

Entwicklungs bereich KIP

KI-Gestütztes Programmieren

Erstellt von: Adrian Meier

Zuständiger Fachexperte: Moritz Kirschmann

Inhalt

Einleitung.....	3
Anwendungsszenario und Entwicklung.....	3
Grundidee.....	3
Entwicklung des Szenarios.....	3
Entwicklungsstrategien.....	4
Anfang	4
Erste Probleme	6
Vergleich mit anderen Projekten.....	7
Strukturierte Analyse.....	8
Änderung des RL.....	9
Optimierung und Verständnis des Bestehenden.....	10
Weitere Features Hinzufügen.....	11
Automatisches Spielen mit Features erweitern	12
Finalisierung	13
Verständnis und Ownership	14
Grundaufbau des Minesweepers	14
Grundstruktur Reinforcement Learning	14
Erweiterung des Solvers	14
Hinzufügen der Features	14
Tests und Dokumentation.....	14
Methoden zur Herstellung von Verständnis.....	15
Reflexion über Ownership	15
Qualitätssicherung.....	15
Fazit	16
Reflexion	16
Empfehlungen für andere Studis.....	16

Einleitung

Ziel dieses Projekts ist es, mithilfe von Cursor ein Projekt zu realisieren. Dabei war das Projekt frei wählbar, sollte aber eine gewisse Grösse und Komplexität aufweisen. In meinem Projekt war das Kernziel, ein Minesweeper zu erstellen und es mit einem Reinforcement-Learning-Programm (RL) zu verknüpfen.

Am Ende des Projekts hatte ich ein Minesweeper mit diversen Basiseinstellungen sowie einigen zusätzlichen Features, die das Spiel abwechslungsreicher gestalten. Zudem ist ein Modell integriert, das das Spiel lernen kann und anschliessend selbst spielen kann.

Das Projekt ist in einem öffentlichen GitHub-Repository unter diesem Link zu finden. Um das Projekt auszuführen, verweise ich auf die Datei README.md im Repository. Dort ist eine Anleitung zur Inbetriebnahme sowie eine Übersicht über die wichtigsten Funktionen und Features zu finden.

Anwendungsszenario und Entwicklung

Grundidee

Wie bereits in der Einleitung erwähnt, ist das Ziel die Entwicklung eines Minesweepers. Im Mittelpunkt steht zunächst das Spiel selbst, welches in einer GUI spielbar sein soll. Anschliessend soll ein Reinforcement Learning Programm in der Lage sein, das Spiel zu spielen und zu gewinnen.

- Neben diesem Hauptziel habe ich mir noch weitere Features überlegt. Diese sollten zusätzlich implementiert werden, wenn das Hauptziel schnell und sauber erstellt werden konnte. Das Spiel selbst könnte zusätzliche Funktionen erhalten, wie z.B. ein Radar, um bei unklaren Situationen zu helfen.
- Die Grösse des Spielfeldes könnte angepasst werden, um zu sehen, ob sich das RL-Modell dadurch anders verhält, insbesondere ohne zusätzliches Training.
- Der Dimensionsraum des Minesweepers könnte um eine Dimension erweitert werden, also in den dreidimensionalen Raum. Dadurch würden sich einige Mechaniken ändern. Auch hier wäre es das Ziel, dies zuerst für den Handbetrieb und dann, wenn genügend Zeit bleibt, für das RL umzusetzen.

Entwicklung des Szenarios

Es hat sich sehr schnell gezeigt, dass eine Anpassung der Feldgrösse sinnvoll ist. Das RL-Modell benötigt auf grösseren Feldern deutlich mehr Zeit und es wurde schnell klar, dass das Programm die Kernlogiken noch nicht verstanden hatte (mehr dazu in den Entwicklungsstrategien). Deshalb wurde dieses Nebenziele vor dem Hauptziel (erfolgreiches RL) umgesetzt.

Im Laufe des Projekts stellte sich heraus, dass es im Fall von Minesweeper sehr schwierig ist, ein reines RL-Modell zu erstellen. Es wurden einige Anpassungen vorgenommen, damit das Programm das Spiel lösen kann.

In den Spielerweiterungen wurden einige kleinere Funktionen hinzugefügt. Mehr dazu im entsprechenden Kapitel des Entwicklungsberichts.

Aus Zeitgründen, gerade weil die Implementierung des Reinforcement Learning Systems sehr lange gedauert hat, habe ich mich entschieden, die Dimensionserweiterung nicht zu implementieren.

Entwicklungsstrategien

Anfang

Das Projekt ist gut angelaufen. Zu Beginn habe ich einen Prompt erstellt, indem ich im Planungsmodus von Cursor versucht habe, mein Projekt zu strukturieren und die Grenzen und Mittel abzustecken (Prompt 1). Ich selbst habe keine grosse Erfahrung mit GUIs und war froh, dass der Agent mir einige Fragen zu den Bibliotheken stellte, mit denen ich arbeiten wollte, und mir einige Fragen zu meinem Konzept stellte, wie ich das umsetzen wollte.

User

Ich will einen Blueprint für ein Projekt haben. Erstelle mir bitte eine Schritt für Schritt Anleitung, welches ich im Anschluss als Prompts eingeben kann um Punkt für Punkt zu meinem Programm zu kommen. Die Idee und das Konzept sind in der PDF Datei angehängt. @Konzept KIP.pdf Im ersten Blueprint will ich nur die Hauptziele haben. Also das Programm sowie den GUI für das Minesweeper und das Reinforcement learning Programm. Wichtig ist dabei, dass das Minesweeper auch manuell gespielt werden kann. Frage mich bei der Erstellung bei jedem Punkt jeweils, in welcher Form ich das gerne hätte.

Prompt 1: Initialer Prompt

Nachdem die Planung schnell abgeschlossen war und die offenen Fragen des Agenten beantwortet waren, liess ich ihn die Programme für den Minesweeper und die RL schreiben. Nach einigen kleinen Fehlern, die aber schnell behoben waren, hatte ich zum ersten Mal eine funktionierende GUI. Es hatte noch kein Auto Fill, also ein automatisches Aufdecken von Feldern die nicht an einer Mine liegen und auch die erste Aktion konnte noch eine Mine treffen und somit das Spiel direkt beenden. Aber auch dies konnte durch eine einfache Anweisung an den Agenten (Prompt 2) relativ schnell korrigiert werden.

User

Beim Manuellen spielen fallen noch mehrere Fehler auf, bitte Korrigiere diese:

Das auto Reveal safe funktioniert nicht, das zeigt das manuelle spielen sowie die Test funktion die ebenfalls Failed.

Wenn eine Bombe aufgedeckt wird, ändert sich visuell nichts, nur die Zeit Stoppt und man kann keine neuen Felder mehr anklicken.

Prompt 2: Auto Reveal und erstes Feld keine Bombe

Nach einigen Stunden, ein wenig Planung und ohne grosses Verständnis für die Materie konnte ich also meine ersten Runden Minesweeper spielen. Das Layout selbst ist zwar noch nicht perfekt, aber alle Kernelemente des Spiels funktionieren und das Spiel war spielbar (Abbildung 1).

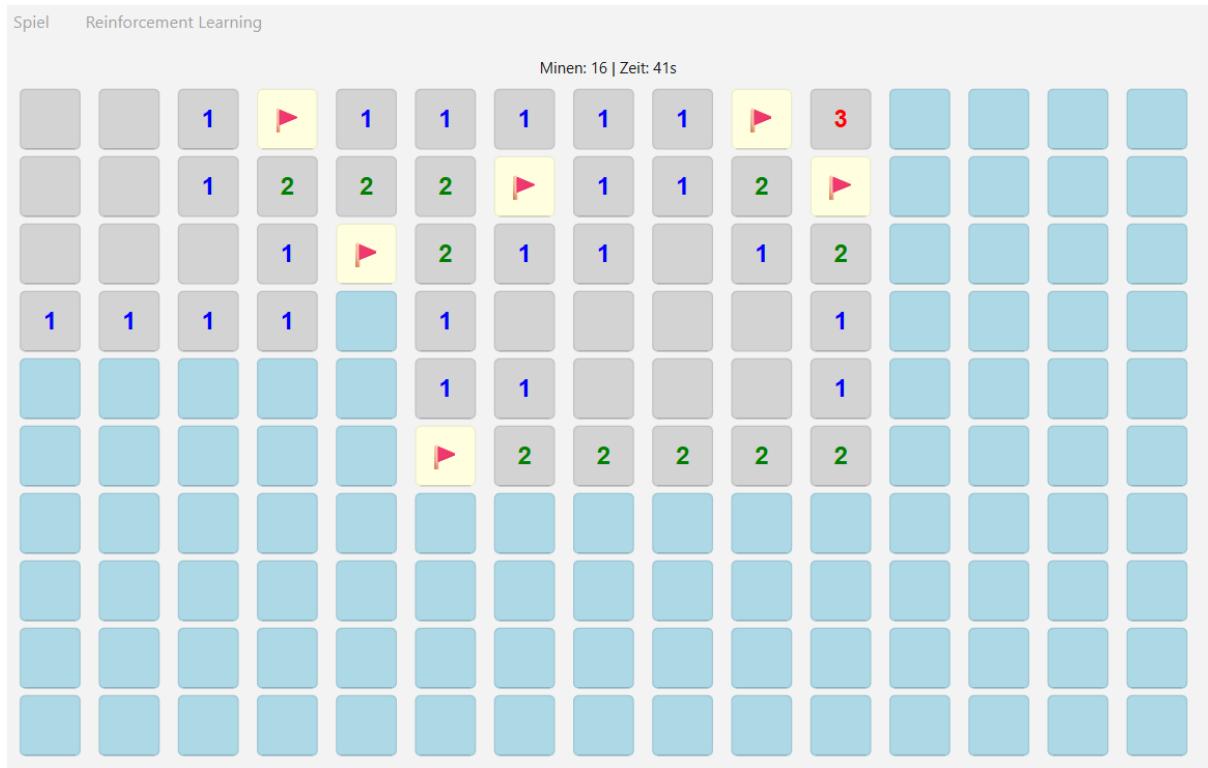


Abbildung 1: Erste Spiele

Ich merkte schnell, dass ich zum Spielen und auch zum Beobachten der RL wahrscheinlich ein kleineres Feld brauche, mein aktuelles war zu gross. Auch dies wurde vom Agenten sehr schnell und effizient angepasst und ich konnte nun im Menü einfache Änderungen der Feldgrösse und des Schwierigkeitsgrades, was nichts anderes als die Anzahl der Minen ist, einstellen.

Erste Probleme

Da bisher alles schnell und problemlos funktionierte, ging ich mit viel Zuversicht und Vertrauen an das Reinforcement Learning heran. Zuerst hatte ich einige kleine Fehlermeldungen, die ich ohne weitere Informationen als Prompt an den Agenten gab, der die Probleme löste.

Dann war es Zeit für den ersten Durchlauf, beim Training gab es nach jeweils 100 Episoden einen Output. Die Belohnung war hoch und die Erfolgsquote 0, aber am Ende des Trainings zeigte das Programm eine Erfolgsquote von 70% an. Beim visuellen Test stellte sich heraus, dass diese Zahl falsch war. Ich konnte diesen Fehler schnell beheben, aber der RL lernte die Spielmechanik nicht.

Da es bei Minesweeper so gut funktioniert hatte, gab ich dem Agenten zunächst nur die Ausgaben der Episoden und zusätzlich meine eigenen Beobachtungen aus den Spielen des RL-Programms. Beim Durchsuchen des Programms schien er dann den Fehler klar erkannt zu haben und änderte daraufhin einige Parameter. Jedoch änderte sich dadurch nicht das Verhalten des RL. Es machte immer noch klare Fehler und hatte eine Win-Rate von 0. Bei einfachsten Aufgaben wie in Abbildung 2 löste er eine Mine aus.

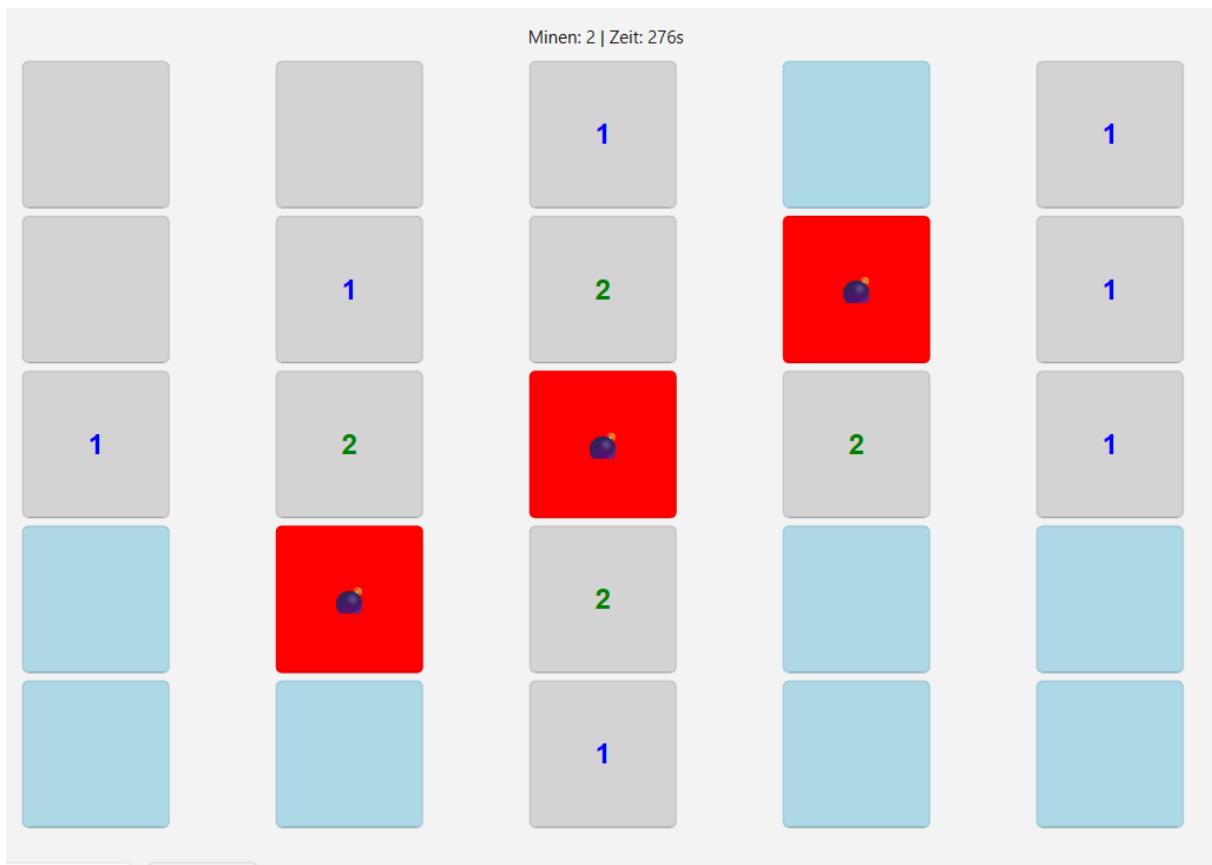


Abbildung 2: Leichte Felder können von RL nicht gelöst werden

Vergleich mit anderen Projekten

Zuerst erstellte ich einen neuen Agenten, das Kontextfenster des alten war inzwischen gefüllt und die ältesten Nachrichten aus dem Chat waren nur noch als Zusammenfassung gespeichert.

Ich erklärte dem neuen Agenten mein Ziel und gab ihm Aufgaben. Er sollte sich den gesamten Code des Repositories ansehen und sich von einem Github inspirieren lassen, der einen RL-Algorithmus zur Lösung von Minesweeper (Prompt 3) programmiert hat. Ausserdem habe ich das zu verwendende Sprachmodell auf den ChatGPT 5.1 Codex gelegt, um selbst ein wenig zu beobachten, wie sich die verschiedenen Sprachmodelle verhalten.

User

Ich probiere ein Minesweeper-Spiel mit einem reinforcement_learning programm zu kombinieren. Schaue bitte alle Programme an und verbessere ds RL programm, es lernt aktuell nicht korrekt. Nutze dabei:
<https://sdlee94.github.io/Minesweeper-AI-Reinforcement-Learning/> Als inspiration

Prompt 3: Optimierungsversuch mit Externer Resource

7. Zusammenfassung

Das RL-System implementiert:

Environment Wrapper: Gymnasium-ähnliches Interface **State Encoding:** Normalisierte 2D-
Repräsentation **DQN Network:** CNN für räumliche Features **Experience Replay:** Stabilisiert Training
Target Network: Verhindert instabile Updates **Epsilon-Greedy:** Exploration/Exploitation Balance
Reward Shaping: Sparse + Shaped Rewards **Action Masking:** Verhindert ungültige Aktionen

Das System ist vollständig funktionsfähig und bereit für Training!

Prompt 4: Output des Agenten

Bei den bisherigen Antworten hatte ich oft das Gefühl, dass der Agent jeweils eine Mischung aus verschiedenen RL-Anwendungen genommen hat und versucht hat, diese auf meinen Fall anzuwenden. Minesweeper hat jedoch einige klare Regeln, die zwar nicht direkt an den RL übergeben werden können, aber durch die Art und Grösse der Reward-Ausschüttung verändert werden können. Ausserdem kann auch die verwendete Grösse im Netzwerk entscheidend sein. Ich hatte die Hoffnung, dass ich den Agenten mit einem Link zu GitHub mehr in die richtige Richtung lenken könnte.

Zunächst sah die neue Strategie vielversprechend aus, einige Details wurden angepasst und Parameter geändert oder hinzugefügt. Der Agent erkannte die Unterschiede zwischen dem GitHub-Repository und meinem Code und passte sie an. Nach allen Ergänzungen habe ich das Programm erneut ausgeführt. Die Ergebnisse blieben leider die gleichen. Auch hier versuchte ich mit einigen Prompts und dem erneuten Verweis auf das andere Projekt eine saubere RL zu erhalten. Aber auch hier ohne Erfolg.

Strukturierte Analyse

Da die Antworten immer verwirrender wurden und der Ansatz immer weniger Wirkung zeigte, begann ich mit einem neuen Agenten zu arbeiten. Dieses Mal wollte ich mit Claude Sonnet arbeiten. Ich nannte ihm wieder mein Ziel als Prompt und gab ihm die Aufgabe, meinen Code zu lesen. Ausserdem gab ich ihm strukturiert weitere Aufgaben. Er sollte mir alle wichtigen Dokumente zusammenfassen, er sollte nach Fehlern im Code suchen, die das RL beeinflussen könnten. Dann soll er die Fehler auflisten und korrigieren. Ich habe auch einen Hinweis geschrieben, dass es sein kann, dass einige Dateien keine Fehler haben und dort alles korrekt ist. Damit wollte ich verhindern, dass er überall einen Fehler finden will und dadurch den korrekten Code korrigiert.

Der Output war dann gut und strukturiert, er hat einige Fehler erkannt und auch kurz erklärt. Anschliessend hat er diese Fehler angepasst und korrigiert. Die Änderungen waren meiner Meinung nach sinnvoll. Als ich das Programm jedoch erneut trainierte, war das Ergebnis wieder nicht sehr vielversprechend. Die Gewinnquote war immer noch gleich Null und als ich den RL beim Spielen beobachtete, wurde schnell klar, dass er die Regeln immer noch nicht verstanden hatte. In Situationen, in denen klar erkennbar ist, wo die Minen liegen und welche Felder sicher sind, werden die Minen trotzdem angeklickt. Dies habe ich auch dem Agenten mitgeteilt, der wiederum verschiedene Änderungen vorgeschlagen hat. Die Anpassung dieser Änderungen brachte jedoch auch nichts, im Gegenteil, die Lernergebnisse des RL wurden sogar noch etwas schlechter.

Als ich dem Agenten die Ergebnisse der Anpassung noch einmal vortrug, war die Antwort, dass DQN für Minesweeper in reiner Form wahrscheinlich nicht gut funktionieren würde. Er schlug mir dann andere Lösungen vor.

🎯 Die harte Wahrheit:

Warum RL hier scheitert:

1. **Sparse Rewards:** Agent stirbt in 95% der Fälle sofort (Mine getroffen)
2. **Kombinatorische Explosion:** $7 \times 7 = 49$ Aktionen, exponentiell viele Zustände
3. **Logische Inferenz nötig:** CNNs können Minesweeper-Logik nicht lernen
4. **Sample-Ineffizienz:** Braucht Millionen Episoden, nicht Tausende

Was die Forschung zeigt:

Erfolgreiche Minesweeper-AI nutzt:

- **Constraint Solver** (nicht RL!)
- **Probability-Based Approaches**
- **Rule-Based Systems**

Reines DQN ist der falsche Ansatz für dieses Problem!

Prompt 5: Agent erkennt das RL nicht funktioniert

Änderung des RL

Nach der Ausgabe des Agenten, dass mein Programm in der geplanten Form wahrscheinlich nicht funktionieren wird, habe ich mich entschlossen, seine Anpassungsvorschläge anzunehmen. Statt einer reinen RL wird nun eine Mischung aus RL und einer vorgefertigten Solverfunktion verwendet. Der erste Durchlauf hat gezeigt, dass die neue automatische Lösung tatsächlich zu Ergebnissen führt, was nach Tagen der Frustration ein schöner Aufwärtstrend war.

Die Anpassungen wurden anhand der Vorschläge des Sonnet-Agenten vorgenommen, ohne selbst in grösserem Umfang in die Anpassungen einzutreten. Nach einigen kleineren Fehlern und dem Hinzufügen weiterer Tests läuft das Programm nun gut und kann die meisten Minesweeper-Felder lösen.

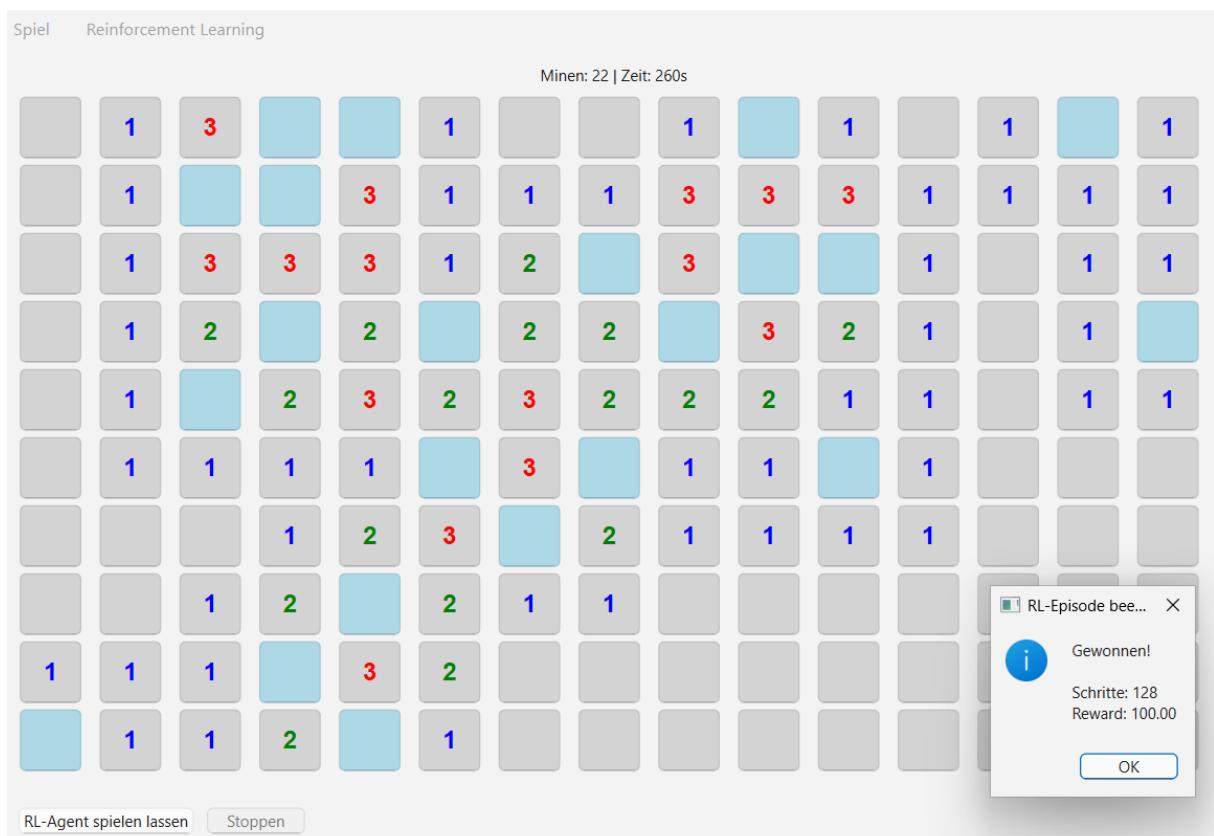


Abbildung 3: Automatisch gelöstes Feld

Optimierung und Verständnis des Bestehenden

Zu diesem Zeitpunkt des Projekts hatte ich meine beiden Hauptziele mehr oder weniger erreicht. Durch das Beheben aller Fehler, dem Neuaufbau des Algorithmus und den vielen verzweifelten Prompts zur Behebung eventueller Probleme war das Projekt etwas unübersichtlich geworden und hatte einige Teile, die ich noch zu wenig verstand. Deshalb wollte ich alles aufräumen, sortieren und verstehen.

Ich öffnete einen neuen Agenten und bat ihn, das Readme, alle Tests und alle Kommentare zu überprüfen und zu aktualisieren. Dann soll er mir alle wichtigen Dateien im Detail zeigen.

User

Ich habe ein Minesweeper programmiert mit zusätzlichen Automatischem Löser. Dabei wollte ich zuerst ein Reinforcement learning Programm implementieren, da dies aber nicht funktioniert hat, habe ich nun eine Mischung aus RL und einem Solver. Nun sind relativ viele unübersichtliche Dokumente während der Entstehung entstanden und ich habe etwas den Überblick verloren. Schau bitte alle Dokumente an, überprüfe ob sie noch benötigt werden.

Optimiere im Anschluss alle Kommentare in den Codes, Überprüffe, ob alle Tests noch sinnvoll sind.

Wenn das alles abgeschlossen ist. Erkläre mir alle Wichtigen Files, was in ihnen genau passiert und was die Logik ist.

Prompt 6: Zwischenstand optimieren

Anschliessend wollte ich noch einige Kleinigkeiten beheben, die mir bei der Problembehandlung aufgefallen sind, aber nicht dringend waren. Dazu zählt beispielsweise, dass der Start mehrere leere Felder aufdecken soll oder dass einige Anpassungen am Layout vorgenommen werden. Zudem soll auch der Algorithmus noch etwas optimiert werden.

Aktuell bin ich mit dem Zwischenstand des Projekts zufrieden. Die Hauptziele sind erfüllt und es gibt noch einige Dinge, die ich schrittweise hinzufügen kann.

Weitere Features Hinzufügen

Nachdem die Basis meines Spiels stand, wollte ich noch einige Features hinzufügen. Ich hatte bereits eine Idee: Ich möchte, dass ich mit einem Radar Teile des Spielfeldes aufdecken kann. Dies habe ich dem Agenten in einem einfachen Prompt mitgeteilt. Die Mechanik selbst konnte der Agent direkt und problemlos einfügen. Anschliessend musste ich jedoch noch einige Details korrigieren, damit es grafisch auch sinnvoll und ansehnlich war.

Danach wollte ich noch mehr solcher Joker. Ich hatte selbst aber gerade wenig Ideen. Deshalb habe ich ihn im Ask-Modus nach Ideen gefragt und erhielt etwa zehn Vorschläge. Diese waren teilweise gut und teilweise nur mässig nützlich. Ich liess ihn zwei weitere Joker und zudem ein Punktesystem implementieren, bei dem man für jedes aufgedeckte Feld einen Punkt erhält, den man dann für die Joker einsetzen kann.

Durch die Joker wurde das Spiel nun leichter, weshalb ich noch einige Hindernisse hinzufügen wollte. Auch hierfür habe ich einen Prompt verwendet, um ihn nach möglichen Hindernissen beim Aufdecken zu fragen. Er hat mir eine lange Liste mit Vorschlägen gegeben, die teilweise gut und teilweise schlecht waren. Ich habe jedoch zwei dieser Vorschläge implementiert: ein Mystery-Feld, bei dem es keine Zahl gibt, ausser man zahlt dafür, und ein Speed-Feld, bei dem man innerhalb der nächsten fünf Sekunden das nächste Feld anklicken muss, sonst verliert man.

User

Gut, jetzt will ich noch etwas Variation ins Spiel bringen also keine Hilfen wie bisher sondern noch Herausforderungen. Hast du mir eine Idee, wie eine solche Herausforderung aussehen könnte?

Cursor

Ausgezeichnete Idee! Herausforderungen als Gegenstück zu den Hilfen würden das Spiel viel dynamischer machen. Hier sind meine Vorschläge:

🔥 Herausforderungs-Ideen:

- ⌚ Zeitbombe**
 - Erscheint zufällig nach X aufgedeckten Feldern
 - Countdown von 30 Sekunden
 - Muss "entschärft" werden (bestimmtes Feld finden/aufdecken)
 - Wenn Zeit abläuft: -20 Punkte oder ein Teil des Feldes wird "nebelig"
- ≈ Nebel**
 - Deckt zufällig einen 5x5-Bereich mit "Nebel" zu
 - Bereits aufgedeckte Zahlen werden temporär verborgen
 - Kostet 40 Punkte zum Entfernen oder verschwindet nach 10 Zügen
 - Visuell: Halbtransparente graue Overlay

Prompt 7: Ideen für neue Herausforderungen

Danach hatte ich noch eine eigene, etwas kompliziertere Idee. Ich wollte ein Feld, das, sobald es aufgedeckt wird, eine Tetris-Form generiert. Beim nächsten Aufdecken würde dann nicht nur ein Feld, sondern die komplette Form aufgedeckt werden. Als ich mir das ausgedacht habe, sind mir noch einige Probleme aufgefallen. Die Form muss im verbleibenden Feld legbar sein und es muss möglich sein, die Form zu legen, ohne dass eine Mine hochgeht. Ich habe dem Agenten meinen Plan gegeben

und meine Bedenken zudem als Regeln formuliert. Auch diese Implementierung hat schnell und gut funktioniert.

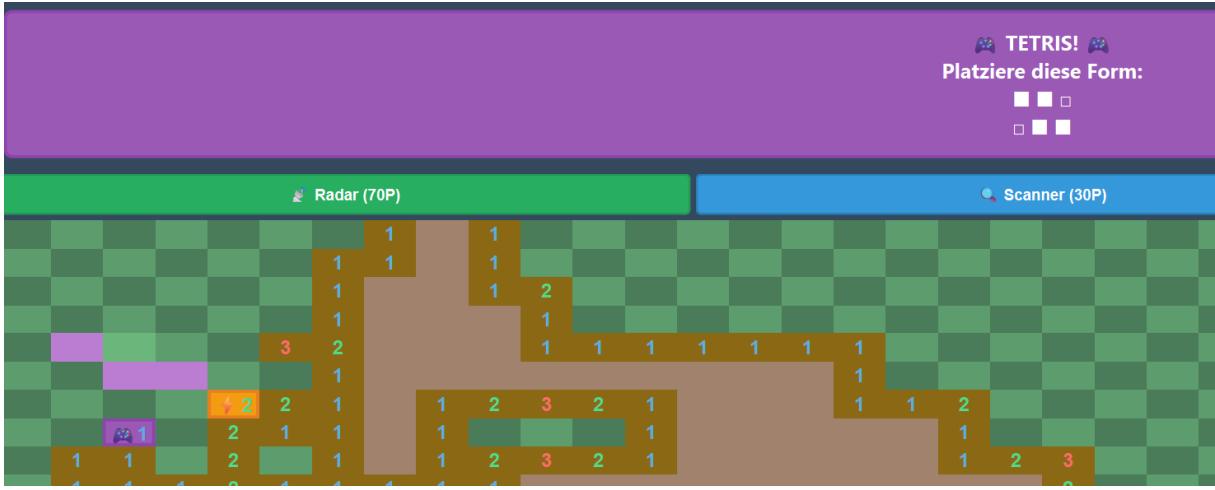


Abbildung 4: Erstes Spiel mit Tetris Block

Automatisches Spielen mit Features erweitern

Nachdem ich die neuen Features implementiert habe, ist das Spiel deutlich abwechslungsreicher geworden. Allerdings scheinen die älteren gespeicherten Modelle die neuen Features, wie zum Beispiel das Tetris-Feld, komplett zu ignorieren. Dies habe ich dem Agenten auch mitgeteilt. Seine erste Lösung für das Problem war, dass er alle Features zum Trainieren ausgeschaltet hat. Nachdem ich gesagt habe, dass er mit den Features trainieren können soll, hat er diese wieder aktiviert. Bei den Tetris-Blöcken hat er immer noch etwas Mühe, aber im Grossen und Ganzen funktioniert es nun wieder.

Finalisierung

Mit einem letzten neuen Agenten möchte ich nun noch das Readme anpassen und alle Kommentare im Code sowie die Anforderungen optimieren. Dazu habe ich ihm eine schrittweise Anleitung geschrieben, die er ausführen soll. Das Ziel besteht darin, ohne den Code zu verändern, alles so anzupassen, dass auch eine Person ohne Vorwissen das Programm ausführen und alle Inhalte nutzen kann.

Die Kommentare hat er sehr gut vereinheitlicht, auch die Tests hat er noch kurz überprüft. Das README war aber nur mässig gut. Für meinen Geschmack war es zu detailliert und umfänglich. Zudem erfand der Agent an einigen Stellen Dinge neu. Am meisten fiel dies bei seinem Punkt „Performance“ auf, den ich nachträglich gelöscht habe. Dort werden Zahlen genannt, die entweder gar nicht auf dieser Grösse getestet wurden oder schlichtweg falsch sind.

Erwartete Performance			
Mit Hybrid Agent:			
Board Size	Difficulty	Expected Win-Rate	Training Time
5x5	Easy	40-70%	500-1000 Episodes
7x7	Medium	20-50%	1000-2000 Episodes
9x9	Medium	15-35%	2000-3000 Episodes

Ohne Hybrid (Pure RL):			
Board Size	Difficulty	Win-Rate	Problem
5x5+	Any	~0%	Lernt nicht effektiv

**→ Hybr
id-Modus ist STANDARD und EMPFOHLEN!**

Abbildung 5: Generiertes README mit unlogischen Werten

Das Problem der erfundenen Performance-Werte zog sich jedoch durch das gesamte Projekt. Der Agent erfand an einigen Stellen einfach, wie gut ein Algorithmus performen könnte, ohne dies effektiv zu testen.

Verständnis und Ownership

Ich hatte verschiedene Projektteile, für die ich unterschiedliche Verständnisstufen hatte. Im Folgenden werde ich die einzelnen Teile durchgehen und reflektieren, was ich exakt verstanden habe und an welchen Stellen das Programm für mich eine Blackbox ist.

Grundaufbau des Minesweepers

Ich hatte zuvor keine Ahnung von der Entwicklung von Spielen oder sonstigen GUIs. Ich war deshalb froh, dass das LLM die Darstellung übernahm. Auch für die Spiellogik selbst habe ich mit dem LLM gearbeitet, hier verstehe ich den Code aber. Er besteht aus vielen Zustandsüberprüfungen, If-Statements und Klassen, die mir bereits bekannt sind. Ich kann deshalb Parameter selbst anpassen oder kleine Teile selbst ändern.

Grundstruktur Reinforcement Learning

Ich wusste bereits durch ein anderes Modul und einige zusätzliche Recherche was Reinforcement Learning ist und wie es in der Grundstruktur funktioniert. Ich verstehe das Konzept von Rewards und weiss auch wie ein solches Model trainiert. Jedoch habe ich noch keine Ahnung von der Implementation, die war für mich eine grössere Blackbox. Ich konnte nur anhand der Testdurchläufe im Training und den Ausgabedaten Informationen entnehmen, um sie dem Agenten zu geben, ohne direkt in den Code einzugreifen.

Erweiterung des Solvers

Nachdem klar war, dass mit reinem Reinforcement Learning kein guter Algorithmus entstehen würde, musste ein Solver für klare Zellen entwickelt werden. Hier habe ich den grössten Teil des Codes verstanden. Er macht im Grunde ähnliche Züge wie ein normaler Spieler.

Hinzufügen der Features

Ich habe natürlich verstanden, was die Features machen und wie sie implementiert sind. Allerdings habe ich nur ein mässiges Verständnis dafür, wie sie anschliessend im Solver angepasst wurden. Gerade weil ich einige Anpassungen vornehmen musste, bis das Programm wieder eine akzeptable Win-Rate erreichen konnte, ist mein Verständnis dafür nur mässig.

Tests und Dokumentation

Die Tests und die Dokumentation habe ich durch die KI generieren lassen. Ich habe mir die Tests jedoch auch selbst noch angeschaut und die entsprechenden Readmes durchgelesen. Bei beidem war es jedoch so, dass ich sie sehr früh im Prozess habe erstellen lassen und sie danach immer wieder geändert wurden, da sich auch die Anwendung geändert hat. Schlussendlich habe ich vor allem ein Bild vom Anfang und vom Ende der Tests und Dokus, über den Prozess dazwischen kann ich weniger sagen.

Methoden zur Herstellung von Verständnis

Die Strategien waren unterschiedlich: In einigen Bereichen hatte ich bereits Vorwissen, in anderen habe ich Informationen aus anderen Quellen genutzt. Bei der Erstellung des Spiels bzw. der Features habe ich vor allem den Code selbst gelesen und verstanden. Bei komplexeren Dingen wie dem RL habe ich vor allem die Zusammenfassungen des LLMs gelesen. Allgemein habe ich mir bei den Änderungen oft die Outputs des LLMs angeschaut. Manchmal ergaben sich daraus Folgefragen oder Aufgaben, manchmal merkte ich aber auch, dass nicht das umgesetzt wurde, was ich eigentlich wollte.

Reflexion über Ownership

Durch den Einsatz von KI habe ich viele Schritte bei der Erstellung an das LLM abgegeben. Zwar fühlt es sich an, als sei es ein Produkt von mir, da meine Ideen massgeblich eingeflossen sind und es nach meinen Wünschen gestaltet wurde. Ich würde mich jedoch nicht als den Entwickler des Produkts verstehen. Für mich ist es eher so, als hätte ich eine Idee in Auftrag gegeben, die ausgeführt und durch meine Verbesserungsvorschläge optimiert wurde.

Ich kann die Aktionen, das Spiel, die Stärken und Schwächen des Programms beschreiben. Ich hätte jedoch Mühe, den Code Zeile für Zeile zu erklären oder in gewissen Bereichen manuell einzugreifen. Teilweise macht es wahrscheinlich auch Sinn, die sich wiederholende Arbeit, die mit Flüchtigkeitsfehlern verbunden war, an die KI abzugeben. Gerade wenn es darum geht, die Regeln zu definieren oder den Solver zu erstellen, der sichere Zellen erkennt.

An anderen Stellen war die Arbeit mit der KI wohl etwas zu schnell. Gerade beim RL hätte mehr Wissen und ein besseres Verständnis des Codes dazu geführt, dass ich nicht so lange an einer Stelle hängen geblieben wäre. Es ging dort weniger um die rein programmiertechnische Umsetzung als um das Verständnis des RL-Konzepts, wobei die KI immer nur versuchte, den Algorithmus zu implementieren.

Qualitätssicherung

In meinem Fall war die Qualitätssicherung einfach. Ich konnte das Spiel starten und spielen. Sehr schnell haben sich dann Fehler oder kleine optische Details herausgestellt, die ich angepasst haben wollte. In letzter Zeit habe ich vermutlich so viel Minesweeper gespielt wie in meinem ganzen Leben zuvor nicht.

Beim Solver habe ich es ähnlich gemacht. Während des Trainings habe ich alle 100 Episoden einen Output generieren lassen und den Agenten in einem Testlauf einmal spielen lassen. Nachdem das Training beendet war, konnte ich den Agenten spielen lassen und habe schnell Fehler entdeckt, die er noch machte. Beispielsweise verlor er nach Erstellung der Tetris-Blöcke sehr oft durch genau diese Felder, weshalb ich den Solver anpassen musste.

Zu diesen visuellen Tests habe ich zu Beginn noch einige Tests im Programm erstellt, die der Agent am Ende einer Ausgabe oft selbst ausgeführt hat. Zudem wurden laufend neue Tests erstellt, wenn neue Funktionen hinzugefügt wurden.

Fazit

Reflexion

KI kann helfen, ist aber nicht die Lösung für alles. Ich habe gelernt, dass einfache Arbeiten oder Korrekturen und Anpassungen von Bestehendem sehr gut durch KI erledigt werden können. Man kommt enorm schnell zu Ergebnissen und kann plötzlich Probleme lösen, für deren Verständnis man sonst vielleicht Jahre gebraucht hätte. Dazu müssen die Prompts aber auch gut und strukturiert geschrieben werden.

Ich habe bei mir selbst gemerkt, dass ich schnell die Übersicht verloren habe, wenn der grösste Teil durch KI geschrieben wurde. Bei Fehlern und Problemen habe ich sofort nur noch die Ausgaben an das LLM weitergeleitet und gesagt, dass es nicht funktioniert. Das war teilweise sehr frustrierend, da ich die Fehler nicht mehr selbst korrigieren konnte, sondern auf die Lösung der KI angewiesen war.

Durch das Projekt hat sich meine Sicht auf KI-gestützte Entwicklung verändert. Ich habe gesehen, wie gut ein Programm wie Cursor ist, das direkt Befehle ausführen kann, um Teile eines Programms zu schreiben, zu testen und zu verbessern. Nach der anfänglichen Euphorie und dem Gefühl, plötzlich alles durch einen Prompt erstellen zu können, wurde mir aber auch schnell klar, dass komplexere Probleme nicht ohne eigenes Wissen mithilfe von KI implementiert werden können. Man muss verstehen, wie gewisse Algorithmen funktionieren. Man muss sie nicht selbstständig komplett programmieren können, aber man muss sie verstehen.

Ausserdem ist mir aufgefallen, dass die LLMs einem meistens zustimmen. Bis auf den einen Punkt beim RL hatte das LLM immer eine „Lösung“. Das war teilweise auch frustrierend, wenn einem gesagt wird, dass es nun zu 100 % funktionieren wird, aber immer noch der gleiche Fehler wie vorher besteht.

Auch die Kreativität ist nur mässig vorhanden. Bei der Frage nach Features waren einige gute, jedoch sehr klassische Vorschläge dabei. Wirklich spannende Funktionen wie zum Beispiel der Tetris Block musste man selbst einbringen.

Empfehlungen für andere Studis

Ich würde anderen Studierenden empfehlen, an bestimmten Stellen in der Entwicklung spezifischer nachzufragen. Gerade in Bereichen, in denen man selbst nur über mässiges Wissen verfügt, kann man viel Zeit verlieren, wenn sich die KI im Kreis dreht oder immer versucht, deine Anforderungen zu erfüllen, ohne dir zu sagen, dass dies wahrscheinlich gar nicht gut funktioniert.

Zudem würde ich bei der Projekterstellung darauf achten, etwas auszuwählen, das nicht sonderlich lange dauert. Ich habe das Problem, dass das RL zum Trainieren enorm lange dauert, gerade auf grösseren Spielfeldern. Wenn man viel verbessern muss und dann jedes Mal neu trainieren muss, benötigt dies enorm viel Zeit.