Assignment 2: One Block to Move

Agent briefing: one command, one rendezvous point. Get your satellite there and hold position.

Your mission

Headquarters has marked a rendezvous point in the arena. Your job: fly your satellite there using **one block**, and park.

Build a program whose main page contains a single `setPos _, _, _` block with your target's three numbers (X, Y, Z) plugged into it. When you simulate, you should see your satellite glide to your target point and then sit perfectly still there for the rest of the match. One block. A whole flight plan.

Then comes the field experiment: your teacher will have you stack a **second** `setPos _, _, _` block with a different target below the first. **Before** you simulate it, write down your prediction: where will the satellite go? A real agent commits to a prediction before testing.

> ⏱ **Every second:** your satellite re-reads the whole main page and runs every block on it, top to bottom, then does it all again the next second. Your `setPos _, _, _` block isn't a push. It announces a destination, once a second, all match long.

Mission rules

- Create the project with game **Free Mode** and the **Graphical Editor** (same as Assignment 1), or keep building in your Assignment 1 project.
- Main page only: leave the init page empty.
- Use exactly **one** `setPos _, _, _` block for the mission. (The two-block version is the experiment, not the mission.)
- Your target must be inside the arena. Keep X and Y between about −0.6 and +0.6, and keep Z at 0 (the game is flat).

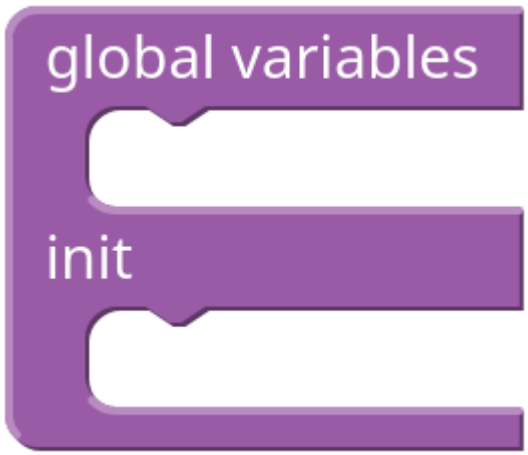
Blocks to explore

- **SPHERES Controls**: the block that gives your satellite a destination lives here.
- **Math**: number blocks live here; one goes in each socket of `setPos _, _, _`, and you can type decimals and negatives into them, like 0.4 or −0.4.
- **Debug** (optional): a `DEBUG [text]` block from here can print a message to the log, if you want proof that your page really runs every second.

Build it, step by step

1. **Start on the main page.** Open your project and make sure you're on the **main** page. Its root block is the one labeled "loop"; everything you build today snaps inside it:

The **init** page keeps its own root block, shown below, and you leave both of its slots empty for this whole assignment:



1. **Get the movement block.** Open the **SPHERES Controls** category and find this block:

Drag it onto the workspace and snap it inside the loop root; you'll feel it click into place. Fresh from the palette it reads `setPos 0, 0, 0`: three number sockets, one for X, one for Y, one for Z.

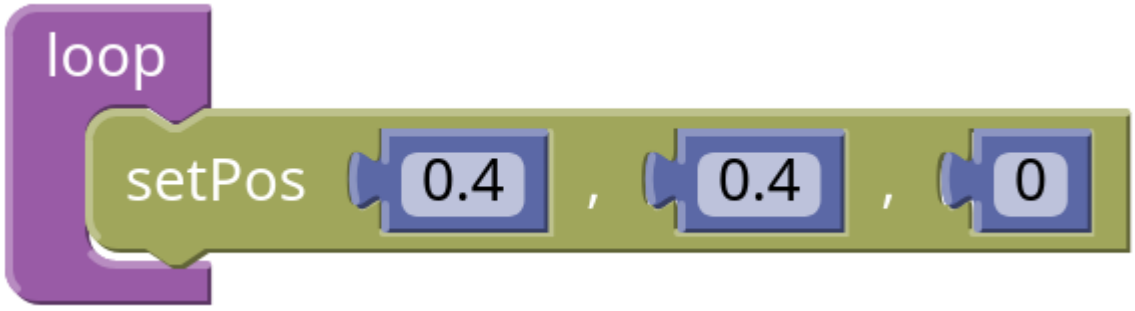
1. **Meet the number block.** Each of those three sockets holds a number block from the **Math** category:

If a socket is ever empty (say, you pulled a number out by accident), drag a fresh number block from **Math** into it. To change a number, click it and type right over it. Decimals and negatives both work.

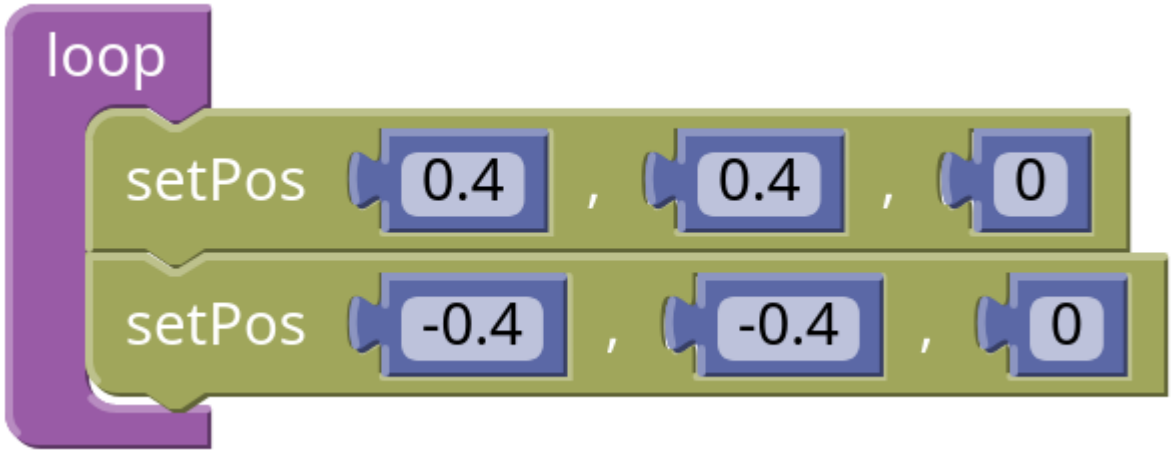
1. **Type in your target.** Pick a rendezvous point inside the arena: X and Y each between about −0.6 and +0.6, and Z exactly 0. Click the first number and type your X, the second for your Y, and leave the third at 0. The demo target is (0.4, 0.4, 0), which looks like this:

Your own corner is fine; just keep it in bounds and write it down.

1. **Compare your workspace.** That's the whole mission build: one `setPos _, _, _` block sitting inside the loop root. Your main page should match this:



1. **Simulate the mission.** Run a simulation and watch the viewer: your satellite should glide to your target and park there, perfectly still, until the match ends. One block, announced once a second, is a whole flight plan.
2. **The field experiment (when your teacher says go).** Drag a **second** `setPos _, _, _` block from SPHERES Controls and snap it directly **below** your first one, inside the same loop root. Give it a different target; the demo uses (−0.4, −0.4, 0):



Stop. Before you press simulate, write down your prediction: where will the satellite go? Now simulate, and compare what happened with what you predicted.

Hint

A `setPos` block is not a push; it is a destination. Telling the satellite the same destination every second is perfectly fine. So what happens if two different destinations are announced every second, back to back?

Check your work

- In the simulation viewer, your satellite glides to your target point and **parks** there (no wandering, no bouncing) until the match ends.
- The viewer's state panel shows your position. Watch pX and pY settle at your target numbers and stay there.
- For the experiment: you wrote your prediction down before simulating, and you can now explain what actually happened using the words "every second."
- Bonus check: click the "C Code" toggle. You should see your `setPos(...)` line inside `void loop()`, the part of the program the satellite runs once per second.

Reflection (write 3–5 sentences)

Explain your block program in your own words, as if to a teammate who missed class: **what does your program do every second**, and why is that enough to fly somewhere and stop? Then answer: in the two-block experiment, why did the satellite fly only to the second target? What happens inside one single second that makes the last destination win?