Assignment 2: One Block to Move (Teacher Guide)

At a glance

Learning objective	Students fly the satellite to a point with a single `setPos _, _, _` block, and can explain why one block is a complete flight plan: the main page re-runs every second, and `setPos _, _, _` sets a **destination**, not a push.
Time estimate	45–60 minutes
Project setup	Game: **Free Mode** · Editor: **Graphical Editor**
Prerequisite assignment	Assignment 1
Blocks introduced	`setPos _, _, _` (SPHERES Controls), number blocks (Math)

The one idea this assignment teaches

In Assignment 1 the class learned that the main page is a checklist the satellite re-reads every second, not a story that plays out over time. Today they feel what that means for movement. The `setPos _, _, _` block does not give the satellite a shove. It tells the satellite, "your destination is this point," and the satellite's built-in controller flies there and stops. So a page with one `setPos _, _, _` block on it tells the satellite the same destination every single second, 180 times in a three-minute match. That sounds wasteful, but it is exactly right: repeating a destination changes nothing. One block really is a whole flight plan.

> ⏱ **Every second:** the satellite reads the whole main page from top to bottom and does every block on it, then does the whole page again the next second. A `setPos _, _, _` block on that page announces the destination once per second, forever. Announcing the same destination again is harmless. Announcing a different destination one millisecond later cancels the first.

That second sentence is the killer demo of this lesson. Students who still secretly believe "blocks run in order over time" will stack two `setPos _, _, _` blocks and expect a two-stop tour. Instead, both blocks run within the same second, back to back; the second destination overwrites the first; the satellite only ever has one destination (the last one set), so it flies only to the second target. Let them predict, let them be wrong, and then give them the explanation. This one surprising result does more to install the every-second model than any speech.

Before class

- [ ] Log in to the Zero Robotics IDE and create a **Free Mode** project with the **Graphical Editor** (or plan to reuse Assignment 1's project and clear the main page). In Free Mode the AsteroidBee game categories will not appear in the palette. That is expected; today you only need **SPHERES Controls** and **Math**.
- [ ] Practice the demo once: one `setPos _, _, _` block with 0.4, 0.4, 0, then the stacked two-block experiment. Make sure you can run a simulation and see the viewer.
- [ ] Know the arena numbers: positions are (X, Y, Z) in meters. X runs roughly −0.6 to +0.6 and Y roughly −0.8 to +0.8; the game is 2D, so Z stays 0 today. "Keep both numbers inside ±0.6" is a safe rule of thumb for the class.
- [ ] Have something for students to write predictions on (the moment in step 6 below where they commit a prediction in writing is the heart of the lesson).
- [ ] Print one copy of `printable.md` per student.

Walkthrough: what to say and do

1. **Set up the project.** Create a new Free Mode graphical project (or open Assignment 1's project and drag yesterday's blocks off the main page into the trash). Remind the class: two pages. **init** runs once at the start, **main** re-runs every second.
2. **Introduce the arena.** Say: "The satellite lives in a box. Every spot in the box has an address: three numbers, (X, Y, Z), measured in meters. The box is small (stay inside about −0.6 to +0.6), and today Z is always 0 because the game is flat." Draw a quick square on the board, mark the center as (0, 0), and mark (0.4, 0.4) in one corner area.
3. **Build the one-block flight plan.** Open the **SPHERES Controls** palette category and drag `setPos _, _, _` into the main page's "loop" slot. Open **Math** and drag a number block into each of the three sockets. Type **0.4**, **0.4**, **0**. That's the whole program.
4. **Ask for a prediction.** Say: "This page re-runs every second, sixty times a minute. Will the satellite fly to the point sixty times? Fly there, leave, come back?" Take a few answers out loud. Don't resolve it yet.
5. **Simulate and watch the viewer** (it looks like ../images/ide-simulation-viewer.png). The satellite glides to (0.4, 0.4) **once** and parks there for the rest of the match. Explain: "`setPos _, _, _` doesn't push the satellite; it sets the destination. Every second the block repeats the same destination, and repeating a destination changes nothing. The satellite's built-in pilot flies there and holds position. One block is a complete flight plan."
6. **THE EXPERIMENT.** Drag a second `setPos _, _, _` block and snap it directly **below** the first, with number blocks **−0.4**, **−0.4**, **0**. Before anyone touches the simulate button, have every student **write down** a prediction: "Where will the satellite go?" Most will write that it visits both corners. Simulate. It flies **only** to (−0.4, −0.4); the first target is never visited. Now explain: "Each second, both blocks run, within a millisecond of each other. The second block overwrites the first. The satellite only ever has one destination: the last one it heard. So the first block isn't skipped; it just gets out-voted every single second."
7. **Connect it back to Assignment 1.** Say: "The page is a checklist re-read every second, not a story. If you want the satellite to visit two places, you can't just stack two destination blocks; you'd need the program to remember which stop it's on. That's exactly where this course is headed next."
8. **Toggle C Code.** Click the IDE's "C Code" toggle and show the class: two `setPos(...)` lines, one after the other, inside `void loop()`. Same program, two views. "The computer runs everything inside `loop()` every second. You can see both lines are there, and you saw which one wins."
9. **Set them loose on the mission.** Each student builds the one-block version with a target corner of their choice (inside the arena), simulates, and confirms the satellite parks there. The two-block version is the experiment, not the mission; the deliverable is one `setPos _, _, _` block plus a correct written prediction and explanation for the experiment.

The mistakes you will see

1. "It skipped my first block!": stacking two `setPos _, _, _` blocks and expecting a two-stop tour

- **What it looks like:** Two `setPos _, _, _` blocks in a stack; the satellite flies straight to the second target and the student insists the first block never ran.
- **Why it happens:** Blocks-as-a-timeline thinking: the belief that the program does block 1, finishes it, then does block 2. In reality the whole page runs top to bottom within each second, so both destinations are announced back to back every second, and the last one always wins. (This is the "I expected my blocks to run in sequence over time" misconception, made visible on purpose.)
- **The question to ask:** "Within ONE second, which destination did your satellite hear last?"

2. Coordinates outside the arena

- **What it looks like:** A target like 4 instead of 0.4. The satellite slams toward the wall and the game keeps pushing it back inside; it jitters near the edge and never settles.
- **Why it happens:** A typo, or not yet feeling that the whole box is barely more than a meter across.
- **The question to ask:** "Is your target inside ±0.6-ish meters?"

3. Hunting for a "stop" or "repeat" block

- **What it looks like:** A student digging through the palette for a block to make the satellite stop when it arrives, or to keep the movement "going."
- **Why it happens:** They still picture `setPos _, _, _` as a push that needs managing. There is nothing to stop: arriving and holding still is what a destination controller does.
- **The question to ask:** "What would the block need to say each second for the satellite to stay put? The same thing it already says."

Reference solution at a glance

The init page is empty: nothing to remember, nothing to set up. The main page has exactly one block in its "loop" slot: `setPos _, _, _` with the number blocks 0.4, 0.4, 0 in its three sockets. That single block, re-run every second, flies the satellite to (0.4, 0.4) and parks it there for the rest of the match.

The experiment version (demonstration only, not the mission deliverable) adds a second `setPos _, _, _` with −0.4, −0.4, 0 snapped directly below the first; it flies only to (−0.4, −0.4). See **reference-solution.md** for the exact block layout and the C code both versions generate.

Wrap-up questions

1. The same `setPos _, _, _` block ran about 180 times during the match. Why didn't that cause 180 separate flights?
2. In the experiment, the satellite never visited the first target. Walk me through one single second of the match: what did each block say, and why did the second one win?
3. Suppose you really do want the satellite to visit (0.4, 0.4) first and then fly to (−0.4, −0.4). What would the program need to be able to do that it can't do yet? (Listen for some version of "remember something between seconds"; that is Assignments 3 and 4.)