

AI Chats Workbook: Pong in 2025

Dr. Yves J. Hilpisch¹

December 11, 2025

¹Get in touch: <https://linktr.ee/dyjh>. Web page <https://hilpisch.com>. Research, structuring, drafting, and visualizations were assisted by GPT 5.1 as a co-writing tool under human direction.

Contents

1	Time Travelling from the C64 to Claude	1
2	The Prompt that Starts Everything	1
3	Claude's Single-Shot Reply	1
4	How the LLM Planned and Built the Game	3
4.1	HTML Skeleton: Canvas and Controls	3
4.2	CSS: Readable Retro-Futurism	4
4.3	JavaScript: Game Loop and Physics	4
4.4	Technologies Involved	6
5	From 1982 to 2025: What to Notice	6
A	Appendix: Full HTML/CSS/JS Listing	7

1 Time Travelling from the C64 to Claude

Fast-forward to November 2025. The C64 workbook sits on our desk, still smelling faintly of 8-bit ozone. Yet we now have a browser tab open to Claude.ai. The goal is the same as in the 1980s: put Pong on a screen. The means are radically different: an LLM that drafts HTML, CSS, and JavaScript in one shot. This workbook mirrors the layout of the C64 guide while documenting the AI-assisted leap.

What Changes in 2025?

- The “assembler” is now a prompt. A few sentences become a runnable game.
- Iteration cycles shrink to seconds; the canvas appears in one response (Figure 1).
- The full source arrives formatted for copy/paste (Figure 2), ready to run locally and in any browser (Figure 3).

The compiled PDF of this workbook is available at <https://hilpisch.com/chats.pdf>.

2 The Prompt that Starts Everything

I open Claude.ai, paste the prompt stored in `prompt.txt`, and hit *Enter*. The listing below is exactly what the model saw.

```
Create a single-page HTML Pong game for a single player only
where the player on the left is playing against a fixed wall
on the right. Add a slider to adjust the speed of the ball.
Paddle up is Q, down is A. Keep score, maximum 3 lives.
```

3 Claude's Single-Shot Reply

Claude returns a fully rendered canvas view, matching the prompt in one go; Figure 1 captures the moment. Beneath the preview sits the complete HTML document (Figure 2), with CSS for styling and JavaScript for physics, controls, and scoring. Saving the file as `pong.html` and opening it locally yields a playable, single-player Pong against a wall (Figure 3).

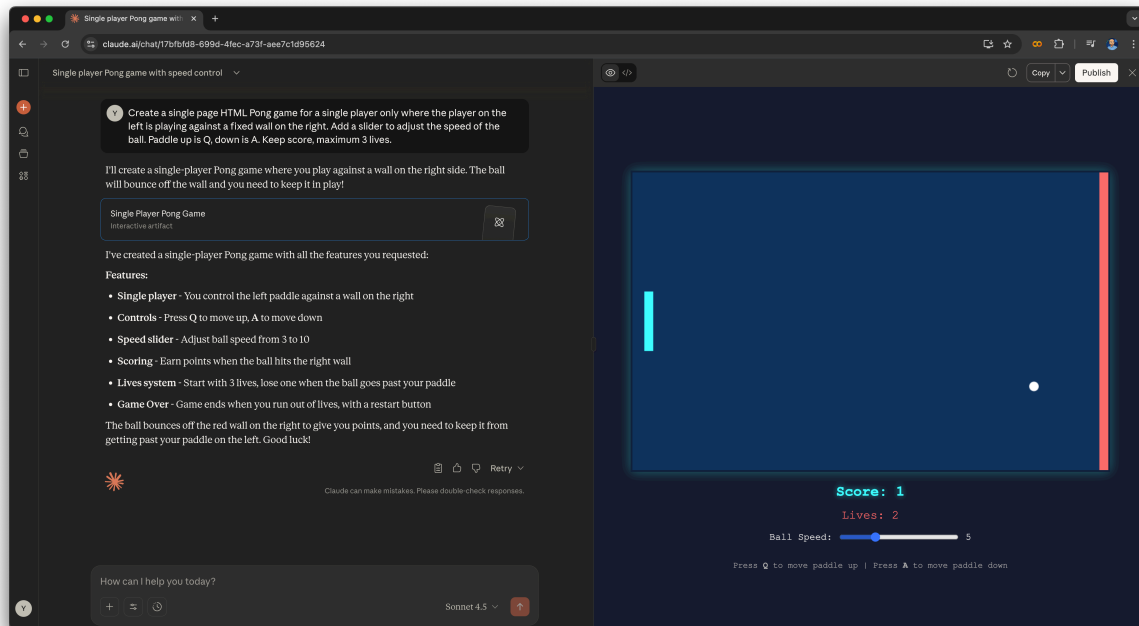


Figure 1: Claude renders the Pong canvas immediately after receiving the prompt.

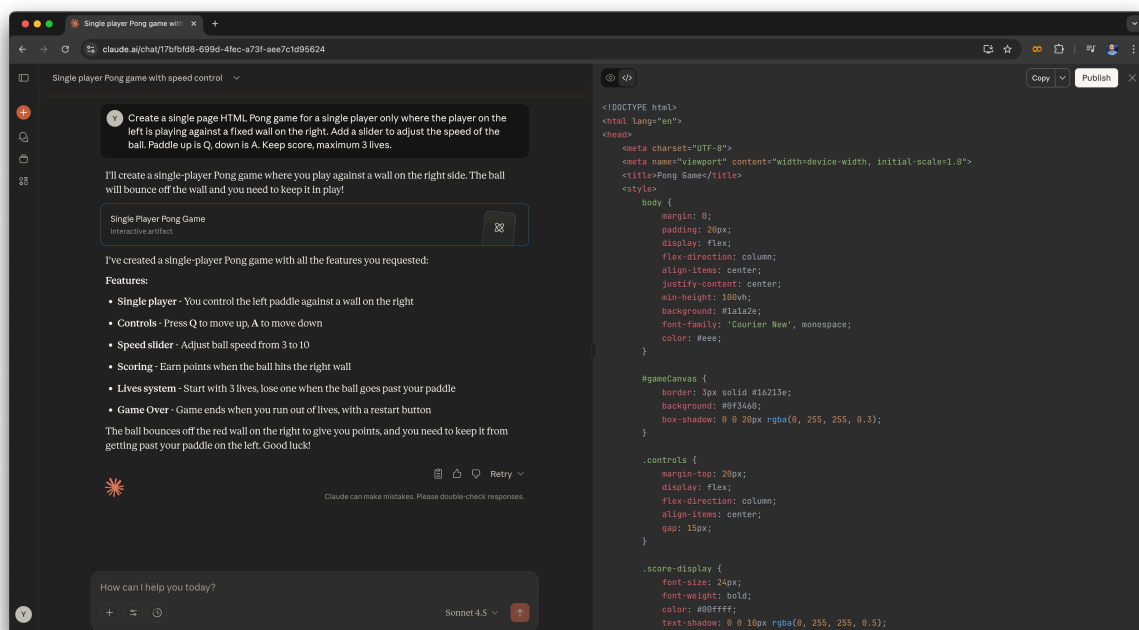


Figure 2: The same reply includes the full HTML/CSS/JS code, ready to copy.



Figure 3: Local play in any browser: load pong.html and start.

4 How the LLM Planned and Built the Game

To replicate the workflow for your own projects, it helps to unpack what the model implicitly planned and implemented. The full code is in the appendix; here we walk through the main pieces.

4.1 HTML Skeleton: Canvas and Controls

Claude scaffolded a minimal document: a canvas, heads-up display, slider, and restart button. The core elements are shown below.

HTML Skeleton

```
<!-- Main drawing surface -->
<canvas id="gameCanvas" width="800" height="500"></canvas>

<!-- Heads-up display and controls -->
<div class="controls">
  <div class="score-display">Score: <span id="score">0</span></div>
  <div class="lives-display">Lives: <span id="lives">3</span></div>

  <div class="speed-control">
    <label for="speedSlider">Ball Speed:</label>
    <input type="range" id="speedSlider" min="3" max="10" value="5" step="0.5">
    <span id="speedValue">5</span>
  </div>
</div>
```

```

<div class="instructions">Press Q to move up, A to move down</div>
<div id="gameOverText" class="game-over" style="display: none;">GAME OVER</div>
<button id="restartBtn" style="display: none;">Restart Game</button>
</div>

```

This answers the prompt requirements directly: single-page app, speed slider, Q/A controls, score, and a finite life counter.

4.2 CSS: Readable Retro-Futurism

The styling mixes neon cyan on dark blue to echo CRT phosphor while remaining legible. Key rules:

CSS Highlights

```

body {
  display: flex;
  flex-direction: column;          /* Stack canvas and controls */
  align-items: center;              /* Center everything horizontally */
  background: #1a1a2e;              /* Dark night-sky background */
  color: #eee;
  font-family: 'Courier New', monospace;
}

#gameCanvas {
  border: 3px solid #16213e;
  background: #0f3460;              /* Slightly lighter playfield blue */
  box-shadow: 0 0 20px rgba(0, 255, 255, 0.3); /* Glowing edge */
}

.score-display {
  color: #00ffff;
  text-shadow: 0 0 10px rgba(0,255,255,0.5);
}

.lives-display { color: #ff6b6b; }

```

This minimal palette communicates “retro arcade” without extra assets, keeping the download to one file.

4.3 JavaScript: Game Loop and Physics

The logic fits into a tight event loop that updates paddle motion, ball physics, scoring, and life management. Three snippets capture the core: one that defines the state objects, one that maps keyboard events into that state, and one that advances the ball and handles collisions.

First, the game needs explicit objects to remember where the paddle and ball live and how fast they move.

Game State Setup

```

// Geometry and movement parameters for paddle and ball
const paddle = {
  x: 20,
  y: canvas.height / 2 - 50,
  width: 15,
  height: 100,
  speed: 8                      // pixels per frame when moving
}

```

```

};

const ball = {
  x: canvas.width / 2,
  y: canvas.height / 2,
  radius: 8,
  dx: baseSpeed,      // horizontal velocity
  dy: baseSpeed * 0.6, // vertical velocity (not too steep)
  speed: baseSpeed
};

// Tracks which keys are currently pressed
const keys = {};

```

Next, the browser's keyboard events are translated into simple boolean flags in the `keys` object so that the game loop can stay focused on movement instead of raw events.

Keyboard Input Handling

```

// Keyboard handlers: remember when Q/A are pressed or released
document.addEventListener('keydown', event => {
  const key = event.key.toLowerCase();
  if (key === 'q' || key === 'a') {
    keys[key] = true;
  }
});

document.addEventListener('keyup', event => {
  const key = event.key.toLowerCase();
  if (key === 'q' || key === 'a') {
    keys[key] = false;
  }
});

function updatePaddle() {
  // Move up while Q is held, but not off the screen
  if (keys['q'] && paddle.y > 0) {
    paddle.y -= paddle.speed;
  }
  // Move down while A is held, but keep the paddle fully visible
  if (keys['a'] && paddle.y < canvas.height - paddle.height) {
    paddle.y += paddle.speed;
  }
}

```

Finally, the ball update routine ties everything together by stepping the ball, bouncing it off edges, reacting to paddle hits, and triggering lives and game-over logic when the player misses.

Ball Update and Collisions

```

function updateBall() {
  // Advance ball by its current velocity
  ball.x += ball.dx;
  ball.y += ball.dy;

  // Bounce off top and bottom edges

```

```

if (ball.y - ball.radius < 0 || ball.y + ball.radius > canvas.height) {
  ball.dy *= -1;
}

// Bounce off the right wall and increment score
if (ball.x + ball.radius > canvas.width - 15) {
  ball.dx *= -1;
  score++;
  scoreDisplay.textContent = score;
}

// Check collision with the left paddle
const hitPaddle =
  ball.x - ball.radius < paddle.x + paddle.width &&
  ball.x + ball.radius > paddle.x &&
  ball.y > paddle.y &&
  ball.y < paddle.y + paddle.height;

if (hitPaddle) {
  // Reflect horizontally and adjust vertical angle based on hit position
  ball.dx = Math.abs(ball.dx);
  const hitPos = (ball.y - paddle.y) / paddle.height; // 0 at top, 1 at bottom
  ball.dy = (hitPos - 0.5) * baseSpeed * 1.5;
}

// Ball missed the paddle: lose a life or end the game
if (ball.x - ball.radius < 0) {
  lives--;
  livesDisplay.textContent = lives;
  lives <= 0 ? endGame() : resetBall();
}
}

```

In one pass the model set up canvas drawing, input handling, collision detection, scoring, life decrement, and restart logic—the full micro loop that once required pages of BASIC or assembly on the C64.

4.4 Technologies Involved

Stepping back from the individual snippets, it is helpful to name the main browser technologies that Claude combined to produce the final game. Seeing them side by side makes the contrast with C64-era tools very clear.

- **HTML5 canvas** for rendering paddle, wall, and ball each frame.
- **Vanilla JavaScript** for the game loop, collision physics, DOM updates, and slider event handling.
- **CSS** for layout and a cohesive retro-futuristic skin without external assets.
- **Browser APIs** (`requestAnimationFrame` and event listeners) replacing the C64's raster interrupts and joystick registers.

5 From 1982 to 2025: What to Notice

On the C64, Pong meant typing opcodes and poking memory. In 2025, a short prompt turns into a fully styled browser game in seconds. The learning task changes: instead of assembling

bytes, we learn to steer an LLM, read the generated code critically, and iterate on design. The spirit of tinkering survives; the tools shift from a blue BASIC screen to an AI chat window.

A Appendix: Full HTML/CSS/JS Listing

The complete file `pong.html` generated by Claude is included below for reference. The tiny font keeps the single-file structure more compact so you can see large portions of the document at once while still fitting comfortably in the layout.

Full Browser Implementation

```
<!-- Created with Claude AI on November 26, 2025 -->

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Pong Game</title>
    <style>
        body {
            margin: 0;
            padding: 20px;
            display: flex;
            flex-direction: column;
            align-items: center;
            justify-content: center;
            min-height: 100vh;
            background: #1a1a2e;
            font-family: 'Courier New', monospace;
            color: #eee;
        }

        #gameCanvas {
            border: 3px solid #16213e;
            background: #0f3460;
            box-shadow: 0 0 20px rgba(0, 255, 255, 0.3);
        }

        .controls {
            margin-top: 20px;
            display: flex;
            flex-direction: column;
            align-items: center;
            gap: 15px;
        }

        .score-display {
            font-size: 24px;
            font-weight: bold;
            color: #0fffff;
            text-shadow: 0 0 10px rgba(0, 255, 255, 0.5);
        }

        .lives-display {
            font-size: 20px;
            color: #ff6b6b;
        }

        .speed-control {
            display: flex;
            align-items: center;
            gap: 10px;
        }

        .speed-control label {
            font-size: 16px;
        }
    </style>

```



```

    #speedSlider {
      width: 200px;
    }

    .instructions {
      margin-top: 15px;
      text-align: center;
      font-size: 14px;
      color: #aaa;
    }

    .game-over {
      font-size: 28px;
      color: #ff6b6b;
      font-weight: bold;
      margin-top: 10px;
    }

    button {
      background: #00ffff;
      border: none;
      padding: 10px 20px;
      font-size: 16px;
      font-family: 'Courier New', monospace;
      cursor: pointer;
      border-radius: 5px;
      margin-top: 10px;
    }

    button:hover {
      background: #00cccc;
    }
  </style>
</head>
<body>
  <canvas id="gameCanvas" width="800" height="500"></canvas>

  <div class="controls">
    <div class="score-display">Score: <span id="score">0</span></div>
    <div class="lives-display">Lives: <span id="lives">3</span></div>

    <div class="speed-control">
      <label for="speedSlider">Ball Speed:</label>
      <input type="range" id="speedSlider" min="3" max="10" value="5" step="0.5">
      <span id="speedValue">5</span>
    </div>

    <div class="instructions">
      Press <strong>Q</strong> to move paddle up | Press <strong>A</strong> to move paddle down
    </div>

    <div id="gameOverText" class="game-over" style="display: none;">GAME OVER</div>
    <button id="restartBtn" style="display: none;">Restart Game</button>
  </div>

  <script>
    const canvas = document.getElementById('gameCanvas');
    const ctx = canvas.getContext('2d');
    const scoreDisplay = document.getElementById('score');
    const livesDisplay = document.getElementById('lives');
    const speedSlider = document.getElementById('speedSlider');
    const speedValue = document.getElementById('speedValue');
    const gameOverText = document.getElementById('gameOverText');
    const restartBtn = document.getElementById('restartBtn');

    let score = 0;
    let lives = 3;
    let gameOver = false;
    let baseSpeed = 5;

    const paddle = {
      x: 20,

```

```

        y: canvas.height / 2 - 50,
        width: 15,
        height: 100,
        speed: 8
    };

    const ball = {
        x: canvas.width / 2,
        y: canvas.height / 2,
        radius: 8,
        dx: baseSpeed,
        dy: baseSpeed * 0.6,
        speed: baseSpeed
    };

    const keys = {};

    document.addEventListener('keydown', (e) => {
        const key = e.key.toLowerCase();
        if (key === 'q' || key === 'a') {
            keys[key] = true;
        }
    });

    document.addEventListener('keyup', (e) => {
        const key = e.key.toLowerCase();
        if (key === 'q' || key === 'a') {
            keys[key] = false;
        }
    });

    speedSlider.addEventListener('input', (e) => {
        baseSpeed = parseFloat(e.target.value);
        speedValue.textContent = baseSpeed;

        const currentDirection = {
            x: ball.dx / Math.abs(ball.dx),
            y: ball.dy / Math.abs(ball.dy)
        };

        ball.speed = baseSpeed;
        ball.dx = currentDirection.x * baseSpeed;
        ball.dy = currentDirection.y * baseSpeed * 0.6;
    });

    restartBtn.addEventListener('click', resetGame);

    function resetGame() {
        score = 0;
        lives = 3;
        gameOver = false;
        scoreDisplay.textContent = score;
        livesDisplay.textContent = lives;
        gameOverText.style.display = 'none';
        restartBtn.style.display = 'none';

        paddle.y = canvas.height / 2 - 50;
        resetBall();
        gameLoop();
    }

    function resetBall() {
        ball.x = canvas.width / 2;
        ball.y = canvas.height / 2;
        ball.dx = baseSpeed;
        ball.dy = (Math.random() - 0.5) * baseSpeed;
    }

    function drawPaddle() {
        ctx.fillStyle = '#00ffff';
        ctx.fillRect(paddle.x, paddle.y, paddle.width, paddle.height);
    }

```

```

function drawBall() {
  ctx.fillStyle = 'ffffff';
  ctx.beginPath();
  ctx.arc(ball.x, ball.y, ball.radius, 0, Math.PI * 2);
  ctx.fill();
}

function drawWall() {
  ctx.fillStyle = 'ff6b6b';
  ctx.fillRect(canvas.width - 15, 0, 15, canvas.height);
}

function updatePaddle() {
  if (keys['q'] && paddle.y > 0) {
    paddle.y -= paddle.speed;
  }
  if (keys['a'] && paddle.y < canvas.height - paddle.height) {
    paddle.y += paddle.speed;
  }
}

function updateBall() {
  ball.x += ball.dx;
  ball.y += ball.dy;

  // Top and bottom wall collision
  if (ball.y - ball.radius < 0 || ball.y + ball.radius > canvas.height) {
    ball.dy *= -1;
  }

  // Right wall collision (score point)
  if (ball.x + ball.radius > canvas.width - 15) {
    ball.dx *= -1;
    score++;
    scoreDisplay.textContent = score;
  }

  // Paddle collision
  if (ball.x - ball.radius < paddle.x + paddle.width &&
    ball.x + ball.radius > paddle.x &&
    ball.y > paddle.y &&
    ball.y < paddle.y + paddle.height) {
    ball.dx = Math.abs(ball.dx);

    const hitPos = (ball.y - paddle.y) / paddle.height;
    ball.dy = (hitPos - 0.5) * baseSpeed * 1.5;
  }

  // Ball goes off left side (lose life)
  if (ball.x - ball.radius < 0) {
    lives--;
    livesDisplay.textContent = lives;

    if (lives <= 0) {
      gameOver = true;
      gameOverText.style.display = 'block';
      restartBtn.style.display = 'block';
    } else {
      resetBall();
    }
  }
}

function draw() {
  ctx.fillStyle = '#0f3460';
  ctx.fillRect(0, 0, canvas.width, canvas.height);

  drawWall();
  drawPaddle();
  drawBall();
}

```

```
    }

    function gameLoop() {
        if (gameOver) return;

        updatePaddle();
        updateBall();
        draw();

        requestAnimationFrame(gameLoop);
    }

    gameLoop();
</script>
</body>
</html>
```