Assignment 3 Teacher Guide: Am I There Yet?

At a glance

Learning objective	Students make the satellite SENSE each second: read its own position into an array with `get My ZRState`, and use an `if … then …` block to detect arrival ("am I within 5 cm of the target?") and announce it with `DEBUG [text]`. First contact with variables/arrays, comparisons, and the idea that a question is re-asked every second.
Time estimate	60–90 minutes
Project setup	New project on the game **AsteroidBee 2026**, using the **Graphical Editor**
Prerequisite assignment	Assignments 1–2
Blocks introduced	array declare `type: ▾ name: __ length: __ …` (Variables, init page only), `get My ZRState ▾ [array]` (SPHERES Controls), array slot get `<name> [ index ]`, `if … then …` (Logic), compare `[a] == ▾ [b]` (Logic), `[a] and ▾ [b]` (Logic), `absolute value` (a dropdown choice on the Math `square root ▾ [a]` block), number blocks

The one idea this assignment teaches

Until now the students' programs only acted: they told the satellite where to go and trusted it to get there. Today the program senses. The block `get My ZRState ▾ [array]` lets the satellite read its own position into a 12-slot list (an "array": think of it as a notebook with 12 numbered lines). Slot 0 is the X position, slot 1 is the Y position. Once those numbers are in the notebook, the program can ask a question about them with an `if … then …` block: "Am I within 5 cm of the target?"

The deep idea is when that question gets asked. It is not asked once. The main page re-runs every second, so the question is re-asked every second, fresh, from scratch, with freshly sensed numbers. Early in the flight the answer is "no" and the `if` does nothing. The moment the satellite settles near the target, the answer flips to "yes", and it stays yes, every second, for the rest of the match. That is why the "Arrived!" message prints over and over instead of once. Students will think that's a bug. It is actually the whole lesson.

> ⏱ **Every second:** the satellite re-reads the entire main page from the top. It senses its position again (`get My ZRState`), re-sets its destination again (`setPos _, _, _`), and asks the `if` question again. The program never "waits at" the `if` for it to become true and never "moves past" it once it is: there is no timeline in the blocks, only a checklist re-read every second.

One more new idea hides in the setup: the array must be declared on the **init page**, in the "global variables" slot. The init page runs once at the start of the match; the declare block creates the notebook there so it exists for the whole match and keeps its values between seconds. Declare blocks only appear in the palette while you are on the init page. Students hunting for them on the main page will not find them (this is the most common stuck point of the day).

Before class

- [ ] Confirm you can create a project on the game **AsteroidBee 2026** (it may be listed under archived/featured games: see the README setup checklist you completed before the unit). Do it once yourself, end to end.
- [ ] Build the reference solution yourself and simulate it (~90 seconds is enough). Confirm you see nothing in the log during the flight, then "Arrived!" once per second afterwards. The full layout is in reference-solution.md.
- [ ] Have the arena map ready to project: `../images/arena-diagram.svg`. You will trace the flight from Blue's start (0.4, −0.6) to the point (0, −0.4).

- [ ] Have the editor layout picture handy for students who get lost in the screen: `../images/ide-graphical-editor-layout.png`.
- [ ] Know where the **init page / main page** tabs are, and where the "global variables" slot is on the init page. You will switch pages live in front of the class.
- [ ] Find the `absolute value` option yourself ahead of time: drag the Math block printed `square root ▾ [a]`, then open its dropdown and pick **absolute value**. Students will not guess this on their own.
- [ ] Print the student handout (or use `printable.md`, one per student).

Walkthrough: what to say and do

1. **Create the project.** Live, on the projector: New project → choose the game **AsteroidBee 2026** (look under archived/featured games if it isn't front and center) → **Graphical Editor**. Point out that the palette now has a new **AsteroidBee** category that wasn't in the Free Mode projects. Say: "That's the game's special equipment: cameras, mirrors, items. We won't touch it until Assignment 5. Today we use the everyday categories."

2. **Show the map.** Project `../images/arena-diagram.svg`. Trace with your finger: "Blue starts here, at (0.4, −0.6). We're flying to here: (0, −0.4). Today's mission isn't just to fly there: it's to make the satellite notice that it got there, and say so."

3. **Declare the notebook (init page).** Switch to the **init page**. From **Variables**, drag the array declare block (the one printed `type: ▾ name: __ length: __ …`) into the **"global variables"** slot. Set type to **float**, name it **myState**, length **12**. Say: "This is the satellite's notebook: 12 numbered lines that keep their values between seconds. We create it here on the init page because the init page runs exactly once, at the start of the match." Warn them now: "This declare block only shows up in the palette while you're ON the init page. If you can't find it later, check which page you're on."

4. **Sense (main page).** Switch to the **main page**. From **SPHERES Controls**, drag `get My ZRState ▾ [array]` into the "loop" slot. It comes with an array plug already attached. Open its dropdown and pick **myState**. Say: "Every second, this block writes my current state into the notebook. Twelve slots: slot 0 is my X position, slot 1 is my Y position. The rest are speed and which way I'm facing. We'll meet those later." Ask the class: "If this block runs every second, how old is the position in the notebook, at most?" (Answer: one second. It's always fresh.)

5. **Act.** Below it, add `setPos _, _, _` with 0, −0.4, 0. They know this block from Assignment 2: it sets the destination, and repeating it every second is fine.

6. **Ask the question.** From **Logic**, drag an `if … then …` block under `setPos _, _, _`. Now build the condition out loud, in English first: "I have arrived if my X is within 5 cm of 0 AND my Y is within 5 cm of −0.4." Build it in two halves and join them with the Logic block `[a] and ▾ [b]`:

7. **X half:** a compare block `[a] == ▾ [b]` set to **<**. Left side: the Math block `square root ▾ [a]` with its dropdown changed to **absolute value**, holding the array slot block `myState [ 0 ]`. Right side: a number block, **0.05**. Read it back: "the absolute value of my X is less than 0.05: within 5 cm of zero, whether I overshoot left or right."

8. **Y half:** the same, but inside the `absolute value`, put `myState [ 1 ]` **+ 0.4** (the Math `[a] + ▾ [b]` block). Pause here. This is the trick of the day. Ask: "My target Y is −0.4. How far am I from −0.4?" Write on the board: distance from −0.4 = |y − (−0.4)| = |y + 0.4|. "Subtracting a negative is adding, so we ADD 0.4, then take the absolute value." Compare: **0.05**, same as before.

9. **Announce.** Inside the `if`'s "then" slot: `DEBUG [text]` with a text block `""`. Type **Arrived!** between the quotes.

10. **Predict, then simulate.** Before running, ask the class to vote: "When the satellite gets there, does 'Arrived!' print once, or more than once?" Take the vote, then simulate (~90 seconds is plenty). They'll see: log silent, silent, silent... then **"Arrived!" every single second** until the end. Discuss: "The main page re-runs every second. On second 50 the answer to the `if` question is yes. On second 51 the page runs again, the question is asked again, and the answer is still yes. The program doesn't remember it already said this: it has no memory unless we give it one. Next assignment, we give it one."

11. **Show the C.** Flip the **C Code** toggle. Point out the matching lines: `api.getMyZRState(myState);`, `setPos(0, -0.4, 0);`, and the `if` with `fabsf` ("that's C's name for absolute value"). Also point out the twelve `myState[0] = 0;` … lines inside `init()`: "That's our declare block setting every notebook line to zero at the start. The blocks and the C are the same program in two costumes."

The mistakes you will see

Mistake 1: expecting the program to "wait at the `if` until it becomes true," then move on (blocks as a timeline).

- **What it looks like:** Students are confused (or annoyed) that "Arrived!" prints forever. Or they drag the `if … then …` block ABOVE `setPos _, _, _`, expecting "first check, then fly."
- **Why it happens:** They picture the blocks as a story that plays out over time: check, then fly, then arrive, then announce, once each. But the main page is a checklist re-read every second; the `if` question is asked fresh every second, before AND after arrival, and the page never "moves on" past anything.
- **The question to ask:** "On second 80, the satellite reads the whole page again. What is the answer to your `if` question now?"

Mistake 2: can't find the declare block.

- **What it looks like:** A student scrolls the Variables category up and down on the main page, insisting the array declare block doesn't exist.
- **Why it happens:** Declare blocks only appear in the palette while you are on the **init page**, because they only fit in the "global variables" slot there.
- **The question to ask:** "Which page are you on?"

Mistake 3: the arrival test never fires.

- **What it looks like:** The satellite clearly parks at the target, but the log stays silent for the whole run.
- **Why it happens:** One of three small slips: the wrong array slot (`myState [ 1 ]` where they meant `myState [ 0 ]`, or vice versa), the wrong sign (testing against 0.4 instead of −0.4, i.e. forgetting the `+ 0.4` or making it `- 0.4`), or a too-tight tolerance like 0.001 that the satellite's real, slightly wobbly position never satisfies.
- **The question to ask:** "Watch pX/pY in the viewer: how close does it actually get? Is your bubble big enough?"

Reference solution at a glance

On the **init page**, one block sits in the "global variables" slot: the array declare, set to type float, name **myState**, length **12** (initial values all 0).

On the **main page**, three blocks stack in the "loop" slot, top to bottom: `get My ZRState ▾ [array]` filling **myState**; `setPos _, _, _` with 0, −0.4, 0; then an `if … then …` block whose condition is two `absolute value` comparisons joined by `[a] and ▾ [b]` (|myState[0]| < 0.05 and |myState[1] + 0.4| < 0.05), with a single `DEBUG [text]` "Arrived!" inside the then-slot.

The exact layout and the C it generates are in reference-solution.md.

Wrap-up questions

1. "Arrived!" prints every second once the satellite is parked. Why doesn't it print just once? What would the program need in order to say it only once? (Tease: a memory. That's Assignment 4.)
2. Why do we read the position with `get My ZRState` on the **main** page instead of the init page? What would the notebook contain if we only filled it once, at the start?
3. Why test "within 0.05" instead of "exactly equal to 0 and −0.4"? Would a compare block set to **=** ever be true for a real, drifting satellite?