**Dungeon** **Crawler**

Ich habe nach dem Tutorial mein Dungeon-Crawler-Projekt noch um folgende Aspekte erweitert:

* Leicht veränderte Steuerung des Spielers
* Vergrößerung des Dungeons
* Weiteres Item: Axt als Nahkampfwaffe
* Lebensanzeige über den Gegnern
* Neuer Gegner: Zombie mit mehr Leben und mehr Schaden

Steuerung des Spielers:

Ich habe den PlayerController so verändert, dass sich der Spieler, solange er die Tasten zum Drehen („Q“ und „E“) drückt, umdreht. Er dreht sich also nicht direkt um 90°, sondern um einen einstellbaren Wert, solange die entsprechenden Tasten gedrückt werden. Das habe ich mit Hilfe der Abfrage „if(Input.GetKey (KeyCode.Q))“ erreicht, die „true“ zurückgibt, solange die angegebene Taste gedrückt wird. Neben dem horizontalen Drehen kann sich der Spieler mit dem Mausrad nach oben und unten umschauen. Dies habe ich durch die Abfrage „Input.GetAxis("Mouse ScrollWheel")“ umgesetzt. Ich habe also die x-Koordinate der Rotation auf den zurückgegebenen Wert von „Input.GetAxis("Mouse ScrollWheel")“ multipliziert mit „scrollValue“ gesetzt. Dadurch ist die Intensität der vertikalen Rotation veränderbar. Außerdem habe ich ein Fadenkreuz in der Mitte des Bildschirms platziert, damit der Spieler mit den Wurfsteinen besser zielen kann.

Vergrößerung des Dungeons:

Da ich eine neue Waffe und den Zombie zum Spiel hinzugefügt habe, habe ich das Dungeon auch um eine Waffenkammer (mit der Axt) und einen weiteren Raum (mit dem Zombie) erweitert. Der zusätzliche Raum ist mit einem Tor verschlossen, welches sich durch das „WaterQuest“-Script, nachdem die Quest erfüllt wurde, öffnen lässt. Dafür habe ich mir mit „door = GameObject.FindGameObjectWithTag("OpenDoor").GetComponentInChildren<AnimateDoor>();“ eine Referenz zum Script des Tors besorgt, um es dann mit „door.isLocked = false;“ zu öffnen.

Axt als Nahkampfwaffe:

Ich habe mir die Axt als Prefab aus dem Asset-Store heruntergeladen. Ich habe ihr einen Box-Collider und die Scripts „HoverEffect“ und „PickableItem“ hinzugefügt. Dann habe ich ein neues Inventory-Item zur Axt erstellt, sodass man sie in das Inventar des Spielers aufnehmen kann. Die Axt kann nur dem Zombie Schaden zufügen, da sie die Fledermäuse mit einem Schlag töten würde und so die Herausforderung verloren geht. Alles zur Axt habe ich in einem neuen Script („AxeBehaviour“) programmiert. Für einen Schlag mit der Axt wird beispielsweise überprüft, ob die linke Maustaste gedrückt wird, der Spieler die Axt im Inventar hat und ob die Entfernung zum Zombie gering genug ist (da es eine Nahkampfwaffe sein soll). Ist das alles der Fall, wird vom Spieler aus in seine Blickrichtung ein „ray“ losgeschickt und geprüft, ob dieser auf den Zombie trifft. Zum Schluss wird dem Zombie der Schaden zugefügt und nach zwei Sekunden und einer kleinen Animation kann erneut zugeschlagen werden.

Lebensanzeige über den Gegnern:

Zur besseren Einschätzung des Gesundheitszustandes der Gegner (vor allem dem Zombie) habe ich eine Lebensanzeige der Gegner zum Spiel hinzugefügt. Im Prinzip soll sie wie die Anzeige beim Spieler aussehen, weshalb ich die gleichen Texturen in etwas kleiner verwendet habe. Dazu habe ich einfach das EnemyHealth Script erweitert, da hier der Gesundheitszustand, auf den ich schließlich zugreifen muss, verwaltet wird. Im Script habe ich die „OnMouseEnter“ Methode ergänzt. Sie wird (wie beim „HoverEffect“-Script) aufgerufen, sobald die Maus auf das GameObject zeigt. Dort wird dann die Lebensanzeige 20 Pixel über der Maus eingeblendet. Außerdem wird mit dem Befehl „InvokeRepeating(„UpdateView“, 0.0f, 0.3f)“ die Methode UpdateView alle 0.3 Sekunden aufgerufen. Das hat den Grund, dass die Anzeige aktualisiert werden soll, auch wenn der Spieler die ganze Zeit mit der Maus auf das Objekt zeigt. Wenn die Maus das Objekt nun verlässt, werden durch die Methode „OnMouseExit“ der oben genannte Befehl gestoppt und die Anzeige ausgeblendet.

**Neuer Gegner: Zombie – Künstliche Intelligenz**

Ich habe mir den Zombie als Prefab aus dem Asset Store heruntergeladen. In dem Prefab waren die fertige Grafik und die Animationen dabei. Ich habe jedoch einen eigenen Animation Controller für den Zombie erstellt, der dem BatController ähnelt. Ich habe die „Idle“-Animation zu „Hit“ umbenannt und wie die „Hit“-Animation der Fledermaus verwendet. Die „Fallingback“-Animation ersetzt die „Dying“-Animation und die „Walk“-Animation die „Flying“-Animation. Die Transitions habe ich dann entsprechend denen der Fledermaus angepasst, da sie zu meiner Vorstellung des Zombies gepasst haben. Außerdem habe ich die speed Variable der „Walk“-Animation auf 0.7 gestellt, sodass die Animation zur Geschwindigkeit des Zombies passt. Dieser soll nämlich langsam und zufällig, durch den Raum, in dem er sich befindet, patrouillieren. Dafür habe ich dem Prefab als erstes die NavMeshAgent-Komponente hinzugefügt. Dann habe ich dort einige Parameter verringert, sodass sich der Zombie langsamer bewegt:

Speed = 0.4 (Geschwindigkeit, mit der sich der Zombie fortbewegt),

Angular Speed = 80 (Geschwindigkeit, mit der sich der Zombie dreht),

Acceleration = 0.5 (Beschleunigung).

Für die Bewegungssteuerung habe ich zunächst das „EnemyAI“-Script dupliziert und in „ZombieAI“ umbenannt. Dort habe ich dann die Methode „WaypointWalk“ mit der Methode „RandomWalk“ ersetzt:

if (agent.remainingDistance <= agent.stoppingDistance)

{

Vector3 position =

new Vector3(Random.Range(upperLeftCorner.position.x, lowerRightCorner.position.x), 0,

Random.Range(lowerRightCorner.position.z, upperLeftCorner.position.z));

StartCoroutine("WaypointPause", waypointPauseTime);

agent.SetDestination(position);

}

Die if-Abfrage „if (agent.remainingDistance <= agent.stoppingDistance)“ habe ich aus dem „EnemyAI“-Script übernommen. Sie gibt „true“ zurück, wenn der Zombie gleich weit oder näher an seinem Ziel ist, als die Distanz, die er für das Abbremsen und Stehenbleiben bis dorthin braucht. Die Abfrage bewirkt also, dass erst ein neues Ziel gesetzt wird, wenn der Zombie an seinem vorigen Ziel angekommen ist. Ist dies der Fall und der Zombie muss ein neues Ziel wählen, wird zuerst ein neuer lokaler Vector3 „position“ deklariert. Ihm werden folgende Koordinaten zugewiesen:

x-Koordinate: Random.Range(upperLeftCorner.position.x, lowerRightCorner.position.x),

y-Koordinate: 0,

z-Koordinate: Random.Range(lowerRightCorner.position.z, upperLeftCorner.position.z)

RandomRange(float min, float max) liefert einen Zufallswert, der zwischen den beiden angegebenen Grenzen liegt. Ich benutze diese Methode um den Aspekt des zufälligen Patrouillierens zu erhalten, da hierdurch das neue Ziel immer variiert. Die jeweiligen Grenzen habe ich den Eckkoordinaten des NavMeshs im Raum des Zombies entnommen. Dazu habe ich die zwei Variablen „upperLeftCorner“ und „lowerRightCorner“ deklariert und ihnen zwei Transforms zugewiesen, die ich in den Ecken des Dungeons platziert habe. Dadurch kann man, wenn man den Raum beispielsweise vergrößern möchte, die beiden Transforms in die neuen Ecken verschieben und das Script funktioniert immer noch. Dem Vector3 wird also ein zufälliger Punkt auf der xz-Ebene, den der Zombie betreten darf (Area Mask = Walkable), in dem Raum zugewiesen. Der Befehl „StartCoroutine("WaypointPause", waypointPauseTime);“ bewirkt das gleiche wie bei der Fledermaus und „agent.SetDestination(position);“ weist dann letztendlich die oben bestimmte Position dem Zombie als neues Ziel zu.