Dokumentation-

Spielentwicklung in Unity

Tower Defense Ultimate

Jaron Liskow und Tim Fischer

Seminarkurs 11 Bindestrichinformatik

Inhalt

[Erste Anfänge 4](#_Toc191557679)

[Erster grober Entwurf des Projekts. 4](#_Toc191557680)

[Tim Vorkenntnisse: 5](#_Toc191557681)

[Jaron Vorkenntnisse: 5](#_Toc191557682)

[Ziele: 5](#_Toc191557683)

[Schritte (Planung & Arbeit): 6](#_Toc191557684)

[Entwicklungsumgebung: 6](#_Toc191557685)

[Versionsverwaltung/Zusammenarbeit: 6](#_Toc191557686)

[Erstes Ziel (Tim): Basisfunktion der Gegner 6](#_Toc191557687)

[Einarbeitung 6](#_Toc191557688)

[Vorgehensweise 7](#_Toc191557689)

[Was macht der Code? 8](#_Toc191557690)

[Erstes Ziel (Jaron): Third-Person-Controller 8](#_Toc191557691)

[Vorbereitung: 8](#_Toc191557692)

[Teil 1 - mein Skript: 8](#_Toc191557693)

[Teil 2 - alles neu 9](#_Toc191557694)

[Zweites Ziel (Tim): Schadensystem 10](#_Toc191557695)

[Teil 1 - Leben von Gegnern 10](#_Toc191557696)

[Teil 2 - Schaden an der Base 12](#_Toc191557697)

[Zweites Ziel (Jaron): Placement Shit 13](#_Toc191557698)

[Teil 1: Was muss ich überhaupt machen? -- Position ermitteln 13](#_Toc191557699)

[Teil 2: Türme auswählen können 15](#_Toc191557700)

[Teil 3: Nicht nur Platzieren sondern auch Bewegen (Uiii spannend) 16](#_Toc191557701)

[Teil 4: Reichweitenvisualisierung 17](#_Toc191557702)

[Drittes Ziel (Tim): Basisfunktion der Türme 19](#_Toc191557703)

[Vorgehensweise 19](#_Toc191557704)

[Was macht der Code? 20](#_Toc191557705)

[Viertes Ziel (Tim): Projektile der Türme 20](#_Toc191557706)

[Vorgehensweise 21](#_Toc191557707)

[Drittes Ziel (Jaron): UI 21](#_Toc191557708)

[Teil 1: Main Menü 21](#_Toc191557709)

[Teil 2: Spawnpanel 22](#_Toc191557710)

[Reflektion Q1 23](#_Toc191557711)

[Tim 23](#_Toc191557712)

[Stolpersteine und Lösung 23](#_Toc191557713)

[Jaron: 24](#_Toc191557714)

[Notizen Q1 25](#_Toc191557715)

[Probleme 25](#_Toc191557716)

[Tim: 25](#_Toc191557717)

[Q2 26](#_Toc191557718)

[Fünftes Ziel (Tim): Gruppiertes Spawning und Extras 26](#_Toc191557719)

[Viertes Ziel (Jaron): BigBabaPrefab 27](#_Toc191557720)

[Sechstes Ziel (Tim): Credits 27](#_Toc191557721)

[Fünftes Ziel (Jaron): Credit Bedingtes Spawning (UI) 28](#_Toc191557722)

[Sechstes Ziel (Jaron): Türme Upgraden 28](#_Toc191557723)

[Siebtes Ziel (Tim): Neue Türme & Neue Bots 28](#_Toc191557724)

[Anfänge mit Blender 28](#_Toc191557725)

[Projektil Skript entfernen 29](#_Toc191557726)

[UI-Anpassungen 29](#_Toc191557727)

[Schussanimation 30](#_Toc191557728)

[Achtes Ziel (Tim): Level Vorbereiten 30](#_Toc191557729)

[Das BotsOnPath Skript im Detail 32](#_Toc191557730)

[Erstes Gemeinsames Ziel (Jaron & Tim): Spawning überarbeiten 33](#_Toc191557731)

[Notizen 34](#_Toc191557732)

[Plan für Q2 34](#_Toc191557733)

[Erst in Q3 34](#_Toc191557734)

[Story 34](#_Toc191557735)

# Erste Anfänge

**Struktur**

* **Projektbeschreibung und Einleitung**
  + Vorstellung des Projekts und eigener Person.
  + Eigene Vorkenntnisse und Projektideen.

# Erster grober Entwurf des Projekts.

* **Ziele des Projekts** *//siehe Oben – Noch nicht strukturiert*
  + Aufteilung der Ziele in:
    - **Obligatorische Ziele** (unbedingt erforderlich)
    - **Optionale Ziele** (zusätzliche Erweiterungen)
  + Unterteilung der Ziele in:
    - **Grobziele** (allgemein formulierte Teilziele)
    - **Feinziele** (detaillierte Arbeitsschritte zum Erreichen der Grobziele)
* **Planung und Arbeitsschritte**
  + Aufgliederung in einzelne Schritte und Aufgabenbereiche. *//siehe oben*
  + Mögliche Checkliste zum Abhaken abgeschlossener Schritte.
* **Entwicklungsumgebung** 
  + Vorstellung der genutzten Tools, Programmiersprachen, und anderer Ressourcen.
  + Begründung der Wahl der Entwicklungsumgebung.
* **Fortschrittsdokumentation**
  + Beschreibung der regelmäßigen Fortschritte im Projekt.
  + Darstellung und Erklärung des verwendeten Codes.
  + Lösung aufkommender Probleme und Herausforderungen.
* **Reflexion der Projektarbeit**
  + Jedes Halbjahr Reflexion des eigenen Arbeitsprozesses.
    - Eigene Motivation und mögliche Schwankungen.
    - Herausforderungen und Hindernisse im Projektverlauf.
    - Vorgehensweise bei der Lösung von Problemen.
  + Zielerreichung und Zeitmanagement:
    - Welche Ziele wurden schneller oder langsamer als erwartet erreicht?
    - Gründe für Abweichungen im Zeitplan und den Erfolg der Erreichung

## Tim Vorkenntnisse:

Informatik Leistungskurs, Vorkenntnisse aus der 10. Klasse Informatik Schwerpunkt, etwas Erfahrung mit Java durch den Java – Hamster (Grobe Struktur der Sprache), Erfahrung mit Spielprogrammierung durch Snap.

## Jaron Vorkenntnisse:

Informatik Leistungskurs, Vorkenntnisse aus der 10. Klasse Informatik Schwerpunkt, etwas Erfahrung mit Java durch den Java – Hamster (Grobe Struktur der Sprache), Erfahrung mit Spielprogrammierung durch Snap, leichte Vorkenntnisse in Unity, Vorwissen in C++.

# Ziele:

* Obligatorisch:
  + 3rd Person (Jaron)
    - Spielfigur
      * Fähigkeiten für die Figur
        + Freeze Potion
        + Direktschaden
        + Münzbooster
        + Blockade
    - Steuerung der Figur mit Maus und WASD
    - Steuerung der Kamera durch Maus und Spielfigur
      * Mausempfindlichkeit
  + Das erste Level (Grundstruktur für In Game)
    - Funktionierendes Gesundheitssystem (Anzeige)
      * Base mit Lebenspunkten
      * Wenn die Base zerstört ist, ist das Spiel verloren
      * Gegner mit Lebenspunkten
      * Wenn ein Gegner stirbt, erhält der Spieler Münzen
    - Gegner Bewegen sich auf einem Pfad zur Base und fügen Schaden zu, wenn sie an der Base ankommen
    - Wellen von Gegnern und gruppiertes Spawning
    - Türme schießen auf Gegner und fügen denen Schaden zu
      * Reichweite der Türme
      * Projektile bewegen sich auf den nächsten Gegner zu
    - Platzieren und Kaufen der Türme durch Spieler
    - Türme können geupgradet werden
  + UI
    - Startbildschirm & „Endbildschirm“ (Game Over)
    - Anzeige der Leben, verfügbarer Münzen, Türme etc.
  + Animation
    - Passende Laufanimation von Gegnern
    - Schussanimation von Türmen
  + Weitere Level
    - Erweiterte UI
      * Levelauswahl
    - Verschiedene Gegner und Türme mit Unterschiedlicher Animation und unterschiedlichen Eigenschaften
      * Türme: Turm mit begrenztem Drehradius; Turm mit Flächenschaden; Turm mit Mindestradius; Turm mit Doppelschuss;
      * Gegner: Gegner mit Störsignal (Stört die Türme), Gegner mit Heilung (heilt umliegende Gegner)
    - Weitere Pfade in einem Level
  + Namen für Spiel
* Optional:
  + 3dimensionale Bewegung des Spielers
    - Hoch – runter Bewegung mit Maus oder Tastatur
  + Level mit 3dimensionaler Bewegung der Gegner und Platzierung der Türme

Ziele / Ideen, die nach Q2 zugefügt wurden 🡪 nicht mehr unbedingt zu erreichen

Erledigte „Q2 -Ziele“

Erledigte Ziele

# Schritte (Planung & Arbeit):

Entwicklungsumgebung:

Wir haben uns dafür entschieden, eine Game Engine zu benutzen, die viele Tools bereitstellt, weil es somit einfacher ist, komplexe Spiele zu programmieren. Dabei haben wir uns zunächst für Unreal Engine entschieden, haben jedoch relativ schnell zu Unity gewechselt, weil Unity bekannter ist, es dadurch mehr Tutorials gibt und dadurch anfängerfreundlicher ist. Mit Unity programmieren wir in C# mithilfe von Visual Studio (2022). - Tim

## Versionsverwaltung/Zusammenarbeit:

Um zu ermöglichen, dass Tim und ich beide an unseren PCs am Spiel arbeiten können und nicht immer einen USB-Stick oder so tauschen müssen, habe ich mich entschieden, Github für die Versionsverwaltung zu nutzen. Also habe ich ein Public Repository erstellt und Tim als Mitbearbeiter eingeladen. Im Folgenden habe ich ein Projekt in Unity erstellt und die kompletten Projektdateien in das Repository hinzugefügt. Danach musste Tim nur noch die Dateien herunterladen und das Projekt in Unity öffnen. Ich habe einfach ein Main Branch erstellt, um alles so simpel wie möglich zu halten. Jetzt können wir immer problemlos am der neusten Version des Spiels arbeiten. - Jaron

## Erstes Ziel (Tim): Basisfunktion der Gegner

### Einarbeitung

Meine erste Aufgabe bestand darin, die erste Funktion für unsere Gegner zu programmieren, sie sollten also zunächst in einem bestimmten Intervall an einer bestimmten Position geklont werden und von dort aus ein Ziel verfolgen. Um anzufangen musste ich mich vorerst in Unity einarbeiten. Dafür habe ich erste Dinge in einem anderen Projekt getestet und mich mit der Umgebung vertraut gemacht. Geholfen haben mir dabei YouTube – Tutorials und ChatGPT. Gestellte Prompts waren zum Beispiel *„Wie Programmiere ich in Unity?“* oder *„Wo finde ich den Unity Editor?“* auch erste Skriptideen, wie: *„ich will, dass eine Kugel sich im Spiel hin und her bewegt“.* Ich habe recht schnell festgestellt, dass das Erstellen von einfachen Skripts recht schnell mit ChatGPT geht und wusste, dass ich weiterhin viel mit ChatGPT arbeiten werde. Da jedoch die Skripts von ChatGPT nicht immer das tun, was man erreichen will, war ich nach kurzer Zeit doch dazu gezwungen, zu verstehen, was der Code von ChatGPT macht und musste mich mit den Befehlen in den Skripten auseinandersetzen. Dazu habe ich ChatGPT Fragen gestellt wie: *„Was bedeutet 3f“* (Das f stellte sich als ein Verweis auf den Datentyp float heraus) oder *„was macht public und was macht private?“.* Ich habe zuerst Stunden damit verbracht, einfache Befehle zu verstehen und anwenden zu können. Schließlich habe ich mich daran getan, zum Einstieg das Sonnensystem zu simulieren. Wichtige Befehl Typen dafür waren: „transform“ und „Mathf“

### Vorgehensweise

Um schließlich mit meiner Aufgabe zu beginnen habe ich in einer neuen Szene („PathEditor“) einen Zylinder namens Spawner erstellt, an dem die Gegner starten sollen, einen Quader namens Base2 an dem die Gegner schlussendlich ankommen sollten, und zwei weitere Objekte (einen Zylinder, Bot1 und eine Kugel, Bot2) erstellt. Nun sollten die bots, die in bestimmten Abständen geklont werden, von Spawner zur Base gelangen. Umgesetzt habe ich dies mit einem Skript, das ich mithilfe von ChatGPT und folgendem Video: [6 Ways to move object in Unity [Tutorial] - YouTube](https://www.youtube.com/watch?v=-thhMXmTM7Q&ab_channel=FeriFm) erstellt hab. Dieses Skript heißt BotsOnPath und wird auf den Spawner angewandt. Zuerst wollte ich das Spawnen der Bots hinbekommen und habe ChatGPT folgenden Prompt gegeben: *ich will einen Zylinder erstellen, der an einem bestimmten Punkt spawned, 8 Schritte nach vorne geht, verschwindet und am Ausgangspunkt wieder ein neuer Zylinder spawned, der wieder 8 Schritte nach vorn geht usw.* Er hat mir erklärt, dass ich die Bots zu Prefabs machen muss und mir den Code für den Prompt gegeben, bei dem wieder mehrere Befehle, von denen ich noch nie etwas gehört habe verwendet wurden. Zum Beispiel „Quanternion“ und „StartCoroutine“. Nachdem ich den Code verstanden habe wollt ich wissen, *„wie ich […] ein Objekt zu einem anderen Objekt laufen lassen?“* kann, um mein Ziel zu erreichen (von Spawner zu Base). Das hab ich jedoch nicht weiter verwendet, weil ich stattdessen auf ein Tool in dem Assetstore von Unity gestoßen bin, mit dem man kurvige Pfade in der Unity Scene erstellen kann, den Bézier Path Creator (siehe [Bézier Path Creator | Utilities Tools | Unity Asset Store](https://assetstore.unity.com/packages/tools/utilities/b-zier-path-creator-136082)). *„Wie kann ich einen weg [mithilfe des Path Creators ] erstellen? Ich kann ihn bis jetzt nur im Spielmodus verwenden, aber nicht wirklich einen Pfad erstellen“*, ChatGPT kannte das Tool und hat mir erklärt, wie ich einen Pfad erstelle und wie ich ein Objekt entlang des Pfades bewegen kann:

void Update() {

distanceTravelled += speed \* Time.deltaTime;

// Setze die Position des Objekts entlang des Pfades

transform.position = pathCreator.path.GetPointAtDistance(distanceTravelled);

}

Diesen Code baute ich dann in den vorherigen Code ein, sodass die Bots nun entlang des Pfades in Richtung Base laufen, in einem bestimmten zeitlichen Abstand zueinander spawnen und wenn sie an der Base angekommen sind, zerstört werden-

### Was macht der Code?

Zu diesem Zeitpunkt machte das Skript BotsOnPath folgendes: es bekommt 3 Public – GameObjects vorgegeben, den Pfad, eine Variable „moveSpeed“ und eine weitere Variable „wait“. In dem Scirpt gibt es bisher keine „void Update()“ – Funktion, sondern nur eine „void Start“ – Funktion. Diese führt eine Coroutine (SpawnBots) aus, die mithilfe einer weiteren Funktion (GetBotToSpawn()), zwischen den public GameObjects durchwechselt und diese mithilfe einer weiteren Funtkion (SpawnNewBot) das zu spawnende GameObject am Anfang des Pfades instanziiert, für „wait“ – Sekunden wartet, den nächsten Gegner instanziiert usw. und mithilfe einer neuen Coroutine (MoveBotsAlongPath) jeden Gegner, sobald er instanziiert wurde, abhängig von der Variable „moveSpeed“, in einer bestimmten Geschwindigkeit den Pfad entlang laufen lässt, bis der Pfad zu Ende ist. Am Pfadende wird der Gegner zunächst zerstört.

## Erstes Ziel (Jaron): Third-Person-Controller

Meine erste Aufgabe war es, einen gut funktionierenden und anpassbaren third-person controller zu erstellen.

### Vorbereitung:

Auf Youtube habe ich ein sehr informatives Video zu meinem Thema gefunden: <https://www.youtube.com/watch?v=4HpC--2iowE>

In diesem Video wurde ein einfacher Controller mithilfe von Cinemachine erstellt. Dieses Konzept habe ich teilweise übernommen, aber noch für unsere Bedürfnisse abgewandelt.

### Teil 1 - mein Skript:

Ganz zu Beginn des Scripts habe ich den Animator für die Charakteranimation, 3 floats für die Kamerasteuerung und die Kamera initialisiert. In der void start Methode wird nochmal der Animationcontroller des Charakters mit folgender Zeile im Code definiert: anim = GetComponent<Animator>();

So wird vom GameObject das Element „Animator“ gesucht und als anim im Script definiert. In der void update Methode, die jeden Frame einmal abläuft, wird zuerst Unitys-Inputsystem mit zwei Floats geupdatet:

float horizontal = Input.GetAxisRaw("Horizontal");

float vertical = Input.GetAxisRaw("Vertical");

Danach wird ein Vector erstellt, der die Beiden Bewegungsrichtungen kombiniert.

Vector3 direction = new Vector3(horizontal, 0f, vertical).normalized;

In der folgenden Schleife wird abgefragt, ob der Vector direction eine Änderung im Wert erhält, wenn ja wird der Kamerawinkel angepasst.

if (direction.magnitude >= 0.1f)

{

float targetAngle = Mathf.Atan2(direction.x, direction.z) \* Mathf.Rad2Deg + cam.eulerAngles.y;

float angle = Mathf.SmoothDampAngle(transform.eulerAngles.y, targetAngle, ref turnSmoothVelocity, turnSmoothTime);

transform.rotation = Quaternion.Euler(0f, angle + mouseX, 0f);

Vector3 moveDir = Quaternion.Euler(0f, targetAngle, 0f) \* Vector3.forward;

controller.Move(moveDir.normalized \* speed \* Time.deltaTime);

}

Um ehrlich zu sein, habe ich diesen Code nur aus dem Video abgeschrieben, da die Syntax sehr komplex ist. Der nächste Teil ist komplett von mir geschrieben und kontrolliert die Animationen während Tastendrucks:

if (Input.GetKey(KeyCode.A) || Input.GetKey(KeyCode.W) || Input.GetKey(KeyCode.S) || Input.GetKey(KeyCode.D))

{

anim.SetFloat("Blend", 1.0f, 0.1f, Time.deltaTime);

}

else

{

anim.SetFloat("Blend", 0.0f, 0.1f, Time.deltaTime);

}

Die Blend Variable bestimmt im Animationscontroller den Übergang zwischen der Idle- und der Laufanimation.

Danach kommt noch eine Abfrage, ob Shift gedrückt wird, dann wird die Speedvariable hochgesetzt und der Charakter bewegt sich schneller. Auch habe ich noch Methoden erstellt, um den Cursor auf dem Screen zu locken oder nicht, diese werden aber nicht mehr verwendet. Nachdem das Script fertig war, konnte ich in Unity alles richtig konfigurieren. Als erstes brauchte ich einen simplen Charakter zu testen, diesen habe ich aus dem Asset Store heruntergeladen. Die Kamerabewegung musste über ein neues leeres GameObject gesteuert werden, welches als FreeLook Kamera von Cimemachiene instanziiert wird. Damit wird auch automatisch die Main Kamera an die Kamerabewegung gebunden, damit man die Bewegung auch sieht. Danach musste ich nur noch den Bewegungsraum für die Kamera einstellen, ein Smoothing hinzufügen und die Kamera war theoretisch bereit. Jedoch hatte ich immer noch Probleme mit der Mausgeschwindigkeit und Kamerklipping. Mit ChatGPT habe ich versuch das Geschwindigkeitsproblem zu lösen, er hat aber die Kamerabewegung von der Bewegung des Charakters getrennt, wonach die Bewegungsrichtung des Charakters nicht mehr mit der Kamerabewegung synchronisiert war.

### Teil 2 - alles neu

Aufgrund von eben genannten Problemen, habe ich mich entschieden den Third Person Controller aus dem Asset Store herunterzuladen.

<https://assetstore.unity.com/packages/tools/game-toolkits/third-person-controller-basic-locomotion-free-82048>

Dieses Tool hat es mir erlaubt, einfach einen fertigen Charakter und ein fast fertiges Script ins Spiel eizuarbeiten. Fast fertig war es nur, da ich umsetzen wollte, dass die Kamera sich nur bewegt solange man die rechte Maustaste gedrückt hält. Also musste ich mich in den Aufbau des Scripts einlesen und eine if-Abfrage an entsprechender Stelle ergänzen.

Danach habe ich noch den Charakter im Spiel hochskaliert. Nun musste ich die besten Werte für Laufgeschwindigkeit, Sprunghöhe und Länge und mehr testen und finden.

## Zweites Ziel (Tim): Schadensystem

Meine zweite Aufgabe war ein Schadenssystem zu erstellen. Das heißt, jeder Gegner benötigt eine Lebensanzeige, die Base benötigt eine Lebensanzeige und die Lebensanzeige muss „weniger“ werden, wenn das Objekt Schaden nimmt.

### Teil 1 - Leben von Gegnern

#### Vorgehensweise

Zuerst habe ich ChatGPT gefragt, *„wie kann ich in Unity eine Healthbar erstellen?*“, da seine Erklärung jedoch für mich noch sehr unverständlich und kompliziert war, habe ich mir ein Tutorial auf YouTube rausgesucht: [How to make a HEALTH BAR in Unity!](https://www.youtube.com/watch?v=BLfNP4Sc_iA&ab_channel=Brackeys).

Nachdem ich dem Tutorial Schritt für Schritt gefolgt bin und ein paar Anpassungen vorgenommen hatte, hatte ich eine Szene, mit einer Healthbar, die Schaden bekommt, sobald ich „E“ drücke. Das Tutorial gab mir auch Hinweise darauf, wie ich die Healthbar an ein Gameobject anfügen kann, sodass es sich in der Scene mit dem Gameobject mitbewegt. In meiner Testszene (Healthbar), hatte ich aber nur dir Healthbar, also eine Canvas, mit einem Gamobject, welches einen Slider und ein „HealthSlider“ – Script besitzt und drei weiter Gamobjects (Fill Area, Fill und Background), die für das Design der Healthbar verantwortlich sind. Außerdem, hatte die Scene ein „DamageTest“ – Script. Dieses war zunächst dafür gedacht, um zu testen, ob die Healthbar funktioniert.

Um die Healthbar nun mit den Gegnern zu kombinieren, kopierte ich die Canvas mit ihren Children (Objekte, die einem anderen Objekt zugeordnet sind und sich verändern, wenn man das übergeordnete Objekt (Parent) verändert) und fügte sie in die „PathEditor“ – Scene ein. Danach erstellte ich einen „TestEnemy“, der zunächst kein Prefab war und hauptsächlich zur Formatierung der Healthbar über dem Gegner diente. Aus dem eben erwähnten Video, entnahm ich auch den Code, der die Healthbar immer zur Kamera gucken lässt und übertrug ihn auf ein Skript namens „Billboard“, das auf der Canvas der Healthbar liegt. Mithilfe eines „CameraRotate“ – Skripts, dass auch ChatGPT für mich erstellt hat *(wie kann ich die Main Kamera um die Scene drehen lassen?)* konnte ich die Funktion des Billboards testen. Um den „TestEnemy“ schließlich zu einem Prefab zu machen, musste ich im Billboard – Skript direkt auf die Main Kamera verweisen, da, sobald durch das BotsOnPath – Skript ein Klon von dem GameObject erstellt wird, der Public Verweis nicht erhalten blieb und eine „MissingReference“ – Exception entstanden ist (Die Kamera war als „public Gameobject“ in dem Billboard definiert, sodass man sie für jedes Gamobject im Inspector definieren muss).

Nun musste ich nur noch den Prefab für BotsOnPath als Public GameObject definieren und hatte damit Gegner mit einer Lebensanzeige, die einen Pfad entlanglaufen und Schaden nehmen, wenn der Spieler „E“ drückt.

#### Was macht der Code?

Die Skripts, die für das Spiel relevant sind, sind das DamageTest – Skript, das Billboard – Skript, welches nur sagt, dass nach jedem Frame (LateUpdate), die Healthbar sich so drehen soll, dass sie zur Hauptkamera guckt, und das HealthSlider -Skript. Auf das CameraRotate – Skript werde ich nicht eingehen, da dies nur zum Testen der Billboard Funktion diente.

DamageTest: Dieses Skript besitzt 3 Public Variablen, public int currentHealth; public int maxHealth und public HealthSlider healthbar; (Referenz zur Lebensanzeige), eine void Start() Funktion, eine public void TakeDamage(int damage) Funktion und zu diesem Zeitpunkt noch eine Update Funktion.

Die Start Funktion setzt die currentHealth auf den Wert von maxHealth und setzt den maximalen Wert der Healthbar des Objekts, auf dem das DamageTest Skript liegt, auf den Wert maxHealth.

Die TakeDamage Funktion fügt dem Objekt, auf dem das DamageTest Skript liegt schaden zu. Es setzt also den Wert von currentHealth auf currentHealth – damage, aktualisiert die HealthBar und zerstört das Objekt, wenn currentHealth kleiner gleich Null ist.

In der Update Funktion steht, dass wenn „E“ gedrückt wird, TakeDamage(20) ausgeführt werden soll.

HealthSlider: Das HealthSlider-Skript zu schreiben, war nicht allzu schwer, da vieles ChatGPT für mich getan hat. Die größte Schwierigkeit bestand darin, das Skript im Inspector richtig anzuwenden, sodass die Funktionen des Skripts wie gewünscht funktionieren. Den Code, den mir ChatGPT gegeben hat, sah folgendermaßen aus:

public class HealthSlider : MonoBehaviour

public class HealthSlider : MonoBehaviour

{

public Slider slider; // Referenz auf den Slider

public Gradient gradient; //Farbe

public Image fill;

// Setze die maximale Gesundheit im Slider

public void SetMaxHealth(int health) // Wie viel soll der Slider MAXIMAL anzeigen?

{

slider.maxValue = health;

slider.value = health;

fill.color = gradient.Evaluate(1f);

}

// Aktualisiere die aktuelle Gesundheit im Slider

public void Sethealth(int health) // Wie viel soll der Slider ANZEIGEN?

{

slider.value = health;

fill.color = gradient.Evaluate(slider.normalizedValue);

}

}

An diesem Code habe ich lediglich die Kommentare geändert (diese Kommentare sind bereits meine Kommentare). Der Code sagt kurzgesagt, dass wenn SetMaxHealth(int health) aufgerufen wird, der maximale Wert der Healthbar auf health gesetzt wird und, wenn Sethealth(int health) aufgerufen wird, der aktuelle Wert von der Healthbar auf health gesetzt wird.

Um das Skript zum Funktionieren zu bringen, musste ich einen Gradient erstellen, diesen als Public Gradient für das Skript definieren, auf das richtige Image hinweisen, auf den richtigen Slider im Inspector hinweisen und im DamageTest-Skript richtig auf die Funktionen hinweisen. In diesem Skript musste ich zuerst verstehen, welche Begriffe vorgegebene Begriffe für etwas sind, also ob Gradient auch anders heißen könnte und ob gradient auch anders heißen könnte, oder ob das jeweils Datentyp von C# sind, wie zum Beispiel „int“. „*kann ich Gradient gradient auch zu Hose hose umbenennen, wenn ich im Inspector auf den richtigen Gradient hinweise*“, es stellte sich heraus, dass ich zwar Gradient nicht zu Hose ändern kann, weil es ein Datentyp ist, aber dafür gradient zu hose ändern könnte, weil dies nur der Name, der für den Datentyp im Skript verwendet wird.

### Teil 2 - Schaden an der Base

Bislang gibt es Gegner, die einen Pfad entlanglaufen und Schaden bekommen können, der im Spiel über den Gegnern angezeigt wird. Es fehlen jetzt die Leben der Base.

#### Vorgehensweise

Die Base benötigt ein Skript, wie das DamageTest-Skript und ein HealthSlider-Skript, um eine Lebensanzeige wie die Gegner zu besitzen. Da wir das DamageTest-Skript für die Base später wahrscheinlich anpassen müssen, ohne das DamageTest-Skript von den Gegnern zu verändern, erstellte ich ein neues Skript, welches genau die gleiche Funktion wie das DamageTest-Skript hatte und nannte es „BaseHealth“.

Zur einfacheren Formatierung bzw. Bearbeitung des Pfades, habe ich die Base als ein Children zu dem Path- Gameobjekt zugefügt, sodass nun alles, das auf dem Pfad stattfindet unter einem Parent (Path) steht. Dazu gehört der Pathspawner, auf dem vor allem das BotsOnPath Skript liegt, der PathCreator mit dem Pfad und die Base mit deren HealthBar und den dazugehörigen Skripts.

Zudem sollte die Base Schaden bekommen, wenn sie von einem Gegner getroffen/ berührt wird. Um in Unity zu erfassen, ob sich zwei Objekte berühren oder miteinander Kollidieren, müssen beide Objekte einen Collider besitzen. Ich habe also der Base einen Box-Collider gegeben, da die Base die Form eines Quaders hat, und dem Gegnerprefab einen Capsule-Collider, da dieser die Form einer Kugel hatte. --[Unity 3D Collisions, Colliders, & Hitboxes (In 3 Minutes!!) - YouTube](https://www.youtube.com/watch?v=mkErt53EEFY&ab_channel=RoyalSkies)—Außerdem benötigt mindestens eines der Beiden Objekte einen Rigidbody, welchen ich auf das Gegnerprefab gelegt habe.

Mit Hilfe des Videos und ChatGPT *(„ich will, dass die clone von den bot prefabs der Base Schaden zufügen“)* habe ich mir ein Skript namens MainDamage zusammengebastelt, das auf jedem Gegnerprefab liegt, damit unterschiedliche Gegner unterschiedlich viel Schaden zufügen können. Ich hatte die Schadenfunktion zuerst auf der Base liegen, habe jedoch festgestellt, dass dadurch alle Gegner gleich viel Schaden zufügen.

#### Was macht der Code?

Wie bereits beschrieben, ist die Schadensanzeige der Base und die BaseHealth nicht viel anders, als die Skripts, die ich bereits beschrieben habe, mit der Ausnahme, dass der Gradient als erstes ein anderer war, der besitzt jedoch jetzt wieder die Farben, wie der der Gegner. Die einzige Änderung am Code ist also das MainDamage Skript.

In diesem wird zuerst eine Variable Damage definiert, die die Anzahl des Schadens angibt und in der Start Methode, wird auf das GameObject „Base2“ hingewiesen. Das Skript besitzt außerdem eine OnCollisionEnter Methode. Eine vorgegebene Methode, die jedes Mal, wenn das Objekt mit einem anderen Objekt mit Collider (Deshalb benötigen sowohl die Base, als auch die Gegner einen Collider) kollidiert, ausgeführt wird. In dieser wird, wenn das Objekt, mit dem der Gegner kollidiert, „Base2“ heißt, der Base damage Schaden zugefügt und danach das Gameobjekt (also der Gegner) zerstört, sodass der Base nicht jeden weiteren Frame damage Schaden zugefügt wird.

## Zweites Ziel (Jaron): Placement Shit

Meine nächste Aufgabe, und auch die, an der mit Abstand am längsten gesessen habe, war die Entwicklung eines Systems zur Platzierung von Gegenständen im Spielbereich. Dies soll ermöglichen, dass der Spieler sein letztendliches Ziel erfüllen kann, die Gegner zu töten. Und ohne irgendwelche Türme hat er nichts, was schießen kann. Diese Türme müssen also irgendwie platziert werden.

Zu Beginn dieses Teils muss ich sagen, dass ich viele Iterationen meines Codes leider nicht gespeichert habe und nur noch die finalen Versionen zur Verfügung habe. Ideen, die ich zwischendurch hatte und ich letztendlich nicht umgesetzt habe, kann ich leider nicht mehr detailliert anbringen. Ich werde jedoch immer an gegebener Stelle skizzieren, welche Ideen und Gedanken ich hatte.

### Teil 1: Was muss ich überhaupt machen? -- Position ermitteln

Das große Ziel war mir natürlich die gesamte Zeit bewusst. Zu Beginn habe ich mich hauptsächlich informiert, wie ich Positionsdaten abrufen kann. Dadurch, dass wir uns für eine Third Person Umgebung entschieden haben, hat sich im Verlauf sehr viel als sehr viel schwieriger als erwartet entpuppt. (Ja komischer Satz 😉) Recht schnell war mir klar, dass ich nicht an einen Raycast als Methode der Positionsbestimmung vorbeikommen würde. Youtube sei gepriesen, so kam ich an ein Tutorial, welches mir ein sehr gutes Verständnis für dieses Thema gegeben hat. So habe ich mich dazu entschieden, einen externen „SpawnHandler“ als ein neues GameObject zu definieren, der all die Logik und Positionsberechnungen zentral verwaltet. Das habe ich auch getan, damit nicht jeder Turm, der im Spiel gespawnt wird, im Hintergrund das Skript zu laufen hat und nur ein GameObject die Positionsdaten an die entsprechenden Prefabs weitergibt und bestimmt, wo diese Initialisiert werden.

void Update()

{

// Wenn das Spawning aktiviert ist und die linke Maustaste gedrückt wird

if (spawnEnabled && Input.GetMouseButtonDown(0))

{

if (gameManager.SpendCredits(cost)) // Tims Änderung -- wenn der Preis bezahlbar ist.

{

// Raycast von der Kamera zum Mauszeiger

Ray ray = Camera.main.ScreenPointToRay(Input.mousePosition);

RaycastHit hit;

// Wenn der Raycast etwas trifft

if (Physics.Raycast(ray, out hit))

{

// Spawnen des ausgewählten GameObjects an der Hit-Position

GameObject spawnedObject = Instantiate(prefabsToSpawn[selectedPrefabIndex], hit.point + new Vector3(0, 3, 0), Quaternion.identity);

Dies ist ein Auszug aus dem Skript, welches jeden Frame des Spiels läuft (void Update Methode). Ein von mir gesetztes Ziel war es, den Prozess des Spawnens des Turms erst zu starten, wen die linke Maustaste gedrückt wurde. Dies geschieht in der ersten Zeile, gemeinsam mit der Abfrage ob ein gesetzter Boolean „SpawnEnabled“ true ist. Dieser Wert wird während des Drückens auf den Knopf auf der UI true gesetzt und die If-Abfrage wird ausgeführt. Danach folgt eine Abfrage, ob der Turm auch bezahlt werden kann, dies gehört aber zu Tims Bereich. Jetzt wird’s spannend. Mit „Ray ray = Camera.main.ScreenPointToRay(Input.mousePosition);“ wird ein simulierter Strahl von der Perspektive des Spielers in das Spiel, auf den Boden, geschossen. Die Referenz zu der Camera.main ist unabdingbar, da der Strahl ansonsten nicht wüsste, wo er starten soll. So jetzt wo der Strahl abgeschossen wurde, muss nur noch geschaut werden, wo er auftrifft. Das gestaltet sich aber nicht so einfach. So soll der Turm ja bspw. nicht im Spieler oder anderen Türmen gespawnt werden. Demzufolge müssen alle Objekte, die nicht der Boden sind, so im Inspektor konfiguriert werden, dass sie vom Raycast Hit ignoriert werden. Jetzt wo der Raycast/Strahl den Boden und nix anderes getroffen hat, kann ein Hitpoint erstellt werden. Dies erfolgt aber nicht so wie man es denkt. Unzwar wird erst mit „if (Physics.Raycast(ray, out hit))“ geprüft, ob der Strahl etwas trifft und dann wird mit diesen Parametern direkt ein GameObject instanziiert.

GameObject spawnedObject = Instantiate(prefabsToSpawn[selectedPrefabIndex], hit.point + new Vector3(0, 3, 0), Quaternion.identity);

Das prefabsToSpawn[selectedPrefabIndex] ist hierbei das aus einem Array ausgewählte GameObject, welches einem Knopf von der Auswahl der Türme zugeordnet ist. Wie das funktioniert erkläre ich später. Der Hitpoint wird nun mit einem Offset versetzt, damit der Turm auch auf dem Boden, und nicht im Boden gespawnt wird.

Danach werden wird erstens der Name des gespawnten Objekts verändert, ein Tag vergeben und ein Boolean spawned auf true gesetzt. Dies geschieht nur, um das Objekt für andere Skripte klar erkennbar zu machen.

Zusammenfassen können wir für diesen Teil also, dass irgendetwas weiß, wohin es muss. Aber nur wenn es darf und bezahlt werden kann. An der Stelle muss ich nochmals betonen, dass allein dieser Teil sehr lange gedauert hat, da ich sehr lange gebraucht habe, um den RaycastHit unabhängig von der direkten Bewegung des Charakters zu halten und die Mausposition entsprechend der Kameraposition zu bestimmen.

### Teil 2: Türme auswählen können

Ein etwas kürzerer Abschnitt bestand darin, es zu ermöglichen verschiedene Arten von Türmen spawnen lassen zu können. Dies gliedert sich in 2 Konstrukte:

* Das Skript, welches ein Array erstellt und bestimmte auf dem UI erstellte Buttons mit einer Liste von Prefabs verbindet
* Eine UI welche funktionierende Buttons hat und die Möglichkeit, diese mit dem Skript zu verknüpfen.

Angefangen habe ich, ein einfaches UI-Element zu erstellen, darauf gehe ich aber noch im Thema erste UI-Versuche ein. Fakt ist nur, wir haben zwei Buttons, die sobald gedrückt das Spawnen des entsprechenden Turms aktivieren sollen.

void Start()

{

// Initialisierung der Buttons:

for (int i = 0; i < spawnButtons.Length; i++)

{

int buttonIndex = i; // Speichern des Button-Index für den Event-Handler

spawnButtons[i].onClick.AddListener(() => {

selectedPrefabIndex = buttonIndex; // Setze den ausgewählten Index, wenn der Button gedrückt wird

spawnEnabled = true; // Aktiviere das Spawning

//Tims Änderung

UpdateCost();

});

}

}

Diese Start Methode läuft einmal ab, sobald das Skript aufgerufen wird. Voraussetzung für diese Methode sind, ein Array von Prefabs, ein Array von Buttons, ein Integer der das ausgewählte Element und dessen Position im Array angibt. In der for-Schleife wird zunächst ein temporärer int erstellt, der das Lesen der Position ermöglicht. Danach wird bei durchlauf der Schleife geprüft, ob ein Knopf gedrückt wird. Jeder Durchlauf der Schleife entspricht einem einzelnen Knopf. Sobald ein Knopf gedrückt wird, wird das Äquivalent als Element im Array der Prefabs als Zahl im int Wert selectedPrefabIndex gespeichert, damit dieser später beim Instanziieren abgerufen werden kann.

Im Verlauf dieses Abschnitts war mir ChatGPT tatsächlich eine sehr große Hilfe, da eines meiner Hauptprobleme der korrekte Syntax in C# ist. Hierbei konnte es mir natürlich sehr gut helfen.

### Teil 3: Nicht nur Platzieren sondern auch Bewegen (Uiii spannend)

An dieser Stelle kann der Spieler also nach Drücken des Buttons, wo der Name des entsprechenden Turms steht, mit einem Mausklick den Turm an einer Stelle fest platzieren. Da dachte ich mir das ist doch irgendwie blöd. Bspw. wenn man sich mal verklickt oder einfach unzufrieden ist, hat man keine Möglichkeit mehr, die Position des Turmes zu ändern? Nein nicht mit mir!!

Um ein späteres Verschieben des Turms zu ermöglichen musste ein zweites Skript her, welches wieder auf dem guten SpawnHandler liegt.

Jetzt greife ich die Identifizierung auf, die ich schon in Teil 1 angebracht habe. Beim Instanziieren des Turms werden im zwei Dinge „gegeben“: ersten ein Tag (LastSpawned“) und ein Name. Für das Preview Skript (sehr kreativer Name) ist nur der Tag wichtig. So wird am Beginn der Update Methode folgendes ausgeführt:

previewObject = GameObject.FindWithTag("lastSpawned");

Das previewObject ist das GameObject, welches eben vom anderen Skript gespawned wurde. Über den Tag kann auch dieses Skript des selbe GameObject problemlos finden und mit ihm interagieren.

if (spawnScript.spawned && previewObject != null && Input.GetMouseButton(0))

{

Ray ray = Camera.main.ScreenPointToRay(Input.mousePosition);

RaycastHit hit;

// Wenn der Raycast etwas trifft

if (Physics.Raycast(ray, out hit))

{

// Setze die Position des Vorschau-Objekts auf den Trefferpunkt

previewObject.transform.position = new Vector3(hit.point.x, 3, hit.point.z);

}

}

Jetzt wird wieder ähnlich wie beim ersten Teil ein Raycast erstellt, der von der Position der Kamera auf den Boden geht. Für ihn gelten dieselben Bedingungen wie für den anderen auch. Der große Unterschied ist nun nur, dass das GameObject nicht instanziiert wird, sondern nur dessen Position verändert wird. Dies geschieht in der Hit if Abfrage mit dem eben genannten Code. Auch wichtig dazu zu erwähnen ist, dass dieser Code auch jeden Frame solange durchläuft bis folgendes passiert: (was folgt, ist der zweite Teil der Update Methode)

if (Input.GetMouseButtonUp(0))

{

spawnScript.spawned = false; // Setze spawned auf false, wenn die linke Maustaste losgelassen wird

if (previewObject != null)

{

previewObject.tag = "Untagged";

}

}

Sobald also die linke Maustaste losgelassen wird, wird die Bedingung spawned aus dem anderen Spawnscript auf false gesetzt und der Tag vom GameObject entfernt, damit das Skript nicht mehr darauf zugreifen kann und dessen Position verändern kann.

### Teil 4: Reichweitenvisualisierung

Für diesen Teil ist mein Ziel eigentlich recht simpel. Nachdem der Spieler auf den Button zum spawnen gedrückt hat und der Turm an einer Stelle instanziiert wird, soll ein Ring oder eine kreisförmige Fläche um den Turm erscheinen, welche die Reichweite des Turms visualisiert. So soll der Spieler direkt und effektiv einschätzen können, wo er seine Türme platzieren muss.

Generell musste ich bevor ich irgendwelchen Code schreiben konnte, ein Teil namens „Range“ in jedem Turmprefab erstellen. Bei mir ist das jetzt einfach nur ein blauer, flacher Zylinder, der auf dem Boden aufliegt. Dieses Teil ist standartmäßig deaktiviert, im Spiel also nicht zu sehen.

void selectPart()

{

previewObject = GameObject.FindWithTag("lastSpawned");

// Find "Range" within the prefab

partToActivate = previewObject.transform.Find("Range").gameObject;

// If "Range" is not found, issue a warning

if (partToActivate == null)

{

Debug.LogWarning("Range Objekt im Prefab nicht gefunden!");

}

}

Mit dieser Methode wird das Objekt „Range“, von ich eben sprach, als ein GameObject im Skript gespeichert bzw. definiert. In der ersten Zeile wird also nach einem GameObject im Spiel gesucht, welches den Tag „lastSpawned“ trägt und dieses wird als previewObject hinterlegt. Der Tag wird von Skript in Teil 1 vergeben. Danach wird im Prefab das Range Object definiert, was dazu führt, dass es in der Update Methode des Skripts aufgerufen werden kann. Der Rest ist nur noch für Troubleshooting.

void Update()

{

// Check if spawnScript is assigned and spawned is true

if (spawnScript != null && spawnScript.spawned)

{

selectPart();

if (partToActivate != null)

{

partToActivate.SetActive(true); // Activate the "Range" part

}

else

{

Debug.LogWarning("Kein Teilobjekt zugewiesen!");

}

}

Hier wird zunächst überprüft, ob ein spawnskript hinterlegt ist(was es immer ist, nur für Debug) und ob der Wert spawned in diesem Skript true ist. Der Wert spawned sagt aus, ob der Turm instanziiert wurde. Passiert dies, wird durch die eben erklärte Methode das Teil Range gefunden und danach aktiviert. So erscheint das Teil im Spiel und es wird erkenntlich, wie weit der Turm schießen kann.

// Check if spawnScript is assigned and spawned is false

else if (spawnScript != null && !spawnScript.spawned)

{

if (partToActivate != null)

{

partToActivate.SetActive(false); // Deactivate the "Range" part

}

}

}

Sobald daraufhin der Platzierungsvorgang durch Beenden des Herumschiebens beendet wird, wird der Wert spawned wieder auf false gesetzt und das eben definierte Teil Range deaktiviert. Es ist im Spiel nicht mehr sichtbar.

Dieser ganze Teil bringt also keine fundamentalen Funktionen ins Spiel, sondern trägt hauptsächlich zur besseren Orientierung im Spiel und zum Spielspaß bei. Wichtig zu erwähnen ist auch noch, dass am Start des Skripts das Teil auf null gesetzt wird. Im Verlauf des Spiels wird der Wert des Teils einfach nur nacheinander überschrieben.

## Drittes Ziel (Tim): Basisfunktion der Türme

Um die Gegner töten zu können, soll man Türme platzieren können, die die Gegner beschießen. Mein Teil bestand darin, die Funktion der Türme zu programmieren. Ich brauchte also einen Turm, der sich in die Richtung des nächstgelegenen Gegners dreht und diesen abschießt, bis er entweder tot ist oder ein anderer Gegner näher an dem Turm dran ist.

### Vorgehensweise

Zunächst habe ich mich auf das Finden, des nächstgelegenen Gegners in einem bestimmten Radius konzentriert. Man soll also eine range angeben können, in der dann das Target gesucht wird. ChatGPT hat mir dann relativ schnell eine UpdateTarget Funktion gegeben, die eiwandfrei funktioniert hat, weshalb ich diese ein zu eins übernommen habe:

void UpdateTarget()

{

// Sucht nach allen Gegnern mit dem Tag "Enemy"

GameObject[] enemies = GameObject.FindGameObjectsWithTag(enemyTag);

float shortestDistance = Mathf.Infinity;

GameObject nearestEnemy = null;

// Finde den nächsten Gegner innerhalb der Reichweite

foreach (GameObject enemy in enemies)

{

float distanceToEnemy = Vector3.Distance(transform.position, enemy.transform.position);

if (distanceToEnemy < shortestDistance)

{

shortestDistance = distanceToEnemy;

nearestEnemy = enemy;

}

}

// Wenn ein Gegner gefunden wurde, setze ihn als Ziel

if (nearestEnemy != null && shortestDistance <= range)

{

target = nearestEnemy.transform;

}

else

{

target = null; // Kein Gegner in Reichweite

}

}

Diese Funktion prüft zwar erst nachdem der nächstgelegene Gegner gefunden wurde, ob er in der Reichweite liegt, erfüllt aber trotzdem die Voraussetzungen. Sie wird in der Start Funktion immer wieder ausgeführt. Außerdem gab mir ChatGPT eine relativ einfache Funktion, die die Reichweite des Turms im Editor anzeigt, auch, wenn das Spiel momentan nicht läuft.:

void OnDrawGizmosSelected()

{

Gizmos.color = Color.red;

Gizmos.DrawWireSphere(transform.position, range);

}

Der Rest des Tower Skripts, den mir ChatGPT gegeben hat, hingegen war weniger überzeugend. Ich musste viele Variablen hinzufügen, die Update Funktion überarbeiten, in welcher der Turm konstant in die Richtung des Targets gucken soll und in einem bestimmten Abstand Projektile instanziieren soll, und die Art, wie die Projektile geschossen werden. Ich entschied mich dagegen, so wie ChatGPT es haben wollte, die Funktion der Projektile in das Tower Skript zu legen und erstellte deshalb ein Projectile Skript für die Projektile, das kontrollieren sollte, wie sich die Projektile bewegen.

Auf dieses Skript bin ich besonders stolz, da ich, bis auf ein paar Syntax – Fehler, das komplette Skript ohne Hilfe geschrieben habe, so dass es anfangs ziemlich gut funktionierte. Ich habe keine weitere Möglichkeit gefunden, mit der ich von dem Projectile Skript aus auf das richtige Tower Skript zugreifen kann, sodass das Projektil die richtige Reichweite und die richtige Anzahl an Schaden usw. hat, als wieder den nächstgelegenen Turm zu suchen, also kopierte ich die Logik der UpdateTarget Funktion und baute sie so um, dass es am Anfang den nächstgelegenen Turm findet. Von diesem Turm sollte das Skript nun die nötigen Variablen nehmen (Entfernung, Speed, Schaden usw.), die dann in der Update Funktion dazu dienen, das Projektil zum Target zu bewegen. Um dem Gegner Schaden zuzufügen, ließ ich das Skript auf das DamageTest Skript von dem Target zugreifen und die TakeDamage Funktion ausführen, sobald das Projektil am Ziel ist.

### Was macht der Code?

Das Tower Skript ist dafür verantwortlich den Turm in die richtige Richtung zu drehen, so wie bereits beschrieben und diverse Variablen abzufragen, die im Inspector definiert werden: range (die Reichweite), projectilePrefab (die Art der Projektils), fireRate (die Schussfrequenz), bulletSpeed (die Geschwindigkeit des Projektils), damageAmount (den Schaden).

Das Projectile Skript ist hauptsächlich für die Bewegung des Projektils verantwortlich, wie bereits beschrieben. Dafür sucht es anfangs den nächstgelegen Tower und nimmt sich von ihm die Variablen range, bulletSpeed und damageAmount. Dann sucht es sich den nächstgelegenen Gegner und „verfolgt“ ihn bis es ankommt und ihm Schaden zufügt oder bis das Ziel selbst zerstört wird.

## Viertes Ziel (Tim): Projektile der Türme

Während unserer Testphase, haben wir einige Bugs festgestellt. Für zwei davon war ich verantwortlich, einmal, dass das BotsOnPath Skript auf die Bots (Gegner Objekte) zugreifen will, auch wenn sie bereits zerstört sind und dass die Projektile in der Luft hängen bleiben, wenn das ursprüngliche Ziel zerstört ist. Es war zwar witzig, den Bug auszutesten, da man mit dem Spieler auf den Kugeln laufen und springen konnte, es war aber nicht Sinn des Spiels. Außerdem stellten wir fest, dass es sinnvoll wäre, eine Markierung für den Weg der Gegner im Spiel zu haben, da man ihn bisher nur im Editor und nicht im Spielmodus sieht.

### Vorgehensweise

Als erstes wollte ich die Markierung einfügen, die ich komplett ohne Hilfe eingerichtet habe. Ich habe ein Prefab erstellt, und diesen mithilfe der Art, wie die Bots den Path langlaufen (Sie errechnen an einer Weite a, die Position auf dem Pfad, teleportieren sich dorthin und ändern a um einen Wert e) den Pfad entlang initialisieren lassen.

Die Bugs zu fixen war etwas aufwendiger bzw. schwieriger für mich. Ich stellte jedoch nach ein paar Stunden fest, dass ich beide mit einer Lösung fixen kann, und zwar mithilfe eines IsAlive public boolean, der für einen Gegner false wird, sobald dieser stirbt. Diesen habe ich dann in das BotsOnPath Skript eingefügt, dass es nur solange versuchen soll, den Bots zu bewegen, bis er Tod ist, und in das Projectile Skript eingefügt, sodass es das Projektil zerstören soll, wenn das Ziel Tod ist.

## Drittes Ziel (Jaron): UI

### Teil 1: Main Menü

Zwischendurch wollte ich mich von der Frustration abschlagen, dass alle möglichen Versuche, das Spawnsystem zu laufen zu kriegen, gescheitert sind, ablenken. Letztendlich habe ich begonnen, ein einfaches erstes UI für unser Spiel zu erstellen. Zum jetzigen Zeitpunkt hat noch alles die Standardtexturen von Unity, aber alles, was ich gemacht hat funktioniert. Zu Beginn habe ich mich darauf fokussiert, ein simples Hauptmenü für unser Spiel zu erstellen. So habe ich auf Youtube ein schönes Video gefunden, wie man ein UI in Unity erstellt. Als Hauptmenü habe erstmal einfach einen farbigen Hintergrund erstellt. Darauf kamen dann drei Buttons. Der erste, und oberste, Button der drei heißt ganz einfach „PLAY“ und öffnet nach dem Drücken das erste Level des Spiels. Wenn in Zukunft eine Möglichkeit zu speichern erstellt wird, wird dieser Knopf den aktuellen Speicherstand laden. Zum jetzigen Zeitpunkt ist dies aber noch nicht der Fall. Um den Wechsel in Level eins zu ermöglichen, wurde ein Skript erstellt, in der eine Methode namens „PlayGame“ definiert ist.

public void PlayGame()

{

SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex + 1);

}

Hierbei wird ganz einfach die nächste Szene im SceneManager geladen. Die Reihenfolge muss dann extern im Buildinspector eingestellt werden.

Der nächste Button öffnet ein einfaches Optionsfenster. Darauf sollen in Zukunft verschiedene Einstellungen wie Lautstärke und mehr einstellbar sein. Bis jetzt wird beim Klick einfach ein GameObject namens „OptionsPanel“ aktiviert. Auf diesem neuen Fenster befindet sich dann wieder ein Cancelbutton, mit dem das Fenster geschlossen werden kann. Dabei wird einfach nur das Fenster als GameObject deaktiviert.

Der unterste Button trägt den Titel „Quit“ und viel mehr macht er auch nicht – er beendet das Spiel. Hierzu wurde wieder im selben Skript wie beim Playbutton eine Methode erstellt.

public void QuitGame()

{

Debug.Log("Game is quitting...");

Application.Quit();

}

Mit dem ersten Befehl wird in der Konsole von Unity eine Meldung ausgegeben, dass das Spiel geschlossen wird. Mit dem zweiten Befehl wird das Spiel dann auch geschlossen. Dabei handelt es sich um eine Unity interne Funktion.

### Teil 2: Spawnpanel

Der Spieler hat bisher die Möglichkeit, über verschiedene Buttons auf dem Bildschirm verschiedene Türme zu spawnen. Diese Lösung ist schlecht skalierbar auf viele verschiedene Arten von Türmen. Daher möchte ich eine Lösung dafür finden.

Die Lösung ist in unserem Fall ein Fenster, was sich über einen Button öffnen lässt. So sieht der Spieler nur einen Knopf, wenn er diesen drückt, öffnet sich ein Fenster, wo alle Türme gespawnt werden können. Dafür müsste ich ein neues GameObject erstellen, welches einerseits das Fenster an sich beinhaltet und andererseits auch die ganzen Buttons für die Türme. Es ist standardmäßig deaktivieret, also nicht sichtbar.

Beim Drücken des Buttons wird nun das Fenster aktiv gesetzt. Auf dem Fenster befindet sich auch ein Fertig Knopf. Dieser deaktiviert das Fenster und lässt es dementsprechend verschwinden.

Eine wichtige Funktion wollte ich noch in das Fenster integrieren. Sobald man auf den entsprechenden Knopf für den Turm gedrückt hat, soll sich das gesamte Fenster auf die rechte Seite des Bildschirms verschieben, damit man genug Platz hat, um den Turm zu platzieren. Dafür habe ich ein Skript erstellt, welches zwei Methoden enthält. Eine davon verschiebt das Fenster nach rechts.

public void goright()

{

RectTransform rectTransform = GetComponent<RectTransform>();

rectTransform.anchoredPosition = new Vector2(700, 0);

}

Das Fenster wird also einfach 700 Pixel nach rechts geschoben. Angepasst ist dies auf eine Bildschirmauflösung von 1920x1080 Pixel. Diese Methode wird beim Klick auf einen beliebigen Knopf aus der Auswahl der Türme ausgeführt. Jeder dieser Knöpfe hat diese Methode hinterlegt.

public void goleft()

{

RectTransform rectTransform = GetComponent<RectTransform>();

rectTransform.anchoredPosition = new Vector2(0, 0);

}

Hiermit wird das Fenster dann wieder nach links bzw. in die Mitte des Bildschirms gebracht. Diese Methode wird beim Klick des Fertig Buttons ausgeführt, damit das Fenster beim nächsten Öffnen in der Mitte des Bildschirms erscheint.

# Reflektion Q1

## Tim

Im Großen und Ganzen bin ich sehr zufrieden mit meiner Arbeit an dem Projekt und mit dem, was ich bis jetzt erreicht habe. Ich kannte vor vier Monaten keinen einzigen Befehl von C#, ich kannte Unity nur als das, was beim Starten mancher Spiele auf dem Ladebildschirm sieht. Jetzt kann ich bereits vollfunktionsfähige Skripte schreiben, die ein bisschen mehr sind als „Bewege dich 3 Einheiten nach vorn“. Ich benötige dennoch bei Vielem Hilfe und da ChatGPT relativ schnell sehr passende Antworten liefert, habe ich das Projekt bis jetzt hauptsächlich mit Hilfe von ChatGPT bearbeitet. Man könnte zwar meinen, dass ChatGPT „cheaten“ wäre und ich muss zugeben, dass es das Lernen der Programmiersprache auf alle Fälle sehr viel leichter gestaltet. Wenn es zum Beispiel um das Verständnis spezieller Code Teile geht, kann man ChatGPT präzise Fragen stellen, ohne dass man sich Stunden lang durch irgendwelche Webseiten schlagen muss. Ich bin mir sicher, dass man ohne ChatGPT auch gut lernen kann, mir gefällt aber die Arbeit mit ChatGPT besser, weil es für mich leichter ist. Cheaten ist ChatGPT meiner Meinung nach aber nicht, da er nicht immer genau das macht, was man will, man kann also selten komplette Codes von ChatGPT schreiben lassen.

Zudem bin ich sehr zufrieden damit, dass ich durchgängig mehr oder weniger motiviert an dem Projekt gearbeitet habe. Ich war vor allem am Anfang sehr motiviert mich mit dem Thema auseinanderzusetzten, da es etwas völlig Neues für mich war. Es ist viel komplexer als zum Beispiel der Java Hamster, den ich aus dem Unterricht kenne, weshalb man viel mehr Sachen machen kann.

Trotz meiner durchgängigen Motivation und meiner Arbeit mit ChatGPT gab es Probleme bzw. bugs, Dinge die nicht so sein sollten, wie sie waren, die mir zeitweise auch Motivation geraubt haben. Während des Programmierens gab es mehrere Stolpersteine, an deren Lösung ich folgendermaßen herangegangen bin:

### Stolpersteine und Lösung

Der erste Stein über den ich gestolpert bin, war der PathCreator. Dieses Tool habe ich relativ früh aus dem Asset Store heruntergeladen, da meine Aufgabe ja war, die Gegner einen Pfad entlang laufen zu lassen. Mein Ziel mit diesem Tool war es, mir einen Großteil der Arbeit zu erleichtern. Am Anfang war ich jedoch noch nicht so vertraut mit der Unity Umgebung und wusste nicht wirklich, wie ich ein Asset richtig importiere bzw. musste mir dazu erstmal Videos anschauen. Als ich den PathCreator schließlich importiert habe, funktionierte die Bedienung irgendwie nicht so, wie in den Videos zu diesem Tool beschrieben. Ich konnte im Editor einfach keinen Pfad erstellen. Also habe ich erstmal die Schlussfolgerung gezogen, dass ich das Tool falsch installiert habe. Später stellte sich heraus, dass dies nicht der Fall war, er funktionierte aber immer noch nicht. Meine Lösung solcher Probleme lautet: Erstmal drüber schlafen – also PC aus und etwas anderes machen. Leider funktionierte der Editor den nächsten Morgen immer noch nicht und ich probierte alle Sachen durch, woran das liegen könnte. Habe ich eine Falsche Version von Unity installiert? Verwende ich die Falschen Befehle? und so weiter. An diesem Punkt habe ich nichts weiter gemacht, als irgendwas zu tippen und irgendwann hat es dann geklappt. Ich weiß bis heute nicht, was das Problem war.

Das nächste größere Problem war das Formatieren der Heatlth Bar beziehungsweise dessen Zuweisung zu dem dazugehörigen Game Object sobald das Spiel startet. Ich war mit dem Konzept von Parents und Children in Unity noch nicht vertraut und habe immer wieder die GameObjects in der Hierarchie hin und her gezogen und mich gewundert, warum das jetzt eingerückt war und das andere dann wieder nicht. Außerdem kannte ich mich noch nicht mir Prefabs und deren Funktion aus. Das Ziel der Healthbar sollte ja sein, sobald der Gegner gespawnt wird ihm Leben zu geben – die Gegner waren zu diesem Zeitpunkt noch keine Prefabs sondern Objekte die geklont wurden. Ich musste dazu also auch erstmal eine Pause vom Programmieren machen und mir Videos zu Unity und Prefabs angucken um zu verstehen, wie ich die Healthbar am besten den Gegnern zuordne. Nachdem ich das Konzept verstanden hatte, habe ich die Gegner mit einer Healthbar zusammen als Prefab gespeichert, was dann gespawnt wurde.

Bei unserer „Bug Test Phase“ haben wir drei Probleme, für die ich verantwortlich war festgestellt. Steine über die ich regelrecht gefallen bin. Die ersten zwei Probleme waren relativ schnell gelöst: Das erste war, dass das BotsOnPath-Skript versucht auch nach dem Sterben der Gegner (durch einen Turm) auf sie zuzugreifen. Dadurch kamen ganz schnell pro Frame zehn, zwanzig, dreißig Missing Referenze-Exceptions. Dazu hatte ich aber direkt die Idee, jedem Bot einen IsAlive Boolean zuzufügen und diesen bei BotsOnPath abzufragen. Dieser Stein wurde also sehr schnell aus dem Weg geräumt. Das zweite Problem habe ich durch die Lösung entdeckt: Ich hatte festgestellt, dass die Base zwei Collider besaß und mich gewundert, ob die Base dadurch vielleicht doppelt so viel Schaden bekommt. Dies hat sich bewahrheitet, weshalb ich den zweiten entfernt habe. Der Dritte Stein war aber irgendwie groß genug, mich zu Fall zu bringen: Die Projektile der Türme sind in der Luft hängen geblieben, wenn deren Target gestorben ist, bevor sie angekommen sind. Zuerst dachte ich, das Problem ließe sich so einfach lösen, wie das mit dem BotsOnPath-Skript. Ich habe es zwar schließlich auch mit demselben IsAlive-Boolean lösen können, es war aber irgendwie komplizierter. Hier habe ich mich in meiner Struktur des Tower- und des Projectile-Skripts verhangen. Diese habe ich so komisch miteinander verkettet, dass ich selber nicht mehr durchsah und mich gute drei Tage dadurch kämpfen musste (Diese Ketten habe ich jetzt nur noch komplizierter gemacht, ich sehe da aber irgendwie durch. Es sollte also kein weiteres Problem darstellen, es sei denn, irgendwer anders will diesen Skripts etwas hinzufügen). Ich musste mir die Skripts bestimmt dreimal angucken damit ich die Stelle finden konnte, an der ich diesen Boolean einbauen musste. Der Boolean musste schließlich an drei oder vier verschiedenen Stellen eingefügt werden. Ich gebe zu, keine sehr effektive Lösung, doch es Funktioniert.

Die ersten Stolpersteine sind darauf zurückzuführen, dass ich mich sozusagen Hals über Kopf in das Projekt reingestürzt habe und mir keine Zeit genommen habe, mich wirklich mit der Umgebung vertraut zu machen. Aus meiner Sicht aber, war das gar keine schlechte Idee, sondern einfach etwas, was mir geholfen hat, motiviert zu lernen. Mit den aufkommenden Problemen entdeckte ich immer wieder Aspekte in denen ich mir noch Wissen aneignen musste. Da ich durch das Lernen auch weiter in dem Projekt vorangekommen bin, war ich auch motiviert zu Lernen und sah es als Möglichkeit, effektiver zu Arbeiten. Also ganz nach dem Motto: Probieren geht über studieren.

Zusammenfassend bin ich sehr stolz darauf, was ich in diesem ersten Halbjahr über Programmieren gelernt habe und was ich zu dem Spiel beigetragen habe, obwohl ich davor noch kein einziges Mal ein Spiel in C# beziehungsweise in Unity programmiert habe.

## Jaron:

Um es einfach mal in den Raum zu werfen: Ich bin zufrieden.

Als Tim und ich im Sommer angefangen haben, für das Thema zu recherchieren, hätte ich niemals gedacht, dass wir schon jetzt, also Ende Q1, so viel Fortschritt haben würden. Und genau am Anfang lief mir auch schon ein recht großes Problem über den Weg. Ursprünglich war geplant, anstelle von Unity UnrealEngine als Entwicklungsumgebung zu verwenden. Nach bestimmt 10 Stunden Youtube-Tutorials hatte ich aber immer noch keinen Plan von grundlegenden Funktionen, die wir gebraucht hätten. Dazu kommt, dass Unreal C++ als Programmiersprache für Skripte benutzt. Diese Sprache hätten Tim und ich also komplett von Grund auf lernen müssen, was den Zeitaufwand, ohne wirklich am Spiel was zu machen, deutlich verlängert hätte. Die Lösung war in diesem Fall ganz simpel, Unity. Nicht nur gibt es zu Unity bedeutend mehr Content im Internet und Youtube, für Privatpersonen angepasst, sondern verwendet C# als Programmiersprache, welches sich vom Syntax her stark Java ähnelt, wir also schon ein gewisses Vorwissen hatten. Nachdem also alles geklärt war und wir uns voll in die Entwicklung geschmissen haben, lief eigentlich alles sehr gut. Ähnlich wie Tim habe ich zur Programmierung ChatGPT verwendet, jedoch lange nicht so oft. Oft habe ich mich auch einfach durch gefühlt unendliche Google-Seiten geprügelt, nur um dann irgendwann ganz unten auf einer Seite etwas Hilfreiches zu finden. Für die Zukunft werde ich wahrscheinlich mehr auf ChatGPT zurückgreifen, oder mal recherchieren, welche Alternativen man zur Codeentwicklung verwenden kann. Bis jetzt habe ich von Blackbox AI gehört, hatte aber bisher noch keine Chance, etwas für das Projekt mir ihr zu machen.

Im Hauptteil der bisherigen Arbeit, also den spawning von Türmen, hatte ich sehr viele Probleme mit komischen Glitches und nicht funktionierenden Skripten. Einerseits hat ChatGPT die Skripte manchmal logisch einfach falsch aufgebaut, in anderen Fällen weiß ich immer noch nicht, was wirklich falsch gelaufen ist. In den Fällen, wo die Skripte fehlerhaft waren, konnte ich oft durch umformulieren meiner Prompts etwas Funktionierendes produzieren. In vielen Fällen habe ich auch gar keine ganzen Skripte erstellen lassen, sondern wirklich nur kurze Abschnitte, da ich so besser den Überblick behalten konnte. An einer Stelle war ich dabei die Vorschau Logik zu Programmieren und dann folgte der Turm meiner Maus auf einmal nur noch der X-Achse. An anderer Stelle sind die Türme einfach im Boden gespawnt oder sind in anderen Objekten drin gespawnt. Diese kleinen Probleme waren aber tatsächlich oft schnell behoben. Lange habe ich jedoch an der richtigen Spawnposition gearbeitet, da anscheinend nichts funktionieren wollte. Letztendlich habe ich heraufgefunden, dass man alle anderen Objekte in der Welt auf eine Layer namens „Ignore Raycast“ setzen musste. Damit habe ich dieses Problem lösen können.

Danach ging es mit nur noch kleineren Holpersteinen weiter und ich konnte mich größtenteils auf die Entwicklung neuer Aspekte für unser Spiel fokussieren. Was ich an dieser Stelle noch hervorheben möchte ist, wie gut die Zusammenarbeit zwischen Tim und mir an diesem Projekt ist. Über unseren Verlauf bei Github sieht man deutlich, dass wir beide immer, wenn auch manchmal mit Pausen, recht aktiv am Projekt waren. Auch ist es schön, dass wir, falls es mal größere Probleme gibt, uns einfach mal treffen können und dann zusammen eine Lösung zu finden versuchen und meisten auch schaffen.

Ich hab jetzt kein crazy Outro mehr, aber das wars für Q1. Fühlt sich gut an.

# Notizen Q1

## Probleme

Stichpunktartig erstmal alle Probleme bzw. Dinge, die ziemlich kompliziert waren und sich nicht so einfach lösen ließen (z.B Bugs)

### Tim:

* Verwendung des PathCreators
  + Es war zuerst nur möglich, den PathCreator bei laufendem Spiel zu verwenden
  + Ich musste mich auch zuerst an das Programm gewöhnen und wusste nicht, was es bedeutet, wenn man zum Beispiel auf ein Objekt klickt.
* Formatieren der Gegner und der dazugehörigen Healthbar
  + Ich hatte das Konzept von Parent und Children in Unity noch nicht verstanden und habe als erstes versucht, die Healthbar am Anfang des Spieles sich dem Gegner anzufügen, sobald der Gegner gespawned wird.
* Projectile Bug
  + Die Projektil sind in der Luft hängen geblieben, wenn das Target gestorben ist, bevor das Projektil angekommen war.
* BotsOnPath Bug
  + Das BotsOnPath Skript hat weiterhin versucht, auf die Gegner zuzugreifen, nachdem sie gestorben sind und es entstanden dadurch für jeden Frame und jeden getöteten Gegner eine MissingReference - Exception.
* Base Damage Bug
  + Der Base wurde doppelt zu viel Schaden zugefügt, weil sie zwei Collider besaß

# Q2

## Fünftes Ziel (Tim): Gruppiertes Spawning und Extras

Mein fünftes Hauptziel war das Gruppierte Spawning. Das heißt, dass die Bots nicht mehr in regelmäßigen Abständen gespawned werden, sondern in verschiedenen Gruppen und Wellen. Um dieses Ziel zu verwirklichen galt es das komplette BotsOnPath Skript zu überarbeiten, da dieses für das Spawning der Gegner verantwortlich ist.

Das aktuelle BotsOnPath Skript war relativ simpel und ließ nicht viel Raum, das Level zu gestalten. Man konnte 3 verschiedene BotPrefabs definieren, die dann durchrotierend ( bot 1, bot 2, bot 3, bot 1, bot 2 usw.) vom Spawner in einem Zeitabstand von float wait mit einer Geschwindigkeit von float moveSpeed losgeschickt werden und den Pfad entlang zur Base laufen. – Ziemlich unkompliziert.

Ich habe viel Zeit damit verbracht, mithilfe von ChatGPT das System von Klassen Konfigurationen – so hat es zumindest ChatGPT genannt – zu entdecken. Davor wollte ChatGPT immer alle Wellen gleich definieren oder im Code für jede einzelne Welle eine neue Funktion schreiben. Das wollte ich aber nicht. Ich kam also zu dem Prompt: *Gibt es einer Möglichkeit, eine Variable zu erstellen, die mit jeder Neuen etwas neues zum Bearbeiten bereitstellt? Also, dass man sagt, ich will n Wellen haben, und mit jeder neuen Welle will ich diese dann bearbeiten können, also den Array an Bots, wie viele Gruppen usw.* Daraufhin hat ChatGPT mir so ziemlich den kompletten Code für BotsOnPath geliefert. ChatGPT erstellte eine eigene Klasse: public class WaveConfiguration in der man botsPerGroup, also die Anzahl der Bots pro Gruppe, groupWaitTime, also die Wartezeit zwischen den Gruppen, groupsInWave, also die Anzahl der Gruppen in dieser Welle, waveWaitTime, also die Wartezeit zwischen den Gruppen und botPrefabs, also ein Array von möglichen Gegner Prefabs, die gespawnt werden können, definieren konnte und das für jede Welle allein. Ich habe zwar eine Weile gebraucht um die Logik dahinter zu verstehen, zumal ChatGPT oft ziemlich ähnliche Namen für alles Mögliche verwendet, was zu Verwirrung führt und es somit manchmal schwer erkennbar ist, welches jetzt die Variable ist und welches die Klasse ist, aber ChatGPT konnte mir alles einigermaßen verständlich erklären. Eine detaillierte Beschreibung des BotsOnPath Skript werde ich an dieser Stelle noch nicht geben, da es im Zuge meines Achten Hauptziels wieder vielen Veränderungen unterzogen wurde.

Mit dem gruppierten Spawning kam die Idee auf, den Bots auch unterschiedliche Geschwindigkeiten zu geben, die in deren DamageTest Skripts definiert werden sollen. Dafür fügte ich im DamageTest Skript eine Variable float speed hinzu, die dann vom BotsOnPath Skript in der Coroutine MoveBotAlongPath(GameObject enemy) – von ChatGPT entworfen um die Gegner entlang des Pfades laufen zu lassen – einberechnet wird, sodass unterschiedliche Gegner nun auch unterschiedlich schnell sein können.

Während Q1 kamen Fehler oder noch hinzuzufügende Features auf, die wir in Q1 nicht mehr behoben bzw. hinzugefügt haben. Darunter befand sich die Animation der Gegner und das Schießen der Türme während des Spawn Prozesses.

War folgendes schon in Q2:

Bug mit Projektilen, die aufgrund der vielen Gegner hinunter wackeln.

Weg mit dem Vorwärtslaufen der Bots bei „W“

Türme schießen nur, wenn „spawned“ nicht aktiv ist. 🡪 Von Jarons Skript

## Viertes Ziel (Jaron): BigBabaPrefab

Meine nächste Aufgabe war es, den Prozess neue Level zu erstellen deutlich zu vereinfachen. Bisher musste man alle Elemente des Spiels manuell in jede neue Szene (dementsprechend auch Level) einfügen. In Zukunft soll es ein großes Prefab geben, was alle elementaren Objekte des Spiels enthält. Also soll einfach mit einem Klick in einer neuen Szene bereit sein, sich um Artwork des Levels zu kümmern. Umgesetzt wird das, indem man ein großes Prefab hat, ich hab es jetzt BigbabaPrefab genannt. Es liegt in den Game-Assets und kann dort immer aufgerufen werden. Es enthält viele verschiedene Elemente, dazu gehören der Spieler, also Graphics als auch Steuerung, die Elemente zur Erstellung des Pfades (Path-Creator Configuration), eine Plane, die den Boden darstellt, die wesentlichen Elemente des UI, also Spawnelemente, Credit-Anzeige und in Zukunft auch die Übersicht zum Upgraden der Türme. Also grob zusammengefasst alles, was wir bisher gemacht haben, was nichts mit Artwork zu tun hat.

Während der Umsetzung bin ich jedoch auf unerwartete Probleme gestoßen. Ich dachte ich kann einfach aus unserer bestehenden Szene alle Elemente in ein großes Prefab ziehen und fertig, ging aber nicht. Ich musste nach dem Kopieren aller Elemente jedoch alle Verbindungen und Definitionen innerhalb aller Objekte neu machen. Dies bedeutet also Verknüpfungen zu Skripten anderer GameObjects (von denen es sehr viele gibt) neu zu konfigurieren. Insgesamt gab es 4 Versionen vom Prefab bevor alles funktioniert hat.

In Zukunft müssen also alle Änderungen innerhalb des Prefabs immer geupdatet werden. Dies gestaltet sich recht einfach, da einfach das bereits vorhandene Prefab in der Szenenstruktur in den Assets gespeichert werden muss, also nur ein Klick. Die neue Version wird dann einfach in jede andere Szene übernommen.

## Sechstes Ziel (Tim): Credits

Das Erste, was ich in diesem Halbjahr erledigen wollte ist das Credit System. Ich hatte bereits im ersten Halbjahr den Drang die Credits einzubauen, da man ohne sie einfach nicht richtig spielen kann. Als aller erstes muss natürlich eine Credit Anzeige auf der UI her. Diese sollte jetzt nicht groß aufwändig gestaltet sein, sondern zuerst einfach nur ihren Zweck erfüllen.

Also habe ich auf der Canvas in der Spiel Scene einen neuen Text (TextMeshPro oder TMP) hinzugefügt, der Credits: <credits> zeigt. Ich hatte mir überlegt, dass ich irgendetwas festes in jeder Szene brauche, um die Credits zu verwalten. ChatGPT schlug eine GameManager Instanz vor, die irgendetwas besonderes sein sollte. Das ganze hat dann aber nicht so funktioniert, wie ChatGPT beschrieben hat – vermutlich, war ich das Problem – also habe ich das wie gewohnt als public variable gemacht, also „public int credits = 0;“. Diese Variable ist auf dem GameManager Skript, welches auf dem Objekt GameManager liegt, es funktioniert jedoch genauso, wie jedes andere normale Skript. Des Weiteren beinhaltet dieses Skript die Funktion AddCredits, die einen bestimmte Wert auf credits hinzuaddiert, und den Boolean Spendcredits, die neben dem abziehen von credits erst guckt, ob man den Wert auch wirklich bezahlen kann.

Es gibt nun mehrere verschiedene Teile vom Spiel, die mit credits interagieren müssen, allen vor ran die credits Anzeige. Auf der Anzeige liegt nun ein ziemlich einfaches Skript, welches jeden Frame den Text zu „Credits:“ + gameManager.credits; setzt. Es aktualisiert also jeden Frame die Anzeige.

Jetzt brauchen die Türme noch einen Preis und die Gegner eine Belohnung. Dafür mussten die Skripts SpawnTurret, Tower und DamageTest erweitert werden. Das Tower Skript brauchte die Variable price und ist diesbezüglich lediglich für die Bereitstellung dieser Variable zuständig, damit man übersichtlich alle Variablen, die ein Turm braucht, auf dem Tower Skript hat. Die Spawn Funktion liegt auf dem Skript SpawnTurret von Jaron, mit dem ich mich, um das Bezahlen einzubauen, erst einmal auseinandersetzen musste. In dem Skript gibt es eine große Update Funktion, die den Spawn Vorgang einleitet, sobald man auf einen spawnButton klickt, und das Spiel auch gleichzeitig in erlaubt einen Turm spawnen zu können. Zu diesen Bedingungen habe ich SpendCredits hinzugefügt. Die Variable amount, die der GameManager verwendet, um den bestimmten Betrag abzuziehen, bekommt das SpawnTurret Skript von dem Preis des Turms, der gespawnt werden soll. Fehlt also nur noch die Belohnung.

Für die Belohnung ist die Funktion addCoins relevant, die das GameManager Skript kontrolliert. Ich dachte, es wäre damit getan, eine Variable „reward“ dem DamageTest Skript hinzuzufügen und dann bei der Funktion „Die“, von DamageTest, diesen reward zu adden (gameManager.addCoind(reward);). Das führte aber dazu, dass man die Belohnung auch bekommt, wenn der Gegner gegen die Base rennt. Also musste ich irgendwie abfragen, ob der Gegner an der Base gestorben ist, oder nicht. Ich entschied mich für den Boolean „didDamage“ (er fragt im Nachhinein doch nicht das ab, was er sagt), der true ist, aber false gesetzt wird, wenn „Die“ ausgeführt wird, weil die Health des Gegners unter oder auf Null gefallen ist.

Damit war das Credit System vorerst abgehakt.

## Fünftes Ziel (Jaron): Credit Bedingtes Spawning (UI)

## Sechstes Ziel (Jaron): Türme Upgraden

## Siebtes Ziel (Tim): Neue Türme & Neue Bots

Mitunter das umfangsreichste Zwischenziel in Q2 ist die Überarbeitung und Erweiterung der Türme und der Bots. Anfangs hatte ich nur vor, interessantere Animationen und interessantere Designs für Türme und Gegner zu erstellen. Jedoch teilte sich dieses Ziel relativ schnell in mehrere weitere kleine Ziele auf.

### Anfänge mit Blender

Ich habe mich dafür entschieden, in Blender einzusteigen. Erstens konnte ich mir nicht vorstellen komplexe Designs in Unity zu erstellen. Zweitens ist Blender kostenlos und bietet mehrere Tools für 3D Modeling und Animation, weshalb ich mir als ersten Schritt Blender heruntergeladen habe. Danach habe ich mir erstmal ein [Blender Tutorial for Complete Beginners - Part 1](https://www.youtube.com/watch?v=B0J27sf9N1Y&ab_channel=BlenderGuru) angeschaut und daraufhin meinen ersten selbstgestalteten Turm in Blender fertiggestellt. Den habe ich dann TowerAlpha getauft und als FBX exportiert, damit er mit Unity kompatibel ist. Dieses 3D Modell (an Animation habe ich mich erstmal noch nicht gewagt) habe ich auf das Snipippie Turm Prefab gezogen.

Wie es zu erwarten war, gab es so einige Problem:

1. Das Spiel wurde unspielbar (Ziemlich großes Problem, das heißt: Panik) ich wollte sofort das Prefab wieder zurücksetzen, jedoch stellte ich fest, dass es daran lag, dass die Kamera vom 3D Modell aus Blender übernommen wurde. Zu dieser Kamera ist das Spiel dann, aus irgendeinem Grund, gewechselt, sobald man den Turm gespawnt hat, wodurch man nicht mehr spielen konnte. Ich musste also die Kamera aus dem Prefab entfernen
2. Danach war der Turm total überbelichtet, weil, wie sich herausstellte, auch das Licht von Blender mit übernommen wurde, welches ich also auch entfernen musste.
3. Am nervigsten ist und war die Formatierung der 3D Modelle. Erstens, ist die Richtung des 3D Modells bereits in Blender festgelegt, sobald ich zum Beispiel die Kanone erstelle. Wenn ich also das 3D Modell in Blender so baue, dass die Kanone in die Y Richtung, also nach vorne zeigt, wird das in Unity übernommen, aber so, dass die Kanone in die Z Richtung zeigt, weil das bei Unity anders ist, als in Blender. Außerdem war die Skalierung nicht wirklich richtig, was bedeutet, ich musste das 3D Modell nicht nur in die richtige Richtung drehen, sondern auch Skalieren und den Firepoint (Punkt, von dem das Projektil losgelassen wird) anpassen.

### Projektil Skript entfernen

Ich wollte die Schadensfunktion auf dem Turm Skript haben. Zudem hat mich das schon relativ lange gestört, dass fast die ganze Funktion eines Towers nicht auf dem Turm liegt, sondern auf dem Projektil. Animationen, vor allem komplexe Schussanimationen, lassen sich auch nicht so einfach in Unity machen. Deshalb habe ich mich dazu entschieden, das Projektil als Objekt komplett zu entfernen und dementsprechend auch das Projektil Skript. Des Weiteren ist es übersichtlicher.

Jetzt instanziiert die Shoot Methode des Tower Skripts kein Projektil Prefab mehr, sondern greift direkt auf die TakeDamage Methode von dem DamageTest Skript des Targets (Anvisiertes Gegnerobjekt) zu und fügt dem Gegner somit „damage“- Schaden hinzu, ohne irgendein Projektil.

Vorher hatte das Spiel noch einen kleinen Bug. Die Szene brauchte einen bereits existierenden Turm, damit die Türme ein Target finden. Dieser wurde durch Zufall mit dem Entfernen des Projektil Skripts gefixed, ich weiß aber bis heute nicht, warum.

### UI-Anpassungen

Nach dem Entfernen des Projektil Skripts habe ich erstmal zwei weitere Türme in Blender entworfen und in Unity exportiert. Diese waren, wie der erste, nicht sonderlich kompliziert und bestanden hauptsächlich aus einem Hauptzylinder, ein paar dekorativen Körpern und einer Kanone. Mir ging es hierbei in erster Linie darum, dass ich sie mit Blender erstelle und nicht um ein ausgefallenes Design. Ich hatte jetzt drei Türme, einen rot-orangen (den ersten), einen blauen (den zweiten) und einen grünen (den dritten). Der erste war immer noch Snippipie, der zweite Turret und den dritten taufte ich Hochmut -die Namen der 3D Modelle unterscheiden sich von den Namen, die sie im Spiel haben, welche schließlich auf den Buttons stehen. Die 3D Modelle bzw. die Prefabs haben die Namen TowerAlpha, - Beta, und -Gamma.- Da nun ein neuer Turm dazu gekommen ist, musste ich für ihn einen neuen Button erstellen, ihn beim Spawnpanel hinzufügen und ihm das Tower Skript geben. Schlussendlich habe ich es nicht richtig hinbekommen, den Turm voll funktionierend in das Spiel zu integrieren, weil die Struktur vom Spawnen und die der Buttons, die von Jaron ist, welche ich seit Neuestem nicht mehr versuche zu verstehen. Das Integrieren der neuen Türme in seine Skripts, habe ich also Jaron in Auftrag gegeben.

### Schussanimation

Dass es nun kein Projektil mehr gab, welches sich zum Target bewegt, führte dazu, dass es keine Schussanimation mehr gibt. Mein erster Ansatz war es, einen Rückstoß einzubauen, der direkt vom Tower Skript kontrolliert werden sollte. Dieser sollte so umgesetzt werden, dass sich die Kanone beim Schießen leicht zurückbewegt und wieder nach vorne. Dafür schrieb ich die Coroutine Recoil in das Tower Skript, die am Ende von Shoot ausgeführt wird, welche die Kanone „recoilDistance“ (recoilDistance ist eine neue public Variable und muss negativ sein, damit sich die Kanone nach hinten bewegt) in einer Zeit von „recoilSpeed“ (eine weitere neue public Variable. – Ich wollte die stärke des Rückstoßes im Inspector anpassen können, weshalb ich beide Variablen als public definiert habe) nach vorne bewegt. Und die Kanone unmittelbar danach in recoilSpeed Sekunden wieder zurückbewegt.

Unterpunkt: Animation für den Schuss erstellen – Recoil (Rückstoß der Kanone) 🡪 Problem mit dem blauen tower

Für jeden neuen Tower die UI anpassen: Neuer Button mit dem DisableButton Skript und dem Namen des Turms einrichten, bei Spawnhandler einen neuen Turm zum Array hinzufügen.

Immer wieder neue Probleme bei dem formatieren der Turm prefabs.

Überlegung, vielleicht Recoil wegmachen und alles mit Blender machen.

+ Prefab Überarbeiten (Alle Turmprefabs, Formatierung überarbeiten)

**LaserTower**

Erster Tower mit richtiger Schussanimation – Laserschuss, diese Animation wurde in Blender erstellt

Problem: Unity konnte die Innenseite der Halbkugel nicht erkennen, weshalb ich nach in Blender, die Kugel dublizieren und flipen musste, damit Unity die innenseite redert

Noch zu tun, um ihn zum Laufen zu bringen: Prefab anpassen, Tower Skript anpassen🡪 Problem: Es gibt Skripts, die nach Objekten mit dem Towerskript suchen. Wenn man für jeden Turm jetzt ein eigenes Skript macht, oder das Skript abändert, wird das nicht mehr funktionieren. Also: Neue Funktion für Spawncancelradius und Co finden, Animation in Skript einbauen, Tower richtig verlinken

Lösung: für jeden Turm ein eigenes Skript erstellen, das neue Funktionen besitzt, 🡪 Recoil weg und mit Animation aus blender arbeiten (für neue Türme zum Beispiel LaserTower). Entdeckung von Elternskript und Kindskript mithilfe von ChatGPT

Dafür erstmal: Im towerskript nur noch recoil, wenn der Turm keinen weiteren Komponenten besitzt

Auch neue Bots: Boos

Am Ende: UI Anpassungen

## Achtes Ziel (Tim): Level Vorbereiten

Zu meinem achten Ziel gehörte, eine erste Levelauswahl zu erstellen und das BotsOnPath Skript so anzupassen, dass ich Anfangen kann einzelne Level zu erstellen und diese zu balancen. Also die Vorarbeit dafür, ein Spiel, das Spaß macht, erstellen zu können. - Zum Balancen bin ich dieses Halbjahr noch nicht gekommen, da es noch viele Funktionen gibt, die wir einbauen wollen, ohne die man Level einfach nicht balancen kann. Bzw. würde jede neue Funktion die Game Balance durcheinanderwerfen und die Arbeit wäre umsonst. -

Die Levelauswahl zu erstellen hatte viel mehr etwas mit Design zu tun, als es Programmierarbeit war. Zum Einstieg habe ich mir also ein Video angeguckt: [[024] Unity JUMP AND RUN - Level Select - Scenen auswahl - Unity Panels - Deutsch/German - YouTube](https://www.youtube.com/watch?v=nKQd0cByPBY&t=514s&ab_channel=nobreakpoints) . Auch wenn dieses Video meines Wissens nach eine Levelauswahl eines 2D Spiels erklärt, hat es mir alles Nötige beigebracht. Ich erstellte eine neue Scene namens LevelAuswahl und fügte dieser als allererstes eine Canvas hinzu. Auf dieser Canvas Legte ich dann erst einmal 5 Buttons (Level 1 – Level 5) und ein Image, welches als Hintergrundbild für die Levelauswahl diene sollte (Quelle: [b12e43df8bc81a989cf8b296c64fe362.jpg (1920×1080)](https://cdn.gamedevmarket.net/wp-content/uploads/20191203164457/b12e43df8bc81a989cf8b296c64fe362.jpg)). Auf jeden Button legte ich ein Skript „LevelSelector“ welches bis zu diesem Zeitpunkt nichts weiter macht, als die entsprechende Scene zu öffnen, wenn man auf den Button klickt. Ich kann mir vorstellen, dass dieses Skript aber noch mehrere Änderungen durchlaufen wird, wenn wir zum Beispiel Features, wie Schwierigkeitsgrade oder das Unlocken von Leveln hinzufügen wollen.

Zu einem ordentlichen Spiel gehört auch, dass man Level beenden kann und gewinnen kann. Dafür sollte man im besten Fall, sobald man das Level beendet hat, wieder zur Levelauswahl oder wenigstens zu irgendeinem Menu zurück gehen können. Um die Funktion in den Skripts einzubauen, wollte ich erst einmal umsetzen, dass man nach dem Beenden des Levels wieder zur Levelauswahl zurückgeschickt wird. Ich brauchte aber erstmal ein richtiges Ende für ein Level. Momentan endete das Level eigentlich nicht wirklich. Das Einzige war, dass in der Konsole „Alle Wellen beendet“ steht, wenn alle Wellen gespawnt wurden. Wenn man aber alle Gegner besiegt hat, passiert nichts.

Um umzusetzen, dass man, sobald man den letzten Gegner erledigt hat, wieder zur Levelauswahl zurückgeschickt wird, musste ich erst einmal den letzten Gegner kennzeichnen. Dazu definierte ich in dem DamageTest Skript, welches auf jedem Gegner liegt, einen public Boolean isLast, der zu false gesetzt werden soll, wenn er nicht der letzte Gegner ist.

Um den letzten Gegner zu definieren, habe ich vorerst im BotsOnPath Skript die einzelnen Coroutinen mit einem Boolean isLast versehen. Also:

*IEnumerator SpawnGroupsInWave(WaveConfiguration waveConfig,* ***bool isLast****)*

Das Gleiche tat ich mit der Coroutine SpawnGroup und bei der Methode SpawnNewBot. „isLast“ wird auf true gesetzt, wenn es das letzte Mal ist, dass die Coroutine oder Methode von BotsOnPath ausgeführt wird, sodass ich nun die letzte Welle, die letzte Gruppe und den letzten Bot definieren kann. Bei SpawnNewBot wird dann schließlich das isLast auf das DamageTest Skript übertragen, sodass auch das DamageTest von Anfang an weiß, ob es dem letzten Gegner gehört. Damit das Spiel dann auch aufhört, habe ich in public void Die in DamageTest eine Abfrage erstellt, die über den SceneManager die Scene LevelAuswahl lädt, wenn isLast true ist.

Daraufhin habe ich festgestellt, dass das Ganze nur die halbe Miete war, denn nun kann man warten, bis alle Bots / Gegner gespawnt wurden und den zuletzt gespawnten einfach töten, um wieder zur Levelauswahl zu gelangen.

Die isLast Boolean habe ich noch im Skript gelassen, weil diese wahrscheinlich noch von Nutzen sein könnten. Jedoch musste ich nun, kurz bevor der Bot stirbt abfragen, ob noch andere Bots auf der Map sind. Also erstellte ich einen public bool IsOnlyEnemy in dem DamageTest skript, der true zurückgibt, wenn es in dieser Scene keine weiteren Gegner gibt und ließ diesen Boolean anstelle von IsLast abfragen. Das hat aber auch nicht ganz funktioniert, denn sobald zwischendurch im Spiel kein Gegner mehr am Leben war, ist man wieder zurück in die Levelauswahl gekommen, obwohl noch Wellen gespawnt werden sollten.

Darauffolgend setzte ich mir folgende Aufgaben, die dazu dienen sollten, dass IsOnlyEnemy nur abgefragt wird, wenn keine Gruppe, keine Welle und auch kein weiterer Bot mehr gespawnt werden soll:

* Die Wellen sollen erst zur nächsten Welle wechseln, sobald der letzte Bot stirbt, sodass zwei Wellen nicht auf einmal aktiv sein können
* Das BotsOnPath Skript so überarbeiten, dass man die Gruppen innerhalb der Wellen ebenso anpassen kann.
* IsOnlyEnemy erst bei der letzten Welle abfragen, weil zwei Wellen nicht auf einmal aktiv sein können.

Dass die nächste Welle erst startet, sobald die Bots der vorherigen alle gestorben sind, wollte ich zuerst über DamageTest machen, jedoch gab mir ChatGPT die Idee, in der Coroutine SpawnWaves einfach zu gucken, ob es noch Bots in der Scene gibt. Wie vermutet gibt es einen Befehl „yield return new **WaitUntil(() => condition**); den ich an dieser Stelle gut gebrauchen konnte.

Nun konnte ich auch mit ein paar Änderungen, wie isLast in BotsOnPath definiert wird, in DamageTest isLast als Zusatzbedingung neben IsOnlyEnemy für den Szenenwechsel einbauen.

Um das BotsOnPath Skript zu überarbeiten habe ich ChatGPT gefragt: *„wie kann ich die Gruppen so ändern, dass man diese auch so wie die Wellen einzeln anpassen kann?“* ChatGPT hat mir daraufhin eine Schritt für Schritt Anleitung gegeben, was ich ändern muss um meinen Wunsch umzusetzen. Ich musste hier und da noch ein Paar Anpassungen vornehmen, da ChatGPT meine Art, wie ich der letzten Welle isLast gebe, nicht gefallen hat. Zwei Stunden und zwei Fragen später hatte ich dann schlussendlich mein gewünschtes Ergebnis

### Das BotsOnPath Skript im Detail

Ich habe über das Projekt über immer wieder Sachen an dem BotsOnPath Skript geändert, sodass es inzwischen ein komplett anderes Skript ist, als es am Anfang war. Deshalb habe ich mich entschieden für einen besseren Überblick über die Funktionsweise des Bot Spawnens an dieser Stelle einmal möglichst detailliert den aktuellen Ablauf / Aufbau des BotsOnPath Skripts zu erklären.

Aktuell besteht das BotsOnPath Skript aus vier Klassen: WaveConfiguration, GroupConfiguration, BotConfiguration und BotsOnPath, sozusagen die Hauptklasse. Die drei Configuration Klassen sind „Serializable“, was ermöglicht, die einzelnen Klassen im Inspector zu sehen und anzupassen.

In der Klasse WaveConfiguration findet man nur eine Zeile, die ein Array von GroupConfiguration darstellt. Dadurch kann man im Inspector in jeder neuen Welle angeben, wie viele Gruppen man in dieser Welle haben will.

In der Klasse GroupConfiguration werden neben einem Array von BotConfiguration, welches hier wieder ermöglicht, im Inspector die Anzahl der Bots in jeder Gruppe einzeln festzulegen, zwei floats. „tillNextGroup“, welches die Zeit angeben soll, die zwischen zwei Gruppen gewartet wird und „tillNextBot“ – selbsterklärend.

In der Klasse BotConfiguration wird einmal ein public Gameobject botPrefab definiert, welches die Art von Gegner, der gespawnt werden soll angibt und ein Intiger „timesBot“, der angeben soll, wie oft diese Art von Bot hintereinander gespawnt werden soll.

In der Klasse BotsOnPath fängt es an mit einem Array von WaveConfiguration, sodass man entscheiden kann, wie viele Wellen es in diesem Level geben soll, einer Referenz zu dem Pfad vom PathCreator und einen float „speedMuliplier“, der die Geschwindigkeit der Bots erhöhen oder verringern soll. Es geht weiter mit der Start Methode, in der die Coroutine SpawnWaves startet. Insgesamt gibt es in dem BotsOnPath Skript vier Coroutinen: SpawnWaves, SpawnGroupsInWave; SpawnGroup, MoveBotAlongPath und eine Methode: SpawnNewBot.

Die Coroutine MoveBotAlongPath kümmert sich darum, den Gegner entlang des Pfades Laufen zu lassen und zwar so: Aus dem Parameter bekommt die Routine ein GameObject namens enemy. Zunächst wird ein float distanceTravelled deklariert, der wichtig für die Funktion des PathCreators ist. Danach nimmt sich MoveBotAlongPath das DamageTest Skript von enemy und tut folgendes solange, wie enemy lebt (damageTest.isAlive): Sie berechnet die distanceTravelled anhand des Speeds des Bots selbst, des speedMulitpliers und der vergangenen Zeit seit dem letzten Frame und setzt somit die Position und Rotation von enemy an der Distance an dem PathCreator (So funktioniert der PathCreator). Sobald enemy nicht mehr „Alive“ ist, endet die Coroutine.

Die Methode SpawnNewBot erhält als Parameter ein GameObject botPrefab und einen Boolean isLast und beginnt damit, das botPrefab an der Position null des PathCreators zu instanziieren und sich dessen damageTest Skript zu nehmen. „damageTest“ besitzt ebenfalls einen Boolean isLast. Dieser wird mithilfe dieser Methode true gesetzt, wenn der Parameter isLast ebenfalls true ist und andersrum. Danach startet SpawnNewBot die Coroutine MoveBotAlongPath mit dem GamObject, was gerade instanziiert wurde.

Eine Ebene höher beginnt die Zuständigkeit der Coroutine SpawnGroup. SpawnGroup hat besitzt zwei Parameter: GroupConfiguration groupConfig, wodurch die Methode Zugang zu den Variablen der entsprechenden GroupConfiguration bekommt, und einen Boolean isLast. Für jede BotConfiguration in der entsprechenden GroupConfiguration führt sie für timesBot von der BotConfiguration SpawnNewBot mit dem prefab von der BotConfiguration aus, gibt SpawnNewBot den isLast boolean weiter und wartet für tillNextBot Sekunden.

Eine Ebene höher kümmert sich SpawnGroupsInWave um das Spawnen der Gruppen. Diese hat wieder zwei Parameter: WaveConfiguration waveConfig und den Boolean isLast. Für die Anzahl an Gruppen in der entsprechenden Welle (Diese wird als Parameter gegeben) geht sie alle Gruppen der Welle durch und startet jeweils die Coroutine SpawnGroup und gibt ihnen den Boolean isLast weiter. Als letztes wartet die Routine für tillNextGroup Sekunden, von der momentanen Gruppe, bis der nächste Durchlauf der For-Schleife beginnt. Nach der For-Schleife endet die Routine.

In der höchsten Ebene kümmert sich die Coroutine SpawnWaves um das starten und enden der Wellen. Diese Routine besteht momentan nur aus einer For-schleife, die so oft ausgeführt wird, wie es Wellen gibt / geben soll. Jeden durchlauf führt sie SpawnGroupsInWave, der momentanen Welle aus. So werden genau so viele Gruppen gespawnt, wie die aktuelle Welle erhalten soll. Beim letzten durchlauf der For-Schleife wird für den isLast Boolean von SpawnGroupsInWave true angegeben. Das heißt, der isLast Boolean gibt an, ob die Aktuelle Gruppe oder der aktuelle Bot der letzten Welle angehörig ist. Am Ende der For-Schleife wird so lange gewartet, bis es keine Gegner mehr in dieser Szene gibt, bevor die For-Schleife erneut durchläuft. So Startet eine nächste Welle erst, wenn alle Gegner getötet wurden oder gegen die Base gelaufen sind.

Der Boolean isLast diente ursprünglich dazu, die nächste Welle erst zu starten, wenn die Gegner der vorherigen Welle alle getötet wurden. Momentan dient er aber lediglich dazu, dass man nach der letzten Welle wieder zur Levelauswahl zurückkehrt. Diese Funktion ist aber im DamageTest Skript eingebaut.

## Erstes Gemeinsames Ziel (Jaron & Tim): Spawning überarbeiten

Jaron: Tim ist gerade dabei mithilfe von Blender neue Türme zu Designen

Tim: Ein wichtiger Aspekt den wir während der Programmierung des SpawnCancelRadius‘ vergessen hatten, war zu Überprüfen, ob auch der Turm, der gespawnt werden soll selbst einen höheren spawnCancelRadius besitzt, als sein Abstand zum nächstgelegenen Turm. Bisher hat der Turm bei dem Platzierprozess nur überprüft, ob es einen bereits platzierten Turm gibt, der einen höheren spawnCancelRadius besitzt, als dessen Abstand zum Turm, der gerade vom Spieler gezogen wird und dementsprechend isPositionValid false oder true gesetzt. Diesen Fehler wollte ich Fixen, also habe ich in dem preview Skript in der Abfrage isPositionValid einen float „ownCancelRadius“ zugefügt, welcher als SpawnCancelRadius vom previewObject definiert wird. Dieser wird dann neben dem tower.spawnCancelRadius mit dem Abstand verglichen, sodass isPositionValid false ist, auch wenn ein Turm innerhalb des SpawnCancelRadius‘ von dem Turm steht, der gerade vom Spieler platziert wird.

# Notizen

Wichtige Fixes: man hat gewonnen, sobald der letzte Gegner erschossen wurde.

Der Turm wird nicht gespawnt, wenn man mit der UI interagiert

Bei Spawncancelradius muss der Spawncancelradius des aktuell zu spawnenden Turms beachtet werden. Außerdem sollte beides angezeigt werden.

Für Reflexion am Ende: Vor allem am Anfang (nach der Abgabe von Q1) schwer strukturiert an einzelnen Zielen weiter zu arbeiten. Liegt daran, dass es zu dem Zeitpunkt viele kleine Ziele gibt: Neue Türme, Level ausbalancieren, Animationen in das Spiel einfügen, das Spawning funktionell und auch visuell spielbar gestalten. Ende der Tutorials auf Youtube, für Probleme findet man nicht mehr so einfach Tutorials, wie zum Beispiel die Funktion von Childskripts.

## Plan für Q2

Beide: Story 🡪 Story Board

Jaron:

1. Upgrading von Türmen
   1. Skript zum Finden von den Türmen
   2. Dynamische Erstellung von Buttons
   3. Hoch leveln
   4. Kosten (CreditSystem)
2. UI Scheiß
   1. Game Over- & Win Screen & Main Menu …
3. SpawnCancelRadius anzeigen lassen
4. Q2 Dokumentation

Tim:

1. Neue Türme und Gegner
   1. Prefab überarbeiten (Turm Kanone muss in positive X-Richtung zeigen)
   2. Mit Blender auseinandersetzen
2. LevelSystem
   1. Speicher System und Level freischalten
3. Q2 Dokumentation

## Erst in Q3

Level balancen, UI design komplett neu (Auch Digital Malen), Umgebung Design, Sounds (Jaron),

## Story

**5. Der Exodus der Menschheit**

**Story:**  
Die Erde ist am Rande der Zerstörung, und die Menschheit versucht, mit riesigen Raumschiffen zu entkommen. Doch aggressive außerirdische Jäger wollen die letzten Überlebenden auslöschen. Dein Job ist es, die Startanlagen der Raumschiffe so lange wie möglich zu verteidigen, bis der letzte Transporter gestartet ist.

**Twist:**  
Die Feinde werden immer stärker, je näher der Abflug rückt. Deine Türme können geopfert werden, um die Schiffe schneller zu starten, aber das schwächt deine Verteidigung. Wie viele Menschen kannst du retten?