Side Quests and A4 (Individual Work)

GenAI Documentation

If GenAI was used (keep each response as brief as possible):

Date Used: Feb 22nd, 2026

Tool Disclosure: ChatGPT 5.2 (OpenAI)

Purpose of Use:
To assist in developing and debugging code features in an existing game file.

Summary of Interaction:
I asked ChatGPT for guidance on how to add Geometry Dash-inspired features to my existing side-scroller code, including auto-panning, increasing panning speed over time, a lose condition if the player falls behind the screen edge, and adding more platforms.

Human Decision Point(s):

After receiving the guidance, I decided which suggestions to implement, how to tune the panning values (speed, acceleration, and loss margin), and how to design the platform layout and difficulty progression.

Integrity & Verification Note:
I used GenAI for guidance and debugging support, not as a final solution to submit unchanged. I tested the code in my own project, adjusted the logic and values manually, and verified that the final implementation worked with my existing files.

Scope of GenAI Use:
Used for brainstorming, implementation guidance, debugging support, and general code suggestions. Final integration, testing, tuning, and level/platform design were completed by me.

Appendix

Prompt:

I have an idea I want to add to this code. I want to add a few features without changing the existing code.

Please provide me a guide on how to implement the following:

1. I want the screen to slowly automatically pan right as soon as the user presses the button to move. The more time goes by, the faster the panning gets. If the user controlling the blob is too slow and touches the border of the screen, then they lose. This is inspired by Geometry Dash.

2. Expand the world and add more platforms

ChatGPT-5.2:

Great — here are the exact updates with no refactor and only the requested changes (with a little code where helpful, but still guide-style).

1) `sketch.js` — add new auto-pan state variables near your existing globals (`level`, `player`, `cam`)
Add a small block to track the auto-scroll state and speed over time.

```
// --- auto-pan / pressure camera (Geometry Dash style) ---
let autoPanStarted = false;
let autoPanX = 0;

// time-based speed (px/second)
let autoPanSpeed = 0;
let autoPanBaseSpeed = 70;      // tune this
let autoPanMaxSpeed = 320;      // tune this
let autoPanAccelPerSec = 22;    // tune this
let autoPanSpeedLerp = 0.08;    // smoothness (0..1)

let autoPanElapsedSec = 0;
let borderLoseMargin = 6;       // lose if blob touches left edge area
```

Why: this keeps all new behavior isolated in `sketch.js` and does not touch your existing classes.

---

2) `sketch.js` — reset the auto-pan state inside `loadLevel(i)` after camera setup/clamp
When the level reloads, reset the new state so the next run starts fairly.

Add this after your camera setup/clamp inside `loadLevel(i)`:

```
autoPanStarted = false;
autoPanX = cam.x;

autoPanSpeed = 0;
autoPanElapsedSec = 0;
```

Why: without this, respawns may inherit high speed / bad camera state.

---

3) `sketch.js` — add a helper function to detect when the player starts moving

Add a small helper so the panning begins only when the player actually starts the run.

```
function playerIsTryingToMove() {
 return (
   keyIsDown(65) || keyIsDown(LEFT_ARROW) ||   // A / ←
   keyIsDown(68) || keyIsDown(RIGHT_ARROW)     // D / →
 );
}
```

Why: this avoids changing `BlobPlayer` and keeps the trigger simple.

---

4) `sketch.js` — add an `updateAutoPan()` function (time-based acceleration, not progress-based)

Create a function that runs every frame and increases speed based on elapsed time.

Use this structure (you can tune values later):

```
function updateAutoPan() {
 // start the first time the player presses move
 if (!autoPanStarted && playerIsTryingToMove()) {
   autoPanStarted = true;
   autoPanSpeed = autoPanBaseSpeed;
   autoPanElapsedSec = 0;
 }

 if (!autoPanStarted) return;

 // deltaTime is ms/frame in p5 -> convert to seconds
 const dt = min(deltaTime / 1000, 0.05); // clamp lag spikes
 autoPanElapsedSec += dt;

 // speed increases with TIME, not player position
 const targetSpeed = min(
   autoPanMaxSpeed,
   autoPanBaseSpeed + autoPanAccelPerSec * autoPanElapsedSec
 );

 // smooth ramp (optional but feels better)
 autoPanSpeed = lerp(autoPanSpeed, targetSpeed, autoPanSpeedLerp);
 // move auto-pan in world space
 autoPanX += autoPanSpeed * dt;
}
```

Important: this is the key change that gives you “the more time goes by, the faster it pans.”

---

5) `sketch.js` — update the camera section in `draw()` to combine follow camera + auto-pan pressure

Keep your existing camera follow, then add auto-pan as a minimum camera x-position.

Replace only the camera update block in `draw()` with this flow:

```
updateAutoPan();

// existing smooth follow camera
cam.followSideScrollerX(player.x, level.camLerp);

// auto-pan pressure: camera cannot lag behind autoPanX
if (autoPanStarted) {
 cam.x = max(cam.x, autoPanX);
}

cam.y = 0;
cam.clampToWorld(level.w, level.h);

// keep autoPan synced after clamp so it doesn't lag/snap
if (autoPanStarted) {
 autoPanX = cam.x;
}
```

Why this order matters:

1. update pan target
2. follow player
3. push camera forward
4. clamp

5. sync

That prevents weird “stops then speeds up suddenly” behavior.

---

6) `sketch.js` — add a lose condition if the blob touches the left border of the screen

After camera updates (and before/after draw is fine), compute the player’s position on screen and reset if they fall behind.

Add a check like this in `draw()`:

```
const playerScreenX = player.x - cam.x;

if (autoPanStarted && playerScreenX - player.r <= borderLoseMargin) {
 loadLevel(levelIndex); // respawn current level
 return;
}
```

Why: this creates the Geometry Dash-inspired pressure mechanic.

---

7) `levels.json` — increase the world width so the camera has room to pan

Update `world.w` to be much larger than the original example.

Example idea (not mandatory exact value):

```
"world": { "w": 6000, "h": 360, "deathY": 560 }
```

Why: the camera is clamped to the world, so if the world is short, the panning will stop early.

---

8) `levels.json` — add more platforms and make them harder over time

Append more platform objects to your existing `"platforms"` array using the same format:

```
{ "x": ..., "y": ..., "w": ..., "h": ... }
```

Difficulty progression guide:

- **Early added section**: wider platforms, smaller gaps (learn timing)

- **Middle section**: more holes, staggered heights, shorter recovery platforms
- **Late section**: narrow landings, longer gaps, back-to-back jumps

You do **not** need to change `Platform.js` or collision logic for this — the existing engine will handle the new layout.

---

9) Keep the original classes unchanged (as requested)
Do not refactor or rewrite:

- `BlobPlayer`
- `Camera2D`
- `LevelLoader`
- `WorldLevel`

All requested features can be added through:

- `sketch.js` (auto-pan + lose condition)
- `levels.json` (world size + more platforms)