Objektorientierte Analyse des gewünschten Gegner-Verhaltens

Was wird zur Umsetzung des Verhaltens benötigt?

Alle C#/JavaScript Klassen welche für die verschiedenen Gegner erstellt werden, sollten eine gemeinsame abstrakte Oberklasse ***Enemy*** haben in der gemeinsames Verhalten / Funktionalität für alle Gegner abstrahiert werden kann. Die Funktionalitäten hier dürfen u.A. folgendes umfassen: Schaden erleiden, erkennen wann der Gegner stirbt und das entfernen aus dem Spiel, ein Interface für den Zustandsautomaten bereitstellen, ein einheitliches Interface für die Darstellung/Animation der Gegner, die Position des Gegners speichern, ~~sowie statische Methoden für den Rest der Anwendung um auf alle Gegner zugreifen zu können~~ *(nicht notwendig, UnityEngine macht das schon gut so)*.

Der Zustandsautomat für jeden Gegner ist notwendig um die verschiedenen Zustände in denen sich ein Gegner befinden kann zu ermöglichen. Für die Bewegung könnten das z.B. sein: Patrouillieren nach rechts/links, in Angriffsposition bewegen, Position halten. Ein weiterer Zustandsautomat für den Angriff, unabhängig von der Bewegung, kümmert sich um gegnerspezifische Angriffsmodi, z.B. um die Schussgeschwindigkeit (von möglicherweise verschiedenen und voneinander unabhängigen Waffen) zu gewährleisten. Zustände können z.B. sein: Angreifen, Nachladen, Ziel-Nicht-in-Reichweite, und Kombinations-Attacken (Schlag 1 → Schlag 2 → Schlag 3 → Schlag 1 → ...).

Bewegung

Einige Gegner bewegen sich, andere Bewegen sich gar nicht, und wiederum andere können klettern ~~oder Fliegen~~. Benötigt wird eine Abstraktion die das alles zulässt und ermöglicht. Dafür soll die Klasse *Enemy* zunächst zwei abstrakte Unterklassen haben: ***ImmovableEnemy*** für unbewegliche Gegner, und ***MovableEnemy*** für bewegende Gegner. *MovableEnemy* hat weitere abstrakte Unterklassen für verschiedenen Arten der Bewegung.

Die Gegner drücken ihren Wunsch nach Bewegung (Steering Behaviour) durch Kraft(Force)-Vektoren aus, welche durch ihre Bewegungs-Zustände ausgewählt werden. Die abstrakten Klassen für die Bewegungsart limitieren die Bewegung des Gegners (Locomotion) entsprechend der artspezifischen Regeln (z.B. kann ein Gegner nach oben gehen wollen, kann dies aufgrund seines Bewegungsmodells aber nichts, dann muss der Vektor parallel zur Ebene, auf welcher der Gegner steht, gedreht werden). *MovableEnemy* setzt den limitierten Wunsch dann in Bewegung um (Translation). ~~Die Klasse *Enemy* sorgt dann dafür dass der Gegner an den neuen Koordinaten angezeigt wird~~ (das tut die UnityEngine schon).

Geschosse

Die Geschosse (also nicht Nahkampf) sind eigene Gegenertypen. Eine abstrakte Klasse ***Projektile*** die Subklasse von *MovableEnemy* ist, abstrahiert die Kollisionsabfrage mit dem Spieler, fügt dem Spieler gegebenenfalls Schaden zu, und lässt das Geschoss sterben, damit es aus dem Spiel von der entsprechenden Routine der Superklasse *Enemy* entfernt werden kann.

Die konkreten Unterklassen von Projektile implementieren die geschossspezifischen Bewegungen, ~~wählen die zu verwendenden Sprites aus~~ (das passiert im Prefab nicht im Code), und legen den zu verursachenden Schaden fest. Die Geschossobjekte werden von den jeweiligen Gegnern erstellt wenn sie sich (mittels eigenem Zustandsautomaten) dazu entscheiden zu schießen, und handeln von dann an autonom.

Konkrete Gegnerklassen

Die konkreten (nicht-abstrakten) Gegner-Klassen, von der für jeden Gegner-Typ eine existiert, müssen die von den Oberklassen benötigten Methoden implementieren, sowie dessen initialen Instanzvariablen setzen (z.B.: Trefferpunkte, Spriteauswahl, Bewegungsgeschwindigkeit, Reichweiten, ~~Gewicht/Masse, Größe/Ausmaße~~, etc.).

Sie bestimmen welche Startzustände der Gegner hat, und dessen Zustände bestimmen die Zustandsübergänge. Die Zustände bestimmen zum Großteil das Verhalten des Gegners, also Entscheiden ob ein Zustandsübergang nötig ist.

Damit die Zustände Entscheidungen treffen können haben alle Gegner ein Nachrichten-System, über das ihnen Events mitgeteilt werden. Diese werden von den Gegnern an ihren aktuellen Zustand übergeben, der darauf dann reagieren kann (je nach Event-Typ).

Verfolgen des Spielers

Ein großes Problem bei diesem zustandsbasierten System mit Steering-Behaviours liegt in der Bewegung, da wir es in unserem Spiel mit mehreren Plattformen zu tun haben. Fliegende Gegner können mit Obstacle/Wall-Avoidance den Plattformen zwar ausweichen, aber was tut ein Gegner wenn sich der Spieler unter ihm befindet? Soll er sich nach Links oder Rechts bewegen um zum Spieler zu gelangen? Der einfache Weg um dieses Problem zu lösen ist den Gegner den Spieler nur dann verfolgen zu lassen, wenn dieser auch in Line-of-Sight (LOS) ist, und ansonsten wieder in den Patrouillieren-Zustand zu verfallen. Das ist aber nicht so schön, weil der Gegner den Spieler vergisst und nicht sehr intelligent wirkt.

Schönere (intelligenter wirkende) Lösungen sind komplexer und schwieriger zu implementieren. Dies sind optionale Features für das Projekt, der einfache LOS-Test wird mindestens implementiert.

Alternative 1: ein sensorisches System (letzte sichtbare Position) mit zeit-beschränktem Gedächtnis das die zuletzt observierte und erreichbare Position des Spielers mit Zeitstempel hinterlegt, und diesem statt der aktuellen Position des Gegners hinterherjagt, sofern nicht eine maximale Zeitdauer vergangen ist.

~~Alternative 2: ein Navigationsgraph. Das Problem hierbei ist vor allem das dieser für jede Map neu erstellt werden muss (z.B. automatisch mit einem Flood-Fill-Algorithmus) und dass sich die Map zur Laufzeit verändern könnte (Türen, Falltüren, ...). Damit der zu durchsuchende Graph bei zu großen Maps nicht zu groß ist, könnte man die Suche auf etwa 3-5 Bildschirmbreiten begrenzen. Um auf sich verändernde Hindernisse oder Türen reagieren zu können, wird bei der Pfadverfolgung die in etwa benötigte Zeit zum erreichen des nächsten Knotens großzügig geschätzt. Beim Überschreiten der geschätzten Zeit wird der Weg neu geplant, und liefert, da sich der Graph verändert hat, einen anderen Weg.~~

Es wurde Alternative 1 umgesetzt.

Übersicht / Zusammenfassung

<table>
<tr><td colspan="3">Enemy:
Schadenssystem
Position
Darstellung / Animation
Sterben
Bewegungs-Zustandsautomat
Angriffs-Zustandsautomat</td></tr>
<tr><td colspan="2">MovableEnemy < Enemy:
Steering-Behaviour-System
Translation</td><td rowspan="2">ImmovableEnemy < Enemy:
unbeweglich</td></tr>
<tr><td>Projektile < MovableEnemy:
Kollisionsabfrage
Schaden zufügen
Locomotion</td><td>Bewegungsarten < MovableEnemy:
Links Rechts Klettern ~~Fliegen~~
Obstacle-/Wall-Avoidance
Locomotion
Bewegungs-Zustände: SteeringBehaviours</td></tr>
<tr><td>Konkrete Geschosse < Projektile:
Spriteauswahl
Schadenswert
geschossspezifische Bewegung</td><td colspan="2">Konkrete Gegner < ImmovableEnemy oder Bewegungsart:
setzen generspezifische Parameter der Oberklassen
setzen Startzustände
Nachrichten-/Event-System
Vefolgung: LOS ~~oder~~/und Gedächtnis ~~oder Navigationsgraph~~</td></tr>
</table>