**Dokumentation Can You Escape**

Von: Nils Kunkel

Matrikelnummer: 25437

Inhalt

[1.Grundidee 2](#__RefHeading__877_159990601)

[2.Endprodukt 2](#__RefHeading__879_159990601)

[3.Entwicklungsprozess 2](#__RefHeading__881_159990601)

[4.Codestruktur 3](#__RefHeading__883_159990601)

[3.1. Gegner 4](#__RefHeading__885_159990601)

[3.2 Waffen 6](#__RefHeading__887_159990601)

[3.3 Nachrichtensystem 10](#__RefHeading__889_159990601)

[3.4 Buttons 11](#__RefHeading__891_159990601)

[5.Ausblick/ Erweiterungsmöglichkeiten 12](#__RefHeading__893_159990601)

[6.Bildverzeichnis 13](#__RefHeading__895_159990601)

[7.Anlagen 13](#__RefHeading__897_159990601)

[8. Quellen 13](#__RefHeading__899_159990601)

1. Grundidee

Die Grundidee war es ein Spiel zu entwickeln bei dem der Spieler aus einem Haus entkommen muss und dabei einem nicht zu besiegendem Gegner ausweichen muss. Ähnlich wie beim Spiel Alien Isolation, bei dem sich der Spieler von einem unbesiegbaren Alien verstecken muss.

Um flüchten zu können sollen Räume durchsucht werden um Schlüssel und andere hilfreiche Gegenstände wie z.B. eine Taschenlampe zu finden. Das durchsuchen war ursprünglich als „Point and Click“ Minispiel gedacht, was allerdings schnell verworfen wurde da es nicht zum restlichen Schleichspiel gepasst hätte und ein eigenes gewesen Spiel wäre.

Damit der Spieler sich auch gegen den Angreifer verteidigen kann wurde die Spielidee noch um Waffen erweitert und geht damit mehr in die Richtung eines Metal Gear Solids als zu Alien Isolation.



Abbildung 1 Paper Prototype

Grundentwurf des Spiels bei dem der Spieler dem Gegner den Schlüssel abnehmen musste. Um dies zu schaffen muss er sich die Waffe aus dem zweiten Raum beschaffen. Die farbigen Türen können nur mithilfe des zugehörigen Schlüssels geöffnet werden.

1. Endprodukt

Das Endprodukt stellt ein Schleichspiel bestehend aus 4 unterschiedlichen Level dar. Bei dem es das Ziel ist möglichst unauffällig vorzugehen um aus dem Haus, welches von Gegner belagert wird, zu entkommen. Um weiterzukommen müssen Schlüssel gefunden werden. Die Schlüssel und Türen sind über Farben zugeordnet.

Dabei gibt es zwei unterschiedliche Gegnertypen. Auf der einen Seite fest stationäre Türme bei welchem der Spieler den richtigen Zeitpunkt finden muss um nicht entdeckt und beschossen zu werden. Die Türme können mithilfe eines Schalters dauerhaft deaktiviert werden.

Auf der anderen Seite patrouillierende Soldaten die den Spieler angreifen und verfolgen sobald dieser bemerkt wurde. Im Verlauf des Spiels kann ein Messer gefunden werden mit welchem der Soldat angegriffen werden kann. Je nachdem ob der Soldat den Angreifer bemerkt hat erleidet er mehr oder weniger Schaden.

Die Oberfläche des Spiels zeigt eine „Minimap“ welche dem Spieler helfen soll den Gegnern auszuweichen. Weiterhin wird die Gesundheit des Spielers angezeigt und die ausgerüstete Waffe. Sollte der Spieler von einem Gegner entdeckt worden sein erscheint unterhalb der „Minimap“ ein Fortschrittsbalken, welcher sich leert wenn der Sichtkontakt unterbrochen wird. Sobald der Balken verschwindet setzt der Soldat das patrouillieren fort.

1. Entwicklungsprozess
2. **Entwurf des Paper Prototype und der Grundidee (siehe Punkt 1)**
3. **Entwicklung der Grundelemente des Spiels und des ersten Levels:**

In der ersten Version wurden die Grundelemente des Spiels entwickelt das heißt der Spieler und der Soldat als Gegner und ein Messer welches der Spieler Aufheben kann um sich zur Wehr setzen zu können. Sowie die Türen und Schlüssel.

Die Oberfläche bestand in dieser Version aus einer Lebensanzeige und Aufmerksamkeitsanzeige in Textform und aus einem rudimentären Pause Menü.



Abbildung 2 Erster Spielentwurf

1. **Erweiterung des Spiels:**

Nachdem die Grundelemente implementiert wurden, wurde das Spiel um den Geschützturm als weiterer Gegnertyp erweitert und die Gegner bekamen eine Taschenlampe um deren Sichtfeld visuell darzustellen.

Es kamen drei neue Levels sowie ein Startmenü zum Spiel hinzu wobei speziell der erste Level als Einführungslevel dient da es bei diesem keinerlei Gegner gibt und der Spieler dadurch erstmal das Spielprinzip erlernen kann.

In dieser Version wurde die Oberfläche verbessert indem, dem Spieler eine „Minimap“ zur Verfügung gestellt wird mit deren Hilfe die Gegner früher erkannt werden können. Und die Textanzeigen für das Leben und die Aufmerksamkeitsanzeige des Gegners wurden durch einen Fortschrittsbalken ersetzt.

Des Weiteren wurden Hilfstexte hinzugefügt die z.B. erklären wie eine Waffe aufgehoben werden kann.



Abbildung 3 Zweiter Spielentwurf

1. **Grafische Anpassung des Spiels und Erweiterung des Startmenüs**

Zum Abschluss wurde den einzelnen Räumen 3d Modelle hinzugefügt sowie Texturen für Wände, Decken, und Böden um das Spiel grafisch zu verbessern.

Weiterhin wurde das Startmenü komplett überarbeitet und ein Intro, Outro, Game Over Screen und Credits hinzugefügt.



Abbildung 4 Dritter Spielentwurf



Abbildung 5 Raum in der dritten Spielversion

1. **Austausch der alten Türme durch neue 3d Modelle.**



Abbildung 6 Alte Türme



Abbildung 7 Neue 3d Modelle

1. Codestruktur

3.1. Gegner



**EnemyBase:** Die Basisklasse enthält die Standardvaribablen die jeder Gegner besitzt wie z.B. eine Waffe und für das Sichtfeld. Des Weiteren die Funktion „LookAtPlayer“ und „LocateTarget“ um den Spieler zu finden und zu folgen.

protected virtual bool LocateTarget()

{

    RaycastHit hit;

    Ray ray = new Ray();

ray.origin = new Vector3(transform.position.x,

transform.position.y + 0.5f,

transform.position.z);

    for (int i = this.VisualAngle / (-2);

 i < this.VisualAngle / 2; i += this.VisualAngleIncrement)

    {

        ray.direction = Quaternion.AngleAxis(i, transform.up) \* transform.forward;

        Debug.DrawRay(ray.origin, ray.direction \* this.VisualRange, Color.yellow);

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

        {

            if (hit.transform.gameObject.name == Player.PlayerName)

            {

                return true;

            }

        }

    }

    return false;

}

**EnemyTower:** Erbt von der EnemyBase Klasse überschreibt allerdings die Funktion „LookAtPlayer“ da die Drehung nur in einem bestimmten Radius erfolgen kann. Die Funktion wird als Coroutine gestartet wenn der Spieler gefunden wurde. Weiterhin enthält die Klasse die Funktion FixedUpdate in der, der Turm gedreht wird und der Angriff auf den Spieler gestartet wird sobald er gesichtet wurde.

**EnemySoldier:** Der Soldat patrouilliert so lange eine über Wegpunkte festgelegte Route bis er den Spieler mithilfe der „LocateTarget“-Funktion findet. Sobald der Spieler gefunden folgt er diesem. Sollte der Kontakt abbrechen läuft ein Timer ab, welcher durch einen Fortschrittsbalken visuell dargestellt wird. Nach dem dieser abgelaufen ist folgt der Soldat wieder seiner vorgegebenen Route. Die Navigation in der Welt wird durch den „Nav Mesh Agent“ sichergestellt.

private void WalkToWayPoints()

{

    Vector3 target = new Vector3(this.Waypoints[this.\_currentWaypoint].position.x,

                                 this.transform.position.y,

                                 this.Waypoints[this.\_currentWaypoint].position.z);

    Vector3 delta = target - transform.position;

    if (delta.magnitude <= 1)

    {

        if (this.\_currentWaypoint < (this.Waypoints.Count - 1))

        {

            this.\_currentWaypoint++;

        }

        else

        {

            this.\_currentWaypoint = 0;

        }

    }

    else

    {

        this.\_navMeshAgent.SetDestination(target);

        this.SetSpeed(this.\_walkSpeed);

    }

}

3.2 Waffen



**Weapon:** Die abstrakte Klasse „Weapon“ enthält Variablen die den Namen, den Schaden und die Reichweite festzulegen. Des Weiteren legt ein bool Wert fest ob die Waffe angreift oder nicht.

**Knife:** In dieser Klasse wird nur die „AttackTarget-“ Funktion der Basisklasse überschrieben. Und das Messer Richtung Bildschirmmitte bewegt, wo sich das Fadenkreuz befindet. Dabei wird die Reichweite mithilfe eines Raycasts ermittelt um zu verhindern das, dass Messer durch Wände oder ähnliches fliegt.

public override IEnumerator AttackTarget()

{

    Camera cam = Camera.main;

    this.transform.rotation = cam.transform.rotation;

    Ray ray = new Ray(transform.position, transform.forward);

    Debug.DrawRay(ray.origin, ray.direction \* this.Range, Color.yellow, 1);

    float realRange = base.Range;

    RaycastHit hit;

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

    {

        Vector3 tmp = hit.transform.position - transform.position;

        realRange = tmp.magnitude - 1;

    }

    if (realRange > 0)

    {

        this.transform.position = cam.transform.position;

        this.transform.position += transform.forward \* realRange;

    }

    else

    {

        this.transform.rotation = transform.parent.rotation;

    }

    yield return new WaitForSeconds(0.3f);

    this.transform.position = this.transform.parent.position;

    this.transform.rotation = this.transform.parent.rotation;

}

**Gun:** Das „Gun-“ Skript leitet von „Weapon“ ab und legt den Angriff für Schusswaffen fest. Die Waffe schießt nur wenn die Attack Variable der vererbenden Klasse gesetzt ist. Weiterhin enthält die Klasse Werte welche die Munition sowie deren Geschwindigkeit und die eigentliche Schussgeschwindigkeit bestimmt.

public override IEnumerator AttackTarget()

{

    while (true)

    {

        if (this.Attack)

        {

            Vector3 position = new Vector3(transform.position.x +

 transform.forward.x \* 0.12f, transform.position.y + 0.12f, transform.position.z);

            GameObject bulletGob = Instantiate(this.BulletPrefab,

                                               position,

                                               transform.rotation) as GameObject;

            Bullet bullet = bulletGob.GetComponent<Bullet>();

            bullet.InstantiateBullet(this.Damage, this.Range, this.BulletSpeed);

        }

        yield return new WaitForSeconds(100 / this.ShootingSpeed);

    }

}

**Bullet:** Die eigentliche Bewegung der Kugel wird in diesem Skript vorgenommen und es wird überprüft ob die Kugel gegen eine Wand oder ein anderes Objekt fliegt. Sollte dies der Fall sein wird die Kugel zersört.

private void Update()

{

    rigidbody.MovePosition(transform.position + transform.forward \* this.\_speed \* Time.deltaTime);

    Vector3 delta = (this.\_startPosition + (transform.forward \* this.\_range)) - this.rigidbody.position;

    Ray ray = new Ray(transform.position, transform.forward);

    Debug.DrawRay(ray.origin, ray.direction, Color.yellow);

    RaycastHit[] hits = Physics.RaycastAll(ray, Bullet.rayDistance);

    if (delta.magnitude <= 1

        || hits.Select(s => s.transform.tag).Contains(Bullet.destroyhitTag))

    {

        Destroy(this.gameObject);

    }

}

3.3 Nachrichtensystem



**Message:** Stellt den Text und die Bedingung für die Anzeige dar. Sollte der Trigger des Messageobjekts vom Spieler ausgelöst werden sendet das Message Objekt eine Anfrage zur Anzeige ans MessageSystem.

**MessageSystem:** Zeigt die Nachrichten auf dem Bildschirm an und entscheidet welche Nachricht angezeigt wird. Dazu wird überprüft ob es bereits eine aktive Nachricht existiert und ob die Bedingung der Nachricht zutrifft. Die Nachricht wird nur eine bestimmte Zeit angezeigt dies wird mithilfe einer Zähl Variable verwirklicht sobald diese auf 0 ist verschwindet die Nachricht.

**MessageInitalize:** Legt zu beginn jedes Levels die Nachrichten und die entsprechenden Bedingungen fest, wie z.B. das existieren eines bestimmten Gegners. Dabei erfolgt die zu Ordnung zum entsprechenden GameObject über einen festgelegten Schlüssel.

    void Start()

    {

        switch (Application.loadedLevelName)

        {

            case "Level1":

                this.GetMessageFromGameObject("OpenDoor").SetMessage("Aktionstaste drücken um die Tuer zu öffnen");

                this.GetMessageFromGameObject("OpenKeyDoor").SetMessage(@"Um Türen mit einem farbigen Schloss zu öffnen

wird ein Schlüssel benoetigt",this.OpenKeyDoorTrigger, 400, 50);

                break;

            case "Level2":

                this.GetMessageFromGameObject("TowerButton").SetMessage("Aktionstaste drücken um die Türme abzuschalten",

                                                                         this.TowerButtonTrigger);

                break;

            case "Level4":

                this.GetMessageFromGameObject("Weapon").SetMessage("Aktionstaste drücken um die Waffe aufzuheben",

                                                                       this.WeaponTrigger);

                this.GetMessageFromGameObject("Attack").SetMessage("Aktionstaste drücken um die Türme abzuschalten",

                                                                       this.AttackTrigger);

                break;

            default:

                break;

        }

    }

3.4 Buttons



**Button:** Die abstrakte Klasse stellt eine Funktion für das drücken eines „Buttons“ bereit und gibt an ob dieser bereits gedrückt wurde zusätzlich ist der allgemeinen Tag für Buttons enthalten.

**ButtonEnemyTower:** Ein spezieller Button zum Deaktivieren der Türme. Dafür ist eine Liste aller zugehörigen Türme enthalten bei welchen bei Betätigung die Variable „IsActivated“ auf „false“ gesetzt wird.



Abbildung 8 Startmenü



Abbildung 9 Spieloberfläche

1. Ausblick/ Erweiterungsmöglichkeiten

Erweiterungsmöglichkeiten wären an erster Stelle weitere Level, Gegnertypen und eine Story. Außerdem mehr Waffen für den Spieler und sammelbare „Medipacks“ mit deren Hilfe die Gesundheit wiederhergestellt werden kann. Auch weitere Schalter wie z.B. Türsteuerungen wären denkbar.

1. Bildverzeichnis

[Abbildung 1 Paper Prototype 2](#_Toc415584290)

[Abbildung 2 Erster Spielentwurf 2](#_Toc415584291)

[Abbildung 3 Zweiter Spielentwurf 3](#_Toc415584292)

[Abbildung 4 Dritter Spielentwurf 3](#_Toc415584293)

[Abbildung 5 Raum in der dritten Spielversion 3](#_Toc415584294)

[Abbildung 6 Alte Türme 3](#_Toc415584295)

[Abbildung 7 Neue 3d Modelle 3](#_Toc415584296)

[Abbildung 8 Startmenü 11](#_Toc415584297)

[Abbildung 9 Spieloberfläche 11](#_Toc415584298)

1. Anlagen

* Klassendiagramm aller verwendeter Klassen

1. Quellen

3d Modelle

* Frédéric Maucher
* <https://www.assetstore.unity3d.com/>
* <http://www.turbosquid.com/index.cfm>
* <http://tf3dm.com/>

Texturen

* <http://cgtextures.com/>