Assignment 4: Two Stops, One Brain

Two dead drops. One satellite. It has to remember which one it's working on.

Your mission

Agent, back in Assignment 2 you proved something painful: stacking two `setPos _, _, _` blocks does NOT make two stops. The satellite only obeyed the last one. Today you fix that, with **memory**.

Fly a two-checkpoint courier route:

1. First, fly to **Checkpoint A at (0, −0.4)**.
2. Once you have truly **arrived** at A, continue to **Checkpoint B at (0.4, −0.4)** and park there.
3. Announce each checkpoint in the log, and each announcement should appear **exactly once**.

Success looks like: the viewer shows your satellite stop at A, then move on to B and stay. The log shows "Checkpoint A!" once, then later "Route complete!" once.

> ⏱ **Every second:** your whole main page runs again, top to bottom. Blocks do not take turns over time on their own. Something in your program has to remember which stop you're working on, and decide, fresh each second, which blocks get to act.

Mission rules

- Project on the **AsteroidBee 2026** game (Graphical Editor).
- Exactly two stops: **A = (0, −0.4)**, then **B = (0.4, −0.4)**.
- "Arrived" means within **0.05 m on each axis** of the stop.
- A `DEBUG [text]` announcement at each stop.
- Your step-tracking variable must be **declared on the init page**.

Blocks to explore

- **Variables**, new territory: a variable remembers a number from one second to the next. Declaring it happens on the **init** page; reading and changing it can happen anywhere.
- **SPHERES Controls**: the same position-reading and flying blocks you used in Assignment 3.
- **Logic**: `if / then` asks a question every single second. You will want more than one question. The compare block's dropdown offers `==`, `!=`, `<`, `<=`, `>`, `>=`.
- **Math**: `absolute value` turns "how far off target am I?" into a number you can compare, just like last time.
- **Debug**: for your checkpoint announcements.

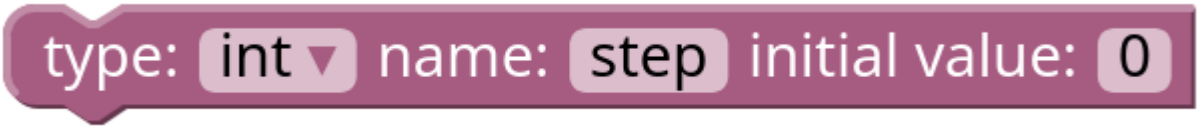
Build it, step by step

Work through these in order. Each picture shows the block you need; the text says exactly where it goes and what to type.

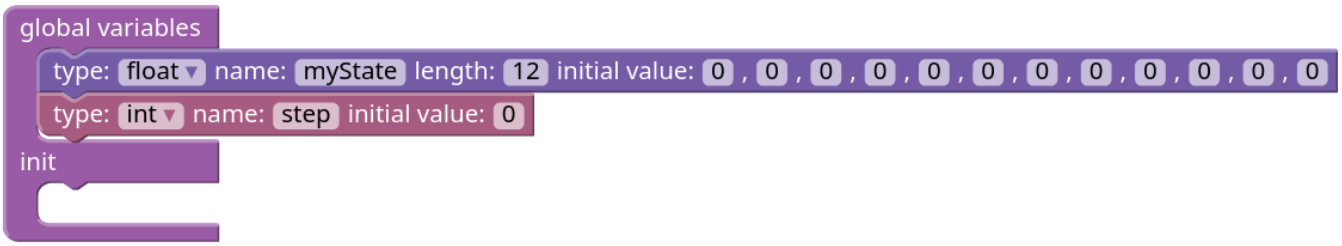
Part 1: give the satellite a memory (init page)

1. Declare the state array. Open the **init** page. In the **global variables** slot, place the same declare-array block you used in Assignment 3: type `float`, name `myState`, length `12`. (Declare blocks only appear in the palette while you are on the init page.)

2. Declare the step variable. Still on the init page, snap a plain declare block underneath it and set its dropdowns and fields to: type `int`, name `step`, initial value `0`. This is the satellite's notebook entry: which stop am I working on?



3. Check the init page. It should now hold exactly these two declares:



Part 2: read where you are (main page)

4. Get your state. Switch to the **main** page. From **SPHERES Controls**, drag `get My ZRState` to the very top of the loop and pick `myState` from its array dropdown. Every second this refills the array: `myState[0]` is your x, `myState[1]` is your y.

Part 3: the step-0 branch, fly to Checkpoint A

5. Place an `if / then` block. From **Logic**, snap an `if / then` block directly under `get My ZRState`. This block will own everything the satellite does while it is working on Checkpoint A.

6. Ask "am I on step 0?". From **Logic**, plug a compare block into the `if` socket. Leave its dropdown on `==`.

Into its left side, plug the variable block for `step` from **Variables**. It shows just the name:

Into its right side, plug a number block from **Math** and type `0`:

The question now reads `step == 0`.

7. Fly toward A. Inside the `then` slot, place a `setPos` block and type the coordinates of Checkpoint A into its three sockets: `0`, `-0.4`, `0`.

8. Add the arrival question. Under `setPos`, still inside the same `then`, snap a **second** `if / then` block. Its question is have I truly arrived at A?, which needs two tests joined together, so plug an `and` block into its `if` socket (leave the dropdown on `and`):

9. Build the x-axis test (left side of the `and`). Plug in a compare block and change its dropdown to `<`. On its left goes an `absolute value` block from **Math** (it is the `square root ▾` block with its dropdown changed to `absolute value`):

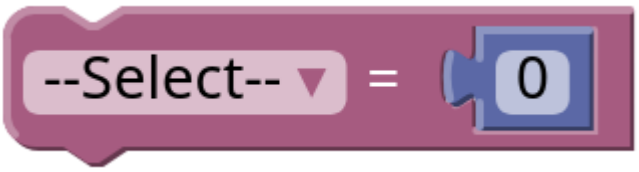
Inside the absolute value, put the array-slot block for `myState` with slot `0`:

On the compare's right, put a number block with `0.05`. The test now reads absolute value of *`myState[0] < 0.05`*, within 5 cm of x = 0.

10. Build the y-axis test (right side of the `and`). Same shape, one extra piece: Checkpoint A's y is −0.4, so the distance from it is `myState[1] + 0.4`. Build that with an arithmetic block from **Math**, dropdown set to `+`:

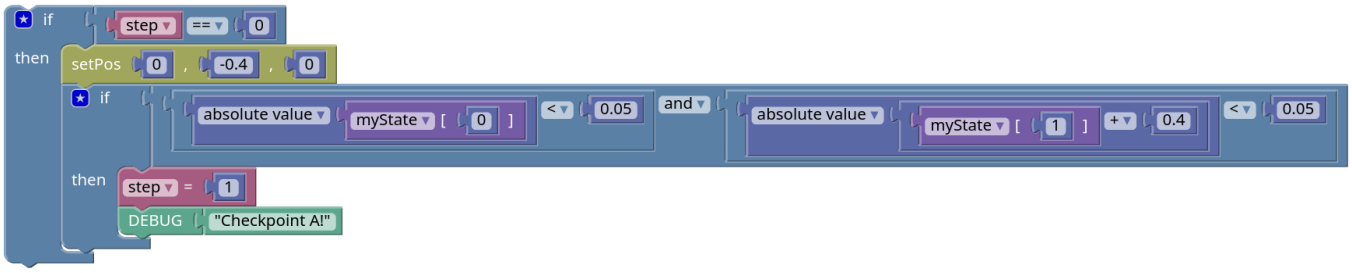
Put the `myState` slot-`1` block on its left and a number block with `0.4` on its right. Drop the whole sum inside an `absolute value` block, and compare it `<` `0.05`.

11. Advance the step and announce it. Inside the arrival `then`, do two things, in this order. First, change the variable. The set block shows the name, an `=`, and a value socket; make it read `step = 1`:



Then announce the checkpoint with a `DEBUG` block from **Debug**: type `Checkpoint A!` into the text block in its socket:

12. Compare your branch. Your whole step-0 branch should now match this:



Part 4: the step-1 branch, fly to Checkpoint B

13. Build the second branch the same way. Snap a second `if / then` block **underneath** the first one (not inside it!), and fill it in:

- Question: `step == 1`.
- Inside its `then`: a `setPos` block with `0.4`, `-0.4`, `0` (Checkpoint B).
- Its arrival question: absolute value of (*`myState[0]`* − *`0.4`*) `<` *`0.05`* `and` absolute value of (*`myState[1]`* + *`0.4`*) `<` *`0.05`*. The x-test needs an arithmetic block this time too: dropdown `−`, `myState[0]` minus `0.4`.
- Inside the arrival `then`: a set block reading `step = 2`, then a `DEBUG` block with the text `Route complete!`.

Setting `step = 2` parks the route: no branch answers to step 2, so the satellite holds at B for the rest of the match.

14. Compare your finished main page. It should match this:

Hint

The satellite needs a notebook entry that says which stop it is working on. A variable keeps its value from one second to the next, and an `if / then` can change it the moment a stop is reached. Then the NEXT second, a different `if / then` wins.

Check your work

Run the simulation (about 120 seconds is plenty) and watch:

- **Viewer:** the satellite flies to (0, −0.4), pauses there, then flies on to (0.4, −0.4) and parks for the rest of the match.
- **Log:** "Checkpoint A!" appears **once**. Later, "Route complete!" appears **once**.

Trouble signs:

- A message prints **every second** instead of once → what should change at the same moment the message prints?
- The satellite skips A and goes straight toward B (or never leaves where it started) → is anything remembering your progress? Is anything erasing it every second?
- "Route complete!" fires while you're still at A → which stop owns that arrival test?

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?** Then answer this: in Assignment 3 your arrival message printed every second, but today "Checkpoint A!" printed only **once**, even though the page that prints it still runs every second. Why?