HOCHSCHULE TRIER
Trier University of Applied Sciences
Informatik - Computer Science

Entwicklung eines 2D Plattformers

Dokumentationn

Fabio Gimmillaro, Daniel Schreiber, Tobias Rühl und Joscha Wülk

Medienprojekt

Betreuer: Prof. Dr. Christof Rezk-Salama

Trier, Abgabedatum

Inhaltsverzeichnis

1

Konzeptionierung

2

Die Kamera

3

GameObject “Player“ & Komponenten

Die Spielfigur, die der Anwender bedient stellte sich während der Entwicklung als komplexestes GameObject heraus. In diesem Kapitel geben wir daher einen Überblick über die Realisierung mithilfe von Sprites sowie der verschiedenen anhängenden und selbsterstellten Skripte

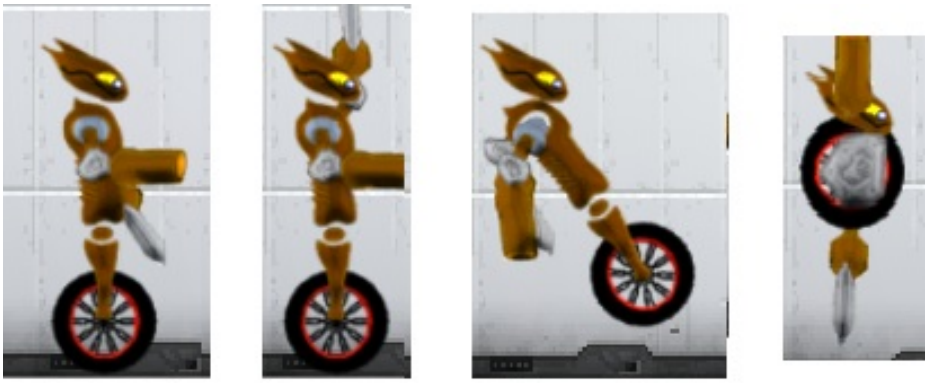

Abb. 3.1. Zusammenschnitt der Player Sprites

3.1 SpriteRenderer

3.2 AttributeComponent Skript

Die AttributeComponent dient dazu die spielmechanischen Daten zu hinterlegen und zu verwalten. Sie kann sowohl dem Spieler als auch freundlich und feindlich Gesinnten NPCs angehÃ¤ngt werden.

Nachfolgend werden die wichtigsten Daten vorgestellt:

```
1  //Lebenspunkte, maximale Lebenspunkte, Ruestung und Schaden
2  public float health, maxHealth, armor, damage;
3  //Maximale Ausdauer, Ausdauer und regenerierte Ausdauer pro Sekunde
4  public float maxStamina, stamina, staminaPerSecond;
5  //Munition, Munitionskapazitaet und Reichweite
6  //Range = 0 -> Nahkampf / Range > 0 -> Fernkampf
7  public int ammo, ammoCap, range;
```

```
8  //Klon aktiv?
9  public bool cloneAlive = false;
10
11 //Cooldown fÃ¼r Plasmaschuss
12 static float cooldown1 = 1.0f;
13 //Laeuft cooldown fÃ¼r Plasmaschuss?
14 bool cooldown1Active = false;
15
16 //Cooldown fuer Klonfaehigkeit
17 static float cooldown2 = 10.0f;
18 //Time-to-live fÃ¼r Klon
19 static float ttl = 5.0f;
20 //Aktuelle Time-To-Live
21 float attl = ttl;
22 //Laeuft cooldown fÃ¼r Klon?
23 bool cooldown2Active = false;
24
25 //Referenz auf das Nahkampfsystem
26 MeleeSystem meleeSys;
```

Die Update-Funktion der AttributeComponent wird ausschließlich dazu benutzt, die Stamina über Zeit aufzufüllen während die fixedUpdate-Funktion fÃ¼r das lÃ¶schen des Klones verantwortlich ist.

```
1  void Update () {
2
3  /*Fuelle Ausdauer ueber Zeit wieder auf solange maximalStamina nicht
       erreicht ist und die
4  Spielfigur sich nicht bewegt*/
5  if (staminaPerSecond > 0.0f && stamina < maxStamina && !meleeSys.
       animationRunning)
6  {
7  //Stelle sicher, dass Stamina nicht kleiner als 0 oder groesser als
       maximalStamina gesetzt wird
8  stamina = Mathf.Clamp(stamina+staminaPerSecond * Time.deltaTime,0,maxStamina
       );
9  }
10 }
11 void FixedUpdate()
12 {
13 if (attl > 0)
14 attl -= Time.deltaTime;
15 if (attl <= 0 && cloneAlive)
16 {
17 Destroy(GameObject.Find("Klon"));
18 cloneAlive = false;
19 }
20 }
```

3.3 CharacterMovement Skript

3.4 MeleeSystem Skript

3.5 HealthSystem Skript

4

Enemies

In unserem Spiel waren zwei sich ähnliche Gegner angedacht, wovon einer als Fernkämpfer und der andere als Nahkämpfer fungierte. Auf Grund mangelnder Animationen hat lediglich der Fernkämpfer seinen Weg ins Spiel gefunden. Von der Implementierung hätten sich die beiden Gegnertypen aber kaum unterschieden, da die KI möglichst modular und austauschbar gehalten werden sollte. Somit unterscheiden sich die zwei Typen lediglich durch eine andere Reichweite für den Angriff, und eine andere Angriffsfunktion, die aufgerufen wird.

Die Gegner wurde eine simple Entscheidungskaskade in Form von verschiedenen If-Anweisungen gegeben, nach denen die Gegner dann ihr Verhalten auswählen. Das Verhalten der Gegner ist stark davon abhängig ob Sichtkontakt zum Spieler besteht, was durch Raycasts zwischen dem Gegner und dem Spieler überprüft wird. Hierbei kann auch ein Blickwinkel definiert werden, in welchem der Spieler wahrgenommen wird.

```
1  [...]
2  //Blickrichtung entweder x=1 oder -1
3  if (actions.facingRight)
4      viewVector += new Vector2(1, 0);
5  else
6      viewVector += new Vector2(-1, 0);
7
8  //Winkel zwischen Blickrichtung und Verbindungsvektor
9  float currentAngle = Vector2.Angle(viewVector, difference);
10 RaycastHit2D hit = Physics2D.Raycast(visionCheck.position,viewVector,
       noticeDistance);
11
12 //Ueberpruefen ob der Winkel maximal dem angegebenen Field of View Winkel
       ist
13 if (currentAngle <= fovAngle)
14 {
15     //Raycast der von Augen zum Ziel geschossen wird, wenn das erste
           getroffene Objekt der Spieler ist, ist die sicht nicht blockiert
16     if (hit.collider != null)
17         playerVisible = hit.collider.gameObject == rigplayer.gameObject;
18    else
19       playerVisible = false;
20 }
21 else
22    playerVisible = false;
```

Mit Hilfe dieses Sichtbarkeitschecks, kann die KI verschiedene Aktionen abhängig von verschiedenen Bedingungen wählen. Diese Entscheidungskaskade könnte man als Behaviour Tree folgendermaßen visualisieren:

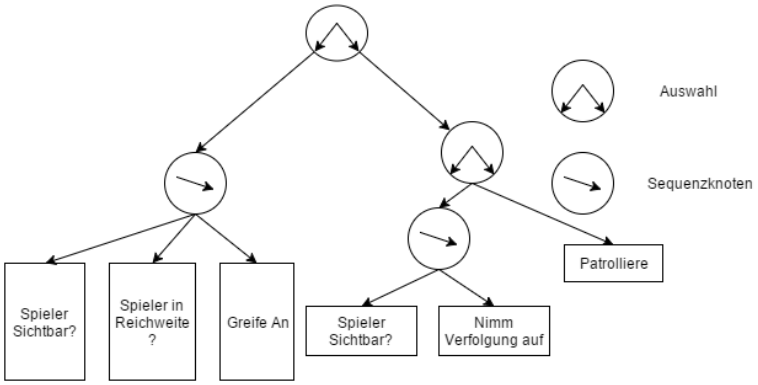

Abb. 4.1. Verhalten des Gegners als Behaviour Tree

Implementiert wurde der Entscheidungsalgorithmus jedoch nur mit if-Bedingungen und nicht mit einem Behaviour Tree oder einer State Machine.

```
1  //Wenn in Angriffsreichweite, und der Spieler sichtbar ist, bleibe stehen
       beginne anzugreifen
2  //Ruft die Angriffsmethode des RangedSystems auf
3  if ((inAttackRangex && inAttackRangey) && vision.playerVisible)
4  {
5     movement.move(0.0f);
6     anim.SetBool("AttackInProgress", true);
7     StartCoroutine(rangeSys.shoot(true));
8  }
9
10 //Wenn sich Spieler nicht in Reichweite zum Angreifen befindet, aber
       Sichtkontakt besteht, nimm Verfolgung auf
11 //Ruft die Bewegungsmethode des Movementssystem auf.
12 if (!inAttackRangex && inAttackRangey && vision.playerVisible)
13 {
14    if (movement.grounded)
15    {
16       if (walkingRight)
17          movement.move(1.0f);
18       else
19          movement.move(-1.0f);
20    }
21    else
22       movement.move(0.0f);
23 }
24
25
26 //Wenn das Ziel nicht sichtbar ist, soll patrolliert werden
27 if (!vision.playerVisible)
28 {
29    //Wenn wir auf dem Boden befinden, beweg dich, ansonsten bleib stehen
30    if (movement.grounded)
31    {
32       //Befinden wir uns vor einer Wand oder einem Abgrund, drehe um.
33       if (hittingWall || onAnEdge)
34          walkingRight = !walkingRight;
35       if (walkingRight)
36       {
37          movement.move(1.0f);
```

```
38            }
39            else
40            {
41                movement.move(-1.0f);
42            }
43        }
44        else
45            movement.move(0.0f);
46    }
```

5

Endboss

Der Kampf mit dem Endboss, welcher auf der obersten Plattform des Levels zu finden ist, wurde speziell für den Bosskampf gescriptet. Er erscheint über eine Colliderbox, die beim betreten den Spawn des Bosses startet. Nach Ablauf eines kurzen Timers beginnt dieser zu erscheinen.

Im Kampf selbst wurden ähnliche Logikabfragen wie bei der normalen Gegner KI implementiert, jedoch in einer spezielleren Form, da sich durch gewünschte Bossfähigkeiten diese nicht mit dem normalen KI Script umsetzen lassen.

Der Boss hat 3 Angriffe die von bestimmte Bedingungen haben:

Schlag mit linker Hand Wenn sich der Spieler in der Trefferzone der linken Hand befindet

Schlag mit rechter Hand Wenn sich der Spieler in der Trefferzone der rechten befindet

Schreien Keine Bedingung. Lässt den Spieler für eine gewisse Zeit bewegungsunfähig werden.

Damit diese drei Angriffe nicht etwa in statischer Reihenfolge ausgeführt werden, und etwas mehr Dynamik in den Kampfablauf bringen, wurde hier ein Zufallsalgorithmus implementiert. Dieser entscheidet zufällig welcher der möglichen Angriffe gewählt werden soll, unter Berücksichtigung der letzten ausgeführten Aktionen. Eine der möglichen Aktionen wird proportional zu ihren vergangenen Aktionsphasen, seit der letzten Verwendung immer wachsen. Beim ersten Angriff besteht also für jede Funktion eine Chance von $\frac{1}{3}$. Wird dann beispielsweise der Schlag mit der linken Hand ausgeführt, ist die Chance für nochmals diese Aktion $\frac{1}{5}$, für den Schrei und den Schlag mit der rechten Hand jeweils $\frac{2}{5}$.

Besonderheit hierbei ist, dass die Schläge zweimal hintereinander ausgeführt werden, der Zähler wird lediglich auf 1 zurückgesetzt, während es ausgeschlossen ist, dass zweimal ein Schrei ausgeführt wird.

Beispielsweise der Code für eine der gewählten Aktionen und für die Berechnung der Wahrscheinlichkeiten.

```
1  void RightHandHit()
2  {
3      //Variablen fuer den Code, welche attacke benutzt werden soll
4      timesSinceScream++;
5      timesSinceLeftHand++;
6      timesSinceRightHand = 1;
7
8      //Variablen fÃ¼r die Animation
9      idleStateExecuted = false;
10     anim.SetTrigger("RightHandAttack");
11 }
12
13
14 void getProbabilities()
15 {
16     int options = timesSinceScream;
17     if (inLeftHitBox)
18     {
19         options += timesSinceLeftHand;
20     }
21     if(inRightHitBox)
22     {
23         options += timesSinceRightHand;
24     }
25     screamProb = (float)timesSinceScream / (float)options;
26
27     leftHandProb = inLeftHitBox ? (float)timesSinceLeftHand / (float)options
           : 0.0f;
28     rightHandProb = inRightHitBox ? (float)timesSinceRightHand / (float)
           options : 0.0f;
29 }
```

Die Entscheidungen des Bosskampfes in einem Behaviour Tree dargestellt könnte folgendermaßen aussehen:

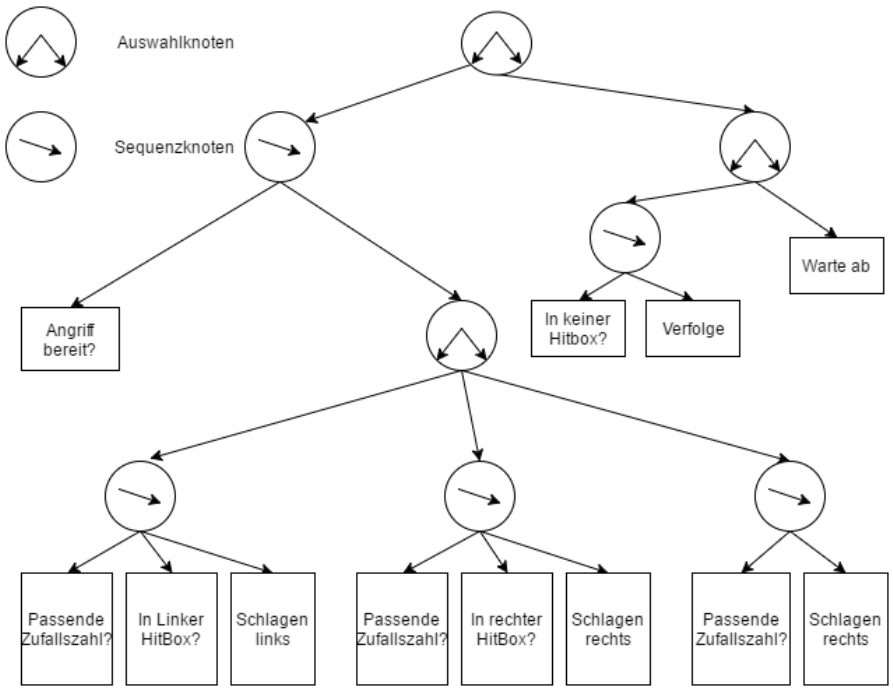

Abb. 5.1. Verhalten des Endboss' als Behaviour Tree

6

Grafiken und Animationen

7

Sounds

8

Spielobjekte

8.1 Truhe

Eine simple Truhe die mit der Taste “E“-Taste geöffnet werden kann, solange man sich im Collider befindet. Hierbei wird aus einer Liste an GameObjects zu”fallig ausgewählt und aus der Kiste gedroppt.

8.2 Pickups

Es gibt verschiedene Pickups, wie etwa Health Potions die Leben regenerieren, oder Munition für den Spezialschuss. Diese sind durch den Collider aufsammelbar, und schweben an der Stelle wo sie liegen leicht auf und ab. Dies wurde mit Hilfe einer Smoothstep Funktion umgesetzt, die in einer glatten Kurve zwischen zwei Extrempunkten interpoliert.

```
1  //Smoothstep Funktion die einen Wert der zwischen 0 und 1 oszilliert "
       glaettet" um eine gleichmaessige Kurve zu erzeugen
2  void smoothStep(float parameter)
3  {
4      xParameter =0.5f* (1.0f - Mathf.Cos(Mathf.PI * parameter * xFrequency));
5      yParameter =0.5f* (1.0f - Mathf.Cos(Mathf.PI * parameter * yFrequency));
6  }
7
8  //Letztendliche Berechnung der Position
9  void Interpolate()
10 {
11     float currentX = minX + xParameter * (maxX - minX);
12     float currentY = minY + yParameter * (maxY - minY);
13
14     this.transform.position = new Vector2(currentX, currentY);
15 }
```

9

Events

10

Demolevel

11

Menüführung

12

Head-Up-Display & Tooltips

13

Zielsetzung

14

Abschlussanalyse