HOCHSCHULE **TRIER**
Trier University of Applied Sciences
Informatik - Computer Science

Entwicklung eines rundenbasierten Strategiespiels

Dokumentation

NAMEN PLACEHOLDER

Interdisziplinäres Teamprojekt

Betreuer: Prof. Dr. Linda Breitlauch, Prof. Dr. Christof Rezk-Salama

Trier, DATUM PLACEHOLDER

Inhaltsverzeichnis

1

Einleitung

2

Konzeptionierung

3

Squad Builder

Der Squad Builder ermöglicht es beiden Spielern ihre Teams zusammenzustellen. Die Spieler wählen abwechselnd ihre Figuren, bis die vorgegebene Teamgröße erreicht wurde.

Die linke Hälfte des Bildschirms zeigt die Teamzusammenstellung von Spieler 1. Er kann für jede Einheit einzeln Primärwaffe, Sekundärwaffe und 2 Utility Items wählen. Die Auswahl erfolgt über eigens programmierte Dropdown-Menüs, die sich dynamisch aus den vorgegebenen Enumeratoren und Icon-Listen generieren. Das Bestätigen der Zusammenstellung erfolgt über den Bestell-Button unterhalb der linken Anzeigenleiste. Die ausgewählten Einheiten werden in der linken Anzeigenleiste durch die ihnen zugeteilte Ausrüstungsgegenstände dargestellt.

Die rechten drei Buttons ermöglichen es Spieler 2 seine Figuren aus vorgegebenen Archetypen auszuwählen. Die gewählten Einheiten werden in der rechten Anzeigenleiste durch ihre jeweiligen Icons dargestellt.

4

Levelaufbau

Das Level wird durch ein Skript (BattlefieldCreator) aufgrund der Plane generiert. Je größer der Localscale der Plane, desto größer wird das Level. Auf einer Skalierung von 1/1/1 entsteht ein 10 * 10 Grid aus Quads(Zellen). Bei Skalierungen im Komma Bereich werden dem entsprechen viele Zellen erstellt. Zum Beispiel bei x = 1,6 und z = 1,7 entsteht ein 16 * 17 Feld.

```
1  //Initialisiert alle Zellen
2  for (float z = 0; z > -(sizeZint); z--) {
3     for (float x = 0; x < (sizeXint); x++) {
4        GameObject zelle = GameObject.CreatePrimitive(PrimitiveType.Quad);
5        zelle.transform.Rotate(new Vector3(90, 0, 0));
6        zelle.AddComponent<Cell>();
7        zelle.tag = "Cell";
8        zelle.name = x + "|" + -z;
9        zelle.transform.position = new Vector3((x + 0.5f), 0.001f, (z - 0.5f)
             );
```

Die Objekte werden durch das Skript ObjectSetter beim Spielstart auf dem Grid platziert. Sollte ein Objekt größer als eine Zelle sein so wird dies in der ObjectComponent durch X und Z wert angegeben. Ebenso wird dort vermerkt ob das Objekt Deckung bietet und die Zelle besetzt.

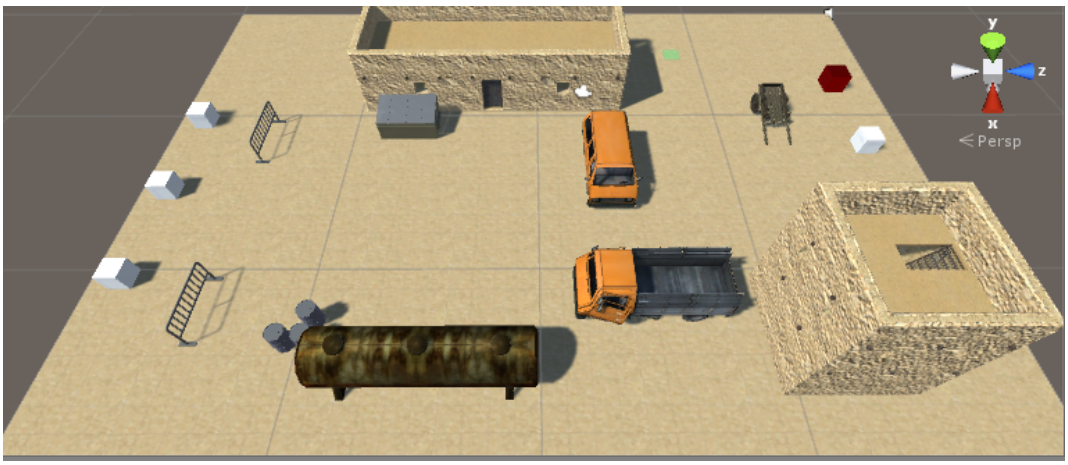

Abb. 4.1. Das finale Level mit Spieleraufstellung

5

Manager Objekt

Um die verschiedenen Systeme, die für den korrekten Ablauf der Spielzüge und allgemein spielregeltechnischen Abläufe zu handeln, wurde ein Spielobjekt, das als Manager bezeichnet wird, erstellt. Im folgenden Kapitel wird auf die einzelnen Skripte die an diesem Manager Objekt hängen genauer eingegangen.

5.1 Manager System

Das Manager System ist fÃ¼r den korrekten Ablauf der einzelnen ZÃ¼ge zustÃ¤ndig. Es zählt die Runden hoch, stellt sicher, dass nur das die Eingabe des Spielers, der aktuell an der Reihe ist, abgehandelt wird, merkt sich die aktuell ausgewählte Spielfigur, damit das User-Interface korrekt dargestellt wird, fügt jedem Spieler seine Spielfiguren zu und setzt die Spielfiguren zu Beginn der Sitzung an zuvor festgelegte Positionen.

5.2 Shooting System

5.3 Inventory System

Das Inventory System wird aufgerufen sobald ein Spieler eine der folgenden Aktionen ausführt um die Anzahl der im Inventar der Spielfigur enthaltenen Gegenstände zu verringern:

- Nachladen der Primärwaffe
- Einsatz von Handgranaten
- Einsatz von Tränengas
- Einsatz von Rauchgranaten
- Einsatz von Molotovcocktails

5.4 Player Assistance System

5.5 Ability System

5.6 Health System

6

Spieler

Das Spieler Objekt enthält als Kindobjekte seine Spielfiguren. Als Skripte hängen ihm eine Player Component, sowie eine Input Component an.

6.1 Player Component

```
1  GameObject[] figurines = new GameObject[3]; //Alle Figuren ueber die ein
       Spieler verfuegt
2  public int actionPoints = 0; //Anzahl an verfuegbaren Aktionspunkten
3  int maxAP; //Maxcap fÃ¼r AP
```

Das Skript speichert die maximale Anzahl an Aktionspunkten, die für die verschiedenen Fraktionen variieren, füllt nach dem Ende der Runde die Aktionspunkte beider Fraktionen auf und stellt sicher, dass dabei die Zahl der erhaltenen Aktionspunkte nicht die Grenze überschreiten.

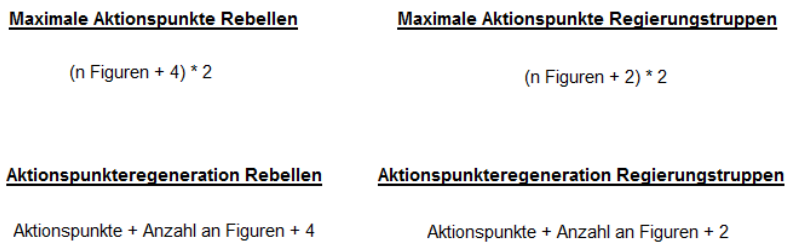

Abb. 6.1. Berechnung der Aktionspunkte

6.2 Input Component

7

Spielfiguren

7.1 Bewegung

Bei unseren Spielfiguren um kleine Plastiksoldaten handelt, die sich in einem kindlich dargestelltem Nah-Ostkonflikt befinden. Die Bewegung der Einheiten wird daher über eine Parabelkurve angedeutet, an der sich die Figur beim laufen entlang bewegt. Somit wird der Eindruck erzeugt, die Figur werde wie von einer unsichtbaren Hand in einem Brettspiel über das Feld bewegt.

7.2 Attribute Component

7.3 Inventory Component

Jeder einzelnen Spielfigur wird eine Inventory Component angehangen. In dieser wird das gesamte Inventar der jeweiligen Figur gespeichert. Das Inventory System kümmert sich dabei um die Berechnungen und Aktualisierung der Inventory Komponenten.
Es folgt ein Auszug der verschiedenen Variablen:

```
1  //Inventar (primaerwaffe, sekundaerwaffe, equipment1, equipment2) siehe
       Enums.cs
2  public Enums.PrimaryWeapons primaryWeaponType;
3  public Enums.SecondaryWeapons secondaryWeaponType;
4  public Enums.Equipment utility1;
5  public Enums.Equipment utility2;
6
7  public WeaponComponent primary; //Primaerwaffe
8  public WeaponComponent secondary; //Sekundaerwaffe
9  public bool isPrimary; //Ist Primaerwaffe ausgewaehlt?
10 public int amountSmokes; //Anzahl Rauchgranaten
11 public int amountTeargas; //Anzahl Teargas
12 public int amountGrenades; //Anzahl Granaten
13 public int amountMolotovs; //Anzahl Molotovs
14 public int amountMediKits; //Anzahl Medikits
15 public int amountMagazines; //Anzahl Magazine//Anzahl Magazine
```

7.4 Selektierte Spielfigur

Die aktuell ausgewählte Spielfigur wird durch eine diese umgebende Box gekennzeichnet.

8

Pathfinding

"Civil War Nation" benutzt ein in Zellen aufgeteiltes Spielfeld. Um die Bewegung der Figuren auf diesem Spielfeld zu ermöglichen, müssen die günstigsten Pfade gefunden werden. Hierbei wird der "Dijkstra Algorithmus" eingesetzt, der von der aktuell ausgewählten Figur die Entfernung zu allen anderen Zellen auf dem Spielfeld zu berechnen. Diese Entfernung wird wiederum benutzt um Aktionen mit begrenzter Reichweite, wie schießen, Granaten werfen, oder Laufen, auf ihre Verfügbarkeit zu überprüfen. Der Dijkstra Algorithmus wurde gegenüber dessen Erweiterung, den A*-Algorithmus gewählt, da wir ungerichtet über den Graphen laufen möchten, und somit die Kosten zu allen umliegenden Knoten erhalten möchten.

9

Kamera

Die Kamera ist eine beweglich Orbitkamera mit Zoom und Rotation die sich nur innerhalb des Levels bewegen kann. Realisiert wird dies durch einen konstanten Focus auf ein bewegliches, unsichtbares Objekt. Die Kamera lässt sich über das Feld bewegen in dem die Maus an den jeweilige Rand bewegt wird der in die gewünschte Richtung führt.

Die folgende Funktion prüft ob die Kamera im Feld ist:

```
1     public bool inBattlefield()
2     {
3        bool inField = true;
4        if (target.transform.position.x < 0) {
5           inField = false;
6           target.transform.position = new Vector3(0, target.transform.
                position.y, target.transform.position.z);
7        }
8        if (target.transform.position.z > 0) {
9           inField = false;
10          target.transform.position = new Vector3(target.transform.position.x
                , target.transform.position.y, 0);
11       }
12       if (target.transform.position.x > (plane.transform.position.x * 2)) {
13          inField = false;
14          target.transform.position = new Vector3((plane.transform.position.x
                 * 2), target.transform.position.y, target.transform.position.z)
                ;
15       }
16       if (target.transform.position.z < (plane.transform.position.z * 2)) {
17          inField = false;
18          target.transform.position = new Vector3(target.transform.position.x
                , target.transform.position.y, (plane.transform.position.z * 2))
                ;
19       }
20    return inField;
21    }
```

10

User-Interface

Das UI besteht aus verschiedenen Komponenten.

10.1 Action-Points Leiste

Die Aktionspunkte Leiste am oberen Bildrand zeigt fÃ¼r beide Spieler die maximalen sowie die aktuell verfÃ¼gbaren Aktionspunkte an

10.2 Dynamische Ability-Icons

Wenn ein Spieler eine Einheit auswÃ□hlt, so werden am unteren Bildrand die erforderlichen Aktionsbuttons angezeigt. Es werden nur die Buttons dargestellt, deren Aktionen von der ausgewÃ□hlten Figur durchgefÃ¼hrt werden kÃ¶nnen.

10.3 HP Leisten

Durch drÃ¼cken der Leertaste kÃ¶nnen fÃ¼r alle Figuren Segmentanzeigen dargestellt werden, die die aktuellen Lebenspunkte wiederspiegeln. Jedes Segment steht dabei fÃ¼r 10 Lebenspunkte.

11

3D Modelling

Sowohl die Charaktere als auch Assets wurden ausschließlich in Blender gemodelt. Hierbei wurde sich stark an reellen Vorgaben, was Kleidung oder einprägsame Details betrifft, orientiert. Um den angestrebten Lowpoly-Stil konstant umzusetzen wurden teilweise erst Highpoly-Modelle erstellt um diese in der sogenannten “retopology“ später detailarmer zu gestalten.

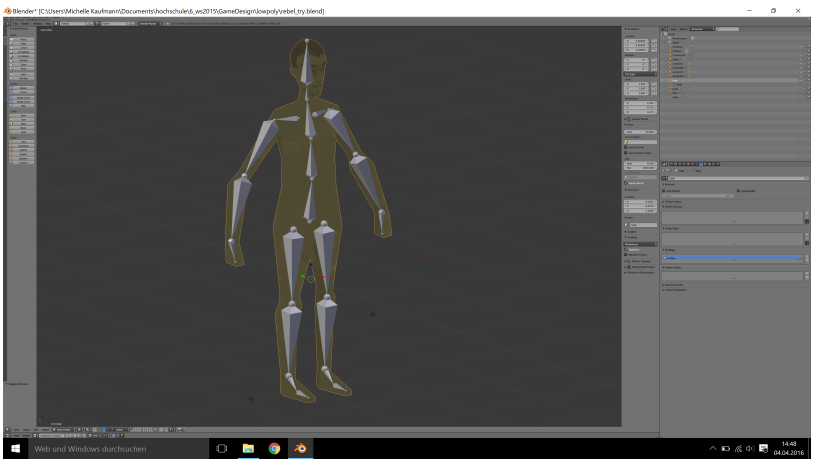

Abb. 11.1. Charaktermodell in Blender

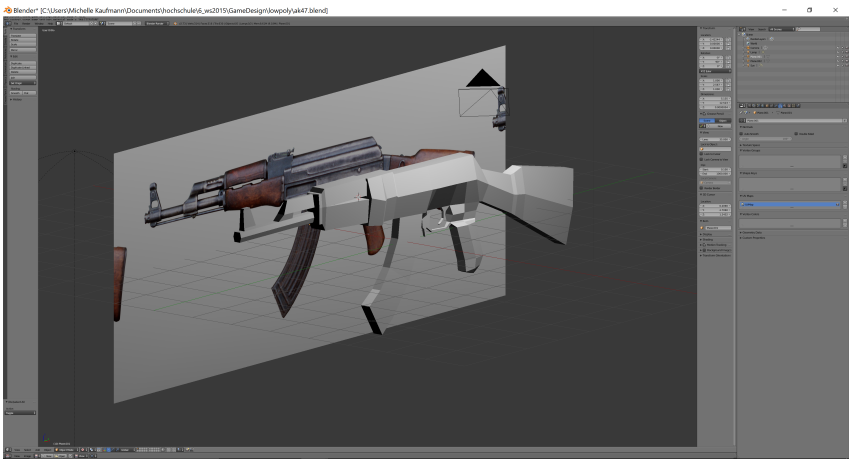

Abb. 11.2. Waffenmodellierung in Blender

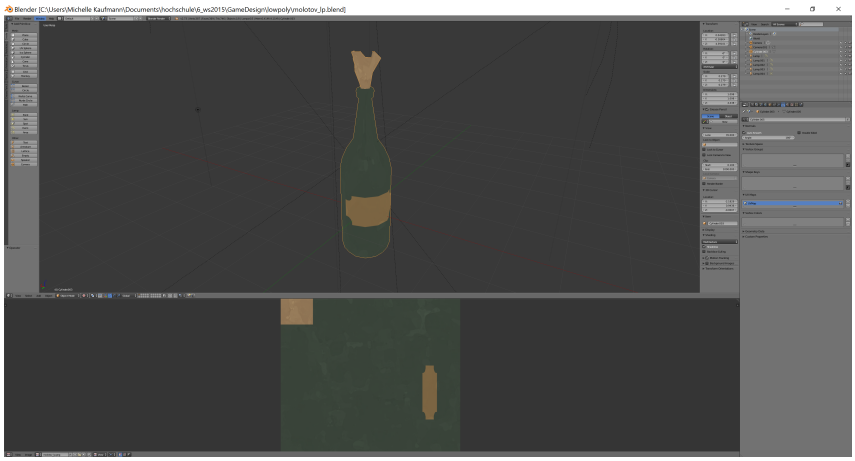

Abb. 11.3. Propmodellierung in Blender

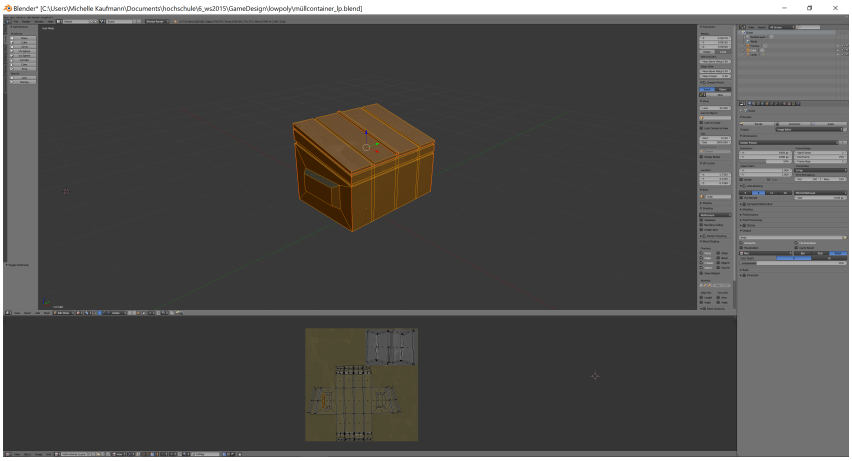

Abb. 11.4. Propmodellierung in Blender

12

Animationen

12.1 Motion Capture Aufnahmen

Alle eingebundenen Animationen wurden zuvor von den Projektmitgliedern mit dem Motion Capture System im Keller der Hochschule Trier aufgenommen. Nachdem eine überschaubare Anzahl an qualitativ hochwertigen Aufnahmen ausgewählt wurden, gingen diese direkt als .bvh Format zur Weiterverarbeitung.

12.2 Einbindung der Animationen

Die Einbindung der Animationen wurde wie das Modelling ebenfalls direkt in Blender vorgenommen. Mithilfe des Add-Ons “MakeWalk“ ist es einfach, falls der selbst erstellte Rig keine Fehler aufweist, die rohen Motion Capturing aufnahmen in die 3D Software zu übertragen. Für jede Charakterklasse im Spiel wurde eine eigene “Pose Libraries“ erstellt. Diese erwiesen sich durch einen strukturiertere und übersichtlichere Importierung in Unity als äußerst nützlich.

Diese Animationen wurden in Unity importiert, und mit Hilfe eines Animators in eine Animation State Machine überführt, die je nach gewählter Aktion und aktueller Haltung (Einhändige Waffe, Zweihändige Waffe, Nahkampfwaffe, Einsatzschild) , die korrekte Animation auswählt und abspielt.

Zusätzlich wurden an einige der Animationen an bestimmten Zeitpunkten in der Animationen Funktionen angehängt. Somit kann erreicht werden, dass beispielsweise der Schusssound zum korrekten Zeitpunkt ausgelöst wird, oder die Granaten im entsprechenden Frame erstellt, oder geworfen werden.

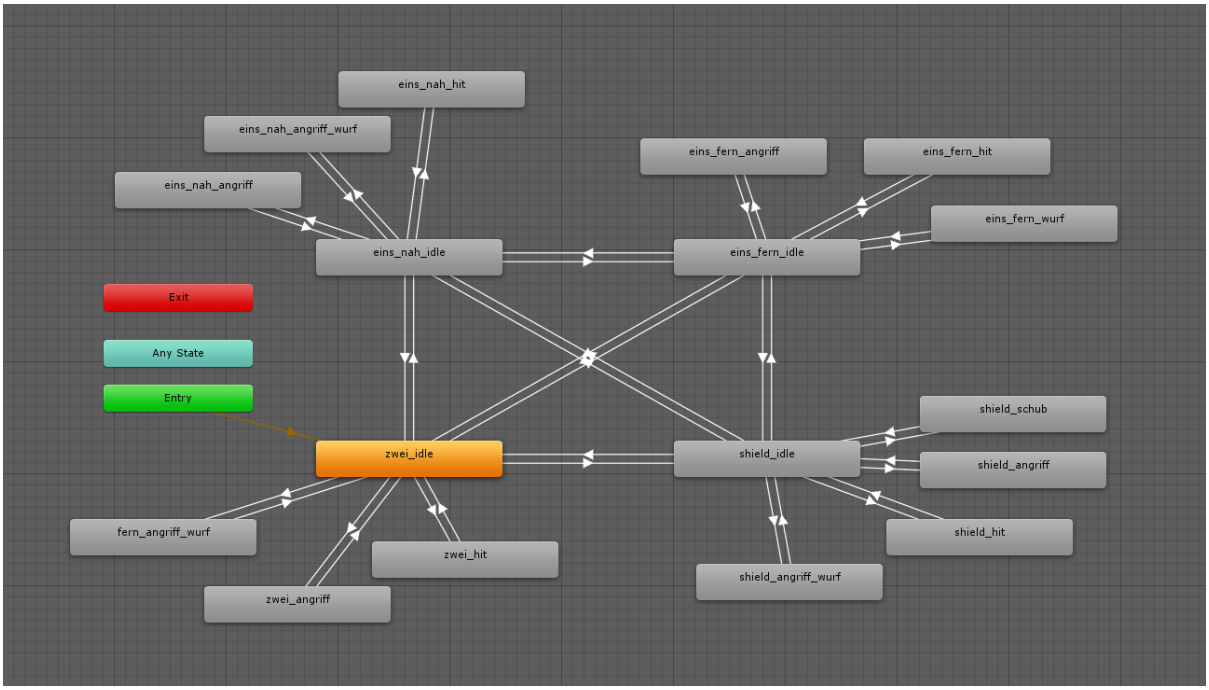

Abb. 12.1. Animation Tree der Polizei Charaktere

13

Sounds

14

Effekte

15

Status quo

Cursor Feedback welche Aktion ausgewaehlt wird.

16

Fazit