

Projektdokumentation



Ein Tower-Defense-Game

ITIG 2022/23

Team “DieGebeetsBrüder”

Lucas Hengelhaupt

Henning Venhorst

Eric Kruse

Henrik Timpel

1. Installationshinweise & Systemanforderungen	3
1.1 Installationshinweis	3
1.2 Systemanforderungen	3
2. Bedienungsanleitung	3
2.1 Steuerung	3
2.2 Spielregeln	3
3. Spielkonzept	4
3.1 Spielidee	4
3.2 Story	4
3.3 Spielverlauf	4
3.4 Game Feature & Unique Selling Point	5
3.4.1 "Eye of Doom" & Gargoyle	5
3.4.2 Endlos-Modus	5
3.4.3 Erstellung des Turm-Platzierungs-Layouts	5
3.5 Umsetzung	7
3.5.1 Level Design	7
3.5.2 Grafik & Ästhetik	7
4. Assets	8
4.1 Eigene Assets	8
4.1.1 Modelle	8
4.1.2 Sprites	11
4.1.3 Scripts	12
4.2 Fremde Assets	28
4.2.1 Unity-Packages	28
4.2.2 Musik	28
4.2.3 Soundeffekte	28
4.2.4 Schriftart	29
5. Weitere Inhalte	29
5.1 Team & Arbeitsaufteilung	29
5.2 Vorgehensweise	30
5.3 Dev-Shortcuts	30
5.4 Bugs & Besonderheiten	30
5.5 Blick in die Zukunft	30
6. Anhang	31
6.1 Glossar	31
6.2 Links	31

1. Installationshinweise & Systemanforderungen

1.1 Installationshinweis

Build ZIP-Archiv (Google Drive): [AnnoyedFiends_Build.zip](#)

Das Build ZIP-Archiv muss zuerst entpackt werden, danach kann die enthaltene “Annoyed Fiends.exe” gestartet werden.

Die Besonderheiten des Builds sind unter “5.3 Dev-Shortcuts” und “5.4 Bugs & Besonderheiten” vermerkt.

1.2 Systemanforderungen

	Minimum	Empfohlen
Betriebssystem	Windows 10	Windows 10
CPU	AMD Ryzen 3	AMD Ryzen 5
GPU	Radeon Vega Mobile GFX	NVIDIA GTX 970
RAM	8 GB	8 GB
Speicherplatz	350 MB	350 MB

2. Bedienungsanleitung

2.1 Steuerung

Taste	Funktion
Linke Maustaste	<ul style="list-style-type: none">- Turm auswählen / platzieren- Schießen (wenn im “Eye of Doom”)
Leertaste	<ul style="list-style-type: none">- Wechsel von Vogelperspektive in das “Eye of Doom”- Wechsel aus dem “Eye of Doom” zurück in die Vogelperspektive
ESC	<ul style="list-style-type: none">- Pausen-Menü

2.2 Spielregeln

Durch das Platzieren von Türmen werden Wellen von Gegnern abgewehrt. Die Türme und das Verbessern von Türmen kostet Geld. Eine Spielpartie endet mit einem Sieg, wenn die vorgesetzte Anzahl an Gegnerwellen abgewehrt wurde. Die Spielpartie endet mit einer Niederlage, wenn alle Leben des Spielers, durch Überschreiten des Ziels der Gegner, aufgebraucht sind.

3. Spielkonzept

3.1 Spielidee

Die Idee war es, ein Tower Defense Spiel zu entwickeln, bei dem die Unterwelt mal nicht die Bösen sind, sondern sich nur verteidigen. So stehen diesmal die Dorfbewohner als die Bösen da. Dieses Setting soll etwas Abwechslung bringen und so die Aufmerksamkeit des Spielers auf sich ziehen.

3.2 Story

Im örtlichen Dungeon werden die Teufel und Dämonen von einer Horde Dorfbewohner überfallen. Diese Dorfbewohner wollen die kostbaren Schätze des Dungeons und diese für sich beanspruchen.

Der Spieler hat die Aufgabe, die Dorfbewohner davon abzuhalten, alle Schätze zu stehlen, damit am Abend wie geplant das Pokerturnier stattfinden kann.

3.3 Spielverlauf

Der Spieler erhält zu Beginn ein Startkapital, welches er nutzen kann, um verschiedene Türme zu platzieren und zu upgraden. Hierzu hat er folgende Auswahlmöglichkeiten:

	Schaden	Schadensart	Besonderheit
Archer	35	1 Gegner	-
Devil	35	Flächenschaden	-
Gargoyle	-	-	blockiert den Weg der Gegner für 5s
“Eye of Doom”	34	1 Gegner	kann direkt vom Spieler gesteuert werden

Mit dem Betätigen des “Start Round” - Button kann eine Welle gestartet werden. In dieser versuchen die Dorfbewohner, über den vorgesetzten Pfad das Ende zu erreichen. Sollte dies gelingen, verliert der Spieler ein Leben . Bei dem Besiegen eines Gegners oder dem Abschluss einer Runde erhält der Spieler Geld, welches er wieder zur Verbesserung seiner Verteidigung einsetzen kann. Wahlweise kann er während einer Runde auch das “Eye of Doom” betreten, um selbst aktiv an der Verteidigung teilzunehmen.

Dabei gibt es drei verschiedene Gegner-Typen denen der Spieler begegnet, welche allmählich mit dem Rundenfortschritt freigeschaltet werden:

	Leben		Geschwindigkeit		Geld bei Tod
Farmer	normal	(100)	normal	(10)	10
Dorfschranzen	wenig	(50)	schnell	(20)	20
Tank	viel	(1000)	langsam	(5)	50

3.4 Game Feature & Unique Selling Point

3.4.1 “Eye of Doom” & Gargoyle

Game Features sind das Platzieren und verbessern von Türmen, das Blockieren des Weges mit dem Gargoyle, sowie das Schießen mit dem “Eye of Doom”. Das “Eye of Doom” sowie der Gargoyle sind dabei die Unique Selling Points, da die Spielmechanik der aktiven Selbstbeteiligung eine ungewöhnliche Spielmechanik für das Genre der Tower Defenses ist, sowie das Blockieren der Gegner die Entwicklung völlig neuer Strategien ermöglicht.

3.4.2 Endlos-Modus

Der Spieler hat außerdem die Möglichkeit, nach erfolgreichem Abschluss eines Levels dieses in einem Endlos-Modus weiterzuspielen und somit auf Highscore-Jagd zu gehen.

3.4.3 Erstellung des Turm-Platzierungs-Layouts

Wir haben nach einer Lösung gesucht, womit wir auf dem Grid die Plätze speichern können, an denen Türme platziert werden dürfen und an welchen nicht, ohne uns im Code mit selbstgeschriebenen Arrays oder Ähnlichem rumschlagen zu müssen. Herausfordernd war hierbei, dass die Türme “Archer” und “Devil” nur auf den erhöhten Positionen neben dem Gegner-Weg platziert werden sollten, der Gargoyle hingegen nur auf dem Weg.

Gelöst haben wir es dadurch, dass wir unter dem “GridBuildingSystem” ein “Tile Layout” erstellt haben, in dem man sehr einfach diese Positionen im Unity-Editor festlegen kann.

Dabei gibt es zwei Tile-Arten: die mit dem Tag “Placeable” (grün, für Archer & Devil) sowie “Path” (grau, für den Gargoyle) (Abbildung 1). Daraus werden beim Laden des Levels an diesen Stellen Platzierungs-Felder generiert, die das “GridBuildingSystem” je nach ausgewähltem Turm an- und ausschalten kann (Abbildung 2 & 3).

An allen Stellen, die nicht im “Tile Layout” definiert sind, kann kein Turm platziert werden. Somit kann man in wenigen Minuten und ohne großen Aufwand für jedes Level das Turm-Layout definieren.

Abbildung 1 (“Tile Layout” im Editor)



Abbildung 2 (im Spiel, Archer ausgewählt)



Abbildung 3 (im Spiel, Gargoyle ausgewählt)



Auszug aus “GridBuildingSystem”

```
private void InitializeGridTiles() {
    ...
    // load Tile-Layout from Map into array
    GameObject[] pathTilesFromLayout =
    GameObject.FindGameObjectsWithTag("Path");

    InitializeGridTileArray(pathTilesFromLayout, GridTile.TileType.Path, ... );
}

private void InitializeGridTileArray( GameObject[] tileArray,
    GridTile.TileType type, ... ) {

    foreach (GameObject tile in tileArray) {
        // convert World-Coordinates in Grid-Coordinates
        var coordinates = grid.GetXZ(tile.transform.position);
        // create the Tile and save it in an array for placement-checking
        gridTiles[coordinates.x, coordinates.z] =
            GridTile.Create( grid.GetWorldPosition(coordinates.x,
coordinates.z),
                            ..., type );
    }
}
```

3.5 Umsetzung

Annoyed Fiends wurde in C# mit Unity 21.3.12f1 entwickelt.

Die selbst erstellten Assets wurden dafür in Blender kreiert.

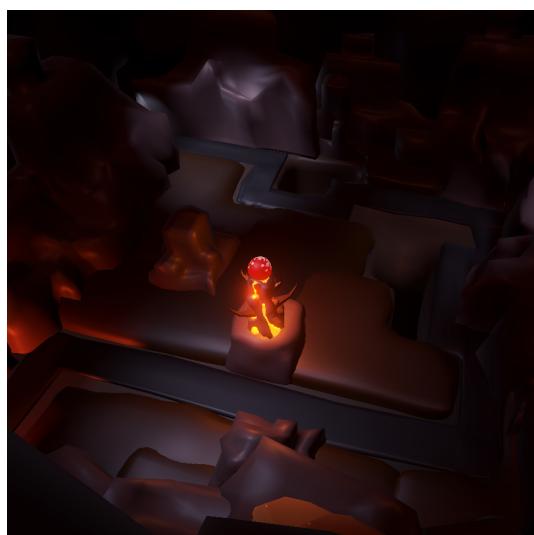
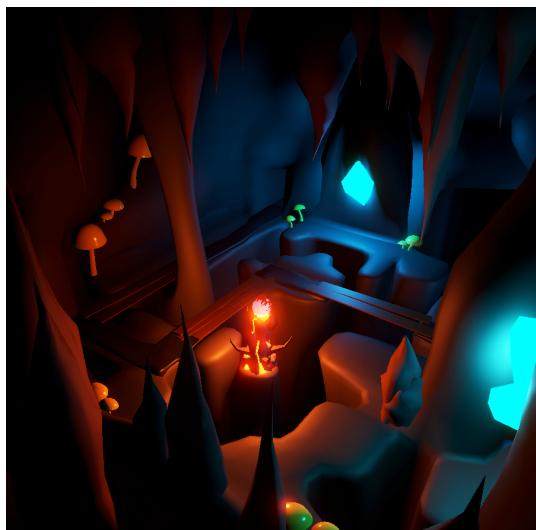
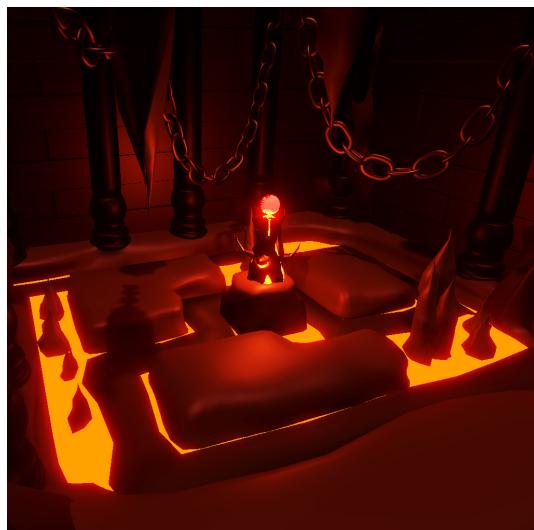
Die eigens erstellten Sprites wurden in Inkscape und Photoshop erstellt.

3.5.1 Level Design

Das Level Design orientiert sich an dem Setting eines Dungeons. Dabei soll das erste Level den tiefsten Bereich des Dungeons darstellen. Mit voranschreitenden Leveln sollen die Dorfbewohner immer mehr heraus getrieben werden. Dadurch stellt das dritte Level den Eingang des Dungeons dar.

3.5.2 Grafik & Ästhetik

Ein stetiger Wechsel aus flüssigen und harten, kantigen Formen verleiht dem Spiel seine eigentümliche Optik. Während die Modelle stilisiert sind, setzen Form- und Lichtkontraste Blickpunkte und Spannung. Kleine Details in den stilisierten Formen und Maps lassen die Welt lebendig wirken.



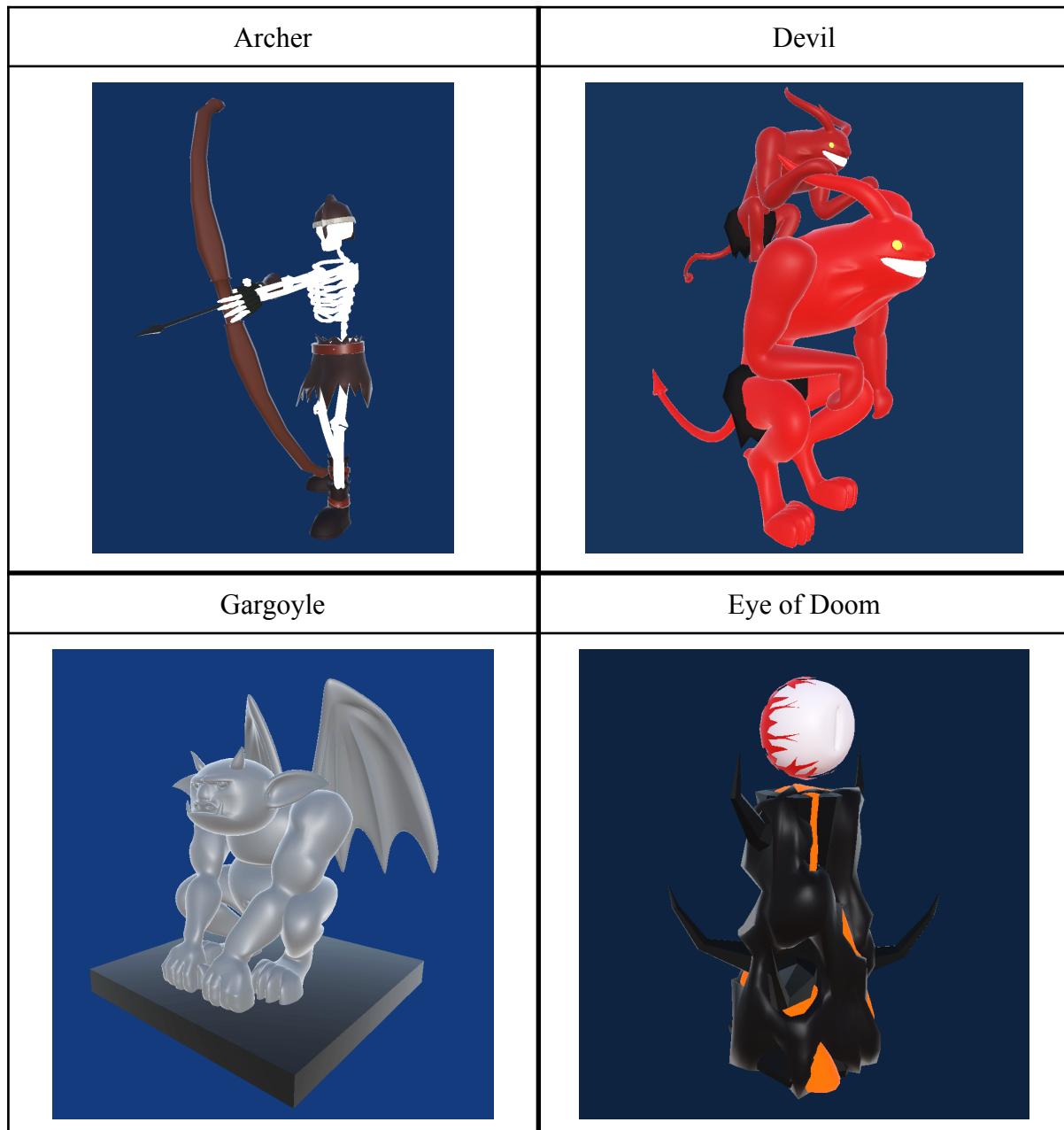
4. Assets

4.1 Eigene Assets

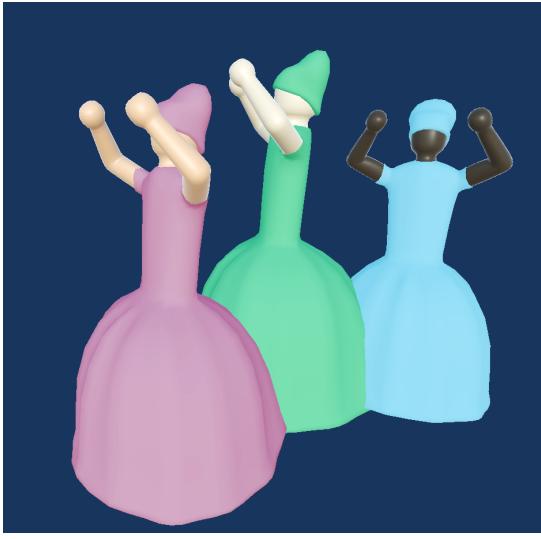
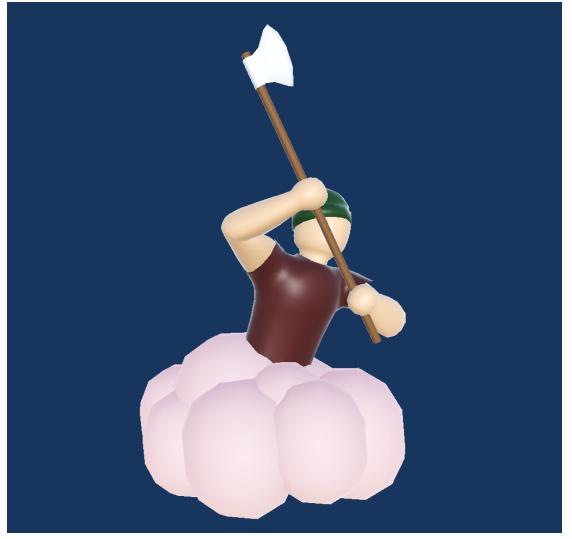
4.1.1 Modelle

Alle 3D-Modelle wurden mit Hilfe von Blender erstellt.

Türme

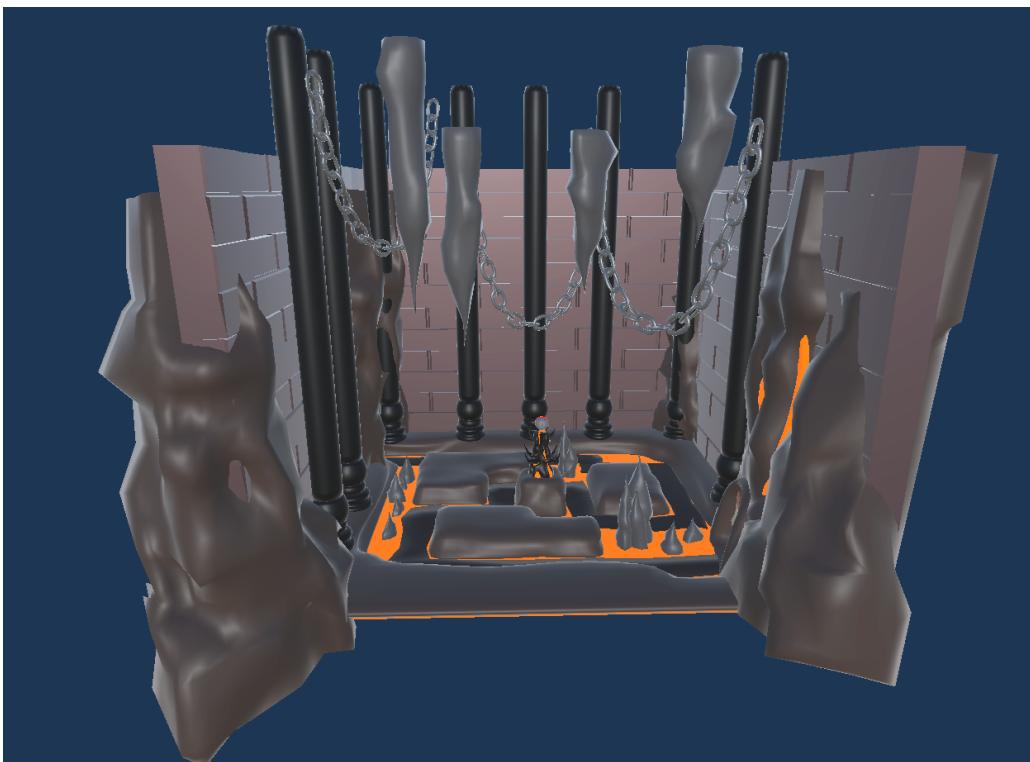


Gegner

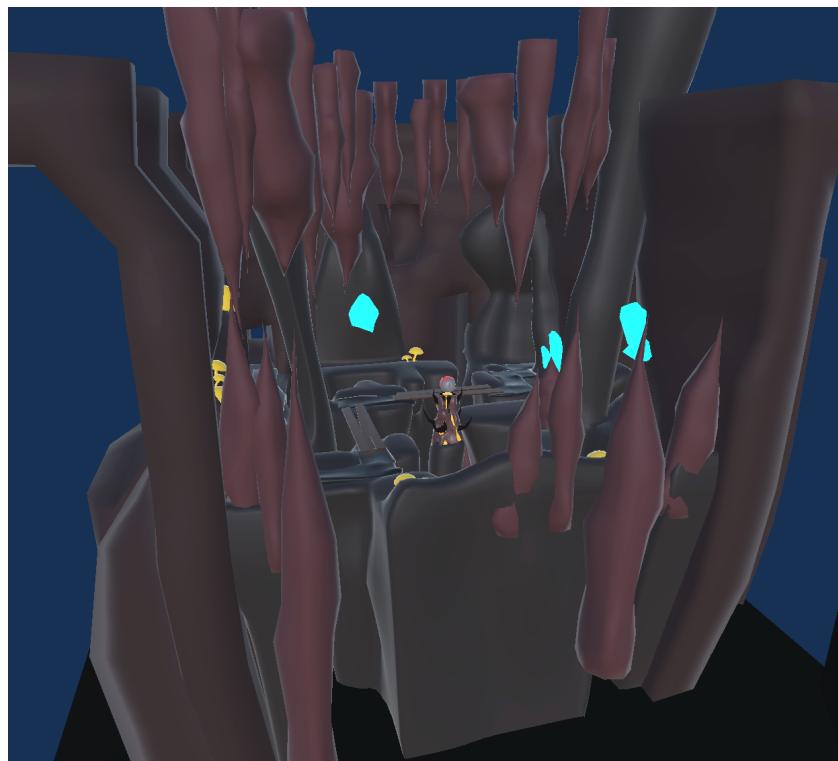
Farmer	Dorfschranzen
	
Tank	
	

Karten

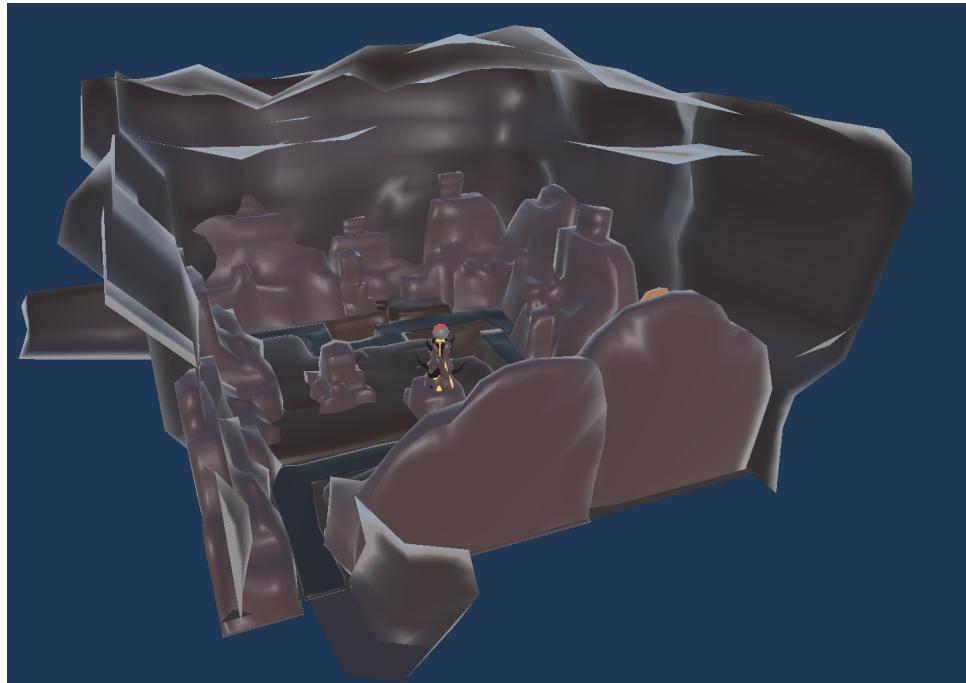
Map 1



Map 2

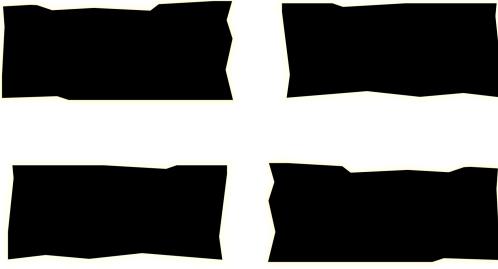
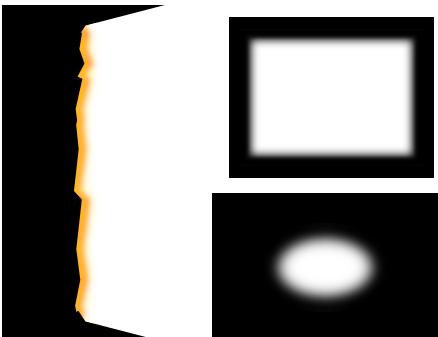


Map 3



4.1.2 Sprites

Die UI-Grafiken wurden alle mit Inkscape & Photoshop erstellt.

Logo / Überschrift	Buttons
 	
Hintergründe	Icon
	

4.1.3 Scripts

Einige Code-Ausschnitte wurden zur Übersichtlichkeit gekürzt, um das Hauptaugenmerk auf die wichtigen Funktionen zu lenken.

GameManager

Autor	Lucas Hengelhaupt, Henning Venhorst
Element	Assets/Scripts/GameManager.cs
Kurzfassung	verantwortlich für den GameLoop

Die Aufgabe des GameManagers ist es zu überprüfen, ob ein Spiel gewonnen oder verloren ist, sowie was danach passiert. Dabei arbeitet er mit den PlayerStats sowie dem WaveSpawner zusammen. Im GameManager ist die Win-Condition festgelegt, also die maximale Anzahl an Runden, die man überleben muss, um ein Level zu gewinnen.

```
public void WinLevel() {  
  
    gameWon = true;  
    levelWonUI.SetActive(true);  
  
    DisableEyeOfDoom();  
  
    Time.timeScale = 0f;  
  
    if (nextLevel == 0) {  
        // Disable the "Next Level"-Button if on last map  
        nextLevelButton.gameObject.SetActive(false);  
        return;  
    }  
  
    if (nextLevel > PlayerPrefs.GetInt("levelsUnlocked", 0)) {  
        PlayerPrefs.SetInt("levelsUnlocked", nextLevel);  
    }  
}
```

Erreicht der Spieler die Win-Condition, wird die Methode “WinLevel” aufgerufen. Hierbei wird dem Spieler das LevelWon-UI angezeigt, die Zeit angehalten und dabei das Wechseln in das “Eye of Doom” deaktiviert. Außerdem wird durch das Gewinnen das nächste Level freigeschaltet, was in den PlayerPrefs gespeichert wird.

Player / PlayerStats

Autor	Lucas Hengelhaupt, Henning Venhorst
Element	Assets/Scripts/Player/PlayerStats.cs
Kurzfassung	speichert das Geld und die Leben des Spielers

Grid / GridBuildingSystem

Autor	Lucas Hengelhaupt
Element	Assets/Scripts/Grid/GridBuildingSystem.cs
Kurzfassung	das Bauen von Türmen auf dem Grid

```
private void Update() {  
  
    if (switchGameMode.IsEyeOfDoomActive()) {  
        ...  
        return;  
    }  
  
    // build Tower on Left-Click  
    if (Input.GetMouseButton(0) && MouseIsNotOverUI()) {  
  
        // convert Mouse-Coordinates into Grid-Coordinates  
        var gridCoordinates = grid.GetXZ(GetClickedTile());  
  
        // get Object on clicked Tile  
        GridObject gridObject = grid.GetGridObject(gridCoordinates.x,  
                                                    gridCoordinates.z);  
  
        // check if clicked Tile is occupied  
        if (TileCanBeBuildOn(gridObject) && PlayerHasEnoughMoney()) {  
            // if tile is free, build on it  
            BuildTower(gridObject);  
  
        }  
        // if Tile already has a Tower -> show Upgrade/Sell + Range  
        else if (TileHasTower(gridObject)  
                 && IsPlaceable(gridCoordinates.x, gridCoordinates.z)) {  
            towerUI.SetTarget(gridObject);  
        }  
        ...  
    }  
}
```

Das GridBuildingSystem ist das “Herzstück” des Projektes. Es kümmert sich darum, dass der Spieler Türme auf dem Spielfeld platzieren und auswählen kann, sowie die ganze Logik, die damit zusammenhängt.

Im Update() wird zuallererst überprüft, ob der Spieler sich im “Eye of Doom” befindet, ist das der Fall, ist das Bauen und Anwählen von Türmen deaktiviert. Ist der Spieler aber in der Vogelperspektive, wird geschaut, wohin der Spieler mit der linken Maustaste auf das Spielfeld geklickt hat. Dabei werden die 3D-Welt-Koordinaten der Maus in Grid-Koordinaten umgewandelt. Das Grid besitzt für jede Zelle ganzzahlige Koordinaten von 0-16 in X- und Z-Richtung mit einer Zellengröße von je 10 Einheiten.

Danach wird geschaut, ob die angeklickte Zelle bereits einen Turm enthält (eine genauere Erläuterung zum “GridObject” erfolgt weiter unten). Je nachdem, ob sich dort schon ein Turm befindet oder nicht, wird entschieden, was als nächstes passieren soll:

TileCanBeBuildOn() überprüft, ob

- der Spieler überhaupt einen Turm ausgewählt hat
- der Spieler überhaupt auf eine Grid-Zelle geklickt hat
- die angeklickte Grid-Zelle nicht schon einen Turm auf sich hat
- der ausgewählte Turm an dieser Stelle gebaut werden darf (Gargoyle nur auf der Straße, Archer und Devil auf den anderen über das “TileLayout” initialisierten Feldern).

Wenn der Spieler nun noch genug Geld zum Kauf des Turms besitzt, wird der Turm an der ausgewählten Stelle gebaut:

```
private void BuildTower(GridObject gridObject) {  
  
    // get World-Coordinates to build on  
    Vector3 placedTowerWorldPosition =  
        grid.GetWorldPosition( gridObject.GetGridPosition().x,  
                               gridObject.GetGridPosition().z );  
    // create Tower in the World  
    PlacedTower placedTower =  
        PlacedTower.Create( placedTowerWorldPosition,  
                            currentlySelectedTowerTypeSO,  
                            pathTileYOffset );  
  
    // write created Tower in the Grid-Array  
    gridObject.SetPlacedTower(placedTower);  
  
    // subtract Tower-costs from Player-Money  
    PlayerStats.SubtractMoney(currentlySelectedTowerTypeSO.price);  
  
    // hide placement-tile on the Tower-position  
    DeactivateGridTile( gridObject.GetGridPosition().x,  
                        gridObject.GetGridPosition().z );  
}
```

Ist auf dem angeklickten Feld allerdings bereits ein Turm vorhanden, wird kein Turm gebaut, sondern stattdessen der Turm über die TowerUI Methode SetTarget() ausgewählt. Dadurch wird an der rechten Seite das Verbesserungs- und Verkaufen-Menü eingeblendet, sowie die Schuss-Reichweite des Turms angezeigt. Dabei kann der Gargoyle nicht ausgewählt werden, da dieser weder verbessert noch verkauft werden soll, und außerdem keine Schuss-Reichweite besitzt. Das wird über die Methode IsPlaceable() realisiert, welche überprüft, dass es sich bei dem ausgewählten Feld nicht um ein “Path”-Feld handelt.

```
private bool IsPlaceable(int x, int z) {  
    return gridTiles[x, z].GetTileType() == GridTile.TileType.Placeable;  
}
```

Grid / GridXZ

Autor	Lucas Hengelhaupt
Element	Assets/Scripts/Grid/GridXZ.cs
Kurzfassung	Generisches Grid auf der XZ Ebene

```
public GridXZ(int width, int height, float cellSize, Vector3 originPosition,
              Func<GridXZ<TGridObject>, int, int, TGridObject> createGridObject)
{
    ...

    // Array that stores the objects placed on the grid
    gridArray = new TGridObject[width, height];

    // initialize the grid-array to prevent null-error
    for (int x = 0; x < gridArray.GetLength(0); x++) {
        for (int z = 0; z < gridArray.GetLength(1); z++) {
            gridArray[x, z] = createGridObject(this, x, z);
        }
    }
}
```

Bei diesem Grid handelt es sich um ein generisches Grid, das auf der XZ-Ebene erstellt wird. Das heißt, es kann jegliche Art von Datentyp, egal ob ein Objekt, eine Klasse oder einfache Datentypen (int, bool...), entgegennehmen und diese auf ihm speichern und anzeigen lassen. Leider hat das vor allem am Anfang einige Probleme bereitet, und später hat sich herausgestellt, dass es für unsere Zwecke eigentlich viel zu komplex ist. Aber an diesem Zeitpunkt war es schon in zu viel Logik eingebaut, um es nochmal komplett neu zu schreiben. Letztendlich hat man sich jedoch damit zurechtgefunden.

Grid / GridObject

Autor	Lucas Hengelhaupt
Element	Assets/Scripts/Grid/GridObject.cs
Kurzfassung	“Container” für ein Objekt auf dem Grid

```
// Object-definition of the object that is placed on the Grid
public class GridObject {

    // Reference to the grid
    private GridXZ<GridObject> grid;
    private int x;
    private int z;
    private PlacedTower placedTower;

    // Constructor
    public GridObject(GridXZ<GridObject> grid, int x, int z) {
        this.grid = grid;
        this.x = x;
        this.z = z;
    }
}
```

Ein GridObject nimmt ein Objekt, das auf dem GridXZ platziert werden soll, entgegen und speichert sich dann selbst auf dem GridXZ. Der Sinn dahinter ist, dass die platzierten Objekte (hier die Türme) “sauber” von Grid-Funktionalitäten bleiben. Das heißt, dass in die Turm-Klasse z.B. keine Platzierungs-Koordinaten oder Get/Set-Funktionen dieser

aufgenommen werden müssen. Beim Start wird im GridXZ auf jeder Grid-Zelle ein solches GridObject erstellt. Um zu überprüfen, ob sich bereits ein Turm auf einer Grid-Zelle befindet, muss man sich das GridObject an dieser Stelle holen und schauen, ob der “placedTower” gleich “null” ist.

Grid / GridTile

Autor	Lucas Hengelhaupt
Element	Assets/Scripts/Grid/GridTile.cs
Kurzfassung	Felder, auf dem ein Turm platziert werden kann

```
private void OnMouseEnter() {
    highlight.SetActive(true);
}

private void OnMouseExit() {
    highlight.SetActive(false);
}

private void CheckIfPlayerHasEnoughMoney() {
    if (gridBuildingSystem.PlayerHasEnoughMoney()) {
        tileRenderer.materialSetColor( "_BaseColor",
                                       gridTileSO.defaultColor );
    }
    else {
        tileRenderer.materialSetColor( "_BaseColor",
                                       gridTileSO.insufficientMoneyColor );
    }
}
```

Wie in “3.4.3 Erstellung des Turm-Platzierungs-Layouts” beschrieben ist, wird über das “TileLayout” auf jeder Grid-Zelle ein GridTile erstellt, auf dem das Platzieren eines Turms möglich sein soll. Wenn der Spieler genug Geld hat, einen Turm zu platzieren, sind die GridTiles grün, wenn er zu wenig Geld haben sollte, werden sie rot. Außerdem ermöglichen sie einen “Hover”-Effekt, wenn man sich mit der Maus über ein GridTile bewegt, um zu visualisieren, wo man den Turm platziert.

Tower / PlacedTower

Autor	Henning Venhorst, Lucas Hengelhaupt
Element	Assets/Scripts/Tower/PlacedTower.cs
Kurzfassung	auf dem Spielfeld platziert Turm

```
private void Update() {  
  
    if (towerName == "Gargoyle") {  
        TowerShop.LockGargoyle();  
        BlockEnemies();  
        DestroySelf();  
        return;  
    }  
  
    if (target == null) {  
        return;  
    }  
  
    LockOnTarget();  
  
    if (fireCountdown <= 0f) {  
        Shoot();  
        fireCountdown = 1f / fireRate;  
    }  
  
    fireCountdown -= Time.deltaTime;  
}
```

Die Klasse PlacedTower hat zwei verschiedene Verhaltensweisen, je nachdem, ob es sich um einen schießenden Turm (“Archer”, “Devil”) oder den “Gargoyle” handelt.

Hat ein schießender Turm einen Gegner in seinem Schuss-Radius entdeckt, schaut der Turm mit LockOnTarget() in die Richtung des gefundenen Gegners. Ist die Abklingzeit seines Projektils abgelaufen, schießt der Turm ein Projektil auf den anvisierten Gegner. Außerdem kann so ein Turm im Zusammenspiel mit dem “TowerUI” verbessert und verkauft werden.

Handelt es sich bei dem platzierten Turm jedoch um einen “Gargoyle”, spannt dieser eine OverlapSphere() an seinem Standpunkt auf, um alle Gegner zu erkennen, die sich auf seinem Feld befinden oder dieses betreten wollen. Die erkannten Gegner werden dann blockiert und in einem Array zwischengespeichert, damit diese wieder befreit werden können, wenn der “Gargoyle” verschwunden ist.

```
private void BlockEnemies() {  
    blockedEnemies =  
        Physics.OverlapSphere( transform.position,  
            gridBuildingSystem.GetCellSize() / 2  
    );  
  
    foreach (Collider enemy in blockedEnemies) {  
        if (enemy.tag == enemyTag) {  
            enemy.transform.GetComponent<Enemy>().BlockEnemy();  
        }  
    }  
}
```

Tower / Projectile

Autor	Henning Venhorst
Element	Assets/Scripts/Tower/Projectile.cs
Kurzfassung	Projektil eines platzierten Turms, das auf den Gegner geschossen wird

```
private void HitTarget() {  
  
    if (damageRadius > 0f) {  
        AoEDamage();  
    }  
    else {  
        Damage(target);  
    }  
  
    // Hit-Particles  
    GameObject particleInstance = Instantiate( impactParticles,  
                                                transform.position,  
                                                Quaternion.identity );  
  
    Destroy(particleInstance, 2f);  
    Destroy(gameObject);  
}
```

Nachdem das Projektil die Position des Gegners erreicht hat, wird entschieden, ob es nur Schaden an dem getroffenen Gegner macht, oder ob es sich um ein Flächenschaden-Projektil handelt. Ist letzteres der Fall, werden mithilfe einer Physics.OverlapSphere() alle Gegner im Schadens-Radius des Projektils gefunden und jedem Schaden zugefügt. Außerdem wird bei einem Treffer ein Partikeleffekt abgespielt, um den Treffer visuell besser zu signalisieren. Danach werden der Partikeleffekt sowie das Projektil gelöscht.

Shop / TowerShop

Autor	Lucas Hengelhaupt
Element	Assets/Scripts/Shop/TowerShop.cs
Kurzfassung	sendet den ausgewählten Turm an das GridBuildingSystem

```
public void SelectArcher() {  
    gridBuildingSystem.SetSelectedTower(archerTowerSO);  
}  
...  
  
private void Update() {  
  
    if (!gargoyleIsPlaced) {  
        return;  
    }  
  
    if (timeLeftUntilGargoyleUnlocked <= 0f) {  
        UnlockGargoyle();  
        return;  
    }  
  
    timeLeftUntilGargoyleUnlocked -= Time.deltaTime;  
    UpdateGargoyleCooldownText();  
}
```

Der TowerShop ist mit den UI-Buttons der Turm-Auswahl an der linken Seite verbunden, damit der im UI ausgewählte Turm an das GridBuildingSystem weitergegeben werden kann, um die richtigen Platzierungs-Felder anzuzeigen und um zu wissen, welcher Turm gebaut werden soll. Des Weiteren wird beim Platzieren eines Gargoyles sein Auswahlknopf für 30s deaktiviert, damit dieser nicht zu häufig verwendet werden kann.

Enemy / Enemy

Autor	Henning Venhorst, Henrik Timpel
Element	Assets/Scripts/Enemy/Enemy.cs
Kurzfassung	Gegner Objekt

```
void Update () {
    if (isDead || isBlocked) {
        return;
    }

    MoveToNextWaypoint();
    RotateToNextWaypoint();

    if (Vector3.Distance(transform.position, nextWaypoint.position)
        <= 1f) {
        GetNextWaypoint();
    }
}
```

Solange der Gegner nicht durch einen “Gargoyle” blockiert oder tot ist, bewegt er sich auf dem durch die Wegpunkte festgelegten Weg vom Anfang des Levels bis zum Ende.

Sollte ein Projektil auf den Gegner treffen, wird diesem Schaden über die Methode TakeDamage() zugefügt. Hat der Gegner genug Schaden erlitten und die Lebenspunkte sind unter 0 gefallen, stirbt er. Dabei wird dem Spieler der Geldwert des getöteten Gegners gutgeschrieben, die Anzahl an noch verbleibenden Gegnern um 1 reduziert und eine Todesanimation abgespielt.

```
public void TakeDamage(float amount) {
    currentHP -= amount;

    if (isDead) {
        Die();
    }
}

private void Die() {

    PlayerStats.AddMoney(killValue);
    EnemySpawner.enemiesAlive--;

    gameObject.tag = "Dead";

    // play Death-Animation
    animator.SetTrigger("death");

    Destroy(gameObject, 0.2f);
}
```

Enemy / Waypoints

Autor	Henrik Timpel
Element	Assets/Scripts/Enemy/Waypoints.cs
Kurzfassung	definieren den Weg der Gegner

```
public static Transform[] waypoints;

void Awake () {
    waypoints = new Transform[transform.childCount];

    for (int i = 0; i < waypoints.Length; i++) {
        waypoints[i] = transform.GetChild(i);
    }
}
```

Die im Level gesetzten Wegpunkte werden der Reihe nach in einem Array gespeichert, worauf jeder Gegner zugreifen kann, um den festgelegten Weg ablaufen zu können.

Enemy / EnemySpawner

Autor	Henning Venhorst
Element	Assets/Scripts/Enemy/Enemy.cs
Kurzfassung	spawnt Gegnerwellen

```
if (currentWaveNumber >= minWaveNumberForDorfsschranze) {
    numberOfEnemiesUnlocked = 2;
}

if (currentWaveNumber >= minWaveNumberForTank) {
    numberOfEnemiesUnlocked = 3;
}

// determine a random next enemy that is spawned, depending on how
// many enemies are "unlocked"
int enemyType = Random.Range(1, (numberOfEnemiesUnlocked + 1)*2 - 1);

switch (enemyType) {
    case 1: case 2: case 6:
        SpawnEnemy(farmerSO);

    case 3: case 4:
        SpawnEnemy(dorfsschranzeSO);

    case 5:
        SpawnEnemy(tankSO);

    default:
        SpawnEnemy(farmerSO);
}
```

Der EnemySpawner instanziert die Gegner, die der Spieler töten muss, bevor sie das Ende des Levels erreichen. Dabei werden mit steigender Rundenzahl weitere Gegnertypen freigeschaltet. Der nächste erschaffene Gegner wird dabei per Zufall ausgewählt, es gibt also kein festes Muster für die Wellen, abgesehen von der Anzahl der Gegner pro Welle. Dabei hat der “Farmer” die höchste Wahrscheinlichkeit zu erscheinen, und der “Tank” die niedrigste.

Eye of Doom / DoomRay

Autor	Henrik Timpel
Element	Assets/Scripts/EyeOfDoom/DoomRay.cs
Kurzfassung	“Projektil” vom “Eye of Doom”

```
void Update() {
    fireTimer += Time.deltaTime;

    if (Input.GetMouseButtonDown(0) && fireTimer > fireRate) {
        fireTimer = 0;

        rayLine.SetPosition(0, doomRayOrigin.position);

        Vector3 rayOrigin = povCam.ViewportToWorldPoint(
            new Vector3(0.5f, 0.5f, 0));

        if (Physics.Raycast(rayOrigin, povCam.transform.forward,
            out RaycastHit hit, rayRange)) {
            rayLine.SetPosition(1, hit.point);

            Enemy enemy = hit.transform.GetComponent<Enemy>();

            if (enemy != null) {
                enemy.TakeDamage(damage);
            }
        }

        StartCoroutine(ShootDoomray());
    }
}
```

Befindet man sich im “Eye of Doom” und betätigt die linke Maustaste, wird zwischen dem Augen-Modell und der anvisierten Stelle mit Hilfe des LineRenderers ein “Feuerstrahl” gezeichnet. Wird dabei ein Gegner getroffen, wird diesem Schaden zugefügt.

Eye of Doom / EyeMovement

Autor	Henrik Timpel, Lucas Hengelhaupt
Element	Assets/Scripts/EyeOfDoom/EyeMovement.cs
Kurzfassung	Übertragung der Mausbewegung auf die “Eye of Doom”-Kamera

```
private void Update() {
    // read Mouse input
    float mouseX = Input.GetAxisRaw("Mouse X") *
                    (Time.deltaTime/Time.timeScale) * sensitivity*10;
    float mouseY = Input.GetAxisRaw("Mouse Y") *
                    (Time.deltaTime/Time.timeScale) * sensitivity*10;

    // Mouse smoothing
    xAccumulator = Mathf.Lerp(xAccumulator, mouseX, smoothing *
                                (Time.deltaTime/Time.timeScale));
    yAccumulator = Mathf.Lerp(yAccumulator, mouseY, smoothing *
                                (Time.deltaTime/Time.timeScale));
    rotationY += xAccumulator;
    rotationX -= yAccumulator;

    // prevent Player to look more than 90° up or down
    rotationX = Mathf.Clamp(rotationX, -90f, 90f);

    // rotates Camera around both axis
    eyeCam.rotation = Quaternion.Euler(rotationX, rotationY, 0);
    // rotates eye to follow Camera
    eyeBall.localRotation = Quaternion.Euler(90, rotationY, 0);
}
```

Befindet man sich im “Eye of Doom”, werden im EyeMovement die Maus-Achsen ausgelesen und als Rotation auf die Kamera sowie das Augen-Modell übertragen. Dabei wird die in den Einstellungen eingestellte Mausempfindlichkeit sowie Smoothing auf die Bewegung angewendet.

Eye of Doom / SwitchCam

Autor	Henrik Timpel
Element	Assets/Scripts/EyeOfDoom/SwitchCam.cs
Kurzfassung	Wechsel zwischen der Vogelperspektive und dem “Eye of Doom”

```
private void SwitchState() {

    if (mainCameraIsActive) {
        animator.Play("EyeCam");
        mainCameraIsActive = false;
    }
    else {
        animator.Play("MainCam");
        mainCameraIsActive = true;
    }
}
```

In jedem Level existieren zwei Kameras: Die “Main”-Kamera, die aus der Vogelperspektive auf das Spielfeld schaut, und die “Eye”-Kamera, die auf dem “Eye of Doom” sitzt. Durch Drücken der Leertaste wird zwischen den beiden Kameras hin-und her gewechselt.

Eye of Doom / SwitchGameMode

Autor	Henrik Timpel
Element	Assets/Scripts/EyeOfDoom/SwitchGameMode.cs
Kurzfassung	de- und aktiviert das “Eye of Doom”

```
private void Update() {  
  
    if (!switchCam.enabled) {  
        return;  
    }  
  
    if (Input.GetKeyDown("space")) {  
        ray.enabled = !ray.enabled;  
        eyeMovement.enabled = !eyeMovement.enabled;  
    }  
}
```

SwitchGameMode hängt sehr mit SwitchCam zusammen. SwitchCam kümmert sich dabei nur um den Wechsel zwischen den Kameras. SwitchGameMode de- und aktiviert das Bewegen der Kamera sowie das Schießen des “Feuerstrahls”. Das Aktivieren dieser Fähigkeiten wird nicht erlaubt, wenn SwitchCam deaktiviert ist, da man sonst im Pausenmenü oder nach Abschluss/Verlieren eines Levels weiterhin wechseln könnte.

UI / PlayerUIButtonHandler

Autor	Lucas Hengelhaupt
Element	Assets/Scripts/UI/PlayerUIButtonHandler.cs
Kurzfassung	beinhaltet die Funktionen aller UI-Buttons in einem Level, abgesehen vom “Start and Speedup”-Button

```
public void TogglePauseMenuVisibility() {  
  
    pauseUI.SetActive(!pauseUI.activeSelf);  
  
    if (pauseUI.activeSelf) {  
        previousGameSpeed = Time.timeScale;  
        Time.timeScale = 0f;  
        gameManager.DisableEyeOfDoom();  
    }  
    else {  
        Time.timeScale = previousGameSpeed;  
        gameManager.EnableEyeOfDoom();  
    }  
}
```

Wir haben uns dafür entschieden, den Großteil der UI-Button-Funktionen in einem Handler zusammenzuführen, statt auf jedem UI-Element ein Skript zu haben.

Dabei muss beim Aufruf des Pausenmenü darauf geachtet werden, das die aktuelle Spielgeschwindigkeit gespeichert (falls während einer Runde die Spielgeschwindigkeit erhöht wurde), die TimeScale auf 0 gesetzt (damit das Spiel nicht im Hintergrund weiterläuft), sowie die Steuerung des “Eye of Doom” deaktiviert wird. Bei der Rückkehr in das Spielgeschehen wird die gespeicherte Spielgeschwindigkeit wiederhergestellt, und der GameManager schaut, ob das Auge aktiv war, bevor das Spiel pausiert wurde.

UI / PlayerUITextHandler

Autor	Lucas Hengelhaupt
Element	Assets/Scripts/UI/PlayerUITextHandler.cs
Kurzfassung	Aktualisierung sich ändernder UI-Textfelder

```
void Update() {
    livesText.text = "Lives: " + PlayerStats.lives.ToString();

    moneyText.text = "$" + PlayerStats.GetMoney().ToString();

    if (enemySpawner.IsEndless()) {
        waveNumberText.text = string.Format("Wave {0}",
                                             gameManager.GetCurrentWaveNumber());
    }
    else {
        waveNumberText.text = string.Format("Wave {0}/{1}",
                                             gameManager.GetCurrentWaveNumber(),
                                             gameManager.GetMaxWaveNumber());
    }

    if (gameOverWaveReachedText.IsActive()) {
        gameOverWaveReachedText.text = "Wave Reached: " +
            gameManager.GetCurrentWaveNumber();
    }
}
```

Auch hier haben wir uns dafür entschieden, sich ständig ändernde UI-Text-Elemente in einem Handler zusammenzuführen, statt auf jedem UI-Element ein Skript zu haben.

UI / LevelSelectUIButtonHandler

Autor	Lucas Hengelhaupt
Element	Assets/Scripts/UI/LevelSelectUIButtonHandler.cs
Kurzfassung	schaltet entsprechend der Anzahl an freigeschalteten Leveln die Level-Auswahl-Buttons frei

```
int levelsUnlocked = PlayerPrefs.GetInt("levelsUnlocked", 1);

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

    if (i < levelsUnlocked) {
        levelButtons[i].interactable = true;
        ...
    }
    else {
        levelButtons[i].interactable = false;
        ...
    }
}
```

UI / MainMenuUIButtonHandler

Autor	Lucas Hengelhaupt
Element	Assets/Scripts/UI/MainMenuUIButtonHandler.cs
Kurzfassung	beinhaltet die Funktionen aller UI-Buttons im Main Menu

UI / SettingsUI

Autor	Lucas Hengelhaupt
Element	Assets/Scripts/UI/SettingsUI.cs
Kurzfassung	Einstellungen für Video, Sound und Maus

In den Einstellungen kann der Spieler sein Spielerlebnis individuell anpassen. Dort kann er die Bildschirmauflösung ändern, zwischen Fenster-und Vollbildmodus wechseln, die Helligkeit des Spiels anpassen sowie die Lautstärke der Hintergrundmusik und der Soundeffekte. Außerdem kann die Mausempfindlichkeit und die Bewegungsglättung der “Eye of Doom”-Kamera eingestellt werden.

Wir haben uns dafür entschieden, bei den Bildschirmauflösungen nur die Auflösungen anzuzeigen, die auch zu der Refresh-Rate des vom Spieler genutzten Monitors passen:

```
private void InitializeScreenResolutionDropdown() {
    screenResolutions = Screen.resolutions;
    currentRefreshRate = Screen.currentResolution.refreshRate;
    filteredResolutions = new List<Resolution>();
    List<string> options = new List<string>();

    // get Screen-Resolutions that fit current Monitors refresh-rate
    for (int i = 0; i < screenResolutions.Length; i++) {
        if (screenResolutions[i].refreshRate == currentRefreshRate) {
            filteredResolutions.Add(screenResolutions[i]);
        }
    }

    // format all Screen-Resolutions and put them into a list
    for (int i = 0; i < filteredResolutions.Count; i++) {
        string resolutionOption =
            filteredResolutions[i].width + "x" +
            filteredResolutions[i].height + " " +
            filteredResolutions[i].refreshRate + "Hz";

        options.Add(resolutionOption);

        // get current Screen-Resolution for setting up the
        // pre-selection of the Dropdown
        if (filteredResolutions[i].width == Screen.width
            && filteredResolutions[i].height == Screen.height) {
            currentResolutionIndex = i;
        }
    }
    resolutionDropdown.value = currentResolutionIndex;
}
```

Das Einstellen der Spiel-Helligkeit wurde über das Post Processing realisiert. Dabei hat jedes Level ein “Volume” mit dem Override “Color Adjustments”, unter dem sich die Eigenschaft “Post Exposure” befindet, dessen Wert durch den Helligkeits-Slider in den Einstellungen verändert wird.

```
public void AdjustBrightness(float brightnessValue) {

    // snap Slider to 0 when inside the threshold
    if (brightnessValue >= -brightnessSliderThreshold
        && brightnessValue <= brightnessSliderThreshold) {
        brightnessValue = 0f;
        brightnessSlider.value = brightnessValue;
    }

    // get PostExposure from PostProcessing-Volume
    postProcessingVolume.profile.TryGet(out ColorAdjustments
                                         colorAdjustments);
    // set PostExposure to Slider-Value
    colorAdjustments.postExposure.value = brightnessValue;
    // save Brightness
    PlayerPrefs.SetFloat("Brightness", brightnessValue);
}
```

UI / StartAndSpeedupButton

Autor	Lucas Hengelhaupt
Element	Assets/Scripts/UI/StartAndSpeedupButton.cs
Kurzfassung	startet eine neue Runde, kann Spielgeschwindigkeit erhöhen

```
public void ChangeGameState() {

    switch (state) {
        case (GameManager.GameState.beforeNewRound):
            Time.timeScale = 1f;
            state = GameManager.GameState.play;
            StartCoroutine(enemySpawner.SpawnWave());
            ...
        case (GameManager.GameState.play):
            Time.timeScale = 2f;
            state = GameManager.GameState.speed2;
            ...
        case (GameManager.GameState.speed2):
            Time.timeScale = 3f;
            state = GameManager.GameState.speed3;
            ...
        case (GameManager.GameState.speed3):
            Time.timeScale = 1f;
            state = GameManager.GameState.play;
            ...
    }
}
```

Beim Betätigen des “Start Round”-Buttons wird die Methode ChangeGameState() aufgerufen, wobei in der “Case”-Anweisung der momentane GameState abgefragt wird. Ist der momentane GameState also auf “play” gesetzt, und der Button wird gedrückt, dann soll die Spielgeschwindigkeit auf x2 und der neue GameState auf “speed2” gesetzt werden. Nach erfolgreichem Abschluss einer Runde wird der GameState immer auf “beforeNewRound” gesetzt.

UI / TowerUI

Autor	Lucas Hengelhaupt
Element	Assets/Scripts/UI/TowerUI.cs
Kurzfassung	Schnittstelle für den Spieler, um auf Turm-Funktionalitäten zuzugreifen

Das TowerUI wird durch das GridBuildingSystem über die Methode SetTarget() aktiviert, wenn ein Feld angeklickt wurde, auf dem sich schon ein Turm befindet. Dabei wird dem TowerUI das ausgewählte Feld mit dem Turm übergeben.

```
public void SetTarget(GridObject gridObject) {

    // if the same tile is clicked again, hide the Upgrade/Sell-Menu
    if (targetedGridObject == gridObject && worldSpaceUI.activeSelf) {

        HideUI();
    }
    // show Upgrade/Sell-Menu, move Range & Arrow over selected Tower
    else {
        targetedGridObject = gridObject;
        worldSpaceUI.transform.position = gridObject.GetWorldPosition();
        UpdateUI();
        worldSpaceUI.SetActive(true);
        screenSpaceUI.SetActive(true);
    }
}
```

Im Update() wird regelmäßig geschaut, ob der Spieler genug Geld hat, um ein Turm-Upgrade zu machen, und de- oder aktiviert den Upgrade-Button dementsprechend.

```
private void Update() {

    if (targetedGridObject != null) {
        ...

        if (PlayerHasEnoughMoney()) {
            upgradeButton.interactable = true;
        }
        else {
            upgradeButton.interactable = false;
        }
    }
}
```

Die Upgrade() und Sell() Methoden greifen über das übergebene GridObject auf die Upgrade- und Sell-Methoden des ausgewählten Turms zu.

UI / TutorialUI

Autor	Lucas Hengelhaupt
Element	Assets/Scripts/UI/TutorialUI.cs
Kurzfassung	Anzeigen des Tutorials

```
private void Start() {  
  
    // show Tutorial if Player hasn't completed it yet  
    if (PlayerPrefs.GetInt("TutorialCompleted", 0) == 0) {  
        tutorialUI.SetActive(true);  
        ShowTutorial();  
    }  
    else {  
        tutorialUI.SetActive(false);  
    }  
}
```

Zeigt das Tutorial bei der Auswahl des ersten Levels, wenn das Spiel zum ersten Mal gestartet wurde. Das Tutorial kann bei Bedarf immer wieder über das Pausenmenü angeschaut werden, wobei direkt die Methode ShowTutorial() aufgerufen wird.

4.2 Fremde Assets

4.2.1 Unity-Packages

- Text-Mesh-Pro
- Cinemachine
- Universal Render Pipeline

4.2.2 Musik

- Main Menu
<https://pixabay.com/music/build-up-scenes-the-time-is-upon-us-127798/>
- Map 1
<https://pixabay.com/music/main-title-grecia-120640/>
- Map 2
<https://pixabay.com/music/upbeat-dreamer-131011/>
- Map 3
<https://pixabay.com/music/pulses-donx27t-leave-me-alone-137106/>

4.2.3 Soundeffekte

- Bogen:
<https://pixabay.com/sound-effects/bow-release-bow-and-arrow-4-101936/>
- Feuerball:
<https://freesound.org/people/SilverIllusionist/sounds/472688/>
- Map 1 “Lava Bubbling”
<https://pixabay.com/sound-effects/gunk-bubbling-in-cave-18880/>

4.2.4 Schriftart

Um das höllische Dungeon-Theme weiter zu unterstützen, haben wir uns für die Schriftart “Devil Helloween” entschieden.

DEVIL HELLOWEEN

<https://www.1001freefonts.com/devil-helloween.font>

5. Weitere Inhalte

5.1 Team & Arbeitsaufteilung

Die Aufgaben wurden über die Gitlab-Issues verteilt und abgearbeitet.

Dabei gab es die Priorisierung in:

- “Functionality”: Funktionalitäten, die unbedingt notwendig sind
- “Enhancement”: Funktionalitäten, die “nice-to-have” wären
- “Bugs”

More UI Functionality	CLOSED	closed 1 month ago
UI #12 · created 2 months ago by Lucas / Diafreak · functionality	closed 1 month ago	closed 1 month ago
UI Tower Selection	CLOSED	closed 2 months ago
UI #11 · created 2 months ago by Lucas / Diafreak · functionality	closed 2 months ago	closed 2 months ago
gargoyle blocking	CLOSED	closed 4 weeks ago
UI #10 · created 3 months ago by Henning Venhorst · functionality	closed 4 weeks ago	closed 4 weeks ago
tower upgrades	CLOSED	closed 1 month ago
UI #9 · created 3 months ago by Lucas / Diafreak · functionality	closed 1 month ago	closed 1 month ago
animations for towers & townfolk	CLOSED	closed 5 days ago
UI #8 · created 3 months ago by Lucas / Diafreak · enhancement · modeling	closed 5 days ago	closed 5 days ago
enemy creation	CLOSED	closed 5 days ago
UI #7 · created 3 months ago by Henning Venhorst · functionality · modeling	closed 5 days ago	closed 5 days ago
tower creation	CLOSED	closed 1 week ago
UI #6 · created 3 months ago by Henning Venhorst · functionality · modeling	closed 1 week ago	closed 1 week ago
map creation	CLOSED	closed 1 week ago
UI #5 · created 3 months ago by Henning Venhorst · functionality · modeling	closed 1 week ago	closed 1 week ago
townfolk targeting system	CLOSED	closed 2 months ago
UI #4 · created 3 months ago by Henning Venhorst · functionality	closed 2 months ago	closed 2 months ago
townfolk pathfinding	CLOSED	closed 1 month ago
UI #3 · created 3 months ago by Lucas / Diafreak · functionality	closed 1 month ago	closed 1 month ago
create townfolk spawner	CLOSED	closed 2 months ago
UI #2 · created 3 months ago by Henning Venhorst · functionality	closed 2 months ago	closed 2 months ago
creating Placementgrid	CLOSED	closed 2 months ago
UI #1 · created 3 months ago by Henning Venhorst · functionality	closed 2 months ago	closed 2 months ago

Mitglied	Aufgaben
Lucas Hengelhaupt	Grid & Turmplatzierung, UI-Funktionalitäten, Gameloop, Zusammenführen der Funktionalitäten
Henning Venhorst	Wavespawner, Turm-Zielerfassung & Schießen
Eric Kruse	Modellierung + Scene-Design, UI-Design & -Funktionalitäten
Henrik Timpel	Gargoyle-Blocking, “Eye of Doom”

5.2 Vorgehensweise

Zu Beginn war es uns wichtig, eine Unterteilung in stellbare Plätze für Türme und den Laufweg der Gegner zu definieren. Dafür haben wir ein Grid erstellt. Als nächstes war dann die Platzierung der Türme auf diesem Grid nötig. Hierfür wurden rudimentäre UI-Funktionalitäten geschaffen, um Türme zum Platzieren auswählen zu können sowie diese wieder vom Spielfeld zu entfernen. Das Anvisieren der Gegner durch die Türme wurde als nächstes umgesetzt, sowie zu Testzwecken ein einfacher Gegner Spawner, was mit der Implementierung der Gegner-Wegfindung über das Spielfeld einherging. Da nun alle Grundfunktionalitäten für ein Tower-Defense standen, haben wir uns um das Hinzufügen von Menüs und die Levelauswahl gekümmert. Deshalb haben wir uns ab diesem Zeitpunkt um unsere “Unique Selling Points” gekümmert: Das “Gargoyle”-Blocking zusammen mit dem “Eye of Doom”. In der letzten Phase sollten keine neuen Features mehr dazukommen, sondern es wurden nur noch “Quality of Life” Veränderungen implementiert. So haben die Türme besser unterscheidbare Modelle, Animationen und Soundeffekte erhalten; das UI wurde noch einmal aufgehübscht und Funktionalitäten wie das Beschleunigen einer Runde wurden hinzugefügt, um die Spielerfahrung angenehmer zu machen.

5.3 Dev-Shortcuts

Um den Funktionsumfang des Spiels voll testen zu können, ohne es unendlich lang spielen zu müssen, kann man mit der Taste “M” immer wieder \$100 zu seinem aktuellen Geld hinzufügen.

5.4 Bugs & Besonderheiten

Im Build werden die “mixed” Point-Lights nicht in Echtzeit auf die Umgebung gerendert, sondern nur auf die Türme und Gegner, wofür wir auch nach viel Herumprobieren leider keine Lösung gefunden haben. Das hat vor allem auf der 2. Karte die Folge, dass die Umgebung sehr schlecht erkennbar ist und ästhetisch auch nicht sehr schön aussieht. Da die Türme und Gegner sowie die Platzierungs-Felder aber Emission-Materialien haben, ist diese trotzdem spielbar.

Leider hat sich zu spät herausgestellt, dass es immer noch zu mehrfacher Ausführung der GegnerMethode “Die()” kommen kann, wenn zwei oder mehr Türme gleichzeitig auf einen Gegner schießen und er in diesem Moment stirbt. Das hat zur Folge, dass es bei der Anzahl der “Enemies Alive” zu Fehlberechnungen kommt und Runden frühzeitig als “Fertig” erkannt werden, was unter anderem auch dazu führt, dass ein Level gewonnen werden kann, obwohl sich noch Gegner auf dem Feld befinden. Wir waren der Meinung das Problem behoben zu haben, indem tote Gegner den Tag “Dead” bekommen und somit die “Die()” Methode nicht mehr ausgeführt wird, aber leider mussten wir feststellen, dass das Problem immer noch besteht.

Außerdem besteht das Problem, dass angewählte Buttons weiterhin unscheinbar angewählt bleiben. Das hat zur Folge, dass z.B. das Betätigen der Leertaste (Wechsel in das “Eye of Doom”) diesen Button erneut ungewollt aktiviert.

5.5 Blick in die Zukunft

Wenn das Projekt noch weitergehen würde, hätten wir gerne noch mehr und bessere Animationen, vor allem für die Türme, eingefügt.

Außerdem muss das Spiel noch besser gebalanced werden, was mit drei Türmen, von denen nur zwei direkt Schaden an den Gegnern machen, sehr schwer ist. Deshalb sollte das Arsenal an platzierbaren Türmen noch mit weiteren verschiedenen Typen erweitert werden.

6. Anhang

6.1 Glossar

Balancing	Werte der Türme und Gegner (Schaden, Reichweite, Lebenspunkte, Geld...) so definieren, dass eine positive Erfahrung bei den Spielern entsteht
Eye of Doom	durch "Leertaste" übernehmbarer Turm auf jeder Karte, womit der Spieler mit der Maus selber auf die Gegner schießen kann

6.2 Links

- GitLab-Repository
<https://git.ai.fh-erfurt.de/gebeetsbrueder/annoyed-fiends>
- Build ZIP-Archiv (Google Drive)
https://drive.google.com/file/d/1_xN7DzOnZ-RPpJQqUIB7tgae1ONEKLA/view?usp=sharing
- Projekt (clean - nur Assets, Packages & ProjectSettings) (Google Drive)
<https://drive.google.com/file/d/13kzvA8K-2mwaxp6F1SNM5Eg13nNyr1di/view?usp=sharing>
- Project Design Document (Google Docs)
<https://docs.google.com/document/d/1HqLwWqXEn6w5v6cUR7kvVh0FtM4FhmCq-et8AbcK0uo/edit?usp=sharing>
- Screencast (YouTube)
<https://youtu.be/4S5I54fL7tI>
- Screencast (Google Drive)
<https://drive.google.com/file/d/1XVgHQIZF0d-1ou0QgDUQgQ5v3NWNojl0/view?usp=sharing>