THE BEACON

Term 1 · Weeks 3–6 · Events + loops · screen only

GOAL

Your console is running but nobody knows it. A supply ship passes every so often, and the only thing it can see is a light.

Build a beacon that:

- flashes **short, short, short, long, long, long, short, short, short**
- keeps repeating forever, on its own
- stops with one key press
- restarts from the beginning with a different key

VISUAL BLOCK HINTS





A long flash is a short flash with a bigger number. Find the one number that changes.

THE BEACON

DONE WHEN…

- The pattern runs 30 seconds with nobody touching it
- Pressing **S** stops it and the lamp goes dark
- Pressing **G** restarts from the beginning of the pattern
- You can name which block repeats and how many times

IF YOU'RE STUCK

1. Build the **short** flash first — one ON, one OFF. Run it.
2. If nothing happens at all, check for the hat block.
3. If it never stops, you are missing `stop other scripts`.
4. Still stuck? Ask a partner to read your blocks out loud to you.

⭐ EXTRA CREDIT — THE SILENT BEACON

> Add a sound on every long flash but not the short ones. A listener who cannot see the lamp must still read the message.
>
> **Then:** one variable `speed` that speeds up or slows the whole beacon without changing any other block.

THE NAVIGATOR

Term 1 · Week 3 onward · Coordinates + heading · screen only

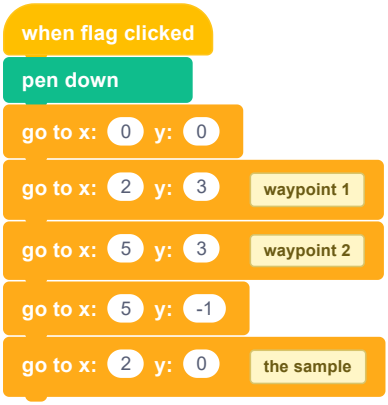
GOAL

A survey drone must reach five markers in order, without crossing the crater squares, and end within one square of the sample.

Build a route that:

- starts at `(0,0)`
- visits every waypoint **in order**
- crosses no crater
- ends within **one square** of the target
- runs from **one tap** — no manual steering

VISUAL BLOCK HINTS



A straight line between two waypoints is a decision. If it crosses a crater, the fix is another waypoint, not better code.

THE NAVIGATOR

DONE WHEN…

- Route drawn on paper before any blocks
- All five waypoints reached in order
- No crater crossed
- Ends within one square of the target
- Runs from one tap, twice in a row, with the same result

IF YOU'RE STUCK

1. Draw the route on the card's grid first.
2. Mark the squares your straight lines cross.
3. Add a waypoint before the crater.
4. Run it once and compare the end point with your drawing.

⭐ EXTRA CREDIT — THE EFFICIENT ROUTE

> Fewer go to blocks for the same route — then argue in writing why fewer is not always better.
>
> **Then:** a second sprite taking a different route at the same time, both avoiding craters.

THE LOTUS LOOM

Term 1 · Weeks 6–7 · Nested loops · screen only

GOAL

Padma means lotus. Every family in the settlement brings one **tile** — a small pattern — and the wall is built from everyone's tiles.

Build a wall that:

- contains **one tile you designed yourself** (4–9 squares, drawn on paper first)
- repeats it in a **3 × 3 grid**
- keeps the gaps exactly equal
- fills at least two-thirds of the stage

You may not copy the tile nine times by hand. The wall must use a loop inside a loop.

VISUAL BLOCK HINTS



The bottom block is the one everybody forgets. Without it your wall drifts sideways and walks off the stage.

THE LOTUS LOOM

DONE WHEN…

- Nine tiles visible and evenly spaced
- The pattern does not drift or tilt
- Removing one repeat block changes the wall predictably
- Notebook says: "*My outer loop does ___ and my inner loop does ___.*"

IF YOU'RE STUCK

1. Draw one tile on grid paper. Count its squares.
2. Build **one** tile. Run it. Look at it.
3. Wrap it in `repeat 3` with a `move` after it. That's a row.
4. Wrap the row in another `repeat 3`. That's a wall.
5. If the wall drifts, you are missing the reset between rows.

⭐ EXTRA CREDIT — THE WEAVER'S WALL

> Choose **one**: brick offset every second row · two motifs in a checkerboard · a border frame using a separate loop.
>
> **Then:** write instructions precise enough for another team to rebuild it with no picture.

THE SLALOM

Term 1 · Weeks 4–6 · Events + broadcast · needs the robot

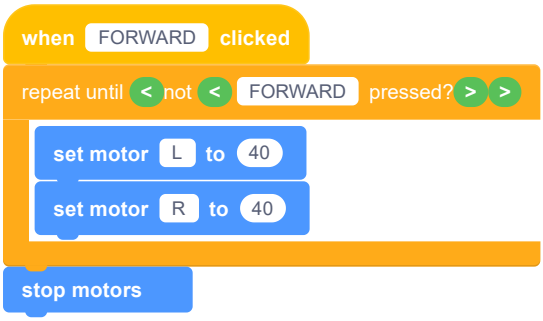
GOAL

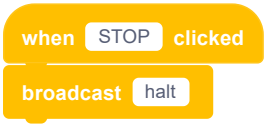
Four buttons. One machine. A course of five marks. Drive it clean.

Build a remote that:

- drives KanKaung forward, back, left and right
- stops the moment you release — no runaways
- lets a stranger drive it with no explanation
- completes a five-mark slalom without touching two marks

VISUAL BLOCK HINTS





The loop condition is the whole design. "Repeat until the button stops being pressed" is a robot that stops when you stop meaning it.

THE SLALOM 🤖

DONE WHEN…

- Four buttons drive, one stops
- Releasing every button stops the robot within half a second
- Another team's Navigator drives it without asking how
- Five marks passed, two or fewer touched

IF YOU'RE STUCK

1. Test one button, one motor, one second.
2. If it runs forever, your command is sent once — wrap it in a loop.
3. If it won't stop, the loop condition is wrong, not the robot.
4. Keep speed at 40. Faster is not better here.

⭐ EXTRA CREDIT — THE SPEED DIAL

> Add two buttons that raise and lower a `speed` variable, and show its value on the stage.
>
> **Then:** a "turtle mode" button that halves the speed for a narrow gap.

TERM 2 — RULES

THE RATION DESK

Term 2 · Weeks 4–5 · Variables + conditionals · screen only

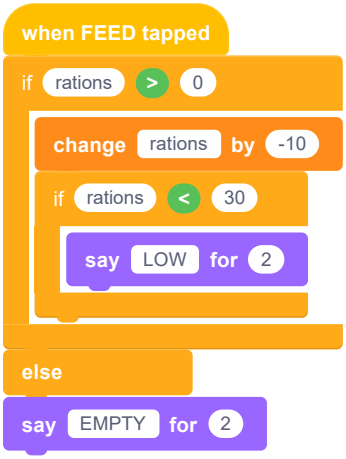
GOAL

The stores are running down. You will not be there every hour, so the **desk** has to decide for itself.

Build a desk that:

- starts with **100 rations**
- gives out **10** each time FEED is pressed
- **never** goes below zero — at zero it says `"EMPTY"` and refuses
- warns below 30, but only **once**
- shows the current amount at all times

VISUAL BLOCK HINTS



The warning must fire once, not on every press after 30. Think: what has to be true *before* you warn?

THE RATION DESK

DONE WHEN…

- Number visible at all times, drops by exactly 10 per press
- Pressing at zero does nothing except the `"EMPTY"` message
- The warning appears once and does not repeat
- Fifteen presses and the number never goes negative

IF YOU'RE STUCK

1. Get `set [rations] to 100` working before anything else.
2. Press FEED once and watch the number. If it does not move you used `set` instead of `change`.
3. If it goes negative, your `if` checks *after* the change — move it before.
4. If the warning repeats, you need a second variable that remembers you already warned.

⭐ EXTRA CREDIT — THE QUARTERMASTER

> Add RESTOCK (adds 50, never above 100) and STATUS (reports how many **minutes** remain at 10 per minute).
>
> **Then:** an operator card — three lines that let a stranger use the desk without breaking it.

THE RIDDLE GATE

Term 2 · Week 3 · Compound conditions · screen only

GOAL

A gate that opens only for the right combination. Three tests, and the rule is not obvious.

Build a gate that opens when:

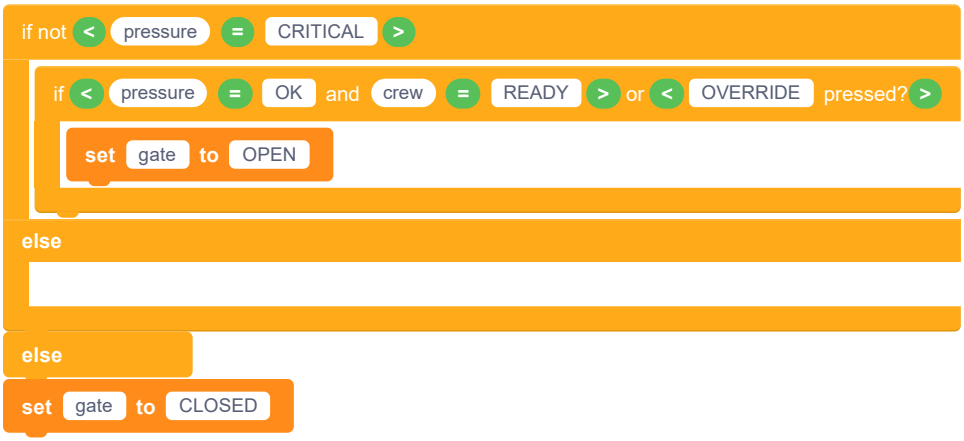
> **pressure is OK AND crew is ready**, *or* **an override key is held** — but never when **pressure is critical**, whatever else is true.

Then:

- show all three tests on the stage as green/red indicators
- show whether the gate is open
- prove it with a **decision table** of at least eight rows

Name the gates first. The rule uses all three: AND, OR and NOT. Fill the four-row truth table for AND and for OR before you attempt the eight-row table — same shape, one input smaller. Three inputs means eight rows; two inputs means four.

VISUAL BLOCK HINTS



Read the brackets carefully. `not A and B` is not the same as `not (A and B)`. One of them is a bug.

THE RIDDLE GATE

DONE WHEN…

- AND and OR truth tables completed before the eight-row table
- Eight-row decision table, hand-filled, before any code
- All eight rows match the code when tested
- Three indicators behave correctly
- You can explain the override in one sentence

IF YOU'RE STUCK

1. Fill the table by hand first. All eight rows.
2. Build **one row** and test it.
3. Add the override last — it is the exception, not the rule.
4. If two rows disagree with the code, the table is probably right.

⭐ EXTRA CREDIT — THE HOSTILE ROW

> Design an input combination that *looks* valid but should be refused. Add it as a ninth row.
>
> **Then:** write two sentences explaining why a stranger might try exactly that combination.

THE MODE MACHINE

Term 2 · Week 6 · State machine · screen only

GOAL

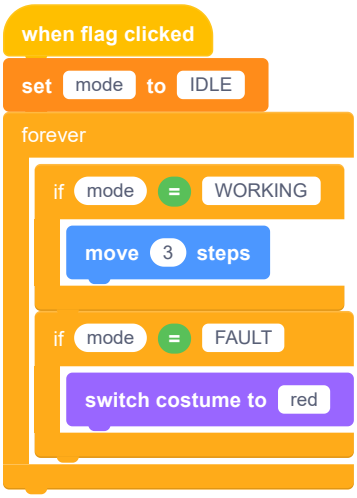
One machine, four jobs, one variable. No mode should ever be reachable by accident.

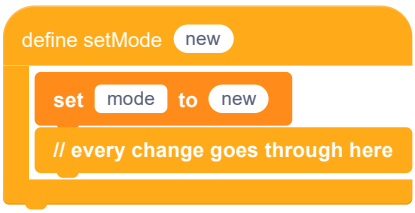
Build a machine with four modes:

- **IDLE** — nothing moves, grey
- **WORKING** — the sprite moves, green
- **PAUSED** — nothing moves, amber
- **FAULT** — flashes red, and only RESET leaves it

Rules: one `mode` variable · one owner script · every transition drawn as an arrow before you code.

VISUAL BLOCK HINTS





Draw the four circles and the arrows first. An arrow you cannot draw is a transition that does not exist.

THE MODE MACHINE

DONE WHEN…

- Mode diagram complete, every arrow labelled with its trigger
- All four modes work and look different
- Only `setMode` ever writes to `mode`
- FAULT can be reached from any mode and left only by RESET

IF YOU'RE STUCK

1. Two modes first: IDLE and WORKING.
2. If the sprite flickers, two scripts write to `mode`.
3. If a mode never triggers, its transition is in a script that never runs.
4. Draw the diagram again — the bug is usually missing from the picture.

⭐ EXTRA CREDIT — THE ILLEGAL MOVE

> Add a guard that refuses an illegal transition and logs it on the stage.
>
> **Then:** write the four legal transitions as a table, and make the code match it exactly.

THE TEST BENCH

Term 2 · Week 10 · Boundary testing · screen only

GOAL

You have a program that works. Now find the values that break it — before a user does.

Build a test bench that:

- tests a rule you wrote in an earlier card
- covers **normal**, **boundary** and **hostile** values
- records pass/fail for at least eight cases
- includes the number **exactly at the threshold**

VISUAL BLOCK HINTS



For every < you wrote, test the value, the value − 1, and the value + 1.

THE TEST BENCH

DONE WHEN…

- Eight rows: at least two normal, four boundary, two hostile
- Every row has an expected result written *before* running
- Failures are recorded as findings, then fixed
- One sentence: "*The nastiest input was ___ because ___.*"

IF YOU'RE STUCK

1. Pick one rule with a number in it.
2. Test 0, 1, the threshold, threshold − 1, threshold + 1, and 1000.
3. If everything passes, you are not testing hard enough.
4. If something fails and you cannot fix it, write it down — that is still a result.

⭐ EXTRA CREDIT — THE SABOTAGE SWAP

> Swap projects with another team. Change **one thing** in theirs that makes it fail. Write a bug report: what you did, expected, actual.
>
> **Then:** they must find it using only your report.

TERM 3 — AUTONOMY

> 🤖 **All six cards in this group need the robot.** They are for teams that finish the weekly task early, or for the weeks after the course is complete.

THE PARKING ASSISTANT

Term 3 · Weeks 1–2 · Motors + timing

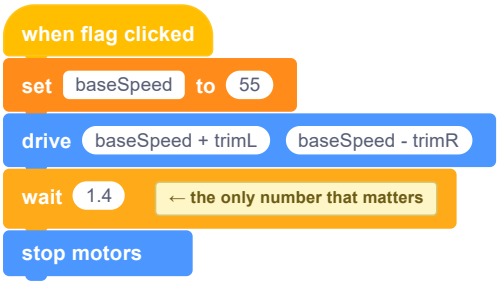
GOAL

Drive forward, stop inside a marked box, stay there. No sensors — only timing and calibration.

Build a robot that:

- drives straight forward at a steady speed
- stops **inside** a 20 cm box
- does it three times in a row from the same start
- never overshoots the far edge

VISUAL BLOCK HINTS



The number inside `wait` is your whole solution. Everything else is calibration.

THE PARKING ASSISTANT 🤖

DONE WHEN…

- Three consecutive runs end inside the box
- The robot is straight, not curved, at the end
- You can state your trim values and how you found them
- Another team can repeat your result from your notes

IF YOU'RE STUCK

1. Run once and measure where it stopped.
2. Too far? Reduce `wait` by 0.2. Short? Increase by 0.2.
3. Curving? That is trim, not timing.
4. Fresh batteries — a weak pack changes everything.

⭐ EXTRA CREDIT — THE DISTANCE BLOCK

> Build `driveFor (seconds)` and call it three times with different values to visit three marks in order.
>
> **Then:** predict the distance for 2 seconds, then measure. How close were you?

THE SQUARE DANCE

Term 3 · Weeks 5–7 · Functions + timing

GOAL

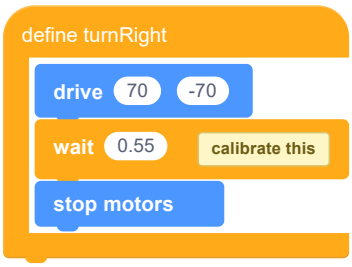
No line. No sensors. The robot, four straight sides, four exact turns.

Build a robot that:

- drives a square, 40 cm per side
- turns 90° at each corner
- returns to its start within 10 cm
- uses **one** custom block for a side and **one** for a turn

VISUAL BLOCK HINTS





Two numbers decide everything: how long a side takes, and how long a turn takes.

THE SQUARE DANCE 🤖

DONE WHEN…

- Four sides, four turns, one loop
- Ends within 10 cm of the start
- Both custom blocks used at least twice
- Turn time calibrated, not guessed

IF YOU'RE STUCK

1. Calibrate **one** 90° turn first. Mark the floor.
2. Then calibrate one 40 cm side.
3. Put them in `repeat 4`.
4. If it spirals, the turn `wait` is wrong — not the side.

⭐ EXTRA CREDIT — THE CALIBRATION TABLE

> Find the turn time at three speeds. Does it change? Why might it?
>
> **Then:** a triangle instead of a square — what changes, and by how much?

THE LINE SPRINT

Term 3 · Weeks 5–6 · Tuning

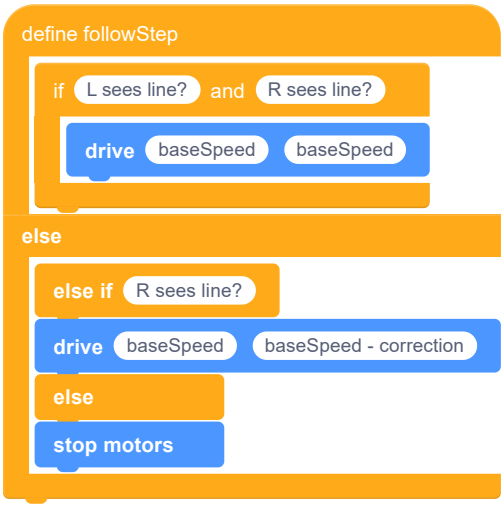
GOAL

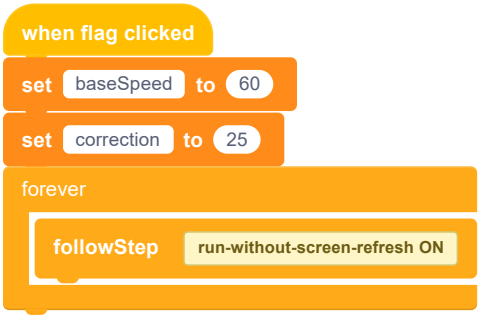
The course is fixed. The clock is not.

Build a follower that:

- completes the full course
- finishes in the shortest time your team can achieve
- succeeds **4 out of 5** attempts at that speed
- does not oscillate on the curves

VISUAL BLOCK HINTS





Fast and stable are opposites. Find where they meet.

THE LINE SPRINT

DONE WHEN…

- Lap time recorded for three speeds
- Success rate recorded for each (out of 5)
- You chose the fastest speed with 4/5 success
- One sentence: "*We chose ___ because ___.*"

IF YOU'RE STUCK

1. Get a **slow** lap working first. Time is meaningless if it fails.
2. Raise speed by 10. Re-test five times.
3. Oscillating on curves? Reduce `correction`.
4. Falling off? Increase `correction`.

⭐ EXTRA CREDIT — THE TRADE-OFF GRAPH

> Plot speed against success rate and explain the shape to another team.
>
> **Then:** slow down automatically on curves — does the lap get faster or slower?

THE GAP RUNNER

Term 3 · Week 10 · Memory

GOAL

There is an 8 cm gap in the line. For about half a second the robot is blind.

Build a robot that:

- follows the line
- crosses the gap without stopping
- comes out the other side back on the line
- stops if it has been blind for **too long**

VISUAL BLOCK HINTS



Short blindness — carry on. Long blindness — stop. The robot needs a memory of being blind.

THE GAP RUNNER 🤖

DONE WHEN…

- Crosses the gap 5 out of 5 attempts
- Stops if lifted off the mat for more than a second
- Comes out of the gap at reduced speed
- You can explain what `blind` counts and why 20 is the limit

IF YOU'RE STUCK

1. Cross **slowly** first. Raise speed later.
2. If it stops at the gap, `blind` is being reset — check the branch.
3. If it never stops when lifted, the counter is cleared somewhere.
4. If it lands off the line, slow down before the gap.

⭐ EXTRA CREDIT — REACQUISITION

> Make it find the line again even if it lands half a robot-width to one side.
>
> **Then:** measure how far off it can recover from.

THE WATCHDOG

Term 3 · Week 7 · Distance + hysteresis

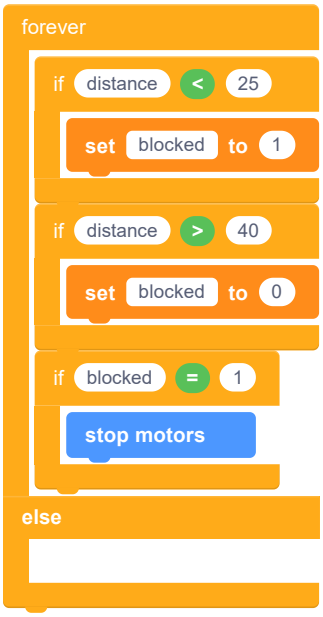
GOAL

The robot must never touch an obstacle — and must never twitch.

Build a robot that:

- stops when something is within 25 cm
- resumes only beyond 40 cm
- shows its state on the stage display
- does not flicker when the obstacle sits at exactly 30 cm

VISUAL BLOCK HINTS



Two numbers, not one. The gap between them is the deadband.

THE WATCHDOG 🤖

DONE WHEN…

- Stops at 25 cm, resumes at 40 cm
- Held at 30 cm for 15 seconds with no flicker
- State clearly visible to someone three metres away
- Notebook explains why one threshold was not enough

IF YOU'RE STUCK

1. Read the distance first. Do not set thresholds blind.
2. Flickering at 30 cm? Widen the gap.
3. Never resumes? Nothing sets `blocked` back to 0.
4. Never stops? Check the sensor is reading at all.

⭐ EXTRA CREDIT — THREE ZONES

> Near / OK / far, each with its own behaviour, and hysteresis on **both** boundaries.
>
> **Then:** measure the sensor's noise for 30 seconds and set the gap to twice that.

THE PATROL LOOP 🤖

Term 3 · Week 8 · State machine on hardware

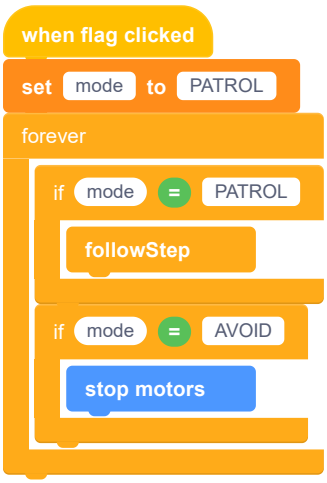
GOAL

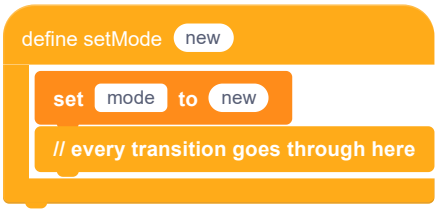
One robot, four jobs, one variable, nobody watching.

Build a robot that:

- has four modes: **PATROL / AVOID / STOP / RETURN**
- runs every behaviour from a single `mode` variable
- only one script may change `mode`
- never enters an illegal transition
- survives 10 minutes without a human

VISUAL BLOCK HINTS





Draw the four circles and the arrows before you code.

THE PATROL LOOP 🤖

DONE WHEN…

- All four modes work and look different
- Every change goes through `setMode`
- STOP is only escapable by the button
- Ten-minute run with no manual correction
- Mode diagram matches the code

IF YOU'RE STUCK

1. Build **two** modes first: PATROL and STOP.
2. Flickering? Two scripts write to `mode`.
3. Nothing happens? The `forever` is outside the hat.
4. A mode never triggers? Its transition is in a script that never runs.

⭐ EXTRA CREDIT — THE HANDOVER

> Write a one-page handover for the next shift: what each mode does, what makes you nervous, one thing you would fix with another hour.
>
> **Then:** swap robots with another team and inject one fault. They find it from your documentation alone.

APPENDIX — CLUB CARD

THE SIGNAL LANGUAGE

Club only · GPIO expansion · needs an LED, buzzer and free pins

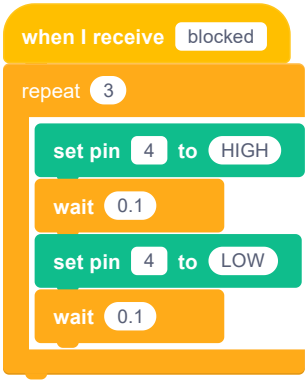
> This card uses the **8 free GPIO** and belongs to the Robotics Club, not the assessed curriculum. It is here for a team that finishes everything in Term 3.

CLUB CARD — THE SIGNAL LANGUAGE

Club · GPIO expansion

Goal. The robot has no screen. Everything it needs to say, it says in light and sound.

Build: at least three distinct states · two channels (LED and buzzer) · each recognisable across the room · written down so anyone can read it.



A signal nobody can read is not a signal. Design the code, then teach it to another team.

Done when: three states, visually distinct · a teacher who was not here can identify each · the language is taped to the robot · wiring diagram includes polarity and resistors.

Padma Challenge Cards v2.0 · Awaken Innovation Lab · Print A4, laminate, two-sided.