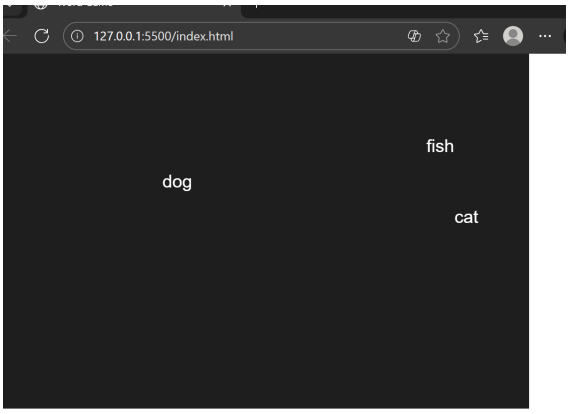
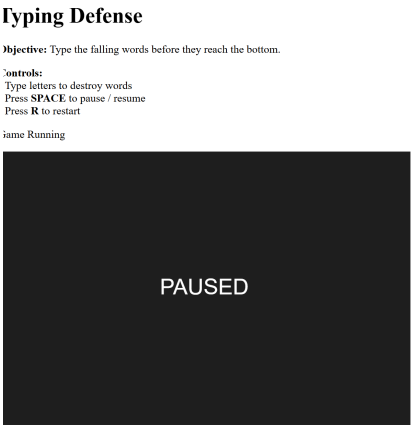
Process & Decision Documentation

Side Quest #4 Decisions

For this week's mini game, I wanted to try using GenAI as a coding tool and to see if it couldincorporate using arrays or JSONs using the basis of Example 3 of Week 4's example in VS-Code. The main idea of using words came from playing Codenames (dealing with words rather than having JSON fixed by tiles of numbers). I wanted to focus on trying out the accessibility of the game and its instructions produced by GenAI and how much input I would have implement. From what I discovered, it does well in keeping things clear and simple, but the more you add to it or incorporate overlapping logic, the more it may mess things up.





(words prompted by Chat include 'r' and the code will overrun it with the command of restart)

Instruction: Rejecting certain language like "destroy" in the instruction. In terms of clarity for users, language might get interpreted different so having straightforward instructions is an objective I want to follow. As for another logic, 'R' is commonly used to represent Restart, a problem I encountered with is that the keyPressed for 'r' will overrun my objective. So I had to change the words in the JSON list to avoid having 'r' in any of the words. I would assume it was something that GenAI would know based on its algorithm and built-in logic, but I guess that was tuned over.

GenAI Documentation
Date Used: February 9, 2026

Tool Disclosure: ChatGPT 5.2

Purpose of Use: Using GenAI to help aid in coding a mini game for sidequest 4

Summary of Interaction: I used ChatGPT to help

Human Decision Point(s): Throughout the entire process, I tested each code on Live Server to ensure that no errors were generated and fixed any problems that occurred with the output. I

Integrity & Verification Note: I ran the code in VS-Code and double-checked on Live Server if the output is the same as what the code states.

Scope of GenAI Use: GenAI is used primarily as a coding tool for this project, with human input.

Limitations or Misfires: Chat did not account for the constraints of the game's rules and the words in its JSON list. For example, press 'r' restart the game, but when it generated the JSON list, a couple of the words contained 'r'.

Summary of Process (Human + Tool):
- Input prompt for game design
- GenAI output code prompts + option
- I go in and run that code in VS-Code to test if it can work within the system

Verification & Judgement
- Each prompt I tested in VSCode to see if the code would run smoothly, and it did
- Chat also suggested with clear highlights of "optional" → rejected to keep code simple and to leave room for my changes

Appendix

Prompt 1:
I want to create a mini-game using arrays/json, maybe instead of having the traditional of storing numbers to use like tiles, what about including words? Something like those speed type games, but instead the words fall down like a typing defense. The difficulty would be after a “wave” the words get longer and fall faster.

Use seperate screens to help organize game using base coding examples(html, css, js, json).

html,
body { margin: 0; padding: 0; } canvas { display: block; } <!doctype html> <html lang="en"> <head> <meta charset="utf-8" /> <meta name="viewport" content="width=device-width, initial-scale=1" /> <title>Example 3 — JSON Levels + Level Class</title> <link rel="stylesheet" href="style.css" /> </head> <body> <script src="https://cdn.jsdelivr.net/npm/p5@1.9.0/lib/p5.min.js"></script> <script src="sketch.js"></script> </body> </html> { "include": ["*.js", "libraries/*.js", "c:\\Users\\david\\.vscode\\extensions\\wmcicompsci.cs30-p5-1.9.3\\p5types\\global.d.ts"] }

Respond 1:

🗂️ Folder / File Structure (VS Code Explorer)

Your project should look like this:

```
typing-defense-game/
│
├── index.html      ← HTML (loads everything)
├── style.css       ← CSS (page + canvas styling)
├── sketch.js       ← JS (game logic, p5.js)
├── words.json      ← JSON (word waves / levels)
└── jsconfig.json   ← (optional, for VS Code + p5)
```

Each file = one “screen” in VS Code.

🟧 index.html (STRUCTURE)

This file **only sets up the page** and loads scripts.
You already have this basically perfect.

```
<!doctype html>
<html lang="en">
```

```
  <head>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width,
initial-scale=1" />
    <title>Typing Defense Game</title>
    <link rel="stylesheet" href="style.css" />
  </head>

  <body>
    <!-- p5 library -->
    <script
src="https://cdn.jsdelivr.net/npm/p5@1.9.0/lib/p5.min.js"></script>

    <!-- your game code -->
    <script src="sketch.js"></script>
  </body>
</html>
```

✅ No game logic here
✅ Just loads CSS + JS

🟦 style.css (LOOK & LAYOUT)

This controls spacing and canvas behavior.
You already have the essentials — here's the clean version:

```
html, body {
  margin: 0;
  padding: 0;
}

canvas {
  display: block;
}
```

Optional upgrades later:

- center canvas

- background color
- custom font for typing games

🟨 words.json (DATA / LEVELS)

This is the **key part for Game Idea #7**.
Words fall based on JSON data.

```
{
  "waves": [
    ["cat", "dog", "fish"],
    ["apple", "banana", "orange"],
    ["elephant", "giraffe", "crocodile"]
  ]
}
```

Why this is good:

- Each array = one wave
- Easy to add difficulty
- No code changes needed to add levels

🟥 sketch.js (GAME LOGIC)

This is where:

- JSON is loaded
- Words are placed using loops
- Typing destroys falling words

Load JSON

```
let wordData;
let currentWave = 0;
```

```
let words = [];

function preload() {
  wordData = loadJSON("words.json");
}
```

Setup canvas

```
function setup() {
  createCanvas(600, 400);
  loadWave(currentWave);
}
```

Load a wave using loops

```
function loadWave(waveIndex) {
  words = [];
  let wave = wordData.waves[waveIndex];

  for (let i = 0; i < wave.length; i++) {
    words.push({
      text: wave[i],
      x: random(50, width - 50),
      y: -i * 40,
      speed: 1 + waveIndex
    });
  }
}
```

✅ This satisfies:

- JSON data
- Loops
- Dynamic placement
- Word obstacles

Draw + move words

```
function draw() {
  background(30);

  fill(255);
  textSize(20);

  for (let word of words) {
    word.y += word.speed;
    text(word.text, word.x, word.y);
  }
}
```

Typing interaction

```
function keyTyped() {
  for (let i = words.length - 1; i >= 0; i--) {
    if (words[i].text[0] === key) {
      words[i].text = words[i].text.substring(1);

      if (words[i].text.length === 0) {
        words.splice(i, 1);
      }
      break;
    }
  }

  // next wave
  if (words.length === 0) {
    currentWave++;
    if (currentWave < wordData.waves.length) {
      loadWave(currentWave);
    }
  }
}
```

🟪 jsconfig.json (OPTIONAL but helpful)

This helps VS Code understand p5 globals:

```
{
  "include": [
    "*.js",
    "libraries/*.js",

"c:\\Users\\david\\.vscode\\extensions\\wmcicompsci.cs30-p5-1.9.3\\p5types\\global.d.ts"
  ]
}
```

You already have this 👍

🧠 How this meets the assignment

✔ Uses JSON for level data
✔ Uses arrays + loops
✔ Dynamically places word obstacles
✔ Organized into separate screens/files
✔ Clean, readable, scalable

If you want, I can:

- Add **lives / game over**
- Add **score**
- Add **difficulty scaling**
- Help you write the **assignment explanation paragraph**

Just say the word 😄

Prompt 3:
I think that it is a good base for starting the game, but in terms of acessibility and objectives, it lacks. Include adding instruction within the html or off the canvas of the game, what about pauses with space? How would the game reset?

Respond 3:

1️⃣ Add instructions & objectives (outside the canvas)

Putting instructions off the canvas improves accessibility and avoids clutter.

Update `index.html`

Add a small UI section above or below the game:

```
<body>
  <div id="ui">
    <h1>Typing Defense</h1>

    <p>
      <strong>Objective:</strong> Type the falling words before they
reach the bottom.
    </p>

    <p>
      <strong>Controls:</strong><br>
      • Type letters to destroy words<br>
      • Press <strong>SPACE</strong> to pause / resume<br>
      • Press <strong>R</strong> to restart
    </p>

    <p id="status">Game Running</p>
  </div>

  <script
src="https://cdn.jsdelivr.net/npm/p5@1.9.0/lib/p5.min.js"></script>
  <script src="sketch.js"></script>
</body>
```

Optional CSS for clarity

```
#ui {
  font-family: sans-serif;
  padding: 10px;
  background: #111;
  color: white;
}
```

✅ Clear objective
✅ Controls visible
✅ Screen-reader friendly

2️⃣ Add pause / resume with SPACE

We'll use a boolean flag.

In `sketch.js`

```
let isPaused = false;
```

Update `draw()`

```
function draw() {
  background(30);

  if (isPaused) {
    textAlign(CENTER, CENTER);
    textSize(32);
    fill(255);
    text("PAUSED", width / 2, height / 2);
    return;
  }

  fill(255);
  textSize(20);

  for (let word of words) {
    word.y += word.speed;
    text(word.text, word.x, word.y);

    // lose condition
    if (word.y > height) {
      gameOver();
    }
  }
}
```

Handle spacebar

```
function keyPressed() {
  if (key === ' ') {
    isPaused = !isPaused;
    updateStatus(isPaused ? "Paused" : "Game Running");
  }
}
```

3️⃣ Prevent typing while paused

Modify `keyTyped()`:

```
function keyTyped() {
  if (isPaused) return;

  for (let i = words.length - 1; i >= 0; i--) {
    if (words[i].text[0] === key) {
      words[i].text = words[i].text.substring(1);

      if (words[i].text.length === 0) {
        words.splice(i, 1);
      }
      break;
    }
  }

  if (words.length === 0) {
    nextWave();
  }
}
```

4️⃣ Game over state + reset

Add flags

```
let isGameOver = false;
```

Game over function

```
function gameOver() {
```

```
  isGameOver = true;
  noLoop();
  updateStatus("Game Over – Press R to Restart");
}
```

Reset function

```
function resetGame() {
  currentWave = 0;
  isGameOver = false;
  isPaused = false;
  loadWave(currentWave);
  loop();
  updateStatus("Game Running");
}
```

Restart key

```
function keyPressed() {
  if (key === ' ') {
    isPaused = !isPaused;
    updateStatus(isPaused ? "Paused" : "Game Running");
  }

  if (key === 'r' || key === 'R') {
    resetGame();
  }
}
```

5️⃣ Status text updates (accessibility win)

```
function updateStatus(text) {
  document.getElementById("status").textContent = text;
}
```

This helps:

- **Screen readers**
- **Players who miss visual cues**
- **Clear game state feedback**