ZOMBIE RPG

...ein rundenbasiertes Java-RPG

https://github.com/aneeven/zombierpg

Autoren:
Andreas Neeven
Florian Petruschke

1. Vorüberlegungen

Wir haben die spannende Aufgabe bekommen, ein Spiel in Java im Rahmen eines Pair-Programming-Projektes zu programmieren. Wir haben uns dazu entschlossen, ein rundenbasiertes RPG zu entwerfen.

- Ein Charakter kann gegen zufällig generierte Gegner kämpfen
- Charaktere haben Attribute
- Attribute werden durch Kämpfe beeinflusst
- Spiel soll in einfachem GUI angezeigt werden
- verschiedene Sounds sollen genutzt werden, um eine - dem Spiel angemessene - Atmosphäre zu schaffen

Nachdem wir uns auf ein Spielprinzip geeinigt hatten, mussten wir uns Gedanken über das Thema – das Setting unsers Spiels machen.

Wir haben uns überlegt, den Spieler in die Rolle eines Zombies zu versetzen, welcher sich gegen Menschen behaupten muss. Die Faszination für die Thematik "Zombies" ist in den letzten Jahren durch diverse Filme und Serien in der Öffentlichkeit gestiegen und wir denken, dass es eine neue und interessante Erfahrung für den Spieler ist, einmal auf der "bösen Seite" zu stehen. Der Spieler findet sich also in der Rolle des Zombies wieder.

2. Spielprinzip

Der Spieler – als Zombie - hat die Aufgabe, Menschen aufzuspüren und zu bekämpfen. Durch einen errungenen Sieg, erhält er einen Zuwachs seiner Attributpunkte. Verliert er den Kampf, steigen seine negativ-Attribute.

Attributpunkte des Zombies

Der Spieler muss seinen Charakter bei Spielbeginn benennen. Dann erhält er

bestimmte Attributpunkte als Startwerte. So liegen „Zerfall" und „Hirnhunger" (negativ-Attributes – je höher desto schlechter) zu Beginn bei 50%. Der Stärkewert unseres Zombies liegt bei 5. Kämpfe und Fluchtversuche haben direkten Einfluss auf die Attribute des Zombies.

Attributpunkte der Gegner

Der Gegner wird zufällig ermittelt. Wir haben vier Mensch-Klassen kreiert: Dummköpfe, Handwerker, Akademiker und Soldaten. Menschen sind - je nach Schwierigkeitsstufe (Dummköpfe sind die einfachsten, Soldaten die schwierigsten Gegner für unseren Zombie) - mit einer bestimmten Anzahl von Leben ausgestattet. Auch Menschen haben Stärkepunkte und „Hirnpunkte". Über die Stärkepunkte wird der Kampf entschieden, die Hirnpunkte kommen dem Zombie bei erfolgreichem Kampfausgang zugute. Jeder Kampf kann mehrere Runden haben – je nachdem, wie hoch Lebensanzahl und Zerfall bei den Kontrahenten sind.

Menschen können außerdem einen Gegenstand bei sich tragen. Ob ein Mensch einen Gegenstand bei sich trägt, wird per Zufall entschieden. Die Wahrscheinlichkeit ist bei dem Dummkopf am geringsten und bei Handwerkern und Soldaten am höchsten. Die Gegenstände haben verschiedene Stärkewerte, welche den Stärkepunkten des Menschen angerechnet werden.

Aktionen des Zombies

Als Zombie kann gebissen, getreten, angesprungen, entwaffnet und geflüchtet werden. Jeder Angriff hat eine bestimmte Erfolgswahrscheinlichkeit und eine bestimmte Auswirkung auf den Kampf. So wird beispielsweise durch einen erfolgreichen Biss der Gehirn-Hunger um $\frac{1}{4}$ der Gehirnstärke des Menschen verringert. Außerdem verliert der Mensch Lebenspunkte. Allerdings ist die Erfolgswahrscheinlichkeit geringer als die eines Bisses. Dieser fügt lediglich geringen Schaden (Lebensabzug) zu, kann aber mit 90%-iger Wahrscheinlichkeit durchgeführt werden.

Aktionen des Menschen

Durch Zufallsauswahl wird eine Aktion des Menschen ermittelt. Diese können entweder Schlagen oder Treten sein. Hat der Mensch zusätzlich einen Gegenstand bei sich, kann er diesen einsetzen, um dem Zombie Schaden zuzufügen.

Ende des Kampfes

Ein Kampf endet entweder mit dem Sieg des Zombies, einer erfolgreichen Flucht des Zombies oder mit der Niederlage des Zombies.

Ende des Spiels

Das Spiel ist zu ende, wenn der Zombie einen Zerfall von 100% erleidet.

3. Programm-Ablaufpläne

Wir haben versucht, unser Programm möglichst einfach anhand von PAP darzustellen. Einige Details (z.B. die Berechnung der Erfolgswahrscheinlichkeiten) konnten wir leider aus Zeitmangel nicht mehr darstellen. Dennoch sollen die Pläne veranschaulichen, wie das Spiel prinzipiell funktioniert.

4. UML-Klassendiagramm

Auch bei der Erstellung des Klassendiagramms konnten wir leider zeitbedingt nicht alle Details aufzeigen.

5. Verwendete Software/Tools

Um das Spiel zu programmieren haben wir die IDE „Eclipse" verwendet. Diese läuft auf allen gängigen Betriebssystemen und wurde für die Java-Programmierung entwickelt. Wir haben Java JRE in der Version 1.8.0_77 verwendet. Da es sich bei dem Projekt um ein Pair-Programming-Projekt handelt, haben wir uns entschieden, mit dem VCS „GitHub" zu arbeiten. So konnten wir unabhängig voneinander zu gleichen Zeitpunkten an unterschiedlichen Programmteilen arbeiten. Die Änderungen konnten dann einfach zusammengefügt werden. Um Grafiken anzufertigen nutzten wir das Online-Tool „draw.io" (http://draw.io). Dieses erlaubt eine sehr nutzerfreundliche Zeichnung von diversen Grafiken und Plänen. Für das Erstellen des Zombie-Bilds nutzen wir Bleistift und Papier. Die Zeichnung wurde dann digitalisert und mit dem Graikprogramm „Gimp" nachbearbeitet.

6. Feedback Pair-Programming

Wir konnten bei der Programmierung des Spiels einen tieferen Einblick in die Entwicklung von Java-Applikationen gewinnen. Wir haben uns im Laufe dieses Projekts mit Klassen und ihren Objekten, mit Konstruktoren, statischen und globalen Variablen, Kompositionen und Vererbungen auseinander gesetzt.

Große Schwierigkeiten hatten wir mit dem Erstellen eines GUIs. Uns erschien die

Erstellung, Konfiguration und Verwendung der hierzu bestehenden Java-Methoden als sehr komplex und sperrig. Die größte Problematik bestand darin, lediglich den Inhalt eines Fensters zu verändern ohne den Zugriff auf alle notwendigen Attribute und Objekte zu verlieren.

6.1 Feedback A. Neeven

Ein UI in Java umzusetzen gestaltete sich deutlich schwieriger als zunächst angenommen, ließ sich aber durch Recherche verschiedener Methoden dennoch gut lösen. Das größte „Learning" war definitiv, dass wir uns durch genauere vorherige Planung einiges an Arbeit hätten sparen können. Selbst bei einer kleinen Applikation merkt man wie wichtig es ist im voraus Architekturentscheidungen zu treffen. Alles in allem bot dieses Projekt eine spaßige Möglichkeit mehr über die Umsetzung und Einbindung von Swing-Komponenten, Bildern und Ton in Java zu lernen.

6.2 Feedback F. Petruschke

Ich habe im Rahmen dieses Projektes gelernt, unter hohem Zeitdruck kreativ zu arbeiten und nebenbei weitere Grundlagen der Java-Programmierung zu vertiefen oder zu erlernen. Meine Haupt-Erkentnisse dieses Projekts liegen in der Nutzung von finalen und statischen Methoden/Attributen, der GUI-Programmierung, der Nutzung von Audiodateien und dem Entwickeln einer komplexen Programmmechanic.
In der GUI-Programmierung und dem Fach-Vokabular liegen allerdings meine größten Schwächen. Ferner war meine Auseinandersetzung mit Threads u.a. zur gleichzeitigen Wiedergabe verschiedener Sounds unzureichend. Dies sind Punkte, die ich mir noch einmal vertieft anschauen müsste.
Mir ist bei diesem Projekt bewusst geworden, wie wichtig eine strukturierte Planung des anzugehenden Programms ist. Ich habe mich etwas schwer damit getan, einen Programmablaufplan und ein Klassendiagramm zu erstellen. Hinterher weiß ich nun,

dass es mir viel Arbeit hätte ersparen können, gleich zu Beginn vollständige Pläne zu entwerfen und nicht während des Programmierens immer neue Ideen und Programm-Schnipsel zu implementieren ohne diese zuvor in den „Bauplan" zu integrieren und auf Logik und Vollständikeit zu überprüfen.

7. Ausblick

Das Spiel ließe sich natürlich optimieren. Man sollte z.B. versuchen, eine klare Trennung von Entitäten, Controllern und Ansichten umzusetzen und eine dementsprechende Datei-Struktur (MVC) anzulegen. Außerdem ist die Mechanik zwischen Gehirn-Hunger und Zerfall des Zombies noch nicht ausgereift. In der Zukunft könnte man mit verschiedenen Grafiken verschiedene Schauplätze veranschaulichen und ja nach Schauplatz die Wahrscheinlichkeit des Auftretens bestimmter Mensch-Klassen verknüpfen. Es könnten zusätzliche Mensch- und Gegenstand-Klassen entworfen werden. Außerdem könnten eigene Sounds und Musikstücke geschrieben und importiert werden.