

Class 5 Browser Game Notes

Intro to Vibe Coding Take-home reference

Porting a game

A port adapts a game to another platform. We use the Python game as a reference and implement its behavior with JavaScript in a browser. AI helps translate the code. You decide which features matter and check the results.

Keep the original Python files. The browser version lives in a separate web subfolder. Changing a filename from .py to .js does not translate the program.

The browser version's files

- index.html creates the page, game canvas, and controls.
- styles.css controls the page's appearance and layout.
- game.js contains game rules, input, state, and the repeating loop.
- assets/ holds local images or sounds, if your game uses them.

The game loop and state still matter

The loop reads controls, updates state and collisions, and draws the next frame. requestAnimationFrame asks the browser to call the game again before a repaint. Elapsed time helps movement stay consistent at different frame rates.

State includes position, score, lives, time, and whether a round is playing, paused, won, or lost. A port should preserve what actions change these values.

Python example: `self.score += 1`

JavaScript example: `this.score += 1;`

Both examples add one to this game object's score. Your generated code may use a different name, such as state.score. Find the real line in your project.

A direct edit and a useful test

Find a named speed, target, or timer setting near the top of game.js. Predict the effect, change one value, save, and refresh the browser. Restart resets round state. A browser refresh loads the page again and resets this version's round.

Compare controls, visual feedback, scoring, endings, pause, and restart with the original. A new bug after a change is a regression. Include browser checks such as resizing and switching tabs.

Reopening your browser game

Open the project in VS Code. Ask your AI tool to start the recorded local server, or use the matching command below from the project folder containing web.

macOS / Linux:

```
python3 -m http.server 8000 --bind 127.0.0.1 --directory web
```

Windows:

```
py -3 -m http.server 8000 --bind 127.0.0.1 --directory web
```

Open `http://localhost:8000` in your browser. If your agent used another port, use its address. Keep the server terminal open. Ctrl+C in that terminal stops it. Python is delivering the files here; JavaScript runs the game in the browser.

Local and public addresses

`file://` opens a file on your computer. `http://localhost:8000` reaches the local server on the computer using that address. Another person's localhost points to their own computer.

A public hosting service supplies your HTML, CSS, JavaScript, and assets at a public web address. Visitors' browsers run our single-player game. Your laptop can be off while the host serves it.

GitHub can keep the project's source files. GitHub Pages can host a static web game. Multiplayer or a shared online leaderboard needs additional services. A personal best saved in one browser is local to that browser's storage.

When something does not work

- Missing game: check that the server serves web, that `index.html` exists, and that the URL uses the right port.
- Missing art: check relative asset paths and exact filename capitalization.
- Old code: save the correct file and refresh; try a hard refresh if needed.
- Broken behavior: report the action, expected result, actual result, and any browser console error. Request one focused fix and retest it.

Stretch and next steps

Add touch controls, improve the layout at small sizes, or save a personal best. Test the new feature and the full round. Keep the working game ready for our upcoming hosting lesson.

Standing practice: check ChatGPT or Claude for application updates each week. Keep your project folder and launch command available.

References: [MDN game anatomy](#); [MDN local testing servers](#); [GitHub Pages documentation](#).