

# Assignment 5: Spy Camera

Aim. Check. Shoot. Every single second.

## Your mission

---

Agent, it's time to score. The rival corporation's satellite is parked 0.8 meters away, sitting in the sunlight like an amateur.

Build a program that:

1. **Aims at the rival automatically**, every second, with no typed-in direction numbers. If they moved, your camera would follow them.
2. **Takes photographs only when the shot will count**. A real spy doesn't waste film (or in your case, energy).


You'll know it works when your **score climbs** in the simulation viewer during the first minute (a couple of points per photo), and your satellite is smart enough to **hold fire** when shooting would be a waste.

## Mission rules

---

- Create your project on the **AsteroidBee 2026** game and simulate as **Blue** against the default (idle) opponent.
- Your satellite must aim itself. **No fixed direction numbers**: the aim must come from a block that measures where the opponent is.
- **Never shoot during the camera cooldown**. After every attempt the camera shuts off for 3 seconds, and an attempt during cooldown still costs you.
- After you refine your program: **do not shoot when the opponent is in the dark**, and **do not shoot when your energy is at or below 1.5**. Every attempt costs 1.0 energy (even a failed one), and you only recharge in the Light Zone.

The arena, drawn as in the official manual: +Y runs to the right, +X runs up. The Light Zone starts on the left (-Y) and the hatched Dark Zone on the right (+Y), and the two halves swap at  $t = 60$  s and again at  $t = 150$  s. Remember those two moments.

 **Every second**: your whole main page runs again, top to bottom. Re-aim at the rival. Re-ask your questions. Shoot only if every answer is yes. Then, one second later, all of it again. Your program doesn't do steps; it re-reads a checklist.

## Blocks to explore

---

- **AsteroidBee MS- Pictures**: everything a spy camera needs lives here: finding the rival, checking your gear, and pulling the trigger.
- **AsteroidBee MS- Light/Dark**: can a camera photograph someone hiding in the dark? Better to know before you spend the energy.
- **SPHERES Controls**: one movement block has a dropdown that changes it from "fly to a place" into "point in a direction."
- **Variables** (on the init page): a direction needs a 3-slot array to live in.
- **Logic**: one `if / then` can ask several questions at once, glued together with "and."
- **Math**: the number block (like `1.5`) for the energy check lives here; the compare block itself is in Logic.

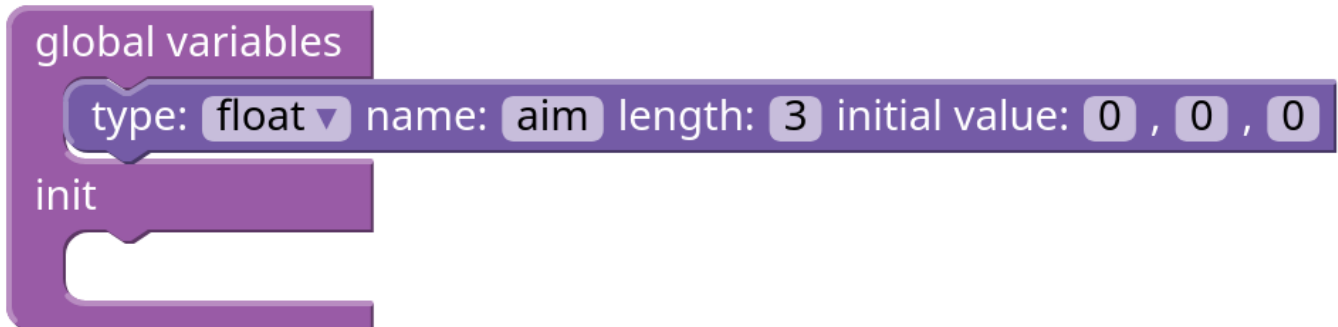
## Hint

You cannot tell the satellite to "wait until it is facing the enemy." But you can ASK, every single second: "facing? camera ready?", and only shoot on the seconds when the answer is yes.

## Build it, step by step

### Part 1: give the direction a home (init page)

1. Switch to the **init** page. Open **Variables**, drag the **declare array** block into the global variables slot, name it `aim`, keep the type **float**, and set the length to **3**. Your init page should match this:



That 3-slot array is where the direction to the rival will live, refreshed every second.

### Part 2: aim the camera (main page)

2. Switch to the **main** page. From **AsteroidBee MS- Pictures**, drag this block into the loop slot:



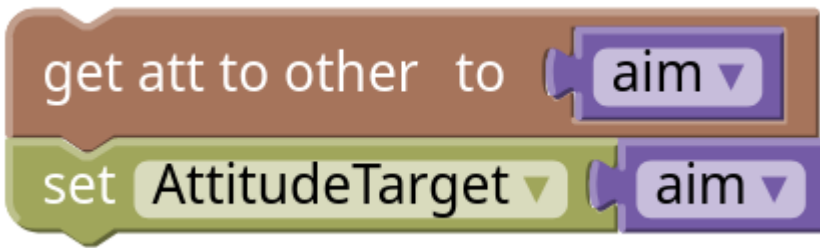
It arrives with a whole-array block already plugged into its socket; use that block's dropdown to pick **aim**. Every second, this fills `aim` with the direction that points at the opponent.

3. From **SPHERES Controls**, snap this block directly underneath:



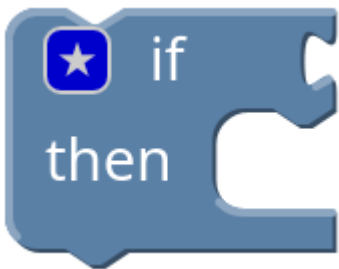
Measuring a direction isn't the same as turning toward it, so switch its first dropdown from `set PositionTarget` to `set AttitudeTarget`, and pick **aim** on its array block. Now the satellite turns to face wherever `aim` points.

4. Checkpoint: your first two blocks should look exactly like this aiming pair:



**Part 3: only shoot when it counts**

5. From **Logic**, snap an if block below the aiming pair:



It reads **if / then**. Anything inside the then slot runs only on the seconds when the question in the if socket answers yes.

6. From **AsteroidBee MS- Pictures**, put the trigger inside the **then** slot:



Don't simulate yet: right now this would fire blindly every second, cooldown or not.

7. From **Logic**, plug this into the if block's question socket:



One **and** block lets the if ask two questions at once; both must be yes.

8. From **AsteroidBee MS- Pictures**, plug the first question into the **and** block's **left** socket:



Am I actually aimed at the rival yet? For the first few seconds, the answer is no.

9. From the same category, plug the second question into the **right** socket:

is camera on

This answers no during the 3-second cooldown after every attempt: your "never shoot during cooldown" rule, asked fresh each second.

**10. First test run.** Simulate as **Blue**. Watch the satellite rotate, then the score climb in steps during the first minute. Now keep watching past **t = 60 s...** the rival is suddenly in the dark, your photos fail, and every failed attempt still costs 1.0 energy. Time to refine.

#### Part 4 — refine: no wasted shots

**11.** From **AsteroidBee MS- Light/Dark**, grab this question:

check in light

Switch its dropdown from **me** to **other**: you care whether the rival is lit. Unplug your two-question chain from the if, drag in a fresh **and** block, plug the old chain into its **left** socket and **check in light other** into its **right** socket, then plug the whole thing back into the if.

**12.** From **Logic**, drag a compare block to an empty spot on the workspace:



Set its dropdown to **>**. This will become the energy question.

**13.** From **AsteroidBee MS- Light/Dark**, plug this into the compare block's **left** socket:

get  energy

Leave its dropdown on **my**: it's your own battery you're guarding.

**14.** From **Math**, plug a number block into the compare block's **right** socket and type **1.5**:



Now the question reads "is my energy above 1.5?": your reserve line.

**15.** One last **and** block from **Logic**: chain on the left, the energy compare on the right, everything back into the if. The full guard now asks four questions and should match this exactly:

```

if [star] if
  [is facing other] and [is camera on] and [check in light other] and [get my energy > 1.5]
then
  take pic

```

16. Checkpoint: your finished main page, aiming pair plus guard:

```

loop
  get att to other to aim
  set AttitudeTarget aim
  if [star] if
    [is facing other] and [is camera on] and [check in light other] and [get my energy > 1.5]
  then
    take pic

```

Simulate again and head to **Check your work** below.

## Check your work

---

Run the simulation and watch for all of these:

- **First few seconds:** your satellite rotates to face the rival. No photos yet: your program is asking, and the answer is still no.
- **First minute (0-60 s):** the scoreboard climbs in steps of roughly **2.3-2.5 points** every few seconds. Each step is one successful photo.
- **Middle of the match (60-150 s):** the lights swap and the rival is hiding in the Dark Zone. Your refined program **stops shooting entirely:** the score pauses, but your energy is not draining away on wasted shots.
- **After 150 s:** the lights swap back, and your camera starts scoring again, all by itself, with no time blocks anywhere in your program.
- **The log shows no failed attempts** after your refinement.

## 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: your program contains no clock and no "wait", so why does it stop taking pictures at exactly  $t = 60$  and start again at  $t = 150$ ?