
Abschlussprüfung Winter 2022/23

Fachinformatiker für Anwendungsentwicklung
Dokumentation zur betrieblichen Projektarbeit

Asset placer

Programmerweiterung zum platzieren von 3D Objekten

Durchführungszeitraum: 04.10.2022-10.10.2022

Prüfungsbewerber:

Durmus Kaan Özkürkcü
Benzweg 19
30165 Hannover



Praktikumsbetrieb:

VISIONME GmbH
Bödekerstraße 69
D-30161 Hannover

INHALTSVERZEICHNIS

GLOSSAR	IV
BILDVERZEICHNIS	V
TABELLENVERZEICHNIS	VI
QUELLENVERZEICHNIS	VII
ABKÜRZUNGSVERZEICHNIS	VIII
1. EINLEITUNG	9
1.1. ALLGEMEINE ANGABEN.....	9
2. ZENSURREGELUNG	9
3. PRAKTIKUMSBETRIEB	9
3.1. KUNDENBEZIEHUNG	9
3.1.1. Hausbearbeitung	9
3.1.2. Wartungsarbeit	9
3.1.3. Prozesserweiterung	9
4. DIE APP „IMMOBILIEN DESIGNER“	10
5. PROJEKTARBEIT ERLÄUTERUNG	10
5.1. DAS PROGRAMMIERWERKZEUG „ASSET PLACER“	10
5.2. PROJEKTHerausforderung.....	10
5.3. PROJEKTZIEL.....	11
5.4. PROJEKTUMFELD.....	11
6. GEWINNANALYSE	11
6.1. KOSTENAPPROXIMATION	11
6.1.1. Beispiel Rechnung der gesparten Arbeitszeit	11
7. PROJEKTPHASIERUNG IN STUNDEN	12
8. DEFINITIONSPHASE	13
8.1. IST/SOLL ANALYSE	13
8.1.1. Ist Zustand	13
8.1.2. Soll-Zustand	13
9. KONZEPTIONSPHASE	14
9.1. ENTWURF DER BENUTZEROBERFLÄCHE (ONGUI FUNKTION)	14
10. DATENANALYSE DES ASSETS	14
10.1. GARTEN	14
10.2. TREPPE	16
11. ASSET PLACER	16
12. SOFTWARESPEZIFISCHE PROGRAMMIERKENNTNISERWEITERUNG FÜR DEN ASSET PLACER	16
13. TECHNISCHE DETAILS	17
13.1. ALLGEMEINE 3D GRUNDLAGEN ERLÄUTERUNG.....	17
14. ABBILDUNG POLYGONE	18

15. BENÖTIGTE SOFTWARE	18
15.1. VISUAL STUDIO CODE	18
15.2. ENTWICKLUNGSUMGEBUNG: UNITY 3D	18
15.2.1. Übersicht der Unity-Benutzeroberfläche	19
16. HAUSINTERNE ERWEITERUNG: JOB AUTOMATION	20
16.1. ERLÄUTERUNG.....	20
17. TEAM-INTERNE KOMMUNIKATION	22
17.1. WERKZEUGE	22
17.2. KONZEPTIONIERUNG VON MÖGLICHEN ERWEITERUNGEN (TECHNISCHE DETAILS DER PROGRAMMIERUNG).....	22
17.3. MÖGLICHE FUNKTIONEN DIE FÜR DIE ERWEITERUNG BENÖTIGT WERDEN.	23
17.3.1. Singleton Methode	23
17.3.2. GUI- Layout	24
17.3.3. Clusterobjekte	24
17.3.4. GUI-Button	24
17.3.5. Select Object	25
17.3.6. Custom Editor	25
17.3.7. Override OnInspectorGUI	25
17.4. BOUNDING BOX - FUNKTION.....	27
17.5. GETCOMPONENTISINCHIDREN -METHODE UND ENCAPSULATE	28
18. WEITERE FUNKTIONEN UND VOLLSTÄNDIGER CODE	29
19. IMPLEMENTIERUNGSPHASE	34
19.1. IMPLEMENTIERUNG DER USER INTERFACE ERWEITERUNG.....	34
19.2. IMPLEMENTIERUNG DER PROGRAMMIERUNG	34
20. TESTPHASE	34
20.1. UI TESTING	34
20.2. ASSET PLATZIERUNG.....	34
21. KONTROLLPHASE (ZWISCHEN IST- UND SOLLZUSTAND)	35
22. ABSCHLUSSPHASE	35
23. DOKUMENTATIONSPHASE	35
24. FAZIT	36
25. SELBSTSTÄNDIGKEITSERKLÄRUNG	36

Glossar

Begriff	Erklärung
3D Engine	Mit 3D Engine ist im Allgemeinen eine Grafikengine gemeint. Die Grafik wird innerhalb eines Programms dargestellt.
Vertices	Eckpunkte.
Edges	Linien zwischen den Vertices.
Polygon	Vieleck.
Euklidischer Raum	Die reelle 3-dimensionale Wahrnehmung des Menschen über den Raum.
Materials	Auf Objekte liegende Materialeigenschaften, z.B. Lichtberechnungen nach Reflektivität, Spiegelung, Durchsichtigkeit, etc.
Texturen	Bilder, die auf dem Objekt liegen.
Scripte	Objektgebundene Programme, die das Verhalten des Objektes bestimmen.
Animation	Objektgebundene Bewegungsangaben, die Rigging basierend oder Vektoren basierend Objekte bewegen können.
Mesh	3D Objekt bestehend aus Polygonen. Synonym mit Objekt.
Rigg/Rigging	Eine Art Skelett, nachdem sich das Objekt bewegt.
Assets/Prefab	Alle Daten, die im Projekt vorkommen als Fertigbau.
Tab	User Interface Element (Fenster).
Local space	Der Schwerpunkt des Objektes relativ zu einem anderen Objekt.
Global Space	Der Koordinatenursprung.
World space	Synonym mit Global Space.
UI	Userinterface.
Minimal-Prinzip	Einen bestimmten Erfolg mit dem geringstmöglichen Aufwand erreichen.
Maximal-Prinzip	mit gegebenen Mitteln maximalen Erfolg erzielen.
Raycast	Ein Raycast ist eine Berechnung zwischen zwei vorgegeben Punkten, um mit den Informationen weitere physikalische Berechnungen in der Engine vorzunehmen, dabei werden Punkte wie origin (Ursprung), direction (Richtung) und maxDistance (maximale Ausbreitungspunkt) benutzt.

Bildverzeichnis

ABBILDUNG 1: IMMOBILIEN DESIGNER APPLIKATION	10
ABBILDUNG 2: TOP-DOWN ANSICHT HAUS, UNGERENDERT	15
ABBILDUNG 3: HAUS MIT ANBAU	15
ABBILDUNG 4: TREPPENARTEN, UNGERENDERT	16
ABBILDUNG 23: KOORDINATENSYSTEM XYZ	17
ABBILDUNG 24: (LINKES BILD) ABBILDUNG OHNE FLÄCHEN (RECHTES BILD) ABBILDUNG MIT FLÄCHE	18
ABBILDUNG 25: BENUTZEROBERFLÄCHE DER UNITY ENGINE	20
ABBILDUNG 26: JOB AUTOMATION	21
ABBILDUNG 5: GUI-ELEMENT, ASSET PLACER	23
ABBILDUNG 6: SINGLETON CLASS BEISPIEL	23
ABBILDUNG 7: GUI-LAYOUT GARTEN	24
ABBILDUNG 8: GUI-BUTTON LAYOUT	25
ABBILDUNG 9 : CUSTOM-EDITOR	25
ABBILDUNG 10: OVERRIDE METHODE	25
ABBILDUNG 11: PREFAB GUI IM INSPEKTOR	26
ABBILDUNG 12 : PREFAB-AUSWAHL IM EDITORMODE ALS CODE	26
ABBILDUNG 13 : GUI-LAYOUT.BUTTON MOVE METHODE	27
ABBILDUNG 14: BOUNDING BOX PARAMETER	27
ABBILDUNG 15: BOUNDING BOX PARAMETER	28
ABBILDUNG 16: BOUNDING BOX METHODE	28
ABBILDUNG 17: GETCOMPONENTSINCHIDREN	29
ABBILDUNG 18: VOLLSTÄNDIGER CODE 1	30
ABBILDUNG 19: VOLLSTÄNDIGER CODE 2	31
ABBILDUNG 20: VOLLSTÄNDIGER CODE 3	32
ABBILDUNG 21: VOLLSTÄNDIGER CODE 4	32
ABBILDUNG 22: VOLLSTÄNDIGER CODE 5	33



Tabellenverzeichnis

TABELLE 1:ZEITERSPARNIS PRO TAG.....	11
TABELLE 2: ZEITERSPARNIS PRO MONAT.....	12
TABELLE 3 : PROJEKTPHASIERUNG	12
TABELLE 5: TEAM-INTERNE KOMMUNIKATION	22
TABELLE 6 : BENÖTIGTE SOFTWARE	22
TABELLE 4 : GUI-BUTTON	25

Quellenverzeichnis

(1) 3D/Grafik-Engine: <https://de.wikipedia.org/wiki/Grafik-Engine>

(2) Vertices: <https://de.wikipedia.org/wiki/Vertex>

(3) Edges: [Ecke – Wikipedia](#)

(4) Polygon: - <https://de.wikipedia.org/wiki/Polygon>

(5) Raycast: <https://de.wikipedia.org/wiki/Raycasting>

<https://visionme.de/>

Abkürzungsverzeichnis

3D	3 – dimensional
IDE	Integrierte Entwicklungsumgebung
VR	Virtuelle Realität
U. I	User Interface
GUI	Graphical User Interface
CMS	Content-Management-System

1. Einleitung

1.1. Allgemeine Angaben

Der Autor hat Im Rahmen des IHK – Abschlussprojektes zur Weiterbildung als Fachinformatiker für Anwendungsentwicklung den Ablauf dieser Projektdokumentation durchgeführt.

2. Zensurregelung

Alle Bilder und Beschreibungen, die zensiert wurden sind, dienen dem betrieblichen und kundenbezogenen Datenschutz.

3. Praktikumsbetrieb

3.1. Kundenbeziehung

Die Praktikumsfirma Visionme GmbH handelt mit den Kunden*innen (Immobilienfirmen) Aufträge aus. Bei einer Einigung der Aufträge werden Häuser (3D – Assets) im Rohzustand von den Kunden*innen (Immobilienfirma) and die Firma Visionme GmbH zugesendet. Hierbei wird die Plattform Strapi als gemeinsame Schnittstelle genutzt. Das Haus wird fertig erstellt und gerendert zurückgesendet. Insgesamt geht es um fertig gerenderte und begehbare Häuser, die je nach Auftrag innerhalb einer Frist bearbeitet werden.

3.1.1. Hausbearbeitung

Die Daten werden von der Firma Visionme inspiziert und auf Fehlerquellen überprüft. Falls das Haus Fehler aufweist, wird es an die Kunden*innen zurückgesendet. Die Häuser werden dann über die hauseigene Erweiterung „Job Automation“ automatisiert überarbeitet. In der Regel muss das Haus noch einmal händisch überarbeitet werden, damit es dann fertig gestellt werden kann. Die Fertigstellung ist hierbei die vollständig ausgerenderte Abbildung der Häuser inklusive automatisierter Möblierung und Garten. Nachdem alle Prozesse der Fertigstellung durch sind, wird das Haus auf die Webdatenbank Strapi hochgeladen und den Kunden*innen per E-Mail benachrichtigt. Von der Datenbank aus kann der Kunde nun die Datei einsehen. Hierzu bekommt den Kunden*innen eine ID und einen 5-stelligen Code, mit dem der Kunde*in über die hausinterne-App „Immobilien Designer“ das Haus online besichtigen kann.

3.1.2. Wartungsarbeit

Bei der Automatisierung kann es passieren, dass gewisse Prozesse beim Durchlaufen nicht einwandfrei funktionieren. Die Programmierung muss dann der Fehlerquelle entsprechend umgeschrieben werden.

3.1.3. Prozesserweiterung

Um den stetig wachsenden Anforderungen des Kunden*innen gerecht zu werden, entstehen individuell dem Kundenwunsch entsprechende Lösungen. Hierzu werden Prozesse für dem jeweiligen Kunden*innen entwickelt, die mit ihrem eigenen Assets-Katalog und Gärten kommen.

4. Die App „Immobilien Designer“

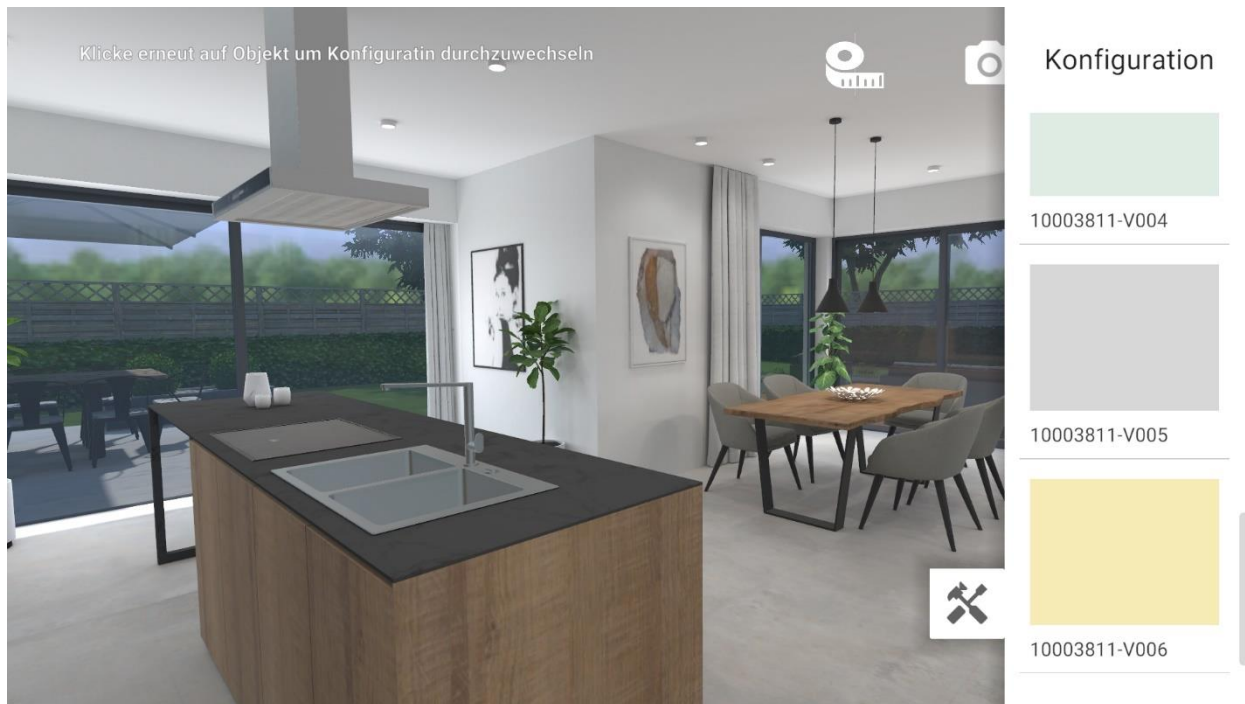


Abbildung 1: Immobilien Designer Applikation

Die Firma Visionme GmbH stellt mit dem „Immobilien Designer“ eine Applikation zur Verfügung, mit dem der Kunde*innen mit dem Smartphone oder Tablet Immobilien virtuell begehen, als auch konfigurieren kann. Man kann mit dem gegebenen Katalog Möbel platzieren, die Farben der Wände ändern und vieles mehr, um sich ein individuelles Haus zu konfigurieren. Dazu klickt man auf ein Konfigurierbares Objekt (z.B. die Außenwand) und stellt die Farbe um.

PROJEKTARBEIT

5. Projektarbeit Erläuterung

5.1. Das Programmierwerkzeug „Asset Placer“

Im Rahmen der gegebenen Projektzeit hat der Autor sich mit den möglichen Programmerweiterungen beschäftigt. Die Hauseigene Programmerweiterung „Job Automation“ wurde als Grundlage genommen, um einen erweiterten Prozess in Form eines Customer-[GUI](#) Tools zum Platzieren von Assets zu programmieren.

5.2. Projektherausforderung

Die Herausforderung bei dem Automatisierungsvorgang besteht darin, dass gewisse 3D Objekte eine komplexe Geometrie besitzen und diese somit nicht immer fortlaufend den individuellen Ziel-Objekten entsprechend passen. Die fehlenden Prozesse werden entweder durch weitere Methoden programmiert oder händisch von den 3D Artists ausgebessert. Die Frage nach der Fertigstellung der Aufgabe hängt stark vom Aufwand ab. Die Programmierer*innen haben hierbei die Herausforderung sich sowohl den anfallenden Wartungsarbeiten zu widmen, als auch den Prozess der Automatisierung voranzutreiben. Der Auftraggeber (Kunden*innen) gibt für die Bearbeitung der Häuser eine gewisse Deadline und dementsprechend werden die Wartungsarbeiten bevorzugt.

5.3. Projektziel

Die Projektarbeit dient der möglichen Unterstützung des Automatisierungsprozesses, in dem es ein Werkzeug zur schnellen und genauen Platzierung der Treppe anbietet.

5.4. Projektumfeld

Die Firma Visionme GmbH stellt insgesamt zwei Abteilungen für das Projekt bereit. Primär wurde in der Entwicklungsabteilung gearbeitet, des Weiteren wurde die 3D Abteilung bereitgestellt. Die jeweiligen Abteilungen sind mit leistungsstarken PCs und herkömmlichen Büromöbel gleichermaßen ausgestattet. Das Projekt wurde im Rahmen der Projekterstellung in der Entwicklungsabteilung erstellt (Hauptsächlich für die Programmierer*innen).

6. Gewinnanalyse

Das Ziel der Firma ist es mit den gegebenen Mitteln des Automatisierungsprozesses den Zeitaufwand so weit zu minimieren, dass man durch die gewonnene Zeit mehr Häuser bearbeiten kann, so dass man dadurch den Profit gemäß des minimal und des maximalen Prinzips erhöht. Für die genauen Kostenberechnung ist die Businessabteilung der Firma Visionme GmbH zuständig und die damit verbundene Gewinn/Verlust Berechnung. Es ist nach den gegebenen Rahmen für den Autoren nicht genau berechenbar, wie sich die jeweilige Zeitgewinnung den Gewinn erhöht. Jegliche Kalkulationen können nur approximiert werden.

6.1. Kostenapproximation

Während sich die Programmierer*innen um die Wartung und Erweiterung des Automatisierungsprozesses kümmern, sind die 3D Artist damit beschäftigt, die Häuser zu kontrollieren und fertigzustellen. In der Regel werden 5 Häuser pro Arbeitstag (8 Stunden Vollzeit) fertig gestellt. Der Gewinn folgt aus der Annahme, dass durch die gesparte Zeit mehr Häuser pro Monat erstellt werden können.

6.1.1. Beispiel Rechnung der gesparten Arbeitszeit

Die Folgende Kalkulation zeigt durch ein Beispiel den Gewinn an Zeit, wenn ein Automatisierungsprozess 5 Minuten Zeit pro Haus einspart:

- Die Bearbeitung pro Haus dauert 96 min.
- $480 \text{ min Gesamtzeit} / 5 \text{ Häuser} = 96 \text{ min}$.
- Bei einer Zeitersparnis von 5 min (4,8%) dauert ein Haus 91 min.
- Insgesamt ergibt sich eine Zeitdifferenz von 25 min pro Tag ($480 \text{ min} - 455 \text{ min}$).
- Wenn man Die Zeitersparnis auf den Monat hochrechnet ($25 \text{ min} \times 5 \text{ Tage} \times 4 \text{ Wochen}$), erhält man 500 min
- Konklusion: der 3D Artist kann pro Monat (mit 20 Arbeitstagen und ausgen. Feiertagen) durch einen weiteren Automatisierungs-Prozess 5,2 Häuser mehr erstellen (was eine gesteigerte Wirtschaftlichkeit von einem gewonnenen Arbeitstag entspricht).

6.1.1.1. Zeitersparnis pro Haus mit Automatisierung

Mit Automatisierungsprozess	Anzahl Häuser	Bearbeitungszeit	Zeitersparnis
Nein	1	96 min	0 min
ja	1	91 min	5 min
ja	5	455 min	25 min

Tabelle 1: Zeitersparnis pro Tag

6.1.1.2. Zeitersparnis pro Monat

Anzahl Häuser	Bearbeitungszeit	Zeitersparnis
5	Pro Tag	25 min
25	Pro Woche	125 min
100	Pro Monat	500 min

Tabelle 2: Zeitersparnis pro Monat

PROJEKT PHASIERUNG

7. Projektphasierung in Stunden

Nummer	Phase	Stunden
1.	Definitionsphase	4
2.	Analysephase	6
2.1	Ist-Analyse	6
2.2	Definition des Soll-Zustandes	5
3.	Konzeptionsphase	
3.1	Datenanalyse des Assets	5
3.2	Konzeptionierung von möglichen Erweiterungen	5
3.3	Softwarespezifische Programmierkenntniserweiterung	10
4.	Implementierungsphase	
4.1	Implementierung der User Interface Erweiterung	10
4.2	Implementierung der Programmierung	25
5.	Testphase	
5.1	UI test	1
5.2	Asset Platzierung	1
6.	Kontrollphase (zwischen Ist- und Sollzustand)	1
7.	Abschlussphase	1
8.	Dokumentationsphase	5
Gesamt		80

Tabelle 3 : Projektphasierung

8. Definitionsphase

Während der Definitonsphase wurde die Notwendigkeit eines fehlenden Arbeitsprozesses definiert. Hierbei wurde recht früh ersichtlich, dass es einige Prozesse gibt, welche die Jobautomatisierung der Firma Visionme GmbH vorerst nicht konnte. Die Schwierigkeit bei der Programmierung von den Prozessen ist es, die komplexen Geometrischen Formen von bestimmten Objekten dem Ziel einzuordnen. Zur Auswahl stand Die Platzierung der Türen, die Geometrie der Gärten und die Platzierung der Treppe.

Aufgrund der hohen Anforderung automatisierte Prozesse zu erstellen, diese zu implementieren und es dem individuellen Kunden*innen gerecht zu konfigurieren, musste ihm Rahmen des Projektes für den Autoren eine gerechte und realistische Herausforderung erstellt werden.

Die internen Programmierer*innen hatten bereits die Platzierung der Türen programmiert aber nicht den verschiedenen Kunden*innen angepasst und wurde demensprechend nicht von den Autoren der Projektarbeit bearbeitet. Es wurde beschlossen, dass ein Tool entwickelt werden muss, welches sich um die Platzierung der Assets kümmert, weil die jeweiligen Prozesse die Platzierung eines Assets darstellen. Das Fehlen eines Prozesses bedeutet jeweils die händische Überarbeitung von den 3d Artists. Hierfür kam die Programmierung eines Tools infrage, welches die Bearbeitungszeit für die Häuser reduziert und somit die Menge der Fertigstellung der Häuser erhöht.

8.1. Ist/Soll Analyse

8.1.1. Ist Zustand

Wohnungsbesichtigungen sind für Kunden*innen der Immobilienunternehmen mühsam und aufwändig. Damit der Kunde*in des Immobilienunternehmens die Möglichkeit hat vorab via Smartphone ein Haus online zu besichtigen, bedarf es eine virtuell begehbare Software. Diese Funktion wird von der Firma Visionme GmbH bereitgestellt, da der technische Aufwand für die internen Mitarbeiter*innen der Immobilienunternehmen zu hoch ist bzw. die Technik gar nicht erst zu Verfügung steht.

Die Firma Visionme stellt für Immobilienunternehmen 3D Visualisierungen (Computer generierte Bilder und begehbare Häuser) her. Es werden 3-dimensionale Häuser unmöbliert, unbeleuchtet und teils noch halbfertig von den Architekten der Immobilienfirmen an die Firma Visionme GmbH überreicht. Diese Daten werden mit einem Automatisierten verfahren in einer 3D Engine (Unity 3D) aufgewertet. Damit die Automatisierung den stetig variierenden Anforderungen und wünschen der Immobilienunternehmen gerecht werden kann, müssen Erweiterungen programmiert werden.

8.1.2. Soll-Zustand

Die Automatisierung spart Zeit, in dem die händische 3d Bearbeitung der täglich anfallenden Häuser im Idealfall wegfällt. Mit der eingesparten Zeit können mehr Häuser pro Tag bearbeitet werden und die daraus resultierenden Fertigstellungen erhöhen den Gewinn für die Firma. Das Ziel ist demensprechend die maximale Automatisierung der Haus-jobs. Leider sind nicht alle Prozesse auf Anhieb Automatisierbar, weil einige 3D Objekte aufgrund von Komplizierten geometrischen Strukturen aufweisen. Die Unity Engine bietet zwar ein integriertes Tool an, bekannt als den „Pro Builder“, welches aber nur in der Lage ist Polygone zu bearbeiten, nicht aber ein Automatisiertes Platzieren und Anpassen

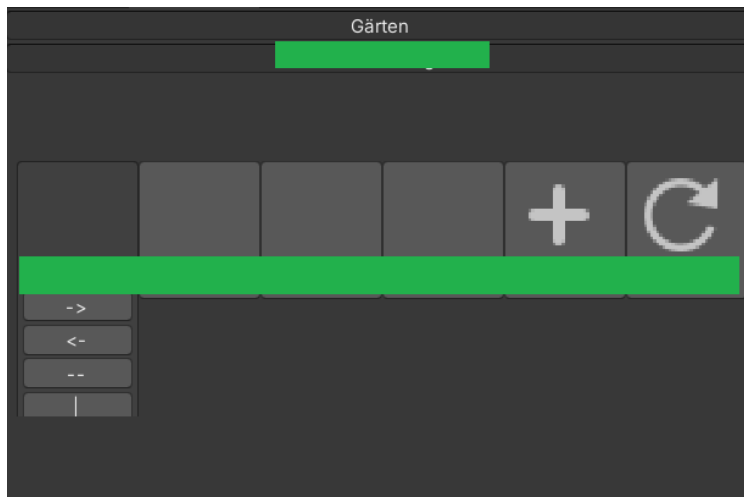
der ausgewählten Objekte. Die Lösung hierfür ist eine Art Universalwerkzeug zum optimierten Platzieren und Anpassen von komplexen Geometrischen 3D Objekten.

KONZEPTIONSPHASE

9. Konzeptionsphase

9.1. Entwurf der Benutzeroberfläche (OnGUI Funktion)

Für den Entwurf der Benutzeroberfläche wurde die Visionme Gartentool als Vorbild genommen.



10. Datenanalyse des Assets

10.1. Garten

Die Gärten sind als Assets vorgefertigt, die über die U.I geladen werden. Die Gärten selbst sind intern von den 3D Artist erstellt worden (dem Kundenkatalog entsprechend). Aufgrund der Individualität der Häuser, die von den Kunden*innen geliefert kommen, passen die Häuser nicht ganz auf die Gärten. Die Häuser dürfen ihre Geometrie nicht verändern und haben auch meistens eine Garage, deshalb muss der Garten um das Haus passend platziert werden. Eine weitere Schwierigkeit ist hierbei, dass die Häuser mehrstöckig sein können und dem entsprechend verschiedene Ausgänge, z.B. vom Keller aus zum Garten, was die Generierung erschwert. besonders die Wege neben den Häusern sind detailreich und können nicht einfach verändert, wenn das Haus einen Anbau hat (Abbildung 3: Haus mit Anbau).

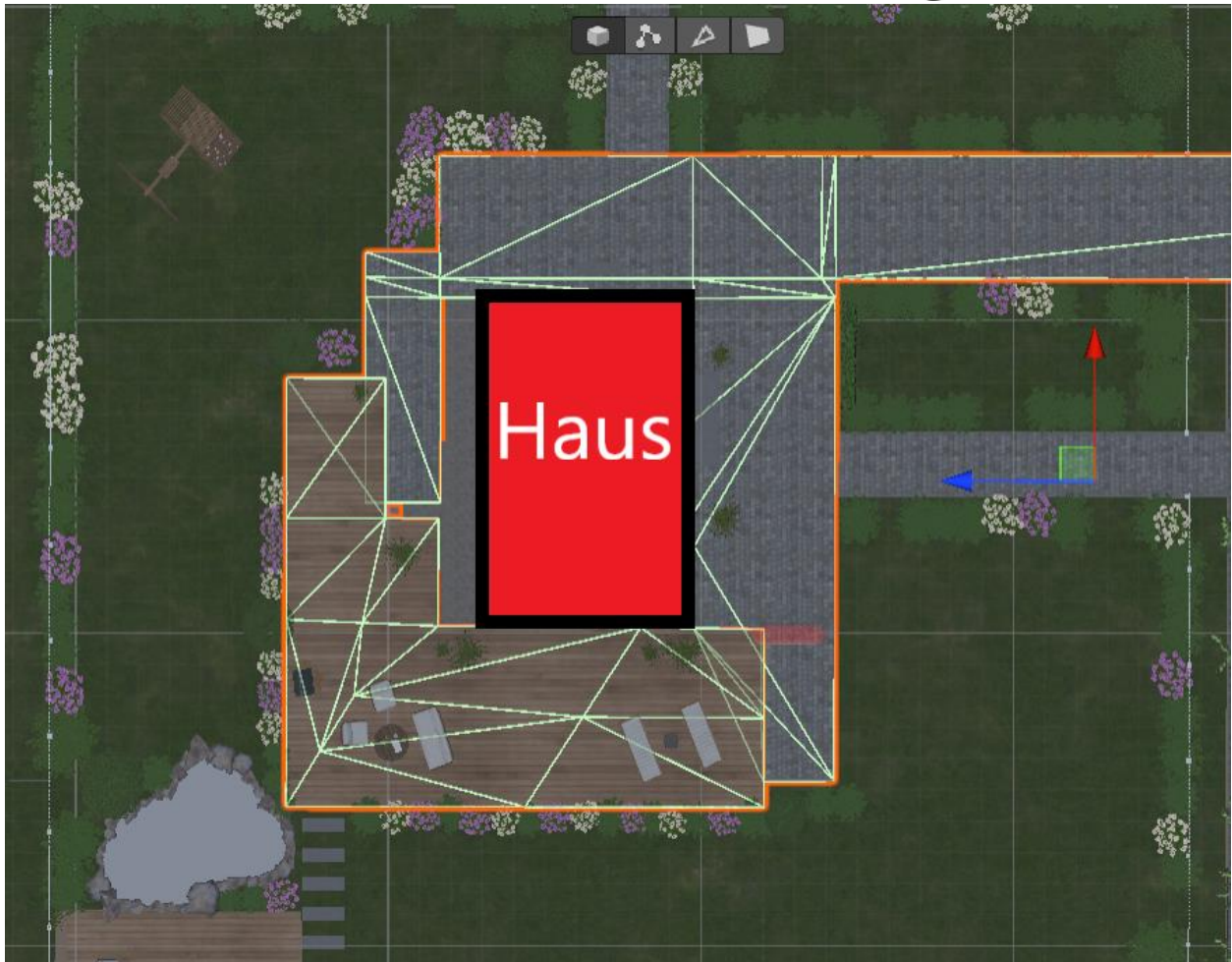


Abbildung 2: Top-Down Ansicht Haus, ungerendert



Abbildung 3: Haus mit Anbau

10.2. Treppe

Die Treppe stellt wegen seiner geometrischen Komplexität eine ähnliche Herausforderung dar, mit dem Unterschied, dass die Treppe immer das Problem mit der richtigen Platzierung zwischen den Etagen hat. Es Gibt für die Treppe drei verschiedene Grundformen, welche ihre Geometrie nicht verändern dürfen, lediglich skaliert werden können (Größenordnung). Die drei Grundformen für den Kunden sind jeweils Gerade, gewandelt um 90 Grad und gewandelt um 180 Grad. (Abbildung 4)

Eine Gerade Treppe, eine sich um 90 Grad gedrehte Wendeltreppe und eine Wendeltreppe, die sich jeweils um 180 Grad dreht.



Abbildung 4: Treppenarten, ungerendert

ASSETPLACER PROGRAMMBESCHREIBUNG

11. Asset Placer

12. Softwarespezifische Programmierkenntniserweiterung für den Asset Placer

Für das Erstellen des Asset Placers bedarf es Grundverständnisse in 3D Design, der Unity 3d Engine, C# als Programmiersprache und der Unity 3d Library, welche sich aus der .NET-Framework ableitet. Das Wissen in allen dieser Bereiche erforderte unerwartet viel Zeit, was die erwartete Zeit für das Aneignen überschreitet.

13. Technische Details

13.1. Allgemeine 3D Grundlagen Erläuterung

Grundsätzlich haben alle üblichen 3D-Engines ein Programm zur visuellen Darstellung des Koordinatensystems mit drei Achsen die jeweils als X-, Y- und Z bezeichnet werden. Das Koordinatensystem bildet die Grundlage zur Einordnung der Objekte relativ im Koordinatensystem.

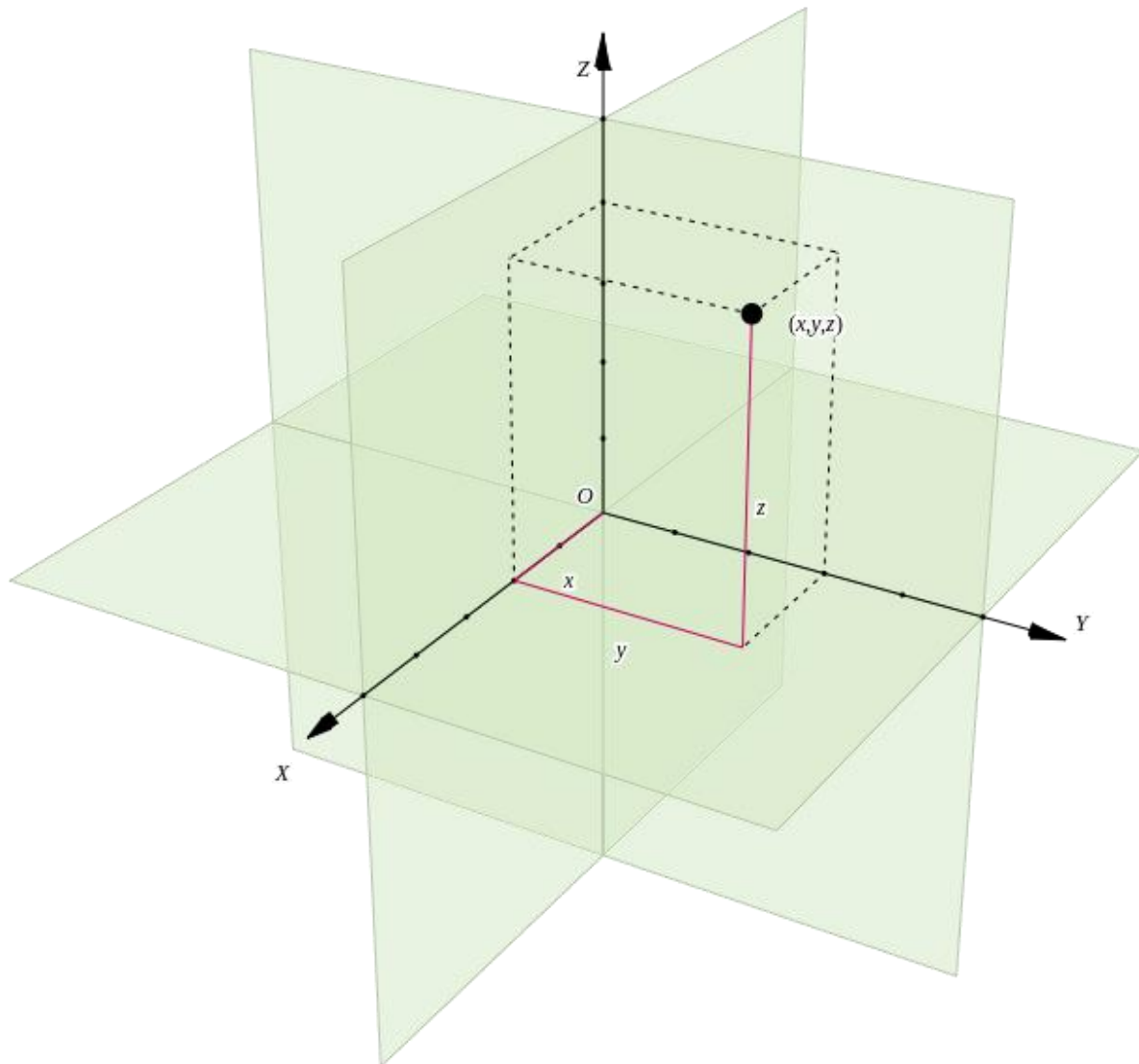


Abbildung 5: Koordinatensystem XYZ

Objekte bestehen aus Polygonen (Vieleck) die jeweils mit Strichen (Edges) verbunden werden, damit der Zwischenraum Flächen (Faces) bilden kann. ([Darstellung auf der nächsten Seite](#))

14. Abbildung Polygone

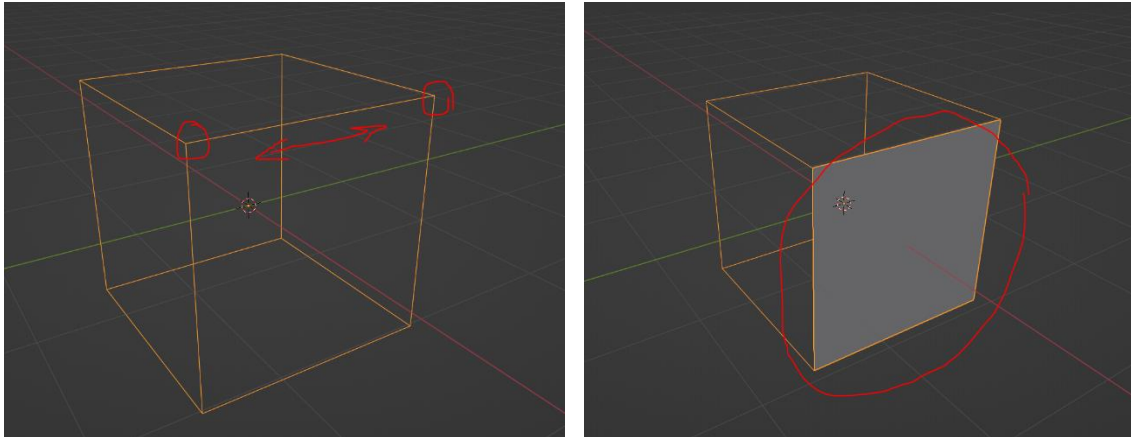


Abbildung 6: (linkes Bild) Abbildung ohne Flächen (rechtes Bild) Abbildung mit Fläche.

15. Benötigte Software

Für die Entwicklung der [U.I](#) Steuerung und die damit verbundene [Assets](#) Programmierung wurde die [IDE](#) Visual Studio code (Microsoft) mit der Programmiersprache C# und der Programmierbibliothek des .NET-Framework(Microsoft) verwendet.

[\(b\)Tabellarische Darstellung der genutzten Entwicklerwerkzeuge.](#)

15.1. Visual Studio Code

Visual Studio Code ist ein kostenfreier Quelltexteditor, der für die Projektarbeit genutzt wurde. Die Firma Visionme GmbH nutzt für die Programmierung die IDE-Rider, Die eine intelligente Lösung durch gute Autokorrektur, Refraktion und gute Performanz bietet. Doch Rider selbst ist kostenpflichtig und für die Aufgabe reicht Visual Studio Code aus. Unity 3D selbst hat die native IDE den Monodevelop, der standardmäßig für die Aufgabe ausreichen würde, dennoch wurde in diesem fall Visual Studio Code genutzt, weil der Visual Studio Code mit umfangreicheren Programmiererweiterung eine bessere Lösung bietet.

15.2. Entwicklungsumgebung: Unity 3D

Für Die Projektarbeit wurde als Hauptentwicklungsumgebung die Unity 3D Engine genutzt. Die Unity 3D Engine wird in der Industrie hauptsächlich zur Spieleherstellung verwendet, kann aber auch anderweitig für Allgemeine graphische Darstellungen genutzt werden. Die Engine verfügt über eine Laufzeitumgebung, welche die 3-Dimensional erstellten Szenen in Echtzeit darstellt. Die fertigen Szenen sind von der Engine auf gängige Plattformen exportierbar, wie z.B. auf Android, IOS, Windows, Mac-OS usw. Die Firma Visionme GmbH nutzt hierbei die Möglichkeiten der Echtzeitdarstellung um die Objekte, welche Sie von den Kunden*innen erhalten, virtuell darzustellen und diese dann auch virtuell zu begehen. Besonders vorteilhaft ist die [VR](#)-Funktion, den der Kunde*innen nutzen kann, um das Haus vorab virtuell zu besichtigen.

15.2.1. Übersicht der Unity-Benutzeroberfläche

15.2.1.1. Scene view

Auf der Szenenansicht ([Abbildung Benutzeroberfläche der Unity Engine](#), rot markiert) sieht man die Scene mit allen Objekten. Objekte werden einem Koordinatensystem (Global-Space) im virtuellen Raum dargestellt. Der virtuelle Raum ist standardmäßig ein [Euklidischer Raum](#).

15.2.1.2. Inspector

Der Inspektor ([Abbildung Benutzeroberfläche der Unity Engine](#), grün markiert) gibt die Parameter der Objekte wieder, darunter auch die wohl wichtigsten Parameter: Location, Rotation und Scale. Mit den angegebenen Parametern lassen sich die wichtigsten ortgebundenen Eigenschaften über das Objekt bestimmen, wie z.B. Wo es sich im Raum (Szene) befindet, in welche Richtung das Objekt sich ausrichtet (nach den X, Y, Z – Koordinaten/Breite, Länge und Höhe) und wie groß das Objekt in Relation zu den anderen Objekten ist. Alle Ausrichtungs-Eigenschaften, die das Objekt betreffen, werden bei der Programmierung als [Lokal](#)-Space ([Pivot-Point](#) des Objektes relativ zu anderen Objekten) und der Raum selbst als [Global](#)-Space (Koordinaten Ursprung) angegeben.

Des Weiteren können Objekte weitere Komponenten besitzen, darunter [Materials](#), [Textures](#), [Skripte](#) und [Animationen](#).

15.2.1.3. Hierarchie

Die Hierarchie ([Abbildung Benutzeroberfläche der Unity Engine](#), gelb markiert) ordnet die Objekte in der Szene an. Objekte können hierbei miteinander verknüpft werden (Child/Parent kann Verhältnis).

15.2.1.4. Projektordner

Der Projektordner ([Abbildung Benutzeroberfläche der Unity Engine](#), blau markiert) Lagert alle Assets, Die in die Scene platziert werden können.

15.2.1.4.1. Visionme – Tab

Die Visionme – Leiste ([Abbildung Benutzeroberfläche der Unity Engine](#), Violet markiert) ist eine Programmerweiterung in Form eines UI-Elements mit eigenen Befehlen (separat von der Job Automation), mit der man auf selbsterstellte Programme zugreifen kann. (Keine beteiligte Leistung des Autors)

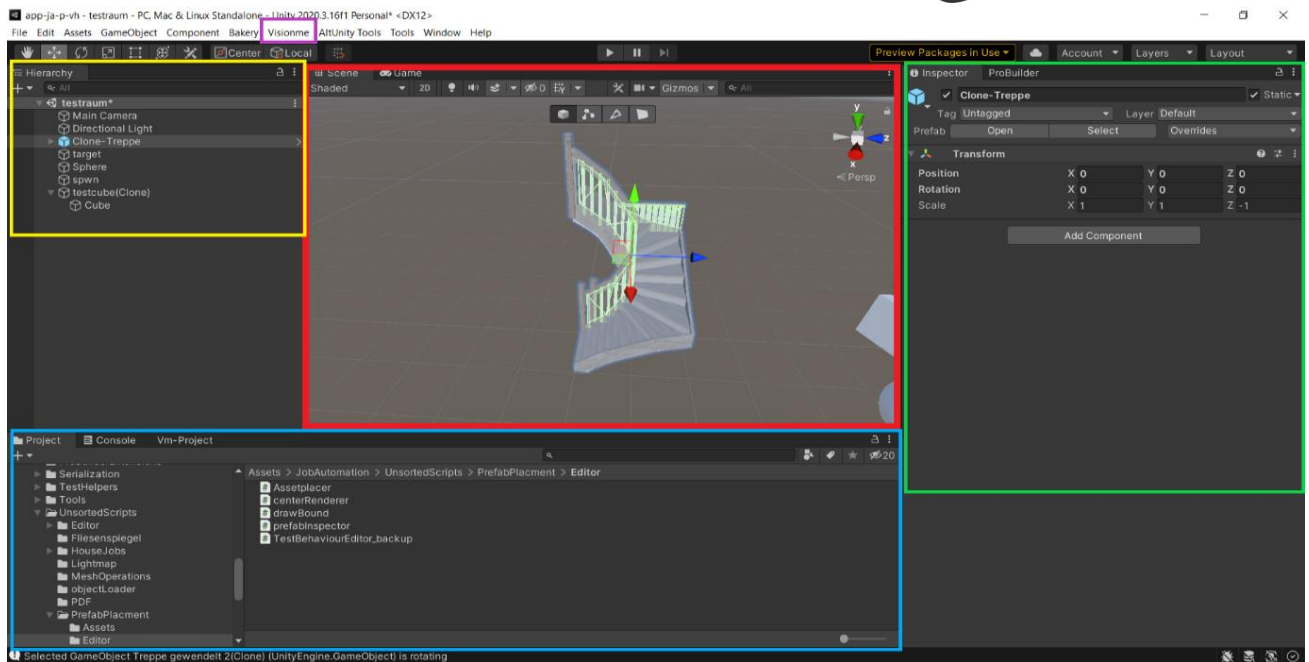


Abbildung 7: Benutzeroberfläche der Unity Engine

16. Hausinterne Erweiterung: Job Automation

16.1. Erläuterung

Die Unity Engine bietet eine interne Entwicklungsumgebung an, was genutzt werden kann, um eigene Programmerweiterungen zu programmieren. Hierbei nutzt die Unity Engine die .NET-Framework – Bibliothek von Microsoft mit C# als hauptsächliche Programmiersprache. Die Praktikumsfirma hat hierzu eine Hauseigene Erweiterung mit der Bezeichnung: „[Job Automation](#)“ erstellt. Die Erweiterung verfügt über viele verschiedene Funktionalitäten, um die unbearbeitete rohe Szene, die von den Kunden*innen geliefert kommt methodisch dem Kundenwunsch entsprechend zu bearbeiten. Zunächst wird über die Datenbank auf Strapi zugegriffen. Dort sind die Häuser in Form von IDs markiert. Die ID wird über den Init- HouseJob Knopf auf die Festplatte des Bearbeiters geladen und in der Unity Engine initialisiert. Nachdem die rohe Fassung des Hauses geladen wurden ist, steht nun die Option eine Präparation vorzubereiten (aufgrund des Betriebsgeheimnisses in der Dokumentation nicht sichtbar.) Die Präparation läuft automatisch durch und bearbeitet alle anfallenden Aufgaben (Möblierung, Beleuchtung, Objekte austauschen, etc.) Die „Job Automation“ selbst war kein Bestandteil der Praktikumsaufgabe und dementsprechend wurde für die Erweiterung keine Leistung des Autors erbracht.

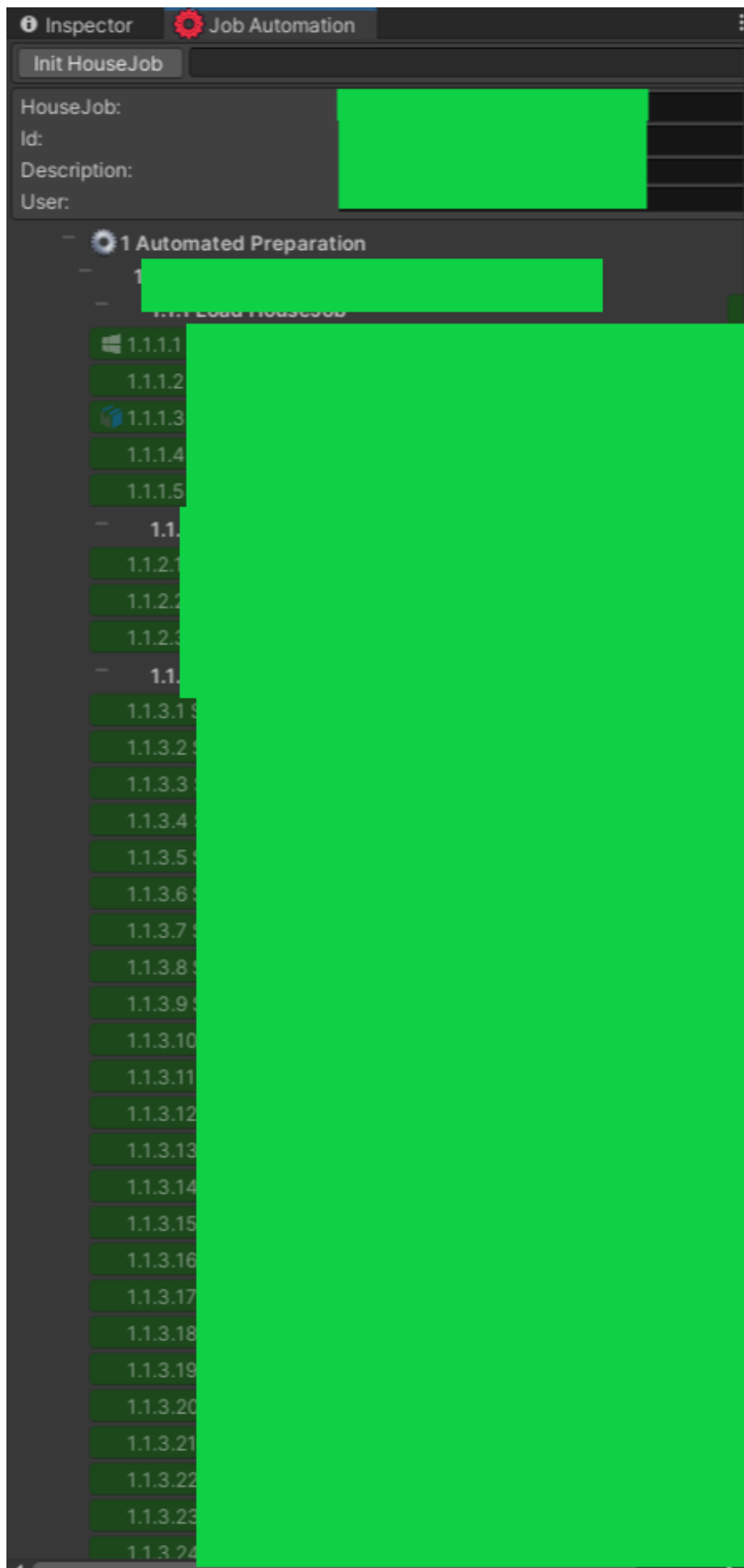


Abbildung 8: Job Automation

17. Team-Interne Kommunikation

17.1. Werkzeuge

Zur Kommunikation von formalen Angelegenheiten wurde der E-Mail-Dienst von Strato genutzt. Innerhalb der Abteilungen wurde der Instant Messenger Discord angewandt und zu Meetings wurde der online Streaming-dienst Microsoft Teams verwendet.




Tool	Discord	Strato	
Funktion	Instant Messenger	Formaler E-Mail Dienst.	Streaming-dienst für Online-Meetings
Logo			

Tabelle 4: Team-interne Kommunikation




Tool	Unity 3D Engine	Visual Studio Code	Strapi
Funktion	Laufzeit- und Entwicklungsumgebung,	IDE-Erweiterung zur Programmierung mit C# und der .NET-Framework	Content-Management-System als Schnittstelle für die Übertragung der Häuser
Logo			

Tabelle 5 : Benötigte Software

17.2. Konzeptionierung von möglichen Erweiterungen (technische Details der Programmierung)

Für das Tool wurde ein klassischer GUI-Tab ausgewählt. (Abbildung 7: GUI-Layout garten) Ein GUI-Tab erlaubt den schnellen Zugriff auf die Erweiterung über den Inspektor. Aufgrund der nicht Fertigstellung des Tools, ist es vorerst nur als Skript mit Buttons vorhanden, bietet aber im Grunde die gleichen Funktionsmöglichkeiten.

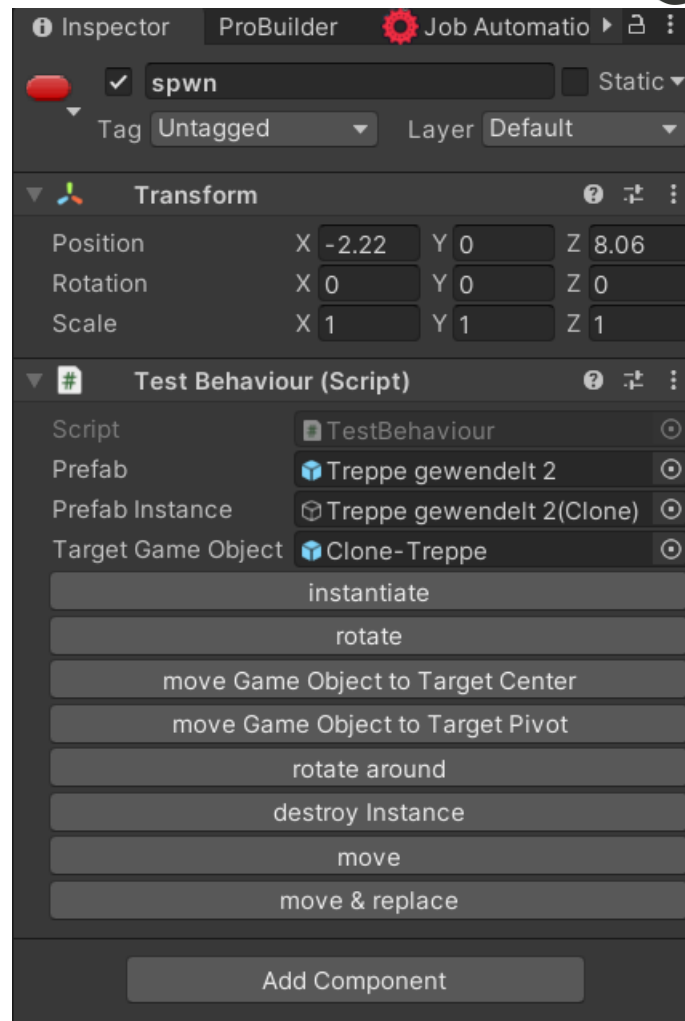


Abbildung 9: GUI-Element, Asset Placer

17.3. Mögliche Funktionen die für die Erweiterung benötigt werden.

17.3.1. Singleton Methode

Die Singleton Methode ist ein Entwurfsmuster, welches nur ein Objekt einer Klasse erlaubt. Somit wird die Einzigartigkeit des Objektes sichergestellt. Beim Erstellen eines Objektes (Garten oder Treppe) aus dem U.I kann dann jeweils ein Objekt erstellt werden, was nicht versehentlich zweimal existiert. Da nur ein Garten gegeben ist, könnte es Praktisch sein. Die Singleton wurde vorerst nicht implementiert, weil der Asset Placer sich primär um die Treppe kümmert. In Einem Haus können mehrere Treppen stehen, was diese Funktion zu mindestens für die Treppe selbst überflüssig macht.

```
public class Singleton : MonoBehaviour
{
    public static Singleton Instance { get; private set; }
}
```

Abbildung 10: singleton class Beispiel

17.3.2. GUI- Layout

Die Entwickler haben einen Gartentool erstellt, welches einen Garten aus dem vorgefertigten Katalog erstellen kann, mit jeweils der Funktion, dass Objekt zu rotieren und zu spiegeln. Die Funktion des Erstellens eines Objektes und es zu rotieren wurden als Vorlage für den Assetplacer übernommen, jedoch wurde der Assetplacer eigenständig ohne Einblick in den Gartentool programmiert, so dass der Lerneffekt nicht verloren geht.

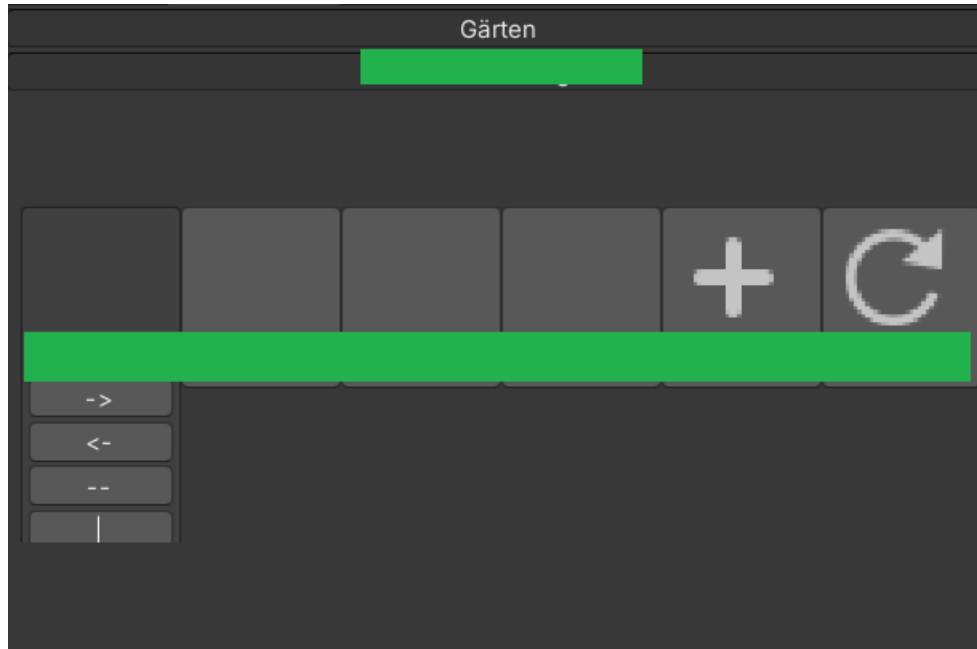


Abbildung 11: GUI-Layout garten

17.3.3. Clusterobjekte

Clusterobjekte sind Objekte, die in einer Parent-Child Beziehung zu einem Übergeordnetem Objekt bestehen. In der App „Immobilien Designer“ werden die Clusterobjekte katalogisiert und auswählbar angezeigt. Alles Spezifische hierzu ist schon vorprogrammiert und bedarf keinen weiteren Programmieraufwand.

17.3.4. GUI-Button

Für die einzelnen Funktionen, die implementiert wurden, gibt es jeweils ein Button. Die jeweiligen Buttons sind tabellarisch aufgelistet.

Funktion	Erläuterung
Instantiate	Erstellt das Objekt
Rotate	Rotiere das Objekt um 90 Grad gegen den Uhrzeigersinn um sich selbst
Move Game Objekt to Target Center	Platziere das Objekt zum Zielobjekt-Mitte
Move Game Objekt to Target Pivot	Platziere das Objekt zum Zielobjekt-Schwerpunkt
Rotate around	Rotiere das Objekt um das Zielobjekt um 90 Grad gegen den Uhrzeigersinn
Destroy instance	Lösche das erstellte Objekt
Move	Bewege das Objekt (reine Testfunktion)

Move & replace

Bewege die erstellten Objekte zum Zielobjekt und lösche das Zielobjekt (ersetze es)

Tabelle 6 : GUI-Button

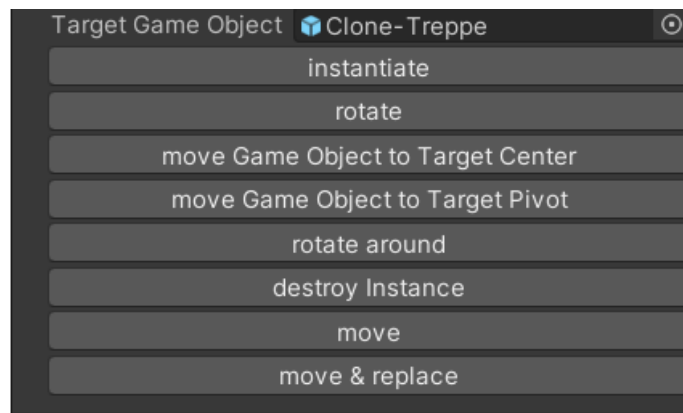


Abbildung 12: GUI-Button Layout

17.3.5. Select Object

Die Grundidee des Asset Placers ist es, das ausgewählte Asset zum Zielobjekt zu bewegen, um das Zielobjekt mit dem ausgewählten Asset zu ersetzen. Hierfür bedarf es den Zugriff auf den Unity Editor. Grundsätzlich wird der Unity Editor und der Unity Runtime unterschieden. Alles, was im Runtime stattfindet wird während der Echtzeit Exekution des Programms abgespielt. Die Editor-Funktion hingegen überschreibt die in der UI angezeigt Elemente (Inspektor). Damit man auf die GUI- Elemente zugreifen kann wird aus der basisklasse von Unity, dem MonoBehaviour zugegriffen und auf auf die Klasse Testbehaviour vererbt (testbehaviour bezeichnet den Asset Placer als Klasse). Sobald man GameObjecte auf public stellt, werden diese Elemente auf der GUI Angezeigt.

17.3.6. Custom Editor

Damit man auf der GUI die Funktionen überhaupt sehen kann, benötigt man einen Custom-Editor, die man aus der Basisklasse des Unity-Editor erbt.

```
using UnityEditor;
using UnityEngine;
[CustomEditor(typeof(TestBehaviour))]
public class TestBehaviourEditor : Editor
{

```

Abbildung 13 : Custom-Editor

17.3.7. Override OnInspectorGUI

Die Override Methode überschreibt den Inspektor und ist notwendig um alle Funktionen im Inspektor anzeigen zu können

```
public override void OnInspectorGUI()
{
    base.OnInspectorGUI();
    var testBehaviourInstance = (TestBehaviour)target;

```

Abbildung 14: Override Methode

17.3.7.1. Prefabs

Das ausgewählte Asset , was auf Public gestellt worden ist, wird per drag & drop auf den GUI-Button „prefab“ draufgezogen. Aus dem Prefab wird dann ein neues Objekt instanziiert. Aus der Vorlage können nun beliebig viele Objekte generiert werden, um den bedarf an Treppen zu decken. Target Game Objekt dient als Zielobjekt. (Abbildung 9: Prefab GUI und Abbildung 10: Prefab-auswahl im Editormode auf der nächsten Seite).



Abbildung 15: Prefab GUI im Inspektor

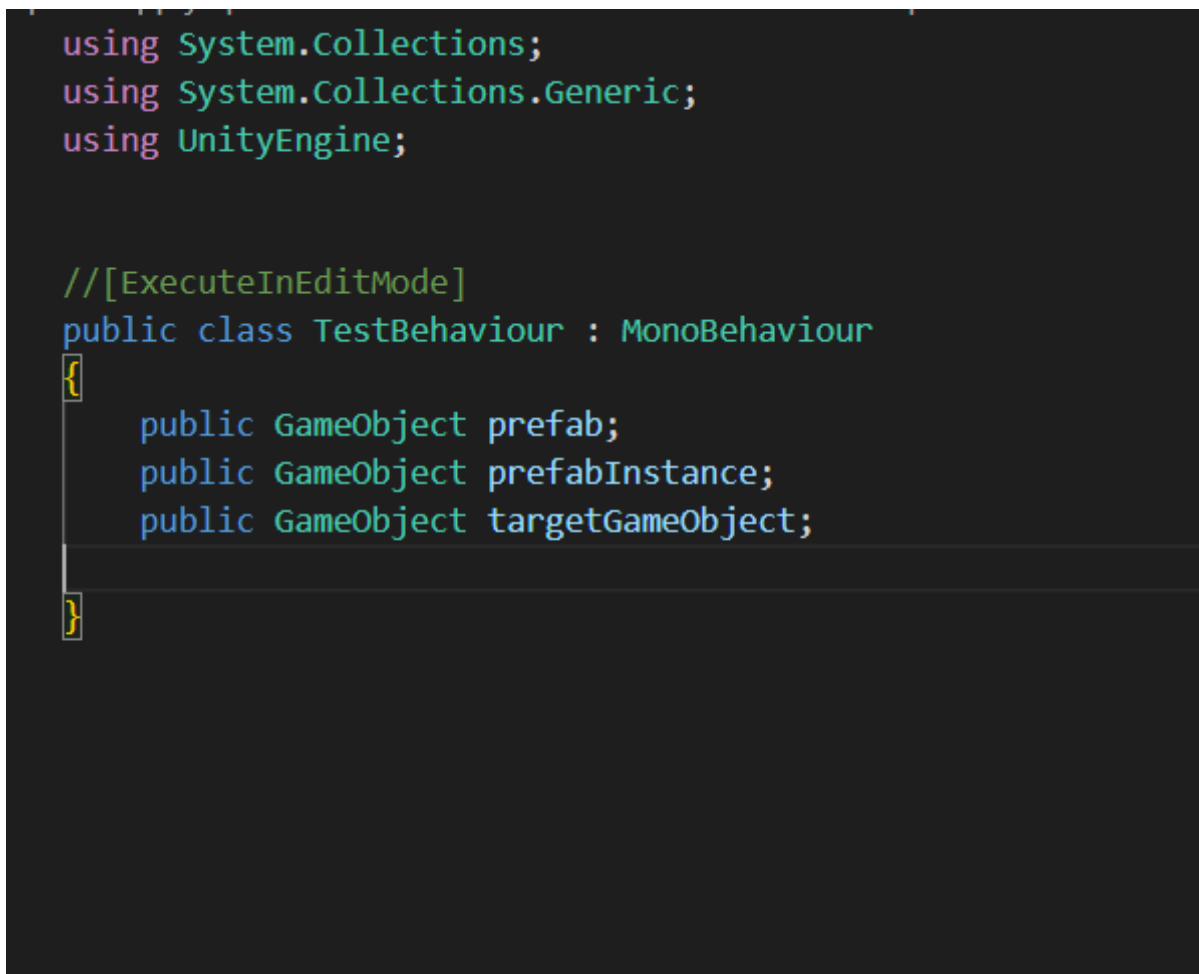


Abbildung 16 : Prefab-auswahl im Editormode als Code

17.3.7.2. Move to Methode (GUILayout.Button)

Mit dieser Methode wird die Prefab Instanze dem Target Center zugewiesen, damit es später mit einer anderen Methode überarbeitet werden kann. Das Target Center ist hierbei der Mittelpunkt des Cluster-Objektes, welches mit einer weiteren Methode definiert wird (Bounding box Methode)

```

}
if (GUILayout.Button("move Game Object to Target Center"))
{
    var gameObjectCenter = GetRenderCenter(testBehaviourInstance.prefabInstance);
    var targetCenter = GetRenderCenterTarget(testBehaviourInstance.targetGameObject);
    testBehaviourInstance.prefabInstance.transform.position = targetCenter;
}
if (GUILayout.Button("move Game Object to Target Pivot"))
{
    var gameObjectCenter = GetRenderCenterTarget(testBehaviourInstance.prefabInstance);
    var BottomBackLeft = GetBottomBackLeft(testBehaviourInstance.targetGameObject);
    testBehaviourInstance.prefabInstance.transform.position = BottomBackLeft;
}

```

Abbildung 17 : GUI-Layout.Button move Methode

17.4. Bounding Box - Funktion

Die Bounding Box ist Objekt in Form eines Quaders. Der Masseschwerpunkt der Bounding Box besteht aus der zentralen Interpolation der einzelnen Objektkomponenten und ist mit der Rotation absolut. Durch die absolute Lage der Achsen kann die Box mit den Parametern Center und Extents ($\frac{1}{2}$ der Größe der Bounding Box) arbeiten. Sinn und Zweck ist es, durch die Interpolation einen gemeinsamen Mittelpunkt des Cluster Objektes zu finden, um das Cluster Objekt an einen Gewünschten Ort einheitlich und präzise zu platzieren. Meistens liegt ein Platzhalter Objekt (z.B. eine Treppe) von dem Kunden*innen bereits in der Szene was anschließend von dem Cluster-Objekt ersetzt wird.

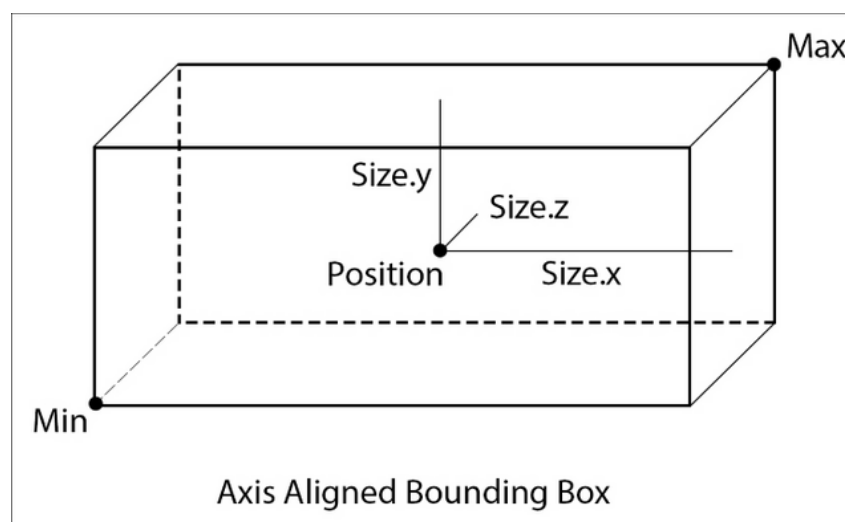


Abbildung 18: Bounding Box Parameter

Bounds

struct in UnityEngine / Implemented in:UnityEngine.CoreModule

[Leave feedback](#)

Description

Represents an axis aligned bounding box.

An axis-aligned bounding box, or AABB for short, is a box aligned with coordinate axes and fully enclosing some object. Because the box is never rotated with respect to the axes, it can be defined by just its [center](#) and [extents](#), or alternatively by [min](#) and [max](#) points.

[Bounds](#) is used by [Collider.bounds](#), [Mesh.bounds](#) and [Renderer.bounds](#).

Properties

center	The center of the bounding box.
extents	The extents of the Bounding Box. This is always half of the size of the Bounds.
max	The maximal point of the box. This is always equal to center+extents.
min	The minimal point of the box. This is always equal to center-extents.
size	The total size of the box. This is always twice as large as the extents.

Abbildung 19:Bounding Box Parameter

Constructors

Bounds	Creates a new Bounds.
------------------------	-----------------------

Public Methods

ClosestPoint	The closest point on the bounding box.
Contains	Is point contained in the bounding box?
Encapsulate	Grows the Bounds to include the point.
Expand	Expand the bounds by increasing its size by amount along each side.
IntersectRay	Does ray intersect this bounding box?
Intersects	Does another bounding box intersect with this bounding box?
SetMinMax	Sets the bounds to the min and max value of the box.
SqrDistance	The smallest squared distance between the point and this bounding box.
ToString	Returns a formatted string for the bounds.

Abbildung 20:Bounding Box Methode

17.5. GetComponentisinChidren -Methode und Encapsulate

Die GetComponentisinChidren- Methode holt alle definierten Child-Objekte aus dem Parent-Objekt und mit Encapsulate wird eine Bounding Box draufgelegt, damit es der Masseschwerpunkt anschließend interpoliert werden kann.

```
static Vector3 GetRendererCenterTarget(GameObject go) // gameobject go placeholder
{
    var renderers = go.GetComponentsInChildren<Renderer>(); // get renderer
    var rendererBound = new Bounds();
    foreach (var renderer in renderers)
    {
        rendererBound.Encapsulate(renderer.bounds); // get renderer.bounds
    }
    Vector3 rbEx = rendererBound.extents;
    Vector3 rbC = rendererBound.center;
    Vector3 rbMin = rendererBound.min;
    Vector3 rbMax = rendererBound.max;

    return rbC;
}
```

Abbildung 21: GetComponentisinChidren

18. Weitere Funktionen und Vollständiger Code

Alle weiteren Funktionen wie Rotate sind mit der Nachfolgenden Abbildung ersichtlich.

```
1  using UnityEditor;
2  using UnityEngine;
3  [CustomEditor(typeof(TestBehaviour))]
4  public class TestBehaviourEditor : Editor
5  {
6
7      static Vector3 GetRendererCenter(GameObject go) // gameobject go placeholder
8      {
9          var renderers = go.GetComponentsInChildren<Renderer>(); // get renderer
10         var rendererBound = new Bounds();
11         foreach (var renderer in renderers)
12         {
13             rendererBound.Encapsulate(renderer.bounds); // get renderer.bounds
14         }
15         Vector3 rbEx = rendererBound.extents;
16         Vector3 rbC = rendererBound.center;
17         Vector3 rbMin = rendererBound.min;
18         return rbC;
19     }
20
21 }
22 static Vector3 GetRendererCenterTarget(GameObject go) // gameobject go placeholder
23 {
24
25     var renderers = go.GetComponentsInChildren<Renderer>(); // get renderer
26     var rendererBound = new Bounds();
27     foreach (var renderer in renderers)
28     {
29         rendererBound.Encapsulate(renderer.bounds); // get renderer.bounds
30     }
31
32     Vector3 rbEx = rendererBound.extents;
33     Vector3 rbC = rendererBound.center;
34     Vector3 rbMin = rendererBound.min;
35     Vector3 rbMax = rendererBound.max;
36
37     return rbC;
38 }
```

Abbildung 22: Vollständiger Code 1

```
// rbPosition = transform.TransformPoint(rbMin);
}

// // TODO: erstelle pivot obj als platzhalter um target pivot zu kreieren...
// [SerializeField]
// Vector3 pivotPosition; // the spawn position of the pivot point

// Transform pivot;

// void Start()
// {
//     targetPivot= new GameObject().transform; // create the pivot point
//     targetPivot.position = pivotPosition; // position the pivot point
//     transform.SetParent(targetPivot); // parenting
// }

static Vector3 GetBottomBackLeft(GameObject go) // gameobject go placeholder
{
    // RectTransform myRectTransform = GetComponent<RectTransform>();
    // myRectTransform.localPosition += Vector3.right;

    var renderers = go.GetComponentsInChildren<Renderer>(); // get renderer
    var rendererBound = new Bounds();
    foreach (var renderer in renderers)
    {
        rendererBound.Encapsulate(renderer.bounds); // get renderer.bounds
    }
    Vector3 rbEx = rendererBound.extents;
    Vector3 rbC = rendererBound.center;
    Vector3 rbMin = rendererBound.min;
    Vector3 rbMax = rendererBound.max;
    var rbZero = Vector3.zero;
    //Vector3 rbPivot.GetComponent<RectTransform>().pivot;
```

Abbildung 23: Vollständiger Code 2

```

    return rbZero;
}

public override void OnInspectorGUI()
{
    base.OnInspectorGUI();
    var testBehaviourInstance = (TestBehaviour)target;

    // {
    //     Debug.Log($"prefabInstance is Null");
    // }
    if (GUILayout.Button("instantiate"))
    {
        Debug.Log($"Button pressed. Selected GameObject {testBehaviourInstance.prefab}");
        var prefabToInstantiate = testBehaviourInstance.prefab;
        //testBehaviourInstance.prefab.name = "Clone-Treppe";
        if (prefabToInstantiate == default)
        {
            return;
        }
        var spwnpointx = testBehaviourInstance.transform.position.x;
        var spwnpointy = testBehaviourInstance.transform.position.y;
        var spwnpointz = testBehaviourInstance.transform.position.z;
        // debug-feedback spwn position
        testBehaviourInstance.prefabInstance = Instantiate(prefabToInstantiate);
        testBehaviourInstance.prefabInstance.transform.position = new Vector3(spwnpointx, spwnpointy, spwnpointz);
        Debug.Log($"spwnpoint (: " + spwnpointx + "| " + spwnpointy + "| " + spwnpointz + ")");
    }
}

```

Abbildung 24:Vollständiger Code 3

```

}
if (GUILayout.Button("rotate"))
{
    if (testBehaviourInstance.prefabInstance != default)
    {
        Debug.Log($"Selected GameObject {testBehaviourInstance.prefabInstance} is rotating");
        var stepx = 0;
        var stepy = 90;
        var stepz = 0;

        testBehaviourInstance.prefabInstance.transform.Rotate(stepx, stepy, stepz);
    }
}
if (GUILayout.Button("move Game Object to Target Center"))
{
    var gameObjectCenter = GetRenderCenter(testBehaviourInstance.prefabInstance);
    var targetCenter = GetRenderCenterTarget(testBehaviourInstance.targetGameObject);
    testBehaviourInstance.prefabInstance.transform.position = targetCenter;
}
if (GUILayout.Button("move Game Object to Target Pivot"))
{
    var gameObjectCenter = GetRenderCenterTarget(testBehaviourInstance.prefabInstance);
    var BottomBackLeft = GetBottomBackLeft(testBehaviourInstance.targetGameObject);
    testBehaviourInstance.prefabInstance.transform.position = BottomBackLeft;
}
}

```

Abbildung 25:Vollständiger Code 4


```
if (GUILayout.Button("rotate around"))
{
    if (testBehaviourInstance.prefabInstance != default)
    {
        var stairsGameObjectCenter = GetRendererCenter(testBehaviourInstance.prefab);
        Debug.Log($"Selected GameObject {testBehaviourInstance.prefabInstance} is rotating");
        var stepx = 0;
        var stepy = 90;
        var stepz = 0;
        testBehaviourInstance.prefabInstance.transform.RotateAround(stairsGameObjectCenter, Vector3.up, stepy);
    }
}

if (GUILayout.Button("destroy Instance"))
{
    var instanceToDestroy = testBehaviourInstance.prefabInstance;
    DestroyImmediate(instanceToDestroy);
}

if (GUILayout.Button("move"))
{
    // var stairsGameObjectCenter = GetRendererCenter( testBehaviourInstance.prefabInstance);
    //testBehaviourInstance.prefabInstance.transform.position.x += 1;
}

if (GUILayout.Button("move & replace"))
{
    var targetGameObjectCenter = GetRendererCenter( testBehaviourInstance.prefabInstance);
    testBehaviourInstance.prefabInstance.transform.position = targetGameObjectCenter;
    DestroyImmediate(testBehaviourInstance.targetGameObject);
    var style = new GUIStyle(GUI.skin.button);
    style.normal.textColor = Color.blue;
    //GUILayout.Button("Select Game Object", style);
}
}
```

Abbildung 26: Vollständiger Code 5



19. Implementierungsphase

19.1. Implementierung der User Interface Erweiterung

Das Tool ist nicht fehlerfrei, z.B. die Rotate around funktioniert nicht so wie es soll. die Bounding Box Methode selbst macht das, was erwartet wird. Das Tool ist benutzbar.

19.2. Implementierung der Programmierung

Das Tool konnte noch nicht implementiert werden, damit es genutzt werden kann.

20. Testphase

Während der Testphase wurden Fehler gefunden. Der Code musste vielfach retirieren werden, um den momentanen Zustand zu erreichen.

20.1. UI testing

Die UI mit Ihren Buttons, die selbsterstellt worden sind, funktionieren einwandfrei.

20.2. Asset Platzierung

Die Assets werden so wie erwartet platziert. Das Programm ist schätzungsweise zu 90% Fertig, lediglich kleinste Funktionen müssen umgeschrieben werden und die richtigen GUI -elemente eingebunden werden (so wie das Gartentool).

21. Kontrollphase (zwischen Ist- und Sollzustand)

Zwischen dem Ist- und Sollzustand ist insgesamt eine große Diskrepanz entstanden.

Die Reiteration und Dokumentation haben jeweils am meisten Zeit beansprucht.

Die Phasierung zum Ende:

Nummer	Phase	Stunden
1.	Definitionsphase	2
2.	Analysephase	2
2.1	Ist-Analyse	2
2.2	Definition des Soll-Zustandes	2
3.	Konzeptionsphase	
3.1	Datenanalyse des Assets	2
3.2	Konzeptionierung von möglichen Erweiterungen	2
3.3	Softwarespezifische Programmierkenntniserweiterung	10
4.	Implementierungsphase	
4.1	Implementierung der User Interface Erweiterung	5
4.2	Implementierung der Programmierung	25
5.	Testphase	
5.1	UI test	1
5.2	Asset Platzierung	1
6.	Kontrollphase (zwischen Ist- und Sollzustand)	1
7.	Abschlussphase	1
8.	Dokumentationsphase	24
Gesamt		80

22. Abschlussphase

Zum Abschluss wurden alle wichtigen Daten zusammen getragen, um es in die Dokumentation zu packen.

23. Dokumentationsphase

Die Dokumentationsphase hat deutlich mehr Zeit beansprucht als erwartet, weil der Technische Aufwand unerwartet groß ist, um es einfach in eine Dokumentation zu packen.



24. Fazit

Zum Abschluss kann man zusammenfassen, dass die Projektzeit insgesamt eine gute Möglichkeit war, seine Programmierkenntnisse im Bereich der graphischen Entwicklung weiterzuentwickeln. Die entwickelte Erweiterung konnte zwar nicht ganz fertiggestellt werden, dennoch kann es als Ansatz genutzt werden, um nicht nur die Treppenerstellung fertigzustellen, sondern als allgemeines Objektplatzierungswerkzeug für diverse Bereiche des Prozesses der Automatisierung genutzt werden.

25. Selbstständigkeitserklärung

Dieses Dokument ist gemäß den Vorgaben des Abschlussprojektes der IHK selbstständig und ausschließlich von dem Autor erstellt worden.