Dokumentation-

Spielentwicklung in Unity

Tower Defense Ultimate

Jaron Liskow und Tim Fischer

Seminarkurs 11 Bindestrichinformatik

Inhalt

[Ziele für das Spiel: 3](#_Toc183251030)

[Grobe Ziele: 3](#_Toc183251031)

[Erste Anfänge 4](#_Toc183251032)

[Erster grober Entwurf des Projekts. 4](#_Toc183251033)

[Tim Vorkenntnisse: 5](#_Toc183251034)

[Jaron Vorkenntnisse: 5](#_Toc183251035)

[Erster Grober Entwurf & Projektideen: 5](#_Toc183251036)

[Ziele: 5](#_Toc183251037)

[Schritte (Planung & Arbeit): 6](#_Toc183251038)

[Entwicklungsumgebung: 6](#_Toc183251039)

[Versionsverwaltung/Zusammenarbeit: 6](#_Toc183251040)

[Erstes Ziel (Tim): 6](#_Toc183251041)

[Einarbeitung 6](#_Toc183251042)

[Vorgehensweise 7](#_Toc183251043)

[Was macht der Code? 7](#_Toc183251044)

[Erstes Ziel (Jaron): 8](#_Toc183251045)

[Vorbereitung: 8](#_Toc183251046)

[Zweites Ziel (Tim) 8](#_Toc183251047)

[Teil 1 - Leben von Gegnern 8](#_Toc183251048)

[Teil 2 - Schaden an der Base 10](#_Toc183251049)

[Drittes Ziel (Tim) 11](#_Toc183251050)

[Probleme 11](#_Toc183251051)

[Tim: 11](#_Toc183251052)

[Jaron: 12](#_Toc183251053)

# Ziele für das Spiel:

## Grobe Ziele:

* 3D Umgebung
* Charakter erstellen
* Charakter soll sich bewegen können
* Map mit Pfad für Monster/Gegner
* Monster sollen spawnen und dem Pfad folgen
* Türme können platziert werden
* Türme fügen den Monstern Schaden zu (Schadensystem)
* Base mit begrenzter Anzahl an Leben
* Wenn Monster in die Base, ans Ende des Pfads, kommen, nimmt die Base Schaden
* Türme können verbessert werden
* Verschiedene Arten von Monstern/Türme/Level

Anfang:

* Projekt Towerdefense Spiel
* Unity installieren
* Github installieren (Zusammenarbeit)
* Jar0n :
  + Character Movement
  + Third-Person Kamera
  + Animation Character
  + Basic UI
  + (Placement Logic)
  + (Erweiterte UI)
* Tim:
  + First Person Kamera (Inaktiv)
  + Enemy Movement
  + (Enemy Animation)
  + Schadensystem – BaseHealth und EnemyHealth bisher
  + (Tower Behaviour)

# 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++.

# Erster Grober Entwurf & Projektideen:

Tower Defense Spiel in einer 3rd Person – Perspektive

# Ziele:

* Obligatorisch:
  + 3rd Person (Jaron)
    - Spielfigur
    - 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
  + UI
    - Startbildschirm & „Endbildschirm“ (Game Over)
    - Anzeige der Leben, verfügbarer Münzen, Türme etc.
  + Weitere Level
    - Erweiterte UI
      * Levelauswahl
    - Verschiedene Gegner und Türme mit Unterschiedlicher Animation und unterschiedlichen Eigenschaften
    - Weitere Pfade in einem Level
* Optional:
  + 3dimensionale Bewegung des Spielers
    - Hoch – runter Bewegung mit Maus oder Tastatur
  + Level mit 3dimensionaler Bewegung der Gegner und Platzierung der Türme

# Schritte (Planung & Arbeit):

Entwicklungsumgebung:

Game Engine, weil einfacher und viele Tools bereitgestellt und mehr Spaß am Programmieren bringt. Von Unreal Engine zu Unity gewechselt, weil Unity bekannter ist und es mehr Tutorials gibt und dadurch anfängerfreundlicher ist. Mit Unity programmieren wir auf C# und mit Visual Studio (2022). Verwendung von GitHub für das Spiel und die Dokumentation.

## 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.

## Erstes Ziel (Tim):

### 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 Befehltypen 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):

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

## Zweites Ziel (Tim)

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.

## Drittes Ziel (Tim)

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.

# 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
* Generelle Nutzung von ChatGPT
  + Zum Einstieg war ChatGPT für einfache Skript sehr Hilfreich und konnte mir konkrete Fragen zu bestimmten Funktionen in Unity gut erklären. Je fortgeschrittener jedoch die Skripte waren, die ich schreiben wollte, desto mehr Fehler machte ChatGPT (nicht im Syntax, aber in der Logik) und ich musste mehr und mehr Fragen stellen, um das Skript, was ChatGPT mir gab zu verstehen und dementsprechend zu verbessern. Dennoch nehme ich ChatGPT bis jetzt als beste Möglichkeit, sich mit Unity auseinander zu setzen, wahr.

## Jaron: