Vergleich und Implementierung von Pfadfindungsalgorithmen zur Erweiterung der Geister von Pac-Man in Java

Inda-Gymnasium, Jahrgangsstufe 12, Schuljahr 2025/2026

Verfasser: Jakob Elijah Pütz

Kursleiter: Herr Minklai

Abgabetermin: 06.03.2026

Vorwort

Entstehung der Arbeit

Diese Facharbeit ist durch mein Interesse an Videospielen und Informatik entstanden. Deshalb war diese Facharbeit eine gute Chance mich näher mit Algorithmen, die in Videospielen verwendet werden, auseinanderzusetzen.

Zudem ist dieses Thema eine Möglichkeit, Wissen aus dem Informatikunterricht praktisch anzuwenden und in einem konkreten Projekt umzusetzen. Da ich im Unterricht bereits grundlegende Kenntnisse in Java erlangt habe, konnte ich auf diesem Wissen aufbauen und habe mich schnell eingefunden.

Das Thema dieser Facharbeit fiel bewusst auf Pac-Man, da dies ein klassisches Arcade-Spiel und deshalb ein passendes Beispiel für den Einsatz von Pfadfindungsalgorithmen darstellt. Die Geister verfolgen den Spieler nicht zufällig, sondern haben bestimmte Verhaltensmuster. Diese Idee inspirierte mich, unterschiedliche Algorithmen miteinander zu vergleichen und deren Stärken sowie Schwächen in einem echten Anwendungsbeispiel zu analysieren.

Besondere Schwierigkeiten

Im Verlauf dieser Facharbeit hat mir die Kombination aus bekannter Programmiersprache und völlig neuen algorithmischen Konzepten gezeigt, wie viel sich durch eigenständiges Einarbeiten erreichen lässt und wie eng die Entwicklung von Videospielen mit der Informatik verbunden ist.

Besondere Schwierigkeiten bereiteten mir vor allem die Umsetzung der theoretischen Kenntnisse über Algorithmen in konkreten Java-Code. Die Algorithmen sind in der Theorie gut verständlich, ihre konkrete

Implementierung erforderte jedoch ein gutes Verständnis der Code-Struktur und der Spiellogik.

Eine weitere Herausforderung stellte die Einarbeitung in ein bestehendes Projekt dar, welches als Grundlage diente, um das vollständige Neuschreiben von Pac-Man zu vermeiden. Der fremde Code musste deswegen vollständig verstanden, analysiert und anschließend gezielt angepasst werden, um die eigenen Algorithmen sinnvoll integrieren zu können.

Dieser Prozess war zeitaufwendig, zeigte mir jedoch, wie wichtig es ist, strukturierten und gut lesbaren Code zu schreiben. Diese Fähigkeit ist auch weit über diese Facharbeit hinaus wertvoll.

Inhaltsverzeichnis

Einleitung

Videospiele sind in der modernen Welt ein fester Bestandteil der Unterhaltung. Hinter ihrer simplen Oberfläche verbergen sich allerdings meist komplexe informatische Konzepte und Algorithmen, die für ein funktionierendes und herausforderndes Spielerlebnis sorgen. Eines der bekanntesten Beispiele hierfür ist Pac-Man, ein Arcade-Klassiker aus dem Jahr 1980[1].

Was Pac-Man dabei besonders interessant macht, ist das Verhalten der Geister. Diese könnten durch gezielte Verhaltensmuster, wie zum Beispiel vorausschauendes Abfangen, das Spielerlebnis deutlich verbessern und interessanter gestalten. Viele dieser Muster lassen sich durch Pfadfindungsalgorithmen modellieren und umsetzen. Genau dieser Aspekt bildet den Kern dieser Facharbeit.

Ziel ist es, verschiedene Pfadfindungsalgorithmen miteinander zu vergleichen, ihre jeweiligen Stärken und Schwächen herauszuarbeiten und sie anschließend in einem Java Pac-Man Spiel praktisch umzusetzen. Dabei wird untersucht, wie sich unterschiedliche Algorithmen auf das Verhalten der Geister und damit auf das Spielerlebnis auswirken.

Die Arbeit richtet sich an Leser mit grundlegenden Kenntnissen in der Programmierung, setzt jedoch kein Vorwissen im Bereich der Suchalgorithmen voraus, da die relevanten Konzepte im Verlauf der Arbeit schrittweise erläutert und implementiert werden.

Im Folgenden wird zunächst das Spiel Pac-Man selbst erklärt, damit auch weniger vertraute Leser wissen, womit sie es zu tun haben. Zudem werden die theoretischen Grundlagen der verwendeten Algorithmen dargelegt, bevor auf die praktische Implementierung und den anschließenden Vergleich eingegangen wird.

---

[1] Bandai Namco Studios Inc., *Pac-Man*, verfügbar unter: https://www.bandainamcostudios.com/en/products/pacman.html (abgerufen am 28.02.2026).

Pac-Man

Pac-Man ist ein Arcade-Spiel, welches 1980 von Namco entwickelt und veröffentlicht wurde. Das Spielprinzip ist dabei denkbar simpel. Der Spieler steuert die Spielfigur Pac-Man durch ein labyrinthartiges Spielfeld und muss alle auf dem Feld verteilten Punkte aufsammeln, ohne dabei von einem der vier Geister berührt zu werden.

Das Spielfeld besteht aus einem festen Labyrinth mit Gängen und Wänden. Neben den normalen Punkten gibt es an vier Ecken des Labyrinths sogenannte Energiepunkte. Sammelt Pac-Man einen dieser Energiepunkte ein, kehren sich die Rollen kurzzeitig um. Dabei werden die Geister blau und können für eine begrenzte Zeit von Pac-Man gefressen werden, bevor sie in ihre Ausgangsposition zurückkehren und das Spiel normal weitergeht. Diese Funktion wurde in dem vorhandenen Projekt von Drew Schuster[2] allerdings nicht implementiert.

Ein Level gilt als abgeschlossen, sobald Pac-Man alle Punkte auf dem Spielfeld eingesammelt hat. Verliert der Spieler alle Leben, endet das Spiel. Ziel ist es, dabei eine möglichst hohe Punktzahl zu erreichen.[3]

Was das Spiel bis heute so besonders macht, ist die Komplexität trotz der einfachen Regeln, die durch das Verhalten der vier Geister entsteht. Jeder Geist besitzt ein eigenes, individuelles Verhaltensmuster, welches das Spiel[4] abwechslungsreich und herausfordernd gestaltet. Auf dieses Verhalten und die zugrundeliegenden Algorithmen wird in den folgenden Kapiteln genauer eingegangen.

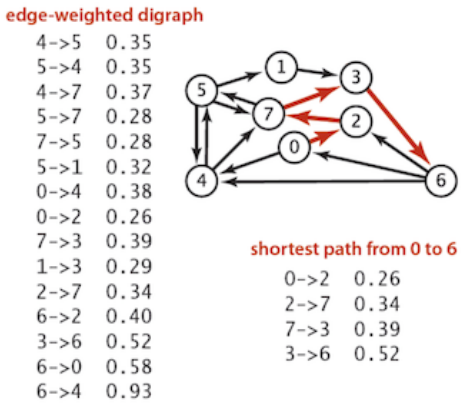
[2] Drew Schuster, *Java Pac-Man*, verfügbar unter: https://github.com/dtschust/javapacman (abgerufen am 19.01.2026).
[3] Pac-Man.com, *The History of Pac-Man*, verfügbar unter: https://pacman.com/en/history/ (abgerufen am 28.02.2026).
[4] Pac-Man.com, *Pac-Man Spielansicht (1980)*, verfügbar unter: https://pacman.com/images/history/y1980/pic_game.png (abgerufen am 28.02.2026).

Dijkstra Algorithmus

Der Dijkstra-Algorithmus ist ein Suchalgorithmus zur Berechnung des kürzesten Pfades in einem gewichteten Graphen. Er wurde von dem niederländischen Informatiker Edsger W. Dijkstra entwickelt und 1959 veröffentlicht.[5]

Ein Graph besteht dabei aus Knoten (engl. *vertices*) und Kanten (engl. *edges*), die jene Knotenpunkte miteinander verbinden. Bei einem gewichteten Graphen besitzt jede Kante zusätzlich ein Gewicht, welches beispielsweise eine Distanz oder einen Kostenwert darstellt. Der Algorithmus findet somit den kürzesten Weg von einem Startknoten zu allen anderen Knoten, sofern die Kantengewichte nicht negativ sind.[6]

Der Algorithmus funktioniert nach einem sogenannten *Greedy-Prinzip*. Ausgehend von einem Startknoten werden allen Knotenpunkten zunächst Distanzwerte zugewiesen. Dem Startknoten den Wert null, allen anderen den Wert unendlich. Dann wird wiederholt der noch nicht besuchte Knoten mit der *kleinsten bekannten*[7] Distanz ausgewählt und dessen Nachbarn werden auf kürzere Wege überprüft. Ist ein neu berechneter Weg kürzer als der bisher gespeicherte, wird dieser



[5] Madkour et al., *A Survey of Shortest-Path Algorithms*, 2017, verfügbar unter: https://arxiv.org/pdf/1705.02044 (abgerufen am 28.02.2026).
[6] Codecademy Team, *A Complete Guide to Dijkstra's Shortest Path Algorithm*, verfügbar unter: https://www.codecademy.com/article/dijkstras-shortest-path-algorithm (abgerufen am 28.02.2026).
[7] Oxford Emory Mathematics Center, *Beispielgraph für Dijkstra's Shortest Path Algorithm*, verfügbar unter: https://mathcenter.oxford.emory.edu/site/cs171/dijkstrasShortestPathAlgorithm/658-00.png (abgerufen am 28.02.2026).

aktualisiert. Dieser Vorgang wird so lange wiederholt, bis alle Knoten besucht wurden.[8]

Für den Einsatz in Pac-Man lässt sich das Spielfeld als Graph modellieren, bei dem jedes begehbare Feld einem Knoten entspricht und benachbarte Felder durch Kanten verbunden sind. Der Dijkstra-Algorithmus kann so den kürzesten Weg eines Geistes zu Pac-Man berechnen und ihn damit direkt und effizient verfolgen.

A Algorithmus*

Der A*-Algorithmus (ausgesprochen „A-Star") ist ein Pfadfindungsalgorithmus, der 1968 von Peter Hart, Nils Nilsson und Bertram Raphael entwickelt und veröffentlicht wurde.[9] Er gilt heute als einer der am häufigsten eingesetzten Algorithmen in der Spieleentwicklung.

A* ist eine Erweiterung des, vorhin erläuterten, Dijkstra-Algorithmus. Der entscheidende Unterschied zu Dijkstra besteht darin, dass A* eine sogenannte *Heuristik* verwendet, um die Suche gezielt in Richtung des Ziels zu lenken und somit effizienter zu arbeiten. Dazu verwendet der Algorithmus die folgende Kostenfunktion: **f(n) = g(n) + h(n)** wobei:

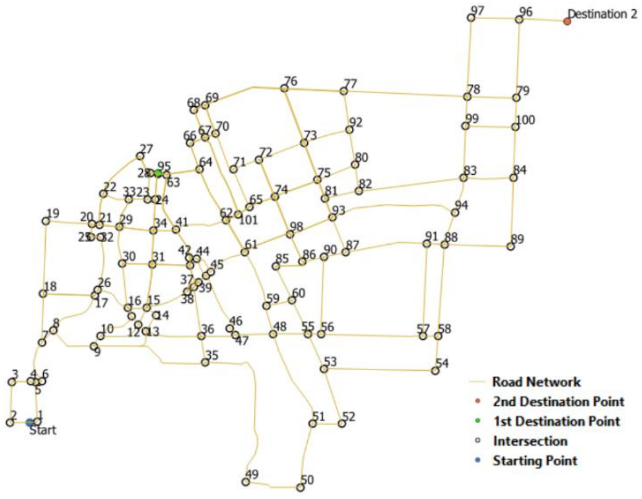
- **g(n)** die bereits angefallenen Kosten vom Startknoten bis zum aktuellen Knoten n beschreibt,
- **h(n)** eine Schätzung der verbleibenden Kosten des aktuellen Knotens bis zum Ziel darstellt (die *Heuristik*),
- **f(n)** die geschätzten Gesamtkosten des Pfades durch Knoten **n** angibt.

[8] Oxford Emory Mathematics Center, *Dijkstra's Shortest Path Algorithm*, verfügbar unter: https://mathcenter.oxford.emory.edu/site/cs171/dijkstrasShortestPathAlgorithm/ (abgerufen am 28.02.2026).
[9] Peter Hart, Nils Nilsson, Bertram Raphael, *A Formal Basis for the Heuristic Determination of Minimum Cost Paths*, 1968, verfügbar unter: https://www.cs.auckland.ac.nz/courses/compsci709s2c/resources/Mike.d/astarNilsson.pdf (abgerufen am 28.02.2026).

Eine häufig verwendete Heuristik für rasterbasierte Probleme ist die Manhattan-Distanz, also die Summe der horizontalen und vertikalen Abstände zwischen zwei Punkten. Wichtig ist dabei, dass die Heuristik zulässig ist, sie darf also die tatsächlichen Kosten niemals überschätzen. Wird **h(n) = 0** gesetzt, verhält sich A* wie der Dijkstra-Algorithmus und garantiert den kürzesten Weg.[10]

Für den Einsatz in Pac-Man bietet A* gegenüber Dijkstra somit einen klaren Vorteil. Da das Ziel, Pac-Mans Position, stets bekannt ist, kann die Heuristik die Suche gezielt dorthin führen, anstatt den kürzesten Weg zu allen Knoten des Graphen zu berechnen. Dies macht A* in der Regel schneller und effizienter.
Dies lässt sich in einer interaktiven Visualisierung nachvollziehen, in der beide Algorithmen auf einem Gitter verglichen werden können und deutlich wird, wie viele Knoten A* im Vergleich zu Dijkstra tatsächlich besucht.[11]

Aufgrund seiner Effizienz ist A* weit über die Spieleentwicklung hinaus verbreitet und findet beispielsweise auch in Navigationsdiensten wie Google Maps Anwendung.[12]



[10] Marko Novaković, *Pathfinding Algorithms in Games*, EdTech Journal, 2023, verfügbar unter: https://doi.fil.bg.ac.rs/pdf/journals/edtech/2023-1/edtech-2023-3-1-5.pdf (abgerufen am 28.02.2026).
[11] Xueqiao Xu, *PathFinding.js Visual*, verfügbar unter: https://qiao.github.io/PathFinding.js/visual/ (abgerufen am 28.02.2026).
[12] IOP Publishing, *Figure 2: Mapping all the possible intersection without map background*, verfügbar unter: https://iopscience.iop.org/article/10.1088/1742-6596/1566/1/012061/pdf (abgerufen am 28.02.2026).

Implementierung

Als Grundlage der Implementierung von den Algorithmen diente ein bestehendes Open-Source Pac-Man Projekt von Drew Schuster.[13] Da das vollständige Programmieren des Spiels den Rahmen dieser Facharbeit überschritten hätte, wurde dieses Projekt als Basis genutzt und gezielt mit den genannten Pfadfindungsalgorithmen erweitert.

Das Implementationsdiagramm zeigt den Aufbau des Projektes mit den Klassen und ihren Attributen sowie Methoden. Die zentrale Klasse für die Algorithmen ist die Klasse *Ghost*, welche eine Unterklasse von *Mover* ist. Die Klasse *Mover* sorgt dabei für wichtige Eigenschaften und Methoden, die sowohl von Pac-Man als auch von den Geistern verwendet werden, wie zum Beispiel die aktuelle Position, die Bewegungsrichtung sowie die Kollisionsprüfung mit Wänden.

Die Klasse *Board* verwaltet das gesamte Spielfeld und enthält die vier Geister Instanzen in einer Liste sowie den Spieler. Die Klasse *Pacman* übernimmt die Spielsteuerung und den Spielablauf.

Für die Auswahl des Algorithmus wurde in der Klasse *Ghost* eine Aufzählung eingeführt, welche die Modi in *RANDOM*, *DIJKSTRA*, *ASTAR*, *TEAM* und *RANDOM_EACH* unterscheidet. Der Spieler kann vor Spielbeginn per Tastendruck den gewünschten Algorithmus wählen. Dieser wird dann an alle vier Geister weitergegeben.

Der Kern der Implementierung liegt in der Methode **findPath()**. Diese Methode implementiert sowohl *Dijkstra* als auch *A** in einer gemeinsamen Funktion. Der einzige Unterschied zwischen beiden besteht darin, ob die Heuristik verwendet wird oder nicht. Wird der Parameter *useHeuristic* auf *false* gesetzt, verhält sich die Methode wie Dijkstra, da **f(n)** dann ausschließlich aus den bisherigen Wegkosten **g(n)** besteht.

[13] Drew Schuster, *Java Pac-Man*, verfügbar unter: https://github.com/dtschust/javapacman (abgerufen am 19.01.2026).

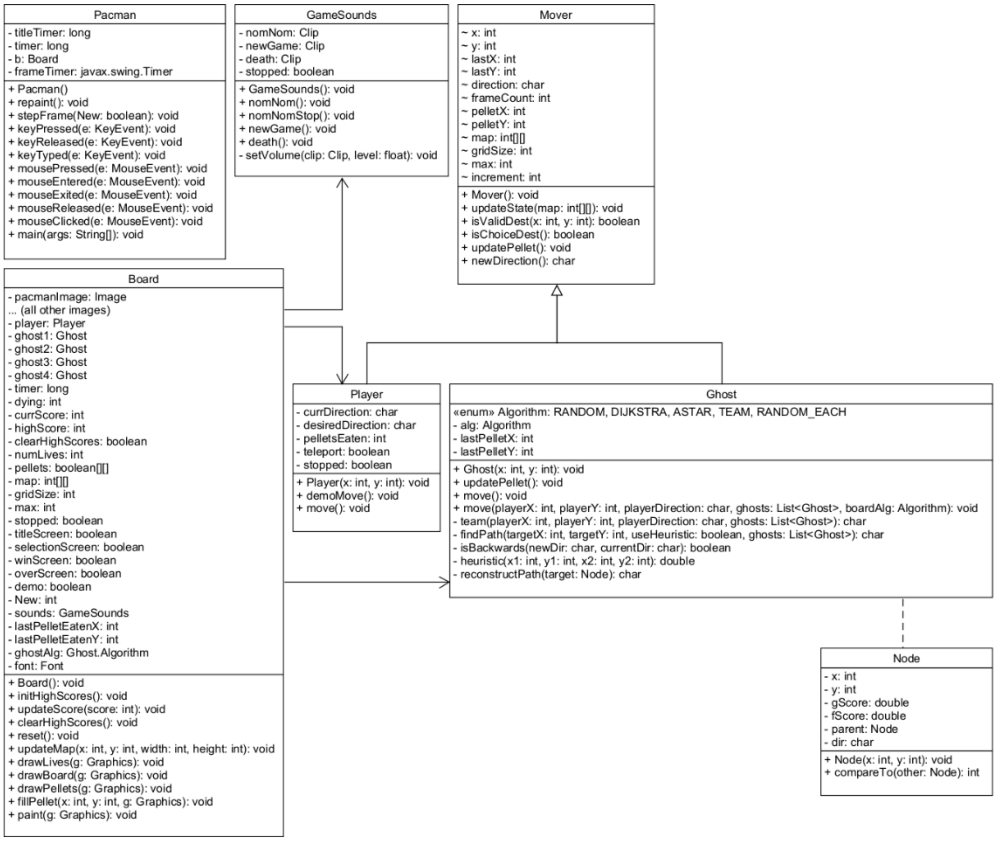
Wird *useHeuristic* auf *true* gesetzt, wird zusätzlich die Manhattan-Distanz als Heuristik **h(n)** hinzugerechnet. Somit arbeitet die Methode dann als *A**.

Das Spielfeld wird dabei als zweidimensionales Feld modelliert, bei dem jede begehbare Kachel einem Knoten entspricht. Wände werden durch den Wert 1 dargestellt und somit vom Algorithmus übersprungen.

Um ein unnatürliches Rückwärtslaufen der Geister zu verhindern, wurden „Strafkosten“ eingebaut. Kehrt ein Geist seine aktuelle Richtung um, werden für diesen Schritt erhöhte Kosten von 10 statt 1 berechnet.

Die fertige Implementierung und der Quellcode zu dieser Facharbeit sind auf meinem persönlichen Github Account verfügbar.[14]

[15]



[14] dev-banane, *Java Pac-Man (Fork)*, verfügbar unter: https://github.com/dev-banane/javapacman (abgerufen am 28.02.2026).
[15] Abbildung 1: Implementationsdiagramm (eigene Darstellung)

Teamspiel

Neben den beiden Pfadfindungsalgorithmen wurde zudem ein Teammodus implementiert, der ein koordiniertes Einkreisungsverhalten der vier Geister verursacht. Das Ziel dieses Modus ist es, dass die Geister den Spieler nicht nur verfolgen, sondern ihn aktiv von allen Seiten einkreisen.

Die besondere Herausforderung dabei war es, jedem Geist jeweils einen sinnvollen individuellen Zielpunkt zuzuweisen, sodass sie gemeinsam ein Kreuz um den Spieler bilden. Geist 1 verfolgt Pac-Man dabei direkt und übt konstanten Druck von hinten aus. Geist 2 zielt auf eine Position vier Felder vor Pac-Man in dessen aktueller Bewegungsrichtung und versucht so, ihn von vorne abzufangen. Geist 3 und Geist 4 flankieren den Spieler von den Seiten. Inky weicht je nach Bewegungsrichtung Pac-Mans um vier Felder zur Seite aus, während Clyde dieselbe Logik in die entgegengesetzte Richtung anwendet.

Die Schwierigkeit bei der Umsetzung war dabei die Zielpunkte korrekt und relativ zur aktuellen Bewegungsrichtung Pac-Mans zu berechnen, sodass die Geister koordiniert von verschiedenen Seiten angreifen und nicht zufällig herumlaufen. Alle vier Geister verwenden intern den A*-Algorithmus, um ihren jeweiligen Zielpunkt anzusteuern. Somit entsteht ein Einkreisungsverhalten, das für den Spieler deutlich schwerer zu entkommen ist als bei einfacher direkter Verfolgung.

Zusammenfassung

Ziel dieser Facharbeit war es, Pfadfindungsalgorithmen zu vergleichen und in einem Pac-Man-Spiel einzusetzen.

Dabei wurde deutlich, dass beide Algorithmen zuverlässig den kürzesten Weg finden, sich jedoch in ihrer Effizienz stark unterscheiden.

Vergleich

Dijkstra berechnet den kürzesten Weg zu allen anderen Knoten des Graphen gleichzeitig, ohne dabei eine Richtung zu bevorzugen. Somit erkundet *Dijkstra* das Spielfeld gleichmäßig in alle Richtungen und besucht daher auch viele Felder, die für das eigentliche Ziel irrelevant sind. Das macht Dijkstra zuverlässig und präzise, allerdings in Situationen mit einem bekannten Ziel vergleichsweise ineffizient.

*A** hingegen verwendet die Manhattan-Distanz als *Heuristik*, um die Suche gezielt in Richtung des Ziels zu bewegen. Somit werden deutlich weniger Knoten besucht und der kürzeste Weg wird schneller gefunden. Da das Ziel in Pac-Man immer bekannt ist, ist *A** daher deutlich effizienter. Dieser Unterschied lässt sich in der interaktiven Visualisierung von Xueqiao (Joe) Xu erkennen. In seiner Visualisierung wird deutlich sichtbar, wie viele Knoten Dijkstra im Vergleich zu A* besucht.[16]

Für den Spieler ist der Unterschied zwischen den beiden Algorithmen kaum spürbar, da beide immer den optimalen Weg berechnen und die Geister sich dadurch sehr ähnlich verhalten. Der praktische Vorteil von A* liegt somit nicht im Spielerlebnis, sondern in der Recheneffizienz. Ein Aspekt, der vor allem in komplexeren oder größeren Spielfeldern relevant werden würde.

[16] Xueqiao Xu, *PathFinding.js Visual*, verfügbar unter: https://qiao.github.io/PathFinding.js/visual/ (abgerufen am 28.02.2026).

Der implementierte Teammodus stellt dafür allerdings, im Gegensatz zu beiden einzelnen Algorithmen, die größte Herausforderung für den Spieler dar. Während Dijkstra und A* alle vier Geister zum gleichen Ziel führen, verteilt der Teammodus die Geister auf verschiedene Zielpunkte rund um den Spieler. Durch diese Kombination aus direkter Verfolgung, vorausschauendem Abfangen und seitlichem Flankieren entsteht ein Einkreisungsverhalten, dem der Spieler deutlich schwerer entkommen kann.

Persönliche Beurteilung

Persönlich hat mich diese Facharbeit, sowohl fachlich als auch praktisch, sehr weitergebracht. Die eigenständige Einarbeitung in das Thema der Pfadfindungsalgorithmen, das bisher nicht Teil des Unterrichts war, hat mir gezeigt, wie man sich neues Wissen selbstständig erschließen und direkt praktisch anwenden kann. Besonders die Umsetzung in einem echten Spielprojekt hat das Verständnis für diese Algorithmen erheblich vertieft, da abstrakte Konzepte wie Graphen und Heuristiken plötzlich ein sichtbares und nachvollziehbares Ergebnis hatten. Somit habe ich beispielsweise bemerkt, dass der beste und effizienteste Suchalgorithmus nicht direkt bedeutet, dass der Spieler ein besseres Spielerlebnis hat.

Ausblick

Als Ausblick wäre es interessant, randomisierte Algorithmen, wie den Monte Carlo Algorithmus, zu vergleichen. Zudem könnte das Verhalten der Geister durch maschinelles Lernen weiterentwickelt werden, sodass sie sich dynamisch an das Spielverhalten des Nutzers anpassen. Ein Ansatz, der in modernen Spielen zunehmend Anwendung findet.

Literaturverzeichnis

Bandai Namco Studios Inc., *Pac-Man*, verfügbar unter: https://www.bandainamcostudios.com/en/products/pacman.html (abgerufen am 28.02.2026). [1]

Codecademy Team, *A Complete Guide to Dijkstra's Shortest Path Algorithm*, verfügbar unter: https://www.codecademy.com/article/dijkstras-shortest-path-algorithm (abgerufen am 28.02.2026). [6]

dev-banane, *Java Pac-Man (Fork)*, verfügbar unter: https://github.com/dev-banane/javapacman (abgerufen am 28.02.2026). [14]

Dijkstra, Edsger W., *A note on two problems in connexion with graphs*, Numerische Mathematik, 1959, verfügbar unter: https://ir.cwi.nl/pub/9256/9256D.pdf (abgerufen am 28.02.2026).

Hart, Peter / Nilsson, Nils / Raphael, Bertram, *A Formal Basis for the Heuristic Determination of Minimum Cost Paths*, 1968, verfügbar unter: https://www.cs.auckland.ac.nz/courses/compsci709s2c/resources/Mike.d/astarNilsson.pdf (abgerufen am 28.02.2026). [9]

IOP Publishing, *Figure 2: Mapping all the possible intersection without map background*, verfügbar unter: https://iopscience.iop.org/article/10.1088/1742-6596/1566/1/012061/pdf (abgerufen am 28.02.2026). [12]

Madkour et al., *A Survey of Shortest-Path Algorithms*, 2017, verfügbar unter: https://arxiv.org/pdf/1705.02044 (abgerufen am 28.02.2026). [5]

Novaković, Marko, *Pathfinding Algorithms in Games*, EdTech Journal, 2023, verfügbar unter: https://doi.fil.bg.ac.rs/pdf/journals/edtech/2023-1/edtech-2023-3-1-5.pdf (abgerufen am 28.02.2026). [10]

Oxford Emory Mathematics Center, *Dijkstra's Shortest Path Algorithm*, verfügbar unter:

https://mathcenter.oxford.emory.edu/site/cs171/dijkstrasShortestPathAlgorithm/ (abgerufen am 28.02.2026). [8]

Oxford Emory Mathematics Center, *Beispielgraph für Dijkstra's Shortest Path Algorithm*, verfügbar unter: https://mathcenter.oxford.emory.edu/site/cs171/dijkstrasShortestPathAlgorithm/658-00.png (abgerufen am 28.02.2026). [7]

Pac-Man.com, *Pac-Man Spielansicht (1980)*, verfügbar unter: https://pacman.com/images/history/y1980/pic_game.png (abgerufen am 28.02.2026). [4]

Pac-Man.com, *The History of Pac-Man*, verfügbar unter: https://pacman.com/en/history/ (abgerufen am 28.02.2026). [3]

Schuster, Drew, *Java Pac-Man*, verfügbar unter: https://github.com/dtschust/javapacman (abgerufen am 19.01.2026). [2] [13]

Xu, Xueqiao, *PathFinding.js Visual*, verfügbar unter: https://qiao.github.io/PathFinding.js/visual/ (abgerufen am 28.02.2026). [11] [16]

Erklärung über die Selbständige Anfertigung der Arbeit

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig und ohne fremde Hilfe verfasst und keine anderen Hilfsmittel als die angegebenen verwendet habe. Ich versichere, dass ich alle wörtlichen und sinngemäßen Übernahmen aus anderen Werken – dazu gehören auch Internetquellen und KI-Tools – als solche kenntlich gemacht habe.

52078 Aachen, 04.03.2026, Jakob Elijah Pütz

Anhang

Abbildung 1: Implementationsdiagramm (eigene Darstellung)