Dokumentation

Inhalt

[Ziel und Motivation des Projekts 1](#_Toc493337153)

[Hardware – HTC Vive 1](#_Toc493337154)

[Software - Unreal Engine 4 1](#_Toc493337155)

[RealDIIN 1](#_Toc493337156)

[RealDIIN Raum in VR 1](#_Toc493337157)

[Implementierte Funktionen 1](#_Toc493337158)

[Einteilung in Klassen 1](#_Toc493337159)

[Speichern der Objekt-Positionen 2](#_Toc493337160)

[Vergleichen der Objekt Positionen 2](#_Toc493337161)

[Punkte 2](#_Toc493337162)

# Ziel und Motivation des Projekts

# Hardware – HTC Vive

# Software - Unreal Engine 4

# RealDIIN

Es handelt sich um eine Art Gedächtnisspiel, in dem die Probanden in einen Raum geführt werden und sich innerhalb von 15 Sekunden die Position von fünf bis zehn sogenannten „Targets“ einprägen soll. Bei diesen Targets handelt es sich um verschiedenste Bürogegenstände wie etwa einer Schere, eines Tackers etc. Nach diesen 15 Sekunden sollen sie die Augen schließen und werden wieder in den Flur gebracht. Der Spielleiter verändert die Positionen einiger Targets, indem er ein paar an einen anderen Platz stellt oder von anderen auch nur ihre Rotation verändert. Der Proband wird nun wieder in den Raum zurückgeführt und soll diese Targets wieder in ihre ursprüngliche Position zurückbringen. Der Spielleiter notiert die Aktionen des Probanden. Dabei gibt es für das Erkennen der Ortsveränderung eines jeden Targets einen Punkt, unabhängig davon, ob das Target nun an den richtigen Ort zurückgelegt wurde oder nicht. Wenn der Proband der Meinung ist, das Level abgeschlossen zu haben, soll er im Flur oder in einem anderen Raum einer anderen Aufgabe nachgehen. Währenddessen bereitet der Spielleiter, nach einer Vorlage, das nächste Level vor.

## RealDIIN Level-Ablauf in VR

Anhand des soeben beschriebenen Spielablaufs ist es unschwer zu erkennen, dass das in einem realen Raum mit realen Gegenständen durchgeführte Gedächtnisspiel einige Nachteile mit sich bringt:

Zwischen allen Levels muss der Spielleiter den Raum immer wieder umräumen: Die vorigen Targets müssen weggeräumt werden und die neuen Targets mit grüner oder roter Unterlage nach einer bestimmten Vorlage wieder aufgebaut werden. Auch wenn der Proband das Level nach den ersten 15 Sekunden verlässt, muss der Spielleiter einzelne Targets verschieben, und zwar in einer möglichst kurzen Zeit, damit der Proband nicht anfängt zu vergessen und somit das Spiel verfälscht wird. Auch bei der Punktevergabe spielt der Spielleiter eine unverzichtbare Rolle. Er notiert alles, was wichtig sein könnte.

Ein weiterer Nachteil, der aus dem letzten Absatz schon deutlich geworden sein dürfte, ist der relativ große Zeitaufwand durch das ständige Umbauen.

In beiden Fällen kann VR aushelfen. VR kann das „Umräumen“ des Raumes und auch die Punktevergabe übernehmen. Die Anwesenheit einer weiteren Person wäre zwar immer noch von Nöten, etwa damit der Proband nicht über das Kabel stolpert oder um sicherzugehen, dass er sich in der VR-Umgebung wohlfühlt. Allerdings fungiert diese ab sofort eher als Betreuer als als Spielleiter.

In der Umsetzung in VR besteht ein Level je aus zwei Teillevels: Für Level1 etwa gibt es das Unreal-Level „Part1Level1“ und das Unreal-Level „Part2Level1“. Jedes dieser Teillevel werden in Unreal durch ein Unreal-Level realisiert.

In Part1Level1 wird dem Probanden 15 Sekunden Zeit gegeben, sich die Gegenstände zu merken. In Part2Level1 haben sich die Positionen einiger Targets dann verändert und der Proband soll sie wieder an die richtige Position zurückstellen. Unterbrochen werden die beiden Unreal-Teillevels durch einen kurzen Aufenthalt des Probanden im Level „Floor“.

Nach Beendigung des gesamten Levels, also beider Unreal-Levels, wird automatisch ein weiteres Level aufgerufen. Das Beenden des Levels und das starten des neuen Levels kann der Proband jeweils bestätigen.

## RealDIIN Raum in VR

- Raum 307 und die Möbel darin grob ausmessen, um sie in Unreal realisieren zu können

- fbx Models aus dem Internet laden:

- importieren über Import ->

# Implementierte Funktionen

## Objekte

Alle Objekte, von denen pro Level ein paar Objekte herausgesucht werden, die sich der Proband dann merken soll, werden im folgenden *Targets* genannt, um sie eindeutig von den anderen sich im Raum befindlichen Gegenständen, wie etwa den Tischen und Regalen oder auch den Distraktoren, zu unterscheiden. Distraktoren und Targets bilden zusammen die Gruppe der *Objects of Interest*, kurz *OoI*.

*OoI* und auch Möbel und andere Gegenstände können aus dem Internet heruntergeladen oder eigenhändig über Blender erstellt werden.

### Internet

Die meisten meiner heruntergeladenen Objekte habe ich von der Website CGTrader.com oder der von Google zur Verfügung gestellten Seite 3D-Warehouse.Sketchup.com heruntergeladen. Hier sind viele kostenpflichtige, aber auch viele kostenlose Objekte verfügbar. Wichtig ist, dass das Format „Autodesk fbx (.fbx)“ entspricht.

### Blender

Über Blender habe ich einfachere Objekte selbst erstellen können. Diese können dann als fbx-file exportiert werden.

### Import

Die fbx-files können in Unreal importiert und so im Spiel verwendet werden, indem man im Content Browser auf „Import“ klickt:

Das importierte Objekt liegt nun als einfaches Static Mesh vor. Ein Static Mesh an sich kann bereits in das Level integriert werden, es lässt sich etwa einfach über den Viewport im Level platzieren. Ein Static Mesh kann aber nicht mit dem Probanden oder mit anderen Gegenständen interagieren, also nicht aufnehmen, wegwerfen,… Möbel, die bloß im Raum stehen und keine weitere Funktion erfüllen müssen, können also einfach als Static Mesh beibehalten werden. Für alles andere bedarf es sogenannte Blueprints.

## Blueprints

Blueprints ermöglichen es, mit einem Objekt interagieren zu können, es etwa hochheben, wegwerfen oder neu platzieren zu können. Dies geschieht, indem man im Blueprint entsprechende Funktionen implementiert. Auch wenn zum Beispiel ein Objekt erst nach einer bestimmten Zeit erscheinen soll, wenn sich eine Tür bei Betreten eines bestimmten Bereiches öffnen soll oder wenn ein Schuss bei Betätigung eines bestimmten Knopfes abgefeuert werden soll, bedarf es Blueprints.

Ein Blueprint ist also quasi eine Ansammlung von implementierten Funktionen. Es bildet für das jeweilige Objekt eine Klasse. Jedes interagierende Objekt hat seine eigenes Blueprint, seine eigene Klasse, in die nur das Objekt selbst eingeteilt ist, etwa tacker\_BP oder alarm\_BP. Jedes Blueprint wiederum kann dann aber noch in weitere, höhere Klassen eingeteilt werden. So lassen sich alle Targets in die Klasse Target Class einteilen und alle Distraktoren in die Klasse Distractor Class. Wenn man wollte, könnte man durch Implementieren von Funktionen in den Oberklassen den Targets andere Eigenschaften geben als den Distraktoren.

Angenommen, es soll für das soeben importierte Static Mesh „tacker“, ein Tacker, ein Blueprint erstellt werden und in die Target Class eingeteilt werden.

Zunächst muss für den Tacker ein Blueprint erstellt werden. Dies geschieht, indem man beim markierten Tacker rechts auf „Add Script/Blueprint“ klickt. Dem neuen Blueprint muss ein Name gegeben werden (etwa „tacker\_BP“) und der Speicherort, optimalerweise im Ordner „Blueprints“, in dem sich alle Blueprints befinden, muss gewählt werden. Nun öffnet sich ein neues Fenster oder ein neuer Tab. Dies ist der erstellte Tacker-Blueprint.

Das Static Mesh tacker besitzt jetzt also eine Klasse, „tacker\_BP“.

Für OoI oder Distraktoren wird anschließend als Parent Class unter „Search Class“ die Oberklasse „Class“ ausgewählt. In dieser sind alle OoI und Distraktoren gesammelt.

Für einen Überblick über alle Klassen und die Einteilungen dient folgende Grafik:

Abbildung : Systematische Darstellung der Hierarchie

## Einteilung in Oberklassen

Um ein erstelltes Objekt und sein Blueprint in seine jeweilige Oberklasse einzuteilen, wählt man im Blueprint unter Class Settings die gewünschte Klasse (Target Class oder Distractor Class) aus. Meist ist es so, dass ein Objekt je nach Level ein Target oder auch ein Distraktor sein kann. Aus diesem Grund kann man das Blueprint Actor duplizieren und je dem einen die Target Class zuweisen und dem anderen die Distractor Class.

## Tags

Alle Targets, also gleichzeitig alle Objekte mit Blueprints, werden mit Tags versehen, um zuzuweisen, in welchen Levels sie vorkommen. Im jeweiligen Level werden dann nur jene Objekte auf Positionsveränderung kontrolliert, die mit dem entsprechenden Tag (Level1, Level2, Level3,…) versehen sind. Tags können im Details Tab unter „Tab“ bearbeitet werden.

## Speicherung

Der gesamte Vorgang der Ortsveränderungsregistrierung läuft im Groben folgendermaßen ab:

1. Abspeichern der Ausgangspositionen der Targets im ersten Teillevel im SaveGame „Original Positions“
2. Vergleich der im zweiten Teillevel teilweise verränderten Positionen der Targets mit den im SaveGame „Original Positions“ gespeicherten Ausgangspositionen. Abspeichern der neuen Positionen im SaveGame „NewLocationSave“
3. Punktevergabe durch Vergleich der veränderten Positionen aus dem SaveGame „NewLocationSave“ mit den aktuellen Positionen der Targets, die sich durch den Nutzer evtl. verändert haben (entweder zurecht oder zu Unrecht). Abspeichern der Punkte im SaveGame „Result“

## Speichern der ursprünglichen Positionen

Der unter 1. grob beschriebene Verlauf lässt sich im sogenannten Level-Bleuprint eines jeden 1. Teillevels einsehen. Hierzu öffne man zunächst ein beliebiges 1. Teillevel und klicke über dem Viewport auf „Blueprint“ und dann auf „Open Level Blueprint“.

Das Level Blueprint bezieht sich nur auf dieses eine Level, kann also auch nur im jeweiligen Level ausgeführt werden. Der Code ist folgender:

Bei Beginn des Levels wird ein Timer, ein Delay, ausgelöst, der 15 Sekunden andauert. Dieses sind die 15 Sekunden, die der Proband Zeit hat, sich die sich im Raum befindenden Targets einzuprägen. Bevor nun nach Ablauf dieser Zeit der Proband in den Flur versetzt wird, beginnt ein Speicherprozess:

Abgespeichert werden:

- die Namen aller Targets

- die Transformation, also der Ort und die Rotation, aller Targets

- in welchem Level wir uns aktuell befinden (wird manuell eingestellt, z.B. Level3)

- wie das nächste Teillevel also heißt (wird manuell eingestellt, z.B. Part2Level3)

Durch den abschließenden Befehl „Save Game to Slot“ werden alle oben genannten Informationen im Slot „Original Positions“ abgespeichert. Das SaveGame findet man im Explorer im Ordner „Saved“. Es ist nicht lesbar. Wie schon erwähnt wird nach erfolgreicher Speicherung das Level „Floor“ geöffnet.

## Automatischer Vergleich der Positionen aus Teillevel 2 mit Teillevel 1

Den Vergleich der Positionen der Targets im Teillevel 2 mit denen aus Teillevel 1 kann man wieder im Level Blueprint des Teillevels 2 einsehen.

Im Teillevel 2 haben sich die Positionen der Targets teilweise verändert. Wenn der Proband in den Raum zurückkehrt, Teillevel 2 also geöffnet wird, wird im Hintergrund direkt wieder ein SaveGame erstellt

Abgespeichert werden:

|  |  |  |  |
| --- | --- | --- | --- |
| Array, das abgespeichert wird | Funktion | Typ | Beispiel |
| ChangedObjectNameArraySave | Die Namen aller Targets (entspricht „Object Name Array, aber evtl andere Reihenfolge der Namen“ | Name | [alarm\_BP, notebook\_BP,…] |
| HasPositionChangedArraySave | ob sich das jeweilige Objekt beim Übergang Teillevel1 -> Teillevel2 verändert hat (entspricht Reihenfolge wie „ChangedObjectNameArraySave“) | Boolean (1 oder 0 bzw. Ja oder Nein) | [1,0,…] |
| ChangedTransformArraySave | Listet die neuen Positionen der Targets, unabhängig davon, ob eine Veränderung stattgefunden hat (entspricht Reihenfolge wie ChangedObjectNameArraySave“) | Transform  (x,y,z, alpha, beta, gamma, scale) | [(100, 300, 200, 0°, 0°, 90°, 1.0, 1.0, 1.0), 120, 140, 200, 90°, 0°, 0°),…] |
| CurrentLevel | AktuellesLevel | Name | Level3 |
| NextLevel | Nächstes Teillevel | Name | Part1Level6 |
| Possible Score | Erreichbare Punktzahl abhängig von der Anzahl der veränderten Targets | Integer | 4 |

Durch den abschließenden Befehl „Save Game to Slot“ werden alle oben genannten Informationen im Slot „NewLocationsSave“ abgespeichert. Das SaveGame findet man im Explorer im Ordner „Saved“. Es ist nicht lesbar. Der Durchlauf des Level Blueprints ist damit beendet.

## Punktespeicherung nach Bestätigung durch Probanden

Der Proband hat die Aufgabe, die Veränderungen zu erkennen und die veränderten Targets wieder an ihren ursprünglichen Ort zu stellen.

Ist er der Meinung, diese Aufgabe erfüllt zu haben, drückt er auf den roten Schalter rechts neben der Tür.

Dieser Schalter ist von einer sog. TriggerBox umgeben. Das heißt, dass sobald ein Actor wie etwa der Proband diese Triggerbox betritt, sie „überlappt“, ein bestimmtes Ereignis ausgelöst wird. Dieses Ereignis kann wieder im Level Blueprint eingesehen werden. Es gibt dort ein Event „On Actor begin Overlap“, was eben genau dann ausgelöst wird. Es bewirkt, dass sich ein Widget öffnet, mit dem man das Beenden des Levels und somit auch das Berechnen der erzielten Punkte auslösen kann. Für mehr Informationen in Bezug auf die Widgets bitte im Kapitel Widgets nachschauen.

Bei Bestätigung der Frage wird wiederum ein Ereignis getriggert, welches sich „Results“ nennt. Das Ereignis kann man im MotionControllerPawn\_6 einsehen. Dieser Pawn ist quasi der Spieler des jeweiligen Levels. Weil der Spieler selbst das Ereignis ausgelöst hat und es sich nicht um ein automatisches Event handelt, befindet sich das Event im diesem Pawn und nicht im Level Blueprint wie die Speichervorgänge zuvor.

Für jeden Fall , dass der Proband eine Ortsveränderung eines Targets erkannt hat, gibt es einen Punkt.

Abgespeichert werden:

|  |  |  |  |
| --- | --- | --- | --- |
|  | Funktion | Typ | Beispiel |
| Punkte | Speichert die Anzahl der erreichten Punkte | Integer | 5 |
| Level | Speichert aktuelle Level, in dem also diese Punktzahl erreicht wurde | Name | Level3 |

Ist er der Meinung, die Aufgabe erfüllt zu haben, geht er durch die Tür und bestätigt seine Eingaben. Bei der Bestätigung wird ein weiteres SaveGame gespeichert, das sich „Results“ nennt. Es werden wieder die (evtl. durch den Probanden veränderten) Positionen aller OoI mit dem jeweiligen Tab durchgegangen und jeweils mit denen aus dem SaveGame „ChangedPositions“ verglichen. Durch kurzes Nachdenken akzeptiert man schnell: Stimmen bei einem Objekt, bei dem im Array „HasPositionChanged“ eine 1, also ein ja, abgespeichert ist, die Position im SaveGame „ChangedPositions“ mit der aktuellen Position überein, so hat der Proband die Ortsveränderung des Objektes nicht erkannt bzw. er hat sie zumindest nicht verändert. Stimmen bei einem Objekt, das sich verändert hat, die Positionen früher und jetzt nicht überein, so hat der Proband die Veränderung erkannt und das Objekt woanders hingestellt. Dabei ist es in dieser Version der Durchführung unwichtig, ob der Proband das Objekt an die richtige Stelle zurückgestellt hat, es zählt nur, dass er es verschoben hat. Für jede Erkennung gibt es einen Punkt. Die Punkte werden im SaveGame „Results“ gespeichert.