Inhaltsverzeichnis

[Einleitung 2](#_Toc169079557)

[Hintergrund des Projekts 2](#_Toc169079558)

[Zweck und Ziel des Projekts 2](#_Toc169079559)

[Team und Thema 2](#_Toc169079560)

[Team 2](#_Toc169079561)

[Thema 2](#_Toc169079562)

[Realisierung der Teilbereiche 2](#_Toc169079563)

[Player 2](#_Toc169079564)

[Collectable 2](#_Toc169079565)

[Character design 3](#_Toc169079566)

[Kartendesign 3](#_Toc169079567)

[Kartengeneration 3](#_Toc169079568)

[Hauptmenü 3](#_Toc169079569)

[Gegner 3](#_Toc169079570)

[Boss 3](#_Toc169079571)

# Einleitung

## Hintergrund des Projekts

Am Anfang konnte sich ein Thema ausgesucht werden, welches mit der Übung oder der Vorlesung zu tun hat. Die wesentlichen Themen der Vorlesung wären zum Beispiel Text, Bild, Audio und Video. Bei den Beispielen in der Übung wurde ein Spiel gezeigt, wodurch die Entscheidung, ein Spiel zu programmieren, getroffen wurde.

## Zweck und Ziel des Projekts

Mit dem Projekt soll das Team in den jeweils favorisierten Bereichen einen Leistungsaufwand von rund 20 Stunden umsetzen und sich dabei neue Fähigkeiten und Techniken aneignen. Im Fall dieses Videospiels sind dies das Design, die Kartengeneration, der Spieler und die Gegner. Um ein lauffähiges Spiel entwickeln zu können, musste das Team gut zusammenarbeiten, da jede dieser 4 Kategorien ineinandergreifen und zum Teil voneinander abhängig sind. Wenn dies nicht der Fall sein sollte, könnten ungewollte Bugs auftreten, das Thema des Spiels nicht zu der eigentlichen Spielerfahrung passen, die Spielerfahrung im Allgemeinen nicht gut sein oder das Spiel erst gar nicht funktionieren.

# Team und Thema

## Team

Unser Projektteam besteht aus vier engagierten Mitgliedern, die jeweils spezifische Bereiche des Projekts übernommen haben, um ein erfolgreiches Ergebnis zu erzielen. Je nach Teammitglied wird verschiedene Programmiererfahrung mitgebracht, jedoch wurde am Anfang des Projektes die Aufgabenteilung so gewählt, dass jedes Teammitglied die Bereiche des Projektes bekommt, die favorisiert wurden.

1. Jan Feldmayer – Player & Item Entwicklung
2. Jonas – Character- & Kartendesign
3. Mathias Kröpfl – Kartengeneration & Menu
4. Wasgen – Gegner & Boss Entwicklung

## Thema und Projektbeschreibung

“Pixel Purge” ist ein spannendes Top-Down 2D-Roguelike-Shooter-Spiel, das in Unity entwickelt wurde. Das Spiel wurde mit einem Singleplayermodus implementiert, in dem der Spieler gegen eine Vielzahl von Feinden kämpfen und versucht sich bis zum Boss durchzuschlagen. Wenn der Boss besiegt wurde, ist das Spiel beendet und man wird zurück ins Hauptmenu geladen. Durch die Tötung der Gegner in den verschiedenen Räumen, können Gegner kleine Health-Items fallen lassen, welche dem Spieler helfen dessen Leben wieder herzustellen. Das Grundprinzip des Spiels ist, dass der Spieler mittels Maus und Tastatur einen Charakter durch ein Level steuert, indem ein Krieg zwischen Kreisen und Polygonen herrscht. Inmitten dieses Krieges ist der Hauptcharakter „Pixel-Bro“, der Anführer der Kreise. Dieser versucht sich durch die Territorien des Feindes zu Kämpfen und die Allianz der Polygone zu zerstören. Um dies zu schaffen, muss der Anführer der Allianz, „Pixel-Dominator“, bezwungen werden. Dieser ist jedoch in seinem Geheimversteck, welches sich nur mithilfe aller Schlüssel aufsperren lässt.

# Realisierung der Teilbereiche

## Player

Ein Bild, das Kreis, Screenshot enthält.

Automatisch generierte BeschreibungDer Spieler ist die Hauptfigur und steht im Zentrum des Spiels steht. Der Player besteht aus einem Körper (Kreis) und ein kleineres Rechteck, welches vor dem Körper platziert wird und eine Waffe simulieren soll. Dieser verfügt über einige Fähigkeiten, welche mittels folgender Funktionen und Skripts implementiert wurden.

Abbildung 1: Player

**PlayerController**: Der PlayerController gibt die Möglichkeit den Spieler mittels Tastatur und Maus zu steuern. Dabei wird das integrierte Input-System von Unity (legt fest mittels welcher Tasten welche Aktionen gefeuert werden) mit einem Skript, welches die physikalischen Eigenschaften (Beschleunigung, Geschwindigkeit, Rotationsgeschwindigkeit) setzt, verbunden. Dabei gibt es die Möglichkeit den Spieler in Richtung der Tastatur oder in Richtung der Maus zielen zu lassen.

Abbildung 2: Input-System am Player setzen

Abbildung 3: Unity-Input System

Ein Bild, das gelb, Kreis, Screenshot, Farbigkeit enthält.

Automatisch generierte BeschreibungEin Bild, das Text, Screenshot, Software, Multimedia-Software enthält.

Automatisch generierte BeschreibungEin Bild, das Text, Screenshot, Software, Schrift enthält.

Automatisch generierte Beschreibung**PlayerShoot**: Dieses Skript erlaubt den Spieler in die gezielte Richtung ein Bullet-Objekt mit einer angegeben Geschwindigkeit zu feuern. Zusätzlich kann man regulieren, wie schnell in Folge man Bullets abfeuern kann.

**Bullet**: Bullet ist ein Objekt, welches einem Projektil einer Waffe entsprechen soll. Dieses Objekt wird mittels dem PlayerShoot (eine Funktion oberhalb beschrieben) gefeuert. Im Bullet Skript selbst wird geregelt was passieren soll, wenn die Bullet auf ein anderes Objekt stoßt. Ist das kolierende Objekt der Spieler selber oder ein Gegner wird mittels der HealtContorller-Funktion TakeDamage() diesem Leben abgezogen. Ist das kolierende Objekt eine z.B. Wand wird die Bullet beim Aufprall zerstört.

Abbildung 4: Player shooting

**HealthContolller**: Der Spieler verfügt über eine Lebenspool und verliert, wenn er von einem Gegner getroffen wird, eine gewisse Menge. Hat der Spieler keine Leben mehr übrig, wird jegliche Spieleraktionen gesperrt und es ist Game Over. Diese Funktion wird über einen universellen HealthController gesteuert und ist im Punkt Health System genauer erklärt.

## Health System

**HealthController**:  
Die Funktionen, welche es ermöglichen das der Spieler/ein Gegner Leben verliert und erhaltet, wird mittels einen universellen HealthController Skript gesteuert. Dieses Skript befindet sich auf jedem Objekt, welches über Leben verfügen soll (Spieler, Gegner) und besteht grundlegend aus zwei Funktionen TakeDamge() und AddHealth() sowie drei UnityEvents OnDied(), OnDamaged() und OnHealthChanged(). In den zwei Funktionen werden mit Logik die Events gefeuert um das gewünscht Verhalten zu erreichen.

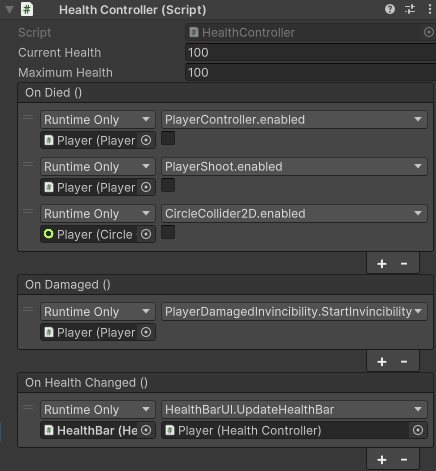


Abbildung 5: HealthController Player

**HealthBar**:  
Das Leben des Spielers wird im UI im linken oberen Eck mittels länglichen horizontalen Rechtecks dargestellt. Diese sogenannte HealthBar besteht aus einem schwarzen Hintergrund und einem roten Vordergrund (stellt das Leben dar). Das Vordergrundbild verfügt über die Eigenschaft Fill Amount, welche von 0 – 1 die Breite des Vordergrundbilds bestimmt. Durch Zusammenspiel aus den Funktionen des Skripts HealthController, HealtBarUI und dem HealtController Event OnHealthChanged zeigt die HealthBar ständig die Menge des aktelles Lebenspools des Spielers an.



Abbildung 6: HealthBar voll



Abbildung 7: HealthBar halb voll

**Invincibility Controller**:  
Der InvincibilityController ist wie der HealthController ein universelles Skript, welches für Spieler und Gegner benutzt wird. Wenn der Spieler/Gegner Schaden erhält wird das Event OnDamaged() des HealthControllers gefeuert und die Funktion StartInvincibility() des InvincibilityController ausgeführt. Dadurch wird der Spieler/Gegner für eine angegebene Zeit unverwundbar. Dieses Verhalten ist für den Spielverlauf notwendig, da sonst der Spieler bei Berühren einen Gegner pro Frame Schaden erleidet und somit „sofort“ Tod ist. Dieser Effekt des kurzen „unsterblich seines“ wird visuell durch einen sogenannten Flash Sprit (wechseln zwischen voller Objektfarben und transparenterer Version der Farbe) erzielt.

Abbildung 9: Gegner Flash Sprite



Abbildung 8: Player Flash Sprite



## Collectable

Collectables ist ein Überbegriff für alle Objekte, welche von einem Spieler gesammelt werden können (z.B. HealthPack, Keys, etc.).  
Für die Umsetzung wurde dafür ein Prefab Collectable mit dem gleichnamigen Skript erstellt und aus diesem die verschiedenen Prefab Variant wie z.B. Health\_Collectable & Key\_Collectable. Zusätzlich wurde ein Interface ICollectableBehaviour erstellt welches die Methode OnCollected (GameObject player) enthält, welche dann in den Klassen z.B. HealthCollectableBahavour konkret implementiert wird.   
Der HealthCollectableSpawner enthält einen Liste an GameObjects(Objekte welche man spawnen will) und ist wie der Name schon sagt dafür zuständig, dass die Collectables an eine übergebene Position erscheinen. Alle Collectables werden nach dem „Aufheben“ zerstört, sodass man diese nicht unendlich oft wiederverwenden kann.

**HealthCollectable(HealtPack)**:

HealthCollectables sind Collectabels, welche beim Aufheben mittels HealthController dem Spieler Leben zurückgeben. Das Skript EnemyCollectableDrop ist dann zuständig, dass besiegte Gegner mit einer angegebenen Wahrscheinlichkeit diese Collectables an die Stelle, wo sie besiegt wurden, fallen lassen.  
  
**KeyCollectables**:



Abbildung 6: Player with Health\_Collectable

KeyCollectables sind Collectables welche beim Aufheben die Anzahl der gesammelten Schlüssel (Keys) erhöht. Diese Schlüssel werden beim Spielstart in zufälligen Räumen auf der Spielfläche verteilt. Wird die benötigte Menge an Schlüssel gesammelt, wird der Boss-Raum geöffnet. Um die Anzahl der bereits gesammelten Schlüssel verfolgen zu können, gibt es im UI im rechten oberen Eck einen Zähler.



Abbildung 7: Player mit Key\_Collectable

# Quellen