Ostbayerische Technische Hochschule Amberg-Weiden
Fakultät Elektrotechnik, Medien und Informatik

Prof. Dr. Tatyana Ivanovska

Studiengang Künstliche Intelligenz

Projektarbeit Katastrophensimulation

Simon Völkl & Sebastian Kleber

13. Mai 2025

Inhaltsverzeichnis

1 Einleitung

In Katastrophenszenarien wie Erdbeben, Bränden oder Gebäudeeinstürzen ist die schnelle und effiziente Rettung von Überlebenden entscheidend. Moderne Technologien wie autonome Roboter können hierbei eine zentrale Rolle spielen, insbesondere wenn der Zugang für menschliche Rettungskräfte erschwert oder zu gefährlich ist.

Im Rahmen dieser Arbeit wurde ein System zur Simulation und Analyse von Rettungseinsätzen in einem Labyrinth entwickelt. Ziel ist es, verschiedene Labyrinthe algorithmisch zu generieren, deren strukturelle Eigenschaften zu evaluieren und daraus Rückschlüsse für den optimalen Einsatz eines Rettungsroboters zu ziehen.

Die im Folgenden dokumentierte Umsetzung orientiert sich direkt an der gestellten Aufgabenstellung, die sowohl den inhaltlichen Rahmen als auch den strukturellen Aufbau dieses Berichts vorgibt. Die Aufgaben dienen im weiteren Verlauf als Referenzpunkte und Leitfaden für die einzelnen Kapitel.

1.1 Aufgabenstellung

1. Implementieren Sie eine Methode zur Erstellung eines Labyrinths
2. Evaluieren Sie jedes erstelle Labyrinth mit den folgenden Metriken:
   - Die kleinste und die mittlere Pfadlänge im Labyrinth
   - Dichte: Verhältnis von Wänden zu offenen Wegen von der gesamten Fläche des Labyrinths
   - Die Anzahl der Ausgänge und die Symmetries des Labyrinths
3. Für ein ausgewähltes Labyrinth erzeugen Sie zufällig einige Überlebende, die gerettet werden müssten.
4. Analysieren Sie für jedes ausgewählte Labyrinth:
   - Wo wäre es am besten für einen Rettungsroboter die Sequenz der Rettungen zu starten, sodass alle Überlebenden am schnellsten gerettet werden könnten? Der Roboter kann nur 1 Person tragen.
   - Wo soll der Roboter die Überlebenden am besten raustragen?
5. Stellen Sie Ihre Ergebnisse visuell dar.
6. Überlegen und implementieren Sie eine eigene Erweiterung des Systems.

1.2 Methodik

Zur Umsetzung der Aufgabenstellung wurde zunächst eine Methode zur algorithmischen Erstellung von Labyrinthen implementiert. Die erste Entwicklungsphase basierte auf der Nutzung von pyamaze, dessen interne Repräsentation auf verschachtelten

Dictionaries beruhte. Diese Struktur erwies sich jedoch als unpraktisch für komplexere Operationen, und man konnte nicht mit eigens erstellten Mazes auf die verwendete Visualisierungs-Methoden zugreifen.

Daraufhin haben wir auf einen objektorientierten Ansatz gewechselt, bei dem einzelne Zellen des Labyrinths als Tiles modelliert und in einem übergeordneten Environment organisiert wurden. Die Generierung der Labyrinthe erfolgt dabei mittels eines zufallsbasierten Tiefensuchalgorithmus (random DFS), wodurch eine gut kontrollierbare Erzeugung zusammenhängender Strukturen ermöglicht wurde. Die Visualisierung wurde daraufhin textbasiert in der Console umgesetzt.

Für die Analyse wurde das Python-Paket NetworkX eingesetzt, was die Realisierung des Labyrinths mittels Graphen erlaubt. Der Graph dient auch als Grundlage für Pfadsuchalgorithmen. Beim Durchstöbern der Bibliothek sind wir auf eine weitere Variante der Visualisierung gestoßen, welche durch den Einsatz von Positionsparametern den generierten Graphen auf ein Gitter abbildet. Mit Hilfe des Graphen wurde später der A*-Algorithmus implementiert, um optimale Pfade vom Roboter zu den Überlebenden und weiter zu geeigneten Ausgängen zu finden.

Die Bewertung der Labyrinthe erfolgte anhand verschiedener Metriken wie minimaler und mittlerer Pfadlänge, Dichte (Verhältnis von Wänden zu Wegen), Anzahl der Ausgänge und Achsensymmetrie. In der Umgebung wurden Methoden implementiert, um zufällig Ausgänge und Überlebende zu generieren.

Parallel zur algorithmischen Entwicklung wurde die Projektstruktur laufend überarbeitet und optimiert, die Umsetzung der neuen Struktur ist jedoch zum aktuellen Zeitpunkt noch nicht vollständig.

Zur weiterführenden Analyse der Labyrinthe wurden ergänzend Jupyter-Notebooks eingesetzt. Diese dokumentieren den zugrunde liegenden Code sowie weiterführende Diskussionen der Ergebnisse im Detail. Sie dienen als sinnvolle Ergänzung zu diesem Bericht und sollten zum vollständigen Verständnis der Umsetzung und Analyse herangezogen werden.

Das erste Notebook `simulation_example` beinhaltet die initiale Generierung eines Labyrinths mittels randomisiertem DFS, die Platzierung von Überlebenden und Savezones sowie die Visualisierung mit NetworkX. Anschließend werden zentrale Metriken wie Pfadlängen, Dichte, Ausgänge und Symmetrie berechnet. In einer schrittweisen Simulation mithilfe von Mesa führt ein Roboterhund Rettungsaktionen durch, wobei Pfade per A*-Algorithmus auf Basis der Manhattan-Distanz bestimmt werden. Zusätzlich wurde die Auswirkung verschiedener Startpositionen des Roboters auf die Effizienz der Rettung analysiert.

Für die Erweiterung wurde ein zweites Notebook `batch_run` entwickelt, das mithilfe der batch-run-Funktion aus Mesa umfangreiche Simulationen unter variierenden Parametern (z.B. Labyrinthgröße, Anzahl der Roboter, Überlebenden und Savezones) automatisiert durchführt. Die dabei gesammelten Daten wurden mit Pandas verarbeitet und mit Seaborn visualisiert. Analysiert wurden unter anderem die Korrelation zwischen Wanddichte und Pfadlänge, der Einfluss der Start- und Zielverteilung auf

die Effizienz sowie die Auswirkungen von Agentenanzahl und Labyrinthgröße auf benötigte Schritte und Bewegungsdistanzen. Die Ergebnisse bieten tiefergehende Einblicke in das Systemverhalten und helfen, realistische Annahmen für Katastrophenszenarien besser zu modellieren.

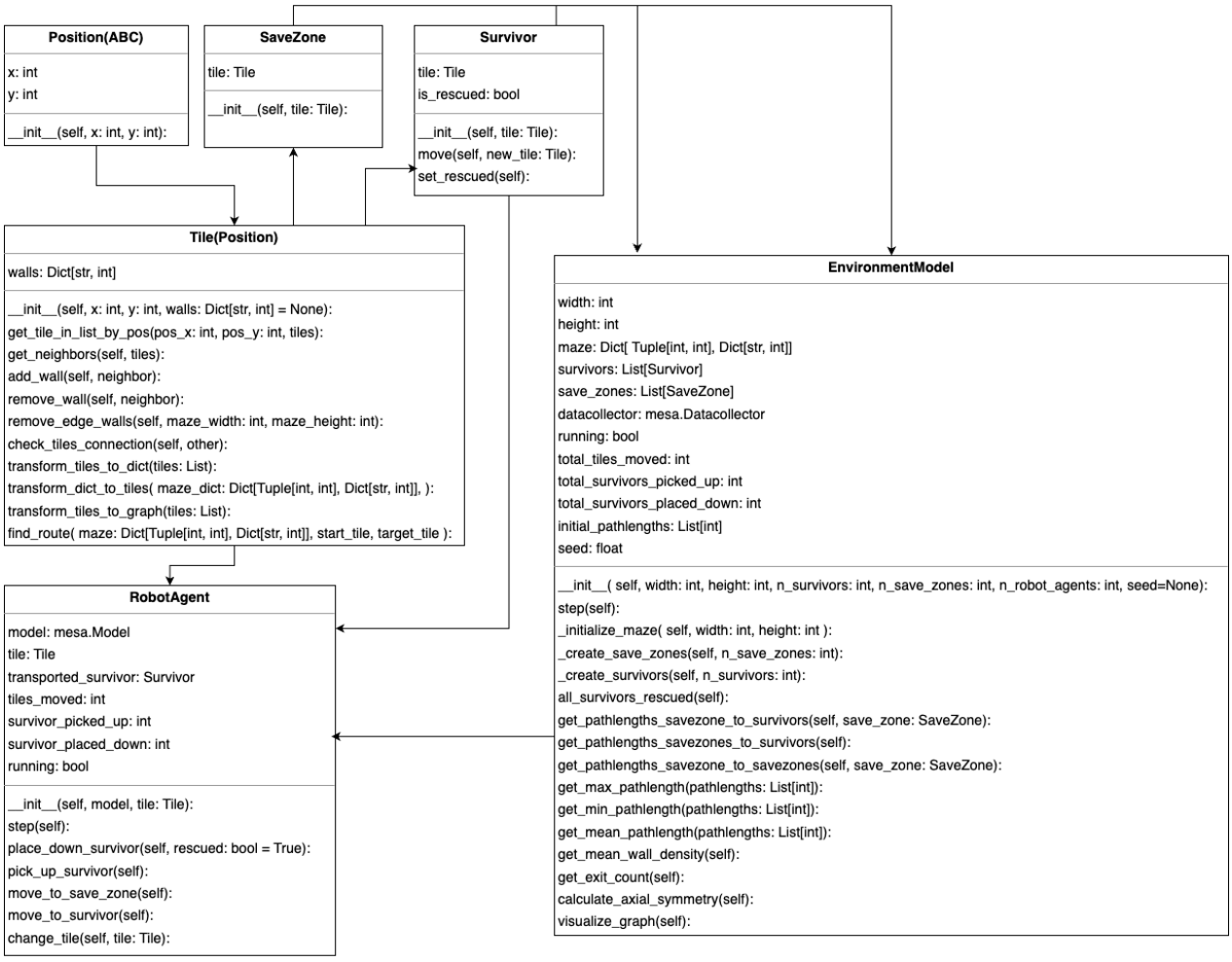

Abbildung 1: Klassendiaramm

2 Implementierung

Die Implementierung basiert auf einem objektorientierten Design, das in Form eines Klassendiagramms 1 die erste Iteration der Implementierung widerspiegelt. Dabei wurde auf eine klare Trennung der Verantwortlichkeiten geachtet: Die abstrakte Basisklasse Position dient als Grundlage für alle Objekte mit Raumbezug im Labyrinth, wie etwa Tile, SaveZone und Survivor. Durch diese Vererbung wird redundanter Code vermieden und eine konsistente Struktur gewährleistet. Die Klasse Tile erweitert Position um Informationen zu Wänden und Nachbarn, die für die Labyrinthstruktur entscheidend sind.

Das zentrale Steuerelement bildet das EnvironmentModel, das die Umgebung samt aller Objekte verwaltet und Funktionen zur Initialisierung, Analyse und Visualisierung bereitstellt. Der RobotAgent agiert innerhalb dieser Umgebung und verfügt über Methoden zur Bewegung, Aufnahme und Platzierung von Überlebenden. Das System ist modular aufgebaut und lässt sich durch die Nutzung von Standardklassen leicht erweitern und testen.

Ein Großteil der technischen Dokumentation und des erklärenden Codes befindet sich, wie bereits erwähnt, in den bereitgestellten Jupyter-Notebooks. Diese enthalten detaillierte Beschreibungen der Implementierung, Zwischenergebnisse sowie weiterführende Analysen und Visualisierungen. Um den vollständigen Umfang des Projekts nachzuvollziehen und ein tieferes Verständnis der entwickelten Logik und Abläufe zu gewinnen, empfehlen wir ausdrücklich, diese Dateien ergänzend zum Bericht gründlich zu betrachten.

Im Ordner `src` befindet sich eine überarbeitete und modularisierte Version der ursprünglichen Projektstruktur. In dieser Version setzen wir verstärkt auf die Verwendung von dataclasses, um datenhaltende Strukturen klarer, kompakter und besser wartbar darzustellen. Diese Überarbeitung bietet eine deutlich verbesserte Codequalität, eine übersichtlichere Architektur und erste Optimierungen im Vergleich zur ursprünglichen Umsetzung. Ebenso wie die Jupyter-Notebooks sollte auch dieser Teil des Projekts ergänzend zum Bericht betrachtet werden, da hier zentrale Konzepte und Abläufe strukturiert und nachvollziehbar implementiert sind. Der Quellcode steht öffentlich im Repository unter https://github.com/s-voelkl/Catastrophe-Simulator zur Verfügung.

3 Zusammenfassung und Ausblick

Aus unserer Sicht konnten die gestellten Aufgaben, wie im Jupyter Notebook ersichtlich, gut bearbeitet werden.

Anfänglich hatten wir einige Probleme mit der gegebenen Struktur von Pyamaze als Visualisierungswahl. Nach dem Umstieg auf den objektorientierten Anstatz konnten die Aufgaben jedoch deutlich besser bearbeitet werden.

Verbesserungspotenzial bestünde in dem Refactoring und Aufwerten des Codes. Beispielsweise wird die Open List bei der Pfadfindung als Dictionary gespeichert und muss bei jeder Iteration aufsteigend nach Werten sortiert werden. Eine Verbesserung sähen wir hier durch die Verwendung einer Warteschlange, die man z.B. als Priority Queue in der Bibliothek heapq implementieren könnte. Weiterhin ist die anfängliche Datenstruktur als Dictionary im späteren Verlauf hinderlich geworden; eine Aufbereitung als NetworkX Graph oder als Liste an Tile-Objekten wäre eine deutlich bessere Datenstruktur gewesen. Viele Verbesserungen wurden bereits im Projektordner src aufbereitet, wobei sie in Tests eine deutlich bessere Performance als die alte Struktur beweisen. Aufgrund des begrenzten Projektzeitraums war es uns nicht mehr möglich, die Verbesserungen zu integrieren.

Für die tiefgehende Datenanalyse als Erweiterung des Systems im späteren Teil des Projekts erwies sich die eingebaute Batch Run-Funktion von Mesa als sehr hilfreich. Mit ihr und anderen Bibliotheken konnten genaue Visualisierungen angestellt werden. Die Correlation Heatmap zeigte bereits einige Korrelationen auf, die im weiteren Verlauf genauer analysiert wurden. Dabei konnten einige erwartete Zusammenhänge bewiesen werden, andere unerwartete Beziehungen konnten aufgezeigt werden.

Vor allem folgende Zusammenhänge wurden besonders beleuchtet:

- Mit steigender Flächen des Labyrinths werden mehr Sackgassen erzeugt, wodurch insgesamt die Pfadlängen sinken, was allerdings bewiesenermaßen kein linearer Zusammenhang ist.
- Eine höhere Fläche hat auch eine steigende Wanddichte, also die durchschnittlichen Wände pro Feld durch die Anzahl aller möglichen Wände, zur Folge, da stetig weniger Randfelder berücksichtigt werden. Dabei nähert sich der Wert der Wanddichte gegen unendliche Labyrinthgrößen an 0,5 von unten an.
- Die Zahlen der Savezones und der Survivors haben einen reduzierenden Einfluss auf die insgesamt bewegten Felder der Roboter. Mehr Roboter verringern zwar die Simulationsschritte, die Feldbewegungen werden jedoch durch gegenseitiges Behindern ineffizienter.

In einer wirklichen Katastrophenrettung sollten folgende Punkte aus den hier gewonnenen Erkenntnissen beachtet werden:

Eine höhere Anzahl an Savezones ist essenziell, damit die Pfadlängen zu den Überlebenden möglichst stark reduziert werden. Gleichzeitig sollten mehrere Roboterhunde eingesetzt werden, damit sich die Anzahl der Schritte, bis jeder Überlebende gerettet

wurde, reduziert. Essenziell dabei ist, dass sich die Roboterhunde nicht gegenseitig behindern, sondern die Kommunikation und Koordination untereinander ausreichend besteht.

Setzt man diese Empfehlungen für eine Katastrophenrettung, basierend auf den Datenanalysen der Katastrophenrettungssimulation um, können erhebliche Verluste stark reduziert werden.

Literatur

[1] ACL. *Why does networkx redraw my graph different each run?* Last accessed 09. May 2025. 2018. URL: https://stackoverflow.com/questions/53734823/why-does-networkx-redraw-my-graph-different-each-run.

[2] Victor Bellot u. a. "How to Generate Perfect Mazes?" In: *Information Sciences* 546 (2021), S. 1–20. DOI: 10.1016/j.ins.2021.03.022.

[3] Caltech. *Summary of the A* Algorithm*. Last accessed 09. May 2025. URL: https://robotics.caltech.edu/wiki/images/e/e0/Astar.pdf.

[4] Stack Exchange. *How to randomly create a fair maze for a multiplayer game?* Last accessed 07. May 2025. 2019. URL: https://gamedev.stackexchange.com/questions/163596/how-to-randomly-create-a-fair-maze-for-a-multiplayer-game.

[5] Jaco Fourie. *Brick Wall GitHub Source Code*. Last accessed 04. May 2025. 2020. URL: https://github.com/Jaco7Fourie/BrickWall/tree/master.

[6] Sebastian Kleber und Simon Völkl. *Catastrophe Simulator*. Last accessed 13. May 2025. 2025. URL: https://github.com/s-voelkl/Catastrophe-Simulator.

[7] Nacer Kroudir. *Maze Generation*. Last accessed 07. May 2025. 2023. URL: https://github.com/NacerKROUDIR/MazeGeneration/tree/main.

[8] Matplotlib. *Using Matplotlib*. Last accessed 11. May 2025. 2025. URL: https://matplotlib.org/stable/users/index.html.

[9] Mesa. *Boltzmann Wealth Model*. Last accessed 05. May 2025. 2025. URL: https://mesa.readthedocs.io/en/stable/examples/basic/boltzmann_wealth_model.html.

[10] Mesa. *Mesa Documentation*. Last accessed 05. May 2025. 2025. URL: https://mesa.readthedocs.io/en/stable/.

[11] Pranjal Mittal. *Plotting networkx graph with node labels defaulting to node name*. Last accessed 09. May 2025. 2015. URL: https://stackoverflow.com/questions/28533111/plotting-networkx-graph-with-node-labels-defaulting-to-node-name.

[12] Muhammad Ahsan Naeem. *Pyamaze GitHub Source Code*. Last accessed 05. May 2025. 2025. URL: https://github.com/MAN1986/pyamaze.

[13] NetworkX. *Labeled 2D Grid*. Last accessed 09. May 2025. 2018. URL: https://networkx.org/grave/latest/gallery/plot_grid.html.

[14] NetworkX. $shortest_path_length$. Last accessed 08. May 2025. 2025. URL: https://networkx.org/documentation/stable/reference/algorithms/generated/networkx.algorithms.shortest_paths.generic.shortest_path_length.html.

[15] Pandas. *Getting Started*. Last accessed 11. May 2025. 2024. URL: https://pandas.pydata.org/getting_started.html.

[16] Seaborn. *Pairwise correlations*. Last accessed 12. May 2025. 2024. URL: https://seaborn.pydata.org/archive/0.11/examples/many_pairwise_correlations.html.

[17] Seaborn. *User guide and tutorial*. Last accessed 11. May 2025. 2024. URL: https://seaborn.pydata.org/tutorial.html.

[18] tylerthemiler. *savefig outputs blank image*. Last accessed 11. May 2025. 2012. URL: https://stackoverflow.com/questions/9012487/savefig-outputs-blank-image.
[19] Wikipedia. *A*-Algorithmus*. Last accessed 09. May 2025. 2025. URL: https://de.wikipedia.org/wiki/A*-Algorithmus.
[20] Wikipedia. *Manhattan-Metrik*. Last accessed 09. May 2025. 2025. URL: https://de.wikipedia.org/wiki/Manhattan-Metrik.
[21] Wikipedia. *Maze generation algorithm*. Last accessed 08. May 2025. 2025. URL: https://en.wikipedia.org/wiki/Maze_generation_algorithm.