Fortnite Drop Calculator

Fortnite Drop Calculator is a small web app I built to practise turning a simple idea into a working interactive tool. You set a **Battle Bus route** and a **target landing point**, and the app shows where to **jump (J)** and where your **glider deploys (G)** using a simplified drop-time model.

What the app does

- You drag **S (Start)** and **E (End)** to match the bus path.
- You click (or drag) **T (Target)** to choose where you want to land.
- The app automatically draws:
    - **J (Jump point)**
    - **G (Deploy point)**
    - The path **J → G → T**
- You can toggle between a blank map and a labelled map (named locations).

Links

- **Live demo:** https://violevo.github.io/projects/fortnite-drop-calculator/
- **GitHub repository:** https://github.com/Violevo/Fortnite-Drop-Calculator

How it works

1. Inputs and outputs

There are three user-controlled markers:

- **S** = bus start (draggable)
- **E** = bus end (draggable)
- **T** = target (click to place, draggable after)

And two computed markers:

- **J** = optimal jump point on the bus segment **S → E**
- **G** = glider deploy transition point (end of freefall / start of glide)

On every update to **S**, **E**, or **T**, the app:

- updates the bus line (S → E)
- recomputes J and G
- redraws the freefall line (J → G) and glide line (G → T)

2. The calculation

The core function samples possible jump points along the bus route to find the optimal jump point **J**:

- The segment from **S to E** is divided into many candidates (using **N = 400** samples).
- For each candidate jump point **J**, the app estimates how long it would take to reach **T** by splitting the journey into:
    1. **time on the bus** (from S to J)
    2. **time falling** (from J to G)
    3. **time gliding** (from G to T)

It uses fixed values for bus, fall, and glide speeds across the map (these could likely be improved).
For each candidate **J** it computes a total time, then picks the point **J** with the smallest total.

Once the best **J** is found, the app calculates **G** (the point where falling ends and gliding begins), and shows both markers on the map.

> The model uses global constants (bus / fall / glide behaviour) across the entire map. These constants may not be perfectly accurate so could mean the model doesn't match in-game movement perfectly.

3. Map rendering

I used **Leaflet.js** with a simple coordinate system. The Fortnite map is loaded as an **image overlay**, so every point is just a pixel coordinate on a 2D plane. This makes the maths and geometry much easier, provided the map images stay consistent between updates.

The map image is fetched from **Fortnite-API.com**. If the request fails, the app falls back to default map image URLs so the tool still works.

Dragging markers triggers lots of updates. To avoid lag, I throttle recalculation using `requestAnimationFrame`, so updates stay smooth while still feeling “real-time”.

Limitations

This is a simplified model and won’t perfectly match in-game movement. If I extended it, I would improve accuracy by:

- Testing and validating constants myself
- Adding map terrain and structure height constraints to the model