

Übungsblatt 03: Spiele

Bonus: Verlässlichkeit von ChatGPT

- Auch wenn in diesem konkreten Beispiel nicht viele Mängel aufgewiesen werden konnten, habe ich noch diesen aktuellen Beitrag der Tagesschau gefunden. Dieser verweist auch auf eine Studie, die gefunden hat, dass die meisten Chatbots halluzinieren →
<https://www.tagesschau.de/wissen/technologie/kuenstliche-intelligenz-fakten-100.html>

Beispiel I: Erstellung einer Zeitleiste eines historischen Ereignisses → Die erste Mondlandung

- Recherche
 - Quelle: <https://airandspace.si.edu/explore/stories/apollo-missions/apollo-11-moon-landing/apollo-11-timeline>
 - 16 Juli 1969 um 13:32 Uhr (UTC) → Die Apollo 11 Rakete hebt aus Cape Kennedy ab
 - 20 Juli um 17:44 Uhr (UTC) → Das Mondmodul mit Armstrong und Aldrin an Bord separiert sich vom Kommandomodul und beginnt Landeanflug auf den Mond
 - 20:17 Uhr (UTC) → Touch Down auf der Mondoberfläche
 - 21 Juli um 2:51 Uhr (UTC) → Neil Armstrong macht seine ersten Schritte auf dem Mond
 - 17:54 Uhr (UTC) → Das Mondmodul tritt den Rückflug an mit dem Ziel sich wieder mit dem Kommandomodul zu vereinigen
 - 4:55 Uhr (UTC) → Apollo 11 verlässt den Mondorbit und beginnt den Rückflug zur Erde
 - 24 Juli um 4:51 Uhr (UTC) → Die Astronauten landen sicher auf der Erde
- LLM Chatbot Chatgpt

Bitte gib mir einen Zeitstrahl der Apollo 11 Mission vom 16 - 24 Juli 1969

Hier ist ein **Zeitstrahl der Apollo-11-Mission** vom 16. bis 24. Juli 1969, die die erste bemannte Mondlandung der Menschheitsgeschichte war.

Zeitstrahl Apollo 11 (16.–24. Juli 1969)

16. Juli 1969 – Start

- 09:32 Uhr EDT (15:32 Uhr MESZ): Start von Apollo 11 vom Kennedy Space Center (Florida) mit einer Saturn-V-Rakete.
- **Besatzung:**
 - Neil A. Armstrong – Kommandant
 - Edwin E. „Buzz“ Aldrin – Lunar Module Pilot
 - Michael Collins – Command Module Pilot
- **Ziel:** Landung auf dem Mond im Mare Tranquillitatis („Meer der Ruhe“).
- Nach etwa 12 Minuten erreicht Apollo 11 die Erdumlaufbahn.
- Nach einer Erdumrundung wird das **Translunar Injection**-Manöver ausgeführt – Kurs zum Mond.

17. Juli 1969 – Flug zum Mond

- Das Raumschiff verlässt den Einfluss der Erde und bewegt sich auf die Mondbahn zu.
- Routineübertragungen zur Erde; das Kommandomodul „**Columbia**“ wird mit dem Mondlandemodul „**Eagle**“ verbunden.
- Überprüfung der Systeme und Kurskorrekturen.

18. Juli 1969 – Halber Weg zum Mond

- Die Crew führt Experimente und TV-Übertragungen durch.
 - Michael Collins überprüft das Kommandomodul, Armstrong und Aldrin inspizieren das Landemodul.
 - Etwa 320.000 km von der Erde entfernt.
-

19. Juli 1969 – Eintritt in die Mondumlaufbahn

- **17:21 Uhr EDT (23:21 MESZ)**: Apollo 11 tritt in die Mondumlaufbahn ein.
 - Die Besatzung macht Fotos möglicher Landegebiete.
 - Armstrong und Aldrin bereiten „Eagle“ auf die Abkopplung vor.
-

20. Juli 1969 – Mondlandung

- **13:46 Uhr EDT (19:46 MESZ)**: „Eagle“ trennt sich von „Columbia“.
 - **16:17 Uhr EDT (22:17 MESZ)**: **Landung auf dem Mond**.
 - Funkspruch: „Houston, Tranquility Base here. The Eagle has landed.“
 - **21:56 Uhr EDT (03:56 MESZ am 21. Juli)**: Neil Armstrong betritt den Mond.
 - „That's one small step for [a] man, one giant leap for mankind.“
 - Buzz Aldrin folgt etwa 20 Minuten später.
 - Die beiden sammeln Proben, stellen eine US-Flagge auf und hinterlassen wissenschaftliche Geräte (z. B. Laserreflektor).
-

21. Juli 1969 – Rückkehr zum Mondorbit

- Nach rund **21½ Stunden** auf der Mondoberfläche startet „Eagle“ wieder.
- **Rendezvous und Andocken** mit „Columbia“.
- Übertragung der Mondproben in das Kommandomodul.
- „Eagle“ wird abgetrennt und bleibt in der Mond**↓** umlaufbahn.

22. Juli 1969 – Rückflug zur Erde

- Apollo 11 verlässt die Mondumlaufbahn und tritt in die Rückflugbahn zur Erde ein.
- Die Crew ruht und führt Routinewartung durch.

23. Juli 1969 – Letzter Flugtag

- Letzte TV-Übertragung aus dem Weltraum.
- Armstrong: „Wir fühlen uns geehrt, die Menschheit auf dieser Reise vertreten zu haben.“

24. Juli 1969 – Rückkehr zur Erde

- 12:44 Uhr EDT (18:44 MESZ): Wiedereintritt in die Erdatmosphäre.
- 12:50 Uhr EDT: Wasserlandung im Pazifik, ca. 1500 km südwestlich von Hawaii.
- 13:08 Uhr EDT: Rettung durch das Flugzeugträger-Schiff USS Hornet.
- Astronauten begeben sich für 21 Tage in Quarantäne.

❖ Missionsergebnis

- Erste erfolgreiche bemannte Mondlandung.
- Dauer der Mission: **8 Tage, 3 Stunden, 18 Minuten, 35 Sekunden**.
- Mitgebrachte Mondproben: **21,55 kg**.

Möchtest du, dass ich diesen Zeitstrahl als **grafische Zeitleiste** (z. B. als Infografik oder PDF) darstelle?



Beispiel II: Erstellen von Quellcode zu einem Standardproblem aus der Informatik → Merge Sort in Java

- Recherche
 - Quelle:
https://www.w3schools.com/dsa/dsa_algo_mergesort.php#:~:text=The%20Merge%20Sort%20algorithm%20is,so%20that%20it%20is%20sorted.

```
def mergeSort(arr):
    if len(arr) <= 1:
        return arr
```

```

mid = len(arr) // 2
leftHalf = arr[:mid]
rightHalf = arr[mid:]

sortedLeft = mergeSort(leftHalf)
sortedRight = mergeSort(rightHalf)

return merge(sortedLeft, sortedRight)

def merge(left, right):
    result = []
    i = j = 0

    while i < len(left) and j < len(right):
        if left[i] < right[j]:
            result.append(left[i])
            i += 1
        else:
            result.append(right[j])
            j += 1

    result.extend(left[i:])
    result.extend(right[j:])

    return result

unsortedArr = [3, 7, 6, -10, 15, 23.5, 55, -13]
sortedArr = mergeSort(unsortedArr)
print("Sorted array:", sortedArr)

```

- LLM Chatbot Chatgpt

Bitte gib mir eine rekursive Implementation von Merge Sort in Java

Natürlich! 😊

Hier ist eine rekursive Implementation von Merge Sort in Java, die ein Array von Ganzzahlen sortiert:

```
public class MergeSort {

    // Hauptfunktion zum Testen
    public static void main(String[] args) {
        int[] arr = {38, 27, 43, 3, 9, 82, 10};

        System.out.println("Unsortiert:");
        printArray(arr);

        mergeSort(arr, 0, arr.length - 1);

        System.out.println("Sortiert:");
        printArray(arr);
    }

    // Rekursive MergeSort-Funktion
    public static void mergeSort(int[] arr, int left, int right) {
        if (left < right) {
            // Mitte finden
            int mid = (left + right) / 2;

            // Linke und rechte Hälfte sortieren
            mergeSort(arr, left, mid);
            mergeSort(arr, mid + 1, right);

            // Teilergebnisse zusammenführen
            merge(arr, left, mid, right);
        }
    }
}
```

```

// Hilfsfunktion zum Zusammenführen zweier sortierter Teilarrays
public static void merge(int[] arr, int left, int mid, int right) {
    // Längen der beiden Teilarrays berechnen
    int n1 = mid - left + 1;
    int n2 = right - mid;

    // Temporäre Arrays erstellen
    int[] L = new int[n1];
    int[] R = new int[n2];

    // Daten kopieren
    for (int i = 0; i < n1; i++)
        L[i] = arr[left + i];
    for (int j = 0; j < n2; j++)
        R[j] = arr[mid + 1 + j];

    // Indizes initialisieren
    int i = 0, j = 0, k = left;

    // Teilarrays zusammenführen
    while (i < n1 && j < n2) {
        if (L[i] <= R[j]) {
            arr[k] = L[i];
            i++;
        } else {
            arr[k] = R[j];
            j++;
        }
        k++;
    }

    // Übrig gebliebene Elemente kopieren
    while (i < n1) {
        arr[k] = L[i];
        i++;
    }
}

```

```

        k++;
    }
    while (j < n2) {
        arr[k] = R[j];
        j++;
        k++;
    }
}

// Hilfsfunktion zum Ausgeben eines Arrays
public static void printArray(int[] arr) {
    for (int value : arr) {
        System.out.print(value + " ");
    }
    System.out.println();
}

```

Erklärung:

- `mergeSort()` teilt das Array rekursiv in zwei Hälften, bis nur noch Einzelemente übrig sind.
- `merge()` fügt anschließend die sortierten Hälften wieder zusammen.
- Der Algorithmus hat eine Laufzeitkomplexität von $O(n \log n)$ und eine Speicherkomplexität von $O(n)$ wegen der temporären Arrays.

Möchtest du, dass ich dir die Version so anpasse, dass sie generisch (für beliebige Typen mit `Comparable`) funktioniert?



Vergleich

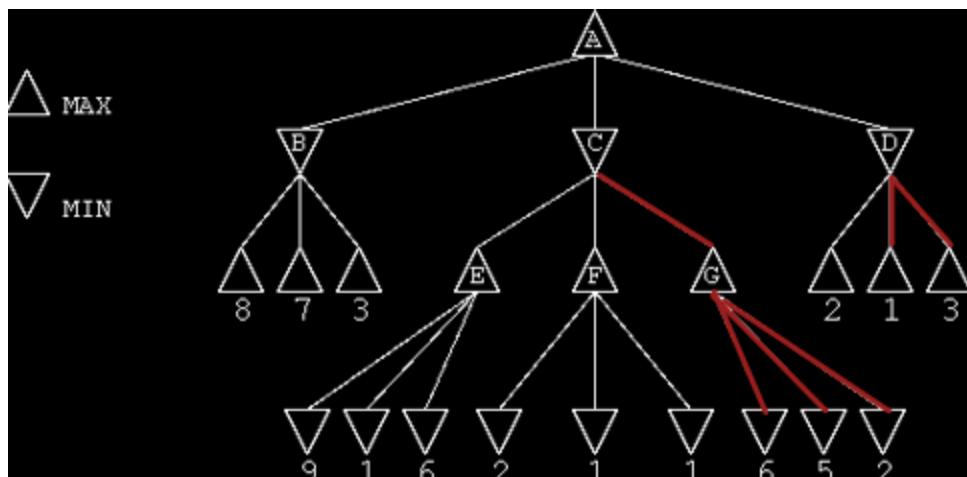
- Die Informationen in diesen Beispielen sind weitgehend korrekt, auch wenn manche Details zu z.B. der Mondmission schwierig zu validieren sind, dies ist allerdings hauptsächlich darauf zurückzuführen, dass es um simple Ereignisse und Quellcode ging die so einfach aus Onlinequellen übernommen werden

können. Würde man hier nach Quellcode für ein komplexeres und einzigartiges Problem der Informatik fragen beispielsweise, dann würde das LLM mehr Probleme haben ein Brauchbares resultat zu liefern, da es nie direkt auf die nötige Kombination an Quellcode trainiert wurde und nicht den Code selber versteht.

- Just for Fun: Aktuelles Video zu aktuellen Problemen/Risiken von AI und LLMs
→ <https://www.youtube.com/watch?v=90C3XVjUMqE&t=913s>

A01: Handsimulation: Minimax und alpha-beta-Pruning

1. A → 3, B → 3, C → 2, D → 1, E → 9, F → 2, G → 6
2. Pruned Kanten sind rot markiert



3. Man müsste E und F tauschen, dann würde man zuerst F traversieren und könnte den kompletten Sub-Tree von E prunen

A02: Optimale Spiele: Minimax und alpha-beta-Pruning

1. Minimax-Algo am Beispiel TicTacToe in Python

```

import math
# Kontrolliert wer gewonnen hat (kann auch unentschieden sein)
def check_winner(board):
    wins = [(0,1,2), (3,4,5), (6,7,8),
             (0,3,6), (1,4,7), (2,5,8),
             (0,4,8), (2,4,6)]
    for a,b,c in wins:
        if board[a] != " " and board[a] == board[b] == board[c]:
            return board[a]
    if " " not in board:
        return "draw"
    return None

# Minimax-Algo
node_counter = 0

def minimax(board, is_maximizing):
    global node_counter
    node_counter += 1

    winner = check_winner(board)
    if winner == "X": return 1
    if winner == "O": return -1
    if winner == "draw": return 0

    if is_maximizing:
        best = -math.inf
        for i in range(9):
            if board[i] == " ":
                board[i] = "X"
                value = minimax(board, False)
                board[i] = " "
                best = max(best, value)
        return best
    else:

```

```

best = math.inf
for i in range(9):
    if board[i] == " ":
        board[i] = "O"
        value = minimax(board, True)
        board[i] = " "
        best = min(best, value)
return best

# Ausgabe
board = [" "] * 9 # Spielfeld leer generieren
result = minimax(board, True)
print("Startzustandsbewertung:", result)
print("Besuchte Knoten:", node_counter)

```

2. Mit Alpha-Beta-Pruning

```

import math

# Kontrolliert wer gewonnen hat (kann auch unentschieden sein)
def check_winner(board):
    wins = [(0,1,2), (3,4,5), (6,7,8),
            (0,3,6), (1,4,7), (2,5,8),
            (0,4,8), (2,4,6)]
    for a,b,c in wins:
        if board[a] != " " and board[a] == board[b] == board[c]:
            return board[a]
    if " " not in board:
        return "draw"
    return None

node_counter_pruning = 0

def minimax_pruning(board, is_maximizing, alpha=-math.inf, beta=math.in

```

```

f):
    global node_counter_pruning
    node_counter_pruning += 1

    winner = check_winner(board)
    if winner == "X": return 1
    if winner == "O": return -1
    if winner == "draw": return 0

    if is_maximizing:
        best = -math.inf
        for i in range(9):
            if board[i] == " ":
                board[i] = "X"
                value = minimax_pruning(board, False, alpha, beta)
                board[i] = " "
                best = max(best, value)
                alpha = max(alpha, best)
                if beta <= alpha:
                    break # Prune
        return best
    else:
        best = math.inf
        for i in range(9):
            if board[i] == " ":
                board[i] = "O"
                value = minimax_pruning(board, True, alpha, beta)
                board[i] = " "
                best = min(best, value)
                beta = min(beta, best)
                if beta <= alpha:
                    break
        return best

# Ausgabe
board = [" "] * 9 # Spielfeld leer generieren

```

```

result = minimax_pruning(board, True)
print("Startzustandsbewertung:", result)
print("Besuchte Knoten:", node_counter_pruning)

```

3. Wir benutzen das folgende Board = ["X", "O", "X", "O", " ", " ", " ", " ", " "]

```

# Vergleich
board = ["X", "O", "X", "O", " ", " ", " ", " ", " "]

node_counter = 0
score_plain = minimax(board, True)

node_counter_pruning = 0
score_pruning = minimax_pruning(board, True)

print("Minimax-Ergebnis:", score_plain, "Knoten:", node_counter)
print("Alpha-Beta-Ergebnis:", score_pruning, "Knoten:", node_counter_pruning)

```

- Mit einem Ergebnis von 234 zu 65 sieht man hier den großen Vorteil des Alpha-Beta-Prunings deutlich

A05: Minimax generalisiert

