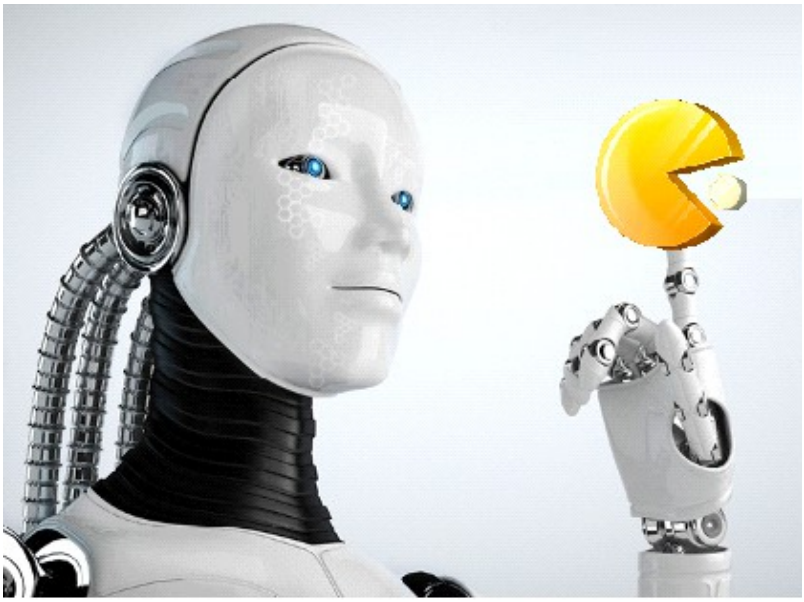


Besondere Lernleistung

Erarbeitung und Anwendung von Konzepten der künstlichen Intelligenz in der Versuchsumgebung des Spiele-Klassikers Pac-Man inklusive einer optionalen Ghoststeuerung über eine Smartphone-App

Manuel Plonski

Ludwig-Geißler Schule, 2.3.2017

Reverse-Pac-Man

Projektdokumentation

Inhaltsverzeichnis

A – Thema und Vorgehensweise

1. Einleitung – Aufbau des Dokumentes

xxx

1.1 Motivation

Der Begriff und die Thematik der „*künstlichen Intelligenz* " (im folgenden mit KI abgekürzt) erweckt eine große Faszination in mir. Deshalb ist mir der Umgang mit der Fragestellung, inwieweit es möglich ist, „auf eine *künstliche* Art und Weise *Intelligenz* und *Rationalität* als eigentlich urmenschlichste Eigenschaft zu erzeugen ", als Problemstellung für diese besondere Lernleistung sehr interessant.

Auch heute hat die künstliche Intelligenz mittlerweile auch unseren Alltag erreicht. Während Siri dank Spracherkennung das Tippen überflüssig macht , arbeitet IBMs Watson nach seiner Karriere als Jeopardy Gewinner als Diagnostiker in der Medizin.

1.2 Thema

Bei der Themenfindung diente im Wesentlichen das Standardwerk über künstliche Intelligenz von Stuart Russel und Peter Norvig, *Artificial Intelligence: A Modern Approach* aus dem Pearsonverlag, 2012 . Mit seinen 1300 Seiten gibt es sowohl ein Überblick als auch eine sehr detaillierte Darstellung der Teilbereiche der Künstlichen Intelligenz.

Ich fokussierte mich auf die Idee, bestimmte KI-Ansätze implementieren und vergleichen zu können. Zur Konkretisierung der Aufgabe schlug Herr Lindenau vor, sich bei der Umsetzung im Gaming-Bereich umzusehen, da es dort sehr gute Möglichkeiten für den Einsatz von künstlicher Intelligenz gäbe.

Auf der Suche nach möglichen Gaming-Anwendungen habe ich im *JgameGrid*-Framework recherchiert. Das Open-Source-Framework *JGameGrid* stellt verschiedene Basisstrukturen bekannter Spiele zur Verfügung. Meine Wahl fiel auf das Spiel Pac-Man. Es eignet sich sehr gut, um die Ansätze der Künstlichen Intelligenz auszutesten, denn bei der Navigation durch das Labyrinth geht es darum, viele sich ändernde Rahmenbedingungen in die Entscheidung für den nächsten Zug miteinzubringen.

Pac-Man soll also von einer künstlichen Intelligenz gesteuert werden und darüber hinaus soll die Funktionsweise des Agenten sichtbar gemacht werden. Im Gegenzug müssen die Ghost des Spiels ja nun von dem User kontrolliert werden und so kam dann noch die Idee der Smartphone-App für die Ghoststeuerung als Themenerweiterung hinzu, sodass die Themenkurzbeschreibung wie folgt lautet:

Entwicklung von „Reverse-Pac-Man “, einer Variante des Spiele-Klassikers Pac-Man mit invertierten Rollen und dabei Pac-Man als KI-gesteuerten Agent mit einer zusätzlichen Ghoststeuerung über eine Smartphone-App.

1.3 Detaillierung des Thema

Im klassischen Pac-Man wird die Pac-Man Figur durch den Menschen kontrolliert. Hierbei besteht die Aufgabe darin, durch das Labyrinth zu steuern, Powerpillen zu verzehren, sowie den Geistern auszuweichen, um einen möglichst hohen Score zu erreichen.

Bei „Reverse-Pac-Man “ sind die Rollen vertauscht. Der Pac-Man wird durch eine „künstliche Intelligenz “ gesteuert, während die Geister über eine App von den Spielern kontrolliert werden.

Der Schwerpunkt dieser Arbeit liegt in der Erarbeitung von Konzepten der künstlichen Intelligenz und deren Anwendung. Das Motiv Pac-Man dient als Versuchsumgebung, um verschiedene klassische Problemstellungen der künstlichen Intelligenz anzuwenden. Hierzu gehören Routenfindung, Ausweichmanöver und Strategieentwicklung.

Ziel ist es, mehrere künstliche Agenten zu entwickeln, die Funktionsweise der Agenten sichtbar zu machen und mit Hilfe geeigneter Kennzahlen miteinander zu vergleichen.

Darüber hinaus wird mit der Entwicklung der Smartphone-App für die Steuerung der Ghosts ein weiterer neuer thematischer Aspekt behandelt.

1.4 Deliverables

Folgende Komponenten sind zu entwickeln:

- verschiedene KI-Agenten
- das Pac-Man-Spiel als KI – Versuchsumgebung auf Basis des JGameGrids
- Visualisierungstools zur Darstellung der KI-Vorgehensweise
- eine Smartphone- App
- eine technische Dokumentation über Javadoc
- die Ausarbeitung in Form einer fachlichen Dokumentation der Komponenten und künstlichen Agenten sowie die Erläuterung einiger Konzepte der künstlichen Intelligenz mit Hilfe des Pac-Man- Szenarios.

2. Vorgehensweise

Nach der **ersten Einarbeitung** in die KI-Thematik und Festlegung der Versuchsumgebung auf das Pac-Man-Spiel habe ich mit mit der Pac-Man Umsetzung im JgameGrid befasst. Die Module und

Architektur wurden untersucht, um die Funktionsweise und vor allem die Schnittstellen zur Steuerung der Ghosts und Pac-Man selbst zu analysieren. Denn der Pac-Man-Reverse vertauscht ja genau die Rollen von Pac-Man und den Ghosts.
Es folgte die **Spezifikation** und Erstellung der **Smartphone-App** sowie der TCP-Bridge.
Im Anschluss wurde eine massive **Modifikation** des sogenannten „**Base-Games** " (xxxxxx9 im Level-Management (Verwaltung der Labyrinth-Level), insbesondere in den Datenstrukturen zur Speicherung des Labyrinths vorgenommen.
Dann folgte eine **Studium**sphase der verschiedenen **KI-Ansätze**, die gleich in Hinblick auf die Tauglichkeit in Bezug auf die Pac-Man-Steuerung untersucht wurden. Nach der Festlegung auf zwei klassische KI-Ansätze xxx reinschreiben welche)xxxx, wurden deren **Umsetzung spezifiziert** und auch gleich **ausgeführt**.
Diese stellte den zeitintensivsten Teil des Projektes dar, da man bei der Entwicklung immer wieder mit Problemen in der Logik und auch in der Umsetzung zu kämpfen hatte.
In dieser Zeit wurden auch eine Art **Toolset entwickelt**, mit dessen Hilfe die Vorgehensweisen innerhalb der **KI sichtbar** gemacht werden können und so die Weiterentwicklung und Optimierung sehr gut unterstützen. Nach Beendigung der Entwicklung folgte der erste **Betatest** auf dem Tag der offenen Tür der Ludwig-Geißler-Schule. Der abschließenden **Competition** der zwei künstlichen Agenten folgte der **Vergleich** und die **Auswertung** der Funktionsweisen und Leistungen der zwei Agenten. Ebenfalls zum Abschluss gehörte die Dokumentation und Ausarbeitung der Projektdurchführung.

2.1 Planung – Projektablauf

xxx insert Graphik Projektplan

3. Pac-Man – Reverse Spielregeln und Bedienungsanleitung

3.1 Spielregeln – Pac-Man

Pac-Man ist ein sehr erfolgreiches Videospiel aus den 1980er Jahren.

Bis heute ist es eines der populärsten Videospiele. Der Spieler manövriert den Pac-Man durch ein Labyrinth, das aus mehreren Pillen besteht.

Ziel ist es, alle Pillen zu fressen. Abbildung xxx1 illustriert einen typisches Labyrinth.

Es enthält xx Pillen, wobei Pille mit 1 Scorepunkten bewertet wird. Außerdem gibt es noch Ghosts im Labyrinth – sie versuchen Pac-Man zu fangen. Sind sie erfolgreich , verliert ist das Spiel verloren. Ein Level ist beendet, wenn alle Pillen oder alle Ghosts aufgefressen sind. (WINxx)

Außerdem gibt es noch Power-Ups, die sogenannten Powerpillen, sie sind mit jeweils 2 Scorepunkten bewertet.

Wenn Pac-Man eine solche Powerpille frisst, werden alle Ghosts essbar, d. h. die Geister verlangsamen sich für die Dauer von 15 Sekunden. In dieser Zeit kann Pac-Man die Ghosts fressen und erhält 2 Scorepunkte pro gefressenen Ghost

Unsere Untersuchungen beschränken sich auf das Labyrinth mit der Bezeichnung "Level1 ". Wobei der maximal erreichbare Score xxx ist.

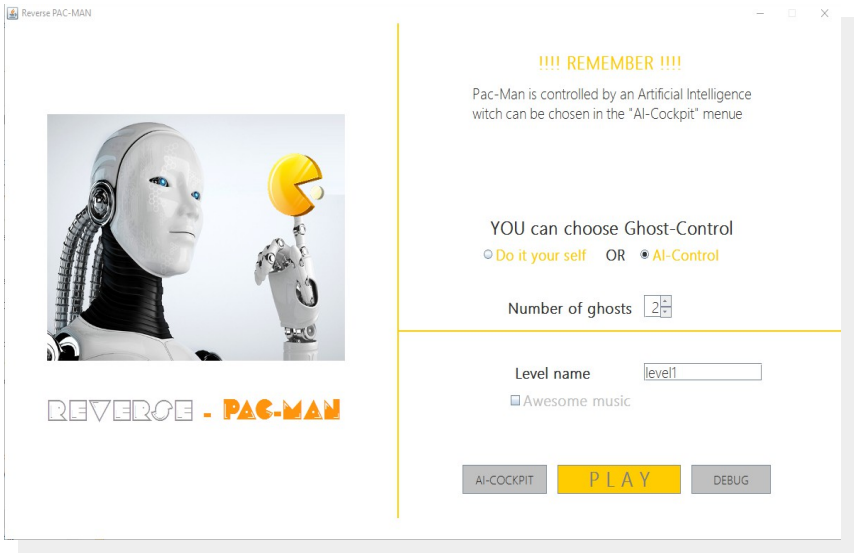
In den Untersuchungslabyrinthen gibt es ca. 290 distincte Koordinaten, die erreichbar sind.

3.2 Bedienung Pac-Man Reverse

Installationshinweise

Die xxx Datei Pacman.jar muss sich zusammen mit dem Ordner xxxxx im … befinden.
Für die Editierung neuer Spiellevels kann in der pacmann datei xxxx entsprechend der Vorlage aus xxxx o/ bzs für xxxx ein neues Level erstellt werden.
Zur optimalen Ausnutzung aller Visualisierungstools und Interaktionsmenues ist es empfehlenswert, die Anwendung auf ausreichend großen Monitoren, im günstigsten Fall sogar auf zwei Monitoren, auszuführen.

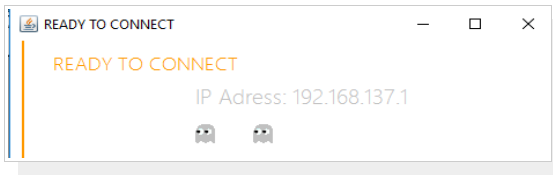
Start von Reverse – Pac – Man durch Aufruf von xxxxx.



Reverse – Pac-Man Menue

❶ Im Hauptmenue wird zunächst die Ghoststeuerung festgelegt:
Entweder kontrollieren ein- oder mehrere User die Ghosts über die **GhostApp,** oder die **Ghost-KI** des Pac-Man-Reverse übernimmt die

Steuerung. Die letzte Variante eignet sich gut, um die Funktionsweise des Pac-Man über das AI Cockpit zu untersuchen.



Ist die **App-Variante** ausgewählt, so müssen sich bei Spielstart entsprechende viele User über die App einwählen.
Zur Unterstützung bei der Einwahl durch die GhostApp öffnet sich der **Ready to connect** Screen: Er zeigt die IP-Adresse des Rechners an, die in der GhostApp für den Connect eingetragen werden muss. Auch werden die noch ausstehenden Verbindungen in Form von grauen Ghosts dargestellt.

❷ Auch kann die Anzahl der Ghosts festgelegt werden.

❸ Im nächsten Schritt wird das Level ausgewählt. Aktuell stehen die folgenden Level zur Auswahl :

(1) LEVEL1 : Labyrinth xkreuzx: Dieses Labyrinth dient zum Vergleich der verschiedenen Kis
(2) LEVEL2 : Labyrinth xkreuzx: Kleineres Labyrinth zum schnellen austesten
(3) LEVEL3 : Labyrinth xkreuzx: BLL – Überraschungslabyrinth

❹ Mit der Checkbox xxxx Kästchen kann die Musikauswahl zum Spielen variiert werden

❺ Vor der Start des Spieles gibt es noch die Möglichkeit, im Debug-Menue verschiedene Optionen zur Speicherung der Nodes und Verbindungen des Labyrinthes anzeigen zu lassen

❻ Eine besonders zentrales Menue ist das AI-COCKPIT, welches vor dem Start des Spiel aufgerufen werden sollte.

Artificial Intelligence Cockpit

Das Cockpit dient neben der **Wahl** des **künstlichen Agenten** zur **Darstellung** und **Analyse** des KI-Vorgehens.

Zusätzlich lassen sich Statistiken zu bestimmten KI-Konfigurationen abrufen sowie die Geschwindigkeit einstellen. Das Cockpit sollte immer vor dem Spielstart aus dem Main-Menue aufgerufen werden.

VOR dem Spielstart kann Folgendes festgelegt werden:

❶ Im Drop-Down-Menue wird die Selektion des künstlichen Agenten vorgenommen und mit APPLY bestätigt – die Ausgewählte KI wird dann im rechten Teil der Anzeige dargestellt.

- MINIMAX: Gametree-Mini-Max-Agent mit limited-depth Baumtiefe unter Nutzung einer Evaluierungsfunktion – eine weitere Spezifikation der dazugehörigen KI-Parameter und Visualisierung dieses Agenten erfolgt über das *Mini-Max-Agent Menue.*
- SMART-REFLEX-AGENT: Smart-Reflex-Agent als regelbasierter Agent mit smarter, planender Komponente - – eine weitere Spezifikation der dazugehörigen KI-Parameter und Visualisierung dieses Agenten erfolgt über das *Smart-Reflex- Menue.*
- RANDOM: Zufalls-Agent: Agent zur Demonstration der Arbeitsweise **ohne KI**
- SIMPLE_FIND_PILLS: Rein pillensuchender Agent, der sich immer die am nächsten gelegene Pille heraussucht – lässt die Ghosts außer Acht. Beim Spielen dieses Agenten sollten im Main-Menue die Anzahl der Ghosts auf 0 gesetzt werden.
- CRAZY_FIND_PILLS: Rein pillensuchender Agent ohne Rücksicht auf die nächstgelegene Pille – lässt die Ghosts außer Acht. Beim Spielen dieses Agenten sollten im Main-Menue die Anzahl der Ghosts auf 0 gesetzt werden.
- GHOST_AVOID: Agent, der den Ghosts ausweicht

❷ Festlegung der Simspeed: Regelung der Spielgeschwindigkeit durch Festlegung der „Wartezeit " zwischen zwei Bewegungen/Zügen der Spielfiguren.

❸ Aufruf des Statistikfiles: Das Statistik-File mit Kennzahlen zur Performancemessung wird für bestimmte KI-Konfigurationen geschrieben, um eine Vergleichbarkeit zwischen den Agenten zu ermöglichen. Nur bei Auswahl von LEVEL1 und 2 Ghosts, sowie einer Baumtiefe von 3 werden die Statistiken geschrieben.

WÄHREND des Spieles :

❹ Während der Spielausführung kann durch Pausieren das Spiel **unterbrochen** werden, um z.B. beim Minimax-Agenten die Graphen der Utility oder oder den Plot des Minimax-Game-Trees anzeigen zu lassen.

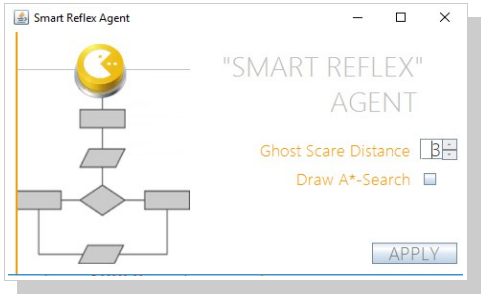
Mit Start kann das Spiel wieder gestartet werden oder auch mit Step eine **schrittweise Verarbeitung** gewählt werden.

❺ Auf der rechten Hälfte des Frames werden wichtige KI relevante Informationen dargestellt. Zum einen wird der Status des Agenten dargestellt. Hier gibt es die Ausprägungen

Go xxx not Runningxxx Runningxxx Thinkingxxx

Pausexxx Step xxx

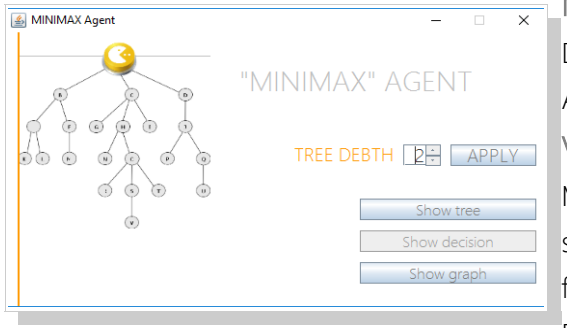
Desweiteren wird mit den Pfeilen die aktuelle „Gehrichtung ", die durch die Ki ermittelt wurde, angezeigt. Außerdem gibt es noch den Zustand des Powermodes, der sich bei vorliegen des Powermodes rot verfärbt.



Smart-Reflex-Agent- Menue

Dieses Fenster wird aktiviert, wenn der Reflex-Agent im AI-Cockpit ausgewählt wird. Hier lässt sich die Größe des Toleranzbereich einstellen, bei dem ein Ghost-Alarm ausgelöst wird.

Zusätzlich gibt es noch die Möglichkeit, dem A*Search beim Auffinden der nächstgelegenen Pillen zuzusehen, in dem die Linie der direkten Verbindung zu den Punkten aufgezeigt wird.



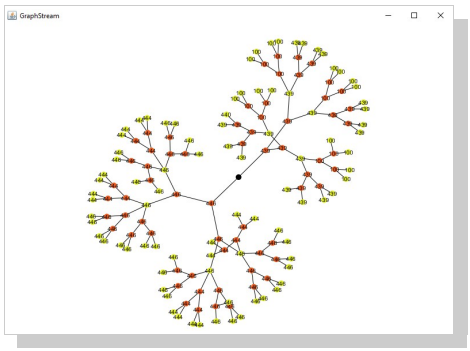
Minimax-Agent- Menue

Dieses Fenster wird aktiviert, wenn der Minimax-Agent ausgewählt wurde.

VOR dem Spielstart :

Mit der Tree-Depth lässt sich die Anzahl im Baum zu simulierenden Spielzüge für Pac-Man und die Ghosts festlegen.

Bei einer Tree-Depth von 3 in einem Spiel mit zwei Ghosts ergibt sich eine gesamte Baumtiefe von 3*(1+2) = 9.

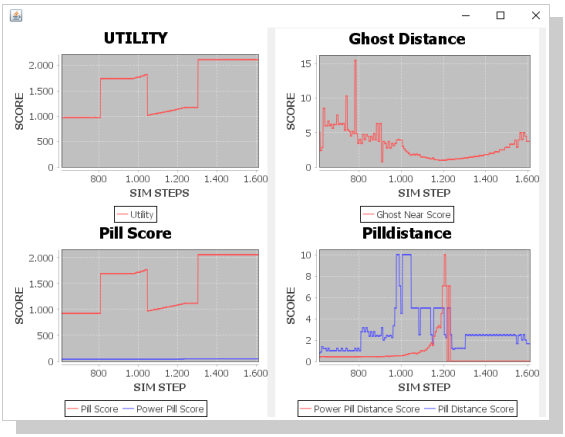


Während des Spieles:

Anzeige des Gametrees mit den Uitility-Werten

Über den Botton „Show Tree " kann der aktuelle Gametree und über den Botton „Show Decision " den für die Entscheidung genutzten Pfad anzeigen lassen.

Zuvor muss das Spiel mit dem „Pause " Botton gestoppt werden !



Anzeige des Gametrees mit den Uitility-Werten

Über den Botton „Show Graph " kann der aktuelle Verlauf der Utility und deren Einzelkomponenten dargestellt werden.

B – Konzepte künstlicher Intelligenz in der Pac-Man Versuchsumgebung

1. Künstliche Intelligenz und deren Definition

Die Ausarbeitung zur den theoretischen Grundlagen der künstlichen Intelligenz basieren auf dem schon zuvor genannten Buch „Artificial intelligence – a modern approach ".

Zum Einstieg in das Thema ist es immer sehr sinnvoll, sich sich mit der Definition des Begriffes der künstlichen Intelligenz zu beschäftigen. Es existiert eine große Vielfalt von Definition zur künstlichen Intelligenz, hierbei lassen sich die Definitionen wiederum bestimmten Kriterien zuordnen.

Eine Gruppierungsmöglichkeit ist die Gruppierung nach dem Kriterium des Denken und Handelns . Es bezieht sich auf Denkprozesse und logisches Schließen, bzw. auf das Verhalten .

Die andere Gruppierungsmöglichkeit unterscheidet zwischen der Wiedergabetreue der menschlichen Leistung versus der Rationalität. .

Damit haben wir also folgende Aufteilung:

Systeme, die wie Menschen denken „[die Automatisierung von] Aktivitäten, die wir dem menschlichen Denken zuordnen, Aktivitäten wie beispielsweise Entscheidungsfindung, Problemlösung, Lernen " (Bellmann ,1878)	Systeme, die rational denken „Das Studium derjenigen mathematischen Formalismen, die es ermöglichen, wahrzunehmen, logisch zu schließen und zu agieren. " (Winston, 1992
Systeme, die wie Menschen handeln „das Studium des Problems, Computer dazuzu zu bringen, Dinge zu tun, bei denen ihnen momentan der Mensch noch überlegen ist. " (Rich und Knight, 1991	Systeme, die rational handeln „Computerintelligenz ist die Studie des Entwurf intelligenter Agenten. " (Poole et al., 1989)

(Tabelle analog „Stuart Russel und Peter Norvig, *Artificial Intelligence: A Modern Approach* . Pearsonverlag, 2012, S. 23 ")

1.1 Turing-Test

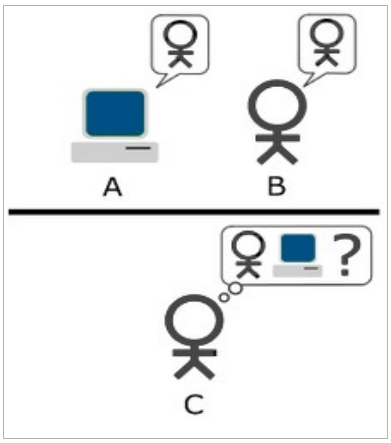


Abbildung 1: Versuchsaufbau Turing-Test

Ein wichtiger Ansatz der Definition von KI in Bezug auf *Systeme, die wie Menschen handeln* ist der **Turing-Test**. Der aus der Anfangszeit der KI stammende, fast schon legendäre Test wurde 1950 von Alan Turing entwickelt. Es ging um den Vergleich des Denkvermögens von Mensch zu Maschine.

Der Versuchsaufbau: Eine reale Person **C** sitzt einer weiteren realen Person und einem Computer durch eine Wand getrennt gegenüber und zu entscheiden, ob es sich bei **A** oder **B** jeweils um einen Computer oder einen Menschen handelt. **A** und **B** versuchen C davon zu überzeugen, dass sie selbst Menschen sind.

1.2 Fokus auf rationale Agenten

Da unsere zu entwickelnde KI die Steuerung des Pac-Man durch den Menschen ersetzt, könnte man erwarten, wir würden den Ansatz **die *KI handele wie ein Mensch*** verfolgen– jedoch liegt der Fokus dieser Ausarbeitung auf Systemen, **die rational handeln**.

Rationales Verhalten bedeutet, „das Richtige zu tun ". Dabei verstehen wir unter dem „Richtigen " das, von dem wir erwarten, dass es das gewünschte Ziel, basierend auf der vorliegenden Information maximieren kann,. Das Handeln kann hier auch vorausgehendes Denken beinhalten, sonst würden wir von „Reflexen " reden.

Die zentrale Aufgabenstellung der besonderen Lernleistung ist der Entwurf von intelligenten Agenten. Ein Agent kann als Entität verstanden werden. Er empfängt Wahrnehmungen aus seiner Umgebung und kann in dieser Umgebungen Handlungen ausführen

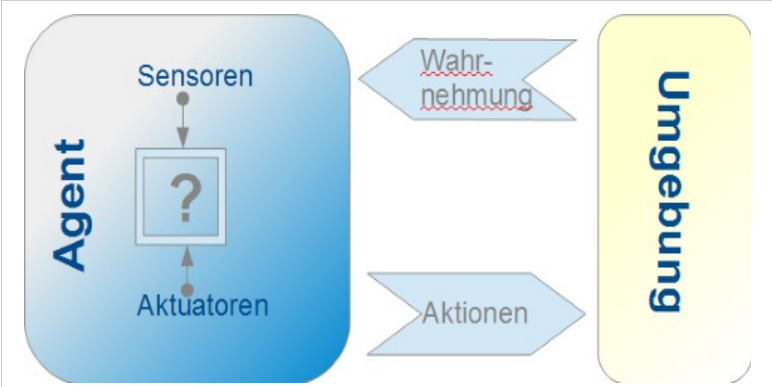


Abbildung 2: Agenten kommunizieren über Sensoren und Aktuatoren mit ihrer Umgebung

Dabei kann ein Agent immer die eigenen Handlungen wahrnehmen, notwendigerweise aber nicht den Effekt der Handlung auf die Umgebung. Ein rationaler Agent optimiert bestimmte Leistungskriterien. Für eine gegebene Aufgabe und eine bestimmte Umgebung suchen wir also den Agenten mit der besten Leistung. In vielen Anwendungen ist durch die Einschränkung auf technischer Seite die Erreichung der perfekten Rationalität unerreichbar.

Unser Agent benötigt **keine** realen **Sensoren** - in unserem Anwendungsfall erfolgt die Wahrnehmung durch die **Auswertung des Gamestate** über den Zugriff auf die Positionen und Laufrichtungen von Pac-Man und den Ghosts.

1.3 Agentenprogramme

Die wesentliche Herausforderung der KI ist es, Programme zu entwickeln, die rationales Verhalten innerhalb weniger Lines of Code erzeugen und die nicht von extrem großen Entscheidungstabellen abhängig sind.

Die wesentlichen Anforderungen für eine KI, die den Pac-Man ersetzt, sind die folgenden:

1. Das Verzehren möglichst vieler Pillen - Durchlaufen des Labyrinths
2. Das Ausweichen vor den Ghosts
3. Taktik – Management der Punkte eins und zwei unter dem Gesamtziel der Scorepunktmaximierung

Es gibt vier Grundtypen von Agentenprogrammen: - sie beschreiben die Prinzipien, nach denen viele intelligente Systeme arbeiten.

Einfache Reflexagenten

Die Aktion wird auf Basis der aktuellen Wahrnehmung ausgewählt – ein möglicher zukünftiger Wahrnehmungsverlauf wird ignoriert.

Modellbasierte Reflexagenten

Mit Hilfe eines internen Modells wird der aktuelle „Zustand " der Welt ermittelt, wenn nicht alle Zustände zu beobachten sind. Aufgrund dieser Basis wird dann wie beim Reflexagenten die Aktion gewählt.

Modellbasierte, zielbasierte Agenten

Mit Hilfe eines internen Modells wird der aktuelle „Zustand " der Welt , wenn nicht alle Zustände zu beobachten sind. **Zusätzlich werden die Ziele, die er erreichen** soll verwaltet und letztendlich wählt er die Aktion aus, die am ehesten zum erreichen dieser Ziele dient.

Modellbasierte, nutzenbasierte Agenten

Auch hier gibt es ein internes Modell zur Verwaltung des aktuellen „Zustand " der Welt. Mit Hilfe einer Nutzenfunktion werden seine Vorlieben in Bezug auf die verschiedenen Zustände der Welt bewertet. Dann wählt er die Aktion aus, die am besten zum erwarteten Nutzen führt.

Nutzenbasierte Agenten versuchen, ihre eigene erwartete „Zufriedenheit " zu maximieren.

Alle Agenten können ihre Leistung durch „Lernen " verbessern !

1.4 Planende Agenten

Ein **Reflexagent** denkt nicht über die Konsequenzen seines Handelns nach. Er wählt die gewünschte Aktion anhand des aktuellen Zustandes der Umgebung aus.

Ganz anders verfahren die **planenden Agenten**.. Sie planen eine Abfolge von Handlungen und überlegen, wie die Umgebung im Anschluss aussehen würde. Optimal wäre der Agent, wenn er die Lösung mit den geringsten Kosten auswählen würde. Endlichxx wäre der Agent, wenn es zu seiner Vorgehensweise auch eine Lösung existiert. Manchmal ist es notwendig, ein „Replanning " einer bereits vorgesehenen Handlungsabfolge durchzuführen.

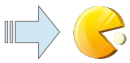
1.4.1 Problemlösen durch Suchen

Wenn eine Problemstellung aus der realen Welt sich in eine sogenannte *Suchproblemstellung* transferieren lässt, dann existiert immer ein Verfahren zur Lösung des Problems.

Ein solches *Suchproblem* kann formal mit Hilfe der folgenden Eigenschaften definiert werden:

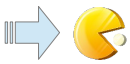
- Statespaces: die Abstraktion der Umgebung, die für das Problem relevant ist
- Ausgangszustand: Statespace, der den Ausgangszustand darstellt.
- einer Beschreibung der möglichen Aktionen
- einer Beschreibung, was jede Aktion bewirkt (Überführungsmodell oder Successorfunktion)
- Zieltest
- Pfadkostenfunktion

Zustandsraum des Problems:
Menge aller Zustände, die von diesem Ausgangszustand durch eine Aktionsfolge erreichbar ist

 Anwendung in der Pac-Man-Umgebung:

Auch im Pac-Man lassen sich typische Suchprobleme identifizieren, z.B. die Bestimmung der kürzesten Route zwischen zwei Punkten im Labyrinth . Den Statespace bilden die begehbaren Punkte des Labyrinth mit Position von Pac-Man, die Ausgangs- und Zielknoten sind bekannt. Als Aktionen dienen die möglichen Bewegungsrichtungen oben, unten, links, rechts, während die Verbindungen zwischen den Punkten als Nachfolgefunktion agieren. Die Pfadkostenfunktion ist relativ simpel mit einem Kostenpunkte pro gelaufenem Schritt. Der Zieltest wird durch den Vergleich der aktuellen Position mit der gewünschten Zielposition realisiert.

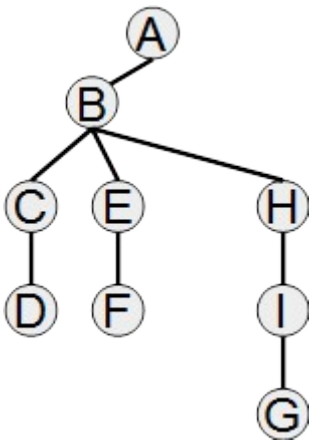
Wichtig ist es, den Statespace auf das jeweilige Problem anzupassen und nur relevante Informationen mit hineinzunehmen. Wenn alle vorhandenen, auch nicht relevanten Eigenschaften bei der Bildung des Statespaces genutzt werden, entstehen sehr schnell eine extrem große Vielzahl von Statespaces.

 Anwendung im Pac-Man:

- Die Varianten in den Statespaces im Pac-Man lassen sind aus dem folgenden Produkt herleiten:
- Anzahl der möglichen Pac-Man-Positionen: (32*52xx)
- Zustände der Pillen /Powerpillen im Grid: 2^{*53xx}
- Möglichkeiten der Ghostposition: (für 4 Ghosts) $(32*52xx)^4$
- Anzahl Aktionen: 4

Daraus ergibt sich ein Produkt in Höhe von xxx.

1.4.2 Der Suchbaum des planenden Agenten



Zeichnung 1: Suchbaum

Der Suchbaum wird in den Algorithmen zur Lösung des Suchproblems genutzt. Vom Start-Statespace – Knoten beginnend (A) wächst der Baum in Abhängigkeit der möglichen Aktionen: Die Anzahl der möglichen Aktionen ausgehend von einem Knoten bestimmt die Anzahl der Children des Knotens.. Die Knoten sind nicht nur eine Ausprägung eines Statespaces, sie stellen außerdem auch einen Plan oder auch Route ausgehend von der Wurzel dar. Dabei wird der Baum meist nur für einen kleinen Ausschnitt des Problem gebildet. Es geht also darum, eine begrenzte Anzahl verschiedener Pläne zu entwickeln.

Zur Lösung eines Suchproblems können verschiedene Algorithmen genutzt werden. Die grundsätzliche Vorgehensweise bei der Baumsuche ist bei allen Algorithmen immer die gleiche. Sie unterscheiden sich lediglich in der *Strategie* bei der Expansion.

```
function TREE-SEARCH (problem, stategy) returns a solution or failure
    initializie the search tree using the initial state of problem
    loop do
        if there are no candidates for expansion then return failure
        choose a leaf node for expansion according to strategy
        if the node contains a goal state then return the corresponding solution
        else expand node and add the resulting nodes to the search tree-search tree
        end
```

Die Algorithmen zur Lösung eines Suchproblems lassen sich unterschieden, je nach dem, ob Informationen über die Lage des Ziel-State-Spaces vorhanden sind oder nicht. Sie werden als *uninformierte* und *informierte Suche* bezeichnet.

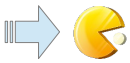
Beim uninformierten Suchen liegt keine Information zur Ziellokation vor. Hier gibt es zum Beispiel den *depth first search*, der den Suchbaum mit der Priorität auf die Baumentwicklung in die Tiefe, also in Richtung der Blätter vorantreibt. Zum anderen gibt es den *uniform cost search*, dessen Expansionsstrategie auf der Kostenminimierung beruht.

Die informierte Suche nutzt Heruristiken, die uns darüber informieren, ob man sich zum Ziel hin- oder wegbegegt. Bei der Heuristik handelt es sich um eine Zuordnung des State-Spaces zu einer realen Zahl. Je nach dem, wie weit man vom Ziel entfernt ist, ist sie höher oder entsprechend geringer.

1.4.3 A*-Search

Eine Variante der informierten Suche ist die A* - Suche . Sie kombiniert die „Kostenminimierung " mit dem „geringsten Abstand zum Ziel " als Heuristik und dient somit als Strategie bei der Expandierung des Baumes.

Die Idee des Algorithmus ist es also, immer zuerst die Knoten zu untersuchen, die am wahrscheinlichsten *schnell* zum Ziel führen. Um diese Knoten zu ermitteln, wird mit Hilfe der Luftliniendistanz zum Ziel der potenziell wahrscheinlichste ausgewählt.

 Anwendung in der Pac-Man-Umgebung:

Route zwischen zwei Punkten

Eine wichtige Aufgabe bei der Navigation in einem Labyrinth ist es, zwischen zwei Punkten eine genaue Route und deren Länge zu bestimmen. Ich habe hierfür den A*-Search-Algorithmus gewählt, weil er effizient den kürzesten Weg zurückgibt.

Voraussetzung zur Anwendung des Algorithmus ist das Vorliegen der Information über alle begehbaren Punkte und für jeden Punkt die Kenntnis über die möglichen Richtungen, in die von diesen Punkten aus verzweigt werden kann.

Für jeden weiterzuentwickelnden Punkt wird eine Kennzahl, der der sogenannte *Wayscore* ermittelt. Er wird aus der Entfernung (Kostenaspekt) der bisher zurückgelegten Strecke vom Startpunkt(Root) des Baumes bis zum aktuellen Punkt ermittelt plus der Luftliniendistanzxx zum Ziel (infomierter Aspekt). Bei der Weiterentwicklung der Knoten wird jetzt derjenige Children-Knoten priorisiert, bei dem der Wayscore am geringsten ist.

So erhält man bei positiven Zieltest die kürzeste Verbindung zwischen den zwei Knoten.

Die Vorgehensweise des A* - Searches lässt sich bei der Ausführung des Reflexagenten im Reverse-Pac-Man beobachten. (Siehe Kapitel xxxx).

2. Reflexagent *Smart-Reflex-Agent*

Auf Basis der bislang dargestellten Inhalte habe ich meinen ersten Agenten mit dem Name „Smart-Reflex-Agent " entworfen. Eigentlich wollte ich einen ganz einfach gehaltenen **Reflex-Agenten** entwickeln. Dieser hätte den Game-State der aktuellen Spielsituation ausgelesen und mit Hilfe einer Entscheidungshierarchie die nächste Aktion bestimmt. Wie schon erwähnt, denkt ein **Reflexagent** nicht über die Konsequenzen seines Handelns nach. Er wählt die gewünschte Aktion anhand des aktuellen Zustandes der Umgebung aus.
Doch bei der Bearbeitung der Algorithmen zur Lösung von Suchproblemen war ich ganz besonders fasziniert von der A*-Suche, die eine „planende Komponente " durch den Blick auf das Ziel darstellt. Und so habe ich bei der Entwicklung des ersten Agenten noch eine „**smarte** " Komponente hinzugenommen und die informierte **A* - Suche** integriert

2.1 Erläuterung des Smart-Reflex-Agenten

 Anwendung in der Pac-Man-Umgebung:

Die Idee dieses Konzeptes ist, dass Pac-Man mit erster Priorität den Ghosts ausweicht. Mit zweiter Priorität bewegt er sich in Richtung der nächstgelegenen Pille. Das Ausweichen der Ghosts wird ignoriert, wenn Pac-Man eine Powerpille gefressen hat.

Generell kann sich Pac-Man also in zwei verschiedenen Modi befinden kann. Zum einen gibt es den *Ghost_avoid*-Modus in dem es das oberste Prinzip ist, den Ghosts auszuweichen und zum anderen gibt den *Power_pill*-Modus, der nach dem Konsum einer Powerpille für xxx Takteinheiten aktiviert wurde. Es kann immer nur ein Modus aktiv sein.

1. **solange sich Pac-Man nicht im *Powerpill*-Modus befindet, befindet er sich im *Ghost_avoid*-Modus:**

   (a) Die Luftlinienentfernung zu allen Ghosts wird bestimmt
   (b) Befindet sich ein Ghost im Toleranzbereich von xxx=3, so wird mit Hilfe des A*-Searches wird die Route zu diesem Ghost bestimmt und als möglichen Labyrinth-Wege-Punkt für Pac-Man geblockt

2. Ebenfalls mit dem A* Search wird die Route zur am nächsten gelegenen Powerpille (nach Luftlinie) bestimmt

(a) Pac-Man verzweigt dann in diese Richtung
(b) Gibt es keine Abzweigungsmöglichkeit, ist Pac-Man umzingelt und verliert (Konsolenoutput xxxxx) weitere Konsolenoutpu nachpflegen
Anstelle der Luftliniendistanz hatte ich auch eine Variante implementiert, bei der tatsächliche Wegdistanz von Pac-Man zu jeder noch verfügbaren Pille im Grid berechnet hat und daraufhin die mit der geringsten Distanz ausgewählt hat. Aus Performancegründen habe ich mich dann doch für die Luftliniendistanz entschieden.

3. Befindet sich Pac-Man im *Powerpill*-Modus so sucht er mit A* Search die Route zur am nächsten gelegenen Powerpille und Pac-Man verzweigt dann in diese Richtung

2.2 Probleme im Verhalten des Agenten

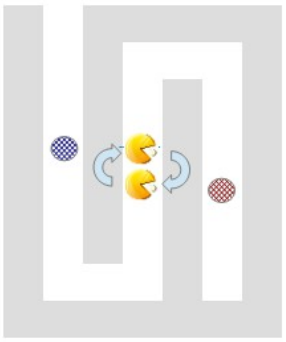


Abbildung 3: Zappeln

Beim der Umsetzung des Smart-Reflex-Agenten mit der oben beschriebenen Vorgehensweise hat sich eine besondere Problematik ergeben, die ich im folgenden als *Zappelproblematik* beschreiben werde. Wie in Abbildung xx zu skizziert, „zappelt " Pac-Man endlos zwischen zwei Positionen hin- und her und kommt nicht mehr von diesen zwei Postionen fort.

Die Ursache liegt im Wechsel der als nächstes anzusteuernden Pillen:

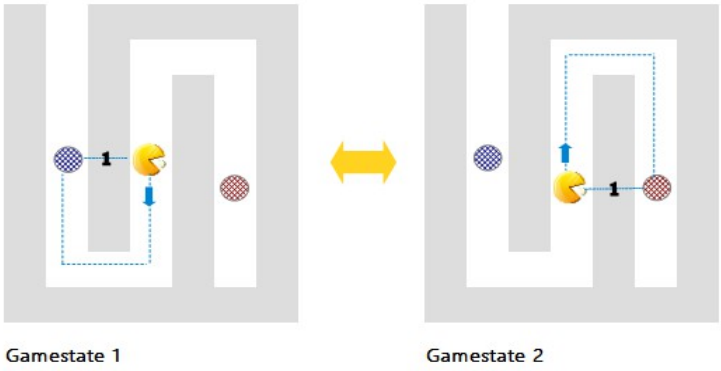


Abbildung 4: Zappelproblematik - Ursache

1. Im Gamestate 1 ist die blaue Pille die am nächsten gelegene. Dementsprechend geht Pac-Man eine Position nach unten.
2. Im Gamestate 2 ist die rote Pille die am nächsten gelegene. Hier geht Pac-Man wieder eine Position nach oben.

3. Das ganze würde sich endlos wiederholen, bis Pac-Man von einem Ghost aufgefressen wird.

Liegt dieser Zustand vor, wird er mit der Konsolenmeldung „Zappelalarm " protokolliert. Zusätzlich wird eine der beiden Zappelpositionen als möglicher Labyrinth-Wege-Punkt kurzfristig geblockt, damit Pac-Man diese „Zappelposition " verlassen kann.

2.3 Herleitung zu Minimax

Der *_Smart-Reflex-Agent* basiert im Wesentlichen auf der Nutzung der Informationen über den aktuellen Spacestate und wählt die nächste Aktion auf Basis dieser Informationen aus.
Im Kapitel 4 werden mögliche zukünftige Züge - sowohl von Pac-Man als auch von den Ghosts berechnet und mit einer Evaluierungsfunktion bewertet, um die Entscheidung für den nächsten Zug zu erhalten.

3. Adversariale Suche

Adversarial Search schließt an die Idee an, zukünftige Ergebnisse in einem Modell zu halten. Zusätzlich zu einer vorausschauenden Single – Agent- Umgebung wird beim adversarialen Suchen die Handlung von „gegnerischen " Agenten mitbetrachtet.

Diese Situation liegt im wesentlichen bei Spielen vor. Es wird also ein Algorithmus benötigt, der sich in einer konkreten Spielsituation für eine passende Handlung entscheidet. Neben der Handlungen des einen Agenten werden auch die Handlungen der gegnerischen Agenten in Hinblick auf das Ergebnis beachtet.

3.1 Single-Agent-Tree

Mit Hilfe des *Single Agent-Tree* wird kurz die Vorgehensweise des Maximum-Prinzip und des Wertepropagierens erläutert, in diesem Fall nur für einen Spieler ohne Gegner. Diese Vorgehensweise ist dann die Basis für den Mini-Max-Search-Tree, der im folgenden Kapitel erläutert wird.

Auch in diesen Gamesituationen, wie z.B. bei Pac-Man, wird mit Bäumen gearbeitet. Die einzelnen Knoten stellen wiederum einen Space -State, der hier als Game-State bezeichnet wird, dar. Ausgehend von der Wurzel, die den Ausgangs-Game-State beinhaltet, wird der Game-Tree analog dem Search-Tree konstruiert.

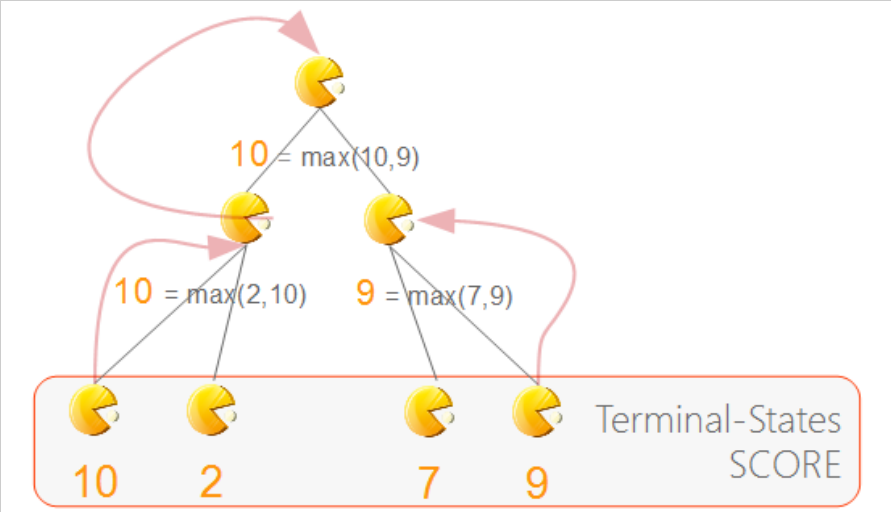


Abbildung 5: Single - Tree - Agent: Maximum der Terminalstates werden zur Wurzel propagiert

Die abhängigen Knoten entsprechen dem Game-State nach Ausführung einer Aktion, die bei Pac-Man eine Bewegung nach oben, unten, links oder rechts entspricht .

Sind die Blätter des Baumes erreicht, liegt ein Terminal-State, also der Spielzustand am Ende des Spieles vor und das Spiel ist beendet. Jetzt kann dem Terminal-State ein Wert zugeordnet werden, nämlich der aktuelle Score bei Spielende. Dieser Wert wird nach dem Maximum-Prinzip zu den höherliegenden Knoten propagiert. Durch diese Methode ist es möglich, auch Game- States, die innerhalb des Baumes liegen oder sogar die Wurzel bilden, einen Wert zuzuordnen. So kann der künstliche Agent von der Wurzel aus zu den Knoten verzweigen, die am höchsten bewertet sind.

3.2 Adversarial-Game-Tree / Mini-Max-Search-Tree

Die Idee des Single-Agent-Trees wird für für das adversariale Suchproblem erweitert. Im *Adversarial – Game-Tree* oder auch *Mini-Max-Search-Tree* bildet die Wurzel den aktuellen Game-State, dann wechseln sich die Game-State nach der Aktion des einen Agenten mit den Game-States nach der Aktion des anderen Agenten (gegnerischen Agenten) ab.

Die Wertzuordnung für innenliegende Knoten funktioniert nach dem gleichen Prinzip wie beim Single-Agent-Tree. Ausgehend von den Terminal-States werden die Werte nach oben propagiert. Während die Game-States, die unter der Kontrolle unseres Agenten stehen, nach dem Maximum-Prinzip aus den Werten der Childs bestimmt werden, werden die Werte der Gamestates aus der gegnerischen Kontrolle nach dem Minimum Prinzip ermittelt. Das Minimum wird hier genutzt, da davon auszugehen ist, dass die Gegnerischen Spieler den für sie optimalen, also für Pac-Man am schädlichsten Spielzug vornehmen.

Mit Hilfe des Mini-Max-Search-Trees kann man einem innerhalb des Baums liegenenden Game-State einen Wert zuordnen.

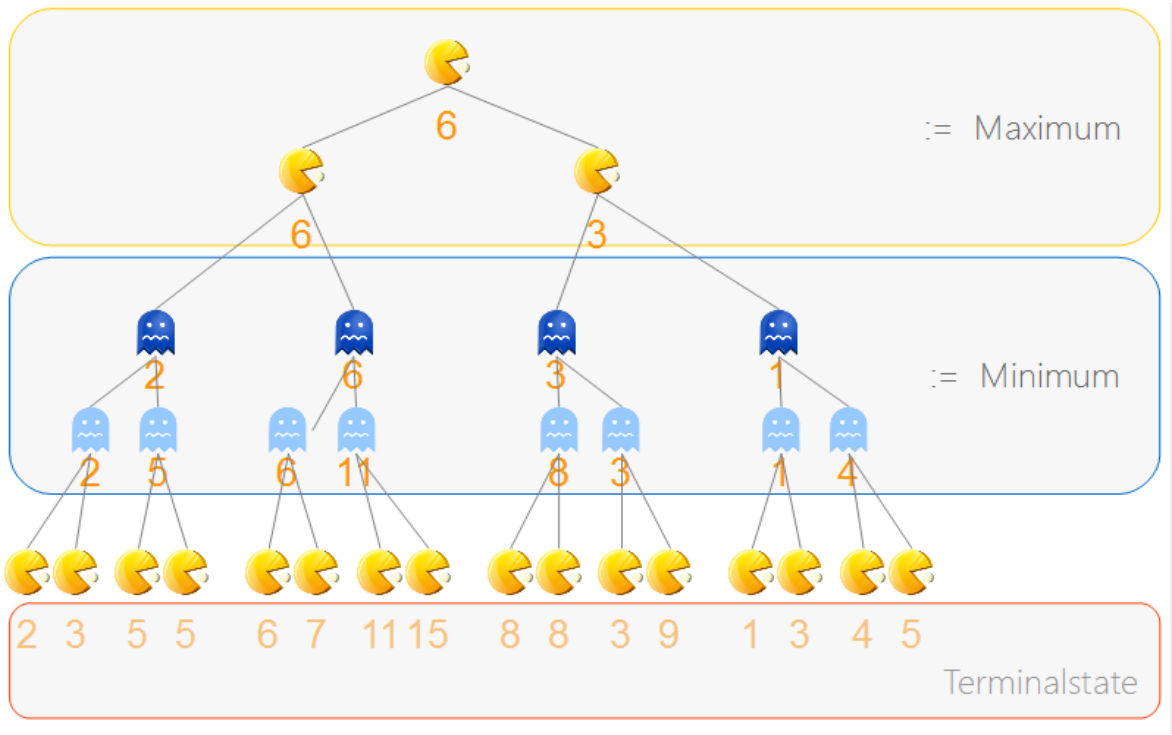


Abbildung 6: Mini-Max-Search-Tree

Auch wenn durch die ebenenweise Etnwicklung des Baumes Spielzüge suggeriert werden, die nacheinander stattfinden, können die diese Züge auch gleichzeitig stattfinden und werden nur schrittweise im Baumdargestellt.

In der dargestellten Version des Mini-Max-Searchs wird die Existenz von Terminal-States vorausgegsetzt, bevor die Werte nach dem Mini-Max-Prinzip nach oben progabiert werden.

3.3 depth limited-Search

Aus Resourcengründen ist es aber bei nahezu allen Spielen nicht möglich, einen terminierenden Game-Tree zu konstruieren.

Im *Depth-limited-Search* wird aus diesem Grund der Suchbaum auf eine bestimmte Tiefe limitiert, d.h. die Blätter des Baumes zeigen die Game-States aus einer Spielsituation mitten im Spielverlauf. Hier liegen nun keine Terminal-States in den Blättern vor und entsprechend können keine Scores ausgelesen iund als Utility-Wert durch den Baum bis zur Wurzel propagiert werden. Eine Evaluierungsfunktion ersetzt den Utility-Wert aus den Blättern.

Die Evaluierungsfunktion wird auf die Konten der untersten Ebene des gekürzent Baumes angewendet und deren Ergebnisse dann nach oben entsprechend dem Mini-Max-Prinzip nach oben hin zur Wurzel propagiert.

Der Evaluierungsfunktion kommt eine ganz besondere Bedeutung zu.

Während der Algorithmus für einen Search-Tree eher universell einzusetzen ist, muss die Evaluierungsfunktion sehr genau auf die Problemstellung angepasst sein. Die Argumente der Funktion leiten sich aus den Werten und Koordinaten des Game-States ab. Der Funktionswert ist um so günstiger, je höher das Ergebnis ist.

4 - Mini-Max Agent mit limited-depth Baumtiefe unter Nutzung einer Evaluierungsfunktion: *Mini-Max-Agent*

Im zweiten von mir implementierten Agent mit dem kurzen Namen Mini-Max-Agent habe ich einen adversarialen Mini-Max-Agenten mit einstellbarer Baumtiefe umgesetzt. Die Besonderheiten dieser Tree Programmierung werden im xxxIT-Dkapitel genauer behandelt. Wichtig ist, dass die im xxx eingepflegte Baumtiefe eigentlich der Anzahl der vorauszuplanenden Spielzüge entspricht. xxt= 3 bedeutet also je 3 Spielzüge von Pac-Man und je drei Spielzuüge von jedem Ghost im Baum darzustellen. So ergibt sich bei einer xxxtiefe von 3, unter dem Einsatz von 3 Ghosts eine Baumtiefe von t_{tree} = 3 + 3* 3 = 12

4.1 Herleitung der Evaluierungsfunktion

 Anwendung in der Pac-Man-Umgebung:

Der Definitionsbereich der Evaluierungsfunktion ergibt sich aus dem Game-State. Sie soll die Elemente des Game-States so bewerten, dass der Game-State in einer ungefährlichen Situation höher bewertet wird als in einer Gefährlichen. Zusätzlich soll die Nähe zu einer Pille oder Powerpille positiv in die Berechnung mit einfließen.
Die Evaluierungsfunktion wird als Summe aus den Komponenten *Pill-Distance-Score*, *Powerpill-Distance-Score* und *Ghost -Distance-Score* gebildet:

1. Der *Pill-Distance-Score* errechnet sich aus dem **Quotienten aus 10** und der **Luftliniendistanz der am nächsten gelegenen Pille:**

   Pill-Distance-Score := 10 / Pill-Distance

   Der Wertebereich liegt zwischen 10 und nahezu 0.

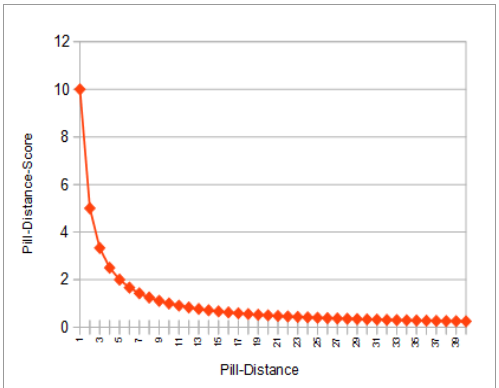


Abbildung 7: Wertebereich des Pill-Distance-Score

2. Der *Powerpill-Distance-Score* errechnet sich wie der *Powerpill-Distance-Score* aus dem **Quotienten aus 10** und der **Luftliniendistanz der am nächsten gelegenen Powerpille:**

   Powerpill-Distance-Score := 10 / Powerpill-Distance

   Der Wertebereich liegt zwischen 10 und nahezu 0.

3. Die dritte Komponente kann ein negatives, oder auch positives Vorzeichen besitzen. Der *Ghost-Distance-Score* errechnet sich in Abhängigkeit, ob Pac-Man eine Powerpille gefressen hat oder nicht. Der Score wird besonders negativ, wenn er sich in der akuten Gefährdungsumgebung (Abstand zu Pac-Man <= 3) befindet. Befindet sich Pac-Man im Powerpill-Modus, so wird der Score besonders postiv in der nähe der Ghosts. **Der Score wird für jeden Ghost ermittlet und aufaddiert.**

   (a) Pac-Man befindet sich nicht im Powerpillmodus:

   Wenn **Luftliniendistanz <= 3,** dann *Ghost-Distance-Score*$_i$:= -20 / Ghost-Distance^3
   Wenn **Luftliniendistanz > 3**, dann *Ghost-Distance-Score*$_i$:= -5 / Ghost-Distance

   (b) Pac-Man befindet sich im Powerpillmodus:
   Wenn Pac-Man die gleiche Position besitzt wie der Ghost, dann *Ghost-Distance-Score*$_i$:= +100

   Wenn **Luftliniendistanz <= 3,** dann *Ghost-Distance-Score*$_i$:= +50 / Ghost-Distance^3
   Wenn **Luftliniendistanz > 3**, dann *Ghost-Distance-Score*$_i$:= +20 / Ghost-Distance

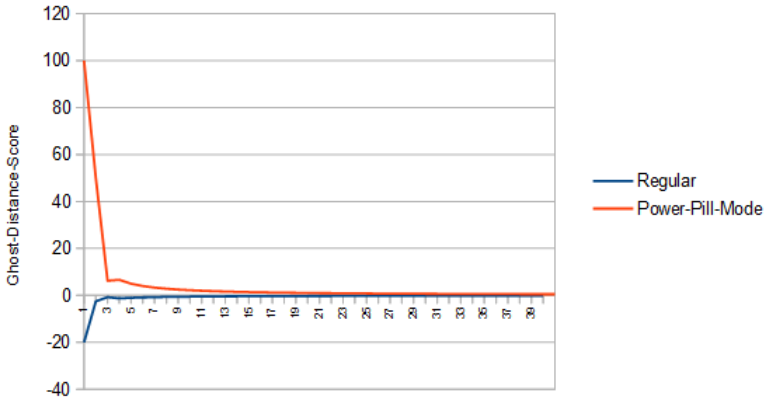


Abbildung 8: Wertebereich des Ghost-Distance-Score

4. Die vierte Komponente, der *Pill-Score* , wird auf 6 gesetzt, wenn Pac-Man in diesem Zug eine Pille auffressen kann.
5. Die fünfte Komponente, der *Power-Pill-Score*, wird auf 10 gesetzt, wenn Pac-Man in diesem Zug eine Power-Pille auffressen kann.

Der Gesamtwert der Evaluierungsfunktion für einen vorliegenden Gamestate ermittelt also wie folgt:

Eval := Pill-Distance-Score + Powerpill-Distance-Score + Ghost-Distance-Score + Pill-Score + Powerpill-Score

4.2 Anwendung im Pac-Man Reverse

Im Pac-Man-Reverse kann über das AI-Cockpit der xxx Agent ausgewählt werden. Im folgenden Menue, dem xxxxx Menue kann dann noch die Schritttiefe xxxx eingestellt werden. Der Agent bildet nun den Mini-Max-Tree für die angegebene Anzahl von Spielschritten und Ghosts. Ist die gewünschte Baumtiefe erreicht, wird die Evaluierungsfunktion aus 4.1 auf die Blätter angewandt und die Werte bis hin zur Wurzel unter der Anwendung des Mini-Max-Prinzips für Pac-Man und Ghost propagiert. Die jeweils ermittelten Werte werden als Konsolenoutput Utility xxx.xxx protokolliert. Haben die propagierten Werte die Wurzel erreicht, kann der Agent die Entscheidung für den nächsten Schritt treffen, der wiederum im Cockpit durch die Pfeile dargestellt wird. Der Verlauf der Evaluierungs- oder Utilityfunktion – lässt sich aus dem Limited-Depth-Mini-Max-Agentxxx -Menuet heraus betrachten, wenn im AI-Cockpit die Min_Max_KI gewählt wurde. Hier kann die mit „Show Graph " die einzelnen Komponenten sowie der Gesamtwert der Evaluierungsfunktion in Echtzeit verfolgt werden.
Wird die Ausführung des Agenten pausiert, kann zusätzlich kann der aktuelle Game-Tree mit den danzugehörigen Werten und show Treexxx dargestellt werden.

Der Game-State, der die Knoten des Baumes bildet, besteht aus den folgenden Elementen:

1. Game-Grid: Spielfeld mit den Koordinaten von Pillen, Power-Pillen, Ghosts und Pac-Man
2. allen Komponenten der Evaluierungsfunktion sowie dem Wert der Evaluierungsfunktion, wenn es sich um einen Knoten ein Baumblatt darstellt, bzw. dem propagierten Wert der Evaluierungsfunktion.

4.3 Vorgehen, besondere Eigenschaften der Logik - (Zappeln)

Auch in dieser KI entstehen Zustände, in denen Pac-Man ins „Zappeln " gerät. Der Wert der Evaluierungsfunktion ist dann wechselseitig immer in der komplementären Richtung am höchsten. So würde Pac-Man unendlich lange zwischen den zwei Positionen hin und her „zappeln ".
Die Situation kann nur aufgelöst werden, in dem sich ein Ghost nähert, der die Evaluierungs -funktionen so abändert, dass Pac-Man aus dieser Situation befreit wird – oder indem der Pill-Distance-Score für eine Richtung, die nicht der komplementären Richtung entspricht, hochgesetzt wird .

Ai Name	Absolute Score	Min	Max	Average Score	Games	Absolute Playt...	Average Playti...	Wins	Looses	ID AI
null	0	0	0	0	0	0	0	0	0	0
null	0	0	0	0	0	0	0	0	0	0
null	0	0	0	0	0	0	0	0	0	0
Regel	6377	24	101	63	100	107000	1070	99	1	4
Min Max	9602	37	251	181	100	1160000	21886	69	31	5
Random	1412	4	26	14	100	4000	40	1	99	6

5. Vergleich der beiden Agenten

Im folgenden werden die beiden Agenten in Hinblick auf bestimmte Kennzahlen verglichen. Im Anschluss werden die Ergebnisse des Vergleiches mit Hilfe des jeweiligen Konzeptes des Agenten erklärt. Die Erkenntnisse hieraus werden im Kapitel 6xx finalisiert und weitere Ideen zur Weiterentwicklung vorgestellt.

5.1 Kennzahlen für den Vergleich inkl. Historie

- Als Kennzahlen dienen zum einen die Scores: hier wird der kumulierte absolute Wert aller Spielausführungen, als auch das Minimum und Maximum, sowie der Averagescore protokolliert.
- Eine andere Kennzahl ist die Spieldauer. Auch hier gibt es den absoluten Wert und die Averagespielzeit. (xxxx).
- Winnings - Defeats Natürlich darf die bloße Zählung der Gewinne/ Verluste auch nicht fehlen.

5.2 Ermittlung der Kennzahlen – unter Anwendung der KI-Ghosts des Gamegrids

Um einen Vergleich der Kennzahlen zu ermöglichen, muss die Erfassung der Statistik für alle zu vergleichenden Agenten unter gleichen Rahmenbedingunen stattfinden. Aus diesem Grund habe ich mich für das Level xx, mit zwei Ghosts die durch den Agenten gesteuert werden xxx entschieden. Die hier vorgenommene Ermittlung der Kennzahlen hat für den Smart-Reflex-Agenten, dem Mini-Max-Agenten und dem „Random-Agenten " stattgefunden
Mit Hilfe einer Batchschleife habe ich die Agenten jeweils 100 mal starten- und entsprechende Statistiken schreiben lassen

.5.3 Vergleich nach Kennzahlen

Dieser Screenshot zeigt die Kennzahlen für die drei verschieden Agenten: Bei einem Agenten handelt es sich um einen Agenten, der nach dem „Random " - Prinzip die Entscheidung für eine Laufrichtung fällt, dieser soll vor allem aufzeigen, welchen Verlauf das Spiel ohne artificial Intelligence nehmen würde. Zum weiteren Vergleich sind die zwei zuvor vorgestellten Agenten, der Smart-Reflex-Agent und Mini-Max-Agent mit seinen Kennzahlen erfasst.

Ai Name	Absolute Score	Min	Max	Average Score	Games	Absolute Playt...	Average Playti...	Wins	Looses	ID AI
null	0	0	0	0	0	0	0	0	0	0
null	0	0	0	0	0	0	0	0	0	0
null	0	0	0	0	0	0	0	0	0	0
Regel	6377	24	101	63	100	107000	1070	99	1	4
Min Max	9602	37	251	181	100	1160000	21886	69	31	5
Random	1412	4	26	14	100	4000	40	1	99	6

Ranking der Gewinnrate
① *Smart-Reflex-Agent* 99%
② *Mini-Max-Agent* 79 %.
③ *Random-Agent* 1 %

Der Smart-Reflex-Agent liegt deutlich vor dem Mini-Max-Agenten – während der Random-Agent eindeutig keine Chance gegen die zwei Ghosts besitzt. Damit ist die positive Funktionsweise der zwei künstlichen Agenten dargestellt.

Das **Ranking** der durchschnittlich erzielten Scores korrespondiert mit dem **Ranking** der Spieldauer: Je länger gespielt wird, desto größer ist der Score.

① *Mini-Max-Agent* **96** Avg. Score - 11.600 millxx Playtime
② *Smart-Reflex-Agent* **63** Avg. Score - 1.070 millxxx Playtime
③ *Random-Agent* **14** Avg. Score - 40 milxx Playtime

Passend zur Spieldauer liegt das größte Maximum beim Mini-Max-Agenten und das Minimum beim Random-Agenten.

5.4 Vergleich im Problemlösungsverhalten

Je nach dem, auf welche Kennzahl der Fokus gesetzt wird, performt der Smart-Reflex-Agent in Hinblick auf die gewonnenen Spiel am besten, während der Mini-Max-Agent bei größerer

durchschnittlicher Spieldauer auch den größeren Score aufweist, aber nicht ganz so häufig das Spiel gewinnt.
Dieses schnelle Gewinnverhalten des Smart-Reflex-Agenten liegt daran, dass er die Siege über das Auffressen der Ghosts während des Powerpill-Modus erreicht.
Ist dieser Modus aktiviert, wird das Ghost-Avoiding ausgeschaltet und Pac-Man kann die sich in der Nähe befindenden Ghosts auf dem Weg zu weiteren normalen Pillen eliminieren.
Zusätzlich ist das Ghost-Avoiding über den eingestellten Toleranzbereich von aktuell 3 sehr effektiv.

Die höheren Scores des Mini-Max-Agenten lassen sich aus dem Verlauf der Evaluierungsfunktion herleiten. Da mit der jetztigen Konzeption der Evaluierungsfunktion immer ein „Kompromiss " aus den Anforderungen Powerpillen-Suche, Pilllen-Suche, Ghost-Avoiding oder Ghost-Hunting ist, sind die Entscheidungen nach der Anwendung der Mini-Max-Vorgehens nicht so eindeutig, um den Sieg durch Ghost-Hunting herbeizuführen. Die Überlebensstrategie des Ghosts-Avoidings scheint gut zu funktionieren, denn Pac-Man überlebt lange und hat so in dieser Zeit die Möglichkeit, den Score durch Pillenfressen zu erhöhen.

6 Fazit

Aufgabenstellung dieser besonderen Lernleistung war es, einige Konzepte der künstlichen Intelligenz am Beispiel Pac-Man durch die Pac-Man-Reverse Anwendung aufzuzeigen. Ausgehend von der Definition der künstlichen Intelligenz habe ich mich auf „Systeme, die rational handeln " spezialisiert und intelligente Agenten bzw. Agentenprogramme und deren Grundtypen kennengelernt. Aus diesen Grundtypen habe ich den Reflexagenten Smart-Reflex-Agent unter Nutzung des A*-Searches sowie dem modell- und nutzenbasierten Agenten, als limited-depth Mini-Max-Agent unter Nutzung einer Evaluierungsfunktion entwickelt.
Bei der Entwicklung ergab sich immer wieder die Notwendigkeit, bestimmte Zustände, Entscheidungen, Gewichtungen und Gamestates sichtbar zu machen, um die Handlungen der künstlichen Agenten besser verstehen zu können. So sind die Graphen der Evaluierungsfunktion, der Plot des Gametrees beim Smart-Reflex-Agenten und die Cockpit- sowie Debugfunktionen im Rahmen der Entwicklung hinzugekommen. Mit Hilfe der Statistik-Aufzeichnungen lassen sich die Agenten sehr gut vergleichen.

6.1 Zusammenfassung – Beurteilung der Ergebnisse

Durch den Vergleich der Kennzahlen sieht man sehr gut, dass die Agenten erfolgreich arbeiten. Je nach dem, nach welchem Schwerpunkt sie arbeiten, erzielen sie unterschiedliche Ergebnisse.
Alles zusammen bildet jetzt eine sehr gute Grundlage sogar noch für weitere Entwicklungen

6.2 Möglichkeiten zur Weiterentwicklung

Im Smart-Reflex-Agent wird die am nächsten gelegene Pille mit Hilfe der Luftliniendistanz bestimmt. Ein Versuch, die nach der Wegedistanz am nächsten gelegene Pille über die A*-Suche zu ermitteln musste ich aus Performancegründen und letztendlich aus Zeitkapazitätsgründen abbrechen. Diese Weiterentwicklung des Smart-Reflex-Agent erscheint mir sehr sinnvoll, zudem sich damit auch die Zappelproblematik erübrigen würde, die ja genau aus der Entscheidung aus der nicht kompartiblen Kombination zwischen A*-Search und Luftliniendistanz entsteht.

Für den Mini-Max-Agenten liegt der Fokus der Weiterentwicklungsmöglichkeiten auf der Evaluierungsfunktion. Hier können, je nach gewünschtem Schwerpunkt, die einzelnen Komponenten noch unterschiedlich gewichtet werden. Durch die Visualisierungsmöglichkeiten über die Graphen lässt sich hier eine Weiterentwicklung sehr gut ausführen .

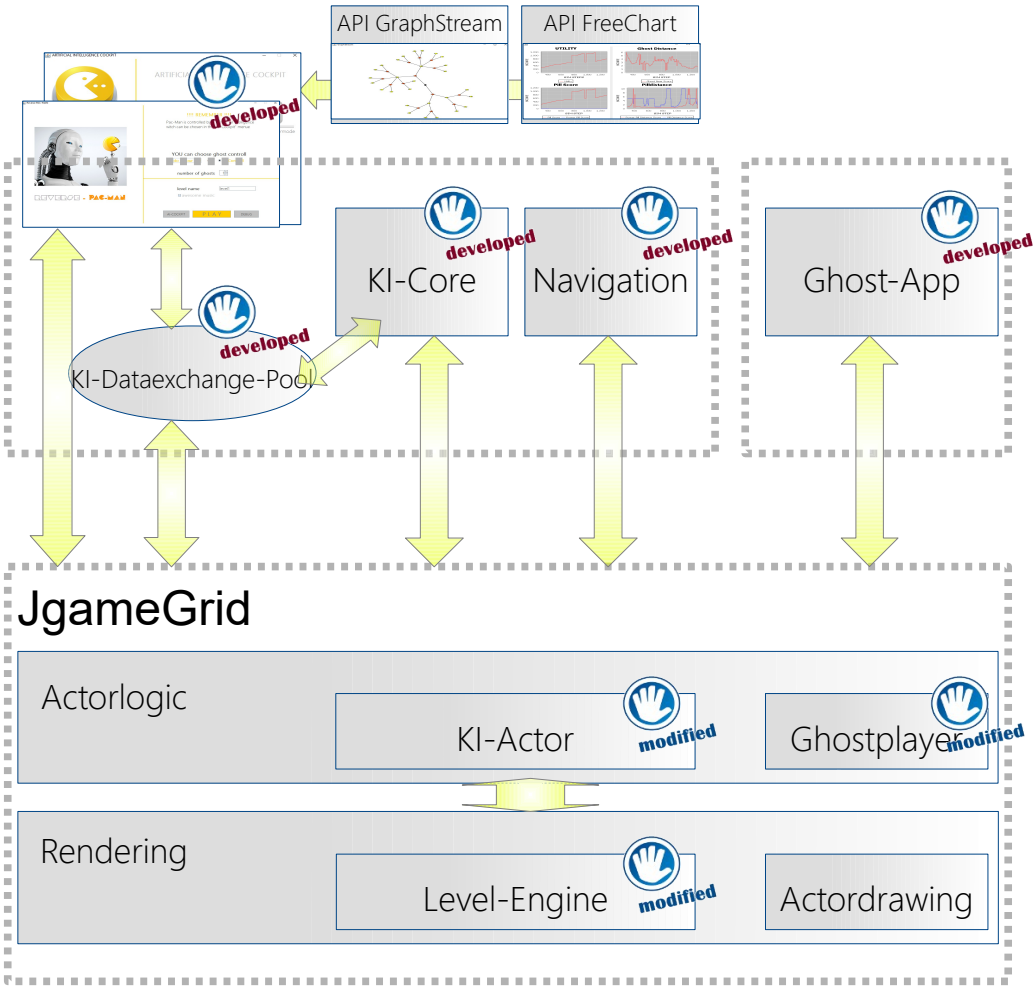
Eine weitere Optimierungsmöglichkeit für den Mini-Max-Agenten ist das Alpha-Beta-Pruning. Es ist eine optimierte Version des Mini-Max-Algorithmus – Mit Hilfe der beiden Werte Alpha und Beta, die während der Suche aktualisiert werden, kann man schließlich entscheiden, bestimmte Teile des Searchtrees nicht zu untersuchen. Durch dieses Verfahren kann der Mini-Max-Agent-Suche in seiner Performance verbessert werden. Auch hier gibt es die Möglichkeit, die Luftliniendistanz durch eine A*-Search Wegedistanzmessung zu ersetzen, wenn eine Systemressourcenfreundliche Lösung zur Verfügung steht.

C – IT-Dokumentation - Realisierung

1. Aufbau der IT-Dokumentation

Der Pac-Man-Reverse Systemkomplex beinhaltet sowohl eigenentwickelte Komponenten als auch Module des JGameGrid-Frameworks – zusätzlich sind zur Visualisierung weitere APIs wie „GraphStream " oder „FreeChart " im Einsatz. Das folgende Chart gibt einen kurzen Überblick über Komponenten und deren Schnittstellen geben. Im Anschluss werden die Komponenten einzeln in Bezug auf deren Nutzung bzw. Programmierung näher dargestellt. Sobald ein Bezug zu einem Java-Objekt herzustellen ist, wird dieses Objekt in der Ausarbeitung in << >> gesetzt Die programmiertechnische Dokumentation erfolgt über Javadoc. xxxx implementiert in java

1.1 Architektur des Reverse-Pac-Man



Zeichnung 2: Reverse-Pac-Man: Komponenten und Schnittstellen

1.2 Zusammenwirken der Komponenten

Das *JGameGrid* ist einer der Hauptkomponenten. Bei diesem Framework handelt es sich um eine Klassenbibliothek für die Entwicklung von Computer-Spielen. Die wichtigsten Funktionalitäten des Gamegrids sind zum einen das *Rendering* und zum anderen die *Actorlogic.* Während das Rendering die Darstellung des Spieles für das Labyrinth (*Levelengine*) und Figuren (*Actordrawing*) ausführt, wird durch die Actorlogic die Steuerung der Aktoren des Pac-Man *(KI-Actor)* oder der Ghosts(*Ghostplayer*) geleistet.

Der KI-Actor ist die relevante Stelle, an der aus dem klassischen Pac-Man der Reverse-Pac-Man entsteht: Denn anstelle die Anweisungen für Pac-Man über die Tastatur zu erhalten, kommen die Befehle aus der weiteren Hauptkomponente, dem *KI-Core*, welcher alle implementierten künstlichen Agenten umfasst. Das Menuecluster nutzt die Verbindung zu den APIs *GraphStream* und *FreeChart* , die zur Darstellung der Mini-Max-Suchbäume sowie des Plots der Utitilty – Funktion und deren Elemente dienen. Der Aktor für die Ghost(Ghostplayer) wird wahlweise über die *Ghost-App* oder über eine kleine im Aktor selbst implementierte KI gesteuert.

Über den *Data-Exchange-Pool* werden Informationen zwischen JGameGrid-Klassen, dem Menue und dem KI-Core ausgetauscht. Die dient zur Routenfindung und Entfernungsbestimmung.

2. Server - Komponenten

2.1 JGameGrid-Framework

Das JGameGrid ist die Gameengine, welche für die Spiele-Architektur verantwortlich ist. Eine wesentliche Funktion ist die **Darstellung** des Labyrinths und die **Fortbewegung der Figuren** in diesem Labyrinth. Es berechnet den aktuellen **Score** und führt in bestimmten Intervallen einen **Reload** der Bildinformation für den neuen Spielstand durch . Damit liefert es die Basisfunktionalitäten, auf die die Pax-Man-Reverse Anwendung aufsetzt.

Zur Einarbeitung in das Gamegrid habe ich den Example-Code für Pac-Man untersucht. Bei dieser Analyse habe ich versucht, herauszufinden, wie z.B. das Spiel gestartet wird und welche Klassen man benötigt, und wie man sie benutzen muss.

Weitere Fragestellungen waren :

1. Wo wird das **Level definiert** ?

   Wie könnte ich eine eigenes **Level** einlesen?

2. Wie können eigene **Gamegrid-Aktoren** hinzugefügt werden ?

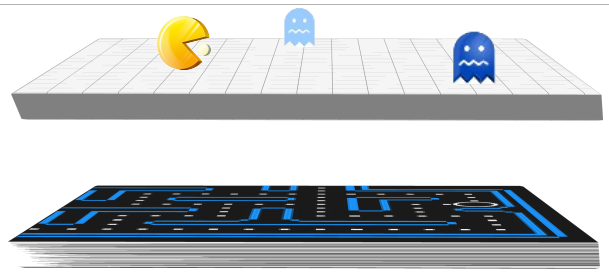
   Wie kann man sie **extern steuern** ?

2.1.1 Modifikation der PacGrid Klasse

Die Definition eines Levels erfolgt im JgameGrid - ExampleCode in der <<PacGrid>> Klasse. Diese ist im Original hart kodiert, es handelt sich also um eine statische Levelarchitektur.

Durch die Hartkodierung war keine flexible Levelgestaltung möglich. Da ich Variationen der in der Levelbildung ermöglichen wollte, habe ich diese Klasse modifiziert.
Desweiteren habe ich bei der Analyse des PacGrids die Speicherung und die Displayausgabe der Level- und Aktoren (Pac-Man) betrachtet. Im JGameGrid werden die Positionen der Aktoren über Tabellen in den Dimensionen des Spielfeldes realisiert, sodass bei der Bewegung der Figuren diese Tabelle auf den Positionen entsprechend aktualisiert wird.

Die Bildschirmdarstellung des Labyrinthes incl. Pillen, Power-Pillen sowie Aktoren wird durch einen Plot auf dem Bildschirm erzeugt.
Auch hier habe ich eine neue Lösung entwickelt Neben dem sogenannten Layer 1, einer Tabelle mit Positionen für Pac-Man und Ghosts, habe ich eine weitere Ebene, das Layer 2 als Array definiert. In diesem Array <<maze>> sind die Blöcke des Labyrinthes und auch die Pillen und Power-Pillen abgelegt. Diese Datenstruktur ermöglicht die einfache Bestimmung von Koordinaten durch die Algorithmen der KI , die z.B. zur Kalkulation von Distanzen über A*-Search oder Luftlinie benötigt werden.



Layer 1: (innerhalb des GameGrids)
Positionen der Aktoren

Layer 2: <<maze>>
Labyrinth und Positionen der Pillen/Power-Pillen

Zeichnung 3: Darstellung der Zwei-Layer-Architektur nach den Modifikationen in der <<PacGrid>> Klasse

Nach der Modifikation im <<PacGrid>> wird das Level-Design wie folgt durch geführt:
Mit Hilfe einer Level-Datei wird das individuelle Leveldesign eingelesen. Diese Datei wird unter pacman.jar\level*levelname.xxx* gespeichert. In der Datei sind verschiedene ASCII-Zeichen als Identifier für bestimmte Labyrinth-Blöcke abgelegt. Diese Datei wird nun eingelesen und die einzelnen Zeichen der Inputdatei werden in Blockobjekte zur Bildschirmanzeige umgewandelt und auch im Layer 2: <<maze>> abgelegt. Die Blockobjekte sind von mir als png-Texturen erstellt worden und befinden sich ebenfalls im Projektordner. Anschließend werden sie vom Renderer des JGameGrids auf den Bildschirm gebracht.
Um Pac-Man und die Ghosts auf die richtigen Startpositionen zu setzen, befinden sich neben den Dimensionen des Labyrinthes auch die Koordinaten der Aktoren in der Datei *levelname*awesome.xxx.

2.1.2 Neuentwicklung des KI-Actors

Ein Aktor ist im Gamegrid ein Objekt (Spielfigur), welches sich auf dem Labyrinth bewegen kann. Nach der obigen Layer-Architektur bewegt er sich also im Level 1. Der dem Pac-Man zugeordnete <<Pac-Actor>> wurde durch den <<KIActor>> von mir ersetzt und die Befehle für Pac-Man werden jetzt nicht mehr durch die Tastatur empfangen. Der << KIActor>> besitzt durch das <<aiinterface>> eine Verbindung zum KI-Core, der die künstlichen Agenten steuert und mit dem KIActor interagiert. Dabei übergibt der <<KIActor>> den aktuellen Gamestate. Der gewünschte, künstlichen Agent berechnet den nächsten Befehl für Pac-Man, wie z.B. „oben ", „unten ", „links " oder „rechts " . Dieser Befehl wird dann an den Aktor zurückgespielt, im Gamegrid ausführt und auf den Bildschirm gebracht.

2.1.3 Modifikation des Ghost-Aktors

Der <<Ghostplayer>> ist der Aktor für die Ghosts. Er besitzt die gleichen Befehle wie Pac-Man: „oben ", „unten ", „links " oder „rechts ", nur ist die KI zur Steuerung im Aktor selbst implementiert. Wenn die Ghost-Steuerung per App gewählt wurde, wird die <<TCPBridge>> benutzt und diese verbindet sich zur App oder PC-Fernbedienung der Ghosts. Die unidirektionale <<TCPBridge>> ist dafür verantwortlich, dass sich die Apps auf den Ghostplayer verbinden können und dass die Nachrichten von der App zum Aktor kommen.
Die genauere Dokumentation der beiden Interfaces ist im JavaDoc und in der Schnittstellen-dokumentation nachzulesen.

2.2 Das Menuecluster

Beim Menuecluster handelt es sich um einen losen Zusammenschluss von verschiedenen Jframes, die die Menue-Hierarchie des Reverse-Pac-Man bilden. In den verschiedene Menues werden die Parameter zur Steuerung erfasst und in den KI-Data-Exchange-Pool (<<KIData>>) geschrieben.
Das JFrame des Main-Menue ist dafür verantwortlich, das Spiel zu starten und erste Spielparameter zu erfassen und an den KI-Data-Exchange-Pool zu übergeben. Bei Aktivierung des Buttons „PLAY " wird das AI-Interface des JgameGrid gestartet.
Das Debug-Menue setzt Debugging-Parameter, welche vor allem durch die Navigation genutzt werden. Im Mini-Max-Menue werden die Parameter wie z.B. die Baumtiefe für den Minmax-Agenten festgelegt und die visuellen Baum- und Utility-Darstellungen über die Graphstream und FreeChart aktiviert.

Das KI-Cockpit greift auf die Daten des KI-Data-Exchange-Pools zu Verarbeitung der aktuellen KI-Situation zu. Auch werden KI und Tempo des Spiels in Form der Dauer eines Spielschrittes in Millisekunden durch den User festgelegt.

2.3 Die Navigationsklasse

Die Klasse <<navigation>> stellt eine Art Serviceeinheit für die KI dar, die benötigt wird, um Routen und Entfernungen zwischen zwei Punkten im Grid zu kalkulieren.

2.3.1 Initialisierung der Navigationsklasse

Zur Initialisierung wird nach Start des Spieles aus dem durch das Layer2 im <<maze>> dargestellten Labyrinth eine logische Repräsentation von möglichen Gehwegen für Pac-Man erstellt. Als Erstes werden **alle begehbaren Punkte bestimmt**.. Dann wird für jeden Punkt untersucht, ob er eine Node darstellt. Dies wird durch einen Patternsearch realisiert. Dieser überprüft, ob es sich bei dem Wegpunkt, also um einen Punkt innerhalb einer geraden Strecke handelt. Falls dieses Kriterium nicht zutrifft, muss es sich um eine Node handeln. Die **Nodes werden** der Reihe nach **ermittelt** und durchnummeriert und in eine ArrayListe mit Elementen der Struktur <<node>> geschrieben.
In einer <<node>> stehen seine Koordinaten, und <<way>> Einträge (Zielnodeinformationen und die Länge des Weges dorthin) für benachbarte Nodes.


Über das Debug-Menue des Reverse – Pac – Man lassen sich die Nodes mit Koordinaten und Namen anzeigen .

Im nächsten Schritt werden **Verbindungen zwischen den Nodes bestimmt**. Bei Verbindungen handelt es sich um freie Wege, um von einem Node zum anderen zu kommen. Diese Verknüpfung von Nodes ist eine wichtige Voraussetzung für den A*-Search und **bildet** aus den einzelnen Nodes ein **Netzwerk**.
Die Funktion <<connectnode>> überprüft für einen Node, ob er einen Partner in Nord-, Ost-, Süd- oder Westrichtung besitzt. Die Indices des entsprechenden Arrays laufen von 0 bis 3. Ein Partnernode liegt vor, wenn er auf „geradem Weg " vom Ausgangsnode erreichbar ist. Der gefundene Zielnode wird bei dem Ausgangsnode zusammen mit der Entfernung als <<way>> abgelegt. Von jedem Ausgangsnode wird dieses Vorgehen für alle Richtungen, bzw. Indizes für die Himmelsrichtungen ausgeführt.
Diese Darstellung des Labyrinths bildet somit einen weiteren Layer. Dieser Layer3 zeigt neben der Anfangs rein graphischen Ablage des Labyrinthes aus Layer2 nun logische Verknüpfungen zwischen den einzelnen Nodes zu einem Netz.

Neben der Erstellung des Nodenetzes gibt es noch zwei wichtige Methoden, die von den Agenten und auch vom A*Search genutzt werden.

<<simplelength>> berechnet die Luftliniendistanz zwischen zwei Punkten über den Satz von Pythagoras

<<simplereach>> gibt zurück , ob zwei Punkte auf „geradem " Weg, also ohne Abzweigungen verbunden sind.

2.3.2 Umsetzung des A*Searches

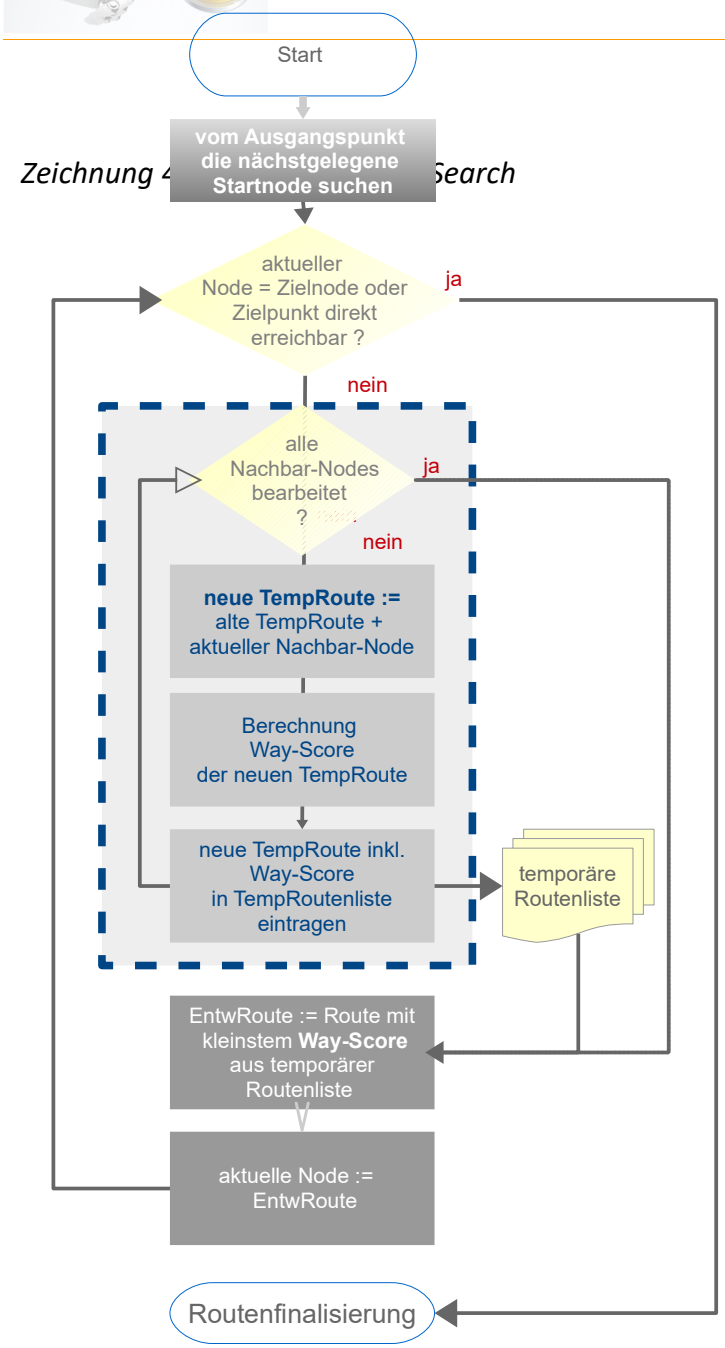
Der A*Search dient der Ermittlung einer Route zwischen zwei Punkten und bildet den ersten von mir entwickelten Algorithmus zur künstlichen Intelligenz in Form einer „informierten Suche “. Er wird in verschiedenen Situationen von den Agenten des KI-Core aufgerufen.

1. Inputparmeter: Start- und Ziellokalion in Form von Koordinaten – diese können jetzt auch Wegpunkte sein, die keine Nodes darstellen.
2. Output:: <<Route>> :
   - Koordinaten von Start und Ziel
   - wayscore: Heuristik des A*Searches: Score aus der bereits zurückgelegten Strecke und der Luftlinie der restlichen Distanz
   - Final-Route als Liste von <<way>> Objekten (Zielnode, Entfernung)
   - Koordinaten liste als Liste von Koordinaten der Routenpunkte der finalen Route (Diese Punkte werden durch die KI's zum tatsächlichen ablaufen der Route genutzt)

(1) Zu Beginn wird ein initiales Routenobjekt generiert

(2) Bevor der A*Search gestartet wird, prüfe ich aus Perfomancegründen, ob nicht eine direkte, gerade Verbindung vorliegt (<<simplereach>>).
Ist dies der Fall, wird der initialen Route ein neues <<way>> Objekt zu gefügt und die Route finalisiert. und zurückgegeben.

(3) Jetzt startet der A*Search, wie im Ablaufdiagramm beschrieben.



Zeichnung 4 ...Search

Bedingung *„aktueller Node = Zielnode oder Zielpunkt direkt erreichbar ":*

Hiermit wird die Zielerreichung überprüft. Diese liegt entweder vor, wenn der Zielpunkt gleichzeitig auch ein Node darstellt. Ansonsten wird ausgehend vom aktuellen Node der Zielpunkt mit Hilfe von <<simplereach>>erreicht werden kann

Der Wayscore errechnet sich aus dem zweifachen der tatsächlich zurückgelegten Strecke plus der noch ausstehenden Strecke als Luftlinie. Diese Heuristik wurde gewählt, um die Priorität auf der kürzesten, bereits zurückgelegten Strecke zu legen.

Bei der Finalisierung wird die EntwRoute in die Final-Route kopiert und die dazugehörigen reinen Wegpunkte in die <<Chordlist>> geschrieben.

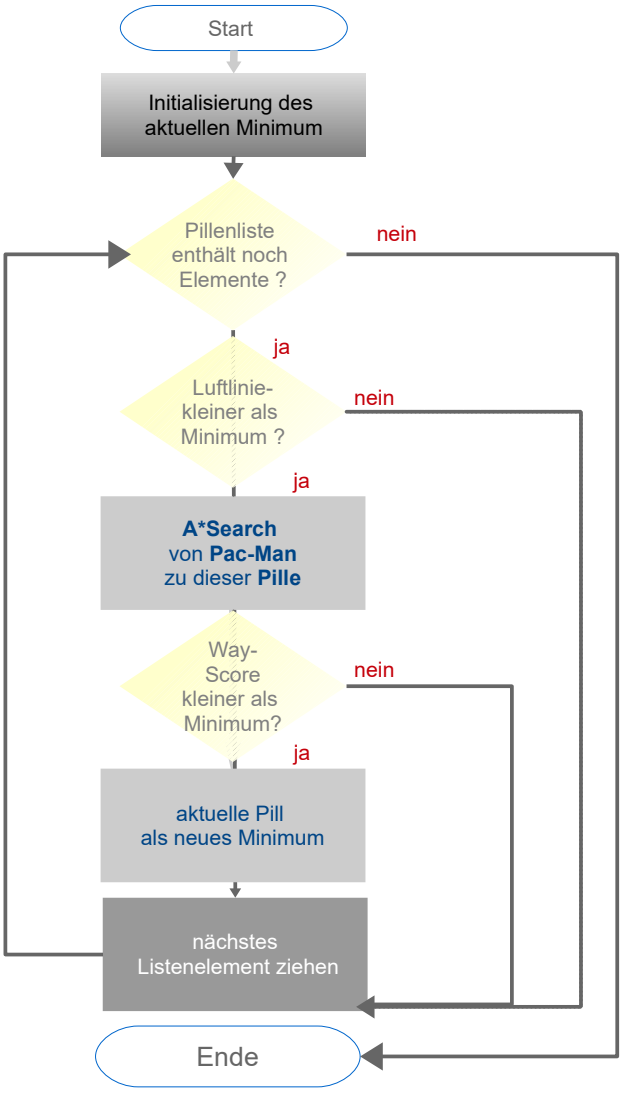
(4) So kann die Navigation zusammen mit der finalen Route und der Liste der Koordinaten der abzulaufenden Wegpunkte die gewünschten Schritte mit Hilfe der Methode <<walkroute>> für Pac-Man ausführen.

Der künstliche Agent „simple_find_pills " führt einen A*Search durch, um alle Pillen des Labyrinthes zu verzehren. Bei der Ausführung dieses Agenten ist es erforderlich, im Main-Menue die Anzahl der Ghosts auf 0 zu setzen, denn dieser Agent ist nicht in der Lage, den Ghosts auszuweichen.

2.4 Der KI-Core und seine Agenten

Die Klasse <<KI>> besteht aus Methoden, die KI – Algorithmen darstellen. Ein Methodenaufruf entspricht einem KI-Berechnungsschritt. Das Ganze wird über <<act>> unter Auswahl der jeweilig gewählten KI gestartet. <<act>> verzweigt dann in den gewählten Agenten.

2.4.1 Die Methode zum Auffinden von Pillen



Zeichnung 5: Ablaufdiagram der find_pill Methode

Die Methode <<findpill>> dient dem Auffinden von noch nicht aufgefressenen Pillen. Die <<findpill>> Implementierung durchsucht die Pillenliste nach einem neuen Minimum in der Luftlinienentfernung zwischen Pac-Man und der Pille. Ist ein solches gefunden, so wird mit Hilfe des A*Searches eine Route dorthin bestimmt und auch dessen Wayscore bestimmt. Ist der Wayscore niedriger als der des aktuellen Minimums, so wird diese Pille als aktuelles Minimum abgelegt.
Das Ganze durchläuft die Liste, bis alle Pillen untersucht worden sind.
Das Ergebnis ist dann die am nächsten gelegene Pille.

Der künstliche Agent „simple_find_pills " das reine Aufspüren und Verzehren von Pillen aus. Bei der Ausführung dieses Agenten ist es erforderlich, im Main-Menue die Anzahl der Ghosts auf 0 zu setzen, denn dieser Agent ist nicht in der Lage, den Ghosts auszuweichen.

Der künstliche Agent „crazy_find_pills " zeigt auf, wie Pac-Man alle Pillen fressen würde, ohne auf die Distanz zu achten. Hier arbeitet er die Pillenliste sequentiell ab. Auch hier ist es erforderlich, im Main-Menue die Anzahl der Ghosts auf 0 zu setzen, denn dieser Agent ist nicht in der Lage, den Ghosts auszuweichen.

2.4.2 Die Methode für einen random gesteuerten Pac-Man

Zur Darstellung des Verhaltens von Pac-Man ohne jegliche KI habe ich diesen Agenten geschrieben. In der Methode <<random>> wird eine Integer-Zufallszahl zwischen 0 und 3 erzeugt. Dann wird in die korrespondierende Richtung 0-Nord – 1 Süd, 2- Ost oder 3 – West verzweigt. xxxxx

2.4.3 Die Methode zum Ghost-Ausweichen

Diese Methode wird zur Erkennung einer Ghost-Bedrohung genutzt. Außerdem ermittelt sie die durch die Bedrohung noch übrigbleibenden Gehrichtungen.

In <<ghostavoid>> wird für jeden Ghost in der Ghost-Liste mit <<simplelength>> geprüft, ob ein Ghostalarm (min. ein Ghost befindet sich in der Toleranzzone) vorliegt.

Bei Ghostalarm:

- werden alle Routen vergessen, und es wird überprüft welche möglichen Geh-Richtungen für aufgrund der Position im Maze möglich sind>> Escaperoutenliste,
- dann ermittelt er die Richtung zum „alarmauslösenden " Ghost (A*Search) und löscht diese aus der Escaperoutenliste.
- Bleibt nur noch eine Richtung übrig, verzweigt Pacman in diese.
- Wenn mehr Auswege vorliegen: dann wird <<findroutetopillwithwhitelist>> aufgerufen (IN: Escaperoutenliste, OUT: Escaperoute), die den Weg zur nächsten Pille berechnet.
- Danach wird der nächste Schritt ausgeführt

2.4.4 Der Smart-Reflex-Agent

Der Smart-Reflex-Agent wird in der Methode <<simpleki>> implementiert und ist im Prinzip eine einfache Schachtelung von <<Ghostavoid>> und <<findpill>> Aufrufen.

Pseudocode des Smart-Reflex-Agenten:

```
IF Powermode = OFF THEN
        <<Ghostavoid>>
        IF Ghostalarm = OFF THEN
          <<findpill>>
         END
ELSE <<findpill>>
END
```

Befindet sich Pac-Man im Power-Mode, so kann er einfach die Pillen suchen und auffressen. Ansonsten muss zunächst Ghostavoid ausgeführt werden. Liegt dann eine Ghostalarm vor, wurde der anstehende Zug zugleich im Ghostavoid ausgeführt, ansonsten muss mit <<findpill>> die nächste Pille angesteuert werden. Arbeitet der Pac-Man im Power-Mode, so kann er gleich sofort mit dem <<findpill>> beginnen.

2.4.5 Der Mini-Max-Agent

Die Klasse <<Gaertner>> beinhaltet das Vorgehen der KI des Mini-Max Agenten und kann in zwei Schritte aufgeteilt werden. Als erstes muss ein Mini-Max-Gametree <<maketree>> generiert werden, auf den dann im zweiten Schritt der Mini-Max-Algorithmus <<minmax>> xxangewandt wird.

Zwei wichtige Objekte im Algorithms sind der

a) der <<Gamestate>>, mit einer Ausprägung des Zustandes des Labyrinthes, der Position von Pac-Man und den Ghosts, sowie der Information über den Power-Mode,

b) der <<Treenode>>, mit Informationen zu dem aktuellen Gamestate, Art der Treenode (ist Pac-MAn oder die Ghosts am Zug ?), sowie Parent- und Childinformationen . Der Treenode bildet den Grundbaustein für den Mini-Max-Gametree.

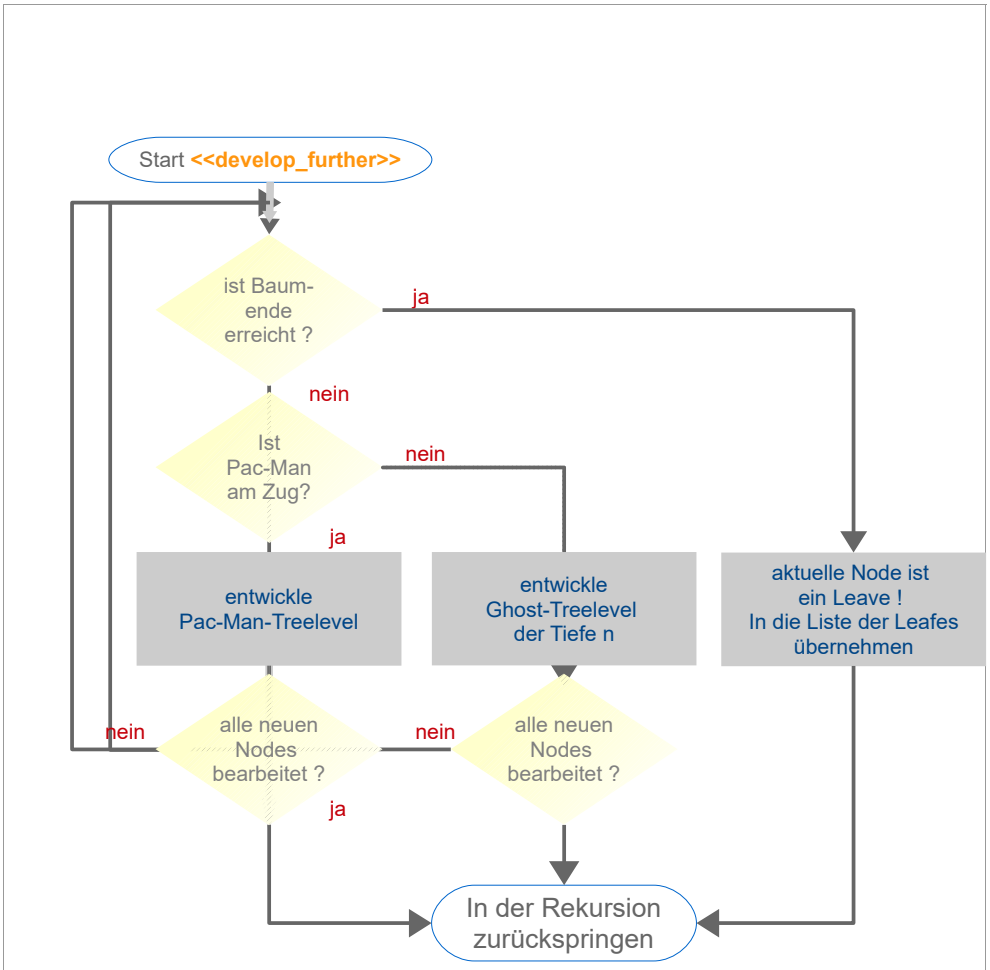
Als weitere wichtige Methode zum Treenode gehört das **Entwickeln** - <<develop>>:
Es bedeutet, alle möglichen **nächsten Spielschritte** ausgehend vom aktuellen Node in Form eines modifizierten Gamestates zu **simulieren** und dann **als Children** an diese aktuelle Node **anzuhängen**.
Insgesamt werden bei der Simulation eines Spielschrittes alle Auswirkungen auf das Spiel in Form von Veränderungen des Scores, der Pillen, Powerpillen, Gewinn/ Verlust , Pillenfressen, Ghost Fressen , Position des Pac-Man oder der Ghosts bestimmt und in der neuen Tree_node abgelegt.
Bei der Entwicklung eines Pack-Man-Zuges wird an der übergebenen Node eine Ebene des Baumes weiterentwickelt (für den Zug des Pac-Man). Sollen die Ghost-Züge entwickelt werden, so wird an der Node, für jeden Ghost über einen rekursiven Aufruf eine Ebene weiterentwickelt, sodass am Ende dann bei n Ghosts n neue Ebenen des Trees erstellt werden.

2.4.5.1 Erstellung des Mini-Max-Search-Trees

Die Aufgabe der tatsächlichen Erstellung eines Gametrees erfolgt durch die Methode <<maketree>>.
Hierbei wird der erste Treenode (Root) zusammen mit dessen Children mit Hilfe von <<develop>> angelegt. Dann wird für jedes dieser angelegten Children die Methode <<develop further>> aufgerufen.

Die Methode <<develop_further>> stellt die rekursive Algorithmik zur Erstellung eines Mini-Max-Trees bereit.

Die Anzahl der zu simulierenden Ghosts sowie die Anzahl der zu simulierenden Spielschritte für Ghosts und Pac-Man sind hier die relevanten Parameter, die die letztendliche Baumtiefe ausmachen.



Zeichnung 6: Ablaufdiagramm rekursive Methode <<develop_further>>

2.4.5.2 Anwendung des Mini-Max-Algorithmus auf den Mini-Max-Search-Tree

Nach der der Ausführung von <<maketree>> liegt nun ein Mini-Max-Searchtree zusammen mit einer Liste von Leaves des gewünschten Gametrees vor. Diese nutzt der Mini-Max-Algorithmus <<minimax>>xx

Der Algorithmus arbeitet sich **ebenenweise von den Leaves bis zur Root** durch den Tree .

Ist die Ebene der Leaves abgearbeitet, so wird die darüberliegende Ebene der Parents zur Leaf-Ebene und die Verarbeitung beginn erneut

Im Detail verläuft die Verarbeitung wie folgt:

❶ Zu Beginn wird die Utility Funktion auf die Leaves angewendet.

❷ Solange Leaves in der Leaf-Liste vorliegen, wird für ein Leaf und seine Geschwister , je nach dem, ob es sich um einen Pac-Man-Zug oder Ghost-Zug handelt, entweder das Maximum bzw. das

Minimum bestimmt und in den Parentnode geschrieben. Zusätzlich wird der Parentnode in einer separaten Liste abgelegt und die Leafs aus der Liste gelöscht.

❸ Ist die Liste der Leafs komplett leer, dann wird die Liste der Parents zur Leaf-Liste und das Ganze startet wieder bei ❷

❹ Handelt es sich in der Verarbeitung bei einem Parentnode um die Root, so ist das Ende des Baumes erreicht und die Entscheidung, in welche Richtung Pac-Man verzweigt, kann getroffen werden.

Pseudocode des <<minimax>>Methode

```
FOR EACH leaf in List DO calculate evaluation_utilty END

UNTIL leaf-List is not empty DO
        IF  leaf's parent IS NOT root
        THEN   copy Parent in parent-list
                IF node is a pac-man-move
                THEN   take maximum of utility out of your brother's nodes
                ELSE    take minimum of utility out of your brother's nodes
                END
                copy utility and it's arguments to parent
                delete leaf and it's brothers from leaf-list
                IF leaf-list is empty
                THEN copy parent-list to leaf-list
                END
        ELSE    take decision on utility-value
        END
```

2.7 Ghost-App

Die App dient zur Steuerung der Ghostspieler, um als Mensch gegen die Pac-Man-KI spielen zu können. Bei der App wird eine andere Form von Java verwendet, die spezifisch für die Android-Umgebung entwickelt wurde. So gibt es z.B. für einzelne Screens und Menues keine speziellen Frames, die mit dem Windowbuilder erstellt werden könnten. Stattdessen wird die Android-View Architektur eingesetzt, welche auf XML basiert. Die einzelnen Views können mit Hilfe des Android-internen Intend angezeigt und gewechselt werden.

Buttenpress-Events werden ebenfalls aufgrund des Fehlens von JFrames anders realisiert – da es keine Actionhandler gibt. Diese werden über das Java eigene Id-System geregelt.

Um vom Smartphone aus die Ghosts auf dem PC steuern zu können, ist eine Verbindung zwischen dem Smartphone und der Reverse – Pac – Man Applikation auf dem PC notwendig.

Dabei funktioniert das Java-Socket-Prinzip nicht. Denn jedes einzelne Button-Click-Event wird von dem Mainthreat gemanagt und so werden Netzwerkoperationen durch die aufkommenden Wartezeiten auf dem Mainthreat nicht zugelassen.
Meine erste Idee war, einen separaten Threat für die Netzwerkoperationen bereitzustellen, was jedoch durch Android nicht zugelassen wurde. Nach ausführlicher Recherche habe ich den Einsatz von asynchronen Tasks getestet. Asynchrone Task unterscheiden sich von Threats durch die Möglichkeit, während der Laufzeit Statusupdates zurückzugeben.
Mit Hilfe der asynchronen Tasks konnte so eine Verbindung zwischen PC und App hergestellt werden. Die App besitzt zwei sogenannte Views :

View I ist zuständig für die Verbindung mit dem Reverse-Pac-Man-Spiel.

View II ist für die Steuerung der Ghosts verantwortlich

Wird ein Botton im View II gedrückt, so werden Zahlen, die stellvertretend für die Spielrichtung stehen, an das Reverse-Pac-Man-Spiel übermittelt

3 Schnittstellen

3.1 tcp bridge (siehe Java-Doc)

Ist ein Modul, welches die Verbindung zwischen Reverse-Pac-Man auf dem PC und der App ermöglicht. Sie komplett dafür verantwortlich, Verbindungen zu akzeptieren und die Nachrichten an die einzelnen Ghost-Aktoren weiterzuleiten. Zusätzlich verwaltet sie diese Verbindungen .
Wird der TCP-Bridge ein neuer Ghost-Spieler hinzugefügt , so wird ein neuer Server-Socket eröffnet. Sobald sich ein Spieler verbunden hat, wird der Listener aktiviert und dieser Verbindung ein Bridgehandler hinzugefügt . Mit Hilfe des Bridgehandlers können die gesendeten Nachrichten von dem Socket direkt zu dem Ghost-Aktor weitergeleitet werden, der die Nachrichten interpretiert und wunschgemäß handelt. Bricht eine Verbindung zusammen, so versucht die TCP-Bridge die Überreste der Verbindung aufzuräumen.

3.2 KI-Data-Exchange-Pool

<<KiData>> Ist eine Klasse mit extrem vielen Attributen, die abgerufen und gesetzt werden können. Eine genaue Beschreibung liegt im JAVADOC vor. Daten wie Utility, aktueller Game-State, Menueparameter über z.B. ausgewählte KI, der letzte Berechnerte Suchbaum oder Plotinfos für die GraphStream-API werden zum Austausch bereitgehalten.
Die Einrichtung des Pools war notwenidig , um von allen Methoden des Projektes heraus zuzugreifen und Daten/Parameter auszutauschen. Zusätzlich dient der Pool als eine Art Nachrichten-Pushdienst: Er benachrichtigt verschiedene Komponenten über z.B. Statusänderungen wie Änderung des Powermodes. Diese Aufgabe wird durch das Interface <<kidataupdatehandler>> geregelt.
I-Interfasce das verschiedene Aktoren verbindung befehle weiterleitet an von ki an ggrid oder von tcp brige und von pac man zu ai core ,
es ist ein java – interface java – doc

D – Abbildungsverzeichnis und Anlagen

xxx Wörter

Quellen:

(1) Dynamo PLV (August 2013): Dynamische und nahtlose Integraiton von Produktion, Logistik und Verkehr – Technische Universität Darmstadt.

http://dynamo-plv.de/universitaeten/technische-universitaet-darmstadt/ (Abgerufen am 21.02.2016).

(2) CYSEC (Januar 2016): Cybersecurity TU Darmstadt – Masterstudiengang IT-Security. https://www.cysec.tu-darmstadt.de/de/cysec/?no_cache=1 (Abgerufen am 21.02.2016).

(3) TU Darmstadt (Januar 2016): Studienangebot. http://www.tu-darmstadt.de/studieren/abschluesse/index.de.jsp (Abgerufen am 16.02.2016).

(4) TU Darmstadt . (Februar 2016): Übersicht über alle Fachbereiche. http://www.tu-darmstadt.de/universitaet/fachbereiche/index.de.jsp (Abgerufen am 16.02.2016).

Abbildungen:

(Abbildung 1) die 13 Fachbereiche der TU Darmstadt:

TU Darmstadt . (Februar 2016): Übersicht über alle Fachbereiche. http://www.tu-darmstadt.de/universitaet/fachbereiche/index.de.jsp (Abgerufen am 16.02.2016).

Datum	Modul	Task	Activity	Length
	Einarbeitung GameGrid – App Programmierung			
10.08.16	**JGameGrid**	Einarbeitung, Info# sammeln über JGameGrid … Link download API LIBARY, PACMAN analysieren		3
		Versionshaltung		0,5
	App-Programmierung			
	SKILL	EVENTS	Erarbeitung von Custom-Events	0,5
11.08.16	**Base Game Pac-Man**	TCP Bridge	Interface für die Eventhandler implementieren Connectionklasse	2
12.08.16	**Base Game Pac-Man**	TCP Bridge	Exception Programming, Cleanup-als Stabilisierung, Management Class, Adding Full Mulitplayer Player Settings	4
13.08.16	**Tools**	Debugging Tools	Ghost-Ersatz mit manueller Steuerung	1
21.08.16	**SKILL**	APP - Einführung in App-Programmierung	Basic Structure- Activity – String- XML -View- Hello World Intent	3
23.08.16	**Reverse-Pac-Man-Remote-App**	Basic -Design		2
27-28		Async Task	Ähnlich wie Threats , mit mehreren Objekten als Rückgabe, Status und Beendet Signal Interfaceinvoke	5
29.08.16		Remote – advanced - Design		1
	Base Game Pac-Man – Modifikationen			
30.08.16	**Base Game PacMan**	Design – Konzept – Umsetzung	Konzeptplanung : Form / Farbe / Pixelspezifikation: 30x30 Pix Wandmittelpunkt bei 20 Pixel	1
31.08.16		Design – Konzept – Umsetzung		2
01.09.16		Design – Konzept – Umsetzung Level Engine: load source – cell select - build - dra	Level Engine – Block- Data – Assets Paint.Net	1,5
02.09.16		Design – Konzept – Umsetzung Level Engine: load source – cell select - build - dra	Assets – Erstellung mit Paint.Net Einlesen Datei, Textdateien bearbeiten	2,5
05.09.16		Design – Konzept – Umsetzung Level Engine: load source – cell select - build - dra	Einarbeitung XML	2
06.09.16		Design – Konzept – Umsetzung Level Engine: **load source** – cell select - build - dra	Level settings – eigene Lösung in java	2
07.09.16		Sound		2
08.09.16		Design – Fehlerbehebung		1
08.09.16		Hauptmenue Entwicklung	Levelname – Spieleranzahl	4
09.09.16		Main Menue		1
	AI – Module	**Netzwerk der Knotenpunkte als Basis für Routenfindung**		
10.09.17	**Planung Vorgehensweise**	Vorgehensweise zur Erarbeitung der KI Theorie und deren Anwendung		2
11.09.17	**KI-Theorie**	Studium von Vorlesungen und Buch	KI Einführung, Definition, planende Agenten: informierte Suche, uninformierte Suche	4
16.09.16	**A*Search**	Studium Theorie aus Buch und Vorlesung	Umsetzungsidee entwickelt	2
17.09.16		Datenstruktur: begehbare Blöcke im Grid darstellen		1
18.09.16		Knotenpunkte innerhalb des Grids identifizieren	Verfahren zur Knotenpunkterkennung , Datenstruktur zur Abspeicherung	3
18.09.16		Verbindung innerhalb der Knotenpunkte	Verfahren zur Verbindung – incl. neuer Debugoption – zur Darstellung Nodes und deren Verbindungen	3
20.09.16		Knotenüberarbeitung		1
21.09.16		A* Search	Asearch implementiert nach Logik siehe Flussdiagramm	2
		A* Search		2