

# PROJEKTDOKUMENTATION

## “I am the People”

Spieleprogrammierung

Nicolas Schön(68622), Marcel Kohnle(76856)

## Table of Contents

Spezifikation/Ziel des Spiels.....	3
Grundlagen .....	3
Aufgabenaufteilung (Game Production) .....	4
Nicolas Schön .....	4
Überwiegend Entwicklung und Programmierung der Grundlegenden Spielmechanik sowie der KI ...	4
Modellierung von einzelnen Gebäuden und Naturobjekten .....	4
Marcel Kohnle.....	5
Hauptsächlich die Gestaltung des Spiels und Erstellung der meisten Modelle. ....	5
Entwicklungsschritte/Implementierung (Game Programming) .....	6
Wechsel zwischen Tabletop und FPS .....	6
Levelgenerator/Mesh-Erzeugung.....	6
Level Manipulation .....	8
Platzierungssystem von Gebäuden .....	9
Auswählen von Personen und senden von Befehlen .....	11
Wegfindung(a*) .....	12
Schiessen des Katapultes .....	13
Interaktionssystem für FPV-Personen.....	14
Rekrutierung von neuen Personen .....	14
Ressourcenerzeugung (Förster und Farm sowie Mine).....	15
Jobsystem .....	17
Speichern und Laden von Leveln .....	19
Spieleinstellungen.....	22
Gegnerverhalten/Kampfsystem .....	22
Modellierung der Spielobjekte.....	25
Charaktere .....	25
Waffen und Werkzeuge .....	25
Gebäude.....	25
Natur .....	25
Erstellung von Animationscontrollern und Einfügen passender Animationen .....	26
Erstellung der Grafiken und Sprites im User Interface .....	27
Erstellung von Partikeleffekten wie z.B. Feuer, Rauch, Regen, usw. ....	27
Tag-Nacht-Zyklus mit Uhr, Tageszähler und Wetter .....	28

Shader für Wasser und Grid .....	28
Wasser .....	28
Grid .....	28
Schießen mit Pfeil und Bogen .....	29
Hit-Detection im FPS Modus .....	29
Szene für Kampagnenmodus .....	30
Post-Processing.....	30

## Spezifikation/Ziel des Spiels

Im Rahmen dieser Dokumentation wird die Entwicklung und die Funktionsweise der wichtigsten Bereiche des entwickelten Spiels erklärt.

Ziel dieses Projektes ist die Entwicklung eines Spieles, welches in einem mittelalterlichen Setting angesiedelt ist. Dabei werden Elemente aus Strategiespielen sowie aus Rollenspielen und First-Person Spielen bewusst miteinander vermischt, um ein neuartiges Spielerlebnis zu ermöglichen. Da das Spiel eher ein Sandbox-Spiel ist, gibt es kein wirklich festes Endziel, der Spieler kann im Prinzip so lange an seiner Stadt bauen, wie ihm beliebt. Jedoch gibt es auch Gegner, welche versuchen, den Spieler anzugreifen. Sind alle Spieler gestorben, so ist das Spiel vorbei.

Der Spieler hat dabei die Möglichkeit, wie in einem klassischen Aufbau-Strategiespiel Gebäude im Austausch gegen Ressourcen zu bauen. Auch kann er eine oder mehrere Personen auswählen und ihnen Befehle geben, wie Beispielsweise Bewegungsbefehle, Angriffsziele oder auch Ressourcen, welche abgebaut werden sollen. Darüber hinaus gehen die Personen auch ihrer eigenen Arbeit nach, das heißt, sie verteidigen sich z.B. automatisch bei Angriffen oder teilweise führen sie jobs automatisch aus. Darüber hinaus gibt es in diesem Spiel noch die Möglichkeit, die Kontrolle über eine einzelne Person direkt zu übernehmen um diese aus der First-Person Perspektive selbst zu steuern. Dabei kann man die Person sowohl bewegen als auch mit Objekten interagieren, Ressourcen abbauen oder Gegner angreifen. Bei der Bevölkerung gibt es im Grunde genommen 2 verschiedene Gruppen: Zivilisten und Militäreinheiten. Während Militäreinheiten nur für den Kampf und die Verteidigung geeignet sind, erledigen Zivilisten verschiedene Aufgaben. Dabei gibt es z.B. den Holzfäller, dieser kann mit seiner Axt Bäume zur Holzgewinnung fällen. Minenarbeiter hingegen können mit ihrer Spitzhacke Steine abbauen und Eisen aus Felsen gewinnen. Baumeister hingegen sind in der Lage, das Gelände zu verändern, um Beispielsweise Gräben zu errichten oder Bauplätze zu erschaffen. Farmer wiederum dienen dazu, Nahrung, welche von Farmen produziert wird, zu ernten.

Bei den Gegnern handelt es sich um einfache Banditen, welche die Spieler angreifen, sobald diese in deren Nähe gelangen. Darüber hinaus verfügen deren Festungen über Katapulte, welche auf Spieler in der Nähe schießen.

## Grundlagen

Das Spiel wurde mithilfe der Unity-Engine erstellt. Dabei wurden (fast) alle Scripte und Modelle selbst entwickelt.

# Aufgabenaufteilung (Game Production)

Nicolas Schön

Überwiegend Entwicklung und Programmierung der Grundlegenden Spielmechanik sowie der KI

- *Wechsel zwischen Tabletop und FPS*
- *Levelgenerator/Mesh-Erzeugung*
- *Level Manipulation*
- *Platzierungssystem von Gebäuden*
- *Auswählen von Personen und senden von Befehlen*
- *Wegfindung( $a^*$ )*
- *Schiessen des Katapultes*
- *Interaktionssystem für FPV-Personen*
- *Rekrutierung von neuen Personen*
- *Ressourcenerzeugung (Förster und Farm sowie Mine)*
- *Jobsystem*
- *Speichern und Laden von Leveln*
- *Spieleinstellungen*
- *Gegnerverhalten/Kampfsystem*

Modellierung von einzelnen Gebäuden und Naturobjekten

## Marcel Kohnle

Hauptsächlich die Gestaltung des Spiels und Erstellung der meisten Modelle.

- *Modellierung der Spielobjekte*
  - *Charaktere*
  - *Waffen und Werkzeuge*
  - *Gebäude*
  - *Natur*
- *Erstellung von Animationscontrollern und Einfügen passender Animationen*
- *Erstellung der Grafiken und Sprites im User Interface*
- *Erstellung von Partikeleffekten wie z.B. Feuer, Rauch, Regen, usw.*
- *Tag-Nacht-Zyklus mit Uhr, Tageszähler und Wetter*
- *Shader für Wasser und Grid*
- *Schießen mit Pfeil und Bogen*
- *Hit-Detection im FPS Modus*
- *Szene für Kampagnenmodus*
- *Post-Processing*

## Entwicklungsschritte/Implementierung (Game Programming)

Aufgrund des großen Umfangs dieses Spieles werden anschließend jeweils die Entwicklung und Funktionsweise der wichtigsten Funktionen erklärt.

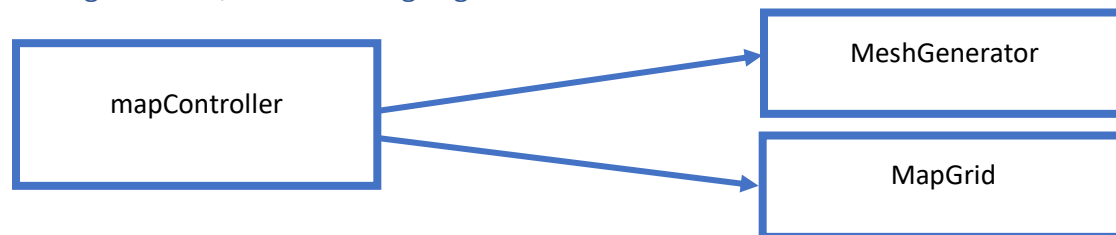
### Wechsel zwischen Tabletop und FPS

Um sowohl einzelne Personen direkt als auch die gesamte Bevölkerung steuern zu können, ist es notwendig, die Steuerung bzw. die Kamera umzuschalten. Beim Wechsel von der Tabletop Kamera hin zu der FPV-Kamera muss daher die Hauptkamera deaktiviert werden, um dann anschließend die Spielerkamera zu aktivieren. Außerdem wird dabei das Attribut „isControlled“ auf true gesetzt, sodass die Tastatureingaben und die Mausbewegung für die Steuerung nun auf den ausgewählten Charakter übertragen werden. Beim Wechsel zu einem Charakter wird außerdem das Jobsystem für den Zeitraum der direkten Kontrolle deaktiviert, sodass der Spieler die Aktionen der Person manuell durchführen kann. Der Wechsel zwischen der Tabletop Kamera und der Spielerkamera wird in den Scripten „cameraController“ und „personController“ behandelt.



Ausgewählte Person, mit ‚C‘ kann die Kontrolle übernommen werden

### Levelgenerator/Mesh-Erzeugung



Struktur des Mapcontrollers mit seinen Objekten ‚MeshGenerator‘ und ‚MapGrid‘

Bei der Mapperzeugung müssen verschiedene Aspekte betrachtet werden: zum einen wird das Mesh generiert, welches das Objekt ist, welches im Spiel dargestellt wird. Um jedoch die Platzierung von Naturobjekten wie Bäumen oder Felsen und auch das Bauen von Gebäuden zu ermöglichen, wird zusätzlich noch ein 2-Dimensionales Array angelegt, welches die Information beinhaltet, welches Objekt an der jeweiligen Stelle platziert ist. Dabei steht ein Wert von 0 bspw. für ein unbelegtes Feld, 1 steht für einen Baum, usw. .

Die statische Klasse mapController stellt dabei eine Schnittstelle zu anderen Objekten innerhalb von Unity dar, um bspw. Gebäude zu platzieren oder das Terrain an einer bestimmten Stelle zu verändern.

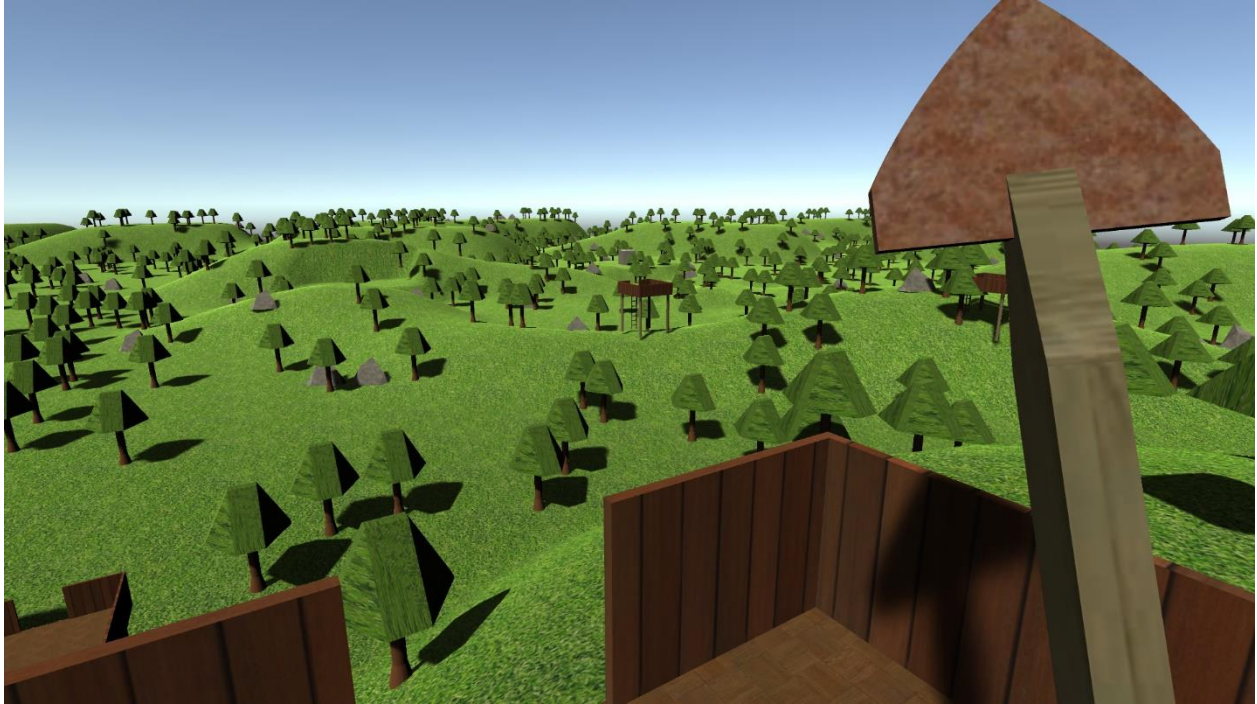
Sie besitzt Objekte vom Typ Meshgenerator und MapGrid.

In der Klasse Meshgenerator wird die Erzeugung und Speicherung des Meshes behandelt, aber auch Veränderungen des Meshes wie bspw. Das graben eines Loches oder das Aufschütten von Hügeln. Bei der Erzeugung des Meshes wird im Prinzip eine Vertices-Liste der Länge ,size' \* ,size' angelegt, wobei ,size' die größe der Spielwelt ist. Diese kann in den Einstellungen angepasst werden, sodass Spielwelten von 64x64 bis 255x255 möglich sind. Um die Welt nun interessanter zu gestalten, wird anschliessend die Höhe bzw. die Y-Position der einzelnen Vertices ermittelt. Dazu wird die Funktion PerlinNoise der Math-Bibliothek verwendet, um zufällige Werte zu ermitteln. Anschliessend werden diese Höhenwerte noch mithilfe von verschiedenen Sinusfunktionen so angepasst, dass die Höhe dem Aussehen von realen Landschaften entspricht. Nach erzeugen des Map-Meshes wird dann noch das Mesh für den Strand erzeugt, welcher die Begrenzung für die Spielwelt darstellt. Dabei werden die Höhendaten der Ränder des Mapmeshes verwendet, um einen fließenden Übergang zu gewährleisten.

Die Klasse MapGrid belegt bei erzeugen einer neuen Karte die Felder anschliessend mit Startwerten, um Bäume, Felsen, Eisenberge und gegnerische Camps und Festungen zu platzieren. Ausserdem wird diese Klasse immer von MapController aufgerufen, wenn die Belegung der Karte abgefragt oder verändert werden soll. Dies ist der Fall, wenn bspw. Ein neues Gebäude gebaut werden soll oder Objekte wie zum Beispiel Bäume oder Steine abgebaut werden und daher aus dem Map-Array entfernt werden müssen, um die Position als frei zu kennzeichnen, damit an dieser Stelle wieder ein neues Gebäude platziert werden kann.

Nach erzeugen Des Meshes und der Belegung der Map im Mapgrid werden die Gegenstände dann noch platziert, wobei mithilfe einer Schleife durch das Mapgrid-Feld gelaufen wird und an der jeweiligen Stelle dann das Prefab des jeweiligen Objektes Instanziiert wird.





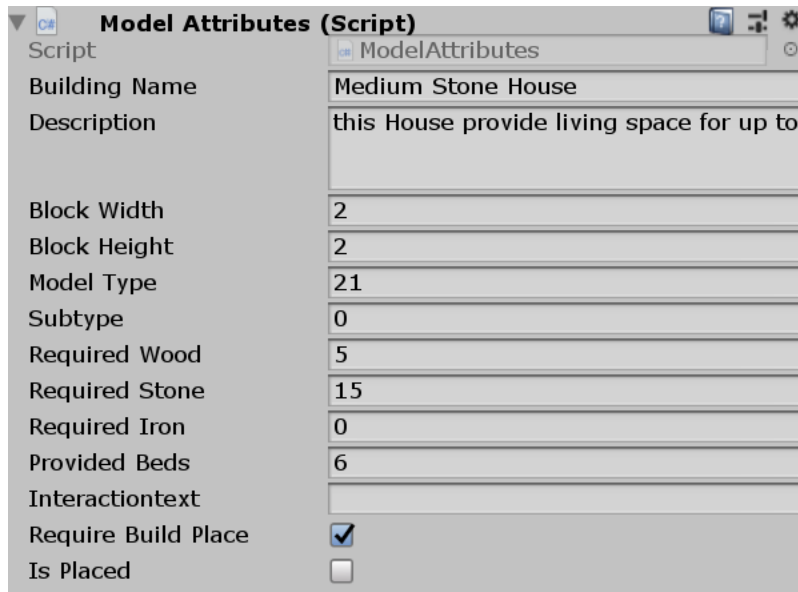
Erzeugtes Map-Mesh mit platzierten Gegenständen

### Level Manipulation

Die Map kann wie bereits erwähnt im Laufe des Spieles modifiziert werden. Dazu gehört zum Beispiel das Erzeugen eines Bauplatzes, wobei ein Bereich in der Größe des zu platzierenden Objektes geebnet wird. Dabei wird der Durchschnittswert der Höhe für die Felder berechnet, welche vom Gebäude belegt werden, und anschließend wird dann die Höhe dieser Felder auf die Durchschnittshöhe gesetzt. Aber auch Personen können die Map verändern, so kann bspw. der Baumeister mit seiner Schaufel Löcher graben bzw. Haufen aufschütten. Dabei wird dann an der Position im Mesh, welche vom Spieler angeklickt wird, die Höhe der Vertices rund um das Feld erhöht bzw. verringert. Auch Kugeln des Katapultes können die Landschaft verändern, wenn sie beim Einschlag explodieren und einen Krater hinterlassen. Dies passiert analog wie beim Graben von Löchern.

Zur Level-Manipulation werden über den MapController Methoden des Mapmeshes aufgerufen.

## Platzierungssystem von Gebäuden



Model Attributes (Script)	
Script	ModelAttributes
Building Name	Medium Stone House
Description	this House provide living space for up to
Block Width	2
Block Height	2
Model Type	21
Subtype	0
Required Wood	5
Required Stone	15
Required Iron	0
Provided Beds	6
Interactiontext	
Require Build Place	<input checked="" type="checkbox"/>
Is Placed	<input type="checkbox"/>

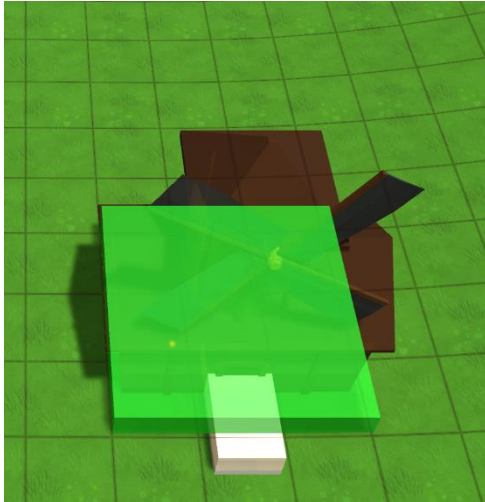
Attribute eines Gebäudes

Um Gebäude bauen zu Können, muss zunächst das Baumenü mit ‚B‘ aktiviert werden. Das Baumenü wird über ein weiteres Panel realisiert, welches je nachdem ob das Baumenü aktiv ist oder nicht, aktiviert oder deaktiviert wird. Die zu Bauenden Gebäude werden innerhalb eine Gameobject-Arrays gespeichert, welches die Prefabs der einzelnen Gebäude enthält. Mit den 2 Buttons kann dann der aktuelle Index des PlacableBuilding-Arrays verändert werden, sodass immer nur ein Objekt zur selben Zeit ausgewählt ist.

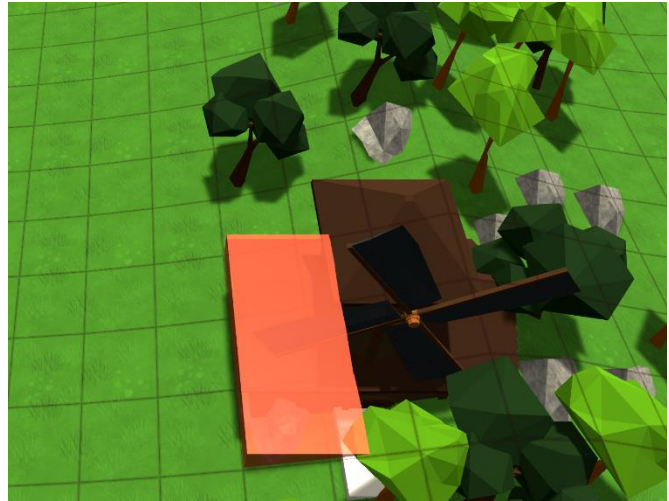
Um visuell darzustellen, ob das Gebäude an der gewünschten Stelle gebaut werden kann oder nicht, wird ein ‚placement Indicator‘ genanntes Gameobject verwendet, welches im Prinzip nur ein Würfel ist. Dieses wird im Baumodus aktiviert und das Material wird entsprechend verändert, je nachdem ob das Spielfeld an der Stelle, auf die die Maus zeigt, bereits belegt ist. Dabei gibt es 3 verschiedene Farben: Grün bedeutet, dass das Gebäude an der Stelle gebaut werden kann, Orange bedeutet, dass zwar der Platz vorhanden ist, jedoch die Ressourcen fehlen, und Rot bedeutet, dass bereits Objekte den benötigten Platz ausfüllen. Die Position des PlacementIndicators und des zu platzierenden Gebäudes wird dann an die Stelle der Map gesetzt, auf die die Maus zeigt. Dabei wird noch der Offset berechnet, welcher sich aus der MapGrid-Position und der Breite und Höhe des Gebäudes ergibt, sodass sich das Gebäude immer in einem oder der Grösse entsprechend mehreren Feldern befindet.

Wenn nun das Gebäude per Linksklick platziert wird, wird zunächst abgefragt, ob das Platzieren an dieser Stelle überhaupt möglich ist. Wenn dies der Fall ist, wird anschliessend überprüft, ob das Gebäude einen Bauplatz benötigt. Wenn ja, wird im MapMesh zusätzlich noch das entsprechende Gebiet geebnet. Anschliessend wird das Map-Array in der MapGrid-Klasse verändert, indem an der Wurzelposition (links unten) der Wert des Gebäudes abgespeichert wird. Gegebenenfalls, wenn das Gebäude grösser ist als 1x1, werden die restlichen Felder mit dem Platzhalterwert 9 belegt, um anzuzeigen, dass auch diese Felder belegt sind.

Das Zerstören bzw. Abbauen von Gebäuden läuft ähnlich ab, nur werden in diesem Fall die belegten Felder im MapGrid wieder auf 0 gesetzt, bevor das Gameobject des Gebäudes zerstört wird.



Positiver Placement-Indicator



Negativer Placement-Indicator



Baumenü

## Auswählen von Personen und senden von Befehlen

Personen können im Tabletopmodus per Linksklick ausgewählt werden. Dabei wird diese Person einer Liste hinzugefügt. Bei der ausgewählten Person wird dann auch eine Select-Methode aufgerufen, welche den Selection-Indicator aktiviert, um visuell anzuzeigen, welche Personen gerade ausgewählt sind. Per STRG + Ziehen der linken Maustaste können auch mehrere Personen ausgewählt werden, in diesem Fall werden dann mehrere Objekte der Liste hinzugefügt. Mit einem Linksklick auf den Boden wird die Auswahl wieder gelöscht, das heisst, bei den Personen-Objekten wird wieder die Select-Methode aufgerufen, um den Selection-Indicator wieder zu deaktivieren. Anschliessend wird die Liste geleert.

Wenn man nun mit der rechten Maustaste auf den Boden klickt, wird bei allen Objekten der Liste der ausgewählten Personen die Methode „setPosition“ mit der Position des Klicks auf der Map aufgerufen. Dabei wird dann der Pfad mithilfe des A\*-Algorithmus berechnet, damit die Personen auch Gebäuden und anderen Objekten ausweichen. Anschliessend bewegen sich die Personen dann zu dem gewählten Punkt.

Bei einem Rechtsklick auf Gegner wird der jeweilige Gegner für die ausgewählten Militäreinheiten als Ziel gesetzt. Mehr dazu findet sich im Kapitel „Gegner/Kampfsystem“.

Im Zerstörungsmodus können sowohl Gebäude direkt zerstört werden, als auch Ressourcen zum Abbau ausgewählt werden. In letzterem Fall wird die ausgewählte Ressource einer globalen Jobliste hinzugefügt, aus der dann die Personen mit dem jeweiligen Beruf einen Job auswählen und selbstständig bearbeiten (siehe Jobsystem).



Ausgewählte Personen



## Wegfindung(a\*)

Damit Personen nicht an Hindernissen hängen bleiben und auch immer den kürzesten Weg zum Ziel finden, wurde zur Wegfindung der A\*-Algorithmus verwendet. Dabei wird die Ausgangsposition der Person und die gewünschte Zielposition benötigt. Anschliessend wird der Algorithmus auf dem MapGrid-Array angewendet, um zu erkennen, ob die benachbarten Felder passierbar sind oder nicht, also ob z.B. ein Gebäude platziert ist und daher den Weg versperrt. Auch wird der Typ des Gegenstandes als Gewichtung mit in die Berechnung des Weges miteinbezogen, sodass z.B. ein Leeres Feld nur Kosten von 1 hat, während ein Feld mit einem Baum 2 und ein Feld mit einem Felsen 3 kostet. Der Algorithmus gibt dann eine Liste mit Positionsvektoren zurück, die dann vom Charakter abgearbeitet wird. Dabei wird überprüft ob die Person bereits in der Nähe des nächsten Wegpunktes ist, wenn ja, wird der nächste Navigationspunkt aus der Liste entnommen und angesteuert. Sobald die Navigationsliste leer ist, hat die Person das Ziel erreicht. Die Berechnung des kürzesten Weges mithilfe des A\*-Algorithmus sowie die eigentliche Bewegung der Person wurden mithilfe des „characterMovement“-Scriptes realisiert.



Hervorgehobene Wegpunkte eines ermittelten Pfades

## Schiessen des Katapultes

Das Verhalten des Katapultes wird im „catapultController“ definiert. Um den Spieler zu bestimmen, auf den geschossen werden soll, besitzt der Katapult einen Trigger-Collider, welcher bei Betreten eines Spielers den Spieler einer internen Liste hinzufügt. Die Spieler werden beim verlassen des Trigger-Colliders auch wieder aus dieser Liste entfernt, sodass ein weiteres Ziel ausgewählt werden kann, welches sich noch innerhalb der Reichweite, also innerhalb des Trigger-Colliders befindet. Soll nun geschossen werden, wird das erste Element dieser Spielerliste ausgewählt. Anschliessend wird die voraussichtliche Flugbahn berechnet, um die Kugel mit einer bestimmten Geschwindigkeit in Richtung des Spielers abzuschiessen. Die Kugel an sich besitzt ihrerseits einen normalen Collider, sobald dieser mit einem anderen Gegenstand kollidiert, wird überprüft, welchen Typs dieser Gegenstand ist. Sollte dieser Gegenstand ein Spieler sein, wird dem Spieler daraufhin Schaden hinzugefügt. Ausserdem kann die Kugel beim Aufschlag auf den Boden auch einen Krater hinterlassen, dazu wird wie beim Ändern des Terrains durch einen Baumeister auch wieder die entsprechende Methode des Mapcontrollers aufgerufen. Desweiteren verursacht eine Explosivkugel Schaden bei Spielern in der Umgebung, und sie wendet auch eine Kraft auf diese Objekte an, sodass diese Objekte von der Druckwelle weggeschleudert werden. Das Verhalten der Kugel wird im Script „bulletController“ umgesetzt.



Katapult

## Interaktionssystem für FPV-Personen

Damit man mit Personen im First Person-Modus mit Objekten wie bspw. Türmen, Metzgern oder Schmieden interagieren kann, ist ein Interaktionssystem nötig. Um dies zu erreichen, wird jeder Person ein TriggerCollider angehängt. Sobald ein Element, welches interagierbar ist, den Collider betritt, wird es einer internen Liste hinzugefügt. Bei Verlassen des Colliders wird dieses Objekt auch wieder aus dieser Liste entfernt. Wenn nun die Taste ‚E‘ gedrückt wird, um mit dem nächsten Objekt zu interagieren, wird das erste Element aus dieser Liste ausgewählt und die Methode „interactWithObject“ auf diesem aufgerufen. Dies wird im Script „personInteraction“ umgesetzt.



Interaction mit einem Turm

## Rekrutierung von neuen Personen

Man kann mittels ‚R‘ das Rekrutierungsmenü aufrufen, um Zivilisten verschiedenen Berufes zu rekrutieren. Dabei wird eine bestimmte Menge an Essen benötigt. Möchte man Militäreinheiten rekrutieren, benötigt man zunächst eine Burg. Wenn man nun mit der rechten Maustaste auf diese klickt, wird das Militär-Rekrutierungsmenü aufgerufen. Dort kann man dann wie im normalen Rekrutierungsmenü auch den Typ auswählen, den man rekrutieren möchte. Jedoch ist zu beachten, dass bestimmte Einheiten zusätzlich zu Essen auch noch Eisen benötigen.

Das Rekrutierungsmenü wird im Grunde genommen mittels des Scriptes „populationManagement“ realisiert, welches überprüft, ob überhaupt genug Ressourcen vorhanden sind, und im positiven Fall dann eine neue Person erzeugt. Die Methoden dieses Scriptes werden über die Buttons im Rekrutierungspanel aufgerufen, welches sich im ‚CanvasTabletop‘, dem GUI-Objekt, befindet.

Die neu erzeugten Personen werden dann an der aktuellen Position gespawnt.



Rekrutierungsmenü

### Ressourcenerzeugung (Förster und Farm sowie Mine)

Da das Erbauen einer Stadt viele Ressourcen benötigt, ist es wichtig, auch dafür zu sorgen, dass diese Ressourcen nachwachsen. Dabei gibt es 4 Ressourcen: Nahrung, Holz, Stein und Eisen. Nahrung kann nur vom Farmer geerntet werden.

Im Script „farmController“ werden neue Felder rund um die Farm erzeugt. Dabei wird zunächst wieder über den MapController überprüft, ob überhaupt Platz dafür ist. Anschliessend wird dann das Feld-Objekt platziert und die Methode „startGrow“ des Feldes aufgerufen. Diese Methode sorgt dafür, dass das Feld im Laufe der Zeit immer weiter wächst, bis es irgendwann seine finale Grösse erreicht hat und geerntet werden kann. Sobald dies geschehen ist, wird dieses Feld der Liste der zu erntenden Felder im JobController hinzugefügt.

Das Script „foresterControl“ ist ähnlich aufgebaut wie „farmController“, mit dem Unterschied, dass anstatt Feldern neue Bäume platziert werden. Auch bei diesen Bäumen wird wieder „startGrow“ aufgerufen, um dafür zu sorgen, dass auch diese mit der Zeit wachsen. Werden die Bäume gefällt, solange sie noch am Wachsen sind, bringen sie nur eine kleine Menge Holz, sodass es sinnvoll ist, zu warten bis sie voll ausgewachsen sind, um die maximal mögliche Menge an Holz von den Bäumen zu bekommen.



Neben diesen Gebäuden gibt es dann noch die Minen. Diese erzeugen entweder Stein oder Eisen. Für deren Abbau wird jedoch keine weitere Person benötigt, sondern die Minen erzeugen die Ressourcen von selbst. Das Verhalten der Minen wird im „mineController“ umgesetzt.



Farm mit erzeugtem Mais und Getreide



Försterhütte mit gepflanzten Bäumen

### Jobsystem

Damit jede Person in der Lage ist, Jobs anzunehmen, auszuführen und auch wieder abzugeben, war die Entwicklung eines globalen Jobsystems notwendig. Dieses wird mithilfe des Scriptes „jobController“ realisiert. Dort existieren Listen für Erntejobs, Holzfällerjobs und Miningjobs. Sobald entweder das Feld ausgewachsen ist oder der Spieler ein Objekt zum Abbau angeklickt hat, wird dieses Objekt der jeweiligen Jobliste hinzugefügt. Personen mit dem jeweiligen Job-Script fragen dann ab, ob die jeweilige Jobliste jobs enthält, und entnehmen im positiven Fall den Job mit dem Objekt, welches sich am nächsten von ihnen befindet. Durch das annehmen des Jobs wird dieser Job aus der globalen Jobliste entnommen, sodass nicht mehrere Personen am selben Job arbeiten. Wenn nun die Person ausgewählt und manuell irgendwo anders hinbewegt oder die Kontrolle direkt übernommen wird, gibt die Person ihren aktuellen Job wieder in die Jobliste ab, sodass evtl. andere Charaktere, welche zurzeit nicht vom Spieler direkt gesteuert werden, diesen Job übernehmen können. Personen führen indessen Jobs nur aus oder nehmen neue Jobs an, solange sie nicht direkt vom Spieler via First Person gesteuert werden.



Holzfäller bei der Arbeit



Farmer beim Ernten von Feldern

### Speichern und Laden von Leveln

Damit Spieler in der Lage sind, auch größere Städte zu errichten, ist es notwendig, dass der Spielstand auch abgespeichert werden kann. Um auch mehrere verschiedene Städte zu ermöglichen, wurde entschieden, verschiedene Spielstände zu erstellen bzw. zu verwenden, welche vom Spieler selbst erstellt werden können. Dabei ist wichtig, alle Relevanten Informationen wie bspw. Die Topografie der zufällig erzeugten Spielwelt, als auch die Belegung durch Objekte festzuhalten. Des Weiteren sollen auch die Positionen und Typen der Spieler-Personen sowie Gegner gespeichert werden. Um dies umzusetzen, wurde das „savegameController“-Script erstellt, welches sowohl das Speichern der Objekte als auch das Laden aus verschiedenen Textdateien ermöglicht.

Um die Objekte zu speichern, werden nur die Relevanten Informationen wie Positionsdaten sowie Rotationsdaten und noch die Typen und evtl. (bei Personen) die aktuelle Gesundheit in einer Textdatei im JSON-Format abgespeichert. Aus diesen Informationen kann dann ein neues Objekt anhand des Prefabs an derselben Position mit derselben Rotation und den selben Attributen erzeugt werden. Dies ist notwendig, um den Speicherplatz möglichst gering zu halten. So ist bspw. Ein Spielstand einer Welt der Größe 100x100 mit ca 3000 Objekten und noch ein par Dutzend Personen gerade einmal ca. 2MB groß. Um die Objekte möglichst Logisch voneinander zu trennen, befinden sich die Informationen der Map wie Anzahl der Vertices, Tris, sowie die Belegung der Karte in der Datei ‚map.save‘. Auch generelle Informationen wie gesammelte Ressourcen befinden sich in dieser Datei. Informationen über Gebäude



wie Position, Rotation sowie name, typ, subtyp(bspw. für unterschiedliche Bäume) und Breite und Höhe befinden sich in ‚buildings.save‘. Informationen über Personen werden in ‚persons.save‘ abgespeichert, und Informationen über Gegner in ‚enemys.save‘. Diese 4 Dateien werden in einem neu erzeugten Ordner abgelegt, welcher den Namen des Spielstandes trägt. Die Ordner der Spielstände werden in dem Ordner ‚saves‘ angelegt, welcher sich unter dem Generellen Data-Ordner befindet.



Speicher-Bildschirm

Möchte man nun einen zuvor gespeicherten Spielstand Laden, so kann man diesen über das Hauptmenü im Laden-Bereich auswählen. Dabei wird eine Liste von den Namen aller Ordner innerhalb des save-Ordners angelegt und deren Namen als Button grafisch dargestellt. Klickt man nun auf einen Spielstand, so wird die Map aus den zuvor abgespeicherten Daten rekonstruiert sowie die Gebäude, Personen und Gegner platziert.



Lademenü

Aufbau eines Spielstandes:

- saves
  - Ordner(Name des Spielstandes)
    - buildings.save
    - enemys.save
    - map.save
    - persons.save

```
{
  "Items": [
    {
      "name": "tree2(Clone)",
      "type": 1,
      "subtype": 0,
      "width": 1,
      "height": 1,
      "posx": 41.5,
      "posy": 0.28050893545150759,
      "posz": 0.5,
      "rotx": 0.0,
      "roty": 46.0,
      "rotz": 0.0
    },
    {
      "name": "tree2(Clone)",
      "type": 1,
      "subtype": 0,
      "width": 1,
```

Auszug einer ‚buildings.save‘ Datei

## Spieleinstellungen

Um Einstellungen des Spieles zu verändern, gibt es einen speziellen Settings-Bereich im Hauptmenü. Dort können Einstellungen wie die Generelle Lautstärke, der Spielmodus (Peaceful, wobei ohne Gegner gespielt wird, und normal, wobei auch gegnerische Camps und Festungen spawnen), die Mapgröße, wobei diese Werte von 64 – 255 annehmen kann, sowie die Schwierigkeit festgelegt werden. Bei der Schwierigkeit gibt es 3 verschiedene Stufen: easy, normal und hard, wobei jeweils unterschiedlich viele Startressourcen und Startpersonen vorhanden sind. Dadurch ist es bspw. in der Schwierigkeit ‚hard‘ zunächst unmöglich, Gebäude zu bauen, da zunächst Ressourcen beschafft werden müssen. Auch gibt es in dieser Schwierigkeit zunächst noch nicht Personen jedes Berufes, diese müssen erst Rekrutiert werden.

Die Schwierigkeitseinstellungen wurden mithilfe einer globalen Statischen Klasse realisiert, welche jeweils diese Attribute festhalten. Dadurch kann auch aus verschiedenen Szenen auf diese Einstellungen zugegriffen werden.



Einstellungs-Menü

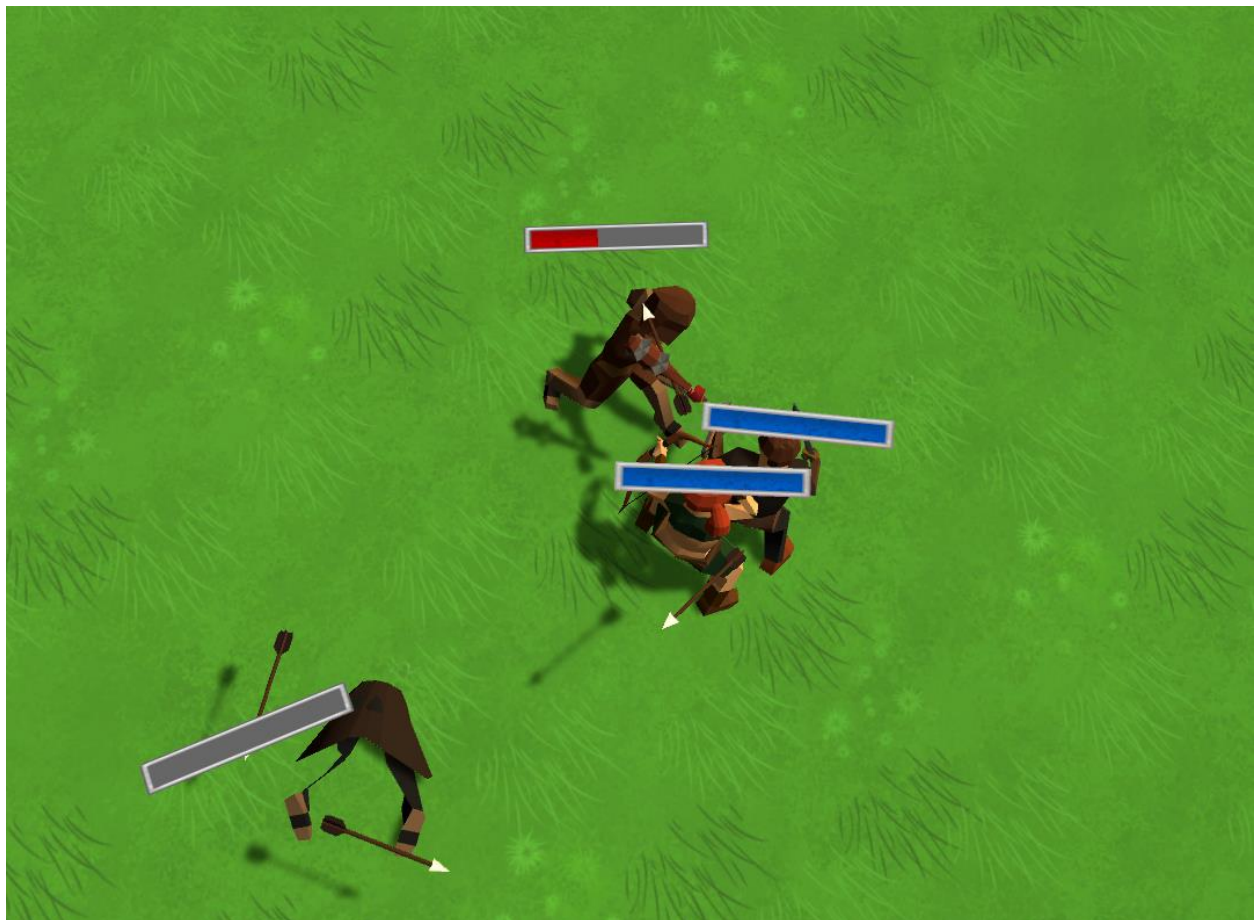
## Gegnerverhalten/Kampfsystem

Das Kampfsystem stellte zunächst eine größere Herausforderung dar, da sowohl aus Tabletopansicht als auch per First-Person View Kämpfe ausgefochten werden können sollen. Dabei wurden Entscheidungsbäume für das Verhalten der Gegner sowie der Spieler verwendet. So wird bspw. bei den Gegnern überprüft, welches der aktuell nächste Spieler ist, dann wird die Entfernung zu diesem berechnet. Befindet sich dieser in Sichtweite bzw. in der Nähe des Gegners, so läuft dieser Gegner dann auf diesen Spieler zu. Sobald sich der Gegner innerhalb der Angriffsreichweite befindet, wird die Angriffsanimation ausgeführt, und über das Event, welches dieser Animation angefügt ist, wird

überprüft, ob der Schlag den Spieler getroffen hat. Bei einem Treffer wird dann dem Spieler die entsprechende Gesundheit abgezogen, verfügt der Spieler zusätzlich noch über eine Rüstung, wird zunächst der Rüstungswert abgezogen, bevor die Gesundheit in Mitleidenschaft gezogen wird. Entweicht der Spieler der Reichweite des Gegners, so läuft der Gegner wieder zurück zu seinem Camp oder seiner Festung.

Der Spieler verteidigt sich standardmäßig nur, wobei er den Gegner nur automatisch angreift, sobald dieser in unmittelbarer Nähe von diesem ist. Er greift die Gegner nur an, wenn der Spieler die Gegner mithilfe der rechten Maustaste zum Angriff anklickt.

Wechselt der Spieler in die First-Person Ansicht, so wird die komplette automatische Kontrolle des Spielers ausgeschaltet, sodass der Spieler die volle manuelle Kontrolle über den Kampf gewinnen kann. Dabei kann ein Schlag mit dem Schwert oder ein Schuss mit dem Pfeil und Bogen mit der linken Maustaste durchgeführt werden.



Kampf im Tabletop-Modus





Kampf im First-Person-Modus

## Modellierung der Spielobjekte

Da die Welt des Spiels zufallsgeneriert wird und eine große Anzahl an Objekten zur selben Zeit angezeigt werden sollen, haben wir uns dazu entschlossen alle Modelle in einem *Low-Poly-Style* zu modellieren.

Beispiele dieser Modelle:

### Charaktere



### Waffen und Werkzeuge



### Gebäude

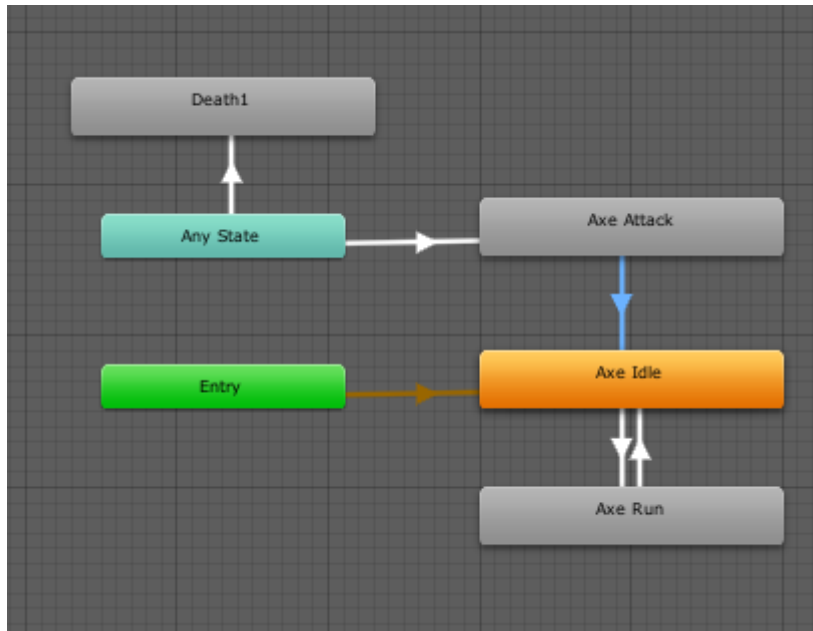


### Natur

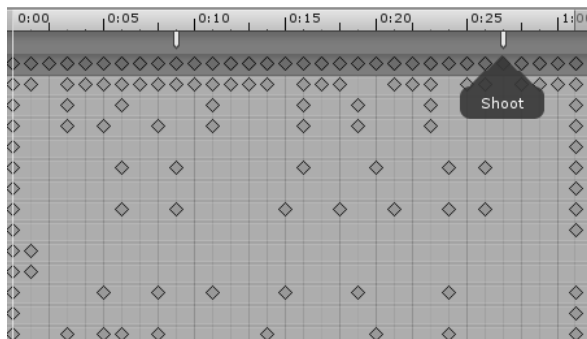


## Erstellung von Animationscontrollern und Einfügen passender Animationen

Fast jeder Charakter hat einen eigenen Animationscontroller mit individuellen Animationen



Events wie z.B. Axt trifft Baum oder Bogen schießt Pfeil werden innerhalb der Animation ausgeführt.



Erstellung der Grafiken und Sprites im User Interface

## Buttons



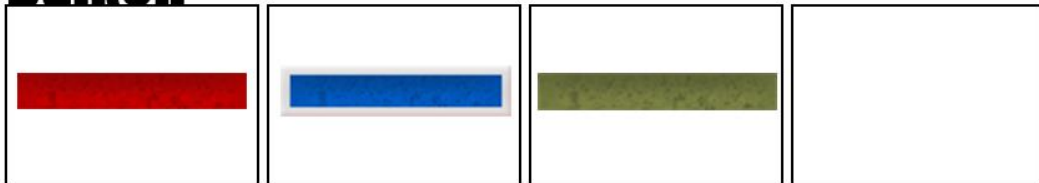
## Icons



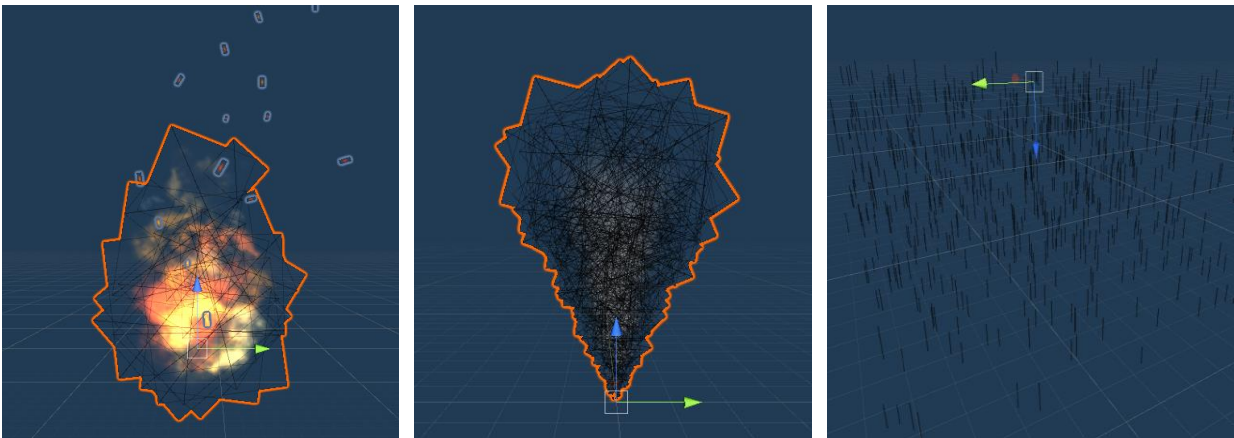
## Hintergründe



## Balken



Erstellung von Partikeleffekten wie z.B. Feuer, Rauch, Regen, usw.



## Tag-Nacht-Zyklus mit Uhr, Tageszähler und Wetter

Der Tag-Nacht-Zyklus wird erzielt indem das *Directional Light*, was als Sonne dient, im Skript **dayLight** um 360° auf der X-Achse rotiert wird. Da die Nacht kürzer sein soll, wird während einer Nacht 70° übersprungen, somit errechnet sich eine Stunde im Spiel folgendermaßen zu Sekunden:

$$(360 - 70) / 24 = 12,0833 \text{ s}$$

Somit dauert ein kompletter Tag im Spiel knapp 5 Minuten.

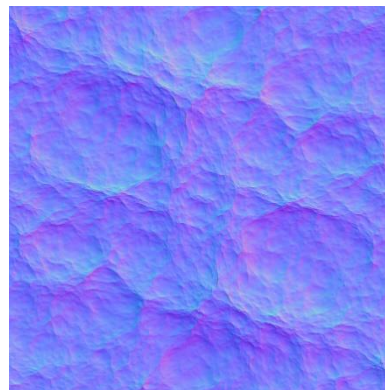
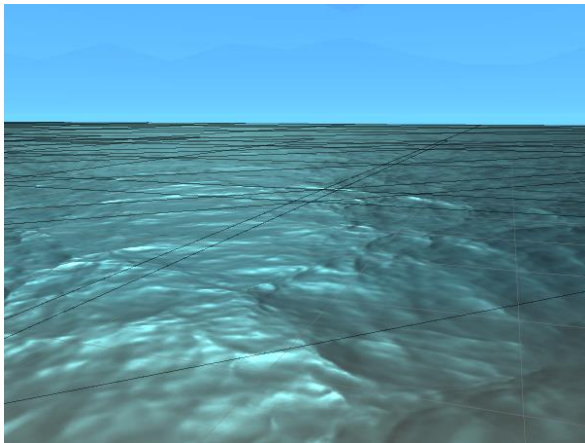
Jeden Tag um 6 Uhr besteht eine Chance, dass es regnet.



## Shader für Wasser und Grid

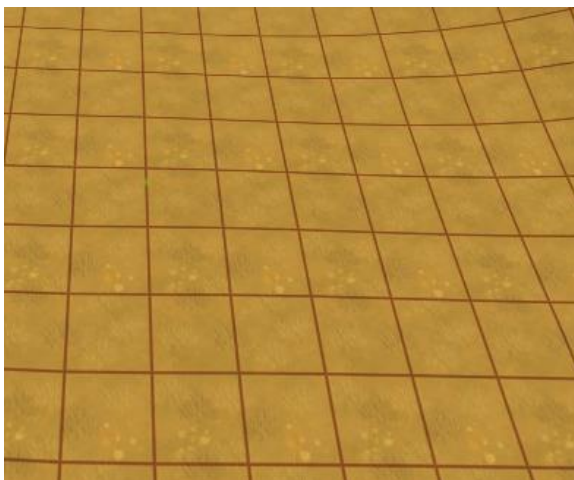
### Wasser

Das Wasser ist ein einfaches Plane-Objekt mit dem Wasser Material. Der Shader bewegt und rotiert die Normal-Map mit verschiedenen Geschwindigkeiten und Intensitäten über die Plane.



### Grid

Das Grid liegt zusätzlich zur Gras Textur auf dem ground-Objekt und wird mit Öffnen des Baumenüs ein und ausgeblendet.



## Schießen mit Pfeil und Bogen

Das Skript **ShootArrow** wird ausgeführt sobald das Event in der Bogenanimation ausgelöst wird. Hierbei wird das Pfeil-Prefab erzeugt und mit einer gewissen Stärke nach vorne geschleudert.

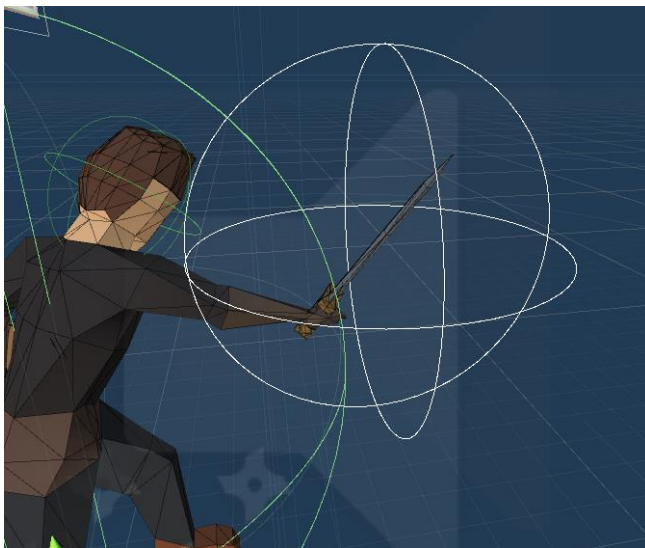
Der Pfeil wird eingefroren sobald er kollidiert und verschwindet nach 5 Sekunden.



## Hit-Detection im FPS Modus

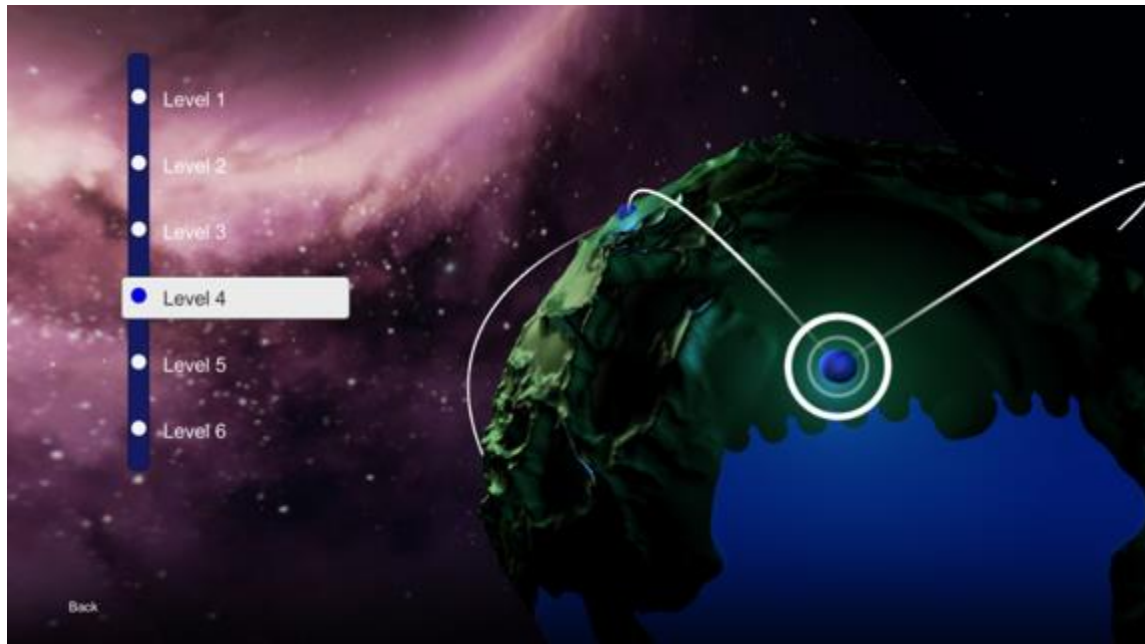
Um einen Gegner zu treffen oder einen Baum zu fällen muss abgefragt werden, ob sich das zu kollidierende Objekt innerhalb eines bestimmten Radius befindet.

Im Skript **MeleeHit** wird um einen bestimmten Punkt an der Waffe oder am Werkzeug ein Radius festgelegt in dem der Hit erkannt wird. Das HitEvent wird je nach Animation jeweils zu einem bestimmten Zeitpunkt ausgelöst.





## Szene für Kampagnenmodus



## Post-Processing

Für eine optische Aufwertung des Spiels wird das Unity Post-Processing Package verwendet. Hierbei werden *Color-Grading*, *Bloom*, *Ambient Occlusion*, *Tiefenschärfe*, *Bewegungsunschärfe* und *Vignette* eingestellt.

