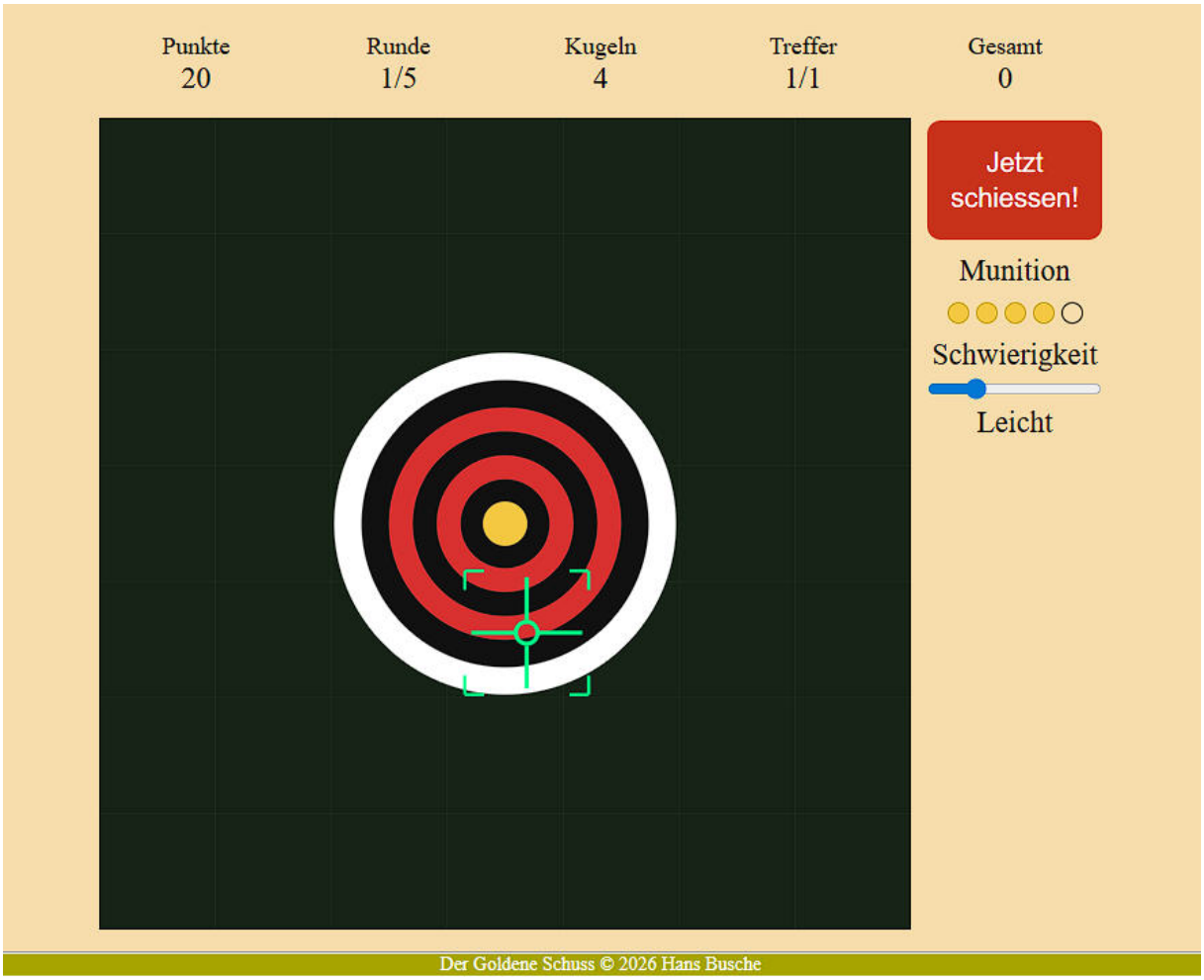
Der Goldene Schuss

"Der goldene Schuss" war ein absoluter Klassiker und ein Meilenstein der deutschen Fernsehgeschichte. Die Spielshow aus den 1960er-Jahren war die erste interaktive Mitmach-Show, bei der Zuschauer per Telefon aktiv ins Spielgeschehen eingreifen konnten. Im Zentrum der Sendung stand eine per Kamera ausgerichtete Armbrust. Der Kandidat am Telefon dirigierte den Kameramann mit Kommandos wie "Links, rechts, hoch, runter, Schuss". Das Ziel war es, die Mitte einer Zielscheibe zu treffen.



Programmiert man so ein Spiel auf einem PC, mit der Programmiersprache Python oder C++, dann kann man mit den vorhandenen Sprachbibliotheken (Mikrofon / Audio) den „Goldenen Schuss“ sehr gut nachbauen. Das ist hier aber nicht meine Strategie. Ich baue nichts mit Mikrofon und Webseiten.

Hier eine kurze Einleitung, wie dieses Spiel funktioniert: Über einer Zielscheibe schwebt ein Fadenkreuz (Visier). Für das Schweben sorgt ein Zufallsgenerator. Dieser, in JavaScript programmierte Generator, ändert sein Verhalten in Abhängigkeit eines Reglers für die Schwierigkeitsstufen: „sehr leicht bis Profi“. Gespielt werden fünf Runden mit jeweils fünf Schüssen. Nach jeder Runde wird die Zielscheibe etwas kleiner. Jedoch bleibt sie immer in der Mitte des Spielfeldes.

Die Strategie: Du bist ein Beobachter, Ermittler, Jäger oder Tierfotograf. In Ordnung! Mein Wunsch! Du bist ein Tierfotograf. Du liegst auf der Lauer und beobachtest durch die Fotolinse ein Rotschwänzchen. Genau zum Zeitpunkt, wenn es den Nachwuchs füttert, Übergabe der Nahrung in den Schnabel (er ist gelb) des Rotschwänzchen-Babys, drückst du den Auslöser. So will es dein Auftraggeber haben. Eine Punktlandung. Zeit spielt keine Rolle. So wird auch das „Goldene-Schuss-Spiel" gespielt. Visier beobachten. Ist das Fadenkreuz genau über dem gelben Punkt. Schuss auslösen. Für den Schuss gibt es zwei Möglichkeiten: Roter Schalter, mit der Beschriftung „jetzt schießen". Oder die Leertaste betätigen. Ist der Schieberegler auf „sehr leicht" eingestellt, dann hast du immer einen Erfolg, weil das Fadenkreuz innerhalb der Zielscheibe bleibt. Für den Goldenen Schuss wird deine Konzentration und das Warten („ein Jäger") auf die Probe gestellt. In der „Profi-Stellung" macht das Visier sehr schnelle Bewegungen und schwebt auch in die Bereiche außerhalb der Zielscheibe.

Nach jeder Spielrunde, insgesamt sind es fünf, wird die Gesamtpunktzahl (Highscore) angezeigt. Es entsteht eine kurze Pause. Zielscheibe wird neu, „verkleinert", ausgerichtet. Zählerstände zurückgesetzt. Nach fünf Durchläufen stoppt die Applikation mit einer Text-Zusammenfassung und ein Schalter wird visualisiert: „Nochmal Spielen?".

Das Spiel besteht aus drei Dateien: HTML (3 kByte) , CSS (4 kByte) und JavaScript (13 kByte). Weiter unten in dieser Beschreibung möchte ich dir CSS und JavaScript etwas näher erläutern. Die größte Herausforderung war die Implementierung der Visiereinrichtung, das Schweben des Fadenkreuzes.

JavaScript

Zeigt der Webbrowser „Google Chrome" das Spiel, öffnet sich durch Betätigung der Taste F12 ein Werkzeug mit dem Namen „DevTools". Danach betätigt man den Reiter „Quellencode". Das Fenster wird, wie bei dem Explorer, auf ein Auswahlmenü erweitert. Selektiere die Datei „script.js". Der gesamte Inhalt der JavaScript-Datei, mit Zeilennummer, wird dir nun angezeigt. Diese Vorgehensweise ist für das Verständnis der weiteren Beschreibung notwendig!

- **Variablen und Einstellungen:**

  Am Anfang werden alle wichtigen Werte festgelegt: die Schwierigkeitsstufen mit ihren Geschwindigkeiten, die Anzahl der Kugeln (5) und Runden (5). Außerdem werden alle HTML-Elemente „gesucht" und in Variablen gespeichert, damit der Code später darauf zugreifen kann – zum Beispiel der Canvas (die Zeichenfläche), der Schuss-Button und die Anzeigen.

- **resize() - Größe anpassen:**

  Diese Funktion misst, wie groß die Spielfläche gerade ist, und passt den Canvas entsprechend an. Die Zielscheibe wird dabei immer genau in die Mitte gesetzt. Sie wird auch aufgerufen, wenn der Browser *verkleinert oder vergrößert* wird.

- **drawBg() - Hintergrund zeichnen:**

Zeichnet den dunklen grünen Hintergrund und ein dezentes Gittermuster darüber – rein zur Optik.

- **drawTarget() - Zielscheibe zeichnen:**

Zeichnet die Zielscheibe als mehrere konzentrische Kreise (Ringe) übereinander – von groß nach klein, abwechselnd weiß, schwarz und rot, mit dem goldenen Mittelpunkt ganz oben. Die Größe der Scheibe wird kleiner, je höher die Runde ist.

- **drawVisier() - Fadenkreuz zeichnen:**

Zeichnet das grüne Fadenkreuz: vier Linien (oben, unten, links, rechts) mit einer Lücke in der Mitte, einem kleinen Kreis im Zentrum und vier Eckmarkierungen. Es wird immer an der aktuellen Visier-Position gezeichnet.

- **moveVisier() – Visier bewegen:**

Das ist das Herzstück des Spiels. Vier interne Zähler (`vOscA` bis `vOscD`) werden bei jedem Frame ein kleines Stück erhöht – mit einem leichten Zufallsanteil. Aus diesen Zählern werden über Sinus- und Kosinus Funktionen verschlungene, unvorhersehbare Bewegungen berechnet. Das Ergebnis: Das Visier wandert wellenförmig und zufällig über die Scheibe. Je höher die Schwierigkeit, desto größer der Zufallsanteil und desto schneller die Bewegung.

- **doShoot() - Schuss auslösen:**

Wenn der Spieler den roten Knopf drückt, passiert Folgendes: Eine Kugel wird abgezogen, der Schusszähler erhöht. Dann wird gemessen, wie weit das Visier gerade vom Mittelpunkt der Scheibe entfernt ist (mit dem Satz des Pythagoras). Je nach Entfernung gibt es unterschiedlich viele Punkte – von 2 Punkten am Rand bis zu 100 Punkten für den Bullseye. Danach erscheinen Partikeleffekte und eine Nachricht.

- **calcPoints() - Punkte berechnen:**

Berechnet anhand des Abstands vom Mittelpunkt, in welchem Ring das Visier war, und gibt die entsprechenden Punkte sowie eine Nachricht zurück.

- **spawnParticles() & drawParticles() - Funkeneffekt:**

Beim Schuss werden kleine Partikel erzeugt, die in alle Richtungen wegfliegen, langsam nach unten fallen (Schwerkraft) und immer durchsichtiger werden, bis sie verschwinden.

- **loop() - die Spielschleife:**

Diese Funktion läuft dauerhaft – ca. 60-mal pro Sekunde. Bei jedem Durchlauf wird der Canvas geleert und neu gezeichnet: Hintergrund, Zielscheibe, Visier, Partikel und Nachrichten. So entsteht die flüssige Animation. *`requestAnimationFrame`* sorgt dafür, dass der Browser den nächsten Durchlauf genau zum richtigen Zeitpunkt startet.

- initRound() & nextRound() – Rundensteuerung:

`initRound()` bereitet eine neue Runde vor: Schwierigkeit auslesen, Visier-Startposition setzen, Zufallswerte für die Bewegung würfeln. `nextRound()` erhöht den Rundenzähler und prüft, ob das Spiel schon vorbei ist.

- startGame() & endGame() - Spielstart und -ende:

`startGame()` setzt alle Werte zurück und blendet das Startmenü aus. `endGame()` stoppt die Animation, prüft ob ein neuer Highscore erreicht wurde, und zeigt das Ergebnisbildschirm an.

- Event-Listener - auf Aktionen reagieren:

Am Ende wird festgelegt, auf welche Benutzeraktionen das Spiel reagieren soll: Klick auf den Schuss-Button, Klick auf „Spiel starten", Änderung des Schwierigkeitsreglers und Größenänderung des Browserfensters.

JavaScript: moveVisier()! Das Herzstück des Spiels. 30 Stunden recherchiert und ausprobiert. Etliche Varianten zusammengeschustert.

Das Grundprinzip:

Stell dir vor, du hältst einen Stift und zeichnest mit geschlossenen Augen Kreise und Schleifen auf ein Blatt Papier. Deine Hand bewegt sich – aber nie völlig unkontrolliert, sondern in weichen, fließenden Kurven. Genau das macht moveVisier().

Die vier Zähler:

```
vOscA += vErratic * (0.7 + Math.random() * 0.6);
vOscB += vErratic * (0.5 + Math.random() * 0.8);
vOscC += vErratic * (0.3 + Math.random() * 0.4);
vOscD += vErratic * (0.9 + Math.random() * 0.3);
```

vOscA bis vOscD sind vier unabhängige Zähler, die bei jedem Frame (60 × pro Sekunde) ein kleines Stück wachsen. Jeder wächst in einem anderen Tempo – und durch Math.random() kommt jedes Mal ein kleiner Zufallswert dazu. Dadurch ticken die vier Zähler nie gleich und nie vorhersehbar. vErratic bestimmt, wie schnell alle vier wachsen – das ist die Schwierigkeit. Bei „Sehr leicht" wachsen sie kaum, bei „Profi" springen sie wild.

Sinus und Kosinus – die Wellenerzeuger:

```
const dx = Math.sin(vOscA) * Math.cos(vOscC * 1.3) * r * 1.05
        + Math.cos(vOscB * 0.7) * r * 0.45
        + Math.sin(vOscD * 2.1) * r * 0.2;
```

Math.sin() und Math.cos() liefern immer einen Wert zwischen -1 und +1 – sie schwingen wie eine Welle auf und ab. Wenn man mehrere solche Wellen addiert, entstehen komplexe, verschlungene Kurven – ähnlich wie Wellen auf einem See, die sich gegenseitig überlagern. Hier werden drei solcher Wellen für die X-Richtung addiert, und drei weitere

für die Y-Richtung (dy). Jede Welle hat eine andere Gewichtung (* r * 1.05, * r * 0.45, * r * 0.2), damit manche Wellen stärker wirken als andere.

Die finale Position:

```
visier.x = Math.max(8, Math.min(W-8, target.x + dx * vSpeed * 0.5));
visier.y = Math.max(8, Math.min(H-8, target.y + dy * vSpeed * 0.5));
```

Die berechneten Wellen-Werte dx und dy werden zum Mittelpunkt der Zielscheibe addiert. Das Visier bewegt sich also immer um die Scheibe herum – mal weiter weg, mal genau in der Mitte. vSpeed streckt die Bewegung – bei hoher Geschwindigkeit schlägt das Visier weiter aus. Math.max und Math.min sorgen dafür, dass das Visier nie aus dem Spielfeld herauswandert.

Warum klingt das so natürlich?

Weil echte Zufallszahlen (Math.random()) ruckartig und unschön wirken würden. Sinus-Wellen hingegen sind glatt und fließend – das Visier zittert nie abrupt, sondern gleitet sanft. Die Kombination aus vier verschiedenen Wellen mit Zufallsanteil erzeugt eine Bewegung, die unvorhersehbar aber trotzdem flüssig wirkt – genau wie eine echte zitternde Hand beim Zielen.

CSS

Schaut man sich meine früheren HTML-CSS-JavaScript-Applikationen an, so stellt man fest, dass der CSS-Part stiefmütterlich behandelt wurde. Lieber Bilden und Canvas, weniger CSS. Mittlerweile ist da meine Lernkurve gestiegen. Hier nun die CSS-Highlights in Worten erklärt:

```
#wrapper {
  display: flex;
  align-items: center;
  justify-content: center;
  min-height: 100vh;
  width: 100%;
}
```

Das ist die Lösung für die Bildschirmmitte. display: flex schaltet das sogenannte Flexbox-System ein – ein modernes Layout-Werkzeug. align-items: center zentriert den Inhalt vertikal, justify-content: center zentriert ihn horizontal. min-height: 100vh bedeutet: der Wrapper ist mindestens so hoch wie der gesamte Bildschirm (vh = viewport height). Zusammen sorgen diese vier Zeilen dafür, dass das Spiel immer exakt in der Mitte sitzt.

```
#game {
  width: 100%;
  max-width: 540px;
}
```

Das Spiel nimmt immer die volle verfügbare Breite ein – aber nie mehr als 540 Pixel. Auf kleinen Bildschirmen passt es sich an, auf großen Bildschirmen bleibt es kompakt und übersichtlich.

```
#scorebar {
  display: grid;
  grid-template-columns: repeat(5, 1fr);
  gap: 6px;
}
```

display: grid ist ein weiteres Layout-System – ideal für gleichmäßige Aufteilungen. repeat(5, 1fr) bedeutet: erstelle 5 gleich breite Spalten, wobei 1fr für „einen gleichmäßigen Anteil" steht. Alle fünf Werte (Punkte, Runde, Kugeln, Treffer, Highscore) haben dadurch exakt dieselbe Breite.

```
#layout {
  display: flex;
  gap: 12px;
  align-items: flex-start;
}
```

Wieder Flexbox – diesmal um die Spielfläche und die rechte Seitenleiste nebeneinander zu platzieren. gap: 12px sorgt für einen Abstand zwischen beiden. align-items: flex-start bewirkt, dass die Seitenleiste oben beginnt und nicht künstlich gestreckt wird.

```
#shoot-btn {
  width: 100%;
  padding: 18px 10px;
  background: #c8321a;
  color: #fff;
  border: none;
  border-radius: 10px;
  cursor: pointer;
}
#shoot-btn:hover:not(:disabled) { background: #a82510; }
#shoot-btn:active:not(:disabled) { transform: scale(0.97); }
#shoot-btn:disabled { background: var(--color-background-secondary); cursor: not-allowed; }
```

Der rote Button hat großzügiges padding damit er gut klickbar ist. :hover verändert die Farbe beim Darüberfahren mit der Maus. :active lässt den Button beim Klicken minimal kleiner werden (scale(0.97)) – das gibt ein natürliches „Drück-Gefühl". :disabled macht ihn grau und zeigt einen Sperr-Cursor, wenn keine Kugeln mehr übrig sind. Das :not(:disabled) stellt sicher, dass die Hover- und Klick-Effekte nur aktiv sind, wenn der Button auch wirklich benutzbar ist.

```
#overlay {
  position: absolute;
  inset: 0;
```

```
  display: flex;
  align-items: center;
  justify-content: center;
  background: rgba(0,0,0,0.78);
}
```

position: absolute und inset: 0 bedeuten: das Overlay legt sich exakt über die gesamte Spielfläche – von Ecke zu Ecke. Der Hintergrund ist schwarz mit 78% Deckkraft (rgba mit dem letzten Wert 0.78), sodass die Zielscheibe dahinter noch leicht durchscheint. Flexbox zentriert den Text und den Button darin.
Im gesamten Code tauchen Werte wie var(--color-background-secondary) oder var(--border-radius-md) auf. Das sind CSS-Variablen – vordefinierte Werte aus dem Design-System dieser Oberfläche. Der Vorteil: Wechselt der Nutzer z.B. zwischen hellem und dunklem Modus, passen sich alle diese Farben automatisch an, ohne dass eine einzige Zeile Code geändert werden muss.

Kurz zusammengefasst: Flexbox kümmert sich um das Nebeneinander und die Zentrierung, Grid sorgt für die gleichmäßige Punkteanzeige, und die Button-Zustände geben dem Spiel ein lebendiges, interaktives Gefühl.

Hans Busche, Juni 2026