

p5.js Spiele-Prompts (Einfache Version)

Optimiert für kostenlose LLMs - nur p5.js, keine zusätzlichen Bibliotheken

Benötigte HTML-Datei

Die index.html muss nur p5.js einbinden:

```
<script src="https://cdn.jsdelivr.net/npm/p5@1.11.11/lib/p5.js"></script>
```

1. Fang-Spiel

Erstelle mit **p5.js** ein Fang-Spiel. Schreibe nur den Code für die **sketch.js**.

Aufbau:

- Canvas: 600x400 Pixel
- Korb: Rechteck (80x20), bewegt sich mit Pfeiltasten links/rechts
- Früchte: Kreise (Durchmesser 30), fallen von oben

Spiellogik:

- Früchte spawnen alle 60 Frames an zufälliger x-Position
- Früchte fallen mit 3 Pixel pro Frame
- Kollision prüfen mit dist()-Funktion
- Gefangen = +10 Punkte, Boden erreicht = -1 Leben
- Start: 3 Leben, Ende bei 0 Leben

Anzeige:

- Punkte und Leben oben links mit text()
- Game-Over-Text in der Mitte, Neustart mit R-Taste

Stelle bei Bedarf Rückfragen.

2. Ausweich-Spiel

Erstelle mit **p5.js** ein Ausweich-Spiel. Schreibe nur den Code für die **sketch.js**.

Aufbau:

- Canvas: 800x400 Pixel
- Spieler: Quadrat (40x40) bei x=100, auf dem Boden
- Boden: Linie bei y=350
- Hindernisse: Rechtecke (30x50)

Spiellogik:

- Leertaste = Sprung (Spieler bewegt sich hoch, dann runter durch Schwerkraft)
- Einfache Schwerkraft: velocityY += 0.8, playerY += velocityY
- Hindernisse spawnen alle 90 Frames rechts, bewegen sich mit 5 Pixel/Frame nach links

- Kollision = Spielende
- Score = überlebte Sekunden (frameCount / 60)

Anzeige:

- Score oben rechts
- Game-Over mit Score, Neustart mit R-Taste

Stelle bei Bedarf Rückfragen.

3. Klick-Spiel

Erstelle mit **p5.js** ein Klick-Spiel. Schreibe nur den Code für die **sketch.js**.

Aufbau:

- Canvas: 600x600 Pixel
- Ziele: Kreise (Durchmesser 50) an zufälligen Positionen

Spiellogik:

- Ein Ziel erscheint, verschwindet nach 90 Frames
- Mausklick auf Ziel ($\text{dist}() < 25$) = +10 Punkte, neues Ziel
- Ziel verschwindet ohne Klick = -1 Leben
- Start: 5 Leben, Ende bei 0 Leben
- Zeitlimit pro Ziel wird alle 5 Treffer kürzer (90 -> 75 -> 60 Frames)

Anzeige:

- Punkte und Leben oben
- Ziele in zufälligen Farben
- Game-Over mit Punktestand

Stelle bei Bedarf Rückfragen.

4. Pong (2 Spieler)

Erstelle mit **p5.js** ein 2-Spieler-Pong. Schreibe nur den Code für die **sketch.js**.

Aufbau:

- Canvas: 800x400 Pixel
- Paddles: Rechtecke (15x80), links bei $x=30$, rechts bei $x=755$
- Ball: Kreis (Durchmesser 20)

Steuerung:

- Spieler 1 (links): W = hoch, S = runter
- Spieler 2 (rechts): Pfeiltasten hoch/runter
- Paddle-Geschwindigkeit: 6 Pixel pro Frame

Spiellogik:

- Ball startet in der Mitte mit zufälliger Richtung
- Ball prallt oben/unten ab ($\text{ballSpeedY} *= -1$)

- Ball prallt von Paddles ab (Kollision prüfen mit Rechteck-Überlappung)
- Ball links/rechts raus = Punkt für Gegner
- Erster mit 5 Punkten gewinnt

Anzeige:

- Punktestand oben Mitte
- Gestrichelte Mittellinie
- Gewinner-Text am Ende

Stelle bei Bedarf Rückfragen.

5. Snake

Erstelle mit **p5.js** ein Snake-Spiel. Schreibe nur den Code für die **sketch.js**.

Aufbau:

- Canvas: 400x400 Pixel
- Raster: 20x20 Pixel pro Feld
- Schlange: Array von {x, y} Positionen
- Futter: Ein Quadrat an zufälliger Rasterposition

Steuerung:

- Pfeiltasten ändern die Richtung
- Keine 180°-Wendung erlaubt

Spiellogik:

- Schlange bewegt sich alle 10 Frames ein Feld weiter
- Futter gefressen = Schlange wächst um 1, neues Futter
- Kollision mit Wand oder sich selbst = Spielende
- Score = Länge der Schlange

Anzeige:

- Schlange grün, Futter rot
- Score oben
- Game-Over mit Neustart (R-Taste)

Stelle bei Bedarf Rückfragen.

6. Breakout

Erstelle mit **p5.js** ein Breakout-Spiel. Schreibe nur den Code für die **sketch.js**.

Aufbau:

- Canvas: 600x500 Pixel
- Paddle: Rechteck (100x15) am unteren Rand, folgt der Maus (mouseX)
- Ball: Kreis (Durchmesser 15)
- Blöcke: 5 Reihen à 8 Blöcke (je 70x20), oben mit Abstand

Spiellogik:

- Ball bewegt sich mit ballSpeedX und ballSpeedY (Start: 4 und -4)
- Ball prallt an Wänden und Paddle ab
- Blockkollision: Block entfernen, Ball prallt ab, +10 Punkte
- Ball unten raus = -1 Leben (Start: 3 Leben)
- Alle Blöcke weg = Gewonnen

Anzeige:

- Jede Blockreihe in anderer Farbe
- Punkte und Leben oben
- Gewonnen/Verloren-Text am Ende

Stelle bei Bedarf Rückfragen.

7. Space Shooter

Erstelle mit **p5.js** einen Space Shooter. Schreibe nur den Code für die **sketch.js**.

Aufbau:

- Canvas: 500x600 Pixel
- Raumschiff: Dreieck (triangle()) am unteren Rand
- Schüsse: Array von {x, y} Objekten
- Gegner: Array von {x, y} Objekten (Rechtecke 40x30)

Steuerung:

- Pfeiltasten links/rechts = Schiff bewegen
- Leertaste = Schießen (max. alle 15 Frames)

Spiellogik:

- Gegner spawnen alle 60 Frames oben an zufälliger x-Position
- Gegner bewegen sich mit 2 Pixel/Frame nach unten
- Schüsse bewegen sich mit 7 Pixel/Frame nach oben
- Schuss trifft Gegner = beide entfernen, +10 Punkte
- Gegner erreicht unteren Rand = -1 Leben
- Start: 3 Leben

Anzeige:

- Schwarzer Hintergrund mit weißen Punkten (Sterne)
- Punkte und Leben oben

Stelle bei Bedarf Rückfragen.

8. Flappy Bird Klon

Erstelle mit **p5.js** einen Flappy Bird Klon. Schreibe nur den Code für die **sketch.js**.

Aufbau:

- Canvas: 400x600 Pixel

- Vogel: Kreis (Durchmesser 30) bei x=80
- Röhren: Array von {x, gapY} - zwei Rechtecke mit Lücke

Steuerung:

- Leertaste oder Mausklick = Vogel fliegt hoch (velocityY = -8)

Spiellogik:

- Schwerkraft: velocityY += 0.4, birdY += velocityY
- Röhren: Breite 60, Lücke 150 Pixel hoch
- Neue Röhre alle 100 Frames, gapY zufällig zwischen 100 und 400
- Röhren bewegen sich mit 3 Pixel/Frame nach links
- Kollision mit Röhre oder Boden/Decke = Spielende
- Röhre passiert = +1 Punkt

Anzeige:

- Hellblauer Hintergrund
- Grüne Röhren
- Gelber Vogel
- Score groß in der Mitte oben

Stelle bei Bedarf Rückfragen.

Tipps bei Problemen

Falls das Modell Probleme hat:

1. Noch einfacher anfangen:

"Erstelle nur einen bewegbaren Spieler mit Pfeiltasten. Kein Score, keine Gegner."

2. Schritt für Schritt:

"Füge jetzt einen fallenden Kreis hinzu, der bei Kollision verschwindet."

3. Konkrete Funktionen nennen:

"Nutze rect() für Rechtecke, ellipse() für Kreise, keysDown() für Tastatureingaben."

4. Fehler beheben lassen:

"Der Ball prallt nicht korrekt ab. Prüfe die Kollisionsbedingungen."