Mario AI Competition

Lukas Kalinowski (937763)
Stefan Gohlke (937313)
Frederic Frieß (947998)

Betreuer: Dipl.-Inform. (FH) Markus Schwinn

Inhaltsverzeichnis

1 Einführung in das Thema

Im Rahmen der Veranstaltung *Anwendungen der künstlichen Intelligenz* sollte ein, durch Methoden der künstlichen Intelligenz gesteuerter Mario-Agent, entwickelt werden. Als Basis dieses Projektes diente die inoffizielle Super-Mario-Java-Portation *Infinite Mario*. Bei dieser handelt es sich um eine in Java implementierte Adaption des Mario-Spiels von Nintendo.

1.1 Die Mario AI Competition

Die Mario AI Competition ist ein, vom IEEE Symposium on Computational Intelligence and Games 2009, erstmals durchgeführter Wettbewerb, welcher auf Infinite Mario basiert.
Infinite Mario bietet den Entwicklern die Möglichkeit, über definierte Schnittstellen einen Mario-KI-Agenten zu entwickeln und in das Spiel zu integrieren. Das Ändern von bestehenden Codestücken oder das Hinzufügen von weiterem Code an anderen Stellen des Spiels, ist nur zu Testzwecken erlaubt.
Um den Erfolg der verschiedenen, gegen einander antretenden Mario-Agenten vergleichen zu können, werden im Vorfeld Regeln für die Punktevergabe vereinbart und den Teilnehmern bekannt gemacht. Damit haben alle Teilnehmer die gleiche Ausgangslage und die Möglichkeit, ihre KI möglichst genau an die Regeln anzupassen. In Anlehnung an dieses Verfahren wurde für das Fach *Anwendungen der künstlichen Intelligenz* ebenfalls ein eigener Regelsatz entwickelt, welcher für alle Teilnehmer den Ausgangspunkt bildet.

2 Aufgabenstellung und Rahmenbedingungen

Der Agent soll innerhalb von 500 Zeiteinheiten das ausgewählte Level heil überstehen und währenddessen möglichst viele Gegner besiegen. Hierbei werden die besiegten Gegnertypen wie folgt mit Punkten belohnt:

- Goomba (brauner Pilz): 1 Punkt
- Koopa (rote und grüne Schildkröte): 2 Punkte
- Flower (Blume): 3 Punkte
- Spikey (Stachelpanzer): 4 Punkte

Weiter gehen der Zustand des Marios und die Anzahl gesammelter Coins, in Abhängigkeit zur Durchlaufdauer, in die Punkteberechnung mit ein. Nach folgender Berechnungsvorschrift wird die Punktezahl des Agenten für jedes Level berechnet:

0.7 totalKills + 0.2 MarioStatus + 0.1 Coins / timeSpentOnLevel

3 Spielumgebung

Infinite Mario erzeugt die Levels mit Hilfe eines randomisierten Levelgenerators. Zuvor gibt es die Möglichkeit, die Zufälligkeit des Levels über einen Startwert (Seed) und einen Schwierigkeitsgrad festzulegen. Zu jedem Seed werden die Levels reproduzierbar erstellt.
Als Orientierung für den Mario steht ein 22x22 Einheiten großes Raster zur Verfügung, welches die Informationen (Element-IDs) über die Umgebung enthält. Dieses wird 24 Mal in der Sekunde aktualisiert. Darüber hinaus lassen sich weitere Informationen, wie „Gesundheitszustand“ oder „Bodenkontakt“ über den Mario ermitteln. Um den Mario zu steuern, können verschiedene, den Tastendrücken nachempfundene Aktionen, wie beispielsweise „nach rechts Gehen“, „Laufen“, „Springen“ usw., ausgelöst werden.

4 Beschreibung der Problemlösung

Da sich modifizierte Varianten des A*-Algorithmus bei der Mario AI Competition als besonders erfolgreich erwiesen hatten und der Algorithmus als ein „Quasi-Standard“ für Pathfinding in Computerspielen gilt, fiel die Wahl für dieses Projekt ebenfalls auf den A*-Algorithmus. Kombiniert wurde dieser mit einem Zustandsautomat.

4.1 Die Problemlösung

Im Allgemeinen basiert die Funktionsweise des A* auf einer Menge an Weg-Knoten, sowie einem Start- und einem Ziel-Knoten. Damit der A*-Algorithmus auch für die Mario-Competition eingesetzt werden kann, müssen diese drei Komponenten aus dem Level extrahiert werden. Der Start-Knoten ist in der Regel immer die Mario Position. Die Menge der Weg-Knoten sind die Bereiche, welche die ID der Luft besitzen oder all jene, die kein Terrain-Element sind. Das Ziel muss immer individuell gesetzt werden. Ein Problem bei diesem Ansatz ist, dass die Ziele meist Spiel-Elemente sind und Terrain-Elemente nicht vom A*-Algorithmus berücksichtigt werden. Um dies zu ändern, wurde zwischen zwei Arten von Zielen unterschieden. Den Subgoals, welche markante Stellen innerhalb des Terrains auszeichnen und den Maingoals, welche Gegner, Items, Fragezeichen usw. markieren.
Der Weg zu einem aktuellen Maingoal wird über die dazwischenliegenden Subgoals ermittelt, wodurch der Mario entlang der Terrain-Elemente geführt wird.

5 Praktische Umsetzung

Dieser Abschnitt behandelt die Realisierung des im vorherigen Abschnitt beschriebenen Verfahrens. Die folgende Auflistung beschreibt die nötigen algorithmischen Schritte, auf welche anschließend im Detail eingegangen wird.

1. Extrahieren der markanten Stellen im Terrain (Subgoals)
2. Markieren der Ziele (MainGoals) z.B. Gegner, Fragezeichen
3. Aussuchen eines MainGoals
4. Finden des nächsten Subgoals zum Maingoal
5. Über den A*-Algorithmus den Pfad vom Mario zum Subgoal berechnen
6. Mario-Aktionen für den ersten Schritt des Pfades ausführen

5.1 Extrahieren der markanten Stellen im Terrain

Das 2D-Array *LevelScene* mit der Größe 22x22 beinhaltet die IDs des Terrains und der Spiel-Elemente abhängig von der Mario-Position. Die Füße von Mario befinden sich immer an der Position x=11 und y=11.

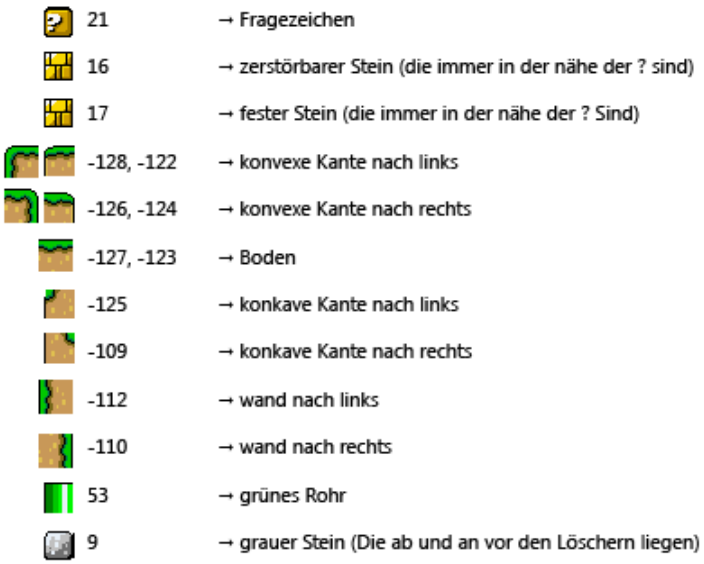

Abbildung 1: IDs der Terrain-Elemente

Durch Überprüfen der Umgebungs-IDs können markante Terrain-Elemente mit Subgoals versehen werden. In der Abbildung 1 sind die IDs mit den entsprechenden Elementen aufgelistet. Die IDs sind unabhängig vom Leveltyp immer gleich. Abbildung 2 zeigt das Level

mit den, in Form von weißen Punkten visualisierten, Subgoals. Die in diesem Abschnitt entnommenen Informationen werden in Abschnitt 5.4 weiterverarbeitet.

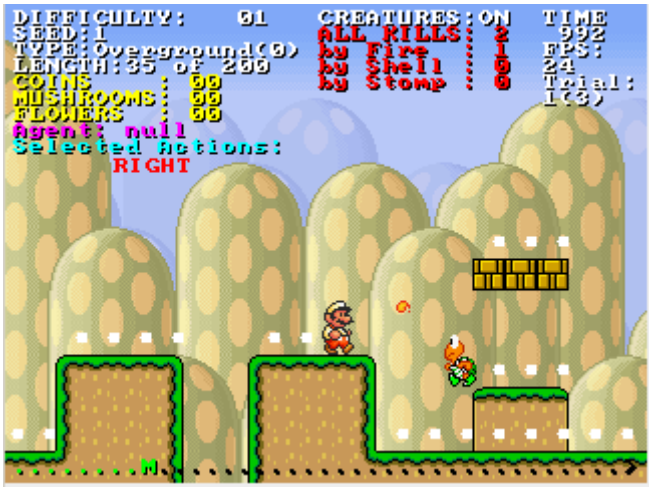

Abbildung 2: Markierte Subgoals

5.2 Markieren der Maingoals

Zum Bestimmen der Maingoals muss die LevelScene in einer anderen Z-Ebene untersucht werden. Die Z-Ebene gibt die Zeichenreihenfolge der Elemente an. Um alle Maingoals zu finden muss in verschiedenen Z-Ebenen gesucht werden. Ebenfalls wie beim Terrain, besitzen die gesuchten Elemente bestimmte IDs, welche nachfolgend aufgelistet werden.

- GOOMBA = 2
- RED_KOOPA = 4
- GREEN_KOOPA = 6
- ENEMY_FLOWER = 12
- MUSHROOM = 14
- FIRE_FLOWER = 15
- QUESTION_BRICK = 21

Abbildung 3: Markierte Maingoals zeigt einen Level-Ausschnitt mit markierten Maingoals. Diese werden immer am Kopf des Gegners angebracht, wodurch der Mario immer recht genau auf den Gegner springt. Darüber hinaus wird stets ein *Default-Maingoal* mit einem festen Abstand zum Mario in die Liste eingefügt. Dieses Goal kommt zum Einsatz, wenn in der Umgebung von Mario keine anderen Maingoals auftauchen, zu denen ein Weg berechnet werden könnte. Damit sich der Agent trotzdem weiterhin nach rechts bewegt, wird so lange ein Pfad zu dem niemals erreichbaren Maingoal berechnet, bis ein neues Maingoal in der Umgebung erscheint.

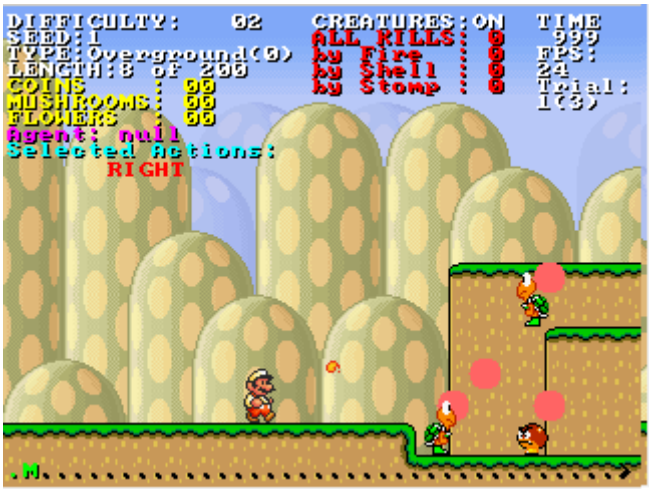

Abbildung 3: Markierte Maingoals

5.3 Aussuchen eines MainGoals

In diesem Schritt wird beschrieben, nach welchen Kriterien die Abarbeitungs-Reihenfolge der Maingoals bestimmt wird, sobald mehrere Maingoals zur Auswahl stehen. Da die verschiedenen Hauptziele für Mario von unterschiedlicher Wichtigkeit sind, werden entsprechende Prioritäten gesetzt. Nachfolgend sind die innerhalb dieses Projekts festgelegten Prioritäten aufgeführt:

1. Die höchste Priorität liegt auf den Items *Pilz* und *Feuer-Blume*. Diese müssen immer als erstes angesteuert werden, damit das Überleben von Mario gesichert wird.
2. Mittlere Priorität haben Ziele links vom Mario. Diese Situation tritt ein, wenn ein Gegner verpasst wurde.
3. Letzte Priorität haben Maingoals rechts von der Mario-Figur. Diese werden nach der Länge des Weges sortiert und das Ziel mit dem kürzesten Weg gewählt.

5.4 Finden des nächsten Subgoals zum Maingoal

In diesem Arbeitsschritt wird versucht, das nächste Subgoal in Richtung Maingoal zu finden (siehe Abschnitt 4.1). Dazu wird eine Winkelberechnung verwendet, mit welcher die Ziele, abhängig zwei Vektoren und deren aufspannenden Winkel, ausgewählt werden. Zunächst wird dazu ein Vektor zwischen Mario und dem aktuellen Maingoal, mit Namen *Maingoal-Vektor*, aufgespannt. Anschließend werden Vektoren zwischen Mario und allen Subgoals berechnet, die *Subgoal-Vektoren*. Nun werden die Winkel zwischen jedem *Subgoal-Vektor* und dem *Maingoal-Vektor* berechnet. Das Subgoal, das den kleinsten Winkel aufweist, ist das nächste Ziel in Richtung des aktuellen Maingoals. In Abbildung 4: Visualisierung der

Vektoren ist das *aktuelle Maingoal* rot und das *aktuelle Subgoal* gelb eingefärbt, sowie die Vektoren eingezeichnet.

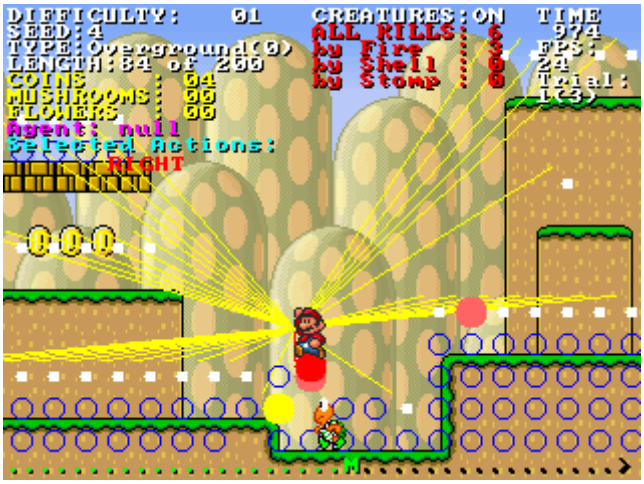

Abbildung 4: Visualisierung der Vektoren

In der Abbildung ist zu erkennen, dass das ausgewählte Subgoal den kleinsten Winkel zum *aktuellen Maingoal* besitzt. Hat Mario nun das, dem *Maingoal* am nächsten liegende *Subgoal* erreicht, wird das *Maingoal* als Zielpunkt verwendet.

5.5 A* Algorithmus zur Pfadberechnung

Die für dieses Projekt verwendete Implementierung des A*-Algorithmus, wurde von [PathFind] entliehen und leicht modifiziert. Die Graphen-Struktur dieses A*-Algorithmus basiert ebenfalls auf einem 22x22 Raster, welches pro Frame initial mit Knoten gefüllt wird. Um nicht begehbare Stellen zu umgehen und dadurch Aktionen von Mario zu erzwingen, gibt es die Möglichkeit Knoten „*invalid*“ zu setzen.

Die "*invaliden Knoten*" werden nicht in die Wegfindung einbezogen und die gesuchten Wege dadurch um sie herum gelegt. Das praktische an diesem Prinzip ist, dass die *Invaliden* dynamisch gesetzt werden können, wie z.B. um Gegner herum oder in Gaps hinein. In Abbildung 5: Invalids sind die nicht passierbaren Knoten als blaue Kreise dargestellt. Aufgrund dieser Bereiche umgeht Mario z.B. die Pflanze automatisch.

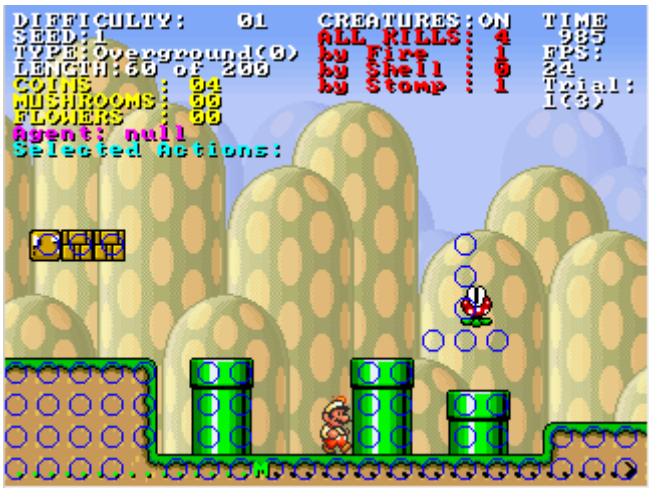

Abbildung 5: Invalids

Der auf diesem Raster basierende A*-Algorithmus arbeitet mit einer *Vierernachbarschaft*. Dies wurde so eingerichtet, da die Diagonalfälle Probleme beim Springen auf Objekte oder Gegner verursachten. Durch verschiedene Tests erwies sich die „Squared-Berechnung“ als eine geeignete Heuristik.

Der Pathfinding-Algorithmus speichert einen gefundenen Pfad. Ein Pfad besteht aus mehreren Punkten, welcher mit Hilfe einer Liste aus mehreren 2D-Koordinaten repräsentiert werden kann. In Abbildung 6: Visualisierung des gefundenen Pfadsist ein solcher Pfad in gelber Farbe dargestellt.

Abbildung 6: Visualisierung des gefundenen Pfads

5.6 Mario-Aktionen für den Pfad ausführen

Sobald ein Pfad zum nächsten Subgoal bekannt ist, muss die Mario-Figur in die entsprechende Richtung bewegt werden. Hierfür wird der erste Schritt aus dem Pfad entnommen. Der Schritt wird durch eine 2D-Koordinate repräsentiert. Diese Position wird von der Mario-Position abgezogen. Der dadurch entstehende Vektor gibt Auskunft über die zu de- oder aktivierenden Tasten. Im Detail werden nun die X- und Y-Werte genauer untersucht. Ist z.B. der X-Wert gleich Null und der Y-Wert kleiner als Null, bedeutet das, dass der nächste Schritt über dem Mario liegt. Die Konsequenz daraus ist, dass Mario durch den Befehl *„action[Mario.jump] = true“* und dem *„false“* setzen aller anderen Aktionen vertikal nach oben bewegt wird. Insgesamt gibt es acht solcher Fälle die eingesetzt werden können. Abbildung 1 visualisiert diese Fälle.

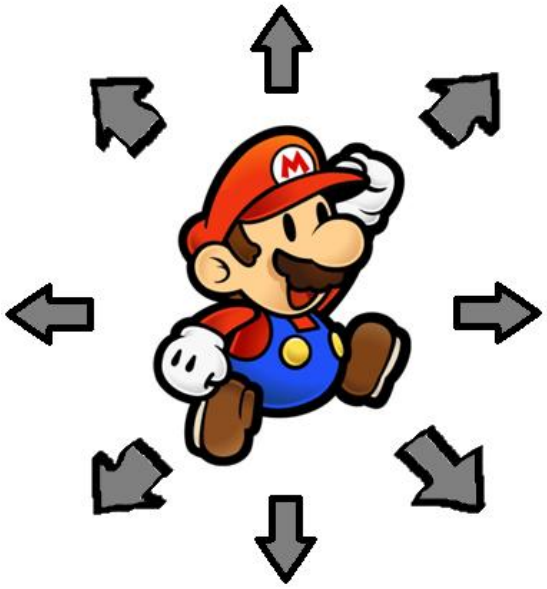

Abbildung 7: Bewegungsrichtungen von Mario

6 Spezielles Verhalten

Da der oben beschriebene Algorithmus die Berechnungen nur in dem 22x22 Raster ausführt, das eigentliche Spiel aber in einem eigenen Pixel-Koordinatensystem stattfindet, treten Differenzen auf, die ebenfalls berücksichtigt werden müssen. Im Folgenden werden Spezialfälle erklärt, die das Mario-Verhalten an schwierigen Stellen beeinflussen.

6.1 Gapverhalten

Das Überspringen von Gaps wird grundsätzlich vom Pfad gesteuert. Es existieren allerdings spezielle Stellen, bei denen der Pfad nicht ausreicht, um den Mario auf die andere Seite zu befördern. Dies ist beispielsweise der Fall, wenn Mario einen Bereich oder ein Hindernis überspringt und dadurch direkt in ein dahinter liegendes Loch fällt. Abbildung 8 zeigt einige spezielle Gaparten.

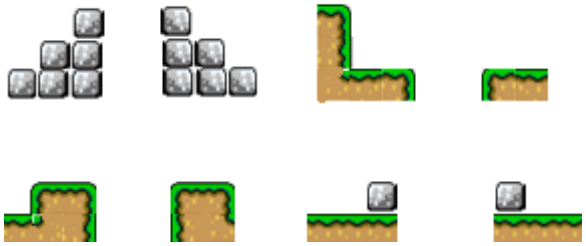

Abbildung 8: spezielle Gapfälle

In diesen Fällen wird die LevelScene nach Steinen und Abhängen durchsucht und bei einem Fund der Mario verlangsamt. Dadurch tastet sich Mario vorsichtiger an gefährliche Stellen an und reagiert präziser auf Gaps.

6.2 Vermeiden von Deadlocks an ungültigen Stellen

Trotz dass die Pfade um die Invalids herum gelegt werden, kann nicht ausgeschlossen werden, dass Mario trotzdem in einen solchen Bereich landet. Tritt dies ein, löst sich der Deadlock durch eine Spring-Aktion von Mario auf. Diese Stellen treten häufig auf, wenn Mario von einer Klippe direkt vor einen Gegner fällt.

6.3 Verhalten wenn kein Pfad vorhanden

Es existieren Bereiche in verschiedenen Levels, in denen der A*-Algorithmus keine Resultate liefert. Daraus folgt, dass Mario keinen Pfad erhält anhand dem er sich fortbewegen kann.
Tritt dieser Fall ein, wird das Level nicht weiter abgearbeitet. Um das Problem zu lösen, wird Mario, je nach Richtung, vorwärts bewegt und die Spring-Aktion wird ausgeführt.

6.4 Reichweite von Zielen

Ziele, die sich außerhalb der Reichweite von Mario, gemessen an der Höhe des Ziels im Bezug zum Mario befinden, werden ignoriert. Oft befinden sich beispielsweise rote Koopa auf hohen Plattformen, welche nicht herunter fallen, wenn sie das Ende einer Plattform

erreicht haben. Mario hatte zunächst immer versucht die hohen Plattformen zu erreichen und ist in die Gegner hineingesprungen. Weiter konnte er an speziellen Stellen die Gegner überhaupt nicht erreichen, was zu einem Deadlock führte. Um dieses Verhalten zu vermeiden, wird die Höhe eines Ziels zur Mario-Y-Position gemessen und nur berücksichtigt, wenn diese einen angemessenen Abstand hat.

6.5 Unerreichbare Ziele

Mit dem vom A*-Algorithmus gelieferten Pfad sind die meisten Ziele gut erreichbar. Es hat sich jedoch in Hinblick auf das Erlangen von Blumen herausgestellt, dass der Pfad an wenigen Stellen so verläuft, dass Mario nicht hoch genug springt um an eine Blume zu kommen, da die Blöcke, auf welchen die Blumen sitzen, nur mit einem „hohen Sprung“[1] erreichbar sind. Daher wird jedes Mal wenn eine Blume auftaucht in einem Timer festgehalten, wie lange das Ziel unerreichbar ist. Wird eine kritische Dauer überschritten, dann ignoriert Mario die Blume und konzentriert sich auf andere Ziele.

6.6 Mario Zustandsautomat

Durch einige Tests wurde festgestellt, dass die Erfolgsquote gering ist, wenn Mario sich im "Klein-Zustand" befindet. Daher wurde ein Zustandsautomat entwickelt, welcher den Mario Status abfragt. Befindet sich Mario nicht im "Blumen-Zustand", so bemüht er sich Fragezeichen abzuarbeiten, um einen Pilz oder eine Blume zu erhalten. Besitzt der Mario hingegen die Feuerblume, so unterbleibt dieses Verhalten.

7 Fazit

Abschließend ist zu erwähnen, dass der in diesem Projekt erstellte Mario-Agent, in den niedrigeren Schwierigkeitsgraden, sowohl gut mit den Gefahren der Umgebung, als auch mit den unterschiedlichen Gegnern zurechtkommt. Aus diesem Grund betrachten wir unsere Arbeit als erfolgreich, wenngleich noch einiger Raum für Optimierung geblieben ist. Das Abdecken sämtlicher Spezialfälle betrachten wir als nahezu unmöglich, da eine Lösung für ein Problem gleichzeitig eine Verschlechterung für eine andere Situation bedeuten kann. Mit wachsender Komplexität ist das Gleichgewicht zwischen allen berücksichtigten Situationen nur schwer beizubehalten. Ebenfalls als schwierig zu beurteilen ist die Integration einer

[1] Sprung- und Speed-Aktion müssen eingeschaltet sein

künstlichen Intelligenz in eine Spielumgebung, in der, z.B. durch eine Physik, sehr feinmaschige Zustände auftreten können. Dadurch, dass das Pathfinding für gewöhnlich für die diskrete Berechnung zwischen zwei Punkten ausgelegt ist, kann es dann zu unvorhergesehenen Ergebnissen führen. Wir folgern daraus, dass die Integration eines KI-Algorithmus in einem solchen Fall nur einen Teil einer Lösung darstellen kann. Zusätzlich müssen verschiedene Spezialfälle, wie in Kapitel 6 beschrieben, abgedeckt werden.

Die Verwendung von Infinite Mario stellte sich in vielen Fällen als schwierig dar. Der vorliegende Code wies Inkonsistenzen in Bezug auf Struktur und Dokumentation auf, was die Einarbeitung und auch den späteren Umgang erschwerte. So waren, entgegen üblicher Konventionen, die X- und Y-Achse im LevelScene-Array vertauscht. Ein anderes Beispiel ist die unstrukturierte Vergabe von IDs. So ist die ID 9 beispielsweise doppelt vergeben - einmal für einen grauen Stein (Terrain) und einmal für den Spikey (Gegner). Der einzige Unterschied besteht in der Z-Ebene. Die Z-Ebenen müssen stellenweise vom Programmierer gesondert gehandhabt werden, was zusätzlichen Aufwand schafft. Die logische Verteilung der Elemente auf die Z-Ebenen war ebenfalls nicht durchgängig.

8 Quellen

[PathFind] Path Finder: Internet 02.03.2010
www.cokeandcode.com/pathfinding

[MaComp] Mario AI Competition: Internet 02.03.2010
http://julian.togelius.com/mariocompetition2009/

[Expl] Mario AI Competition: Internet 02.03.2010
http://julian.togelius.com/mariocompetition2009/CIG2009Competition.pdf

[ZOrder] Zoom: Internet 02.03.2010
http://sites.google.com/site/imarioproject/Home/zoom-levels-zlevels-

[PathFind2] Path Finder: Internet 02.03.2010
http://www.policyalmanac.org/games/aStarTutorial_de.html