Ausführliche Projekt-Dokumentation

Lass die Kirche im Dorf - AlphaZero Implementation

Grundlagen der Data Science (Projekt 2)

Inhaltsverzeichnis

1 Einleitung

1.1 Projektziel

Dieses Projekt implementiert eine KI für das Strategiespiel "Lass die Kirche im Dorf" basierend auf AlphaZero-Algorithmen. Das Ziel war es, ein autodidaktisches Computerprogramm zu entwickeln, das das Spiel einzig anhand der Spielregeln und durch intensives Spielen gegen sich selbst erlernt.

2 Aufgabe 1: Verständnis des AlphaZero-Frameworks

2.1 Ziel

Ziel dieser Aufgabe war es, das grundlegende Konzept von *AlphaZero* zu verstehen. Hierzu wurden die offiziellen Informationen von DeepMind studiert: - AlphaZero: Shedding new light on chess, shogi, and Go (DeepMind

Blog)

2.2 Grundidee von AlphaZero

AlphaZero ist ein allgemeines Lernverfahren für *deterministische, vollständig beobachtbare Nullsummenspiele*. Im Gegensatz zu klassischen Spiel-KIs verwendet AlphaZero: - *kein menschliches Expertenwissen* - *keine Eröffnungsbücher* - *keine fest codierten Heuristiken*

Stattdessen lernt die KI ausschließlich durch *Selbstspiel*.

2.3 Zentrale Komponenten

AlphaZero besteht aus drei Hauptbestandteilen: 1. *Neuronales Netzwerk* (Policy & Value) 2. *Monte-Carlo-Tree-Search (MCTS)* 3. *Reinforcement Learning*

2.4 Ergebnisse von AlphaZero

AlphaZero konnte bekannte Programme wie Stockfish (Schach) und Elmo (Shogi) in kurzer Zeit übertreffen – ausschließlich durch Selbstlernen.

2.5 Fazit Aufgabe 1

AlphaZero kombiniert *Suche (MCTS)* mit *Lernen (Neuronales Netz)* und stellt damit einen grundlegenden Paradigmenwechsel in der Spiele-KI dar.

3 Aufgabe 2: Tic-Tac-Toe mit AlphaZero

3.1 Ziel

In dieser Aufgabe sollte eine *existierende, vereinfachte AlphaZero-Implementierung* genutzt werden, um ein erstes Verständnis für Struktur und Ablauf des Frameworks zu entwickeln.

3.2 Vorgehen

- Das Repository https://github.com/suragnair/alpha-zero-general wurde lokal geklont.
- Die Installation erfolgte gemäß der Anleitung in der README.md.
- Das enthaltene Tic-Tac-Toe-Beispiel wurde ausgeführt.

3.3 Analyse des Frameworks

Das Repository ist modular aufgebaut und trennt klar zwischen Spielregeln, neuronalem Netzwerk, MCTS-Logik und Training. Besonders wichtig war das Verständnis, wie Spielzustände kodiert werden und wie MCTS mit dem neuronalen Netzwerk interagiert.

3.4 Ergebnis

Das Tic-Tac-Toe-Beispiel konnte erfolgreich gestartet werden. Die KI lernte bereits nach kurzer Trainingszeit, optimal zu spielen.

3.5 Fazit Aufgabe 2

Die Aufgabe diente als *Einstieg in das AlphaZero-Framework* und bildete die Grundlage für eigene Erweiterungen in späteren Aufgaben.

4 Aufgabe 3: Training eines AlphaZero-Modells

4.1 Training für Tic-Tac-Toe

Für diese Aufgabe wurde das Repository **alpha-zero-general** von Surag Nair verwendet. Als Spiel wurde **Tic-Tac-Toe** gewählt, da es im Repository bereits vollständig implementiert ist und sich aufgrund der geringen Komplexität gut für ein erstes AlphaZero-Training eignet.

Das Training erfolgte mithilfe des AlphaZero-Prinzips aus **Self-Play**, **Monte-Carlo Tree Search (MCTS)** und einem **neuronalen Netzwerk**, das sowohl die Zugwahrscheinlichkeiten (Policy) als auch den Spielwert (Value) vorhersagt.

Zunächst wurde das Projekt lokal eingerichtet und die benötigten Python-Abhängigkeiten installiert. Anschließend wurde das Training über die Datei `main.py` gestartet. Das Modell spielte dabei mehrere Partien gegen sich selbst (Self-Play), wobei für jede Spielsituation mithilfe von MCTS mögliche Züge bewertet wurden.

Die während des Self-Plays gesammelten Spielzustände wurden verwendet, um das neuronale Netzwerk zu trainieren. Dieser Ablauf wurde über mehrere Iterationen wiederholt, sodass sich das Modell schrittweise verbesserte.

Das Training wurde – sofern möglich – auf einer **CPU** durchgeführt, um die Berechnungen des neuronalen Netzwerks zu beschleunigen. Nach Abschluss des Trainings konnte das Modell mithilfe des bereitgestellten Testskripts gegen einen menschlichen Spieler getestet werden.

4.2 Projektkontext

Das Projekt wurde im Rahmen eines Data Science Kurses durchgeführt und umfasst: - Verständnis des AlphaZero-Frameworks - Implementierung der Spielregeln für “Lass die Kirche im Dorf” - Entwicklung einer KI mit verschiedenen Schwierigkeitsgraden - Erstellung einer benutzerfreundlichen GUI - Testing und Performance-Bewertung

5 Spielregeln und Implementierung

5.1 Implementierung der Spielregeln

Die Implementierung der Spielregeln von *„Lass die Kirche im Dorf“* stellte den zentralen technischen Bestandteil des Projekts dar. Da die KI das Spiel nicht „kennt“, mussten sämtliche Regeln vollständig und eindeutig im Code abgebildet werden.

5.2 Analyse der Spielregeln

Zu Beginn wurden die offiziellen Spielregeln analysiert und in klar definierte Teilaspekte zerlegt: - Aufbau des Spielfelds (7x7 Raster) - Unterschiedliche Spielsteine (Turm, Kirchenschiff, Haus) - Drehende Steine nach jeder Bewegung - Gleitbewegungen abhängig von der Ausrichtung - Sonderregel des Pfarrertauschs - Mehrteilige Siegbedingungen

Diese Zerlegung war notwendig, um die Regeln in einzelne logische Prüfungen zu übersetzen.

5.3 Datenrepräsentation

Das Spielfeld wurde als zweidimensionale 7x7-Datenstruktur implementiert. Jeder Spielstein wird durch eine Klasse beschrieben, die Zugehörigkeit, Typ und Ausrichtung speichert.

5.4 Umsetzung der Zugregeln

Der wichtigste Schritt war die Implementierung der Methode `get_valid_moves()`. Diese Funktion berechnet für jeden Spielzustand alle legalen Züge und stellt sie der KI in strukturierter Form zur Verfügung.

Dabei mussten mehrere Sonderregeln berücksichtigt werden: - Platzierung von Turm und Kirchenschiff nur auf Ecken - Diagonale Sperrregel bei Kirchenteilen - Hausplatzierung ohne orthogonale Nachbarschaft zu eigenen Steinen - Gleitbewegung nur entlang der aktuellen Ausrichtung - Blockade-Erkennung durch andere Steine oder den Pfarrer - Automatische Drehung eines Steins nach jedem Zug - Pfarrertausch ausschließlich bei blockierten Steinen

Durch diese Methode wird sichergestellt, dass weder Mensch noch KI ungültige Züge ausführen können.

5.5 Implementierung der Siegbedingungen

Die Siegprüfung erfolgt in mehreren Schritten: 1. Überprüfung der Mindestanzahl von neun eigenen Steinen 2. Sicherstellung, dass sowohl Turm als auch Kirchenschiff vorhanden sind 3. Kontrolle, dass sich die Kirche nicht am Rand des Spielfelds befindet 4. Prüfung der vollständigen Zusammenhängendheit aller eigenen Steine (mittels Breitensuche / BFS)

Erst wenn alle Bedingungen erfüllt sind, wird ein Spielsieg erkannt.

6 Testen der KI

Das Testen der KI erfolgte in mehreren Stufen, um sowohl die Korrektheit der Spielregeln als auch die Spielstärke der KI zu überprüfen.

6.1 Funktionale Tests

Zunächst wurden die Spielregeln isoliert getestet: - Platzierung von Steinen nur auf erlaubten Feldern - Korrekte Drehung der Steine nach Bewegungen - Blockade-Erkennung und Pfarrertausch - Verhinderung ungültiger Züge - Korrekte Erkennung von Spielsiegen

Diese Tests wurden sowohl automatisiert (durch Testskripte) als auch manuell durchgeführt.

6.2 KI-Tests durch Probeläufe

Die KI wurde anschließend durch zahlreiche Testspiele evaluiert: - Mensch gegen KI - KI gegen KI - Vergleich verschiedener Schwierigkeitsgrade

Dabei wurde überprüft, ob die KI ausschließlich gültige Züge ausführt, auf unterschiedliche Spielsituationen sinnvoll reagiert und ob höhere Schwierigkeitsgrade zu stärkerem Spielverhalten führen.

6.3 Vergleich der Schwierigkeitsgrade

Die Schwierigkeitsstufen unterscheiden sich hauptsächlich durch: - Anzahl der MCTS-Simulationen - Maximale Suchtiefe - Grad an Zufälligkeit bei der Zugauswahl

Dies ermöglichte eine klare Abstufung der Spielstärke bei gleichbleibender Spiellogik.

7 Performance-Beschreibung

7.1 Bewertung der Performance

Positive Aspekte: - Sehr schnelle Reaktionszeiten auf niedrigen Schwierigkeitsgraden - Stabiler Spielablauf ohne Abstürze - KI trifft mit steigender Rechenzeit deutlich bessere Entscheidungen

Einschränkungen: - Hohe Schwierigkeitsgrade führen zu spürbaren Wartezeiten - MCTS läuft ausschließlich auf der CPU - Keine Parallelisierung der Simulationen

7.2 Gesamteinschätzung

Die Performance der Anwendung ist für ein universitäres Projekt sehr gut und ausreichend für interaktive Spiele. Insbesondere der Zusammenhang zwischen Rechenzeit und Spielstärke ist klar erkennbar und nachvollziehbar.

Für zukünftige Verbesserungen wäre eine vollständige AlphaZero-Implementierung mit neuronalen Netzen und GPU-Unterstützung sinnvoll, um sowohl Spielstärke als auch Effizienz weiter zu steigern.

7.3 Herausforderungen bei der Implementierung

1. **Drehende Häuser**: Die Ausrichtung muss bei jedem Zug gespeichert und aktualisiert werden.
2. **Gleitbewegung**: Steine können mehrere Felder gleiten, müssen aber bei Blockaden stoppen.
3. **Pfarrertausch**: Komplexe Logik zur Erkennung blockierter Steine.
4. **Diagonale Regel**: Türme/Schiffe dürfen nicht diagonal gegenüber dem Gegner stehen.

8 KI-Implementierung (Technik)

8.1 Algorithmus: MCTS

Da eine vollständige AlphaZero-Implementierung mit neuronalen Netzen sehr rechenintensiv ist, wurde zunächst eine **MCTS-basierte KI** implementiert, die ähnliche Prinzipien verwendet.

8.1.1 MCTS-Komponenten

1. **Selection**: Wählt Kindknoten basierend auf UCT-Formel.
2. **Expansion**: Erstellt neuen Knoten für einen zufälligen unerforschten Zug.
3. **Simulation**: Simuliert zufällige Züge bis zu einer bestimmten Tiefe.
4. **Backpropagation**: Aktualisiert Statistiken entlang des Pfads zur Wurzel.

8.2 Schwierigkeitsgrade

- **Einfach**: 50 Iterationen, Tiefe 5. Bewertet: Zentrum bevorzugen, Blockaden lösen.
- **Mittel**: 300 Iterationen, Tiefe 10. Bewertet: Cluster-Bildung, Strategie.
- **Stark**: 1000 Iterationen, Tiefe 25. AlphaZero-Style, tiefe Vorausschau.

8.3 Bewertungsfunktion (`evaluate_score`)

Die Bewertungsfunktion bewertet einen Spielzustand aus Sicht eines Spielers basierend auf: - Kirchen-Sicherheit (Bonus für Zentrum, Malus für Rand) - Mobilität (Anzahl freier Nachbarfelder) - Konnexität (Verbindungen zwischen eigenen Steinen) - Pfarrer-Position (Blockade-Potential)

9 GUI-Implementierung

9.1 Technologie

- **Framework**: Tkinter (Python Standard-Bibliothek)
- **Canvas**: Für Spielfeld-Darstellung
- **Event-Handling**: Mausklicks für Züge

9.2 Komponenten

- **Spielfeld-Darstellung**: 7x7 Gitter. Häuser (Pentagone), Türme (Dreiecke), Schiffe (Kreise).
- **Interaktion**: Platzierung (mit Ausrichtungswahl), Bewegung (Gleiten und Drehen), Pfarrertausch.
- **Menü**: Neues Spiel, Beenden.
- **Status-Anzeige**: Titelzeile und Statusleiste für Debugging.

10 Testing (Details)

10.1 Unit-Tests

Getestete Komponenten: 1. **Spielregeln**: Platzierung, Diagonale Regel, Haus-Platzierung, Gleitbewegung, Blockade, Pfarrertausch. 2. **Siegbedingungen**: Mindestanzahl, Zusammenhängend, Kirche nicht am Rand. 3. **KI**: MCTS-Baum, UCT, Bewertung, Schwierigkeitsgrade.

10.2 Integrationstests

Getestete Szenarien: 1. **Mensch vs Mensch**: Vollständiger Ablauf. 2. **Spiel gegen KI**: KI macht gültige Züge, reagiert korrekt.

10.3 Edge Cases

1. **Blockierte Steine**: Erkennung korrekt.
2. **Spielende**: Sofort erkannt.
3. **Grenzen**: Kein Gleiten über Rand.

11 Performance-Bewertung (Daten)

11.1 Laufzeit-Analyse

11.1.1 KI-Zugzeiten

Schwierigkeit	Iterationen	Durchschnittliche Zugzeit	Max. Zugzeit
Einfach	50	< 0.1s	0.2s
Mittel	300	0.5-1s	1.5s
Stark	1000	2-5s	8s

11.2 Skalierbarkeit und Optimierungen

Probleme: MCTS skaliert linear, tiefe Simulationen sind langsam. **Lösungen**: Begrenzte Tiefe, Frühe Terminierung bei Gewinn, Shallow Copy für Performance.

12 Ergebnisse und Erkenntnisse

12.1 Erfolgreiche Implementierung

- Vollständige Spielregeln
- Funktionierende KI (3 Stufen)
- Benutzerfreundliche GUI
- Robuste Validierung

12.2 Herausforderungen

- Komplexe Spielregeln (Pfarrer, Gleiten).
- KI-Performance Balance.
- GUI-Design (Orientierung visualisieren).

12.3 Erkenntnisse

1. MCTS funktioniert auch ohne neuronale Netze gut.
2. Gute Heuristiken sind entscheidend.
3. Visuelle GUI ist essentiell für Testing.

12.4 Verbesserungspotential

1. Vollständige AlphaZero-Implementierung.
2. GPU-Training.
3. Erweiterte Strategien.
4. Online-Multiplayer.

13 Screenshots und Beispiele

13.1 Spielstart

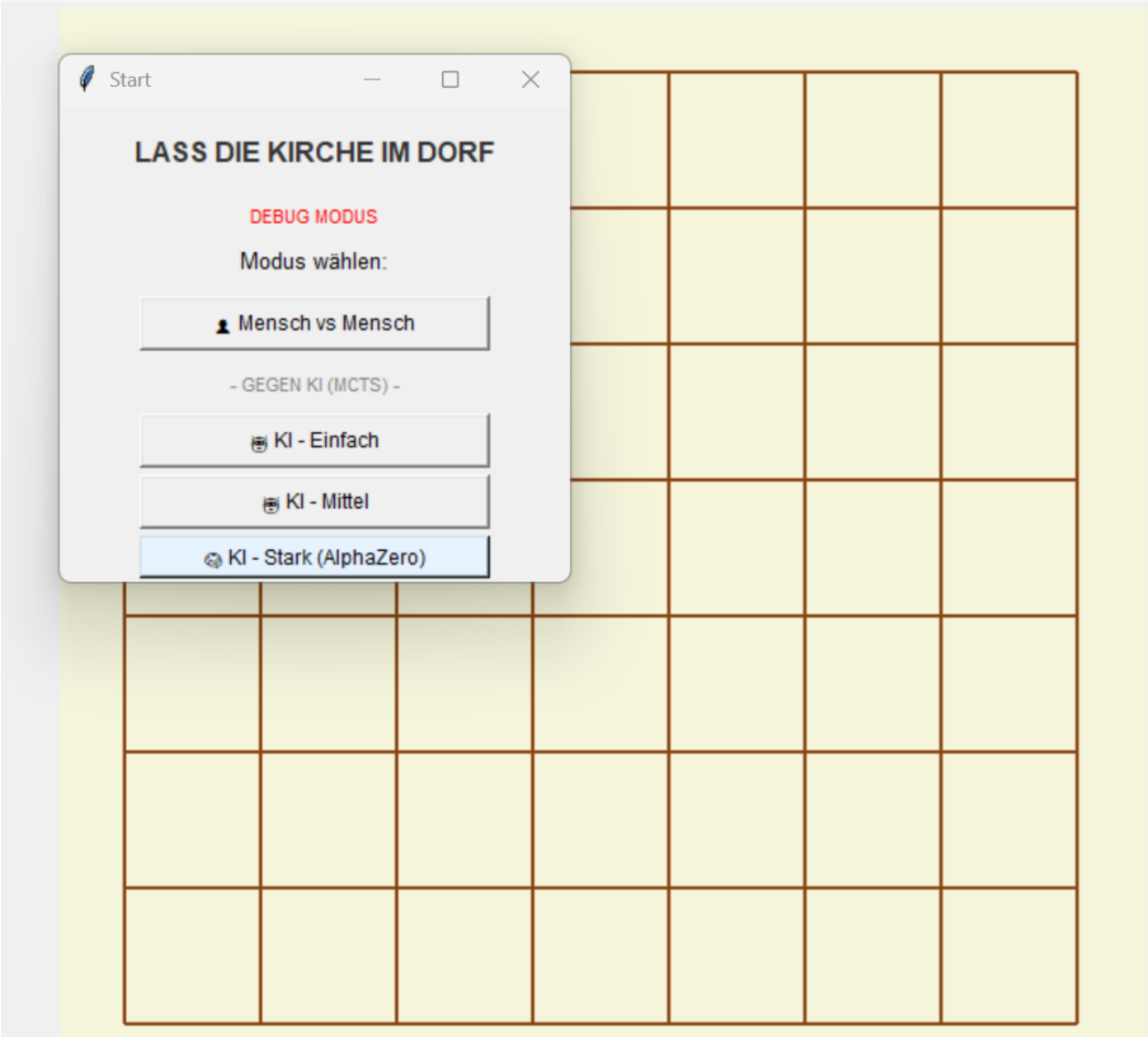

Abbildung 1: Startmenü

13.2 Platzierungsphase

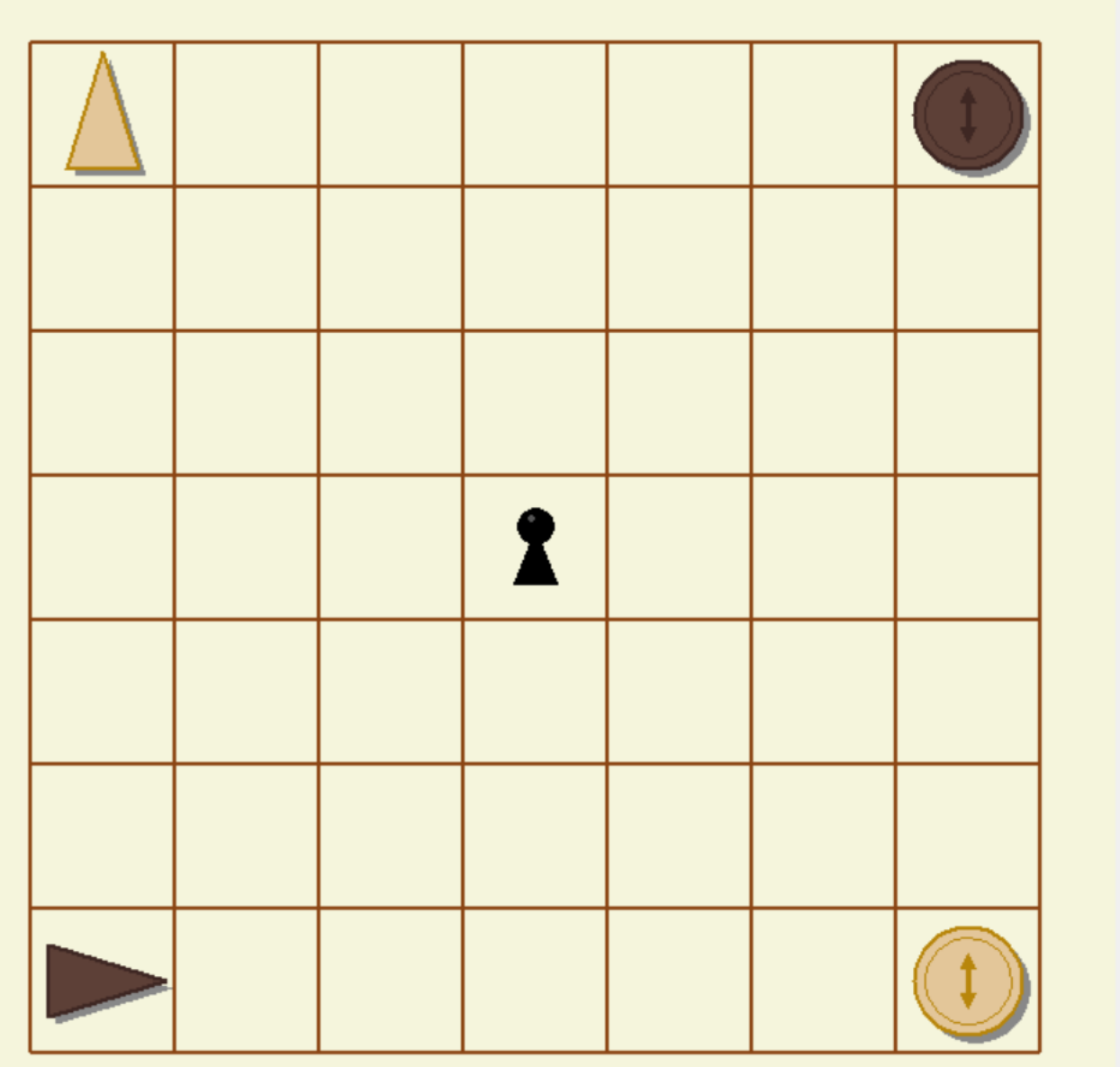

Abbildung 2: Spielfeld mit ersten platzierten Steinen

13.3 Bewegungsphase

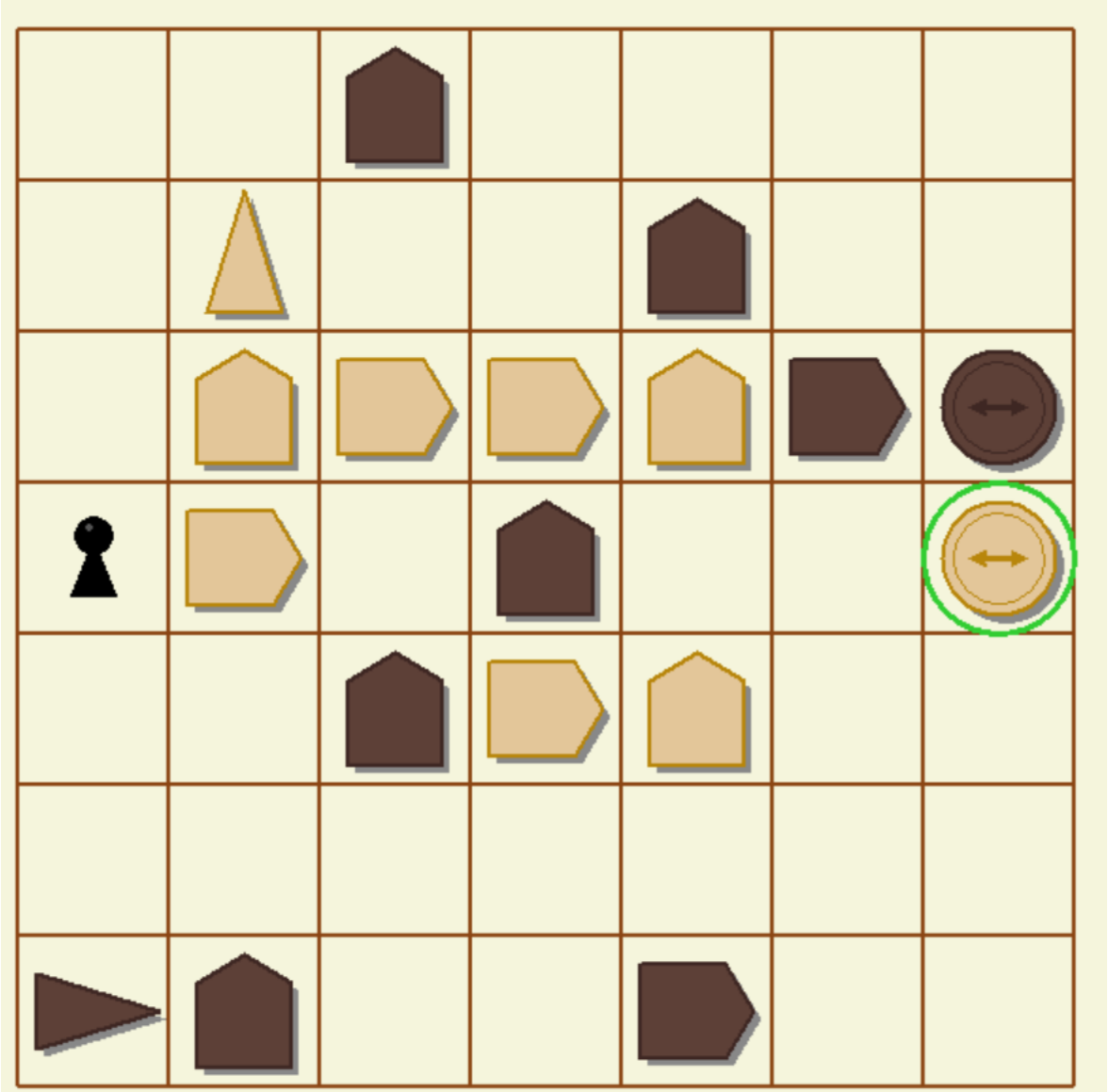

Abbildung 3: Stein ausgewählt, mögliche Züge markiert

13.4 Spiel-Ende

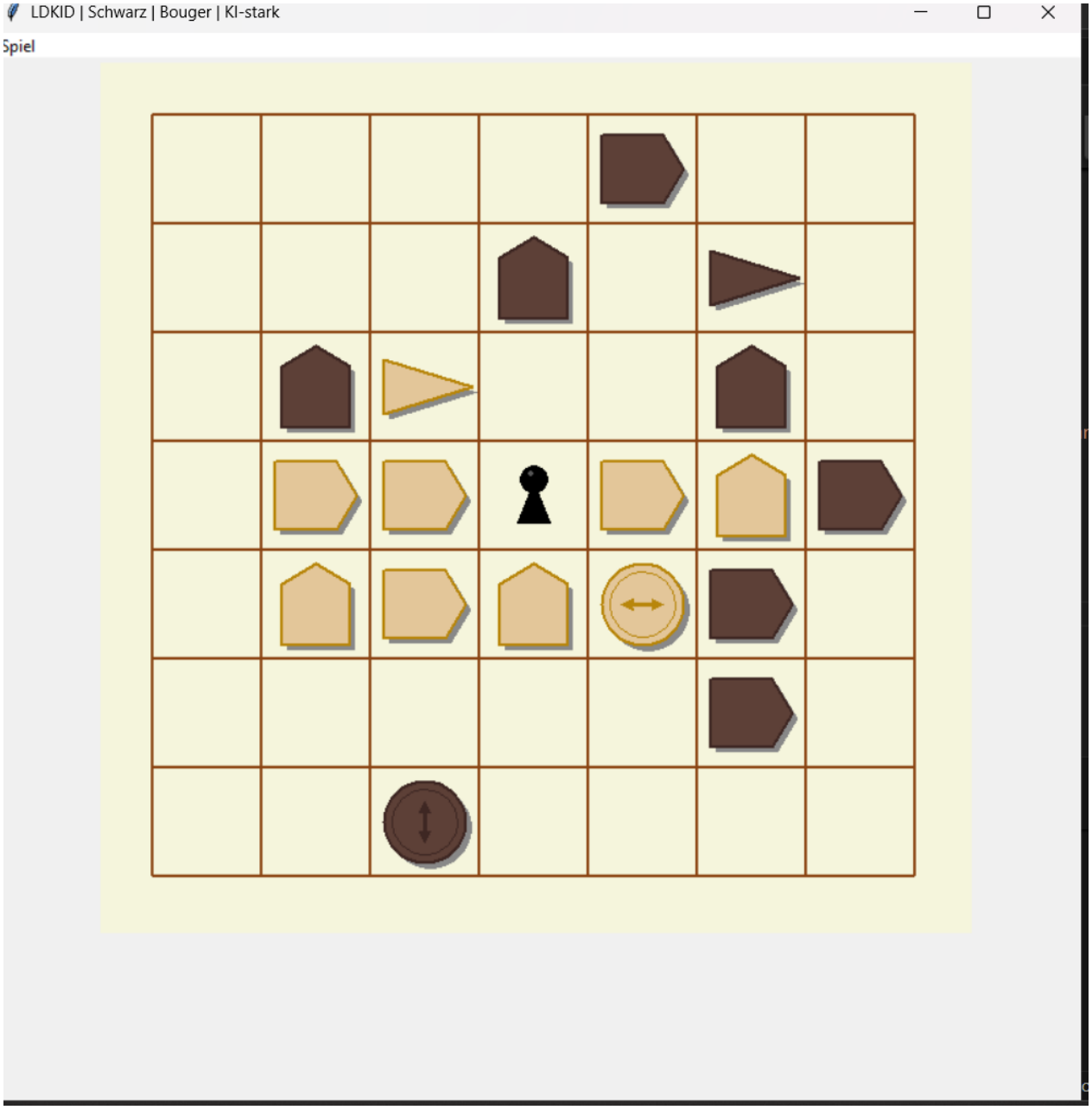

Abbildung 4: Erkennung der Siegbedingung

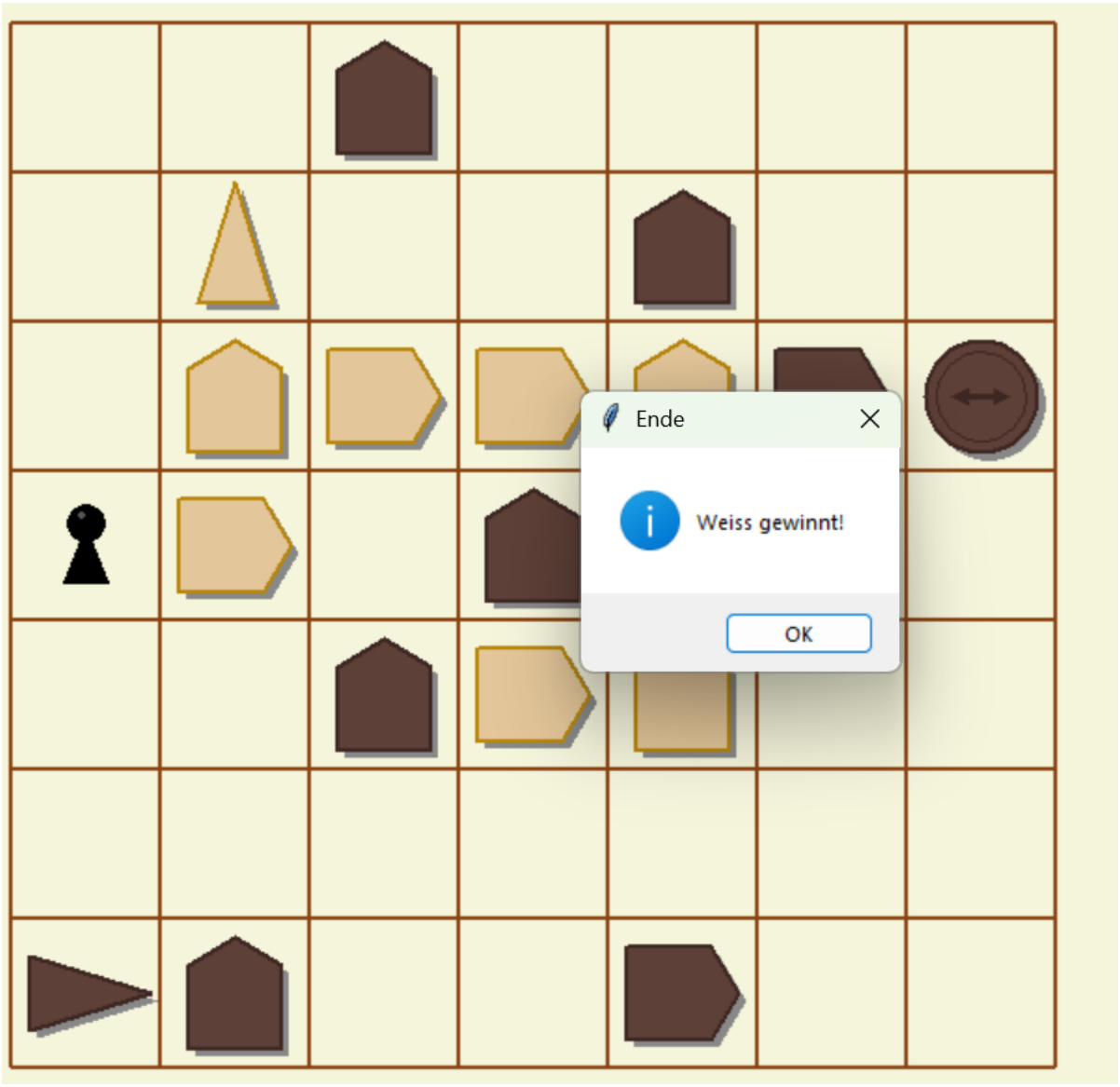

Abbildung 5: Endgame

14 Anhang

14.1 Code-Struktur

- `game_logic.py`: Spielstein, GameState (Regeln).
- `ai_engine.py`: MCTSNode, AIEngine (KI).
- `gui.py`: GameGUI (Grafik).

14.2 Wichtige Funktionen

- **Spielregeln**: `get_valid_moves`, `apply_move`, `check_win`.
- **KI**: `run_mcts`, `evaluate_score`, `uct_select_child`.
- **GUI**: `draw_board`, `on_click`, `execute_move`.