Dame Spiel

Auf einem Spielfeld sind 8x8 Kacheln positioniert. Diese 64 Kacheln sind im Farbwechsel, in Weiß und in Grün, abgebildet. Auf der oberen Hälfte des Spielfeldes sind 12 schwarze Steine auf jeweils einer grünen Kachel platziert. Äquivalent auf der unteren Hälfte des Spielfeldes sind weiße Steine auf jeweils einer grünen Kachel positioniert. Du spielst gegen den Computer und benutzt die weißen Steine. Weiterhin beginnst du mit dem Spiel. Weiß beginnt.

Ein weißer Stein, dessen benachbarte diagonale grüne Kachel(n) frei ist/sind, wird mit der Maus angeklickt. Sofort wechselt die grüne(n) Kachel(n) die Farbe in ein helleres Grün. Das ist eine Hilfe für dich. Klick nun eine der mit hellem Grün markierten Kachel an, und der Stein bewegt sich zu dieser neuen Zielposition. Der Computer reagiert mit den schwarzen Steinen entsprechend.

Ziel des Damespiels ist es, die gegnerischen Steine zu entern. Erreicht ein Stein die Grundlinie des Gegners, wird dieser Stein zur Dame gekrönt. Die nachfolgende Textpassage „Schlagzwang“ erklärt die Eliminierung eines Steins.

Schlagzwang

Ist ein gegnerischer Stein diagonal benachbart und das Feld dahinter frei, muss dieser übersprungen und geschlagen werden. Bestehen verschiedene Möglichkeiten, zu schlagen, darf frei gewählt werden, welche Variante genutzt wird. Wenn das Zielfeld eines Sprungs auf ein Feld führt, von dem aus einem weiteren Stein übersprungen werden kann, wird der Sprung *nicht* fortgesetzt. Eine Dame (durch das Erreichen der Grundlinie des Gegners) kann über mehrere freie Felder hinweg schlagen.

Die technischen Details

Das Downloaden der drei Dateien HTML, CSS und JavaScript ist jetzt vom Vorteil. Das Webentwicklertool von Google Chrome bietet hier alles was du brauchst. Wenn das Damespiel im Browser läuft, betätige die Taste F12 und schon hast du Zugriff.

Wir konzentrieren uns auf das, was das Spiel im Kern zusammenhält: Die Mechanik, die Regeln und wie der Computer überhaupt weiß, dass du gerade auf einen Stein geklickt hast. Stell dir den Code wie einen digitalen Schiedsrichter vor, der im Hintergrund mit einem Notizblock sitzt. Hier ist die Erklärung der wichtigsten Bausteine in "Menschensprache":

Das Spielfeld (Board).

Alles beginnt mit einer Liste von Listen. Für den Computer ist das Spielfeld kein Bild, sondern eine Tabelle mit Zahlen:

- **0 bedeutet: Das Feld ist leer.**
- **1 oder -1: Ein normaler Stein (Weiß oder Schwarz).**
- **2 oder -2: Eine Dame.**

Das Auge, die Funktion "renderBoard()"

Diese Funktion ist der "Maler". Jedes Mal, wenn sich etwas ändert (ein Stein zieht um), löscht sie das alte Bild im Browser und zeichnet es anhand der Zahlen-Tabelle neu.

- **Sie schaut: "Ist hier eine 1? Dann mal einen weißen Kreis."**
- **Sie prüft auch: "Hat der Spieler gerade diesen Stein angeklickt? Wenn ja, mal einen gelben Rahmen drumherum."**

Die Klick-Logik (handleSquareClick()).

Das ist das Gehirn der Steuerung. Es arbeitet in zwei Schritten (der "Zwei-Klick-Logik").

- **Erster Klick: Du klickst einen Stein an. Der Computer prüft: "Gehört der Stein dir?" Wenn ja, merkt er sich die Position (selectedSquare).**
- **Zweiter Klick: Du klickst auf ein Zielfeld. Jetzt ruft der Computer laut nach dem Regelbuch (getMoveType()), um zu fragen: "Darf er das überhaupt?"**

Das Regelbuch (getMoveType())

Das ist der komplizierteste Teil. Die Funktion berechnet den Unterschied zwischen Start und Ziel.

- **Normaler Zug: Ist das Ziel genau ein Feld diagonal entfernt und leer?**
- **Schlag-Zug: Sind es zwei Felder Distanz? Und steht auf dem Feld dazwischen ein Gegner? Wenn ja, gibt die Funktion das Signal: "Zug erlaubt, und lösche den Gegner!"**

- **Dame-Zug: Hier scannt der Code die gesamte Diagonale. Er schaut Feld für Feld: "Ist der Weg frei? Steht da nur maximal ein Gegner?"**

Der Schiedsrichter (canPlayerCapture())

Bevor du ziehst, lässt der Schiedsrichter einmal kurz den Blick über das ganze Feld schweifen. Er sucht nach Schlagmöglichkeiten für den aktuellen Spieler. Findet er eine, tritt der Schlagzwang in Kraft. Er sagt dem Klick-System: "Blockiere alle normalen Züge! Der Spieler darf nur klicken, wo er springen kann."

Die Ausführung (executeMove())

Wenn alles genehmigt ist, werden die Zahlen in der Tabelle getauscht:

- **Der Startplatz wird auf 0 gesetzt.**
- **Das Ziel bekommt die Zahl des Steins.**
- **Falls ein Gegner übersprungen wurde, wird dessen Platz ebenfalls auf 0 gesetzt.**
- **Die Krönung: Erreicht ein Stein die gegenüberliegende Seite, wird aus der 1 eine 2 (die Dame).**

Der Rundenwechsel

Am Ende jeder Aktion dreht der Computer das Vorzeichen um (currentPlayer *= -1). Aus 1 (Weiß) wird -1 (Schwarz). Jetzt ist der andere dran. Ganz zum Schluss prüft er noch: "Hat ein Spieler keine Steine oder keine Züge mehr?" Dann wirft er das "Game Over"-Fenster auf den Schirm.

Kurz gesagt: Der JavaScript-Code ist ein ständiger Kreislauf aus Anschauen (Tabelle lesen), Abgleichen (Regeln prüfen) und Umrechnen (Tabelle ändern).

Die Sache mit der Dame ist sehr, sehr komplex!

Stell dir vor, der Computer macht nach jedem einzelnen Steinwurf eine kleine Abschlusskontrolle, bevor er das Brett wieder für den nächsten Spieler freigibt.

Die "Ziellinien"-Kontrolle

Der Computer weiß jederzeit, in welcher Reihe (r) ein Stein gelandet ist. Er kennt die Grenzen seines Spielfeldes:

- **Reihe 0 ist die oberste Kante (das Ziel für Weiß).**
- **Reihe 7 ist die unterste Kante (das Ziel für Schwarz).**

Sobald eine Bewegung ausgeführt wurde, stellt der Computer zwei kurze Fragen:

- **"Bist du ein weißer Stein (1) und stehst jetzt in Reihe 0?"**
- **"Bist du ein schwarzer Stein (-1) und stehst jetzt in Reihe 7?"**

Das nachfolgende Bild zeigt die Zusammenhänge:

```
217  function executeMove(from, toR, toC, move) {
218    board[toR][toC] = board[from.r][from.c];
219    board[from.r][from.c] = 0;
220    if (move.type === "capture") board[move.captured.r][move.captured.c] = 0;
221
222    // Dame-Umwandlung
223    if (
224      (toR === 0 && board[toR][toC] === 1) ||
225      (toR === 7 && board[toR][toC] === -1)
226    ) {
227      board[toR][toC] = Math.sign(board[toR][toC]) * 2;
228    }
229
```

Manchmal nutzen Programmierer auch einen mathematischen Trick: board[toR][toC] *= 2. Aus der 1 wird eine 2, aus der -1 eine -2. Die Zahl verdoppelt sich, und damit weiß das System: "Achtung, das ist jetzt ein Super-Stein!"

Was ändert sich für den Stein?

Sobald aus der 1 eine 2 geworden ist, ändern sich zwei Dinge schlagartig:

- **Die Optik: Die Funktion renderBoard() sieht die 2 und sagt: "Ah, eine zwei! Ich klebe diesem Stein jetzt ein Krönchen-Symbol (Dame 👑) auf oder gebe ihm eine goldene CSS-Klasse."**
- **Die Superkräfte: Wenn du das nächste Mal mit diesem Stein ziehen willst, schaut die Logik getMoveType ins Regelbuch. Sie sieht: "Oh, das ist kein normaler Stein mehr, das ist eine 2. Er darf jetzt nicht nur ein Feld weit springen, sondern wie ein Laserstrahl über das ganze Brett fliegen!"**

Warum ist das so wichtig?

Ohne diese kleine "Checkliste" am Ende der executeMove-Funktion würde dein Stein einfach am Rand stehen bleiben und könnte nichts mehr tun. Er wäre wie ein Tourist, der am Ende der Welt angekommen ist und nicht weiß, wie er umdrehen soll. Erst die Zahl 2 gibt ihm die Erlaubnis, die Richtung zu ändern und wieder zurück ins Feld zu stürmen.

Zugeigenschaften der Dame! Für uns ist es ein langer diagonaler Weg, für den Computer ist es eine Schritt-für-Schritt-Analyse. Stell dir vor, die Dame ist wie ein Laserpointer. Wenn du ein Ziel anklickst, schaltet der Computer den Laser an und prüft jeden einzelnen Zentimeter des Strahls, bevor der Stein losfliegt.

Hier ist der Ablauf in drei einfachen Schritten:

Die Richtung finden

Zuerst berechnet der Computer die Richtung. Er schaut: "Geht es nach oben-rechts, oben-links, unten-rechts oder unten-links?" Im Code ist das eine einfache Subtraktion der Koordinaten. Er weiß jetzt: "Ich muss pro Schritt immer +1 zur Reihe und -1 zur Spalte rechnen."

Der diagonale Scan (Die "Schleife")

Der Computer springt jetzt nicht sofort zum Ziel. Er geht das Brett Feld für Feld auf der Diagonale ab. Das ist wie eine Patrouille:

- **Feld 1: "Ist hier jemand?" -> Nein, leer. "Weitergehen."**
- **Feld 2: "Ist hier jemand?" -> Ja, ein Stein.**

Die entscheidende Prüfung: "Gehört der Stein mir?"

- **Wenn JA: Der Laser prallt ab. Der Zug wird sofort als ungültig markiert. Der Stein darf nicht mal auf diesem Feld landen, geschweige denn darüber fliegen.**
- **Wenn NEIN (Gegner): Der Computer macht sich eine Notiz: "Gegner gefunden an Position XY."**

Die "Ein-Gegner-Regel"

Hier ist der Computer sehr streng. Damit die Dame nicht zu mächtig wird, darf sie in einem Zug nur genau einen Gegner schlagen.

- **Findet der Scan auf dem Weg zum Ziel einen zweiten Gegner? -> Stopp! Der Zug ist ungültig.**
- **Ist das Feld direkt hinter dem ersten Gegner belegt? -> Stopp! Der Zug ist ungültig.**

Zusammenfassend: Der Computer nutzt eine sogenannte while-Schleife (eine "Solange-bis"-Wiederholung). Er sagt: "Solange ich das Ziel noch nicht erreicht habe, prüfe das nächste Feld. Wenn du einen eigenen Stein siehst, brich sofort ab."

```
163     while (currR !== toR) {
164       const p = currentBoard[currR][currC];
165       if (p !== 0) {
166         if (Math.sign(p) === player || enemy) return null;
```

Zeile 166: "Hier steht einer von uns! Zugriff verweigert." Es ist also eine ganz einfache "Wenn-Dann"-Logik, die verhindert, dass deine Dame deine eigenen Leute einfach "überfährt".

Wer hat gewonnen? Wie erkennt der Computer das?

Das ist eine der "fiesesten" Arten zu verlieren: Man hat eigentlich noch Steine, aber man steht sich selbst oder dem Gegner so im Weg, dass nichts mehr geht. Der Computer geht dabei vor wie ein Detektiv, der nacheinander alle Türen prüft. Hier ist der Ablauf:

Die Inventur (canMove())

Nach jedem Zug schaut der Computer den Spieler an, der jetzt eigentlich dran wäre. Er macht eine Bestandsaufnahme:

- **Er sucht auf dem Brett nach allen Steinen dieses Spielers.**
- **Für jeden Stein, den er findet, stellt er die Frage: „Gibt es irgendein Feld auf diesem Brett, auf das du legal ziehen oder springen darfst?“**

Der „Was-wäre-wenn“-Test

Der Computer probiert im Kopf (im Code) alle Richtungen aus. Er fragt das Regelbuch (getMoveType()):

- **„Darf dieser Stein nach vorne links?“ -> Antwort: Nein, besetzt.**
- **„Darf dieser Stein nach vorne rechts?“ -> Antwort: Nein, Rand.**
- **„Darf dieser Stein springen?“ -> Antwort: Nein, kein Platz.**

Die Sackgasse

Wenn der Computer alle Steine des Spielers durchgegangen ist und bei jedem einzelnen ein „Nein“ vom Regelbuch bekommen hat, zieht er die Reißleine. Für den Computer ist das ein logischer Schalter:

- **Mögliche_Züge > 0? -> Weiterspielen.**
- **Mögliche_Züge === 0? -> Spiel beenden.**

Wir Menschen übersehen oft eine kleine Lücke oder eine Schlagmöglichkeit am anderen Ende des Bretts. Der Computer hingegen ist gnadenlos: Er scannt

alle 64 Felder in Millisekunden. Er „sieht“ die Blockade schon, bevor du überhaupt merkst, dass du dich festgefahren hast.

```
30  //*****************************************************************************
31  // Hilfsfunktion Gewinner visualisieren. Steinebewegung erschöpft?
32  //*****************************************************************************
33  function canMove(p) {
34    for (let r = 0; r < 8; r++)
35      for (let c = 0; c < 8; c++)
36        if (
37          Math.sign(board[r][c]) === p &&
38          getValidMovesForPiece(r, c, false, board).length > 0
39        )
40          return true;
41    return false;
42  }
```

Wenn diese Funktion false liefert, wird sofort das Game Over Overlay aufgerufen und verkündet den Sieger.

Das war jetzt ein tiefer Einblick in die Mechanik! Jetzt bin ich auch etwas erschöpft. Sozusagen bin ich ja ein Alleinunterhalter und einen Lektor habe ich auch nicht. Noch einen Hinweis! Fehlerfrei kann diese Software nicht sein! Ohne Gewähr!

Liebe Leserinnen und Leser. Hallo du!

Zum Schluss möchte ich die Karten auf den Tisch legen. Mit der Programmierung in Python und C++ beschäftige ich mich ja schon viele Jahre. Jedoch mit der JavaScript-Programmierung habe ich erst im Jahr 2022 angefangen. JavaScript ist Neuland für mich. Einige Programmierer, die meine Webseite besuchten, haben sich dahingehend geäußert: „Guck an! Man sieht das er aus der Python-Ecke kommt“! „Vom strukturellen schlanken JavaScript versteht er noch nicht viel“. Zurück zu der Implementierung des Brettspieles „Dame & Mühle“. Vor Jahren habe ich das Brettspiel mit Python bereits gemacht. Mitte des Jahres 2025 habe ich dann begonnen diese Spiele mit JavaScript umzusetzen. Die Implementierung für zwei Spieler hätte ich schon geschafft, jedoch die Programmintegration „ein Spieler gegen den Computer“, das war eine echte Herausforderung! Ich brauchte Hilfe! Folgende Quellen habe ich angezapft: Gravitar (https://www.youtube.com/@Gravitar), Github.com und https://stackoverflow.com/. Ich habe für jedes Spiel einen kleinen Fahrplan erstellt mit kleinen Funktion-Bausteinen. Diese sogenannten Code-Snippets habe ich dann kopiert. Einen kleinen Fragekatalog erstellt und Google Gemini mitgeteilt. Die Resonanz sah dann meisten folgendermaßen aus: Geht mit CSS besser, einfacher. Mach es mit Listen in Listen. Soll ich dir das als Class umschreiben? Usw. Der KI-Service ist ja noch kostenlos. In ein paar Jahren kostet das mit Sicherheit Geld! Trotz alledem, der Debugger ist schon ziemlich heiß gelaufen! Wie bereits oben erwähnt habe ich in der Jahresmitte 2025 damit angefangen, mit vielen Unterbrechungen und jetzt Februar 2026 ist es soweit fertig.

Februar 2026 Hans Busche