Programmierbeleg

# Rollenspiel: Tactix

Verfasser: Andy Schönfeld

IF-16wS-B

43469

Hochschule Mittweida

Auftraggeber: Prof. Dirk Pawlaszczyk

# 0. Inhaltsverzeichnis

1. Aufgabenstellung
2. Problemanalyse und Lösungskonzeption
3. Programmentwurf
4. Details der Implementierung
5. Fazit
6. Bedienungsanleitung
7. Quellen und Hilfsmittel
8. Selbstständigkeitserklärung

# 1. Aufgabenstellung

Die vorliegende Arbeit befasst sich mit der Erstellung eines Rollenspiels. Die Aufgabenstellung ist folgende:

Erstelle eine lauffähige Demoversion nach dem Vorbild von Final Fantasy Tactics Advance.

Es soll ein Spielfeld, mit Einheiten die durch den Spieler gesteuert werden und Einheiten die durch den Computer gesteuert werden, geben.  
Die Einheiten müssen Health Points aufweisen und brauchen ausgewogene Stärken und Schwächen.  
Außerdem muss das Spiel gewonnen oder verloren werden können.

# 2. Problemanalyse und Lösungskonzeption

Ein Spiel besteht im Allgemeinen aus mehreren Zuständen (Splash, Menü, das Spiel selbst, …). Diese Zustände werden im weiteren *States* genannt. Es braucht also eine Klasse, welche diese *States* verwaltet und einen einfachen Wechsel zwischen den *States* ermöglicht, quasi das Spiel an und für sich.

Das Spielfeld ist eigentlich ein 3-dimensionaler Raum. Allerdings ließe sich dieser Raum nur mit Virtual Reality wirklich erfassen, also muss dieser auf 2D projiziert werden. Dabei fiel die Wahl auf eine durch isometrische Projektion inspirierte Darstellung.  
Um eine Position im Raum (bekannt als Rasterkoordinaten, Rastergröße: 24px) also auf dem Bildschirm darzustellen sind folgende Berechnungen notwendig:

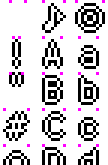
Da keine Libraries verwendet wurden muss eine Möglichkeit bereitgestellt werden um Fonts zu rendern. Dabei wurde auf eine einfache Methode zurückgegriffen: Die Buchstaben des Fonts sind in einem Bild (= Font Sprite Sheet) und anhand des Charcodes werden x- und y-Koordinate des Buchstaben im Bild bestimmt, der Abschnitt kopiert und auf dem Bildschirm dargestellt. Dabei entsteht das Problem, dass kein monospace-Font gerendert wird sondern ein proportional. Die Buchstaben haben also unterschiedliche Breiten. Dieses Problem kann gelöst werden, indem im Sprite Sheet links und rechts der Buchstaben markiert wird, wo der Buchstabe losgeht und wo er aufhört. Beim Start des Spiels werden diese Abstände abgemessen, die magentafarbenen Punkte auf *argb(0, 0, 0, 0)* gesetzt, also unsichtbar gemacht.  
Wenn nun ein Text in diesem Font gerendert wird haben die Buchstaben also nicht immer denselben Abstand zum vorherigen, sondern nur die eigene Breite plus ein definiertes *letter spacing* (= Abstand der zwischen allen Buchstaben gleich eingefügt wird)

Abbildung 1: Ausschnitt aus einem Font Sprite Sheet (300% skaliert)

Die Einheiten im Spiel müssen sich von einem Feld zum nächsten Bewegen und dabei eventuelle Hindernisse umlaufen. Es muss also ein Wegfindungsalgorithmus implementiert werden. Die Wahl fiel dabei auf A\*.  
A\* errechnet einen möglichen Weg, indem die Luftlinie zwischen einem Feld auf dem Weg und dem Ziel und die Gesamtlänge des eigentlichen Weges so kurz wie möglich wird. So werden immer nur Felder in diesem Weg eingeschlossen, die den kürzesten Weg bilden.

Ein letztes Problem in einem Rollenspiel ist, dass gegnerische Einheiten unter Umständen vom Computer kontrolliert werden. Also muss eine Art künstliche Intelligenz entwickelt werden, die möglichst sinnvolle, zielführende Entscheidungen trifft, dabei aber nicht zu perfekt vorgeht, so dass dem Spieler eine Chance gelassen wird. Hierbei gibt es viele Optionen, Wege und Methoden zur Realisierung, die auf den Anwendungsfall und auf die Aggressivität/Defensivität der gewünschten KI zugeschnitten werden müssen.  
In Tactix hat die KI 3 Optionen (Laufen, Angreifen, Warten) und 2 Züge, wobei der erste Zug entscheidet, welche Optionen im ersten Zug noch übrigbleiben.

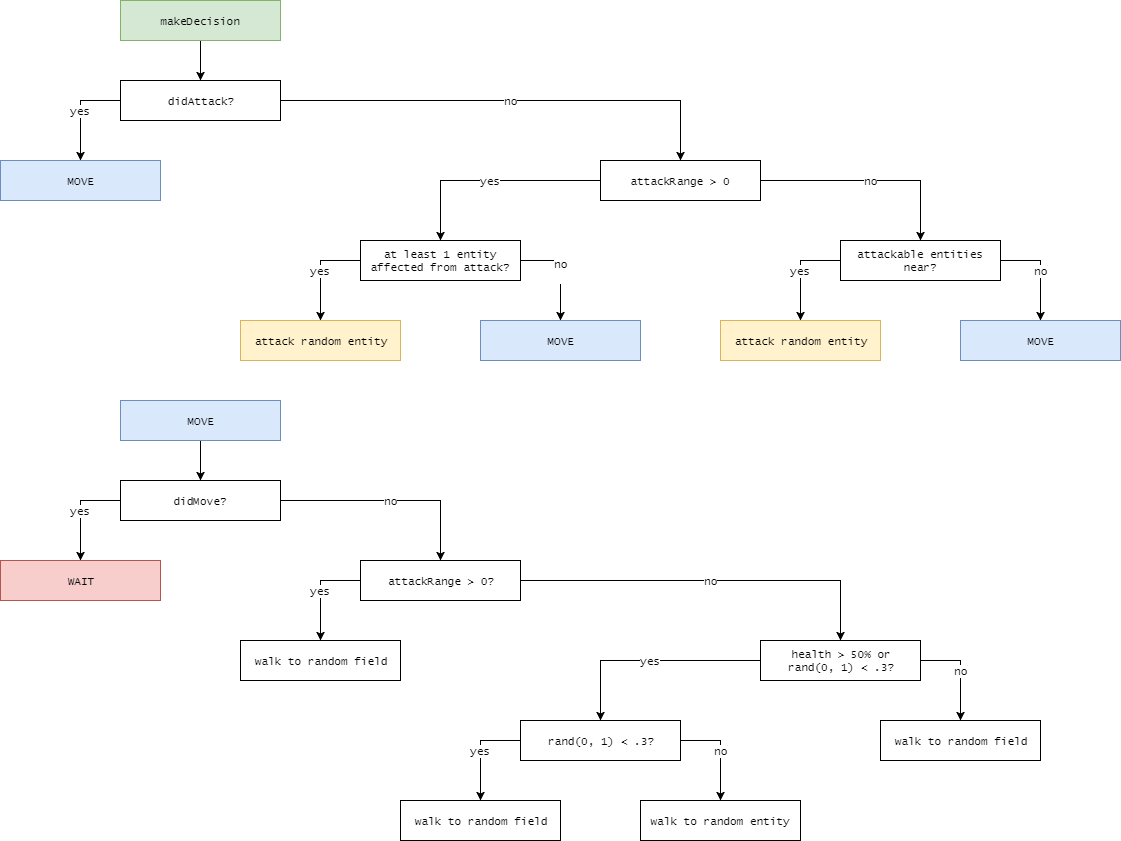
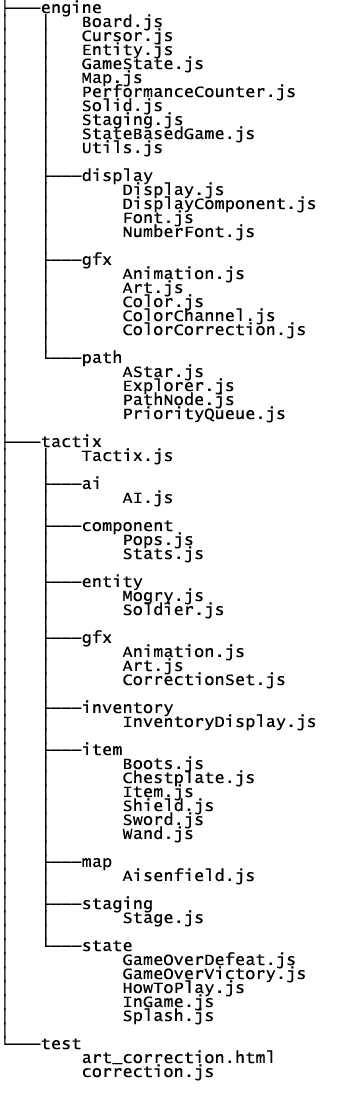


Abbildung 2: Flowchart der KI in Tactix

# 3. Programmentwurf



Das Programm ist unterteilt in Engine und Tactix (das Spiel selbst). Die Engine ist, wie der Name schon sagt, der „Motor“ des Spiels. Sie enthält das Management der States, Grafiken und Updates sowie Rendering.

Tactix enthält spielspezifischen Code, also spezielle Typen an Entities, Items, Maps und States.

# 4. Details zur Implementierung

Das Spiel ist nicht nach dem neusten JavaScript Standard (ES2017), sondern nach ES5 umgesetzt.

So wurde also nicht *class* aus dem neusten Standard verwendet, sondern es werden Funktionsobjekte quasi als Factory genutzt.

Heutzutage nicht wirklich eine Problematik, aber bei Konsolenspielen bspw. damals durchaus kritisch war der Speicherplatz. So wurden beim Gameboy Advance zum Beispiel mit mehreren Paletten gleichzeitig gespeichert, als eine Datei. Um Aufwand zu sparen wurde hier zu ähnlichen Mitteln gegriffen. Viele im Spiel sichtbaren Elemente wurden durch eine CMYK-Farbkorrektur verändert. Die Grafiken der Einheiten bspw. existieren nur in blau bzw. grün. Durch eine selektive Farbkorrektur, die beim Laden des Spiels angewandt wird werden sie zu roten (feindlichen) Einheiten. So wurde effektiv benötigter Speicherplatz gespart.

# 5. Fazit

Ich bin mit dem Resultat sehr zufrieden, auch wenn der Fokus sehr oft zu sehr auf der Engine lag, statt auf dem eigentlichen Spiel.

Viele Methoden in Board und auch Entity sind aus der Not heraus entstanden und einige davon sollten so refractored werden, dass Methoden, die nicht Instanz spezifisch sind, statisch verfügbar sind. Auch ist die *tick* des *InGame-States* ihrer Funktion geschuldet sehr aufgeblasen und hätte in Einzelteile zerlegt werden können.

Für ein vollwertiges Rollenspiel fehlen einige Features. Folgendes wäre durch Erweiterungen noch denkbar:

* Maps mit Höhenunterschieden, durch eine höhere Position anderen Einheiten gegenüber könnte so ein taktischer Vorteil entstehen
* Mehr Items, die einen individuellen Buff/Nerf der Charaktere ermöglichen
* Selbstbestimmbares Inventar der Charaktere
* Mehr Klassen für Einheiten, so dass ein individuelleres Team zusammengestellt werden kann

Außerdem sollten Angriff/Verteidigung, so wie die Boosts der Items *gebalanced* sein, so dass ein fairer Kampf zwischen den Teams möglich ist.

# 6. Bedienungsanleitung

Die Einheiten nehmen Befehle als Aktionen entgegen. Es gibt 3 Aktionen: Laufen, Angreifen, Warten.

In jeder Runde werden höchstens 2 Aktionen und anschließend Warten durchgeführt. Warten ist natürlich auch als eine der beiden vorherigen Aktionen möglich.

Wenn die eigenen Einheiten (blau) am Zug sind wir die momentan aktive Einheit mit einem grünen Feld und einem Zeiger über dem Kopf markiert.

Mit den Tasten Q, W, E wechselt man nun den Modus der Einheit. Je nach Modus werden durch die Entität erreichbare Felder markiert und von Angriffen eventuell betroffene Einheiten blinkend dargestellt.

* Q: **Bewegen**. Alle blauen Felder können von der Einheit erlaufen werden
* W: **Warten**. Dieser Modus wird am Ende der beiden Aktionen automatisch gewählt. Sie beendet nach Ausführung den Zug der Einheit und bestimmt, mit welcher Blickrichtung die Entität stehen bleiben soll
* E: **Angreifen**. Alle gelben Felder können von der Einheit attackiert werden. Dabei gilt es zu beachten, dass Einheiten mit magischen Angriffen bspw. mehrere Felder gleichzeitig attackieren. Von Angriffen betroffene Einheiten werden blinkend dargestellt

Bei abgeschlossenem Zug wird zur Einheit mit dem höchsten Tempo weiter gewechselt. Es kann also sein, dass nach der eigenen Einheit mehrmals die gegnerischen Einheiten am Zug sind.

Das Tempo und das (Demo-)Inventar öffnet sich mit einem Druck auf die F-Taste. Hier sieht man auch die Klasse und Statistiken der Einheit.

Es empfiehlt sich, Soldaten mit Magiern und Magier mit Soldaten anzugreifen, da sich dabei die Angriffs- und Verteidigungswerte auskontern.

Abschließend gilt es noch zu erwähnen, dass ein Angriff von vorn eine große Chance hat daneben zu gehen. Angriffe von der Seite oder noch besser von Hinten weisen eine höhere Trefferquote und einen leicht erhöhten Schadensdurchsatz auf.

# 7. Quellen und Hilfsmittel

* Adobe Photoshop
* Aseprite
* Eclipse
* Atom
* Perl Interpreter
* JDK
* https://www.spriters-resource.com/
* https://opengameart.org/

# 8. Selbstständikeitserklärung

Ich erkläre, dass ich die vorliegende Arbeit selbständig und nur unter Verwendung der angegebenen Literatur und Hilfsmittel angefertigt habe.