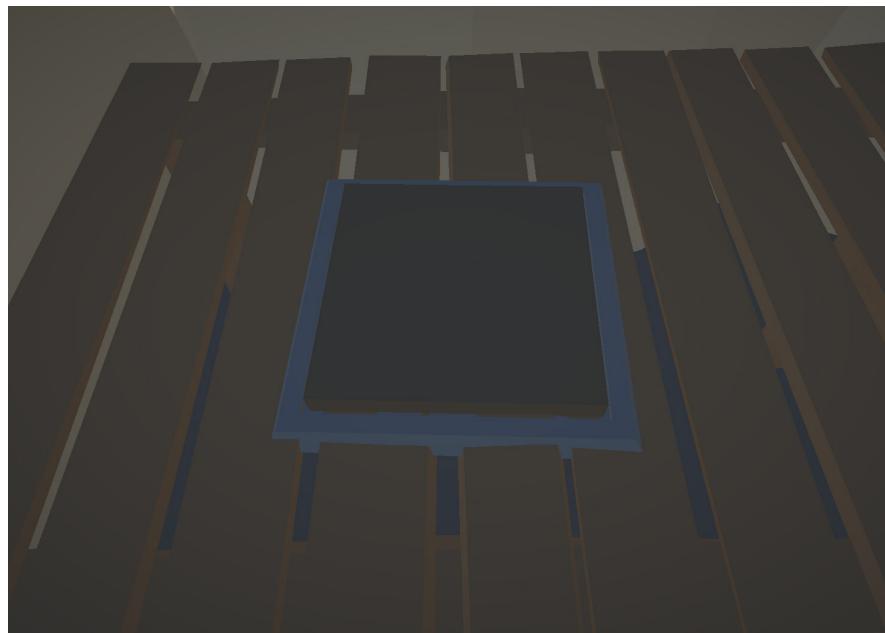


Abbildung 59.: Druckplatte, (Quelle: eigene Darstellung)

“Abbildung 59” zeigt eine Druckplatte, wie sie in der Spielwelt eingebaut wurde. Der hölzerne Kern stellt den Sensor dar, der eingedrückt wird, sobald das Gewicht des Chronologen auf ihm lastet. Jede Druckplatte enthält ein Partikelsystem, wie es in “Abschnitt 4.5.1: Partikelsysteme” bereits erklärt wurde.

4.6.2. Hebel



Abbildung 60.: Hebel, (Quelle: eigene Darstellung)

Derzeit sind Hebel, wie sie in “Abbildung 60” zu sehen sind, in der Spielwelt verbaut. Sobald der Chronologe einen Hebel betätigt, fährt der Hebelstab nach oben und signalisiert so dem Spieler, dass er aktiviert wurde. “Abbildung 61” zeigt einen aktivierte Hebel.



Abbildung 61.: Mit Hebel wurde interagiert, (Quelle: eigene Darstellung)

Der Hebel bekommt ein separates Interface an der Seite, auf dem zwei Lampen zu sehen sind (vgl. “Abbildung 62”). Sobald die obere grüne Lampe leuchtet, ist der Hebel aktiviert. Leuchtet die untere Lampe rot, so ist er deaktiviert. So erfährt der Spieler zusätzlich, wann ein Hebel aktiviert wurde.

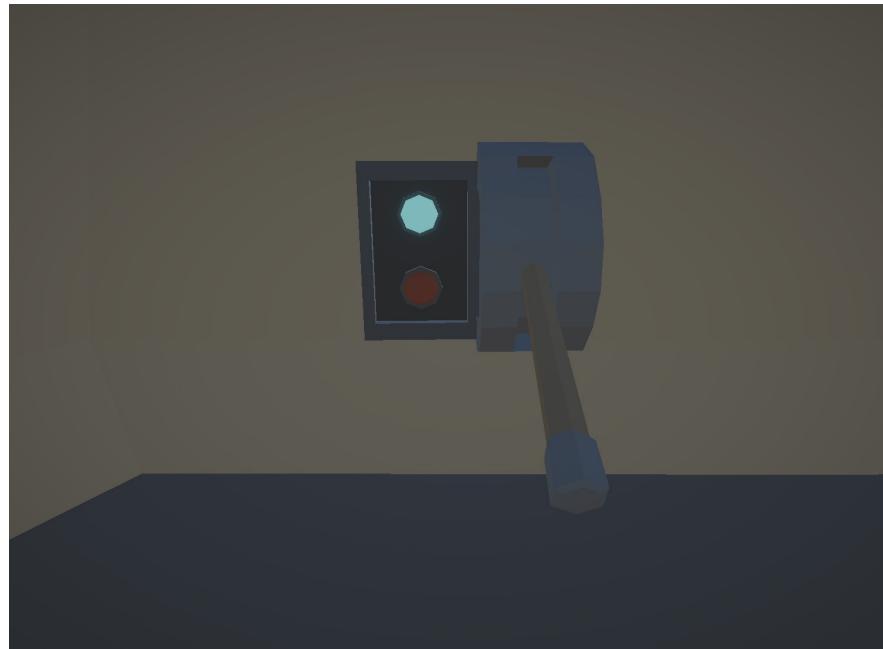


Abbildung 62.: Hebel mit Lampeninterface, (Quelle: eigene Darstellung)

4.6.3. Ventil

Die Abbildungen 63, 64 und 65 zeigen drei unterschiedliche Versionen eines Ventils, mit denen der Spieler interagieren kann.



Abbildung 63.: Ventil Version 1, (Quelle: eigene Darstellung)

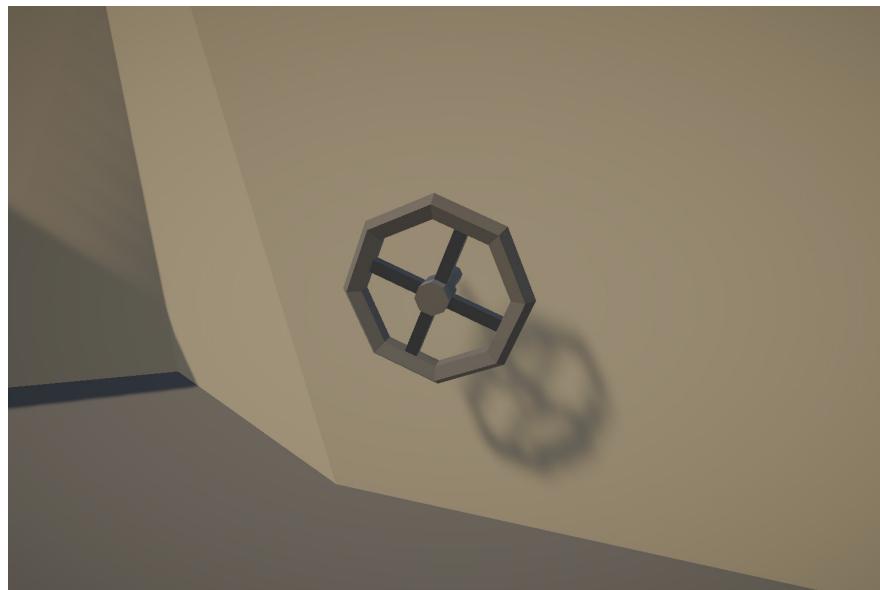


Abbildung 64.: Ventil Version 2, (Quelle: eigene Darstellung)



Abbildung 65.: Ventil Version 3, (Quelle: eigene Darstellung)

Wie bereits in “Kapitel 3.10.1.3: Ventil” gezeigt, muss der aktive Zustand des Ventils nicht dauerhaft gehalten werden. Ein Ventil ist entweder dauerhaft aktiv oder dauerhaft inaktiv.

4.6.4. Stützen

“Abbildung 66” zeigt die Zeichnung einer Stütze, die der Spieler tragen muss und auf welcher er klettern kann. Diese dient als ein erster Entwurfsversuch für die im Level eingebaute Stütze, die in “Abbildung 32” durch einen gestreckten Würfel dargestellt wird. Jede Stütze hat Zahnräder an ihrer Oberfläche montiert, wodurch diese Stützen an laufende Zahnräder angeschlossen werden können, welche in späteren Levels zu finden sein werden.

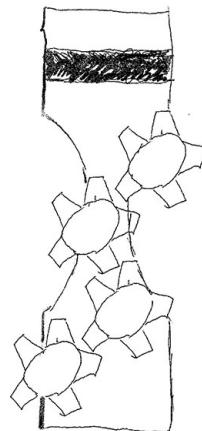


Abbildung 66.: Zeichnung einer einfachen Stütze mit Zahnrädern, (Quelle: eigene Darstellung)

In “Abbildung 67” kann diese Anreihung besser betrachtet werden.

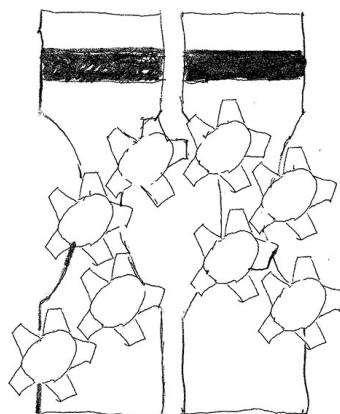


Abbildung 67.: Zeichnung einer Reihe an Stützen angeschlossen an Mechanismus, (Quelle: eigene Darstellung)

Die erste Stütze wird an den Beginn des Mechanismus ‘, ein aus der Wand ragendes Zahnrad, gestellt, sodass das Drehmoment des entsprechenden Zahnrades die Zahn-

räder an der Stütze drehen lässt. Anschließend muss eine zweite oder dritte Stütze an die erste Stütze gestellt werden, damit das Drehmoment weitergegeben werden kann. Diese Stützen werden verteilt in einem Raum zu finden sein. Die Aufgabe des Spielers ist es dann, diese in die richtige Reihenfolge zu stellen.

4.6.4.1. Treppe

Die Stütze dient nicht als einziges tragbares Objekt in der Spielwelt, sondern es können auch andere Objekte sein. Als Beispiel für das in der Spielwelt integrierte tragbare Objekt dient hier eine Treppe, wie sie in „Abbildung 68“ zu sehen ist.



Abbildung 68.: tragbares Treppenelement, (Quelle: eigene Darstellung)

Dieses Treppenstück ist das Endstück einer Treppe, die der Spieler vervollständigen muss, um darüber auf bestimmte Objekte zu gelangen.

4.6.5. Portal Steuereinheit

In der folgenden „Abbildung 69“ ist ein erster Entwurf des Computers des Chronologen zu sehen. Das Interface, mit dem der Chronologe interagieren wird, besteht aus Kurbeln und vereinzelten Zahnrädern. Der Wissenschaftler kann über den kleinen Bildschirm seine Eingaben überprüfen und dem visuellen UI des Computers folgen. Auf der Rückseite des Computers befinden sich weitere Zahnräder, die die Eingaben an die Dienste des Computers übermittelt.

Der Computer dient nicht nur der Aufnahme und Speicherung von Notizen, sondern auch dem Bedienen der Erfindung des Chronologen. Über diesen Computer kann der Spieler im Labor seine nächsten Ziele auswählen.



Abbildung 69.: Computer des Chronologen, erstes Konzept, (Quelle: Stable Diffusion)

Über das Portal, das in „Abbildung 70“ als ein erster Entwurf skizziert ist, kann der Spieler seine Entdeckungsreisen beginnen.



Abbildung 70.: Portal der Maschine, erstes Konzept, (Quelle: Stable Diffusion)

4.6.6. Lore



Abbildung 71.: Lore, (Quelle: eigene Darstellung)

Wie in "Abbildung 71" abgebildet, kann der Spieler Loren in der Spielwelt verschieben. Es sind nicht nur Loren die der Spieler verschieben kann, sondern es sind alle Objekte mit einem entsprechendem Tooltip. Die Lore aus der oben genannten Abbildung kann vom Spieler nach Abschließen des ersten Kapitels an das Gerüst geschoben werden, dadurch hat er die Möglichkeit das Gerüst hinaufzuklettern.

4.7. Gebrauchsgegenstände

Im folgenden Kapitel werden visuelle Konzepte der Gebrauchsgegenständen vorgestellt. Es wird darauf eingegangen, welche Gegenstände bereits in ihrer Form umge-

setzt wurden und welche nur auf konzeptioneller Ebene existieren.

4.7.1. Steuereinheit

Die Steuereinheit ist in zwei Teile zu unterteilen. Zum einen ist sie Teil der Maschine, die der Chronologe erfunden hat. Dabei besteht der stationäre Teil der Maschine aus den Abbildungen 69 und 70: Portal der Maschine, erstes Konzept, (Quelle: Stable Diffusion). Der Chronologe hat das tragbare Gegenstück dazu immer bei sich.



Abbildung 72.: Steuereinheit, (Quelle: eigene Darstellung)

Sobald der Spieler die Steuereinheit öffnet, sieht er das Gerät aus "Abbildung 72".

Diese Steuereinheit ist in fünf Abschnitte eingeteilt.

4.7.1.1. 1. Abschnitt, Zahnrad

Jede Zacke des Zahnrades steht für das Ich aus den anderen Zeitlinien, die der Spieler zur Verfügung hat. Dabei wird jede Zacke des Zahnrades nach der Farbe des verfügbaren Ichs markiert. Die Farben dienen der Unterscheidbarkeit für den Spieler.

4.7.1.2. 2. Abschnitt, aktive Zacke

Sobald der Spieler ein Ich des Chronologen über das Rad auswählt, so dreht er das Zahnräder entweder mit oder gegen den Uhrzeigersinn. Sobald eine Zacke des Zahnrades an die Position der Markierung 2 gedreht wird, wird der darin enthaltene Chronologe ausgewählt.

4.7.1.3. 3. Abschnitt, Status des Chronologen

Sobald über das Zahnrad ein Chronologe ausgewählt wurde, wird im Feld der Markierung 3, der Status des Chronologen, genauer gesagt seine einhergehende Zeitlinie im Zeitkontinuum, angezeigt. Dabei geht der Betrachtungswinkel der Steuereinheit vom derzeitig gesteuerten Chronologen aus.

Gesplittete Zeitlinie: „Abbildung 73“ zeigt den Status einer gesplitteten Zeitlinie. Das bedeutet, der Spieler hat von seinem jetzigen Standpunkt in der Zeitlinien Hierarchie bereits einen „*Split*“ zu diesem Ich des Chronologen absolviert und ist dann auf seinen jetzigen „*zurückgemergt*“. Zu diesem Chronologen kann der Spieler erst wieder wechseln, sobald die Zeitlinie des Chronologen nicht mehr aktiv ist.



Abbildung 73.: gesplitteter Zeitlinien Status, (Quelle: eigene Darstellung)

Aktuelle Zeitlinie: „Abbildung 74“ zeigt das Statussymbol der Zeitlinie, wenn eine Zacke an das Auswahlfeld gedreht wird, an dem der derzeitige Chronologe farblich markiert wurde.



Abbildung 74.: aktueller Zeitlinien Status, (Quelle: eigene Darstellung)

Zeitlinie zum Zurückmergen: Sobald eine Zeitlinie den Status aus „Abbildung 75“ besitzt, kann der Spieler auf dieses Ich des Chronologen „*zurückmergen*“. Dabei springt er in der Zeit auf den letzten Zeitpunkt, die diese Zeitlinie erlebt hat, zurück. Nach der Auswahl dieses Ichs startet der „*Mergvorgang*“.



Abbildung 75.: Zeitlinienstatus zum Zurückmergen, (Quelle: eigene Darstellung)

Verfügbare Zeitlinie: Erscheint auf dem Feld keins der drei Symbole aus den Abbildungen 73, 74 und 75 so ist dieses Ich frei und kann für einen “*Splitvorgang*” ausgewählt werden.

4.7.1.4. 4. Abschnitt, Name des Chronologen

Auf dem oberen Anzeigeelement erscheint der Name des ausgewählten Chronologen. Zusätzlich zu der Position des Zahnrades kann der Spieler hierdurch feststellen, dass er das richtige Ich des Chronologen ausgewählt hat.

4.7.1.5. 5. Abschnitt, Auswahl bestätigen

Über den Knopf auf dem Zahnrad kann der Spieler seine Auswahl bestätigen. Der Knopf verändert in Abhängigkeit der Auswahl seine Farbe. Die Farben weisen den Spieler darauf hin, ob er seine Auswahl bestätigen kann oder ob er gerade einen nicht verfügbaren Chronologen auswählen möchte. Der Knopf leuchtet grün, sobald die Auswahl auf einen freien Chronologen gefallen ist oder er auf einen Chronologen “zurückmergen” möchte, oder rot, sobald er sein derzeitiges Ich oder ein “gesplittetes” Ich ausgewählt hat.

4.7.2. Datenleser

Wie bereits in “Kapitel 3.9.2: Datenleser” gezeigt, kann der Spieler über den Datenleser wichtige Informationen erhalten. Im derzeitigen Prototyp ist der Datenleser ein skaliertes Würfel. Dieser dient lediglich als ein Platzhalter für das eigentliche 3D-Modell des Datenlesers.

Die Abbildungen 76 und 77 zeigen einen ersten Konzeptentwurf, wie der spätere Datenleser aussehen kann. Der Datenleser enthält ähnlich wie der Computer des Chronologen mechanische Elemente, über die die Eingaben erfolgen können. Diese Eingabeelemente wie Zahnräder und Kurbel befinden sich auf der Rückseite des Datenlesers. Über die Vorderseite erhält der Spieler nun das visuelle Interface, das bereits in “Kapitel 4.4.9: Datenleser” vorgestellt wurde.

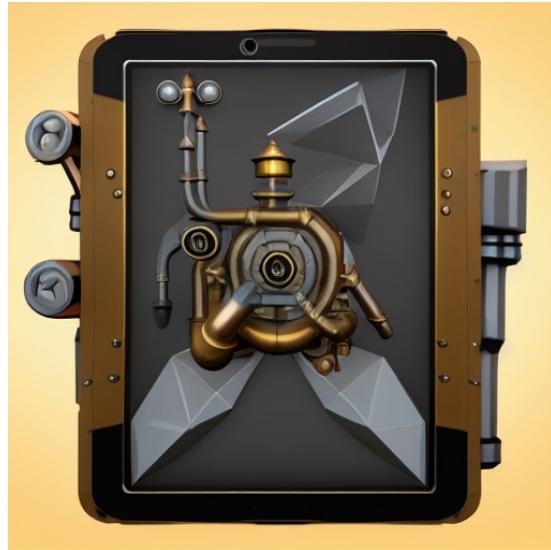


Abbildung 76.: Vorderseite Datenleser, Konzeptzeichnung, (Quelle: Stable Diffusion)



Abbildung 77.: Rückseite Datenleser, Konzeptzeichnung, (Quelle: Stable Diffusion)

4.7.3. Pager

Die “Abbildung 78” zeigt ein erstes Konzept des Pagers, den der Spieler im späteren Verlauf des Spiels findet und erhält.

Über die mit der 1 markierten haptischen Interfaces aus joystickähnlichen Elementen und mechanischen Tasten kann der Chronologe die Ideen und Wünsche des Spielers in das Gerät eingeben. Auf dem kleinen Bildschirm, der mit der 2 markiert ist, kann der Spieler seine Eingabe kontrollieren, ehe er diese mit einem Knopfdruck abschickt. Der Knopf ist das Element, das mit der Ziffer 3 markiert wurde.

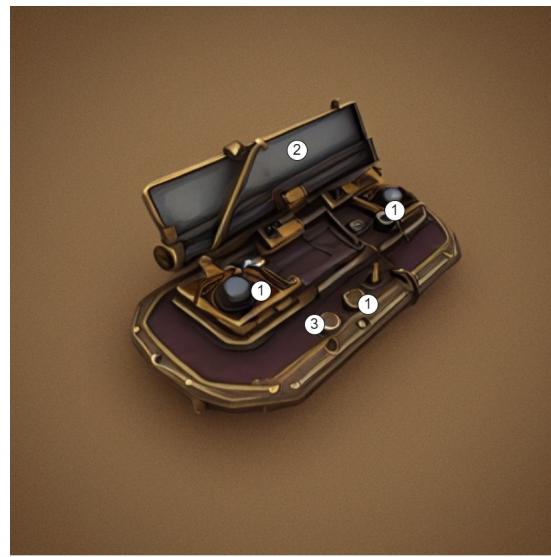


Abbildung 78.: Pager, (Quelle: Stable Diffusion)

4.7.4. Imitator

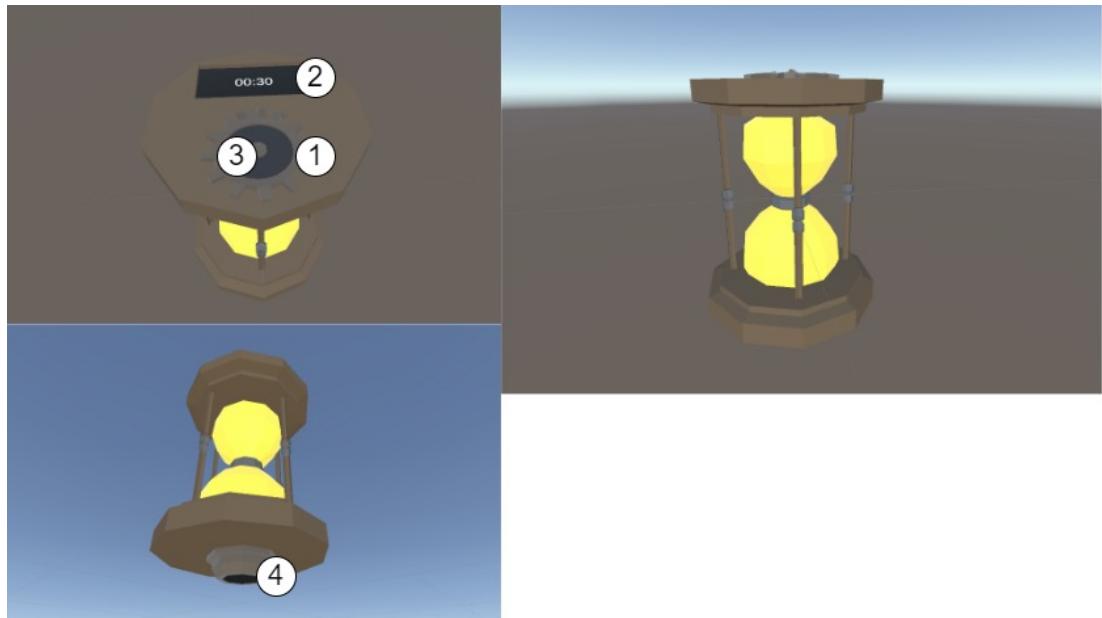


Abbildung 79.: Imitator, (Quelle: eigene Darstellung)

Wie in "Abschnitt 3.9.3: Imitator" angesprochen, erhält der Spieler nach Erreichen eines bestimmten Levels einen Imitator. Die Umsetzung des Imitators ist in "Abbildung 79" dargestellt.

Der Imitator ist in vier Abschnitte unterteilt, die in den folgenden Unterkapiteln dargestellt werden.

4.7.4.1. 1. Abschnitt, Auswahlrad

Über das Auswahlzahnrad kann der Chronologe die Zeit einstellen, wie lange die Kopie von ihm an der gesetzten Stelle stehen soll. Durch das Drehen des Zahnrades wird ein Mechanismus in der Sanduhr freigesetzt, die diese Zeit einstellt.

4.7.4.2. 2. Abschnitt, Bildschirm

Über den Bildschirm, der auf der Oberseite des Imitators zu finden ist, kann der Spieler den über das Zahnrad gestellten Timer einsehen. Er hat die Wahlmöglichkeit zwischen 5, 10, 15, 30, 45, 60 und 90 Sekunden, die die Kopie des derzeitig aktiven Ichs des Chronologen an der derzeitigen Stelle imitiert.

4.7.4.3. 3. Abschnitt, Auswahl bestätigen

Über den Knopf in der Mitte des Zahnrades kann der Spieler seine Auswahl bestätigen.

4.7.4.4. 4. Abschnitt, Erstellung der Kopie

Nachdem der Spieler seine Auswahl bestätigt hat, legt der Chronologe den Imitator an seine derzeitige Position. Dadurch wird die Kopie des derzeitigen Ichs in die Spielwelt gesetzt. Im Anschluss an den erfolgreichen Kopiervorgang hebt der Chronologe den Imitator wieder auf und verstaut ihn in seinen Taschen. Der Spieler erhält, wie in Abschnitt 4.4.2 Timer gezeigt, unterhalb des Timers eine Ansicht, wie lange die Kopie noch aktiv ist.

4.8. Menü

Das Spiel besitzt zwei Arten von Menüs (Start und Pause). Das Startmenü erscheint immer nach dem Start der Anwendung, das Pausemenü kann in jedem Level des Spiels geöffnet werden.

4.8.1. Pause

Der Spieler kann das Spiel über die Escape (ESC)-Taste pausieren. Dadurch öffnet sich der Datenleser des Chronologen und das Pausemenü erscheint auf dem Monitor des Datenlesers, worüber der Spieler Informationen über das Spiel nachlesen kann.



Abbildung 80.: Pausemenü, (Quelle: eigene Darstellung)

In “Abbildung 80” ist ein Mockup des Pausemenüs zusehen, über das der Spieler das Spiel beenden, bestimmte Einstellungen ändern und das Spiel wieder fortsetzen kann.

4.8.2. Start

Sobald das Spiel geladen ist, erscheint das in “Abbildung 81” zu sehende Startmenü des Spiels. Über dieses kann der Spieler seinen aktuellen Spielstand fortsetzen, ein neues Spiel starten, die Einstellungen öffnen oder das Spiel wieder verlassen.



Abbildung 81.: Startmenü, (Quelle: eigene Darstellung)

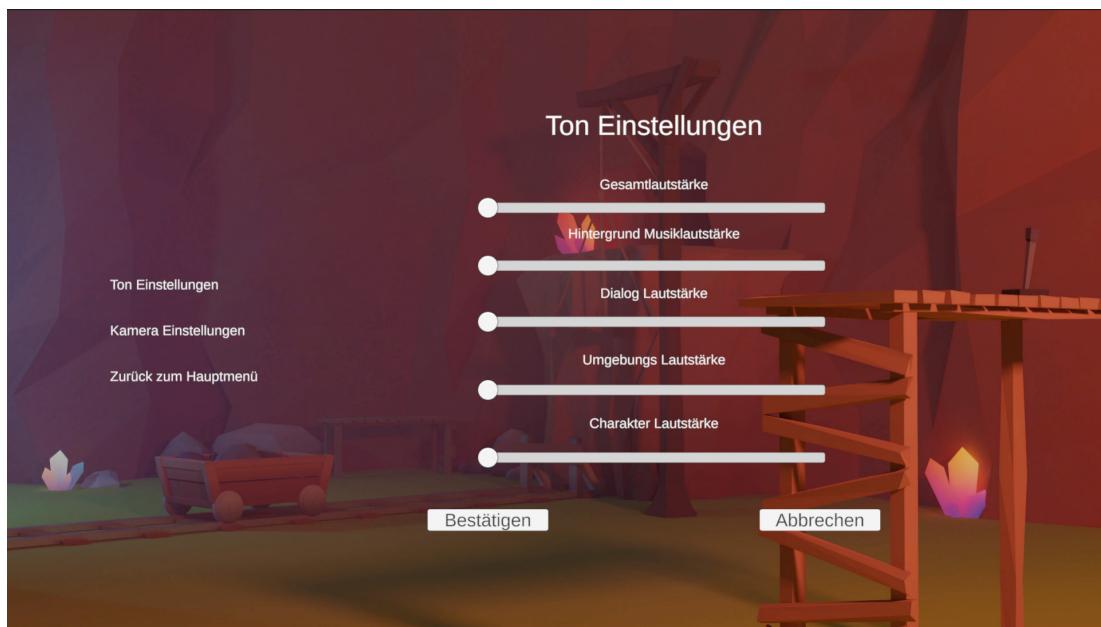


Abbildung 82.: Toneinstellungen, (Quelle: eigene Darstellung)

Im derzeitigen Prototyp kann der Spieler über das Einstellungsmenü in den Abbildungen 82 und 83 Ton- und Kameraeinstellungen tätigen. Über dieses Menü werden weitere Einstellungen getätigkt werden können.

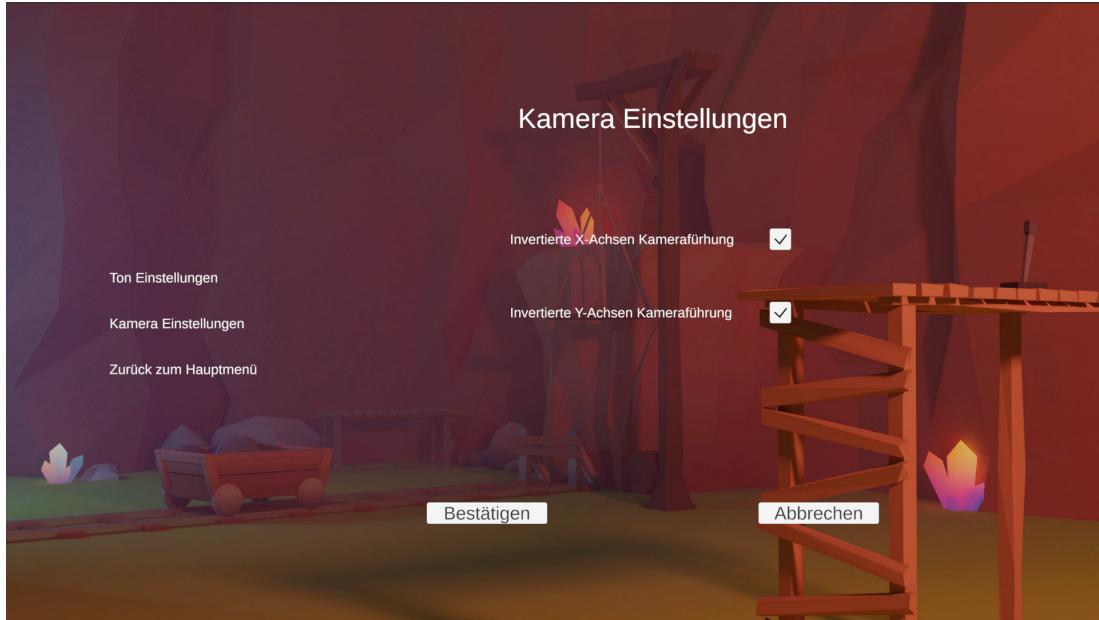


Abbildung 83.: Kameraeinstellungen, (Quelle: eigene Darstellung)

4.9. Leveledesign

Im Folgenden wird auf die umgesetzten Level des Prototyps eingegangen.

4.9.1. Tutorial

Das erste Level im Prototypen ist das in „Abbildung 84“ gezeigte Tutorial. Es dient als Einführung in das Spiel.

Das Setting des Tutorials ist ein verlassener Steinbruch, der nach dem Start der Maschine des Chronologen verlassen werden musste, weil die Arbeiter Angst vor einer Zeitlinienkonvergenz (vgl. „Kapitel 3.5.1: Zeitlinienkonvergenz“) hatten. Der Spieler muss versuchen zu verstehen, wie er den Steinbruch für seine ehemaligen Bewohner wieder sicher gestalten und einen Schutz gegen eine Konvergenz aufbauen kann. Dafür muss der Spieler das Level gründlich erforschen. In der Spielwelt sind Hinweise darauf versteckt, die der Spieler finden muss.

Nachdem der Spieler das erste Kapitel erfolgreich abschließen konnte, ist es ihm möglich erneut in den Steinbruch zurückzureisen, um ihn weiter zu erkunden. Dadurch hat er die Möglichkeit andere und neue Elemente zu entdecken, wie versteckte Gegenstände oder neue Wege zu finden um das Level abzuschließen.

Diese Hinweise sind jedoch in dieser Version des Steinbruchs nicht integriert. Diese Spielwelt dient in diesem Prototyp als ein Tutorial, in dem der Hauptfokus auf dem Erlernen der Spielmechanik liegt. Im Vergleich zu einer fertigen Version des Spiels hat

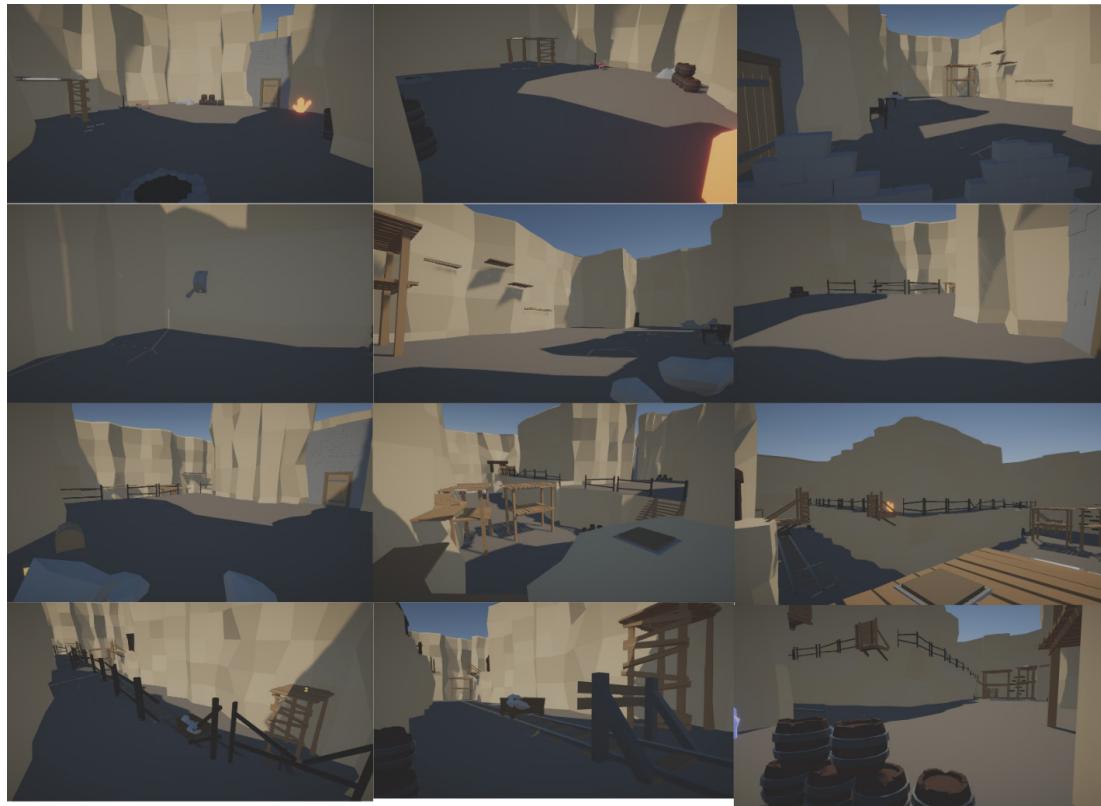


Abbildung 84.: Collage Tutorial, (Quelle: eigene Darstellung)

der Spieler in diesem Prototyp die Auswahl zwischen zwei weiteren Ichs des Chronologen. In einer fertigen Version wird der Spieler in seinem ersten Level nur mit einer weiteren Person des Chronologen starten. Dieser Steinbruch ist dabei ein zweites Level und nicht mehr das Tutorial. Diese Spielwelt wurde deshalb als Tutorial ausgewählt, da sie zum einen dafür im Gamedesign Workshop konzipiert wurde und so die Spielmechanik verstanden werden konnte und zum anderen war sie bereits umgesetzt und konnte erneut verwendet werden. Ein neues Szenario für das Tutorial zu entwickeln war aus diesem Grund nicht sinnvoll, da der Fokus auch darauf bestand ein weiteres Szenario zu entwickeln, in dem neuartige Rätsel zu lösen sind.

4.9.2. Labor



Abbildung 85.: Collage Labor, (Quelle: eigene Darstellung)

Dieser Prototyp umfasst ein zweites Level, welches Labor genannt wird. Dieses Level ist das letzte Level, um Kapitel 1 abzuschließen. Durch das Bestehen dieses Levels findet der Spieler einen Weg zurück in das Labor des Chronologen.

Wie in "Abbildung 85" zu sehen ist, enthält das Laborlevel zusätzlich zu seinem Labor zwei weitere Räume. Der Spieler muss die Rätsel beider Räume lösen, um den Weg in das Labor zu öffnen. Sobald der Spieler in die Spielwelt eintritt, macht er sich auf Anweisung des Ichs des Chronologen aus der Zeitlinie, in der sich das Labor befindet,

auf den Weg zum Laboreingang. Im Eingangsraum zum Labor muss der Spieler und das Ich des Chronologen allerdings feststellen, dass die Eingangstür verschlossen ist und auch kein Strom angeschaltet wurde. Der Spieler muss nun in den Raum der zwei Kräne gehen, da er dort den Strom anstellen kann. Nachdem der Strom angestellt wurde, betritt der Spieler das Labor und findet einen Weg zurück in sein eigenes Labor.

Nach dem Betreten des Labors fällt dem Chronologen die ungefähre Ähnlichkeit zu seinem eigenen Labor auf. Es muss sich also um eine andere Version seines Labors handeln. Die Ähnlichkeit der Labore beruht darauf, dass durch die Zeitlinien Konvergenz am Anfang des Spiels Teile des Labors vom Chronologen in dieses Labor aus einer anderen Zeitlinie übergegangen sind. Darunter ist ein Teil der Maschine, durch welche der Chronologe in eine andere Zeitlinie reisen kann. Durch das gefundene Teil seiner Maschine kommt er in sein eigentliches Labor zurück. Es befindet sich allerdings nicht mehr in der Zeitlinie des Chronologen, sondern an einem Ort ohne Raum und Zeit.

Der genaue Grund, wieso dieses Maschinenteil den Chronologen in sein Labor zurückführt, wird für den Spieler in der Handlung des fertigen Spiels behandelt. Sie wird in diesem Prototyp nicht erzählt.

Das Ich aus dieser Zeitlinie ist selbst kein Chronologe geworden. Allerdings forscht er ebenso wie der Chronologe in einem Labor in einer anderen Disziplin. Das Ich war in seiner Vergangenheit einmal in diesem Labor und wusste daher, wo sich der Eingang befindet und wo sich der Strom anstellen lässt.

Das gesamte Level besteht aus einem Höhlen- und Laborkomplex, welche mit einer alten Kanalisation verbunden sind. Durch die alten Röhren der Kanalisation kann der Spieler die einzelnen Räume der Spielwelt betreten.

Der Spieler hat für das Meistern des zweiten Levels zwei weitere Ichs zur Verfügung, da es sich im Vergleich zum Tutorial an einem späteren Zeitpunkt der Handlung befindet.

4.9.3. Labor des Chronologen

“Abbildung 86” gibt einen ersten Eindruck, wie das Labor des Chronologen aussehen wird. Zusätzlich zu dieser Collage besitzt das Labor, das aus “Kapitel 4.6.5: Portal Steuereinheit” beschriebene Portal und seinen Computer. Über dieses Labor kann der Spieler wie bereits beschrieben seine zuvor erlebten Level erneut besuchen, ebenso seine kommenden Reisen starten.



Abbildung 86.: Collage Labor des Chronologen, (Quelle: Plane (o. D.) (rechts oben), Stable Diffusion (rest))

Das Labor befindet sich an einem Ort ohne Raum und Zeit. Durch das erstmalige Benutzen der Zeitlinien-Reisemaschine erfolgte eine Zeitlinien Konvergenz, wodurch Teile des Labors in eine parallele Zeitlinie katapultiert wurden. Es entstehen Kopien des Labors, auf welche der Spieler am Ende des ersten Kapitels stößt. Das Labor als Ganzes blieb unbeschädigt, jedoch befindet es sich nicht mehr in der Zeitlinie des Chronologen.

5. Umsetzung des Prototyps

Im Folgenden wird die Umsetzung des entwickelten Prototyps vorgestellt. Dabei wird zunächst auf die verwendeten Technologien eingegangen. Anschließend erfolgt eine Analyse des alten Prototyps, auf deren Ergebnisse die Entwicklung einer neuen Version des Prototyps aufbaut. Im Anschluss wird das überarbeitete Replay und Aufnahmesystem des Prototyps vorgestellt. Zum Schluss wird das integrierte detailliert Soundsystem erklärt.

5.1. Verwendete Technologien

In der kommenden Aufzählung werden die verwendeten Unity Packages vorgestellt. Im Package Manager des im Anhang beiliegender Version der Arbeit befinden sich mehr Packages im Registry, als hier aufgezählt werden. Das liegt daran, dass die hier genannten Pakete genutzt und die zusätzlichen Pakete im Registry zum Testen und für den Ausblick eingebunden wurden.

5.1.1. Unity Version 2021.3.16f1

Der Prototyp wurde in der Version 2021.3.16f1 Long Term Support (LTS) umgesetzt. Diese Version war zum Startzeitpunkt der Thesis die aktuellste LTS der 2021er-Version des Unity Editors. In ihr ist es für den Ausblick ebenfalls möglich, den Experimental-Release von DOTS einzubinden. (vgl. Gram (2021))

5.1.2. Blender 3.3.1

Blender dient in der Umsetzung als Bearbeitungstool von 3D Objekten. Aufgrund der Tatsache, dass Blender (.blend) Dateien direkt in Unity integriert werden können, und nicht als Filmbox (FBX) exportiert und importiert werden müssen (vgl. Technologies (o. D.b)). Als die Umsetzung des Prototyps begann, war die Version 3.3.1 von Blender die aktuellste, die heruntergeladen wurde.

5.1.3. Stable Diffusion

Stable Diffusion ist ein Text-zu-Bild Generator, welcher durch Deep-Learning Algorithmen und die Eingabe von Textanweisungen Bilder erzeugen kann. Diese Technologie kann lokal auf jedem Rechner installiert werden. Sie wurde benutzt, um Bilder für die Konzeption zu generieren (vgl. Katzlberger (2022)). Die KI wird durch den Quellenverweis “Stable Diffusion” gekennzeichnet. Sie ist erreichbar unter der im folgenden zitierten Verweis erreichbar (Sygil-Dev (o. D.)).

5.1.4. Mixamo

Die im Prototyp verwendeten Animationen des Spielercharakters stammen alle aus der Bibliothek von Mixamo. Mixamo ist in der Lage, importierte FBX Modelle zu riggen. Dabei erhält das Objekt ein Skelett, welches animiert werden kann. Das dabei entstandene Skelett und die integrierten Animationen werden in diesem Prototyp verwendet.

5.1.5. Envato Elements

Über ein Abo bei Envato Elements konnten atmosphärische Sounds und Sounds, die der Spielercharakter im Spiel von sich gibt, heruntergeladen und eingesetzt werden.

5.1.6. Cinemachine 2.8.9.

Die Cinemachine wird für die Kamerafahrt des Spiels verwendet. Durch sie ist es möglich, eine Third-Person-Sicht in das Spiel zu haben. Zusätzlich ist es möglich verschiedene virtuelle Kameras zu besitzen, durch welche zwischen Third- und First-Person Sicht des Spielers gewechselt werden kann. Durch sie ist es möglich, dass der Spieler den Chronologen in einer Third-Person Ansicht steuert und in einer First-Person-Ansicht seine Gebrauchsgegenstände sieht. Dadurch hat das Spiel eine immersive Wirkung.

5.1.7. Filmbox Exporter 4.1.3.

Da der Prototyp dieser Arbeit auf einem Prototyp aus dem Gamedesign Workshop aufbaut, für den bereits einzelne 3D Modelle erstellt und eingebaut wurden, existieren schon einige 3D Modelle in der Bibliothek. Allerdings reichen diese nicht vollständig aus, um den Bedarf an 3D Objekten zu decken. Daher mussten Bibliotheken hinzugefügt werden. Diese werden in “Kapitel 5.1.10: Unity Prefab Assets” aufgezählt. Diese 3D-Model Packages enthielten teilweise keine FBX Dateien, weshalb über den

FBX Exporter diese aus Unity Prefabs erzeugt wurden. Das war insofern notwendig, da nicht alle 3D Modelle dem Spielanspruch genügten und darüber hinaus nicht editierbar waren. Durch das Erstellen von FBX Dateien aus den Prefabs, konnten diese in Blender importiert und bearbeitet werden. Um sie letztlich benutzen zu können, wurden diese wiederum im Unity Projekt abgespeichert und importiert.

5.1.8. TextMeshPro 3.0.6.

TextMeshPro wird in der Kombination mit dem veralteten "Unity UI" und "legacy Text Mesh" als Darstellung und Text verwendet. Durch TextMeshPro werden alle UI Elemente auf einer Canvas angezeigt. Das ermöglicht unter anderem, dass in einer späteren Version auch bspw. Videos im Datenleser eingebunden werden können.

5.1.9. Universal Render Pipeline 12.1.8.

Im Unity Editor hat man die Möglichkeit eine Render Pipeline einzubinden oder nicht. In diesem Fall wurde entschieden, die Universal Render Pipeline (URP) einzubinden. Dadurch besteht die Möglichkeit der in Kamera ein Post-Processing einzubinden, wodurch spezielle Effekte und das Rendering der Spielwelt verändert werden kann. Im Abschnitt 3.5.7 Einflüsse auf die Spielwelt wurden Effekte erwähnt, die auf das Visuelle der Spielwelt wirken werden. Diese Effekte werden zukünftig über die URP eingebunden und visualisiert.

5.1.10. Unity Prefab Assets

Zum Start der Entwicklung dieses Prototyps gab es bereits einige erstellte 3D Modelle, die in die Szenen eingebaut wurden. Die im Gamedesign Workshop erstellen 3D Modell haben allerdings nicht ausgereicht. Es wurden aus anderen Packages 3D Modell importiert:

- Fissure: Low Poly Caverns 1.1. Lands (2020)
- Low Poly Rock Models 1.1. Glacier (2018)
- Low Poly Underground Mega Pack 1.0. Mercuzot (2020)
- Steampunk Low Poly Pack 1.0. Creations (2018)
- Victorian Low Poly Pack 1.0. Creations (2021)
- Unity Particle Pack 1.6. Technologies (2021)

5.2. Ausgangssituation

Wie in “Kapitel 1: Einleitung” beschrieben, existiert ein erster kleiner Prototyp des Spiels aus dem Gamedesign Workshop. Dieser ist jedoch sehr anfällig für Fehler und noch unstrukturiert. Das liegt zum einen daran, dass vier verschiedene Personen mit einem unterschiedlichen Wissensstand in Unity an diesem Prototyp gearbeitet haben. Zum anderen, dass im grundlegenden Aufbau der Mechanik von einem falschen Ausgangspunkt ausgegangen wurde. Zu Beginn der Entwicklung wurde das Erstellen von verschiedenen Version des Chronologen als Ausgangspunkt angesehen, was zu Problemen führte. Zusätzlich war ein Teil des damaligen Konzepts eine Ursache dafür, dass das Aufbauen des Graphenbaums zu Fehlern führte. Dabei war die Idee, dass der Spieler eine bereits erstellte Zeitlinie erneut “*Splitten*” kann, um sie zu ändern. In dem überarbeiteten Konzept ist diese Möglichkeit nun nicht mehr möglich. Im Folgenden werden auf die Aspekte eingegangen, die von ihrer Grundidee wichtig für die Umsetzung sind. Ebenso werden die Aspekte beschrieben, die es zu überarbeiten galt.

Die Grundlagen des Entwicklungskonzeptes entstanden dabei aus dem Zusammenschluss der Artikel “Creating a Replay System in Unity” von Teddy Engel (vgl. Engel (2020)) und “Developing Your Own Replay System” von Cyrille Wagner (vgl. Wagner (2004)).

5.2.1. Übernommene Aspekte

5.2.1.1. Aufbau eines Baumes aus Zeitlinien

Der Spieler erstellt durch die Auswahl eines Ichs des Chronologen eine Zeitlinie, auf welcher sich dieses Ich zeitlich bewegen wird. Dazu wird zunächst eine Hauptwurzel des Baumes benötigt, welcher im folgenden Codeausschnitt 5.1 zu sehen ist:

```

1 public class GraphController : Controller
2 {
3     [SerializeField]
4     private Timeline rootTimeline;
5     [SerializeField]
6     private Timeline currentTimeline;
7     [SerializeField]
8     private List<Timeline> timelinesToHandle = new List<Timeline>();
9 ...
10 }
```

Codeausschnitt 5.1: Ausschnitt aus GraphController.cs aus dem alten Prototyp

Sowohl diese Hauptzeitlinie als auch die angegliederten Zeitlinien besitzen eine Liste an angegliederten Zeitlinien und eine Referenz auf die Zeitlinie, an welche sie angegliedert wurden. Zudem erhalten diese Zeitlinien ein Level, auf welcher Ebene der Verschachtelung sie sich befinden (vgl.: folgender Codeausschnitt 5.2 und “Abbildung 87”).

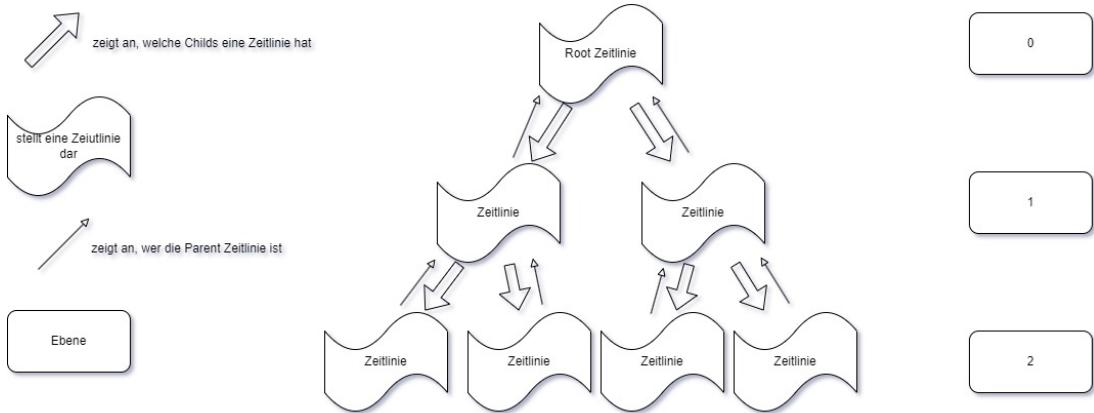


Abbildung 87.: Schaubild zu Codeausschnitt „Ausschnitt aus Timeline aus dem alten Prototyp“, (Quelle: eigene Darstellung)

```

1 public class Timeline
2 {
3     [SerializeField]
4     private int _level = 0;
5     [SerializeField]
6     private List<Timeline> _children = new List<Timeline>();
7     [SerializeField]
8     private float _startTimestamp;
9     [SerializeField]
10    private CharacterData _player;
11    [SerializeField]
12    private Timeline _parent = null;
13    [SerializeField]
14    private Shadow _ghost = null;
15    [SerializeField]
16    private string ID;
17    [SerializeField]
18    private float[] _startPosition = new float[3];
19    private SavePlayerData _data = null;
20    ...
21    public void InsertChild(Timeline child)
22    {
23        if(!_children.Contains(child))
24        {
25            _children.Add(child);
26        }
27        else
28        {
29            _children.Insert(_children.FindIndex(a => a.GetId() == child.GetId()),
30            child);
31        }
32    }
33 }
```

Codeauschnitt 5.2: Ausschnitt aus Timeline.cs aus dem alten Prototyp

In „Abbildung 23“ wird ein solcher Baum gezeigt. Die schwarze Zeitlinie ist dabei die `_rootTimeline` aus „Kapitel 5.2.1.1: Aufbau eines Baumes aus Zeitlinien“. Diese

Zeitlinie besitzt das Level 0. Umso weiter die Zeitlinien ineinander verschachtelt werden, so steigt das Level der Zeitlinien. Sobald der Spieler nun das erste Mal “*splittet*”, wird diese neue Zeitlinie (aus der Abbildung die blaue Zeitlinie) über die Methode *InsertChild()* an die *_rootTimeline* angegliedert. Zusätzlich erhält dieses Child auch eine Referenz auf seinen Parent. Jedes angegliederte Child besitzt das Level des Parents plus 1: *_level = _parent.level + 1*. Das Level der blauen Zeitlinie beträgt nun 1. Erzeugt der Spieler viele parallele Zeitlinien, so wie die rote Zeitlinie aus “Abbildung 23”, so haben diese ebenfalls das Level 1. Hierbei wird nicht unterschieden, die wievielte Zeitlinie in den Children diese ist.

5.2.1.2. Welche Zeitlinien müssen nach einem Mergevorgang betrachtet werden?

Nach einem “Mergevorgang” wird die aktuelle Zeitlinie durch eine bereits existierende Zeitlinie ersetzt. Befindet sich der Spieler in “Abbildung 7” auf der blauen Zeitlinie, so wechselt er zum Beispiel auf die schwarze Zeitlinie zurück. Ist dieser Wechsel abgeschlossen, müssen die nach einem Split entstandenen rote und blaue Zeitlinie betrachtet werden.

```

1 public class GraphController
2 {
3     ...
4     private void HandleMerge(Player player)
5     {
6         float startPoint = -1;
7         Timeline timeline = GetTimelineOfNewPlayer(player.GetCharacterData());
8         if (timeline != null)
9         {
10             List<Timeline> childrenOfNewTimeline = timeline.GetChildren();
11             if (childrenOfNewTimeline.Count > 1)
12             {
13                 foreach (Timeline child in childrenOfNewTimeline)
14                 {
15                     if (child.GetId().Equals(currentTimeline.GetId()))
16                     {
17                         startPoint = child.GetStartTimestamp();
18                     }
19                 }
20             }
21             if (childrenOfNewTimeline.Count == 1)
22             {
23                 foreach (Timeline child in childrenOfNewTimeline)
24                 {
25                     startPoint = child.GetStartTimestamp();
26                 }
27             }
28             if (startPoint != -1)
29             {
30                 TimeController.Instance.SetGameTime(startPoint);
31                 _currentTimeline = timeline;
32             }
}

```

```

33     HandleRemoveOldCharacter(_rootTimeline);
34     timelinesToHandle = new List<Timeline>();
35     CheckForInteractions(_rootTimeline);
36 }
37 }
38 ...
39 }
```

Codeausschnitt 5.3: HandleMerge Methode aus dem alten Prototyp

Durch das vorangegangene “*Splitting*” wurden neue Verzweigungen in der `_rootTimeline` und in ihren Children hinzugefügt (vgl. “Kapitel 5.2.2.2: Undurchdachte Split und Merge Methoden”), welche nun durch eine rekursive Methode durchiteriert werden können. Folgende Methode wird wie im vorherigen Codeausschnitt 5.3 am Ende der `HandleMerge()` Methode aufgerufen (`CheckForInteractions()`). Es wird überprüft, welche Zeitlinien im Baum das gleiche oder ein höheres Level haben als die derzeitig aktive Zeitlinie. Durch die Überprüfung `!timeline.GetId().Equals(_currentTimeline.GetId())` wird ausgeschlossen, dass es sich hierbei um die aktuelle Zeitlinie handelt.

```

1 public class GraphController
2 {
3 ...
4     private void CheckForInteractions(Timeline timeline)
5     {
6         if (timeline.GetLevel() >= _currentTimeline.GetLevel())
7         {
8             if (!timeline.GetId().Equals(_currentTimeline.GetId()))
9             {
10                 timeline.ResetData();
11                 _timelinesToHandle.Add(timeline);
12             }
13         }
14
15         List<Timeline> children = timeline.GetChildren();
16         if (children.Count > 0)
17         {
18             foreach (Timeline child in children)
19             {
20                 CheckForInteractions(child);
21             }
22         }
23     }
24 ...
25 }
```

Codeausschnitt 5.4: HandleMerge Methode aus dem alten Prototyp

Trifft der Fall zu, dass `timeline.GetLevel() >= _currentTimeline.GetLevel()` dem Wert “true” entspricht, so werden diese Zeitlinien in eine Liste abgespeichert, welche die zu beachtenden Zeitlinien enthält (`_timelinesToHandle`).

5.2.1.3. Wann sind die zu beachtenden Zeitlinien aktiv?

Der *TimeController* des Prototyps übergibt an den *GameController* über die Methode *HandleGameTime()* den aktuellen Zeitpunkt des Timers. Sofern ein “*Mergvorgang*” absolviert wurde, existiert auch eine Liste an zu betrachteten Zeitlinien. Diese *_timelinesToHandle* werden nun sequenziell durchiteriert. Dabei wird zunächst geprüft, ob diese Zeitlinie eine Kopie des Chronologen besitzt, der die Aufnahme der Zeitlinie reproduziert. Falls bislang noch keine Kopie existiert, wird geprüft, ob die übergebene *_gametime* bereits größer ist als die letzte in der Zeitlinie aufgenommene Interaktion. Trifft das zu, wird die Zeitlinie aus der Liste, der *_timelineToHandle* entfernt. Trifft es nicht zu und ist der übergebene Zeitpunkt größer oder gleich des Startpunktes der Zeitlinie, so wird eine Kopie erstellt und der Zeitpunkt wird an diese Kopie übergeben.

Im letzten Schritt wird in der Methode *HandlePurge()* überprüft, welche erstellten Zeitlinien gelöscht werden können, da ihre letzte aufgenommene Interaktion im Vergleich zur *_gametime* in der Vergangenheit erfolgte und sie deshalb nicht mehr beachtet werden muss. Diese Methode wird allerdings nur dann aufgerufen, sofern der Spieler den Hauptchronologen steuert und sich auf der *_rootTimeline* befindet.

```

1 public class GraphController
2 {
3     ...
4     public void HandleGameTime(float gametime)
5     {
6         PurgeOldTimelines(gametime);
7         if (_timelinesToHandle != null && _timelinesToHandle.Count > 0)
8         {
9             foreach (Timeline timeline in _timelinesToHandle.ToArray())
10            {
11                Shadow ghost = timeline.GetGhost();
12                if (ghost != null)
13                {
14                    ghost.ReconstructRecord(gametime);
15                }
16                else
17                {
18                    if (timeline.IsTimestampStillValid(gametime))
19                    {
20                        if (timeline.GetStartTimestamp() <= gametime)
21                        {
22                            Shadow shadow = _playerController.CreateShadow(timeline.
GetPlayer());
23                            Vector3 positionVector = new Vector3(timelineGetPosition
()[0], timelineGetPosition()[1], timelineGetPosition()[2]);
24                            shadow.gameObject.transform.position = positionVector;
25                            timeline.InsertGhost(shadow);
26                            shadow.ReconstructRecord(gametime);
27                        }
28                    }
29                }
30            if (!timeline.IsTimestampStillValid(gametime))

```

```

31             {
32                 _timelinesToHandle.Remove(timeline);
33             }
34         }
35     }
36 }
37
38 private void PurgeOldTimelines(float gametime)
39 {
40     if (_currentTimeline.GetId().Equals(rootTimeline.GetId()))
41     {
42         HandlePurge(gametime, _rootTimeline);
43     }
44 }
45 private void HandlePurge(float gametime, Timeline timeline)
46 {
47     List<Timeline> children = timeline.GetChildren();
48     if (children.Count > 0)
49     {
50         foreach (Timeline child in children.ToArray())
51         {
52             HandlePurge(gametime, child);
53         }
54     }
55     else
56     {
57         if (!timeline.IsTimestampStillValid(gametime))
58         {
59             timeline.GetParent()?.GetChildren()?.Remove(timeline);
60         }
61     }
62 }
63 ...
64 }
```

Codeausschnitt 5.5: Zentrale Logik der Mechanik aus dem Gamedesign Workshop

5.2.1.4. Wie rekonstruiert die Kopie des Chronologen die aufgenommene Interaktion?

Der GraphController übergibt in der *HandleGameTime()* Methode den derzeitigen Zeitpunkt des Timers über die Methode *shadow.ReconstructRecord()* an die *Shadow* Klasse weiter. In der dieser Methode wird aus der Liste der aufgenommenen Interaktionen die Interaktion gesucht, die zu diesem übergebenen Zeitpunkt passiert ist. Die durch diesen Aufruf gefundene Interaktion: *_interactions.Find(i => (float.Equals(i.TimeStamp, timestamp)))* wird nun in im “Switch-Case” ausgewertet. Bei diesem Prototyp konnte der Spieler nur laufen und springen. Der Mechanismus der Rekonstruktion erfolgt nur in der *Shadow* Klasse.

```

1 public class Shadow : Player
2 {
3     ...
4     public void ReconstructRecord(float timestamp)
```

```

5     {
6         if(_interactions.Count > 0)
7         {
8             _lastInteraction = _interactions.OrderByDescending(item => item.
9                 TimeStamp).First();
10            }
11            InteractionSaveData interaction = _interactions.Find(i => (float.Equals(i.
12                 TimeStamp, timestamp)));
13            SetLastTimestamp(timestamp);
14            if(interaction != null)
15            {
16                InteractionType type = (InteractionType)Enum.Parse(typeof(InteractionType),
17                interaction.Type);
18                switch(type)
19                {
20                    case InteractionType.WALK:
21                        MoveShadow(interaction.Target);
22                        break;
23
24                    case InteractionType.JUMPSTART:
25                        MoveShadow(interaction.Source);
26                        gameObject.GetComponent<AnimationHandler>().jump();
27                        break;
28
29                    case InteractionType.JUMPSTOP:
30                        MoveShadow(interaction.Source);
31                        gameObject.GetComponent<AnimationHandler>().stopJump();
32                        break;
33
34                    default:
35                        break;
36                }
37            if(_lastInteraction != null && _lastInteraction.TimeStamp < timestamp)
38            {
39                Destroy(this.gameObject);
40                DestroyShadow?.Invoke();
41            }
42        }
43        private void MoveShadow(GameObjectSaveData data)
44        {
45            Vector3 positionVector = new Vector3(data.Position[0], data.Position[1], data.
46            Position[2]);
47            Vector3 moveVector = new Vector3(positionVector.x - transform.position.x, 0,
48            positionVector.z - transform.position.z);
49            transform.forward = moveVector;
50            transform.position = positionVector;
51            transform.localScale = new Vector3(data.Scale[0], data.Scale[1], data.Scale
52            [2]);
53            gameObject.GetComponent<AnimationHandler>().SetGhostPosition(positionVector);
54        }
55        ...
56    }

```

Codeausschnitt 5.6: Rekonstruktionsmethode aus Shadow.cs

5.2.1.5. Wie wurden die Interaktionen aufgenommen?

Wie bereits in den Grundlagen erwähnt, muss das Aufnehmen und Abspielen der Interaktionen in einem deterministischen System stattfinden. Unity ist grundlegend nicht deterministisch. Deswegen werden beim Abspeichern der Bewegungen des Spielercharakters alle Zustände in den gerenderten Frames abgespeichert.

```
1 public class MovementAbility : Ability
2 {
3     ...
4     protected override void HandleInput()
5     {
6         float horizontal = Input.GetAxisRaw("Horizontal");
7         float vertical = Input.GetAxisRaw("Vertical");
8         Vector3 direction = new Vector3(horizontal, 0f, vertical).normalized;
9         if (direction.magnitude >= 0.1)
10        {
11            //code was taken from the following video: https://www.youtube.com/watch?v=4HpC--2iowE line 12-14, 17-18
12            float targetAngle = Mathf.Atan2(direction.x, direction.z) * Mathf.Rad2Deg
13            + _cam.eulerAngles.y;
14            float angle = Mathf.SmoothDampAngle(_playerTransform.eulerAngles.y,
15            targetAngle, ref _turnSmoothVelocity, _turnSmoothTime);
16            Vector3 moveDir = Quaternion.Euler(0f, targetAngle, 0f) * Vector3.forward
17            ;
18            if (!GetComponent<Animator>().GetBool("isJump"))
19            {
20                _playerTransform.rotation = Quaternion.Euler(0f, angle, 0f);
21                _controller.Move(moveDir.normalized * _speed * Time.deltaTime);
22                if (!_cam.parent.GetChild(2).GetChild(3).GetComponent< AudioSource >().
23                    enabled)
24                {
25                    AudioPlayerScript.instance.playSpecificAudio("walking");
26                }
27                if (GetComponent< Animator >().GetBool("isJump"))
28                {
29                    _controller.Move(GetComponent< JumpAbility >().GetJumpdirection() *
30                    _speed / 3 * Time.deltaTime);
31                }
32                SubmitAction(InteractionType.WALK, this.gameObject.GetComponent< Player >()
33                , this.gameObject, this.gameObject.transform, TimeController.Instance.GetGameTime
34                ());
35            }
36            else
37            {
38                if (_cam.parent.GetChild(2).GetChild(3).GetComponent< AudioSource >().
39                    enabled)
39                {
40                    AudioPlayerScript.instance.stopSpecificAudio("walking");
41                }
42            }
43            if (GetComponent< Animator >().GetBool("isJump"))
44            {
45                _controller.Move(GetComponent< JumpAbility >().GetJumpdirection() * _speed
46                / 3 * Time.deltaTime);
47            }
48        }
49    }
```

```
42 ...
43 }
```

Codeauschnitt 5.7: Aufnahme von Bewegungen, Ausschnitt aus MovementAbility.cs

Dieses Vorgehen hat Jonathan Blow beim Rewind-System in “the Braid” verwendet (vgl. GDC (2016)). Sein Vorgehen besagt, dass Positionen oder in diesem Fall Interaktionen nicht interpoliert werden, sondern in einem 1:1 Abbild abgespeichert werden sollen. Dadurch wird ein fehlerhaftes Reproduzieren vermieden. Daher war die Überlegung, die Bewegungs-Interaktion des Spielercharakters in jedem seiner bewegten Frames in Form einer Interaktionsklasse abzuspeichern. Die Methode *SubmitAction()* wird daher immer ausgeführt, sobald sich der Spielercharakter durch die Eingabe des Spielers bewegt. Die dabei erzeugten Interaktionen (vgl. “Codeausschnitt 5.8”) werden in der *Character* Klasse, genauer gesagt der entsprechenden Zeitlinie abgespeichert. Diese vollständig abgespeicherten Interaktionen können nun auf dem oben beschriebenen Weg reproduziert werden.

```
1 public class Interaction
2 {
3     public InteractionType type;
4     public Player actor;
5     public GameObject interactedObject;
6     public Transform interactionPosition;
7     public float timestamp;
8     public Interaction(InteractionType i, Player p, GameObject g, Transform t, float ti)
9     {
10         type = i;
11         actor = p;
12         interactedObject = g;
13         interactionPosition = t;
14         timestamp = ti;
15     }
16 }
```

Codeauschnitt 5.8: Interaktionsklasse

5.2.2. Aspekte zum Überarbeiten

Die in den Kapiteln “5.2.1.1 bis 5.2.1.5” beschriebenen Aspekte sind in der Theorie richtig umgesetzt, allerdings besteht auch dabei noch Verbesserungspotenzial. Diese Aspekte werden im kommenden “Kapitel 5.3: Aufbau des Prototyps” aufgegriffen und verbessert.

5.2.2.1. Falscher Bezug bei der Spielmechanik

```
1 public class PlayerController : Controller
2 {
3     ...
4     public void HandleCharacterSelection(String name)
5     {
6         TimeController.Instance.SetActive(true);
7         _currentSelection = name;
8
9         if(IsMerge())
10        {
11            HandleMerge();
12        }
13        else
14        {
15            HandleSplit();
16        }
17    }
18    ...
19    private bool IsMerge()
20    {
21        GraphController graphController = this.gameObject.GetComponent<
22        GraphController>();
23        bool isM = graphController.IsSelectionInTimelineYet(_currentSelection);
24        if(isM)
25        {
26            Debug.Log("habe ich grad gemerget: " + isM);
27        }
28        else
29        {
30            Debug.Log("habe ich grad gesplittet: " + !isM);
31        }
32        return isM;
33    }
}
```

Codeausschnitt 5.9: HandleCharacterSelection in der PlayerController.cs

Der grundlegende Fokus der Hauptmechanik ist falsch ausgelegt. Im Gamedesign Workshop wurde damals das Überwachen und Kontrollieren des Spielercharakters als Hauptpunkt der Mechanik ausgemacht, das hatte aber zur Folge, dass das Erstellen des Graphen daran angegliedert werden musste. Das macht in dem Prinzip der Mechanik keinen Sinn, da der Spieler und dessen Avatar nicht der Kernpunkt ist, sondern lediglich ein Teil der Mechanik. In der Klasse *PlayerController* wurde durch die Mithilfe der Klasse *GraphController* entschieden, wann eine Auswahl ein “*Split*”- und wann ein “*Mergeprozess*” stattzufinden hat. Deswegen ist es sinnvoller, diesen Prozess über die Klasse *GraphController* zu bestimmen.

5.2.2.2. Undurchdachte Split und Merge Methoden

Eine weitere Ursache aus dem falschen Fokus der Hauptmechanik wurden die *HandleSplit* und *HandleMerge* mit nebensächlichen Aktionen überladen und nur auf den Spielercharakter bezogen.

```

1 public class PlayerController : Controller
2 {
3     ...
4     private void HandleSplit()
5     {
6         BeforeCharacterCreated?.Invoke(_currentCharacter);
7         ChangeSpawnPoint(_currentCharacter);
8         Destroy(_currentCharacter.gameObject);
9         CharacterData newCharacter = GetCharacterData(_currentSelection);
10        if(newCharacter)
11        {
12            RemoveShadow(newCharacter);
13            GameObject playerObject = InstantiateCharacter(newCharacter.PREFAB);
14            Player player = playerObject.AddComponent<Player>();
15            player.Init(newCharacter);
16            AfterCharacterCreated?.Invoke(player);
17            Split?.Invoke(player.GetCharacterData());
18        }
19        GameObject shadow = InstantiateCharacter(_temporalOldPlayer.GetCharacterData
() .PREFAB_GHOST);
20        shadow.GetComponent<AnimationHandler>().SetGhostMode(true);
21        _temporalOldPlayer = null;
22    }
23    ...
24    private void HandleMerge()
25    {
26        BeforeCharacterCreated?.Invoke(_currentCharacter);
27        RemoveCharacter(_currentCharacter.GetCharacterData());
28        SavePlayerData player = StateManager.LoadPlayer(_currentSelection);
29        CharacterData playerData = GetCharacterData(_currentSelection);
30        RemoveShadow(playerData);
31        GameObject newSpawn = new GameObject();
32        newSpawn.hideFlags = HideFlags.HideInHierarchy;
33        float[] position = player.Position.Position;
34        float[] rotation = player.Position.Rotation;
35        newSpawn.transform.position = new Vector3(position[0],position[1],position
[2]);
36        newSpawn.transform.rotation = Quaternion.Euler(new Vector3(rotation[0],
rotation[1],rotation[2]));
37        newSpawn.transform.localScale = new Vector3(1,1,1);
38        _spawnPoint = newSpawn.transform;
39        GameObject newPlayer = InstantiateCharacter(playerData.PREFAB);
40        Player newPlayerScript = newPlayer.AddComponent<Player>();
41        newPlayerScript.Init(playerData);
42        Merge?.Invoke(newPlayerScript);
43        AfterCharacterCreated?.Invoke(newPlayerScript);
44        _temporalOldPlayer = null;
45    }
46 }
```

Codeausschnitt 5.10: Alte HandleSplit und HandleMerge Methode aus PlayerController.cs

Sowohl die *HandleSplit* als auch die *HandleMerge* löschen den alten Platzhalter des ausgewählten Spielercharakters, ohne dabei zu wissen, ob es noch mehr dieser Platzhalter geben kann. Sobald der Programmablauf eine der beiden Methoden aufruft, weiß die *PlayerController* Klasse nicht, welche anderen Platzhalter oder Kopien des Chronologen in der Welt aktiv sind. Dieser Zustand birgt Gefahren durch Fehler mit sich und muss durch ein zentraleres System ausgetauscht werden. Außerdem kann ein richtiges Positionieren der neuen Charaktere nach einem "Merge" nicht gewährleistet werden, da ein verschachteltes "Mergen" und "Splitten" zu fehlerhaften Referenzen in der Startposition führen kann. Würde der Spieler nach einem "Split" "zurückmergen" und nach einer Zeit wieder "Splitten" und erneut, "zurückmergen" so würde der im ersten "Split" erstellte Charakter verschwinden oder erneut instanziert werden, da das System kein Wissen über alle Zeitlinien besitzt. Es ist aber auch möglich, dass das Spielobjekt an den Beginn der Zeitlinie instanziert wird, da es nur die diese Information hat.

5.2.2.3. Das Erstellen des Graphen ist ein Nebenprodukt

Durch die Thematik aus dem vorherigen Kapitel wird erkennbar, dass das Bauen des Graphen schwierig und unstrukturiert ist.

```
1 public class GraphController : Controller
2 {
3     ...
4     void Start()
5     {
6         playerController = this.gameObject.GetComponent<PlayerController>();
7         playerController.InitTimeline += HandleInitTimeline;
8         playerController.Split += HandleAddChild;
9         playerController.Merge += HandleMerge;
10        timelinesToHandle = new List<Timeline>();
11        CheckForInteractions(rootTimeline);
12    }
13    ...
14    private void HandleMerge(Player player)
15    {
16        float startPoint = -1;
17        Timeline timeline = GetTimelineOfNewPlayer(player.GetCharacterData());
18        if (timeline != null)
19        {
20            List<Timeline> childrenOfNewTimeline = timeline.GetChildren();
21            if (childrenOfNewTimeline.Count > 1)
22            {
23                foreach (Timeline child in childrenOfNewTimeline)
24                {
25                    if (child.GetId().Equals(currentTimeline.GetId()))
26                    {
27                        startPoint = child.GetStartTimestamp();
28                    }
29                }
30            }
31        }
32    }
33 }
```

```

31         if (childrenOfNewTimeline.Count == 1)
32         {
33             foreach (Timeline child in childrenOfNewTimeline)
34             {
35                 startPoint = child.GetStartTimestamp();
36             }
37         }
38         if (startPoint != -1)
39         {
40             TimeController.Instance.SetGameTime(startPoint);
41             currentTimeline = timeline;
42         }
43         HandleRemoveOldCharacter(rootTimeline);
44         timelinesToHandle = new List<Timeline>();
45         CheckForInteractions(rootTimeline);
46     }
47 }
48 ...
49 private void HandleAddChild(CharacterData playerData)
50 {
51     Timeline timeline = new Timeline(currentTimeline.GetLevel() + 1,
52     TimeController.Instance.GetGameTime(), playerData, currentTimeline,
53     playerController.GetSpawn());
54     currentTimeline.InsertChild(timeline);
55     currentTimeline = timeline;
56 }
```

Codeausschnitt 5.11: HandleSplit und HandleMerge Methode aus GraphController.cs

In “Abbildung 5.10” in der *HandleSplit()* Methode wird die Action *Split* ausgeführt, durch welche in der *GraphController* Klasse eine angehängte Methode aufgerufen wird. Dieses Aufrufen ist in diesem Fall das Erzeugen einer neuen Zeitlinie und das Angliedern dieser Zeitlinie an die derzeitige. Außerdem wird diese neu erzeugte Zeitlinie die neue *_currentTimeline*. In der *HandleMerge()* Methode in “Abbildung 5.10” wird die *Merge* Action getriggert, welche ebenfalls im *GraphController* eine Methode aufruft, die *HandleMerge()*. In dieser Methode wird auf eine umständliche Art ermittelt, welche bereits existierende Zeitlinie nun die neue *_currentTimeline* wird und zu welchem Zeitpunkt diese Zeitlinie das letzte Mal aktiv war.

5.2.2.4. Eine Zeitlinie besitzt keinen offiziellen Endzeitpunkt

Die Zeitlinien im alten Prototyp aus dem Gamedesign Workshop besitzen keine Referenz auf einen Endzeitpunkt. Dieser Endzeitpunkt wurde entweder über die letzte Interaktion der Interaktionsliste gelöst, oder dem Startzeitpunkt der Zeitlinie von der auf die neue bereits erstellte Zeitlinie “*gemergt*” wurde. Beide Fälle sind fehleranfällig. Zum einen kann es sein, dass die letzte Interaktion der Liste nicht an dem Zeitpunkt stattfand, an dem entweder ein “*Split*” oder ein “*Merge*” ausgeführt wurde, da der

Spieler sich bis dahin nicht bewegt hat. Bewegungen werden als Interaktionen erfasst. Bewegt sich der Spieler nicht, so findet auch keine Interaktion statt. Zum anderen ist der fehlende Endzeitpunkt der Zeitlinie bei einem “Merge” hinderlich, da das System den richtigen, neuen Zeitpunkt nicht bestimmten kann. “Mergt” der Spieler auf eine Zeitlinie zurück, so wurde der Startzeitpunkt der Zeitlinie, von der aus “gemergt” wurde, als neuer Zeitpunkt angesetzt. Dabei kann es sein, dass der Timer nicht auf den richtigen Zeitpunkt gesetzt wird, da sich der Startpunkt um Millisekunden unterscheiden könnte. Ein weiteres Problem stellen parallele Zeitlinien dar, die auf derselben Ebene angegliedert, aber zu einem unterschiedlichen Zeitpunkt erstellt wurden. Dabei muss zusätzlich überprüft werden, welcher Index diese angegliederte Zeitlinie aus der Liste der Children hat, um so den neuen Zeitpunkt des Timers zu setzen.

5.2.2.5. Grundlegende Struktur der Fähigkeiten des Spielers

Wie im Codeausschnitt 5.6 zu sehen, wird das Reproduzieren der Interaktionen in einer Methode in der *Shadow* Klasse vollzogen. Das birgt ein Hindernis mit sich, da nicht immer überprüft werden kann, ob ein Objekt, mit dem interagiert werden soll, an der richtigen Stelle steht. Dafür benötigt die Rekonstruktionsmethode das Wissen über mögliche entstandene Kollisionen, wie es beim Aufnehmen der Interaktionen der Fall war. Die Klassen, die nun für das Reproduzieren einer Fähigkeit des Spielers genutzt werden sollen, sollten im best-möglichen Fall ein Abbild der Fähigkeitsklasse sein. Jede Fähigkeitsklasse sollte eine Rekonstruktionssklasse haben, welche zur Laufzeit Änderungen und Kollisionen durch das Spielobjekt aufzeichnet, um dadurch die aufgenommenen Interaktionen fehlerfrei rekonstruieren zu können.

5.3. Aufbau des Prototyps

In diesem Kapitel wird der Aufbau des Prototyps vorgestellt.

5.3.1. Vorstellung der Spiellogik

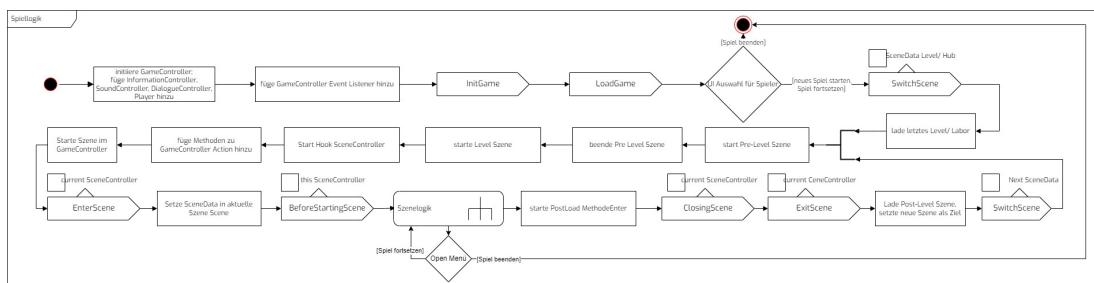


Abbildung 88.: Spiellogik des Prototyps, (Quelle: eigene Darstellung)

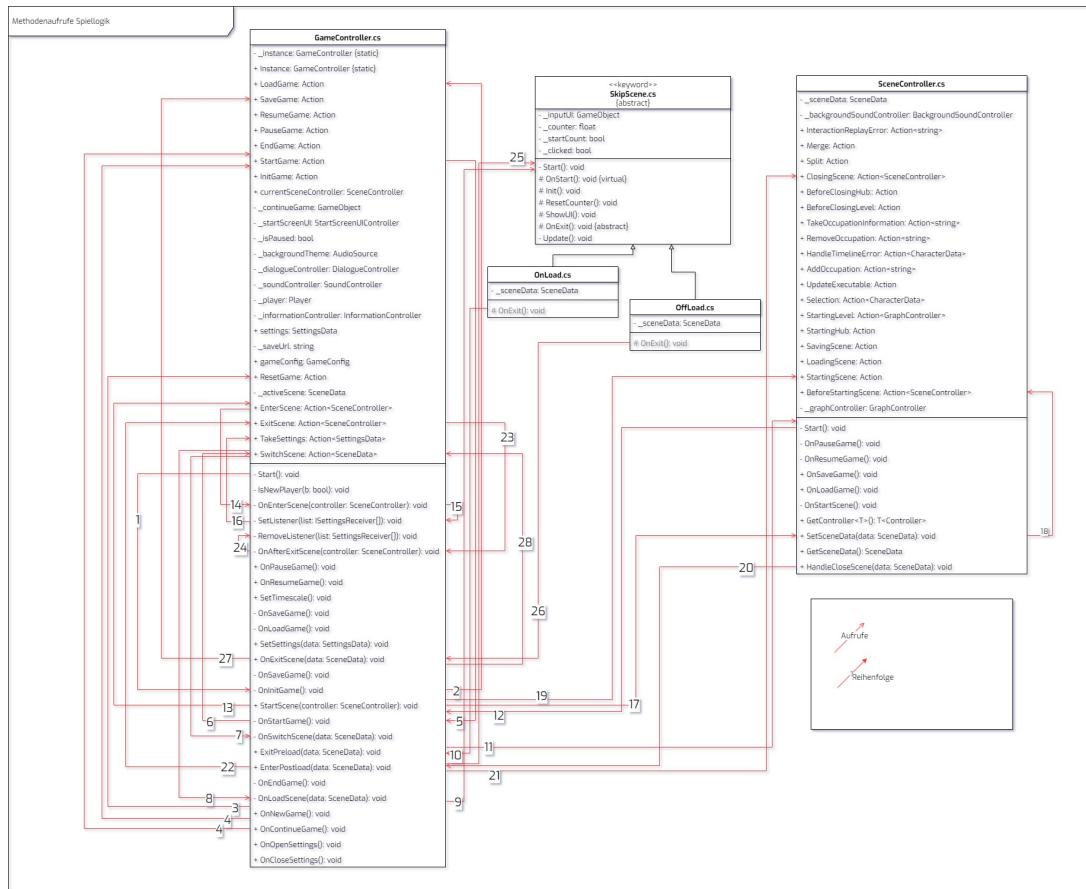


Abbildung 89.: Spiellogik im Klassendiagramm eingezeichnet, (Quelle: eigene Darstellung), (In groß im Anhang A 1.3.2)

Bemerkung vornweg: Die in den Klammern referenzierten Markierungen beziehen sich auf die Markierungen aus „Abbildung 89“. Die zusammenfassende Schilderung der Ereignisse bezieht sich auf „Abbildung 88“.

Der Prototyp startet in der *OnLoad* Szene, in welcher zunächst der *GameController* und weitere Singletons wie der *InformationController*, *SoundController*, *DialogueController* und der *Player* instanziert und initialisiert werden. Die *Player* Klasse bildet dabei den Spieler ab und enthält den erreichten Fortschritt des Spiels (vgl. Markierung 1).

Sobald die Singletons instanziert wurden, wird der Prototyp initialisiert, das bedeutet, die Konfigurationsdatei des Spiels wird geladen. In ihr befinden sich wie in Codeausschnitt 5.12 zu sehen die einzelnen verfügbaren Szenen sowie die Laborszene, die Tutorial Szene, die verfügbaren Ichs des Chronologen, sowie die UIData des Dialogs und alle Informationen, die diese Version des Spiels enthalten soll.

```

1 public class GameConfig : ScriptableObject
2 {
3     public List<SceneData> levelScenes;
4     public SceneData startScene;
5     public SceneData hub;
```

```

6     public SceneData tutorial;
7     public List<CharacterData> playableCharacters;
8     public bool isDev = true;
9     public UIData dialogueUiData;
10    public List<TutorialData> tutorialUiData;
11 }

```

Codeauschnitt 5.12: GameConfig.cs des Prototyps

Darauffolgend werden alle Daten, die der Spieler im Spiel gesammelt hat, geladen. Das ist sein Spielstand. Dieser zeigt an, welche Level er bereits erfolgreich beenden konnte, seine verfügbaren Ichs des Chronologen, seine Spieleinstellungen und seine gesammelten Informationen (vgl. Markierung 2).

Über das Startmenü, das nun erschienen ist, kann der Spieler das Spiel starten oder an seinem Spielfortschritt forsetzen. Würde der Spieler ein neues Spiel beginnen, so wird die Action *ResetGame* ausgelöst, wodurch seine Spielstände resettet werden und er wieder im Tutorial beginnen würde (vgl. Markierungen 3 und 4).

Nachdem der Spieler das Spiel fortgesetzt oder neu gestartet hat, wird die aktuelle *SceneData* auf die *SceneData* des Labors oder des nächsten Levels gesetzt (vgl. Markierungen 6 und 7). Das System lädt nun die Beitreitsszene der neuen Szene. Diese wird *On[Name des Levels]* (Eintrittsszene) genannt und beinhaltet in einer finalen Version eine Zwischensequenz. Im derzeitigen Prototyp ist es ein Screenshot der kommenden Spielwelt und ein einführender Text. Nachdem der Inhalt der Szene betrachtet wurde, wechselt das System nach einer Eingabe des Spielers die Szene auf die Szene der aktuellen *SceneData* (vgl. Markierungen 8, 9 und 10). Sobald die Szene geladen hat, wird die *Start()* Methode des *SceneControllers* aufgerufen, der die lokalen Methoden an die *Actions* des *GameControllers* anhängt. Dadurch erhält die Szene ein Wissen darüber, welcher Zustand im Spiel nun herrschen wird. Der *SceneController* übergibt eine Instanz von sich als Parameter an den *GameController*. Dadurch existiert ein globaler Kenntnisstand auf die aktuelle Szene. Über die *EnterScene* Action werden die Einstellungen des Spielers an alle in der Szene relevanten Empfänger übermittelt. Der *GameController* übergibt an den *SceneController* die *SceneData* dieser Szene, in der die Einstellungen zu den jeweiligen Szenen definiert werden können.

```

1 public class SceneData : ScriptableObject
2 {
3     public string sceneName;
4     public string visibleUI;
5     public SceneType sceneType;
6     public Reward reward;
7     public SceneData destination;
8 }

```

Codeauschnitt 5.13: SceneData.ts aus Prototyp

Über die *SceneData* wird das sichtbare UI zum Start der Szene definiert. Jede Szene erfährt durch ihre *SceneData* zu welcher Szene anschließend gewechselt wird. Außerdem ist in ihr der Gewinn des Spielers definiert und um welche Szene es sich handelt. Es kann entweder das Labor sein, ein neues Level oder das erreichbare Startmenü (vgl. Codeausschnitt 5.13).

Nachdem die *SceneData* im *SceneController* gespeichert wurde, startet der *SceneController* die *BeforeStartingScene* Action, durch welche alle *Controller* Klassen der Szene initialisieren werden können. Abschließend startet der *GameController* durch das Ausführen der *StartingScene* Action die Szene (vgl. Markierungen 11 bis 19).

Wie die Levelszenen aufgebaut sind, wird in "Kapitel 5.3.3: Aufbau der Szenelogik" vorgestellt.

Nach Beendigung des Levels ruft der *SceneController* über die *HandleCloseScene()* Methode die

EnterPostLoad() Methode des *GameControllers* auf. Die *EnterPostLoad()* Methode führt in der noch aktiven Szene die Action *ClosingScene* des *SceneControllers* auf. Sie gibt der Szenelogik den Hinweis darauf, dass die Spiellogik die Szenen verlassen wird. Nachdem über die Action *ExitScene* Methodenverknüpfungen für die Einstellungen entfernt wurden, wird die *Off[Name des Levels]* Szene (Austrittsszene) des Levels geladen. In der Austrittsszene wird in einem fertigen Spiel eine Zwischensequenz zu sehen sein, wie der Chronologe das Level verlässt. In diesem Prototyp wird ein Screenshot des absolvierten Levels angezeigt, auf welchem eine Zusammenfassung des Chronologen geschrieben steht. Nachdem die Austrittsszene fertig ist, wechselt das System zur neuen *SceneData* die in der vorherigen *SceneData* der abgeschlossenen Szene als *destination* hinterlegt war. Der Kreislauf beginnt über die Action *SwitchScene* erneut. Der Spieler sieht nun die Eintrittsszene des neuen Levels oder des Labors (vgl. Markierungen 20 bis 28).

5.3.2. Vorstellung der Spielszenen

Grundlegend werden einzelne Szenen im *Scenes* Ordner in den Assets angelegt.

Um eine Übersichtlichkeit gewährleisten zu können, werden verschiedene Ordner angelegt, die nach dem Namen der darin enthaltenen Szene benannt werden. Auf der obersten Hierarchieebene des *Scenes* Ordner befindet sich lediglich die *OnLoad* Szene (Initialisierungsszene), die für das Initialisieren des Spiels zuständig ist. Sie ist der Anfangspunkt des Spiels. Neben der Initialisierungsszene wird sich auch das Hauptmenü darin befinden, zu welchem der Spieler zur Laufzeit der Levels wieder zurückkehren

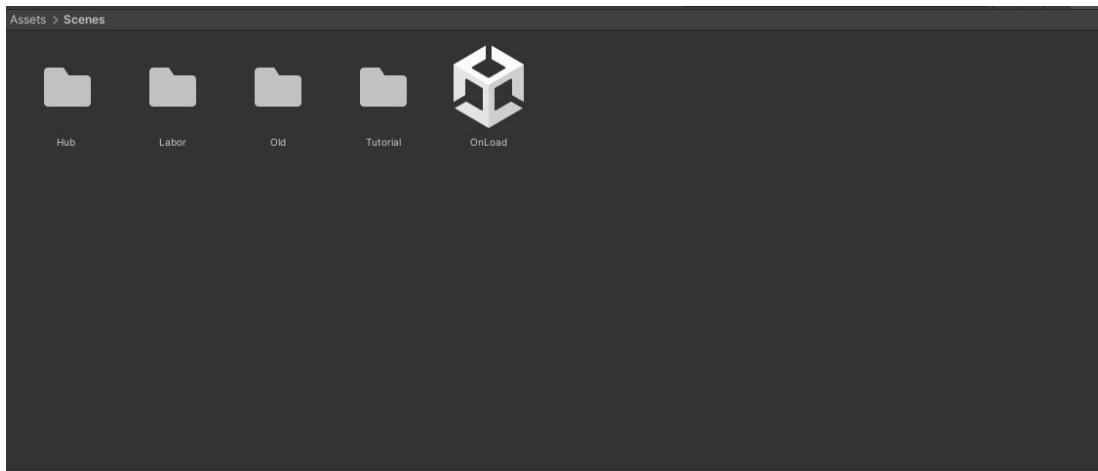


Abbildung 90.: Szenen Ordner in Unity, (Quelle: eigene Darstellung)

kann.



Abbildung 91.: Szenen Ordner im Detail, (Quelle: eigene Darstellung)

In den jeweiligen Szenen Ordnern befinden sich zu jeder Hauptszene eine Eintrittsszene (*On-Load*) und Austrittsszene (*Of-Load*), die im Suffix den Namen der eigentlichen Szene führen. In ihnen wird der Spieler, wie bereits im "Kapitel 5.3.1: Vorstellung der Spiellogik" erläutert, das Intro und Outro zu jedem Level oder dem Labor sehen.

Der Aufbau einer einzelnen Spielszene ist in "Abbildung 92" dargestellt und wir im Folgenden beschrieben.

Als erster Kernknoten der Szenen wird der *SceneController* definiert, der als leeres Spielobjekt mit dem Namen "—SceneController—" erstellt wird. Unter ihm sind die wichtigen Elemente jeder Szene mit derselben Berechnungsart benannt und erstellt worden. Auf dem Spielobjekt "—Camera—" befindet sich der *CameraController*. Das Spielobjekt "—Light—" ist die Heimat des *LightControllers*. Das Spielobjekt "Ground" ist der Hierarchieknotenpunkt für das Level Prefab. Auf diesem befindet

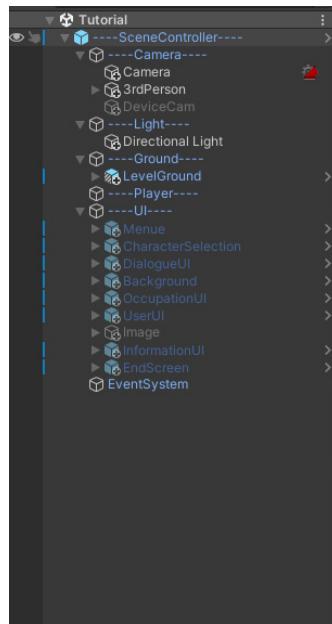


Abbildung 92.: Szenehierarchie, (Quelle: eigene Darstellung)

sich ebenfalls der *GroundController*. Das Spielobjekt “—Player—” wird von den Klassen des *CharacterControllers* und des *InputControllers* verwaltet und ist als Hierarchie für die Ichs des Chronologen wichtig. Das UI des Spiels wird unter dem Spielobjekt “UI” angegliedert und wird vom *UIController* verwaltet.

Die Eintritts- und Austrittsszenen sind im Vergleich zu Levelszenen nicht hierarchisch aufgebaut.

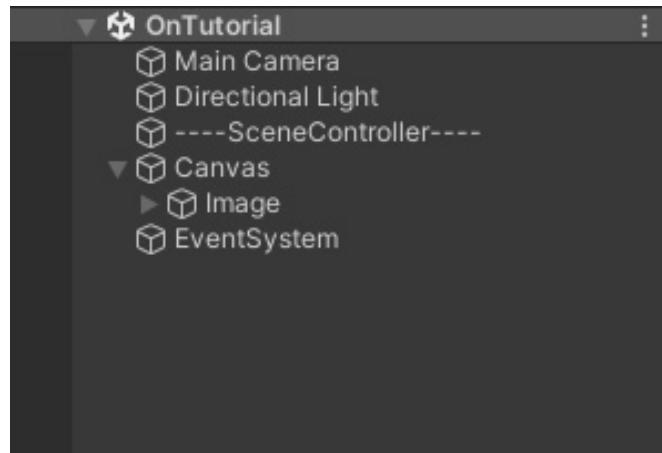


Abbildung 93.: Eintritts- und Austritts-Szenehierarchie, (Quelle: eigene Darstellung)

Sie besitzen als wichtige Spielobjekte ebenfalls ein “—SceneController—” Spielobjekt, allerdings beinhaltet er je nach Typ der Szene entweder eine *OnLoad* oder *OffLoad* Klasse. Diese Klasse ist zum einen für die Darstellung des Inhaltes wichtig und zum anderen für das Weiterspringen zur nächsten Szene, nachdem der Spieler durch seine Tasteneingabe das Signal dafür gegeben hat.

5.3.3. Aufbau der Szenelogik

Nachdem der Aufbau der Szenen grundlegend erklärt wurde, wird jetzt der Ablauf der Szene anhand eines Aktivitäts- und Klassendiagramms erklärt.

Anmerkung zur Abbildung: In “Abbildung 94” handelt es sich um ein nicht vollständiges Klassendiagramm (vollständig im Anhang dargestellt).

Es gilt hierbei wieder dieselbe Bemerkung wie in “Kapitel 5.3: Aufbau des Prototyps”. Es wird das Diagramm aus “Abbildung 95” beschrieben. Die Kennzeichnungen in den Klammern beziehen sich auf die Methodenaufrufe in “Abbildung 94”.

Die Szenelogik beginnt mit der Methode *OnStartScene()* die beim Aufruf der Action *StartScene* ausgeführt wird. Die Action wurde vor dem Start der Szenelogik vom *GameController* aufgerufen. Zu Beginn der *OnStartScene()* Methode wird die Action *LoadingScene* aufgerufen, durch welche die Methoden, die durch diese Action ausgeführt werden, ihre wichtigen Informationen zum Beispiel aus der *Player* Klasse laden können (vgl. Markierungen 1 und 2). Da beispielhaft in der *SceneData* dieser Szene, der *SceneType* auf *LEVEL* steht, wird die Logik eines Spiellevels und nicht des Labors ausgeführt. Eine Logik des Labors existiert in dieser Arbeit des Prototyps

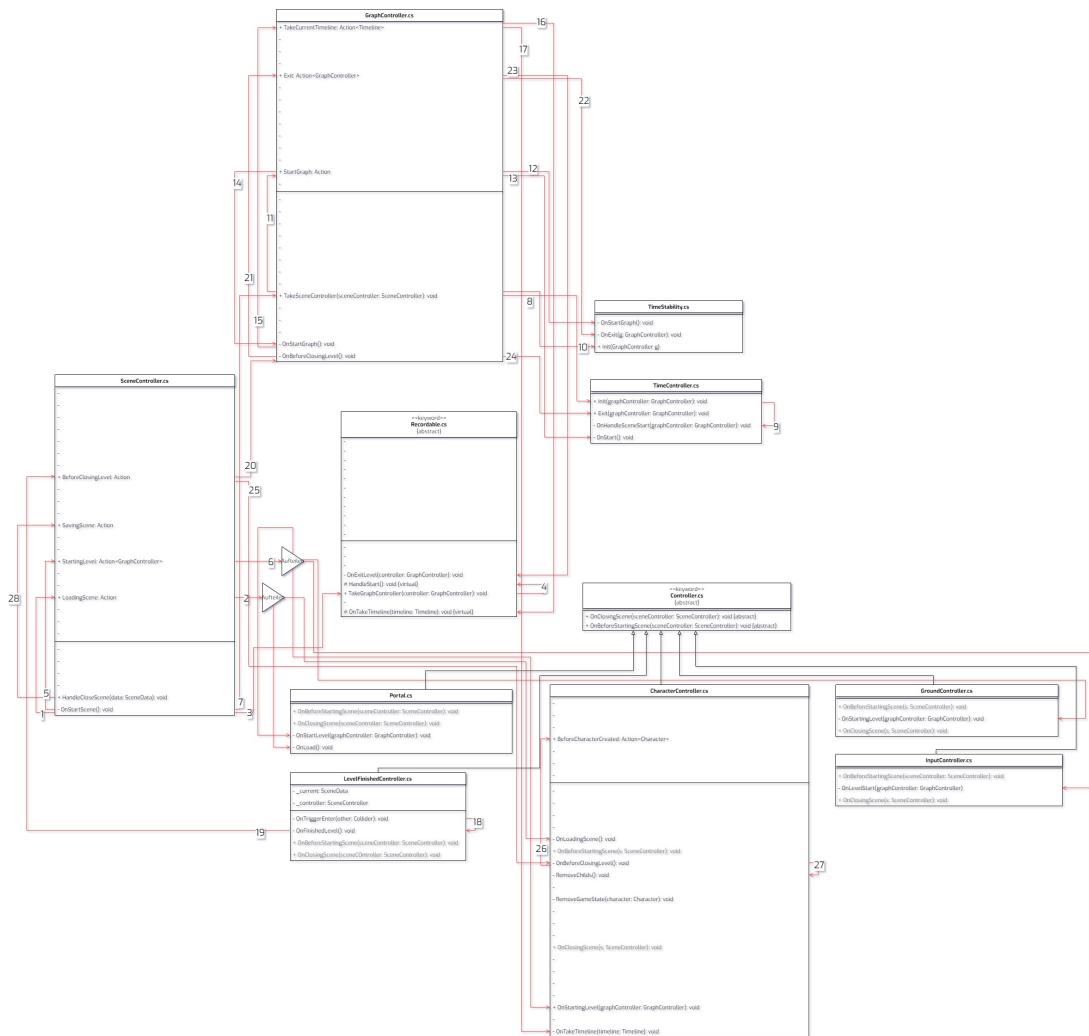


Abbildung 94.: Szenelogik ohne Kernmechanik, (Quelle: eigene Darstellung), (Vollständig im Anhang A 1.3.2)

nicht, da der Fokus auf die Gestaltung und Entwicklung der Hauptspielmechanik lag. Aufgrund der in "Abbildung 92" gezeigten und in "Kapitel 5.3.2: Vorstellung der Spielszenen" beschriebenen Gegebenheit ist es nun möglich über die unityinterne Methode `GetComponentsInChildren < T > ()` alle *Recordable* Objekte aus den Hierarchieunterknoten zu referenzieren und den am *SceneController* angehängten *GraphController* an die *Recordable* Klassen zu übergeben. Anschließend wird das Level über die Action *StartingLevel* aus dem *SceneController* gestartet. Dabei erhalten alle *Controller* Klassen eine Referenz auf den *GraphController* und können ihre Methoden an die Actions des *GraphControllers* hängen. Im Anschluss wird der *GraphController* gestartet, über welchen nun die Szenelogik gesteuert wird (vgl. Markierungen 3 bis 7).

Die Elemente *Recordable*, *GraphController* und *Controller* werden in den folgenden Kapiteln 5.3.4 und 5.3.5 näher erklärt.

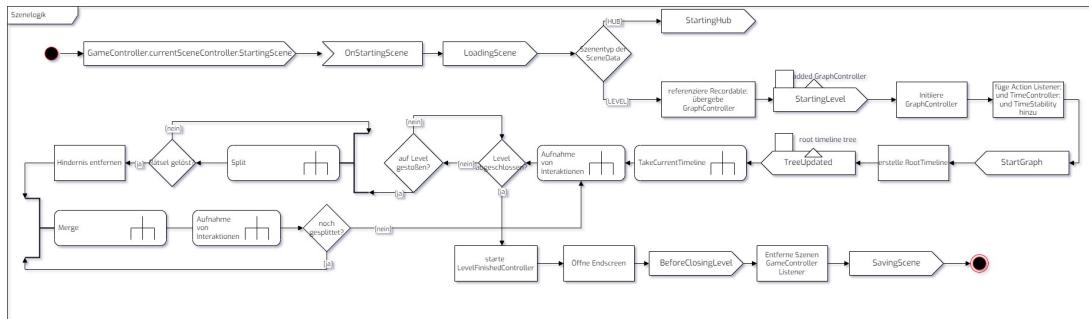


Abbildung 95.: Szenelogik des Prototyps, (Quelle: eigene Darstellung), (In groß im Anhang A 1.3.2)

Der *GraphController* initialisiert zuerst seine eigenen Actions mit seinen Methoden, ehe er den *TimeController* und die *TimeStability* initialisiert.

Die Funktion des *TimeControllers* und die *TimeStability* werden ebenfalls in "Kapitel 5.3.4: Vorstellung der einzelnen Komponenten" erklärt.

Anschließend führt der *GraphController* seine Action *StartGraph* aus, welche das Startsignal für das Level ist. Es wird nun die *_rootTimeline* erzeugt. Durch das Ausführen der Action *TakeCurrentTimeline*, welche ebenfalls als Element in den späteren Kapiteln 5.3.4 bis 5.3.8 vorgestellt wird, wird der Chronologe in die Spielwelt gesetzt und als Referenz in der *_rootTimeline* abgelegt (vgl. Markierungen 8 bis 17).

Die eingezeichnete Schleife, die ab dem Aktivitätsknoten "Aufnahme von Interaktionen" beginnt, wird in den Kapiteln 5.3.4, 5.3.5, 5.3.8, 5.3.6 und 5.3.10 näher erklärt.

Sobald der Spieler im Ziel des Levels angekommen ist und mit dem *LevelFinishedController* Spielobjekt kollidiert, wird zum einen das Ziel-UI des Levels angezeigt und zum anderen wird das Level auf die Beendigung durch die Action *BeforeClosingLevel* vorbereitet (vgl. Markierungen 18 und 19). Mit der Action wurden Methoden verknüpft, die sowohl im *GraphController*, *TimeController* und *CharacterController* ihre Komponenten der Spielmechanik und der Szenelogik beenden. Dazu gehören unter anderem das Stoppen des Timers, das Entkoppeln aller *Recordable* Objekte und das Entfernen aller Chronologen und deren Kopien aus der Spieldatenbank (vgl. Markierungen 20 bis 27). Sobald der Spieler im Beendigungs-UI den Button zum Weiterspielen (dieser befindet sich in der Klasse *LevelEndScreen*, welche nicht auf dem Klassendiagramm abgebildet wird) und damit dem Beenden des Levels geklickt hat, wird die *HandleCloseScene* Methode des *SceneControllers* aufgerufen, welcher alle Szenemethoden von den Spiel-Actions entfernt und das Level über die Action *SavingScene* speichern lässt (vgl. Markierung 28).

5.3.4. Vorstellung der einzelnen Komponenten

5.3.4.1. Zeitlinie (Timeline)

Wie bereits der alte Prototyp aus dem Gamedesign Workshop wird auch dieser auf einem System von ineinander verschachtelten Zeitlinien aufgebaut. Jeder Zeitlinie enthält dabei jeweils nur eine *CharacterData* die für das Ich des Chronologen steht.

Die *CharacterData* stellt auf Datenebene ein Abbild des Chronologen dar. Sie enthält den Namen des Chronologen, die jeweiligen Prefabs, die das Spiel instanziert und den Farbwert, der für die Unterscheidung in der visuellen Gestaltung wichtig ist. Sie wird in "Codeausschnitt 5.14" dargestellt. Außerdem wird über sie bestimmt, ob die Interaktionen des Spiels aufgezeichnet werden müssen.

```

1 public enum CharacterType
2 {
3     ORIGIN, SUBVERSION, NULL
4 }
5 public class CharacterData : ScriptableObject
6 {
7     public string Name;
8     public string ID;
9     public GameObject Ghost;
10    public GameObject Prefab;
11    public GameObject Copy;
12    public bool ToRecord;
13    public CharacterType CharacterType;
14    public List<AbilityData> Abilities;
15    public Sprite Preview;
16    public Color Color = Color.green;
17 }
```

Codeausschnitt 5.14: CharacterData.ts des Prototyps

```

1 public class Timeline
2 {
3     [SerializeField]
4     private GameObject _character = null;
5     [SerializeField]
6     private int _level = 0;
7     [SerializeField]
8     private List<Timeline> _children = new List<Timeline>();
9     [SerializeField]
10    private float _startTimestamp;
11    [SerializeField]
12    private float _endTimestamp;
13    [SerializeField]
14    private CharacterData _characterData;
15    [SerializeField]
16    private Timeline _parent = null;
17    [SerializeField]
18    private string _id;
19    [SerializeField]
20    private RectTransform _ui = null;
21    [SerializeField]
22    private RectTransform _link = null;
```

```
23     [SerializeField]
24     private Dictionary<string, List<Interaction>> _interactions = new Dictionary<
25         string, List<Interaction>>();
26     [SerializeField]
27     private Dictionary<string, float[]> _startPositions = new Dictionary<string,
28         float[]>();
29     [SerializeField]
30     private Dictionary<string, float[]> _startRotations = new Dictionary<string,
31         float[]>();
32     [SerializeField]
33     private Dictionary<string, float[]> _endPositions = new Dictionary<string, float
34         []>();
35     [SerializeField]
36     private Dictionary<string, float[]> _endRotations = new Dictionary<string, float
37         []>();
38     [SerializeField]
39     private Dictionary<string, bool> _customBools = new Dictionary<string, bool>();
40     [SerializeField]
41     private Dictionary<string, string> _customStrings = new Dictionary<string, string
42         >();
43     [SerializeField]
44     private Dictionary<string, float> _customFloats = new Dictionary<string, float>()
45     ;
46     ...
47     public void TakeCharacter(Character player)
48     {
49         if (_id != null)
50         {
51             return;
52         }
53         _characterData = player.GetCharacterData();
54         string parentID = "";
55         if (_parent != null)
56         {
57             parentID = _parent.GetId();
58         }
59         _id = _startTimestamp.ToString() + "_" + _level.ToString() + "_" + parentID +
60             "_" + _characterData.name;
61     }
62     ...
63 }
```

Codeausschnitt 5.15: Ausschnitt aus Timeline.ts dieses Prototyps

In der Zeitlinie wird jeweils das Abbild der Spielwelt abgespeichert. Das heißt, jedes Objekt, das im Spielsystem eine Referenz auf die derzeitig aktuelle Zeitlinie erhält, kann seine Startposition, Startrotation, Endposition und Endrotation sowie seinen aktuellen Zustand auf der Zeitlinie speichern. Diese Zustände werden, nachdem bestimmte Zeitlinien geladen werden müssen, auf diesen Objekten wieder angewandt. Jede Zeitlinie hat dabei verschiedene weitere Zeitlinien untergegliedert und enthält ebenso eine Referenz auf die Zeitlinie, an welche sie selbst angegliedert wurde. Dadurch entsteht eine baumähnliche Verzweigung, wie man sie in "Abbildung 87" dargestellt sehen kann. Diese Baumform ermöglicht später, dass das System rekursiv durch diesen Baum iterieren kann und dabei die gewünschten Zeitlinien identifiziert werden können.

Jede Zeitlinie erhält eine eindeutige und unwiederholbare Identifikationsnummer (ID). Diese Zeitlinie besteht aus der `_parentTimeline`, der in der `CharacterData` enthaltene ID des Chronologen-Ichs, dem `_level` auf welchem sich diese Zeitlinie befindet und den Startzeitpunkt, wann diese Zeitlinie erstellt wurde.

5.3.4.2. GraphController

Der `GraphController` ist für das System des Levels das zentrale Element. In ihm wird der Zeitlinienbaum aufgebaut und gespeichert. Dadurch hat der `GraphController` den gesamten Überblick über die Ichs des Chronologen. Der `GraphController` hält ebenso eine UI Referenz auf die Steuereinheit, wodurch der Spieler jederzeit den Graphen der Zeitlinien einsehen kann.

In "Abbildung 96" ist das Klassendiagramm abgebildet. Der `GraphController` besitzt ebenfalls die für das System wichtigen Actions. Diese Actions werden im Folgenden vorgestellt.

GraphController.cs	
<ul style="list-style-type: none"> - _playerController: CharacterController - _timeController: TimeController + MoveBackTo: Action<Timeline> + ResetTimeline: Action + TakeDamage: Action<float> + TreeUpdated: Action<Timeline> + Exit: Action<GraphController> + LoadTimeline: Action<Timeline> + SaveTimeline: Action<Timeline> + TakeCurrentTimeline: Action<Timeline> - _timeStability: TimeStability - _timelineToDelete: List<Timeline> - _currentTimeline: Timeline - _rootTimeline: Timeline - _timelineToHandle: List<Timeline> - _environmentSoundController: EnvironmentSoundController + MoveFromHere: Action<Timeline> - _timestamp: float - _warningCount: int - _foundByCharacterData: Timeline - _mergeIn: Timeline - _selectionType: SelectionType - _isInUse: bool + AfterSplit: Action<GraphController> + BeforeSplit: Action<GraphController> + AfterMerge: Action<GraphController> + BeforeMerge: Action<GraphController> + StartGraph: Action - _executable: List<Executable> 	<ul style="list-style-type: none"> + OnHandleTimeUpdate(time: float): void - OnRemoveOldTimelines(time: float): void - RemoveOldTimelines(time: float, timeline: Timeline): void + OnTakeWarning(): void + GetCurrentTimeline(): void + GetTimelineByCharacterData(characterData: CharacterData): void - GetByCharacterData(timeline: Timeline, data: CharacterData): void + HandleTimelineManager(time: float): void + TakeSceneController(sceneController: SceneController): void - OnAfterUpdate(graphController: GraphController): void - OnUpdateExecutable(g: GraphController): void - GetExecutable(): void - OnStartGraph(): void - OnBeforeClosingLevel(): void + IsSelectorInUse(data: CharacterData): bool + ResolveSelection(data: CharacterData): SelectionType - InUseValidation(timeline: Timeline, selection: string): void - SelectionTypeValidation(timeline: Timeline, characterData: CharacterData): void - OnSelection(selection: CharacterData): void - OnSplit(): void - OnMerge(): void - OnRemoveOldPlaceholder(graphController: GraphController): void - RemoveOldPlaceholder(timeline: Timeline): void - OnTimelinesToHandle(graphController: GraphController): void - ValidateTimelinesToHandle(timeline: Timeline): void + AreChildrenInside(): bool + OnResetSplittedTimelines(): void - RemoveTimelines(): void - GetTimelinesToDelete(timeline: Timelines, toCompare: Timeline, lower: bool): void - OnHandleTimelineError(character: CharacterData): void - Start(): void

Abbildung 96.: Klassendiagramm aus GraphController.cs, (Quelle: eigene Darstellung)

TakeCurrentTimeline: Durch diese Action wird die aktuell neu erstellte Zeitlinien an den *CharacterController* und alle *Recordable* Klassen in der Szene übermittelt. Der *CharacterController* hängt Methoden an diese Action, um den ausgewählten Spielercharakter für den Spieler zu instanziieren. Sobald das Spielobjekt des Chronologen erstellt und seine *Character* Klasse am Spielobjekt initialisiert wurde, wird die *CharacterData* dieses Chronologen in die übergebene Zeitlinie eingetragen. Die *Recordable* Klassen speichern nun lokal in ihrer Klasse die Startposition und Startrotation. Diese abgespeicherten Werte werden bei an der Action *SaveTimeline* angebundenen Methoden in der übergebenen Zeitlinie abgespeichert.

SaveTimeline:

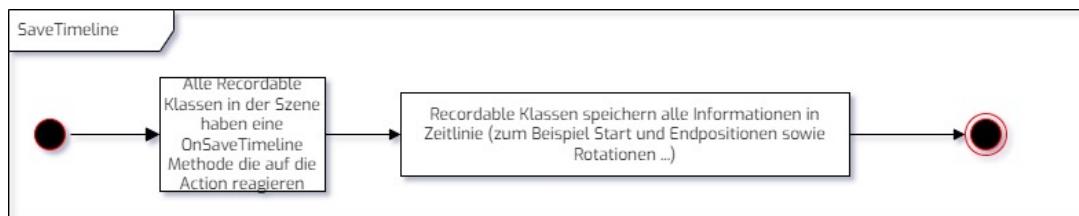


Abbildung 97.: Aktivitätsdiagramm SaveTimeline, (Quelle: eigene Darstellung)

Diese Action wird im *GraphController* aufgerufen, sobald die aktuelle Zeitlinie durch eine andere ausgetauscht wird. Das ist bei einem "Split" und einem "Merge" der Fall. Bevor die *_currentTimeline* ausgetauscht wird, speichert die *Recordable* Klasse die Startposition, Startrotation, Endposition, Endrotation und alle gesammelten Interaktionen in die übergebene Zeitlinie. Zusätzlich können alle Kinderklassen der *Recordable* Klasse ihre eigenen Kindklassen abspeichern. Das wird in "Kapitel 5.3.11: Fehlverhalten der Programmlogik" genauer erklärt.

LoadTimeline:

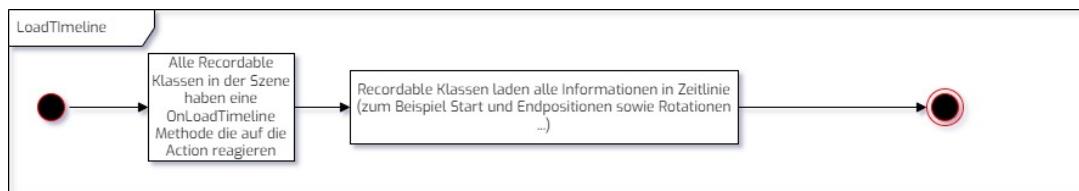


Abbildung 98.: Aktivitätsdiagramm LoadTimeline, (Quelle: eigene Darstellung)

Diese Action wird im *GraphController* zum Ende des "Merges" aufgerufen. Hierbei wird die bereits existierende Zeitlinie, auf die der Spieler zurückwechselt wird, an die

Recordable Klassen übergeben, damit diese ihren Zustand aus dieser Zeitlinie wiederherstellen können. Dabei werden die einzelnen Objekte auf die abgespeicherten Endpositionen und Endrotationen gesetzt, da dies der Zustand war, der zuletzt in dieser Zeitlinie galt. Zusätzlich wird der letzte Zeitpunkt dieser Zeitlinie im *TimeController* gesetzt, da ein “Merge” wie ein “Sprung zurück in der Zeit” ist.

MoveFromHere / MoveBackTo: *MoveFromHere* und *MoveBackTo* sind in der Anwendung ähnliche Actions wie *TakeCurrentTimeline* und *LoadTimeline* allerdings werden diese im “Split” oder “Merge” in der Mitte des Prozesses ausgeführt. Sie sind für die *GroundController* Klasse von Nutzen, da dieser aus den abgespeicherten Startpositionen und Startrotationen des in der Zeitlinie enthaltenen Ichs die Position des Startpunktes gesetzt werden kann.

5.3.4.3. TimeController

Der *TimeController* ist für das Zählen der Spielzeit zuständig. Bei Beginn des Levels wird das Zählen der Spielzeit gestartet. Durch den Timer, der im *TimeController* vorhanden ist, kann nun festgestellt werden, wann Zeitlinien erstellt und beendet werden. Außerdem ist es nun möglich Interaktionen mit einem eindeutigen Zeitstempel zu versehen, damit diese nach einem “Merge” zum richtigen Zeitpunkt und in der richtigen Reihenfolge rekonstruiert werden können.

Für das System ist es wichtig, dass die Zeitpunkte immer gleich reproduziert werden. Sonst kann es passieren, dass Interaktionen übersprungen oder falsche Interaktionen zum Gesehenen ausgeführt werden. Um in Unity bei der Zeitzählung über ein deterministisches System zu verfügen, wird die Methode *FixedUpdate()* verwendet. Sie ist unabhängig der Frame-Rate des Unity Spiels und kann somit reproduziert werden (vgl. Technologies (2023)). Die Frame-Rate des Spiels kann sich ändern, dadurch ist es möglich, dass eben genannte Fehler auftreten und das System nicht mehr funktioniert.

```

1 public class TimeController : MonoBehaviour
2 {
3     ...
4     void FixedUpdate()
5     {
6         if (_isCounting)
7         {
8             TakeTimestamp?.Invoke(_timestamp);
9
10            _timestamp += Time.deltaTime;
11
12        }
13    }
14    ...
15 }
```

Codeausschnitt 5.16: Ausschnitt aus TimeController.ts dieses Prototyps

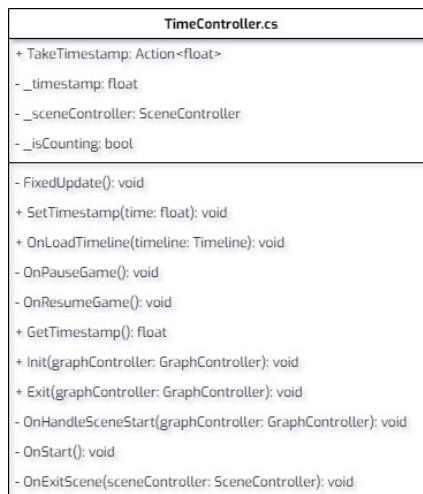


Abbildung 99.: Klassendiagramm der TimeController.cs, (Quelle: eigene Darstellung)

In “Abbildung 99” ist das Klassendiagramm des *TimeControllers* abgebildet.

5.3.4.4. TimeStability

Die *TimeStability* Klasse bildet das Kontinuum ab. Sobald der Spieler nach einem “*Split*” auf seine ursprüngliche Zeitlinie “zurückgemerget” ist, wird das Kontinuum im Kontext der parallelen Zeitlinien instabiler. In diesem Prototyp wird das als ein Faktor dargestellt, die nach bestimmten Zuständen an Werten verliert. Sobald mindestens zwei Zeitlinien parallel zur selben Zeit aktiv sind und die Chronologen einen Einfluss auf die Spielwelt haben, sinkt der Stabilitätswert des Kontinuums. Sobald der Wert der Stabilität auf einen kritischen Punkt gesunken ist, aktiviert es eine Action, die den *GraphController* und den *SceneController* anweist, alle aktuellen aktiven Zeitlinien, außer die des Hauptchronologen, zu löschen und dass der Spieler wieder auf die Zeitlinie des Chronologen wechseln muss.

5.3.4.5. Recordable

Recordable Klassen sind Kernobjekte der folgenden Aufnahmen und Replay Systems. Sie können ihre Zustände und Positionen in der aktuellen Zeitlinie abspeichern, und bei einem Laden der Zeitlinien ihre Zustände wiederherstellen.

Damit diese Klasse auf die Actions des *GraphControllers* reagieren, besitzen sie wie in folgender „Abbildung 100“ entsprechende Methoden.

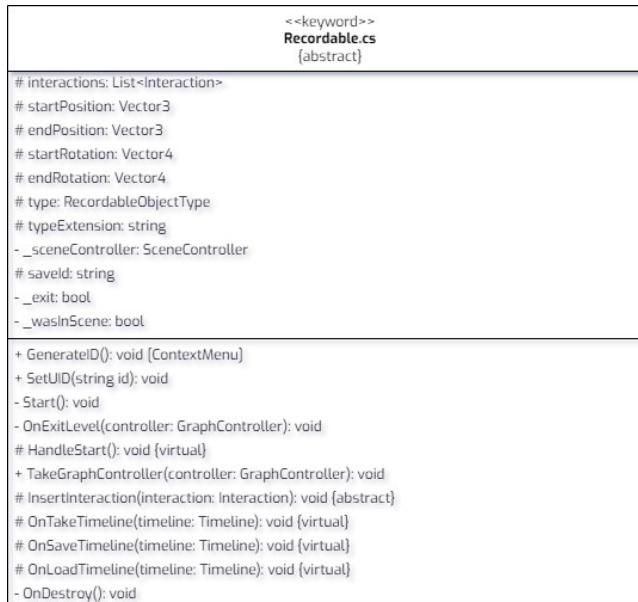


Abbildung 100.: Klassendiagramm der Recordable.cs, (Quelle: eigene Darstellung)

Sie ist eine abstrakte Klasse, die als eine Blaupause für die vererbten Klassen dient. Wie in der Abbildung zu sehen, besitzt sie die Methoden *OnTakeTimeline*, *OnLoadTimeline* und *OnSaveTimeline*. Diese werden mit dem Kennzeichner *virtual* gekennzeichnet, damit diese von den Kind-Klassen überschreiben werden können. Es ist dadurch möglich, dass etwa eine *Crane* Klasse seine eigenen Klassenvariablen abspeichern kann, statt nur die Positionen und Rotationen oder Interaktionen.

Das Chronologen-Modell besitzt eine *Character* Klasse, welches von der Basisklasse *Recordable* erbt. Deswegen besitzt die Klasse *Recordable* eine Liste an Interaktionen, die über die Methode *InsertInteraction* gefüllt wird. Die *Character* Klasse muss seine erzeugten Interaktionen abspeichern und wieder laden können, damit diese reproduziert werden können.

5.3.4.6. Executable

Klassen, die dazu fähig sind Interaktionen zu reproduzieren, besitzen das Interface *Executable* aus „Abbildung 101“.

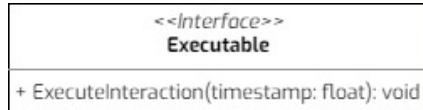


Abbildung 101.: Klassendiagramm des Executable.cs, (Quelle: eigene Darstellung)

Über den *GraphController* können alle Objekte, die ein *Executable* Interface implementiert haben, aufgerufen werden. Der *GraphController* übergibt an alle gefundenen *Executable* Objekte den aktuellen Zeitpunkt des Spiels. Das implementierende Objekt nimmt diesen Zeitpunkt entgegen und untersucht seine Liste an Interaktionen nach der Interaktion dieses Zeitpunktes. Alle Objekte, die ein *Executable* Interface implementiert haben, müssen vom Objekt *Recordable* geerbt haben. Andernfalls ist dieses Interface nicht funktionsfähig.

Im folgenden Codeausschnitt 5.17 wird die Implementierung in die *Character* Klasse gezeigt.

```

1 public class Character: Recordable, Executable
2 {
3     ...
4     public void ExecuteInteraction(float timestamp)
5     {
6         List<Interaction> interaction = \_interactions.FindAll(i => (float.Equals(i.
7             timestamp, timestamp)));
8         if (interaction != null && interaction.Count > 0)
9         {
10            foreach (Interaction i in interaction)
11            {
12                Reconstruct?.Invoke(i);
13            }
14        }
15        ...
16    }
  
```

Codeausschnitt 5.17: Ausschnitt aus Character.ts dieses Prototyps

Nachdem der Zeitpunkt (*timestamp*) entgegengenommen wurde, wird in der Liste der geladenen Interaktionen nach der Interaktion dieses Zeitpunktes gesucht. Da zum Beispiel bei einem Sprung und einer Bewegung nach vorne zwei Interaktionen zum selben Zeitpunkt stattfinden können, wird die Helfermethode *FindAll* genutzt. Sobald Interaktionen zum derzeitigen Zeitpunkt gefunden werden konnten, werden diese über die Action *Reconstruct* weitergeschickt.

Das Kapitel 5.3.10 setzt an diesem Punkt an und erklärt die Aufnahme und Reproduktion der Interaktionen.

5.3.4.7. CharacterController

Im alten Prototyp aus dem Gamedesign Workshops war der *CharacterController* (damals *PlayerController*) das zentrale Element des Spiels. Im überarbeiteten System ist der *CharacterController* nur noch für das Verhalten der unterschiedlichen Objekte des Chronologen zuständig. Sowohl der aktuelle Chronologe als auch seine Platzhalter und Kopien werden wie in der “Abbildung 92” beschriebenen Position des “—Player—” Knoten als Gameobjekte als Childs instanziert. Dadurch hat der *CharacterController*, der auf diesem “—Player—” Objekt liegt, Kontrolle über seine Transform Children. Über den *GroundController* erhält der *CharacterController* das Wissen über die Position des Startpunkts, auf welchem er nach einem “*Split*” die Platzhalter des vergangenen Ichs des Chronologen instanziieren muss, oder die Kopien des Chronologen, die sich nun im örtlichen und zeitlichen Sinn auf der aufgenommene Zeitlinien bewegen werden.

Das Klassendiagramm ist in “Abbildung 102” zu sehen.



Abbildung 102.: Klassendiagramm der CharacterController.cs, (Quelle: eigene Darstellung)

Sobald Kopien des Chronologen in die Spielwelt gesetzt wurden, werden diese ebenfalls in ihrer jeweiligen Zeitlinie referenziert, damit diese, sobald eine Zeitlinie nicht mehr gültig ist, wieder gelöscht werden können, falls diese noch in der Spielwelt existieren.

5.3.4.8. GroundController

Der *GroundController*, dessen Klassendiagramm in “Abbildung 103” zu sehen ist, ist im Kontext der Spielmechanik für das Setzen und Mitteilen der Positionen des Startpunktes wichtig. Er enthält an die Actions *MoveFromHere* und *MoveBackTo* des *GraphControllers* gebundene Methoden, um dadurch aus den übergebenen Zeitlinien die Endposition des in ihr enthaltenen Chronologen Ichs zu laden, damit der Startpunkt auf diese Position gesetzt werden kann.



Abbildung 103.: Klassendiagramm der *GroundController.cs*, (Quelle: eigene Darstellung)

Der *GroundController* übergibt mit seiner eigenen Action *TakeSpawn* dem *CharacterController* die Position des Startpunkts, damit an diesem die Ichs des spielbaren Chronologen instanziert werden können.

Bei der Rekonstruktion der Interaktionen spielt der *GroundController* ebenfalls eine wichtige Rolle, da über ihn überprüft wird, ob Objekte, mit denen interagiert wurden, auch an den richtigen Stelen positioniert sind und ob die Objekte, mit denen der Spielercharakter kollidiert ist, auch der ID des abgespeicherten Objektes entspricht.

5.3.5. Vorstellung des Algorithmus, der entscheidet, wann gesplittet und wann gemerget wird

Sobald der Spieler auf ein Rätsel trifft, muss er in ein anderes Ich des Chronologen „*Splitten*“. Der Spieler weiß, wann er bereits mit einem Chronologen gespielt hat und wann nicht. Das System hat darüber allerdings kein Wissen. Daher wurde folgender Algorithmus entwickelt, der die `_rootTimeline` rekursiv durchchirtert, um zu überprüfen, ob das ausgewählte Ich bereits in einer anderen Zeitlinie existent ist.

Das Aktivitätsdiagramm aus „Abbildung 104“ beschreibt den grundlegenden Ablauf der Überprüfung.

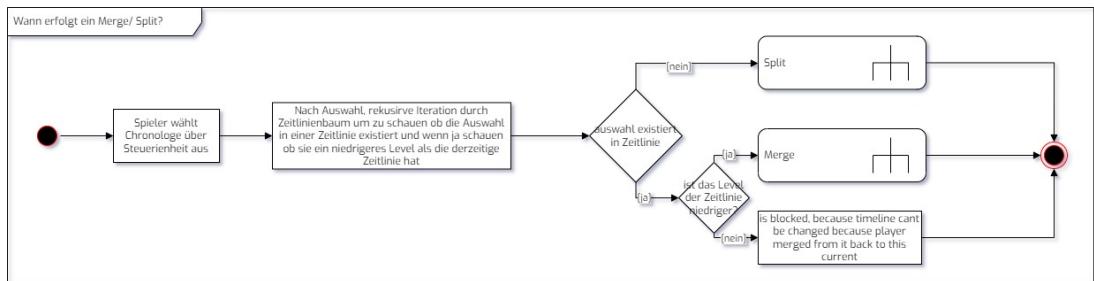


Abbildung 104.: Aktivitätsdiagramm wie überprüft wird, ob ein Split oder Merge Prozess startet, (Quelle: eigene Darstellung)

```

1 public class GraphController: MonoBehaviour
2 {
3     ...
4     public bool IsSelectorInUse(CharacterData data)
5     {
6         _mergeIn = null;
7         _isInUse = false;
8         InUseValidation(_rootTimeline, data.ID);
9         return _isInUse;
10    }
11    private void InUseValidation(Timeline timeline, string selection)
12    {
13        if (selection.Equals(timeline.GetCharacterData().ID))
14        {
15            _mergeIn = timeline;
16            _isInUse = true;
17            if (timeline.GetLevel() > _currentTimeline.GetLevel() && timeline.
18                GetEndTimestamp() != 0 && timeline.GetEndTimestamp() < _timestamp)
19            {
20                _isInUse = false;
21                _mergeIn = null;
22            }
23        }
24        else
25        {
26            List<Timeline> children = timeline.GetChildren();
27            if (children.Count > 0)
28            {
29                foreach (Timeline child in children)
29                {
  
```

```

30             InUseValidation(child, selection);
31         }
32     }
33 }
34 public SelectionType ResolveSelection(CharacterData data)
35 {
36     _selectionType = SelectionType.FREE;
37     SelectionTypeValidation(_rootTimeline, data);
38     return _selectionType;
39 }
40 private void SelectionTypeValidation(Timeline timeline, CharacterData
characterData)
41 {
42     if (timeline.GetCharacterData().ID == characterData.ID)
43     {
44         if (timeline.GetLevel() > _currentTimeline.GetLevel())
45         {
46             _selectionType = SelectionType.BLOCKED;
47             if (timeline.GetEndTimestamp() < _timestamp)
48             {
49                 _selectionType = SelectionType.FREE;
50             }
51         }
52         if (timeline.GetLevel() < _currentTimeline.GetLevel())
53         {
54             _selectionType = SelectionType.MERGE;
55         }
56         if (timeline.GetLevel() == _currentTimeline.GetLevel())
57         {
58             _selectionType = SelectionType.CURRENT;
59         }
60     }
61     else
62     {
63         List<Timeline> children = timeline.GetChildren();
64         if (children.Count > 0)
65         {
66             foreach (Timeline child in children)
67             {
68                 SelectionTypeValidation(child, characterData);
69             }
70         }
71     }
72 }
73 private void OnSelection(CharacterData selection)
74 {
75     _isInUse = false;
76     _mergeIn = null;
77     if (IsSelectorInUse(selection))
78     {
79         GameController.Instance.currentSceneController.Merge?.Invoke();
80         _enviromentSoundController.PlayEnviromentSound(EnviromentSoundType.MERGE)
81     };
82     }
83     else
84     {
85         GameController.Instance.currentSceneController.Split?.Invoke();
86         _enviromentSoundController.PlayEnviromentSound(EnviromentSoundType.SPLIT)

```

```
87     ;
88 }
89 ...
90 }
```

Codeausschnitt 5.18: Split/Merge Überprüfung aus GraphController.ts

Die Auswahl des neuen Ichs erfolgt nach zwei Stufen. Das Ergebnis der ersten Stufe erhält der Spieler auf der Steuereinheit als visuelles Feedback der Auswahl. Die zweite Stufe entscheidet, ob eine “*Split*” oder “*Merge*” Action ausgelöst werden soll.

Sobald der Spieler die Steuereinheit geöffnet hat, kann er das Zahnrad drehen, auf dem die unterschiedlichen Ichs des Chronologen gespeichert sind. Sobald ein Ich ausgewählt wurde, wird die Methode *ResolveSelection()* aufgerufen. Sie übergibt die Auswahl und die *_rootTimeline* an die *SelectionTypeValidation()* Methode, welche eine rekursive Methode ist. Diese geht nun alle im Graphenbaum enthaltene Zeitlinien durch und übergibt die gefundene Zeitlinie an sich zurück.

Die übergebene Zeitlinie wird nun so lange durchforstet, bis die Zeitlinie gefunden wurde, welche die ausgewählte *CharacterData* enthält (*timeline.GetCharacterData().ID == characterData.ID*). Im nächsten Schritt wird überprüft, auf welcher Ebene sich diese Zeitlinie zur derzeitig aktuellen Zeitlinie im Verhältnis des Levels (*timeline.GetLevel()*) verhält. Liegt die *_currentTimeline* auf demselben Level wie die Zeitlinie der Auswahl, so ist es dieselbe Zeitlinie und kann nicht ausgewählt werden. Die Methode *ResolveSelection()* würde *SelectionType.CURRENT* als Rückgabewert an die Steuereinheit übermitteln. Ist die gefundene Zeitlinie im Level unterhalb also größer als die derzeitige Zeitlinie (vgl. *timeline.GetLevel() > _currentTimeline.GetLevel()*), so hat der Spieler bereits einen “*Split*” zu der Zeitlinie ausgeführt und kann diese nun nicht mehr verändern und auch nicht erneut erzeugen. Der Chronologe aus dieser Zeitlinie darf immer nur einmal existieren. Sofern sich der Spieler nicht auf der *_rootTimeline* befindet, existiert das Ich dieser Zeitlinie noch im Graphenbaum der *_rootTimeline*. Befindet sich der Spieler auf der *_rootTimeline*, so wird eine bereits abgeschlossene Zeitlinie aus dem Baum gelöscht und kann erneut erzeugt werden. Da er sich beispielhaft nun nicht auf der *_rootTimeline* befindet, kann der Spieler in dieses Ich erneut “*Splitten*”, sobald die Zeitlinie abgelaufen ist und der derzeitige Zeitpunkt sich nach dem Endzeitpunkt der Zeitlinie befindet (*(timeline.GetEndTimestamp() < timestamp)*).

Sobald der Spieler das Zahnrad dreht und das Level der gefundenen Zeitlinie kleiner ist als das derzeitige (*timeline.GetLevel() < _currentTimeline.GetLevel()*) kann er in diese Zeitlinie “zurückmergen”.

Da im vorherigen Überprüfungsapparat nun die derzeitige und die bereits gesplittet Zeitlinie ausgeschlossen wurde, muss die zweite Überprüfung ausschließlich identifizieren, ob die Zeitlinie ein niedrigeres Level hat. Ist das der Fall, würde die Methode *IsSelectorInUse()* den Wert “true” zurückgeben und die angebundenen Methode auf die Charakterauswahl *OnSelection()* würde die Action *Merge* ausführen. Würde die rekursive Methode *InUseValidation()* keine bereits existierende Zeitlinie finden, so würde die Methode *IsSelectorInUse()* den Wert “false” zurückgeben und der Spieler würde in ein weiteres Ich des Chronologen “splitten”. Es gibt einen Sonderfall bei der Überprüfung in der Methode *InUseValidation()*. Dieser Fall kann dann auftreten, sobald sich der Spieler nicht auf der *_rootTimeline* befindet. Es kann nun ebenfalls wie in der ersten Überprüfung der Fall sein, dass eine Zeitlinie bereits als Angliederung existiert, aber schon abgelaufen ist. Befindet sich der Spieler auf der *_rootTimeline* wird diese Zeitlinie nach Ablauf der Gültigkeit gelöscht. Dieses Verhalten wird in “Kapite 5.3.9” behandelt. Tritt dieser Fall in der Überprüfung von Codezeile 17 (vgl. Codeausschnitt 5.18) auf, so kann eine weitere Zeitlinie mit derselben *CharacterData* erzeugt werden. Dadurch gibt die Methode *IsSelectorInUse()* ebenfalls den Wert “false” zurück und der Spieler “splittet” erneut in dieses Ich.

5.3.6. Vorstellung des Splits

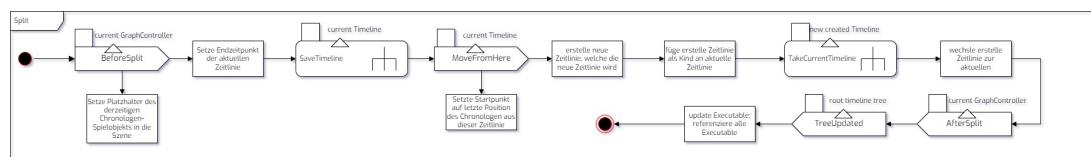


Abbildung 105.: Aktivitätsdiagramm eines Splits, (Quelle: eigene Darstellung)

In “Abbildung 105” ist der Ablauf des “*Splits*” in Form eines Aktivitätsdiagramms dargestellt. Er wird im Folgenden mithilfe des Klassendiagramms in “Abbildung 106” beschrieben. Die Zahlen in den Klammern beziehen sich auf die markierten Pfeile des Klassendiagramms.

Nachdem über den *SceneController* die *Split* Action ausgeführt wurde, startet die Methode *OnSplit()* des *GraphController*s seine Tätigkeit. Zunächst führt er zum Start der Methode die *BeforeSplit* Action aus, damit Klassen, die an diese Action angebundene Methoden haben, zum Start des “*Splittes*” Aktionen ausführen können. Der *CharacterController* erstellt vor dem “*Split*” Prozess einen Platzhalter an der Stelle, an der der Spieler vor dem “*Split*” mit dem Chronologen stand (vgl. Markierungen 1 bis 4). Im nächsten Schritt wird der jetzige Zeitpunkt als Endzeitpunkt in die *_currentTimeline* gesetzt (vgl. Markierung 5). Im Anschluss wird der Zustand der Spielwelt auf die derzeit aktive Zeitlinie über die Action *SaveTimeline* gespei-

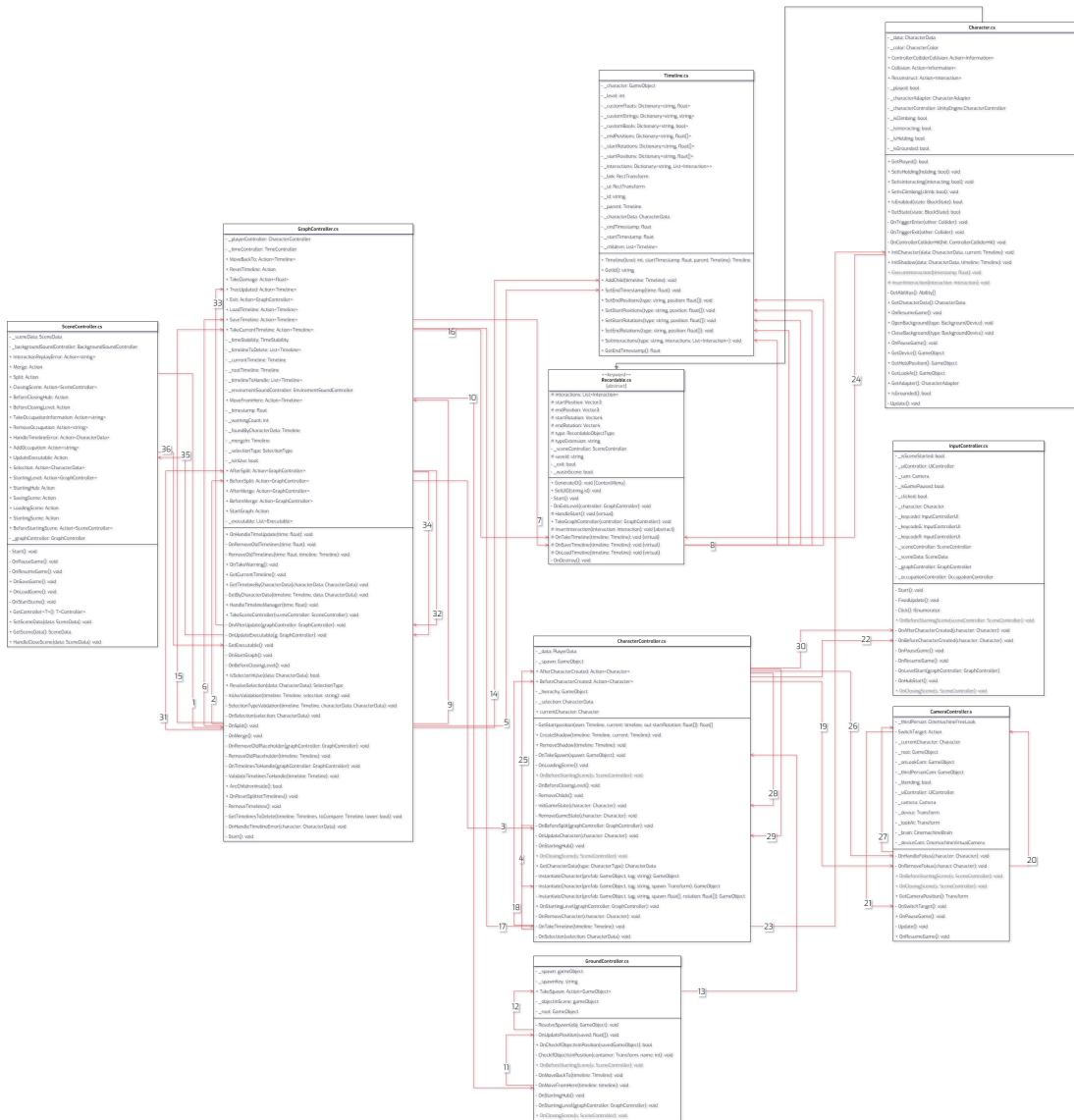


Abbildung 106.: Klassendiagramm eines Splits mit der OnTakeTimeline Methode in der CharacterController.cs, (Quelle: eigene Darstellung), (In groß im Anhang A 1.3.2)

chert. Die Klasse *Recordable* besitzt an diese Action angebundene Methoden und speichert seine gesammelten Startpositionen, Startrotationen, Endpositionen, Endrotationen und gesammelte Interaktionen in der Zeitlinie ab (vgl. Markierungen 7 und 8). Im nächsten Schritt wird die Action *MoveFromHere* ausgeführt, durch welche die noch derzeitige Zeitlinie an den *GroundController* übermittelt wird, damit dieser den Startpunkt auf die Stelle der letzten Position dieser Zeitlinie des Chronologen setzen kann. Durch die Action *TakeSpawn* des *GroundControllers* erhält der *CharacterController* die Information über die Positionen des Startpunkts. (vgl. Markierungen 9 bis 13). Im Anschluss wird eine neue Zeitlinie erstellt, die die *_currentTimeline* als *Parent* erhält. Diese neu erstellte Zeitlinie wird nun an die derzeitige Zeitlinie als *Child* angegliedert (vgl. Markierung 14). Im nächsten Schritt

erfolgt das Ausführen der *TakeCurrentTimeline* Action, die ebenfalls angebundene Methoden in der *Recordable* Klasse besitzt. In Bezug auf die neue Zeitlinie werden nun die Klassenvariablen *_startPositions* und *_startRotations* neu gesetzt, da diese jeweils den Startpunkt und die Startrotation in einer neuen Zeitlinie darstellen. Der *CharacterController* hat ebenfalls Methoden auf diese Action angehängt. Die Action *TakeCurrentTimeline* ist für den *CharacterController* das Signal den alten gespielten Chronologen durch den neu ausgewählten Chronologen zu tauschen (vgl. Markierungen 15 bis 17). Nach der Beschreibung des “*Splits*” wird der Prozess des Wechsels anhand des Klassendiagramms und eines Aktivitätsdiagramms erklärt. Nachdem die neu erstellte Zeitlinie die vergangene Zeitlinie als *_currentTimeline* ersetzt hat, wird die Action *AfterSplit* ausgeführt. Dadurch wird für das UI die Action *TreeUpdate* ausgeführt, die den gesamten Graphenbaum übermittelt. Außerdem wird über die Action *UpdateExecutable* im *SceneController* die Methode *OnUpdateExecutable()* des *GraphControllers* aufgerufen, welche die Anweisung in der Methode *GetExecutable* gibt, alle *Executable* Klassen aus der Szene zu sammeln, damit diese gegebenenfalls ihre Aufnahmen rekonstruieren können (vgl. Markierungen 31 bis 36).

5.3.7. Vorstellung der Methode *OnTakeTimeline*

Wie bereits in “Kapitel 5.3.6: Vorstellung des *Splits*” angesprochen, wird nun die Methode *OnTakeTimeline* im *CharacterController* erklärt.

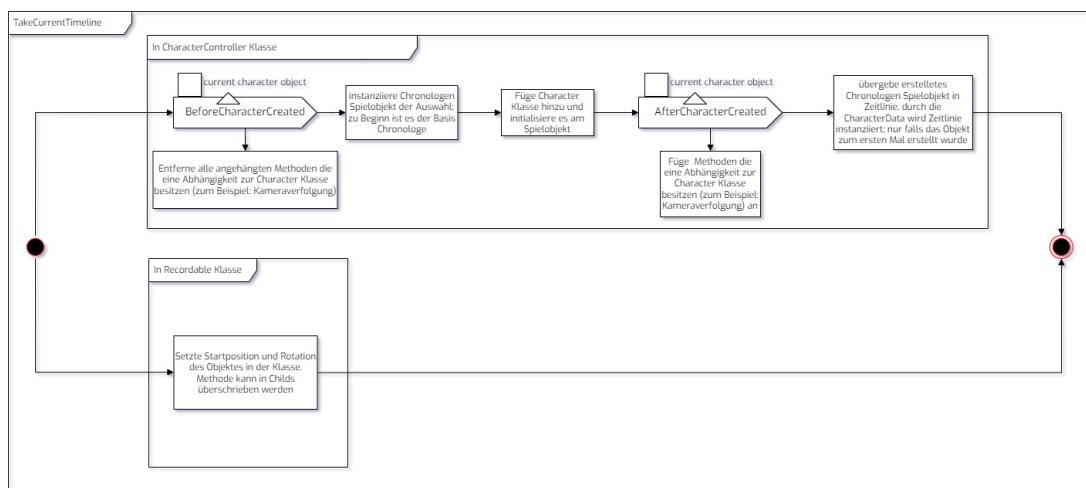


Abbildung 107.: Aktivitätsdiagramm der *TakeCurrentTimeline* Methode, (Quelle: eigene Darstellung)

Anhand der Abbildungen 107 und 106 wird die Methode des *CharacterControllers* erklärt.

Zu Beginn der Methode führt der *CharacterController* die Action

BeforeCharacterCreated aus. Dadurch werden alle Referenzen auf die vorherige *Character* Klasse entfernt. Dazu zählt die Kamera, die in der *Cinemashine* nun nicht mehr dem alten Chronologen folgen muss, und der *InputController* über den die Spielerinputs an die jeweiligen *Abilitys* des Chronologen weitergereicht werden können (vgl. Markierungen 18 bis 22). Nachdem der Spieler über die Steuereinheit seine Auswahl getroffen hat, wird diese Auswahl ebenso im *CharacterController* in der Variable *_selection* abgespeichert, wodurch er nun das 3D Objekt des neuen Chronologen an den vom *GroundController* übermittelten Startposition in die Spielwelt setzen kann. Im nächsten Schritt fügt der *CharacterController* die *Character* Klasse an das Spielobjekt und initialisiert es über die *Init()* Methode. Im selben Schritt ruft die *Character* Klasse seine von der *Recordable* vererbte Methode *OnTakeTimeline()* auf, um seine Startposition und Startrotation zu setzen (vgl. Markierungen 23 und 24). Nachdem die neue Version des Chronologen initialisiert wurde, wird die Action *AfterCharacterCreated* aufgerufen, damit die Abhängigkeiten, die bei der Action *BeforeCharacterCreated* entfernt wurden, wieder hinzugefügt werden können. Zusätzlich wird der neu erstellte Chronologe im *CharacterController* als Referenz abgespeichert, damit für einen nächsten Split, der richtige Platzhalter an der richtigen Stelle erstellt werden kann (vgl. Markierungen 25 bis 30). Im letzten Schritt wird nun der neu erstellte Platzhalter in die durch die Action *TakeCurrentTimeline* übergebene Zeitlinie übergeben, damit diese eine eindeutige und unwiederholbare ID erstellen kann.

5.3.8. Vorstellung des Merges

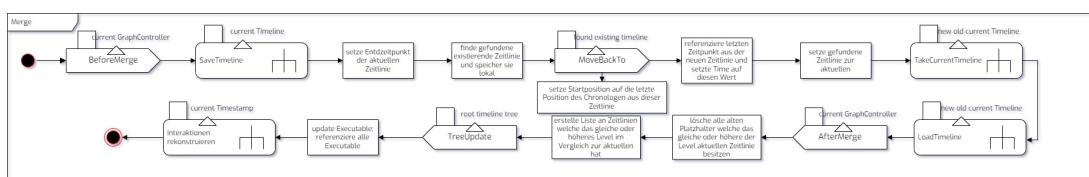


Abbildung 108.: Aktivitätsdiagramm eines Merges, (Quelle: eigene Darstellung), (In groß im Anhang A 1.3.2)

Die folgende Erklärung bezieht sich auf das in „Abbildung 108“ abgebildete Aktivitätsdiagramm und das mit Methodenaufrufen beschriftete Klassendiagramm aus „Abbildung 109“.

Nachdem der Algorithmus erkannt hat, dass ein „Merge“ Prozess gestartet werden muss, wird über die Action *Merge* im *SceneController* die Methode *OnMerge()* aufgerufen. Dieser startet mit der Action *BeforeMerge* damit Aktivitäten vor dem Merge Prozess gestartet werden können (vgl. Markierungen 1 und 2). Im Anschluss wird die Action *SaveTimeline* ausgeführt, damit der aktuelle Zustand der Spielwelt

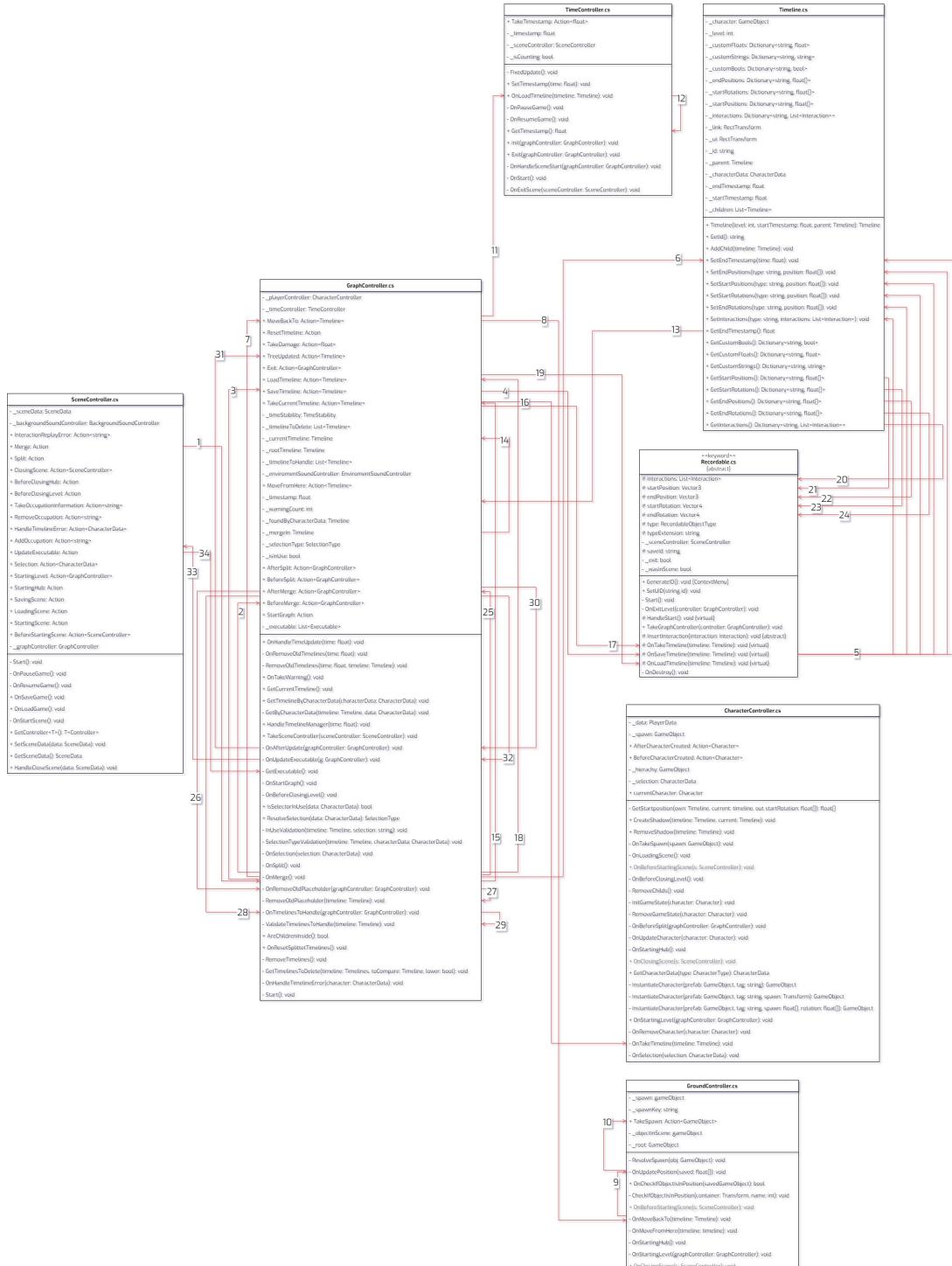


Abbildung 109.: Klassendiagramm eines Merges ohne OnTakeTimeline Methode in CharacterController.cs, (Quelle: eigene Darstellung), (In groß im Anhang A 1.3.2)

auf die übergebene aktuelle Zeitlinie gespeichert werden kann (vgl. Markierungen 3 und 4). Da der Zustand der Spielwelt zu dieser Zeitlinie nun abgespeichert wurde, wird der aktuelle Zeitpunkt als Endzeitpunkt dieser Zeitlinie festgelegt und ebenfalls abgespeichert (vgl. Markierungen 5 und 6). In der Methode *OnSelection()* aus Codeausschnitt 5.18 speichert bei der Überprüfung, ob ein “Merge” Prozess starten soll, die gefundene Zeitlinie lokal im *GraphController* ab, damit sie im Verlauf des “Mergeprozesses” verwendet werden kann. Diese gefundene Zeitlinie *_mergeIn* ist die Zeitlinie, die nun wieder geladen und zu welcher zurückgekehrt werden muss. Diese Zeitlinie wird nun bei der Action *MoveBackTo* übergeben, damit der *GroundController* die Position des Starts setzen und ihn an den *CharacterController* weiterreichen kann. Zusätzlich setzt der *TimeController* nun den Timer auf den letzten Zeitpunkt der Zeitlinie, damit der Timer ab dieser Zeit weiterlaufen kann. Der *GraphController* holt sich ebenfalls den letzten Zeitpunkt aus der Zeitlinie und setzt ihn auf die in der Klasse gespeicherte Referenz, die vom *TimeController* übergeben wurde. Das ist für den späteren Verlauf des “Merges” wichtig (vgl. Markierungen 7 bis 13). Die in der Methode gespeicherte Zeitlinie auf die “zurückgemerget” wird, wird nun die neue, *_currentTimeline* welche bei der Action *TakeCurrentTimeline* übergeben wird. Hierbei wird das Ich aus der geladene Zeitlinie in die Szene gesetzt, damit der Spieler diesen Charakter steuern kann. Außerdem werden die fortgesetzten Startpositionen der *Recordable* in ihrer Klasse gesetzt, welche bei einem “Merge” nicht mehr wichtig sind, da diese Objekte in der Zeitlinie bereits eine Startposition besitzen, welche nicht überschrieben werden kann (vgl. Markierungen 14 bis 17). Nachdem das Spielobjekt Abbild des Chronologen erstellt und seine Abhängigkeiten gesetzt wurden, wird über die Action *LoadTimeline* die aktuelle *_currentTimeline* übermittelt, damit die Spielwelt und die damit einhergehenden *Recordable* Objekte den Zustand dieser Zeitlinien wiederherstellen können. Dazu laden die *Recordable* Objekte in ihrer Methode *OnLoadTimeline()* alle wichtigen Informationen und setzen diese in ihren lokalen Variablen. Zusätzlich setzen sie die geladene Endposition der Zeitlinie auf ihre Transform Komponente, damit sie wieder in die richtige Position gesetzt werden kann (vgl. Markierungen 18 bis 24). Zum Ende des grundlegenden “Mergeprozesses” wird die Action *AfterMerge* ausgeführt (vgl. Markierung 25).

```
1 public class GraphController: MonoBehaviour
2 {
3     ...
4     private void OnRemoveOldPlaceholder(GraphController graphController)
5     {
6         RemoveOldPlaceholder(_rootTimeline);
7     }
8     private void RemoveOldPlaceholder(Timeline timeline)
9     {
10        if (timeline.GetLevel() >= _currentTimeline.GetLevel())
11        {
12            _playerController.RemoveShadow(timeline);
```

```

13
14     List<Timeline> children = timeline.GetChildren();
15     if (children.Count > 0)
16     {
17         foreach (Timeline child in children)
18         {
19             RemoveOldPlaceholder(child);
20         }
21     }
22 } else
23 {
24     List<Timeline> children = timeline.GetChildren();
25     if (children.Count > 0)
26     {
27         foreach (Timeline child in children)
28         {
29             RemoveOldPlaceholder(child);
30         }
31     }
32 }
33 }
34 }
35 ...
36 }
```

Codeausschnitt 5.19: OnRemoveOldPlaceholder aus GraphController.ts

Im ersten Schritt der *AfterMerge* Action werden zunächst alle alten Platzhalter aus der Szene gelöscht, die aus den angegliederten Zeitlinien, der *rootTimeline* ein höheres oder gleiches Level im Vergleich zur derzeitigen Zeitlinie (*timeline.GetLevel() >= _currentTimeline.GetLevel()*) aufweisen (vgl. Markierungen 26 und 27; Codeausschnitt 5.19).

Im nächsten Schritt wird, wie im alten Prototyp und in Codeausschnitt 5.4 gezeigt, darauf geschaut, welche Zeitlinien ein höheres oder gleiches Level wie die derzeitige Zeitlinie hat. Diese rekursive Methode wurde im hier entstandenen Prototyp folgendermaßen in Codeausschnitt 5.20 übernommen (vgl. Markierungen 28 und 29).

```

1 public class GraphController: MonoBehaviour
2 {
3     ...
4     private void OnTimelinesToHandle(GraphController graphController)
5     {
6         ValidateTimelinesToHandle(_rootTimeline);
7     }
8     private void ValidateTimelinesToHandle(Timeline timeline)
9     {
10        if (timeline.GetLevel() >= _currentTimeline.GetLevel())
11        {
12            if (!timeline.GetId().Equals(_currentTimeline.GetId()))
13            {
14                _timelineToHandle.Add(timeline);
15            }
16        }
17 }
```

```

18     List<Timeline> children = timeline.GetChildren();
19     if (children.Count > 0)
20     {
21         foreach (Timeline child in children)
22         {
23             ValidateTimelinesToHandle(child);
24         }
25     }
26 }
27 ...
28 }
```

Codeauschnitt 5.20: OnTimelinesToHandle aus GraphController.ts

Zuletzt wird, wie bereits bei der Erklärung des “*Splits*” in “Kapitel 5.3.6: Vorstellung des *Splits*”, die Action *UpdateExecutable* im *SceneController* aufgerufen und im *GraphController* darauf reagiert (vgl. Markierungen 30 bis 34).

5.3.9. Vorstellung des HandleTimelineManagers

Der *HandleTimelineManager()* ist die zentrale Logik, die das Replay-System ausführt. Durch die Anweisungen des *HandleTimelineManagers()* werden die richtigen Kopien des Chronologen erzeugt und ebenso die richtigen Interaktionen an die richtigen Kopien übergeben.

```

1 public class GraphController: MonoBehaviour
2 {
3     ...
4     public void HandleTimelineManager(float time)
5     {
6         OnRemoveOldTimelines(time);
7         if (_timelineToHandle != null && _timelineToHandle.Count > 0)
8         {
9             foreach (Timeline timeline in _timelineToHandle.ToArray())
10             {
11                 if (timeline.GetStartTimestamp() <= time && !timeline.GetCharacter())
12                 {
13                     _playerController.CreateShadow(timeline, _currentTimeline);
14                 }
15
16                 if (timeline.GetEndTimestamp() <= time)
17                 {
18                     _playerController.RemoveShadow(timeline);
19                     _timelineToHandle.Remove(timeline);
20                 }
21             }
22         }
23         foreach (Executable executable in _executable.ToArray())
24         {
25             executable.ExecuteInteraction(time);
26         }
27     }
28     private void OnRemoveOldTimelines(float time)
29     {
```

```

30         if (_currentTimeline.GetId().Equals(_rootTimeline.GetId()))
31     {
32         RemoveOldTimelines(time, _rootTimeline);
33     }
34 }
35 private void RemoveOldTimelines(float time, Timeline timeline)
36 {
37     List<Timeline> children = timeline.GetChildren();
38     if (children.Count > 0)
39     {
40         foreach (Timeline child in children.ToArray())
41     {
42         RemoveOldTimelines(time, child);
43     }
44     }
45     else
46     {
47         if (timeline.GetEndTimestamp() <= time)
48     {
49         timeline.GetParent()?.GetChildren()?.Remove(timeline);
50         TreeUpdated?.Invoke(_rootTimeline);
51     }
52     }
53 }
54 private void GetExecutable()
55 {
56     List<Component> components = new List<Component>();
57     Component[] executable = this.GetComponentsInChildren(typeof(Executable));
58     if (executable.Length > 0)
59     {
60         foreach (Component executableComponent in executable)
61     {
62         if (executableComponent.gameObject.name != _playerController.
currentCharacter.gameObject.name)
63         {
64             components.Add(executableComponent);
65         }
66     }
67     }
68     _executable = components;
69 }
70 ...
71 }
```

Codeausschnitt 5.21: HandleTimelineManager aus GraphController.ts

In der *HandleTimelineManager()* Methode wird von der Action *TakeTimestamp* des *TimeControllers* der aktuelle Zeitpunkt des Timers übermittelt. Diese Methode ist in drei Bereiche untergliedert.

Der erste Bereich bezieht sich auf das Instanziieren der Kopien des Chronologen aus den Zeitlinien, die nach einem Merge betrachtet werden müssen. Hierbei wird über eine for-Schleife durch die Liste der *_timelineToHandle* iteriert. Zunächst wird überprüft, ob die Zeitlinie, die in der Liste betrachtet wird, bereits gestartet, es aber noch keine Kopie instanziert wurde (vgl. *timeline.GetStartTimestamp() <=*

`time && !timeline.GetCharacter()).` Trifft diese Bedingung zu, wird über den *CharacterController* die Kopie instanziert (vgl. `playerController.CreateShadow(timeline, currentTimeline)`). Die zweite if-Bedingung bezieht sich auf die Überprüfung, ob eine Zeitlinie bereits abgelaufen ist (vgl. `timeline.GetEndTimestamp() <= time`). Wird bei der Überprüfung der Wert "true" zurückgegeben, so wird die Kopie aus der Szene und der Liste der `_timelineToHandle` gelöscht.

Der zweite Bereich kümmert sich um das Übergeben des derzeitigen Zeitpunktes an die *Executable* Klassen, die im ersten Bereich erstellt und wieder gelöscht werden. Nachdem die Methode `CreateShadow()` des *CharacterControllers* aufgerufen wurde, wird die Action `UpdateExecutable` des *SceneControllers* ausgeführt, auf welcher der *GraphController* angehängte Methoden besitzt. In dieser Methode werden über die Methode `GetComponentsInChildren(typeof(Executable))` alle Objekte aus der Szene referenziert, die das Interface *Executable* erben. Im Fall des Prototyps sind das die Kopien des Chronologen. Sobald diese Liste an *Executablen* existent ist, wird an sie der aktuelle Zeitpunkt übermittelt.

Der dritte Bereich ist die Methode `OnRemoveOldTimelines()`, die zu Anfang der Methode aufgerufen wird. Sie iteriert rekursiv durch den Graphenbaum der `_rootTimeline` und durchforstet ihn, nach abgelaufenen Zeitlinien. Das macht die Methode allerdings nur, wenn sich der Spieler derzeit auf der `rootTimeline` befindet, also wenn die `_currentTimeline` der `_rootTimeline` entspricht. Tritt der Fall `timeline.GetEndTimestamp() <= time` ein, so wird die Zeitlinie aus der Anglie-derung an seinen Parent entfernt.

5.3.10. Vorstellung der Erzeugung und Reproduktion von Interaktionen

In den letzten Kapiteln wurde darüber berichtet, wie das System funktioniert. Es fehlt nun nur noch die letzte Komponente, und zwar das Aufnehmen und Abspielen der Interaktionen. Wie das Aufnehmen und Reproduzieren funktioniert, wird anhand der Interaktion mit einem Hebel erklärt.

Bevor auf das beispielhafte Erklären der Aufnahmen und Reproduktion einer Interaktion mit einem Objekt eingegangen wird, wird zunächst auf den Aufbau der *Abilitys* und *ReconstructAbilitys* eingegangen und wie die *Character* Klasse zentral mit ihnen kommuniziert. In "Abbildung 110" ist das Klassendiagramm des Systems abgebildet.

Zunächst wird auf den Aufnahmezyklus (rote Pfeile) eingegangen. Grundlegend erhalten alle *Ability* Klassen, welche die Fähigkeiten des Spielers darstellen, eine Referenz

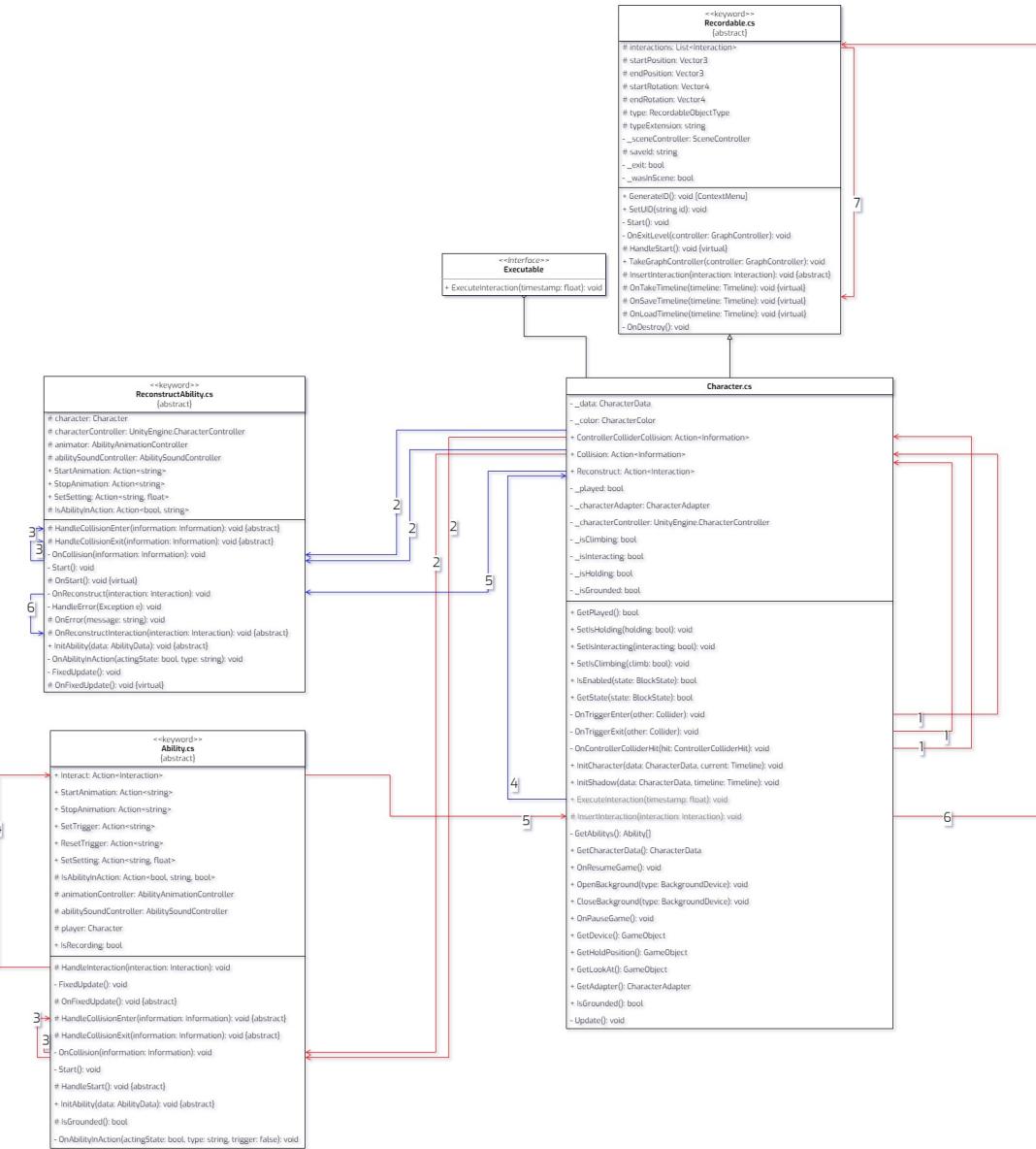


Abbildung 110.: Klassendiagramm der grundlegenden Struktur bei Aufnahme und Reproduktion, (Quelle: eigene Darstellung), (In groß im Anhang A 1.3.2)

auf die Kollisionen, die der *UnityEngineCharacterController* oder der Sphärkollider des Charakter-Objektes auslösen (vgl. Markierungen 1 und 2). Dabei wird eine Action ausgeführt, welche die Information in Form der *Information* Klasse enthalten.

```

1 public enum CollisionType
2 {
3     COLLISION_ENTER, COLLISION_EXIT, NULL
4 }
5 public class Information
6 {
7     public string type;
8     public GameObject origin;
9     public string extension;
10
11    public Information(string type, GameObject origin, string extension)
12    {
13        this.type = type;
14        this.origin = origin;
15        this.extension = extension;
16    }
17}
18 }
```

Codeausschnitt 5.22: Information.cs des Prototyps

Diese *Information* beinhaltet, wie in Codeausschnitt 5.22 zu sehen ist, die Kollisionsart (*Collision_Enter* oder *Collision_Exit*) sowie das Objekt mit dem kollidiert wurde. Zusätzlich wird über den Wert *extension* mitgeteilt, ob es eine Kollision des Sphärkollider oder des *UnityEngine.CharacterController* war.

Über die Methode *OnCollision()* wird in der *Ability* Klasse nun die Art der Kollision an die entsprechenden Methoden *HandleCollisionEnter()* und *HandleCollisionExit()* weitergereicht (vgl. Markierung 3). Die Art der Fähigkeit wird in den Kindklassen der *Ability* Klasse definiert. Sobald eine *Ability* Klasse eine Interaktion ausführt, genauer gesagt der Spieler eine Interaktion in der Welt ausführt, wird die Methode *HandleInteraction()* ausgeführt. Diese gibt über die Action *Interact* die erstellte Interaktion an die *Character* Klasse weiter. Diese Klasse hat Methoden, die auf die *Interact* Action reagieren (vgl. Markierungen 4 und 5). Diese Methode ist eine vererbte Methode der *Recordable* Klasse, welche die entgegengenommene Interaktion in die Liste der Interaktionen speichert, die die *Recordable* Klasse besitzt. Diese Liste an Interaktionen kann nun durch die Methode, die an der Action *SaveTimeline* gebunden ist, gespeichert werden (vgl. Markierungen 6 und 7).

Im Anschluss wird der Rekonstruktionszyklus (blaue Pfeile) erklärt. Wie im Aufnahmemyklus haben auch hier die *ReconstructAbility* Klassen eine Methode, die an die Action *Collision* gebunden ist und auf die Kollisionen der Kopien des Chronologen reagieren. Die Methode unterscheidet ebenso, welche Art der Kollision auf-

trat und übergibt sie an die entsprechenden Methoden (vgl. Markierungen 1 bis 3). Nachdem die *Recordable* Klasse, genauer gesagt die *Character* Klasse auf der Kopie des Chronologen liegt, und beim Erstellen der Kopie angehängt wird, werden die gespeicherten Interaktionen aus der entsprechenden Zeitlinie geladen. Wie bereits in "Kapitel 5.3.9: Vorstellung des HandleTimelineManagers" erklärt, übergibt der *HandleTimelineManager()* an Klassen, die das Interface *Executable* implementiert haben, den aktuellen Zeitpunkt. Die *Character* Klasse hat das Interface implementiert und sucht nun wie im Codeausschnitt 5.17 die Interaktionen heraus, welche zu diesem übergebenen Zeitpunkt stattfanden. Sobald eine Interaktion gefunden wurde, wird diese mit der Action *Reconstruct* an die angebundenen Methoden der *ReconstructAbility* weitergegeben. Die reagierende Methode gibt sie nun an die Methode *OnReconstructInteraction()* weiter, in welcher die Kindklassen das Rekonstruieren übernehmen (vgl. Markierungen 4 bis 6).

Aufnahme der InteractionAbility

```

1 public class InteractionAbility : Ability, IInteract
2 {
3     private Interactable _interactable;
4     private bool _clicked = false;
5     ...
6     protected override void HandleCollisionEnter(Information information)
7     {
8         if (information.origin.TryGetComponent(out Interactable i))
9         {
10             _interactable = i;
11             _isColidedWithInteractable = true;
12         }
13     }
14     protected override void HandleCollisionExit(Information information)
15     {
16         if (information.origin.TryGetComponent(out Interactable i))
17         {
18             _interactable = null;
19             _isColidedWithInteractable = false;
20         }
21     }
22 }
23 private void OnPullInteractableObject()
24 {
25     _interactable.Interact?.Invoke();
26 }
27 public void TakeInput()
28 {
29     if (_isColidedWithInteractable == false)
30     {
31         return;
32     }
33     if (_clicked)
34     {
35         return;
36     }

```

```
37     if (_interactable == null)
38     {
39         return;
40     }
41     if (!_player.IsEnabled(BlockState.INTERACT))
42     {
43         return;
44     }
45     if(_interactable.IsActive())
46     {
47         return;
48     }
49     _player.SetIsInteracting(true);
50     SetTrigger?.Invoke("Interact");
51     HandleInteraction(new Interaction(InteractionType.INTERACT, _player, Utils.
MigrateGameObjectToSafe(_interactable.gameObject), Utils.
MigrateGameObjectToSafe(_interactable.gameObject), Utils.GetTimestamp()));
52     _abilitySoundController.PlayAbilitySound(AbilitySoundType.INTERACTION);
53
54     StartCoroutine(OnClick());
55 }
56 ...
57 }
```

Codeausschnitt 5.23: Ausschnitt aus InteractionAbility.cs

Über die *HandleCollisionEnter()* Methode des Codeausschnittes 5.23 wird ermittelt, ob eine Kollision mit einem Interaktionsobjekt vorliegt. Ist dem der Fall, wird diese wie in der *HandleCollisionEnter()* Methode zu sehen in die Klassenvariable *_interactable* gespeichert, damit diese in der vererbten *TakeInput()* Methode referenziert werden kann. In der *TakeInput()* Methode werden, sobald der Spieler eine Eingabe der Taste "E" tätigt, die Fälle überprüft, welche eine Interaktion ausschließen würden. Im ersten Fall wird überprüft, ob eine Kollision mit einem Interaktionsobjekt vorliegt (*!_isColidedWithInteractable == false*). Der zweite Fall behandelt das Überprüfen, ob der Spieler keine andere Tätigkeit währenddessen ausführt, die eine Interaktion mit einem Objekt verhindern würde, etwa ein Objekt tragen (*_player.IsEnabled(BlockState.INTERACT)*). Im letzten Schritt wird überprüft, ob mit dem Objekt nicht bereits interagiert wurde (*!_interactable.IsActive()*). Kann der Spieler mit einem Objekt interagieren, wie die entsprechende Interaktion erstellt und über die *HandleInteraction()* Methode an die *Character* Klasse weitergegeben.

Dabei wird für das Darstellen in der Szene die Animation gestartet, die das Interagieren bspw. mit einem Hebel darstellen soll. Sobald der Punkt in der Animation erreicht wurde, bei dem die Animation den Hebel in die Hand nehmen würde, wird eine Action an die Klasse zurückgeschickt, auf welche die Methode *OnPullInteractableObject()* hört. Hier geschieht nun die Interaktion mit dem Objekt.

Rekonstruktion der ReconstructionInteractionAbility

```

1 public class ReconstructionInteractionAbility : ReconstructAbility
2 {
3     private Interactable _interactable;
4     ...
5     protected override void HandleCollisionEnter(Information information)
6     {
7         if (information.origin.TryGetComponent(out Interactable i))
8         {
9             _interactable = i;
10        }
11    }
12    ...
13    protected override void HandleCollisionExit(Information information)
14    {
15        if (information.origin.TryGetComponent(out Interactable i))
16        {
17            _interactable = null;
18        }
19    }
20    private void OnCompleteInteract()
21    {
22        abilitySoundController.PlayAbilitySound(AbilitySoundType.INTERACTION, false);
23    }
24    protected override void OnReconstructInteraction(Interaction interaction)
25    {
26        if (interaction.type != InteractionType.INTERACT)
27        {
28            return;
29        }
30        if (_interactable == null)
31        {
32            throw new Exception("Das Interaktionsobjekt existiert nicht oder nicht
mehr!");
33        }
34        if (_interactable.gameObject.GetInstanceID() != interaction.interactedObject.
Name)
35        {
36            throw new Exception("Das Intraktionsobjekt in der Naehe stimmt nicht mit
dem interagiertem Objekt berein ");
37        }
38        if (_interactable.IsActive())
39        {
40            throw new Exception("Das Intraktionsobjekt wurde bereits aktiviert.");
41        }
42        abilitySoundController.PlayAbilitySound(AbilitySoundType.INTERACTION);
43        animator.GetAnimator().SetTrigger("Interact");
44        _interactable.Interact?.Invoke();
45    }
46 }
```

Codeausschnitt 5.24: Ausschnitt aus ReconstructionInteractionAbility.cs

Die *ReconstructAbility* Klassen erarbeiten nicht nur die Rekonstruktion der Interaktionen, sondern existieren ebenso zur Laufzeit des Spielobjekts. Wie in der Einleitung dieses Kapitels beschrieben wurde, haben die *ReconstructAbility* Klassen Informationen darüber, mit welchen Objekten kollidiert wurden. Dadurch kann im Vorlauf

einer Interaktion, die in die Methode *OnReconstructInteraction()* übergeben wird, überprüft werden, ob etwa eine Kollision mit einem Interaktionsobjekt vorlag. Dieser Vorlauf ist bei der Validierung der Interaktion wichtig, da auf diese Weise überprüft werden kann, ob das Objekt, mit dem interagiert wurde, überhaupt an dieser Stelle existent ist.

Im ersten Schritt (*_interactable == null*) der *OnReconstructInteraction()* Methode wird überprüft, ob aus der Sicht des Spielobjektes eine Kollision mit einem Interaktionsobjekt stattfand. Wenn nein, dann wird hier eine Exception erzeugt, welche einen Fehler in der Rekonstruktion erkannt hat und damit die Zeitlinie gelöscht wird. Die Zeitlinie wird deshalb gelöscht, weil nicht sichergestellt werden kann, wieso keine Kollision vorlag und in erster Linie der Spieler durch sein Handeln in der Spielwelt diese so verändert hat, dass es einen Einfluss auf die Zeitlinie, die hier rekonstruiert wird, hatte.

Im zweiten Schritt (*_interactable.gameObject.GetInstanceID() != interaction.interactedObject.Name*) wird überprüft, ob das Spielobjekt, mit dem eine Kollision stattfand, das Objekt der Aufnahme ist. Falls dieser Fall eintreten würde, müsste der Spieler bestimmte Objekte vertauscht haben und hat damit ebenfalls in die Rekonstruktion eingegriffen. Daher wird hierbei die Zeitlinie ebenfalls zerstört.

Im dritten Schritt (*_interactable.IsActive()*) wird überprüft, ob der Spieler das Interaktionsobjekt bereits aktiviert hat und damit der Zustand ebenfalls verändert wurde. Dieser Zustand existierte bei der Aufnahme nicht, weshalb die Zeitlinie ebenso gelöscht werden muss.

Sollten keine Ausnahmen bei der Überprüfung aufgefallen sein, so kann die Interaktion auf dieselbe Weise wie die Aufnahme rekonstruiert werden. Die Kopie des Chronologen macht bei einer Rekonstruktion dieselben Aktivitäten, wie es bei der Aufnahme der Fall war.

5.3.11. Fehlverhalten der Programmlogik

Grundlegend werden bei den *Recordable* Klassen die Startposition, Startrotation, Endposition, Endrotation oder auch Interaktionen in den Zeitlinien abgespeichert. Diese fünf Basiswerte sind allerdings nicht bei allen Klassen notwendig. Im Folgenden werden Fälle vorgestellt, bei denen diese fünf Basiswerte nicht notwendig oder nicht einsetzbar sind. Darüber hinaus werden Fälle dargestellt, in denen diese fünf Werte einen anderen Kontext benötigen.

5.3.11.1. Das Erstellen der Chronologen-Kopien

Im Kontext der Reproduktion der Zeitlinien sind diese in sich geschlossen. Die einzelnen Zeitlinien, die nach einem “Merge” in der `_timelineToHandle` Liste liegen, besitzen einen Startzeitpunkt, eine Startposition, sowie einen Zeitpunkt bis wann diese existiert haben und an welcher Position in der Spielwelt sie aufhören. In der Theorie reichen diese Werte aus, um eine Zeitlinie vollständig und ohne Fehler zu rekonstruieren. Allerdings unterscheidet sich diese Theorie von der praktischen Anwendung dieser Spielmechanik.

Anfangs wurde der Charakter der Zeitlinie an seinen Anfangspunkt der Zeitlinie instanziert und wartet anschließend darauf, dass eine Interaktion zu einem übergebenen Zeitpunkt geschieht und die Kopie bewegt wird.

```

1 public class CharacterController : Controller
2 {
3     ...
4     public void CreateShadow(Timeline timeline)
5     {
6         CharacterData data = timeline.GetCharacterData();
7         float[] startPosition = timeline.GetStartPositions() [RecordableObjectType.
8             PLAYER.ToString() + "_" + data.ID];
9         startPosition[1] = startPosition[1] + 0.5f;
10        GameObject shadowObject = InstantiateCharacter(data.Ghost, "Shadow",
11             startPosition);
12        Character character = shadowObject.AddComponent<Character>();
13        character.InitShadow(data, timeline);
14        GameController.Instance.currentSceneController.UpdateExecutable?.Invoke();
15    }
16    ...
17 }
```

Codeausschnitt 5.25: Ausschnitt aus CharacterController.cs

In der `HandleTimelineManager()` Methode aus Codeausschnitt 5.21 wird, sobald eine der Zeitlinien aus der `_timelineToHandle` Liste aktiv werden muss, aus dem `CharacterController` die Methode `CreateShadow()` aufgerufen. Diese Methode instanziert das Objekt des Chronologen, welches der Spieler bei der Aufnahme der Zeitlinie gesteuert hatte, in die Szene, damit die aufgenommenen Interaktionen der Szene rekonstruiert werden können.

In der Theorie würde die Kopie nun seinen aufgenommenen Interaktionen nachgehen und sich in der Welt bewegen. In der Praxis wird der Spieler nun aber einen weiteren “Split” und anschließend einen weiteren “Merge” ausführen. Tritt nun der Fall ein, dass der aufgenommene Chronologe zum Zeitpunkt des “Splits” einen Hebel aktiviert hat und bei diesem Hebel steht, damit dieser aktiviert bleibt, wird die Kopie zunächst gelöscht werden. Das ist in erster Linie nicht das Problem. Das Problem tritt erst beim Instanziieren der Kopie auf. Das Objekt würde nun nämlich an den Anfangspunkt

seiner Zeitlinie gesetzt und wartet darauf, dass Interaktionen stattfinden. Bei der Aufnahme stand der Spieler eine Zeit am Hebel und hat sich nicht bewegt. Die Kopie, die nun in der Welt instanziert wurde, steht nun ebenfalls herum. Allerdings nicht am Hebel, sondern am Anfangspunkt seiner Zeitlinie. In der aufgenommenen Zeitlinie ist der nachfolgende „*Split*“ nicht vorgekommen, weshalb die Zeitlinie den veränderten Startpunkt der Zeitlinie nicht berücksichtigen konnte.

Die Lösung für dieses Problem ist naheliegend, wenn man sich den Aufbau der Mechanik ansieht. Bei einem „*Split*“ wird der aktuelle Zustand der Welt in der Zeitlinie, die gerade durch den Spieler gesteuert wird, abgespeichert. Der Zustand dieser Welt wird im „*Mergprozess*“ wie erwähnt wieder geladen und auf die Welt angewendet. Das bedeutet, die Position des Charakters aus der Zeitlinie, die vor dem zweiten „*Split*“ am Rekonstruieren ist, wird nun beim zweiten Split auf der Zeitlinie abgespeichert. Während des „*Mergvorganges*“ wird dieses Objekt gelöscht und in der *HandleTimelineManager()* Methode erneut instanziert. Die *CreateShadow()* Methode muss nun so angepasst werden, dass zunächst geklärt wird, ob eine Startposition bei der geladenen Zeitlinie existent ist und wenn ja, dann wird die Kopie an dieser Position instanziert. Wenn auf der aktuellen Zeitlinie keine Startposition zu finden ist, dann gab es während der Rekonstruktion keinen weiteren „*Split*“ und das Spielobjekt wird auf der Startposition seiner Zeitlinie positioniert.

Der folgende Codeausschnitt 5.26 beinhaltet diese Lösung.

```
1 public class CharacterController : Controller
2 {
3     ...
4     public void CreateShadow(Timeline timeline, Timeline current)
5     {
6         CharacterData data = timeline.GetCharacterData();
7         float[] startPosition = GetStartPosition(timeline, current, out float[] startRotation);
8
9         if (timeline.GetCharacter() == null)
10        {
11            GameObject shadowObject = InstantiateCharacter(data.Copy, "Shadow",
12                startPosition, startRotation);
13            Character character = shadowObject.AddComponent<Character>();
14            timeline.SetCharacter(character.gameObject);
15            character.InitShadow(data, timeline);
16            GameController.Instance.currentSceneController.UpdateExecutable?.Invoke()
17        }
18        else
19        {
20            GameObject character = timeline.GetCharacter();
21            character.transform.position = new Vector3(startPosition[0],
22                startPosition[1], startPosition[2]);
23            character.transform.rotation = new Quaternion(startRotation[0],
24                startRotation[1], startRotation[2], startRotation[3]);
25        }
26    }
27 }
```

```

23
24     }
25     private float[] GetStartPosition(Timeline own, Timeline current, out float[]
26     startRotation)
27     {
28         CharacterData data = own.GetCharacterData();
29         string key = RecordableObjectType.PLAYER.ToString() + "_" + data.ID;
30         float[] startPosition = new float[3];
31         startRotation = new float[4];
32         if (current.GetEndPositions().ContainsKey(key + "_" + own.GetStartTimestamp()
33             .ToString()))
34         {
35             float[] csP = current.GetEndPositions()[key + "_" + own.GetStartTimestamp
36             ().ToString()];
37             float[] csR = current.GetEndRotations()[key + "_" + own.GetStartTimestamp
38             ().ToString()];
39             startPosition = new float[3] { csP[0], csP[1], csP[2] };
40             startRotation = new float[4] { csR[0], csR[1], csR[2], csR[3] };
41         }
42         else
43         {
44             float[] tsP = own.GetStartPositions()[key + "_" + own.GetStartTimestamp()
45             .ToString()];
46             float[] tsR = own.GetStartRotations()[key + "_" + own.GetStartTimestamp()
47             ().ToString()];
48             startPosition = new float[3] { tsP[0], tsP[1], tsP[2] };
49             startRotation = new float[4] { tsR[0], tsR[1], tsR[2], tsR[3] };
50         }
51         return startPosition;
52     }
53     ...
54 }
```

Codeausschnitt 5.26: Ausschnitt aus CharacterController.cs

Würde der Fall aus dem Beispiel nun auftreten, dann stünde die Kopie nach dem "Merge" wieder am Hebel und hält diesen der Rekonstruktion aktiv.

5.3.11.2. Abspeichern des Keyframes einer Keyframe Animation

Im zweiten Level des Prototyps muss der Spieler Kräne aktivieren, um über Plattformen, die an den Kränen montiert sind, an das Ziel des Levels zu gelangen. Die Kräne werden nach der Aktivierung, bspw. durch eine Druckplatte, durch eine Keyframe Animation bewegt. Durch die Bewegung des Krans gelangt der Spieler an eine Stelle, an der er wie in "Kapitel 3.10.2.1: Labor, Stromkammer, Element 2:" beschrieben auf die Plattform eines weiteren Krans. Der Kran besitzt dabei eine Klasse in seiner Hierarchie, die von der Basisklasse *Recordables* erbt. Nachdem der Kran aktiviert wurde, fährt der Kran an die Stelle, an der der Spieler auf die Plattform springen kann. Dabei ist es aber so, dass der Spieler während der Kranfahrt einen "Split" ausführt, um mit einem anderen Ich des Chronologen etwas anderes zu machen. Nach

diesem „*Split*“ wird zunächst nichts passieren, da das Ich auf der Druckplatte immer noch darauf steht und der Kran dadurch weiterfährt. Bei diesem „*Split*“ werden die Rotationen und Positionen des Krans auf der Zeitlinie gespeichert. Diese Rotationen und Positionen sind bei einem „*Mergvorgang*“ wichtig, da der Spieler hinterher wieder auf der Kranplattform stehen möchte. Der Kran rotiert sich primäre auf seiner Rotationsachse, dem Gerüst, an dem er verankert wurde.

Führt der Spieler nun einen „*Mergevorgang*“ aus, so würde Folgendes passieren. Das Ich, das auf der Druckplatte steht, wird für einen Frame entfernt und wieder auf der Druckplatte instanziert. Der Spieler würde wieder im vorherigen Ich des Chronologen stecken, an der Position, wo er sich beim „*Split*“ befand. Der Kran hingegen würde nicht an seiner gespeicherten Stelle stehen. Warum macht der Kran das nicht?

Der Prozess des Ladens der Rotation und der Position würde den Kran an die gespeicherte Rotation setzen. Allerdings überschreibt die Keyframe Animation den gespeicherten Wert. Weil das auf der Druckplatte stehende Ich des Chronologen neu instanziert wurde, beginnt die Keyframe Animation wieder am Start der Animation. Die Ausgangslage des Kranes ist allerdings eine andere als im Moment der Speicherung.

```

1 public abstract class Moveable : Recordable
2 {
3     ...
4     protected MoveableAnimationController animationController;
5     ...
6     protected override void OnLoadTimeline(Timeline timeline)
7     {
8         base.OnLoadTimeline(timeline);
9         string animationName = timeline.GetCustomStrings()[saveId + "frame-name"];
10        float keyFrame = timeline.GetCustomFloats()[saveId + "key-frame"];
11        animationController.SetCurrentFrame(keyFrame, animationName);
12    }
13    protected override void OnSaveTimeline(Timeline timeline)
14    {
15        base.OnSaveTimeline(timeline);
16        string animationName = animationController.SaveCurrrentFrame(out float
17        keyFrame);
18        timeline.SetCustomString(saveId + "frame-name", animationName);
19        timeline.SetCustomFloat(saveId + "key-frame", keyFrame);
20    }
21 }
```

Codeausschnitt 5.27: Ausschnitt aus Moveable.cs

Die Kranklasse *Crane.cs*, die im Anhang im Klassendiagramm zu finden ist, erbt von der Klasse *Moveable* welche im Codeausschnitt 5.27 dargestellt ist. Es müssen nun die Methoden *OnLoadTimeline()* und *OnSaveTimeline()* aus der *Recordable* Klasse überschrieben werden, da diese lediglich Positionen und Rotationen abspeichern, wel-

che für Objekte mit einer Keyframe Animation irrelevant sind. Es wird nun mithilfe der *MoveableAnimationControllers* Klasse aus Codeausschnitt 5.28 der Keyframe und die aktive Animation in der Zeitlinie abgespeichert. Jede *Moveable* Klasse hat eine *MoveableAnimationController* Klasse.

```

1 public class MoveableAnimationController : AnimationController
2 {
3     private Animator _animator;
4     ...
5     public string SaveCurrentFrame(out float keyframe)
6     {
7         AnimatorStateInfo currentAnim = _animator.GetCurrentAnimatorStateInfo(0);
8         float normalizedTime = currentAnim.normalizedTime;
9         keyframe = normalizedTime * _animator.GetCurrentAnimatorClipInfo(0)[0].clip.
10        frameRate;
11        string name = _animator.GetCurrentAnimatorClipInfo(0)[0].clip.name;
12        return name;
13    }
14    public void SetCurrentFrame(float keyFrame, string frameName)
15    {
16        AnimatorClipInfo[] clips = _animator.GetCurrentAnimatorClipInfo(0);
17        foreach (AnimatorClipInfo clip in clips)
18        {
19            if(clip.clip.name == frameName)
20            {
21                _animator.Play(clip.clip.name, 0, keyFrame / clip.clip.frameRate);
22            }
23        }
24    }

```

Codeausschnitt 5.28: Ausschnitt aus MoveableAnimationController.cs

Der *MoveableAnimationController* ist für das Handling des Animators und die damit einhergehenden Animationen zuständig.

In der Methode *OnSaveTimeline* wird nun die Methode *SaveCurrentFrame()* aus dem *MoveableAnimationController* aufgerufen, durch welche über den *Animator* die aktuelle Animation und den aktuellen Keyframe herausgefunden und gespeichert werden können.

Diese gespeicherten Informationen werden nach einem “Merge” in der *OnLoadTimeline()* Methode aus der Zeitlinie geladen und an den *MoveableAnimationController* in der Methode *SetCurrentFrame()* übergeben. Diese Methode sucht nun den richtigen Clip, also die richtige Animation, heraus und startet die Animation zum abgespeicherten Keyframe. Dadurch wird der Kran nun an der richtigen Stelle gestartet und der Spieler steht nach einem “Merge” immer noch auf der Kranplattform.

Dieser Fall gilt analog für Türen und ausfahrbare Plattformen, da diese ebenfalls von der Klasse *Moveable* erben.

5.4. Technische Einbindung von Sounds

Das Sounddesign des Prototyps ist durch verschiedene Komponenten aufgebaut. Zum einen gibt es Soundtrigger, die nach Actions von bestimmten Objekten ausgeführt werden. Es gibt unter anderem die Klasse *MoveableSoundTrigger*, die Methoden auf die Actions *StartMovement* und *ResetMovement* der *Moveable* Klasse besitzen (vgl. „Abbildung 111“). Es gibt für jede Art von Soundauslöser einen solchen *SoundTrigger* Komponente. Sie sind im Anhang A.1.3.2 im großen Klassendiagramm zu finden. Die Interaktionen des Spielers lösen ebenfalls Geräusche am Avatar aus, hierbei handelt es sich nicht um *SoundTrigger* sondern um den direkten Aufruf der zweiten Komponente, die im Folgenden erklärt wird (vgl. Codeausschnitt 5.23 Codezeile 52, *_abilitySoundController.PlayAbilitySound()*).

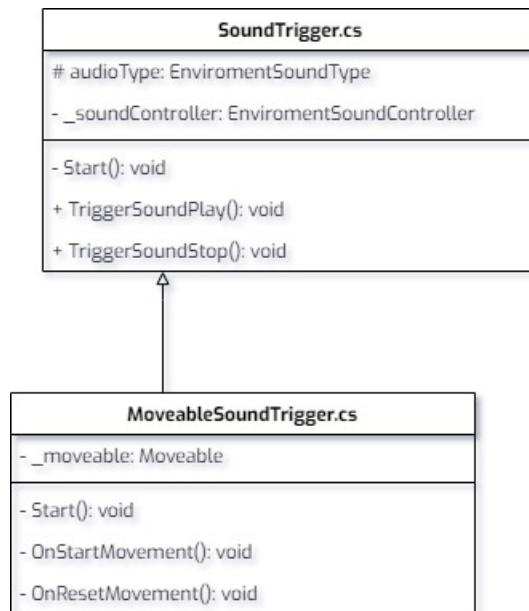


Abbildung 111.: Klassendiagramm einer SoundTrigger.cs Komponente, (Quelle: eigene Darstellung)

Die *SoundTrigger* Komponenten interagieren mit der Klasse *AudioSourceController*. Ein *AudioSourceController* enthält alle Audioquellen, die die entsprechenden Audio-clips abspielen. Über die *PlaySound()* Methoden wird die entsprechende *AudioSource* aus den *AudioSourceControllern* abgespielt. Die *AudioSourceController* sind in verschiedene Sound Unterarten unterteilt, da es unterschiedliche Kategorien von Sounds gibt. Zum einen gibt es ein Hintergrundambiente, welches der Spieler im Level und im Menü hört. Zum anderen gibt es Umgebungsgeräusche, wie die, die *Moveable* Objekte erzeugen. Etwa das Aus- oder Einfahren einer Plattform. In diesem Zusammenhang gibt es zusätzlich *AudioSourceController* für den Sprachdialog des Chronologen und anderen Sprachdialogteilnehmern (vgl. „Abbildung 112“).

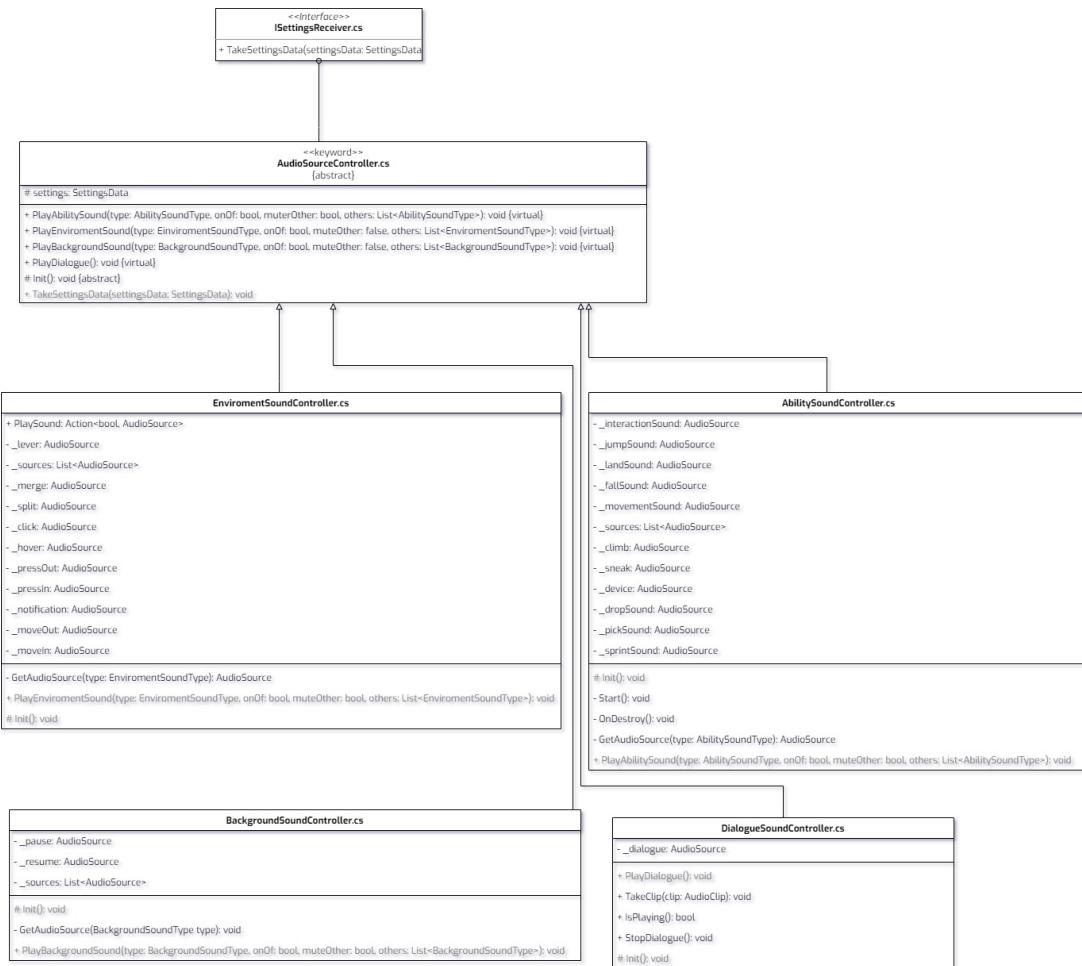


Abbildung 112.: Klassendiagramm der `AudioSourceController.cs` Komponenten,
(Quelle: eigene Darstellung), (In groß im Anhang A 1.3.2)

6. Evaluation

In diesem Kapitel wird der Prototyp und das Spielsystem anhand von ausgeführten Nutzertesten evaluiert. Dabei wird sowohl die definierte Zielsetzung, die Planung und Durchführung und die Auswertung der Ergebnisse vorgestellt.

6.1. Zielsetzung

Durch die abschließenden Nutzertests sollen die Leitfragen dieser Arbeit beantwortet werden. Dabei soll erfasst werden, inwiefern das entstandene Tutorial durch seine Dialoge und Informationen im Spiel die Spielmechanik und die UI des Spiels erklärt. Außerdem wird ein zweites Szenario geprüft, welches im Vergleich zum Tutorial komplexer aufgebaut werden soll.

6.2. Planung und Durchführung

Für die Nutzertesten wurde ein Fragebogen entworfen, welcher im beiliegenden digitalen Anhang zu finden ist.

Dabei geht es zunächst um ein grundlegendes Verständnis zu Zeitreisen und alternativen Realitäten sowie erste Erfahrungen mit Rätselspielen. Der darauffolgende Bereich handelt vom Erlernen der Spielmechanik durch Erklärungen des Dialogs und der im Spiel eingebundenen Informationen sowie dem Verhalten der Spielwelt.

Der zweite große Bereich handelt vom Leveldesign der Spielwelten. Dabei gehen die Fragen auf das in "Kapitel 4.5: Führung durch das Level" behandelte Thema der Gestaltung der Spielwelt ein. HUDs und Partikelsysteme sollen dem Spieler Interaktionsmöglichkeiten anzeigen. In diesem Bereich erfolgt auch das Überprüfen des zweiten Levels, inwiefern es in seiner Gestaltung komplexer ausgearbeitet wurde und ob es nicht zu überfordernd für den Nutzer ist.

Im dritten Bereich kann der Nutzer sein zusätzliches Feedback mitteilen und zusätzliche Verbesserungsvorschläge einbringen. Des Weiteren wird hier das Gefallen und nicht Gefallen sowie die Spielweiterempfehlung des Nutzers festgehalten.

Die Durchführung der Nutzertests erfolgte nach der Methodik des “Concurrent Think Aloud (CTA)”. Dabei sollten die Testpersonen ihre Gedanken zum Prototyp aussprechen und ihr Feedback einbringen (vgl. usability.de GmbH & Co. KG (o. D.)). Die Kenntnisse des “CTA” werden ebenfalls wie der Fragebogen gleichermaßen ausgewertet und im Folgenden “Kapitel 6.3: Auswertung der Tests” vorgestellt.

Insgesamt wurden sechs Nutzertests durchgeführt, bei denen die Testpersonen einen unterschiedlichen Bezug zu Rätselspielen oder Computerspielen im Allgemeinen haben, sowie unterschiedliche Kenntnisstände zu Zeitreisen aufwiesen. Die Zeitlänge der Tests variierte bei jeder Testperson, da die Level in unterschiedlichen Geschwindigkeiten gelöst wurden. Der Zeitraum, in dem die Tests ausgeführt wurden, überstreckte sich auf drei verschiedene Phasen, in denen jeweils die Technik des Prototyps auf einem unterschiedlichen Stand war. Nach jeder Testphase wurde versucht das Feedback einzubauen und den Prototyp weiterzuentwickeln.

6.3. Auswertung der Tests

Durch die Nutzertests konnte gezeigt werden, dass die Spielmechanik und das Anwenden der Spielmechanik durch die eingebundenen Dialoge und Zusatzinformationen verstanden werden konnte. Zusätzlich zeigten sie ebenfalls Schwächen des Prototyps auf, welche in einer Fortentwicklung des Prototyps angepasst werden sollten. Da die Testphasen des Prototyps jeweils zu verschiedenen Entwicklungszeitpunkten des Prototyps durchgeführt wurden, werden die folgenden Erkenntnisse in vier Kategorien unterteilt.

6.3.1. Umgesetztes Feedback

- **Menü des Datenlesers hat im Prototyp kein schönes UI, außerdem fehlt in der Liste der Information eine Markierung für die Neuheit der Information.**
- **Für den Dialog sollte es ein Logbuch geben, damit wichtige Inhalte nachgelesen werden können.**

Das UI des Datenlesers wurde in “Kapitel 4.4.9: Datenleser” in Form von Mockups überarbeitet. Diese Mockups werden in einer ähnlichen Art umgesetzt. Es enthält sowohl eine überarbeitete Liste der Informationen und ihre Darstellung, als auch ein Logbuch der Story.

- **Die eingebundenen HUDs sind in der Form gut, wie sie dargestellt werden.**

- Der Hebel benötigt ein zusätzliches Interface, auf welchem der Spieler sehen kann ob er aktiv ist oder nicht.

In "Kapitel 4.5.3: Tooltips" werden die eingebildeten Tooltips überarbeitet aufgezählt, damit diese einen besseren Nutzen haben werden. In "Kapitel 4.6.2: Hebel" wird das neue Interface eines Hebels dargestellt und beschrieben.

- Das Warten an einem Hebel oder einer Druckplatte ist mühsam und auf Dauer nervig.

Wie in "Kapitel 3.9.3: Imitator" beschrieben erhält der Spieler einen Imitator, wodurch er eine Kopie von sich an eine beliebige Stelle platzieren kann. Dadurch wird zumindest in einem eingeschränkten Rahmen Wartezeiten verkürzt.

- Plattformen waren trotz Inaktivität erklimmbar.

Plattformen, die durch Hebel oder Druckplattformen aktiviert werden können, waren bis zur letzten Testphase noch erklimmbar, obwohl sie in der Wand eingefahren waren. Im Zuge dessen wurden die Kletterkanten bei Inaktivität deaktiviert.

- Das UI für das Ablegen von Objekten benötigt eine andere Darstellung.

Wie im Mockup in "Kapitel 4.4.8: Tragen von Objekten" zu sehen, erhält der Spieler, sobald er ein Objekt trägt, ein neues UI welches in Form des Mockups eingebunden werden soll.

- Der Zeitliniengraph sollte noch Informationen darüber darlegen, wie lange eine Zeitlinie existent war und wie lange sie noch existent sein wird.

Wie in "Kapitel 4.4.1: Graph" beschrieben, erhält der Graph Informationen darüber, wie lange die entsprechenden Zeitlinien aktiv waren und wie lang sie nach einem "Merge" noch aktiv sind.

- Durch den Dialog sollte wie in "Kapitel 3.12.1: Eingeworfene Dialoge Hinweise zu Interaktionen beinhalten als Unterstützung zum sichtbaren Effekt der Interaktion."

6.3.2. Anstehenden Verbesserungen und Optimierungen

Im Folgenden werden die zu optimierenden Aspekte vorgestellt.

- Die Cinemachine-Kameraführung ist bei schnellen Bewegungen ruckelig und Kollisionen mit Wänden führt zu unnatürlichen Kame-

raschwenks, die behoben werden müssen.

- Die Bewegungsanimation des Chronologen reagiert manchmal zu langsam oder führt zu ruckeln in der Kameraführung.
- Das in "Abbildung 52" dargestellte Partikelsystem kann ein anderes Aussehen wie rote Linienmarkierungen erhalten.
- Kanten von denen der Spieler aus springen kann, sollten farblich markiert werden, damit ein farblicher Unterschied zum Boden entsteht.
- Die Schleichfunktion des Chronologen benötigt mehr Integrationen, damit seine Daseinsberechtigung erfüllt wird.
- Objekte, mit denen der Spieler interagieren kann, benötigen ein markanteres Aussehen, damit der Spieler weiß, mit welchen Objekten interagiert werden kann. Bspw. ein Schimmern, das das Objekt umfasst.

6.3.3. Feedback, das im Ausblick mit einem neuen System angepasst werden sollte

Manche Aspekte, die bei den Nutzertests herausgestellt werden konnten, ließen sich nicht auf die kurze Zeit umsetzen oder in das Designdokument integrieren. Diese werden nun in Verbindung auf einen Ausblick hier aufgezählt und im "Kapitel 7.2: Ausblick" erneut aufgegriffen.

- Viele Anmerkungen haben sich auf den Dialog bezogen, welcher eine Überarbeitung benötigt.

Für die zukünftige Integration des Dialoges soll ein vorgefertigtes Dialogsystem genutzt werden, über welches der Dialog erstellt und gesteuert werden soll. Das im Prototyp implementierte System ist für den Zweck nur für einen kleinen Prototypen geeignet und sollte daher abgelöst werden. Allgemein ist der Dialog ziemlich sperrig und zu informativ gehalten. Der Dialog sollte für den Prototyp noch angepasst werden. Selbiges gilt für das Sammeln von Informationen, welche durch eine Anbindung an das Dialogsystem eine ähnliche Funktionalität besitzen sollte, wie es im Prototyp der Fall ist.

- Das Steuerungsprinzip der Steuereinheit benötigt eine Überarbeitung.

Im Prototyp kann der Spieler über die Tasten "A" und "D" das Auswahlrad drehen, um ein Chronologen-Ich auszuwählen. Für die Zukunft wäre es besser, wenn der Spieler auf die Zahnradspitze des gewünschten Ichs klicken kann, welcher dadurch ausgewählt

wird. Der RayCast hat allerdings ein Problem, welches zunächst angepasst werden muss.

- Für den Start in das Spiel besitzt der Spieler eine zu hohe Anzahl an Chronologen zur Auswahl. Außerdem kann der Aufbau des Tutorials verbessert werden.

Der erste Raum des Tutorials sollte umstrukturiert werden. Die Druckplatte, die die Tür im ersten Raum öffnet, sollte für den Spieler sichtbar auf dem Boden platziert werden. Auf dem Weg von der Druckplatte zur Tür sollte das Gerüst platziert werden, damit der Spieler die Wahl hat, durch das Gerüst durchzuschleichen oder über das Gerüst zu klettern. Dadurch erhält der Spieler einen direkteren Einblick in das Verständnis der Rätsel.

6.3.4. Allgemeines Feedback

Zusätzlich zu den bereits thematisierten Aspekte gibt es noch ein allgemeines Feedback zum Prototyp.

Dabei war das zielgerichtete “Trial and Error” Prinzip beim Lösen der Rätsel ein Höhepunkt und erforderte logisches Denken und Geduld.

Ein nicht zuordenbarer Aspekt ist der, dass nicht klar definiert ist, wann der Spieler erhaltene Informationen nachlesen soll. Das Spiel sollte das grundlegende Verständnis der Steuerung aller Objekte mitteilen, ohne dass der Spieler etwas nachlesen muss. Die Informationen sollten daher einen Hintergrund-Rahmen schaffen. Das hat Auswirkungen auf die Gestaltung des UI da darüber die Steueranweisung von Objekten angezeigt werden soll. Diese sind im “Kapitel 4: Visuelles Design des Prototyps” vermerkt.

7. Fazit

In diesem Kapitel werden die Ergebnisse der Arbeit zusammengefasst. Zum Schluss wird ein Ausblick auf die Zukunft des Spiels gegeben.

7.1. Zusammenfassung

In der vorliegenden Arbeit wurde ein verbesserter Prototyp und eine verbesserte Konzeption aus der Grundlage aus dem Gamedesign Workshop 2021/2022 entwickelt.

Dafür wurde zunächst in "Kapitel 2: Grundlagen" eine Analyse von genreähnlichen Spielen sowie Spielen mit einer ähnlichen Spielmechanik durchgeführt, welche in der Spielart Bezüge zu diesem Prototyp aufweisen. Außerdem wurden die Grundlagen der Spielmechanik des Prototyps erklärt.

Im weiteren Schritt wurde das Konzept aus dem Gamedesign Workshop in "Kapitel 3: Konzeption" überarbeitet und in einen passenden Rahmen gesetzt. Das Konzept umfasst nun die wichtigsten Elemente, die das Spiel auf der konzeptionellen Seite haben sollte. Darunter zählen die Hintergrundgeschichte des Spiels, die Welt in der sich der Spieler bewegt, die Möglichkeiten, die ihm geboten werden, um der Geschichte in der Welt zu folgen und die Hilfsmittel, Informationen und Rätsel, die eine solche Geschichte enthält.

In "Kapitel 4: Visuelles Design des Prototyps" wurde eine visuelle Gestaltung der Welt erschaffen, die einen ersten Eindruck des Spiels vermitteln soll. Darunter zählen das Aussehen des Chronologen, erste prototypische Mockups des UIs sowie Weltgegenstände, mit denen der Spieler interagieren kann nun eine Auswirkung auf weitere Weltgegenstände zu haben. Im Rahmen der Spielwelt wurden die konzipierten und umgesetzten Rätsel vorgestellt, welche der Spieler lösen darf. Hinzukommen die bereits konzipierten aber noch nicht umgesetzten Rätsel.

In "Kapitel 5: Umsetzung des Prototyps" wurde die Umsetzung der Spielmechanik vorgestellt. Zunächst wurden verwendete Technologien vorgestellt, welche der Entwicklung des Prototyps zugrunde liegen. Im Anschluss wurden zunächst die technischen Mängel des ursprünglichen Prototyps analysiert und festgehalten, welche in der

Weiterentwicklung des Prototyps verbessert werden mussten. Das überarbeitete System wurde ebenfalls in seinen Kernelementen vorgestellt und in der Art so angefertigt, dass sie für den Ausblick des Spiels generisch erweiterbar sind. Außerdem konnte das System hinreichend deterministisch umgesetzt werden, sodass es nur in bestimmten Sonderfällen zu Ungenauigkeiten in der Kollisionsabfrage kommen kann. Und zwar ist das beim „*Splitten*“ direkt vor Interaktionsobjekten der Fall. Die Kollisionsabfrage des kopierten Spielobjektes des Chronologen ermittelt dabei in seltenen Fällen keine Kollision.

Abgeschlossen wurde diese Arbeit in „Kapitel 6: Evaluation“ mit Nutzertests, welche das grundlegende Prinzip und die Umsetzung des Prototyps auf das Verstehen der Kernmechanik überprüft haben. Die dabei gewonnenen Aspekte wurden teilweise im Kapitel der Konzeption und der Umsetzung eingebbracht. Die restlichen Aspekte werden in der Zukunft der Entwicklung des Spiels einbezogen, um den Prototyp zu verbessern. Durch die Durchführung und Auswertung der Nutzertests konnten ebenso die Forschungsfragen zu einem bestimmten Teil beantwortet werden. Die Komplexität der Rätsel kann in einem ansteigendem und vereinzelt absinkenden Grad weiter entwickelt werden.

7.2. Ausblick

Es ist geplant, das Spielkonzept und den Prototyp weiterzuentwickeln. Daher sollten die durch die Nutzertests erkannten Fehler des Programmverhaltens behoben werden, damit das Spielgefühl, das durch die Fehler verschlechtert wurde, erhöht wird. Zusätzlich müssen weitere Level und neue Szenarien konzipiert und umgesetzt werden, da sie den Hauptteil des Spiels ausmachen. Dazu zählt das Labor, welches der Spieler als „Hub“ benutzt.

Außerdem soll das Text- und Sprachdialog System überarbeitet und durch Zwischensequenzen ergänzt werden, in denen der Spieler die Haupthandlung verfolgen wird. Zudem fehlt das Konzept, welche Gegenstände und Notizen der Spieler finden und welche Rückschlüsse er daraus ziehen kann. Die dabei entstehenden Hintergrundgeschichten sollen nach einer Weiterentwicklung integriert werden.

Im Zuge dessen kann die URP durch die High-Definition Render Pipeline (HDRP) gewechselt werden, um durch das Post-Processing noch mehr grafische Gestaltung zu gewinnen. Um das Entdeckungsgefühl des Spiels und das Aussehen der Levelszenarien auszubauen und feinere grafische Details einzubinden, ist ein Wechsel vom normalen Low-Poly Stil zu einem texturierten Low-Poly Stil eine Möglichkeit. Hierbei können durch Texturen auf den Oberflächen der Spielobjekte, unter anderem an Kan-

ten Details hinzugefügt werden, welche durch den normalen Low-Poly nur aufwendig zu integrieren sind. Ein Beispiel dafür ist das Spiel Ashen (vgl. Gamestar (o. D.)), welches einen ähnlichen Art-Stil besitzt.

Um der nicht deterministischen Spiel-Engine vorzubeugen, sollte in Zukunft das DOTS System in das Spiel eingebunden werden, um dadurch eine bessere Spielperformanz zu bieten. Infolgedessen können bereits existierende Abläufe der Spielmechanik überarbeitet und leistungsstärker umgesetzt werden.

Das zukünftige Ziel wird es sein, dieses Spiel als ein Indie Spiel zu vermarkten und zu verkaufen.

Literaturverzeichnis

A44 (o. D.), 'ashen'. [Online],[zuletzt abgerufen am 22.02.2023].

URL: <https://i.pinimg.com/originals/ec/a8/5e/eca85ea70bcba16144080ef8fbaa56b4.jpg>

Activision Publishing, I. (2021), 'Call of Duty®: Black Ops 2'. [Online],[zuletzt abgerufen am 14.02.2023].

URL: <https://www.callofduty.com/de/blackops2>

Arts, E. (2017), 'A Way Out – Eine offizielle EA-Website'. [Online],[zuletzt abgerufen am 14.02.2023].

URL: <https://www.ea.com/de-de/games/a-way-out>

Arts, E. (2022), 'Entdecke It Takes Two, den preisgekrönten Titel von Hazelight – Electronic Arts.'. [Online],[zuletzt abgerufen am 14.02.2023].

URL: <https://www.ea.com/de-de/games/it-takes-two>

Bronstring, M. (2012), 'What are adventure games?'. [Online],[zuletzt abgerufen am 14.02.2023].

URL: <https://adventuregamers.com/articles/view/17547>

Creations, T. N. (2018), 'Steampunk Low Poly Pack | 3D Environments | Unity Asset Store'. [Online],[letzte Veröffentlichung am 18.07.2017], [zuletzt abgerufen am 16.02.2023].

URL: <https://assetstore.unity.com/packages/3d/environments/steampunk-low-poly-pack-121624>

Creations, T. N. (2021), 'Victorian Low Poly Pack | 3D Historic | Unity Asset Store'. [Online],[letzte Veröffentlichung am 21.01.2019], [zuletzt abgerufen am 16.02.2023].

URL: <https://assetstore.unity.com/packages/3d/environments/historic/victorian-low-poly-pack-136611>

Croteam (2014), 'The Talos Principle | Devolver Digital Games'. [Online],[zuletzt abgerufen am 21.02.2023].

URL: <https://www.devolverdigital.com/games/the-talos-principle>

Daniel, M. (2022), 'Steampunk [Das Lexikon der Filmbegiffe]'. [Online],[letzte Bearbeitung am 09.03.2022], [zuletzt abgerufen am 14.02.2023].

URL: <https://filmlexikon.uni-kiel.de/doku.php/s:steampunk-8272>

Engel, E. (2020), 'Creating a Replay System in Unity'. [Online],[zuletzt abgerufen am 22.02.2023].

URL: <https://www.kodeco.com/7728186-creating-a-replay-system-in-unity>

Entertainment, R. (2016), 'Kaufen Quantum Break | Xbox'. [Online],[zuletzt abgerufen am 14.02.203].

URL: <https://www.xbox.com/de-DE/games/store/quantum-break/BPNDKQN84N6L>

Gama, N. & Mientjes, R. (o. D.), 'Fonts Knowledge'. [Online],[zuletzt abgerufen am 22.02.2023].

URL: <https://fonts.google.com/specimen/Exo?query=exo>

Games, B. (o. D.), '599d33f75cf0308cc12fc5d9870bfffd88debd1d42077e7ffe46975ee54a3af.jpg (1368×770)'. [Online],[zuletzt abgerufen am 22.02.2023].

URL: <https://pitfallplanet.com/resources/s12.png>

Games, K. (2013), 'BioShock Infinite - 2K'. [Online],[zuletzt abgerufen am 14.02.203].

URL: <https://2k.com/en-US/game/bioshock-infinite/>

Games, P. (2018), 'Forza Horizon 4 | Xbox'. [Online],[zuletzt abgerufen am 14.02.2023].

URL: <https://www.xbox.com/de-DE/games/forza-horizon-4>

Games, T. M. (k.D.), '2020_wwhtg_image-003.png (1920×1080)'. [Online],[zuletzt abgerufen am 22.02.2023].

URL: https://totalmayhemgames.com/wp-content/uploads/2020/05/2020_WWHTGImage-003.png

Gamestar (o. D.), 'Ashen Bilder - Screenshots und Galerien'. [Online],[zuletzt abgerufen am 21.02.2023].

URL: <https://www.gamestar.de/spiele/ashen,bilder,1665.html>

Gamestar & Schemes, H. (2016), 'Necropolis - Roguelike im Dark-Souls-Stil wegen Konsolenversion verschoben'. [Online],[zuletzt abgerufen am 22.02.2023].

URL: <https://www.gamestar.de/artikel/necropolis-roguelike-im-dark-souls-stil-wegen-konsolenversion-verschoben,3268589.html>

GameTube (2014), 'The Talos Principle #9 - Und immer, immer wieder geht die Sonne auf.'. [Online],[zuletzt abgerufen am 21.02.2023].

URL: <https://youtu.be/GyYJWlwBuT8?t=84>

GDC (2016), 'The Implementation of Rewind in Braid'. [Online],[zuletzt abgerufen am 14.02.2023].

URL: <https://www.youtube.com/watch?v=8dinUbg2h70>

Glacier, B. (2018), 'Low Poly Rock Models | 3D Environments | Unity Asset Store'. [Online],[letzte Veröffentlichung am 20.06.2020], [zuletzt abgerufen am 16.02.2023].

URL: <https://assetstore.unity.com/packages/3d/environments/low-poly-rock-models-119245>

Gram, M. (2021), 'Official - Notice on DOTS compatibility with Unity 2021.1 - Unity Forum'. [Online],[zuletzt abgerufen am 14.02.2023].

URL: <https://forum.unity.com/threads/notice-on-dots-compatibility-with-unity-2021-1.1091800/>

- Katzlberger, M. (2022), 'Stable Diffusion - Verständlich erklärt'. [Online],[zuletzt abgerufen am 23.02.2023].
URL: <https://katzlberger.ai/2022/10/24/stable-diffusion-verstaendlich-erklaert/>
- Key, F. (o. D.), 'AER Memories of Old bei Steam'. [Online],[zuletzt abgerufen am 22.02.2023].
URL: https://store.steampowered.com/app/331870/AER_Memories_of_Old/
- Lands, D. (2020), 'Fissure: Low Poly Caverns | 3D Landscapes | Unity Asset Store'. [Online],[[letzte Veröffentlichung am 13.10.2020], [zuletzt abgerufen am 16.02.2023]].
URL: <https://assetstore.unity.com/packages/3d/environments/landscapes/fissure-low-poly-caverns-173733>
- Life is Strange Remastered / SQUARE ENIX* (o. D.). [Online],[zuletzt abgerufen am 21.02.2023].
URL: <https://lifeisstrange.square-enix-games.com/en-gb/games/life-is-strange-remastered-collection/>
- LLC, N. D. (2022), 'The Last of Us™ Part I'. [Online],[zuletzt abgerufen am 16.02.2023].
URL: <https://www.playstation.com/de-de/games/the-last-of-us-part-i>
- Mercuzot, S. (2020), 'Low Poly Underground Mega Pack | 3D Urban | Unity Asset Store'. [Online],[[letzte Veröffentlichung am 15.07.2020], [zuletzt abgerufen am 16.02.2023]].
URL: <https://assetstore.unity.com/packages/3d/environments/urban/low-poly-underground-mega-pack-166074>
- Naughty Dog, I. (2021), 'UNCHARTED entdecken'. [Online],[zuletzt abgerufen am 14.02.2023].
URL: <https://www.playstation.com/de-de/uncharted/>
- Plane, C. (o. D.), 'Dribbble - USSR01.png by CG Plane'. [Online],[zuletzt abgerufen am 22.02.2023].
URL: <https://dribbble.com/shots/12441464-UFO-Laboratory/attachments/4054403?mode=media>
- Plays, C. (o. D.), 'operation-tango-v1-694625.jpg (1920×1080)'. [Online],[zuletzt abgerufen am 22.02.2023].
URL: <https://images.everyeye.it/img-screenshot/operation-tango-v1-694625.jpg>
- Project, T. W. . (2021), 'New Feature Spotlight: Replays'. [Online],[zuletzt abgerufen am 14.02.2023].
URL: <https://wz2100.net/news/new-feature-replays/>
- SaferDan (2013), '(400) Pinterest'. [Online],[zuletzt abgerufen am 22.02.2023].
URL: <https://www.pinterest.de/pin/9499849204532344/>
- sathak (2019), 'Sci Fi Lab | 3D model'. [Online],[zuletzt abgerufen am 22.02.2023].
URL: <https://www.cgtrader.com/3d-models/science/laboratory/sci-fi-lab-4bd5bc7b-1ffb-4b83-a0d0-5c3699edfa2e>

Schell, J. (2020), *Die Kunst des Game Design, Bessere Games konzipieren und entwickeln*, 3 edn, mitp.

Schick, J. (2017), 'stylized dungeon game environement , Jason Schick'. [Online],[zuletzt abgerufen am 22.02.2023].

URL: <https://www.artstation.com/artwork/xyPLm>

Sun, R. (2019), 'Game Networking Demystified, Part II: Deterministic'. [Online],[zuletzt abgerufen am 14.02.2023].

URL: <https://ruoyusun.com/2019/03/29/game-networking-2.html>

Sygil-Dev (o. D.), 'Sygil-Dev/sygil-webui: Stable Diffusion web UI'. [Online],[letzte Veröffentlichung am 08.02.2024],[zuletzt abgerufen am 23.02.2023].

URL: <https://github.com/Sygil-Dev/sygil-webui>

Sykoo (2018), 'Asset Review: Sci-Fi City Pack (Low Poly) | Unity'. [Online],[zuletzt abgerufen am 22.02.2023].

URL: <https://www.youtube.com/watch?v=FThdKCte9kMt=21s>

Technologies, U. (2021), 'Unity Particle Pack | Tutorial Projects | Unity Asset Store'. [Online],[letzte Veröffentlichung am 10.06.2021], [zuletzt abgerufen am 16.02.2023].

URL: <https://assetstore.unity.com/packages/essentials/tutorial-projects/unity-particle-pack-127325>

Technologies, U. (2023), 'Unity - Scripting API: MonoBehaviour.FixedUpdate()'. [Online],[zuletzt abgerufen am 17.02.2023].

URL: <https://docs.unity3d.com/ScriptReference/MonoBehaviour.FixedUpdate.html>

Technologies, U. (o. D.a), 'DOTS - Unity's Data-Oriented Technology Stack'. [Online],[zuletzt abgerufen am 14.02.2023].

URL: <https://unity.com/dots>

Technologies, U. (o. D.b), 'Unity - Blender und Maya mit Unity nutzen'. [Online],[zuletzt abgerufen am 14.02.2023].

URL: <https://unity.com/de/how-to/beginner/using-blender-and-maya-unity>

usability.de GmbH & Co. KG (o. D.), 'Concurrent Think Aloud (CTA) im Glossar von usability.de'. [Online],[zuletzt abgerufen am 20.02.2023].

URL: <https://www.usability.de/usability-user-experience/glossar/concurrent-think-aloud.html>

Wagner, C. (2004), 'Developing Your Own Replay System'. [Online],[zuletzt abgerufen am 22.02.2023].

URL: <https://www.gamedeveloper.com/programming/developing-your-own-replay-system>

Eidesstattliche Erklärung

Ich erkläre hiermit an Eides statt, dass ich die vorliegende Thesis selbstständig und ohne unzulässige fremde Hilfe angefertigt habe. Alle verwendeten Quellen und Hilfsmittel sind angegeben. Der Einsatz von KI-Anwendungen ist dem betreffenden Thesisteil, der Art sowie dem Umfang nach detailliert benannt.

Stuttgart, 28. Februar 2023 Nick Philipp Häcker

A. Anhang

A.1. Konzeption

A.1.1. Gamedesign Dokument

Im beiliegenden Datenträger im Verzeichnis 01_Konzeption/GDD

A.1.2. Mockups

Im beiliegenden Datenträger im Verzeichnis 01_Konzeption/Mockups.

A.1.2.1. Icons

Im beiliegenden Datenträger im Verzeichnis 01_Konzeption/Mockups/Icons.

A.1.2.2. Texturen

Im beiliegenden Datenträger im Verzeichnis 01_Konzeption/Mockups/Texturen.

A.1.3. UML Diagramme

A.1.3.1. Konzeptionelle Abläufe

Im beiliegenden Datenträger im Verzeichnis 01_Konzeption/UML/Konzeption.

A.1.3.2. Systemarchitektur und erklärte Abläufe

Im beiliegenden Datenträger im Verzeichnis 01_Konzeption/UML/Umsetzung.

A.2. Entwickelter Prototyp

A.2.1. Alter Prototyp

Im beiliegenden Datenträger im Verzeichnis 02_Prototyp/Alt.

A.2.1.1. Build des alten Prototyps

Erreichbar unter <https://nickhaecker.github.io/A-fraction-of-time/index.html>

A.2.2. Neuer Prototyp

Im beiliegenden Datenträger im Verzeichnis 02_*Prototyp/Neu*.

A.2.2.1. Windows Build

Im beiliegenden Datenträger im Verzeichnis 02_*Prototyp/Build/x64*.

A.2.2.2. MacOS Build

Im beiliegenden Datenträger im Verzeichnis 02_*Prototyp/Build/OS*.

A.2.2.3. Build Hisorie

Erreichbar unter <https://github.com/NickHaecker/a-fraction-of-time-demo-releases>

A.3. Durchgeführte Nutzertests

A.3.1. Leitfaden der Nutzertests

Im beiliegenden Datenträger im Verzeichnis 03_*Nutzertest/Leitfaden*.

A.3.2. Ergebnisse der Nutzertests

Im beiliegenden Datenträger im Verzeichnis 03_*Nutzertest/Ergebnisse*.

A.4. Präsentation

Im beiliegenden Datenträger im Verzeichnis 04_*Praesentation*