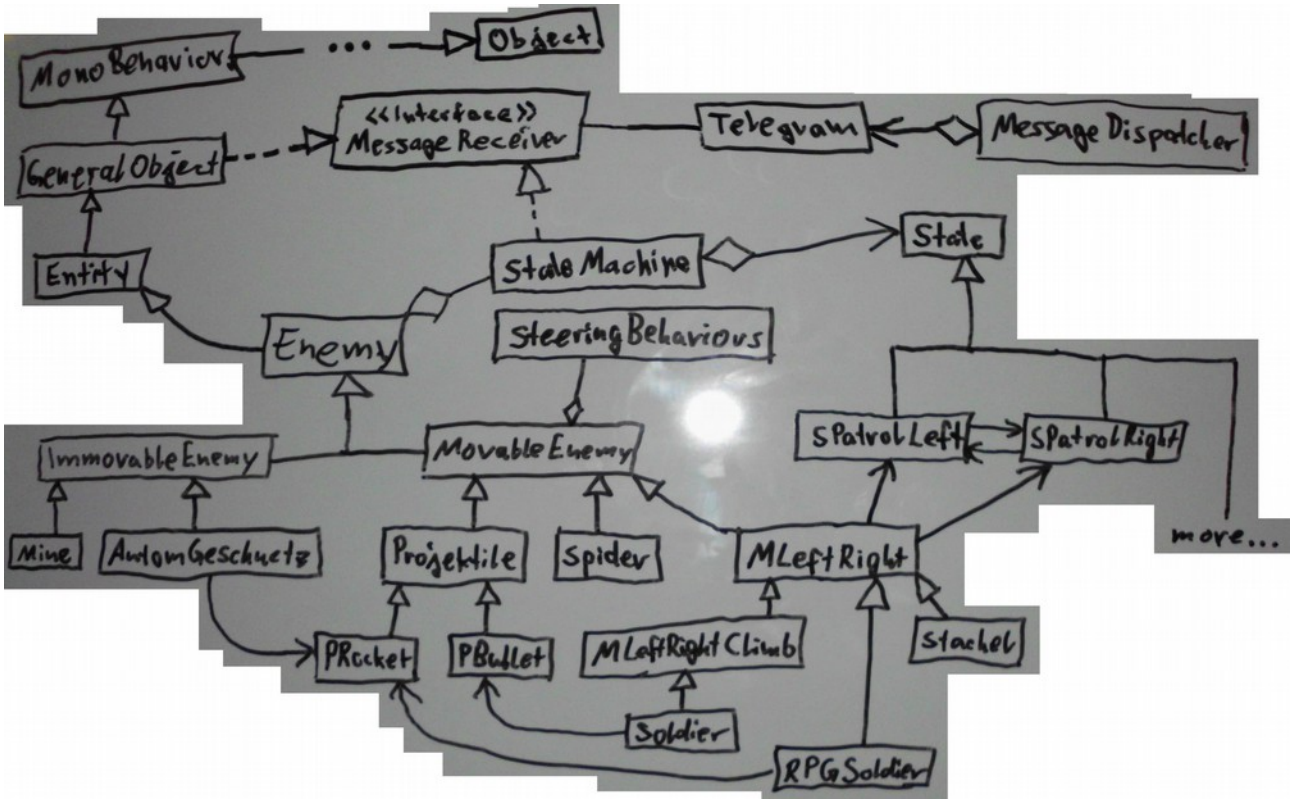
Objektorientiertes Design der Gegner in „Dark Side Of Morra“

Inhaltsverzeichnis

1. Einführung

Aufgrund der Menge der Klassen zunächst ein Überblick aller Klassen und deren Beziehungen ohne deren Instanzvariablen und Methoden. Auch alle nachfolgende Klassendiagramme und Zustandsautomaten sind keine vollständigen und korrekten Abbildungen des Codes, sondern sollen die Idee der jeweiligen Komponente einfach vermitteln ohne ins Detail gehen zu müssen. So sind einige Methoden ohne Paramete z.B. als C# Properties implementiert, was die Angabe von Klammern einspart, oder angegbene Instanzvariablen sind Properties, obwohl dahinter vielleicht gar keine Instanzvariable steckt sondern eine komplexe Berechnung.



Es gibt noch mehr als die gezeigten Klassen, so existieren für die meisten Gegner noch Klassen für die Zustände, die im folgenden noch beschrieben werden.

2. Nachrichtensystem

Das Nachrichtensystem dient dazu Nachrichten zwischen verschiedenen Objekten austauschen zu können. Es bietet im Gegensatz zu SendMessage von Unity die Möglichkeit Nachrichten Zeitverzögert auszuliefern.
Die C# Implementation basiert auf der in „Mat Buckland - Programming Game AI by Example“ angegebenen C++-Version.

2.1 MessageReceiver

Das MessageReceiver Interface definiert die Methode HandleMessage, die Implementiert werden muss um in der Lage zu sein Nachrichten empfangen zu können. Diese Methode bekommt ein Telegram und gibt als Wahrheitswert zurück ob die Nachricht angenommen wurde.

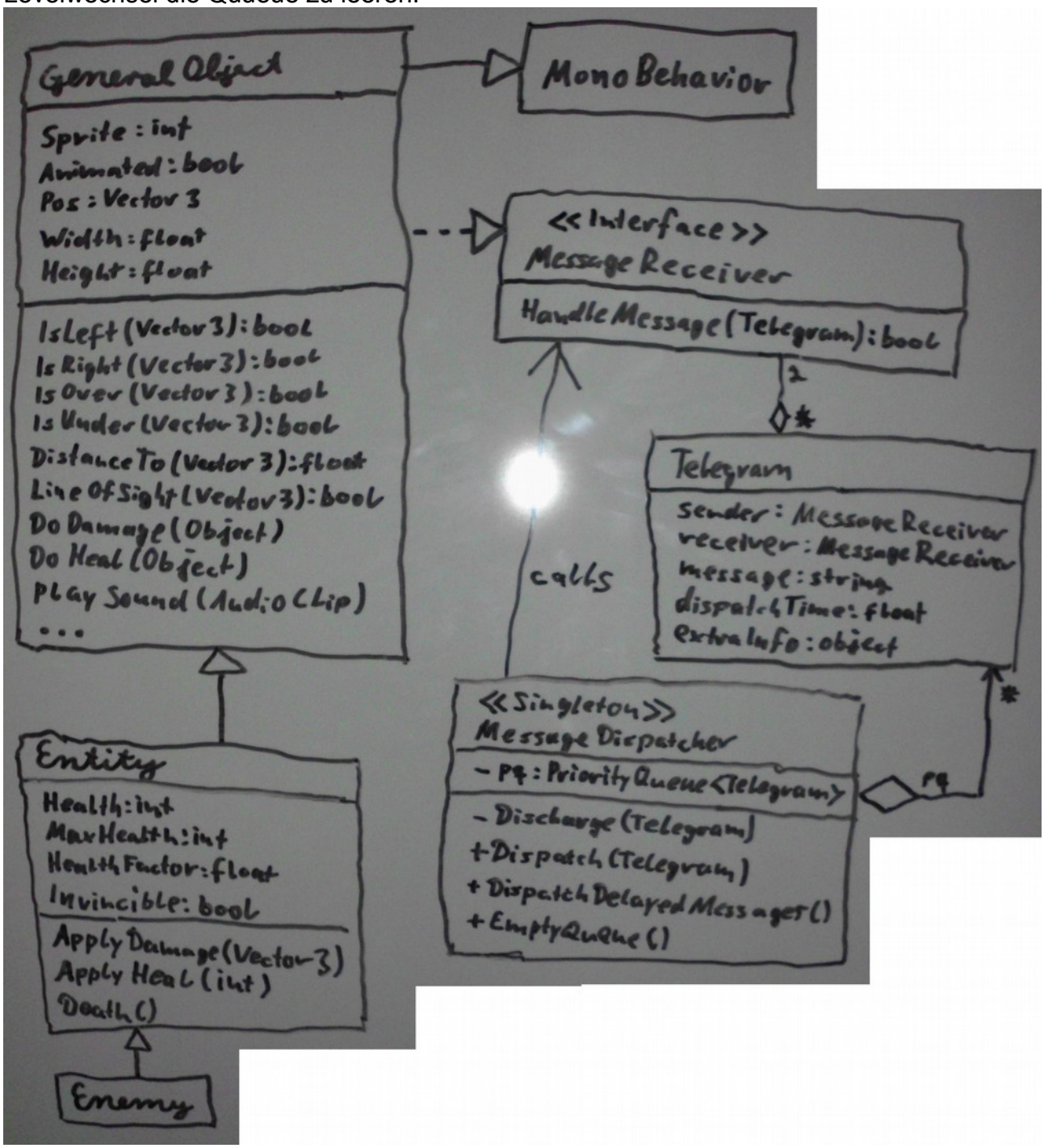
2.2 Telegram

Ein Telegram ist eine Klasse um die Verschiedenen Eigenschaften einer Nachricht in einem Objekt zu sammeln, sowie sie miteinander anhand des Auslieferungszeitpunktes für die Priority Queue sortieren zu können.
Folgende Eigenschaften hat ein Telegram: Sender, Empfänger, Nachricht, Auslieferungszeitpunkt, beliebige zusätzliche Daten (object).

2.3 MessageDispatcher

Der MessageDispatcher erhält von Objekten die anderen Objekten (oder sich selbst zeitverzögert) eine Nachricht schicken möchten, ein Telegram über die Methode Dispatch.

Wenn der Auslieferungszeitpunkt bereits überschritten wurde wird das Telegram sofort ausgeliefert (Discharge) und ansonsten in eine PriorityQueue (Eigenimplementation) einsortiert um später ausgeliefert zu werden. Dazu bietet der Dispatcher die Methode DisplayDelayedMessages, die regelmäßig aufgerufen werden muss. Da dies ein Singleton mit statischer Referenz ist, wird es beim Laden eines anderen Levels nicht neu erstellt, sondern bleibt erhalten, inklusive der Inhalte der PriorityQueue, dessen Referenzen dann ungültig sind. Deshalb bietet die Klasse die Methode EmptyQueue um manuell beim Levelwechsel die Quueue zu leeren.



3. Abstraktion von Nicht-Gegner-Funktionalität

Da bei der Programmierung von Gegnern auch Code anfällt der auch für nicht Gegner relevant ist, befinden sich über der Enemy-Klasse noch zwei weitere Klassen: GeneralObjekt und Entity.

3.1 GeneralObject

Die Abstrakte-Klasse GeneralObject würde GameObject heißen, wenn dies nicht bereits der vergebene Name einer Unity-Klasse wäre, sie soll allgemein alle möglichen Spielobjekte repräsentieren können, belebt oder unbelebt. Als Klasse die von MonoDevelop erbt ist sie eine Skript-Komponente die auf GameObjects gezogen werden kann.

GeneralObject dient vor allem dazu das Programmieren einfacher zu machen in dem es jede Menge Methoden definiert die häufig benötigte Funktionalitäten von Unity einfacher zugänglich macht. Als Beispiel sei hier Pos erwähnt - die Position des GameObjects in der Spielwelt - was normalerweise mittels „this.collider.bounds.center“ Aufgerufen werden muss kann nun mit „this.Pos“ aufgerufen werden.

Über die Properties Sprite und Animated kann der SpriteController gesteuert werden, der von GeneralObject automatisch dem GameObject hinzugefügt wird sollte er benötigt werden (kein manuelles zuweisen im Inspektor nötig).

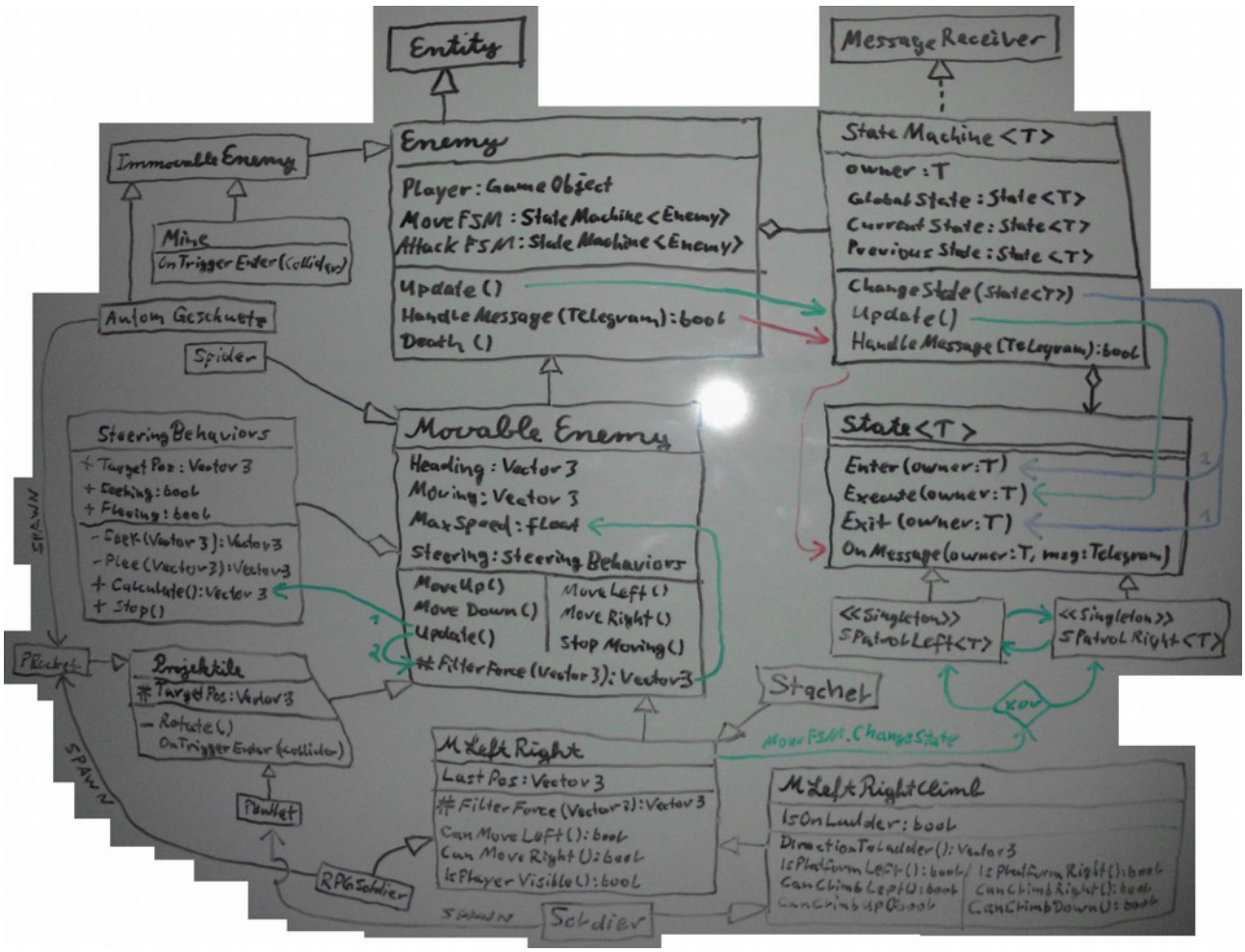
Die Klasse implementiert das Interface für das Nachrichtensystem um Nachrichten empfangen zu können, auch wenn in dieser Klasse damit noch nichts getan wird. Die Methoden DoDamage und DoHeal verschicken zwar Nachrichten an andere Objekte, aber nicht über das Nachrichtensystem, sondern über die Unity SendMessage Methode um mit dem Code anderer Projektmitglieder, die nicht das Interface des Nachrichtensystems implementieren wollen, kompatibel zu sein. Es wurde sich auf die Nachrichtennamen ApplyDamage und ApplyHeal geeinigt.

3.2 Entity

Die Abstrakte Klasse Entity spezialisiert die Abstraktion von GeneralObject auf Entitäten, die Trefferpunkte haben und dadurch in der Lage sind zu sterben. Dies trifft sowohl auf die meisten Gegner als auch auf den Spielercharakter zu – aufgrund von getrennten Zuständigkeitsbereichen bleibt der Spieler außerhalb dieser Klassen-Hierarchie.

Um die Trefferpunkte zu speichern sind die Instanzvariablen Health und MaxHealth notwendig. Der HealthFactor ist der zwischengespeicherte Wert der Division von Beidem (Health / MaxHealth). Für vorrübergehend unsterbliche Entitäten existiert die Eigenschaft Invincible.

Die Methoden ApplyDamage und ApplyHeal die durch DoDamage und DoHeal von anderen Objekten ausgelöst werden können verändern die Instanzvariablen. Sollten die Trefferpunkte auf 0 fallen stirbt die Entität. Dis Signatur von ApplyDamage wirkt zunächst fehlerhaft, es handelt sich aber um die normalisierte Richtung aus welcher der Schaden kommt, multipliziert mit dem Schadenswert. Dadurch ist es theoretisch Möglich Entitäten die Schaden erleiden in die entsprechende Richtung zurückzustoßen.



4. Gegner

4.1 Enemy

Die gemeinsame Oberklasse für alle Gegner ist Enemy, sie legt für jeden Gegner zwei Zustandsautomaten an. Einen für den Angriff und einen für die Bewegung. Beide werden in der Update-Methode aufgerufen, welche wiederum die Execute-Methode des aktuellen Zustandes aufrufen. Bei einer eingehenden Nachricht bekommen beide Automaten die Nachricht und haben die Möglichkeit darauf zu reagieren. Die Player-Referenz damit die Gegner auf ihn reagieren und mit ihm interagieren können wird hier gesetzt. Stirbt ein Gegner lässt er vielleicht ein Health Globe fallen (Regenerieren Trefferpunkte beim Spieler), das ist zufallsbasiert, und die Wahrscheinlichkeiten können sich von Gegner zu Gegner unterscheiden.

4.2 State Machine und State

Ein Zustandsautomat verwaltet die Referenz auf den aktuellen Zustand, merkt sich den vorigen Zustand, und bietet einen Globalen Zustand. Die Update-Methode ruft zuerst die Update Methode des Globalen Zustandes auf, und anschließend die des aktuellen Zustandes. Andersrum verhält es sich bei HandleMessage von MessageHandler Interface: Hier wird zuerst die Nachricht dem aktuellem Zustand zugestellt, und nur wenn dieser die Nachricht nicht annimmt geht sie an den Globalen Zustand.
Wird der aktuelle Zustand gewechselt wird erst die Exit-Methode des aktuellen, und anschließend die Enter-Methode des neuen Zustandes aufgerufen.
Zustände vom Typ State sind für gewöhnlich Singletons, weshalb sie die Referenz auf den Gegner dem sie gehören als Parameter bei Methodenaufrufen bekommen. Die vier Methoden sind:

Enter	wird aufgerufen wenn der Zustand betreten wird.
Execute	bei jedem Update wird Execute des aktuellen Zustandes ausgeführt.
Exit	wird ausgeführt bevor der aktuelle Zustand gewechselt wird
OnMessage	ein Telegram vom Nachrichtensystem ist eingetroffen, und kann vom Zustand ausgewertet werden

Die C# Implementation des Zustandsautomaten basiert auf der in „Mat Buckland - Programming Game AI by Example“ angegebenen C++-Version.

4.3 Steering Behaviors

Steering Behaviors drücken den Wunsch nach Bewegung aus. Es besteht aus verschiedenen Komonenten die ein oder ausgeschaltet werden können und einen Kraft-Vektor berechnen, der dann in Bewegung umgesetzt werden soll. Auch wenn vier Steering Behaviors implementiert sind, werden derzeit nur die beiden Steering Behaviors Seek (sich einer Position annähern, sie anstreben) und Flee (vor einer Positiion fliehen, sie vermeiden) verwendet.
Die C# Implementation der Steering Behaviors basiert auf der in „Mat Buckland - Programming Game AI by Example“ angegebenen C++-Version.

4.4 (Im)movableEnemy

Die Enemy-Klasse hat zwei Subklassen, zum einem die nichts tuende ImmovableEnemy-Klasse, der die Mine und das Automatische Geschütz, als sich nicht bewegende Gegner, angehören, und zum anderem die MovableEnemy-Klasse, welche als Oberklasse aller beweglichen Gegner gilt.

Die MovableEnemy Klasse verwendet SteeringBehaviors in der Update Methode zur Bewegung, und Filtert mittels FilterForce die Bewegung auf die Maximale Geschwindigkeit. Unterklassen können FilterForce überschreiben um andere Anforderungen / Limitierungen durchzusetzen.

Heading ist die Blickrichtung des Gegners und Moving die davon unabhängige Bewegungsrichtung. Beides sind eine vereinfachte Sicht auf die eigentliche Bewegung, um beides für die Texturauswahl auswerten zu können.

MoveUp, MoveDown, MoveLeft, MoveRight und StopMoving sind einfache Methoden die für Basisbewegungen verwendet werden können.

4.5 Projektile

Projektile sind Geschosse die wenn sie etwas berühren sterben und Schaden verursachen wenn dies der Spieler ist. Sie bewegen sich zu einer Target Position, bei Raketen ist dies die zuletzt gesehene Position des Gegners, und bei Bullets die Position vor ihnen entsprechend ihres Headings. Dazu muss sich das Geschoss in Richtung Ziel drehen.

4.5 MLeftRight

Diese Klasse Filtert die Bewegung mittels FilterForce auf Bewegungen nach Links oder Rechts, durch Rotation des Eingangsvektors auf Vector3.left oder Vector3.right, je nach dem was Dichter am Eingangsvektor ist.
Die Klasse setzt einen Move Zustand auf zufällig nach Links oder Rechts Patrolieren.
Sie bietet hilfreiche Funktionen, wie ob es überhaupt möglich ist sich weiter in eine bestimmte Richtung zu Bewegen, oder ob der Spieler sichtbar ist (in Sichtreichweite, LOS und vor ihm entsprechend Heading), und wenn er es ist merkt sie sich dessen Position.
Gegner von diesem Typ sind der Stachel und der RPG-Soldat, wobei der RPG-Soldat neben dem Links und Rechts Patolieren noch weitere Zustände hat.

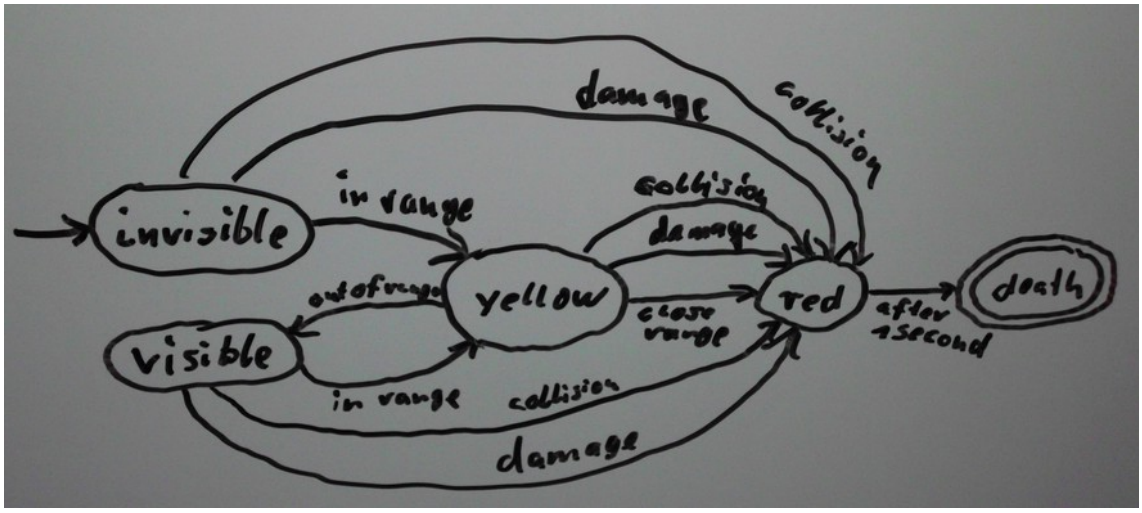
4.6 MLeftRighClimb

Erweitert die MLeftRight Klasse um die Möglichkeit auf Leitern nach oben und unten zu gehen.
Dafür werden einige der Methoden aus MLeftRight überschrieben, sowie weitere nützliche Helferfunktionen definiert. Darunter die Möglichkeit festzustellen, dass wenn man auf einer Leiter ist, Links oder Rechts sich eine Platform befindet, oder ob die Möglichkeit besteht in einer Richtung auf einer Leiter zu gehen.

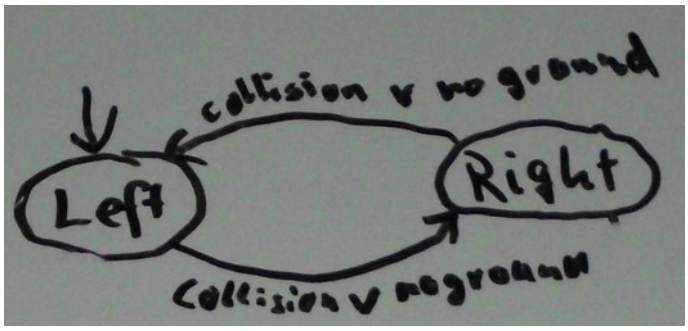
Die einzige konkrete Klasse von diesem Typ ist der Soldat, als einziger Gegner der sich über Leitern bewegen kann.

5. Gegnerzuständ

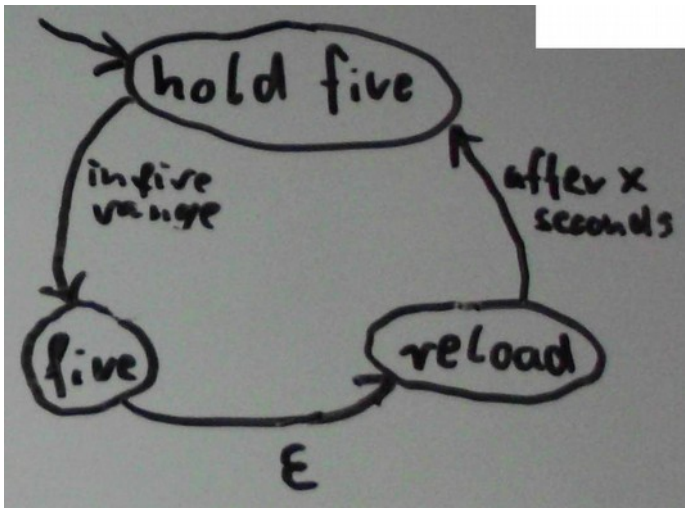
5.1 Mine



5.2 Patrouiere Links/Rechts (Stachel, RPG-Soldat, Soldat)

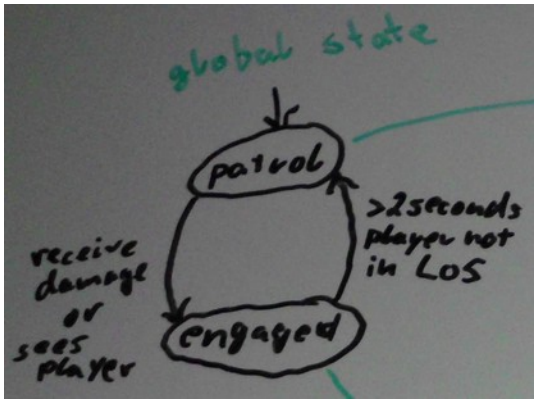


5.3 Angriff (AutomGesch, RPG-Soldat, Soldat)

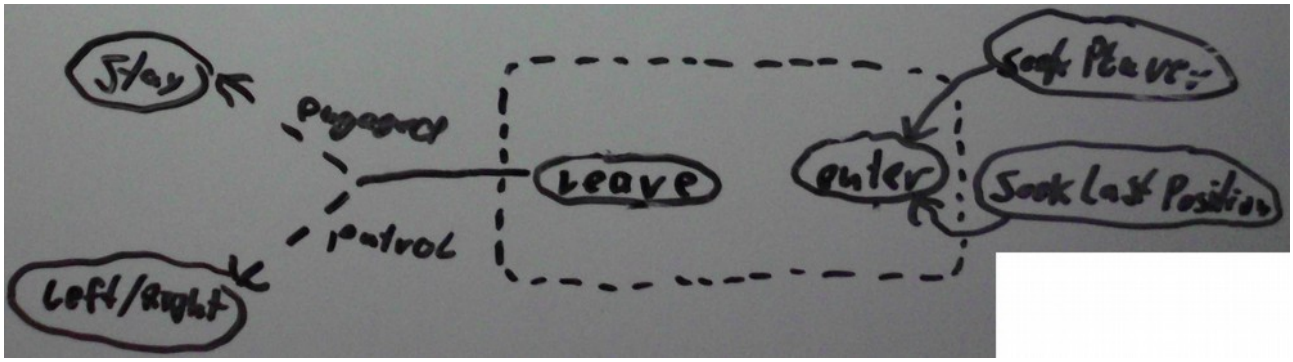


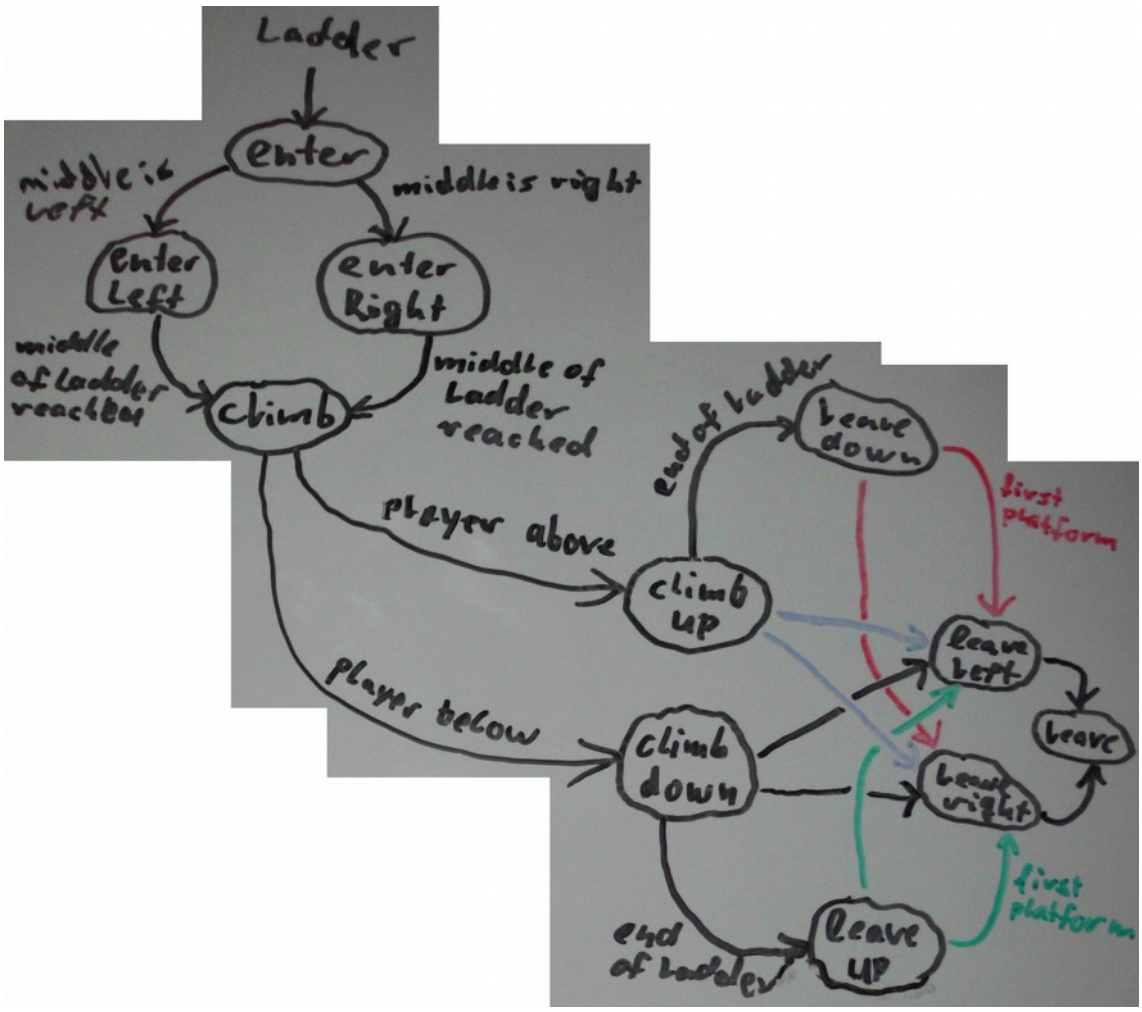
5.4 Bewege (RPG-Soldat, Soldat)

5.5 Global (RPG-Soldat, Soldat)



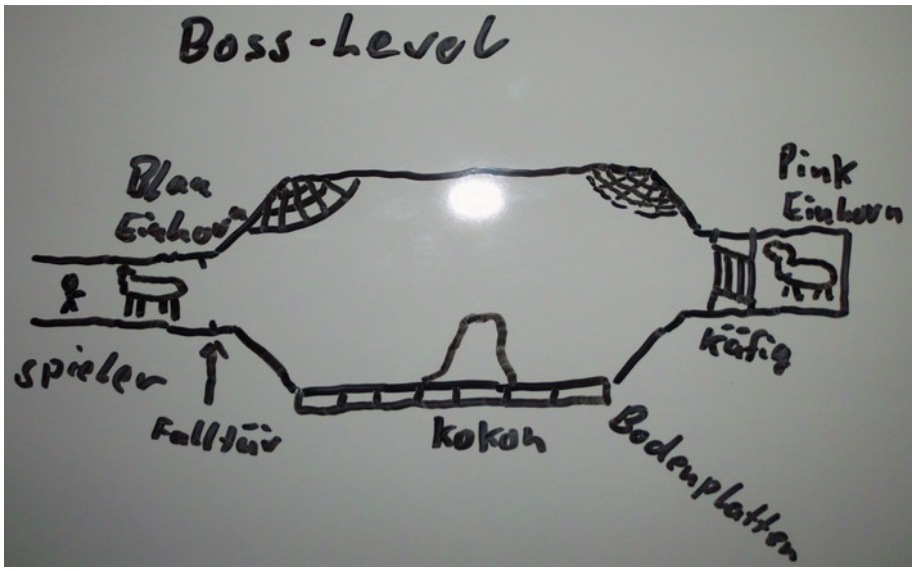
5.6 Leiter (Soldat)



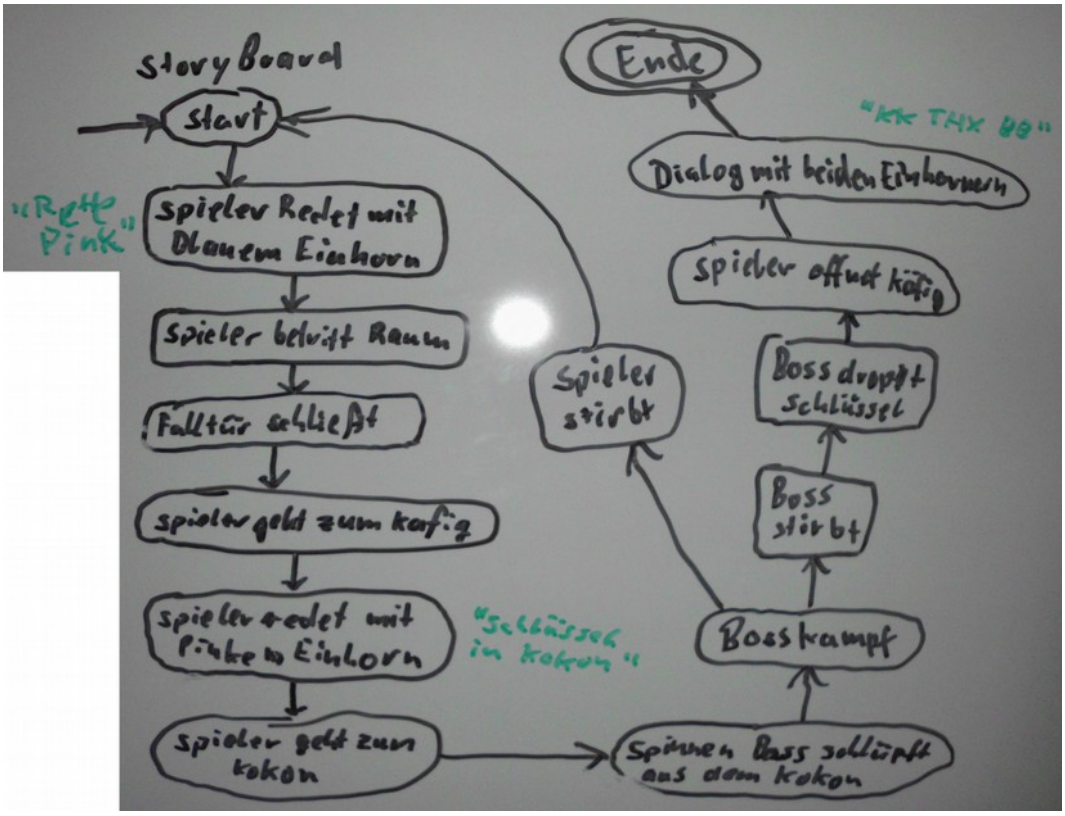


6. Bossgegner

6.1 Boss-Level



6.2 Boss-Story



6.3 Boss-Zustände