Assignment 6: Power and Pickups

Name: ____ Date: _____

Cross into the dark, grab the goods, top off your batteries, and slip back out.

Your mission

Headquarters has a shopping list. Out in the arena there are nine pieces of debris at known locations. Your teacher will show you the arena map, drawn like the official manual: **+Y runs to the right, +X runs up**, the **Light Zone** is on the left (−Y), the hatched **Dark Zone** is on the right (+Y), and the halves swap at **t = 60 s** and **t = 150 s**. The **squares** (score items 4, 5, and 6, the "asteroids") are worth **+1.5 points** each. The **circles** (energy packs 0–3) refill your energy to a full **5.0**. The **diamonds** (mirrors 7 and 8) are not on today's shopping list.

Build a three-step mission plan:

1. **Collect score item 4**, the square sitting in the Dark half at (0, 0.6).
2. **Collect energy item 0**, the circle at (0.25, −0.40).
3. **Park at (0.25, −0.2)** and hold there for the rest of the match.

Here's the secret the game manual buries: there is **no "grab" block**. To pick up an item, you must be within **5 cm** of it and moving slower than **1 cm/s**. In other words: **park on it**. Your position controller already knows how to park. Your job is to give it the right destination and know when the pickup actually happened.

You'll know you succeeded when the log announces both pickups, your score jumps by 1.5, your energy reads full again, and your satellite ends the match parked at (0.25, −0.2).

> ⏱ **Every second:** your main page runs again from the top. It checks your step number, hands the movement system its destination again, and asks the game "do I have that item yet?" Nothing waits; the answer is just "no" for a while. The first second it's "yes," your step number changes, and the next second a different branch of your plan runs.

Mission rules

- Create your project on the **AsteroidBee 2026** game, using the Graphical Editor.
- The plan must run in this order: score item **4**, then energy item **0**, then hold at **(0.25, −0.2)**.
- Steps must advance using `check have item _ me`, the game's own confirmation. Do **not** advance on a coordinate check. Being at the item is not the same as having the item.
- Your step variable must be declared on the **init page**.

Blocks to explore

- **AsteroidBee MS- Items**: the game will tell you where any item is, and who holds it.
- **SPHERES Controls**: one movement block here takes a whole 3-slot array as its destination; another takes numbers you type.
- **Variables**: your plan needs a step number, and a 3-slot array for the game to fill in. Both belong on the init page.
- **Logic**: your plan must ask "which step am I on?" and "is this step finished?" every single second.
- **Debug**: leave yourself a note in the log the moment each pickup lands.
- **AsteroidBee MS- Light/Dark**: not required today, but peek. Where do you recharge, and when do the halves swap?

Hint

The game itself can tell you two things every second: WHERE an item is (into an array), and WHETHER you have it yet. One of those is your destination; the other is your "step finished" question.

Build it, step by step

(The block pictures are on the projector and the class handout.)

Part A: the init page (give the satellite a notebook)

☐ **1. Create the project.** New project, **AsteroidBee 2026** game, **Graphical Editor**. You start on the **init page**, the page that runs once, at the very start of the match.

☐ **2. Declare your two variables.** Open the **Variables** category (the declare blocks only appear in the palette while you're on the init page) and build this inside the **"global variables"** slot:

- A `declare array` block: type **float**, named **dest**, length **3**. This is the empty array the game will fill in with an item's location, every second.
- Below it, a `declare` block: type **int**, named **step**, initial value **0**. This is the satellite's memory of which step of the plan it's working on.

Part B: the main page, the step-0 branch (go get the square)

Switch to the **main page**. Everything you build here runs once per second, every second, for the whole match.

☐ **3. Start with a question.** From **Logic**, drag an `if / then` block into the "loop" slot.

☐ **4. Make the question "am I on step 0?"** From **Logic**, plug a compare block into the `if` socket and leave its dropdown on `==` (it also offers `!=`, `<`, `<=`, `>`, `>=`; not today). From **Variables**, drop the `step` block (it shows just the variable's name) into the left side. From **Math**, put a number block on the right side and leave it at **0**. The header now reads `if step == 0 then`.

☐ **5. Ask the game where item 4 is.** From **AsteroidBee MS- Items**, drag a `get item 0 loc to [array]` block into the `then` slot. Change the item dropdown from **0** to **4**. The whole-array block comes pre-attached. Set its dropdown to **dest**. Every second this runs, the game writes item 4's location into your `dest` array.

☐ **6. Make that location your destination.** From **SPHERES Controls**, drag a `set PositionTarget [array]` block directly underneath. Leave its dropdown on **PositionTarget**. From **Variables**, plug a whole-array block into its socket and set it to **dest**. That's the entire flight plan for this step: wherever item 4 is, park there.

☐ **7. Ask "do I have it yet?"** Drag a second `if / then` block inside the same `then` slot, below `set PositionTarget`. Into its `if` socket, plug a `check have item 0 me` block from **AsteroidBee MS- Items**. Change the item dropdown to **4** and leave the second dropdown on **me**. This is the game's own pickup confirmation, the only test allowed to advance your plan.

☐ **8. When the answer is yes, advance and announce.** Inside the inner `then`:

- From **Variables**, drag the `step = [value]` block (it shows the name, an equals sign, and a value socket) and put a number block reading **1** in its socket.
- Below it, from **Debug**, drag a `DEBUG` block, plug a text block into its socket, and type **Score item collected!** between the quotes.

☐ **9. Compare your branch against the picture on the projector.** Your whole step-0 branch should match it exactly.

Part C: the step-1 branch (top off the batteries)

☐ **10. Build the same shape again,** directly below the first branch, with exactly four changes:

- The question is `if step == 1 then`.
- The `get item _ loc to [dest]` dropdown is item **0** (the energy circle).
- The `check have item _ me` dropdown is also item **0**.
- When it's yes: `step = 2`, and the DEBUG message is **Energy topped up!**

Part D: the step-2 branch (park and hold)

☐ **11. One last question, one last block.** Below the second branch, add a third `if / then` asking `if step == 2 then` (compare block, `step`, number **2**). Inside its `then`, drag a `setPos` block from **SPHERES Controls** and type the parking spot into its three number slots: **0.25**, **−0.2**, **0**.

☐ **12. Check your whole workspace** against the finished-layout picture on the projector: three branches, top to bottom. If yours matches, you're ready to fly.

Check your work

Simulate the full 180 seconds as **Blue** against an idle opponent. You should see:

- Your satellite crosses into the dark half and **settles onto item 4**: it stops on the spot, it doesn't just fly past.
- The moment item 4 is yours: your **score jumps +1.5** and your first DEBUG message appears in the log.
- It then travels to item 0; your **energy snaps back to a full 5.0** (it doesn't recharge in the dark, so it drifted down a hair) and your second DEBUG message prints.
- For the rest of the match, the satellite sits parked at **(0.25, −0.2)**.

If your satellite flies over an item and nothing happens, did it stop on the item, or just visit? And if it collects item 4 but then sits on the empty spot forever, what is supposed to change your step number?

Finished early? The game can also tell you what kind of debris an item is: the `get item 0 type` block reports **0** for score, **1** for energy, **2** for mirror. Can you make your plan pick its destination by kind, instead of hard-coding which item number is which?

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: **if the opponent grabbed item 4 before you got there, what would your program do for the rest of the match, and why?**