UNIVERSITY · PRESENTATION 3 · LEARNING WITH C

Gold hunt

The robot’s camera: what it sees, how to read its pixels, and a program that finds gold and leaves iron and silver.

TheBotCup 2026 · AiBotCup Simulator · University



RECAP

Two kinds of sensing so far

Sessions 1–11

Time and distance

Time, ultrasonic, compass: drive, avoid walls, solve mazes.

Sessions 12–13

Position

PositionX, PositionY, atan2d and angleDiff: aim at a point.

Session 14 · today

The camera

See what is on the floor, and where.

The Advanced robot has **no colour sensors**. Everything it knows about the floor comes from one camera.

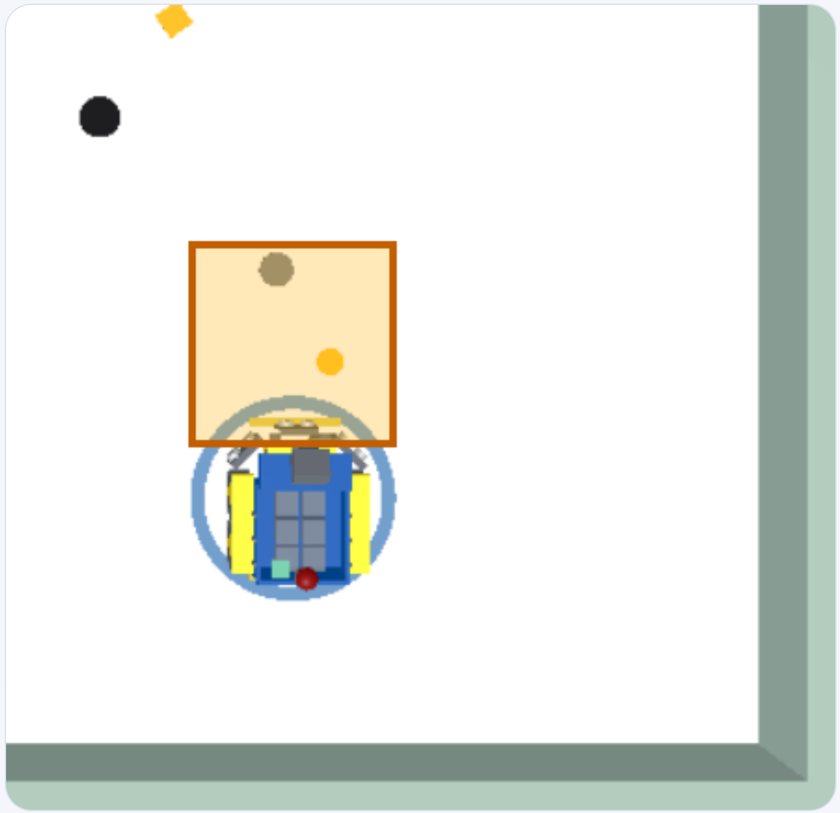
PART 1

The camera

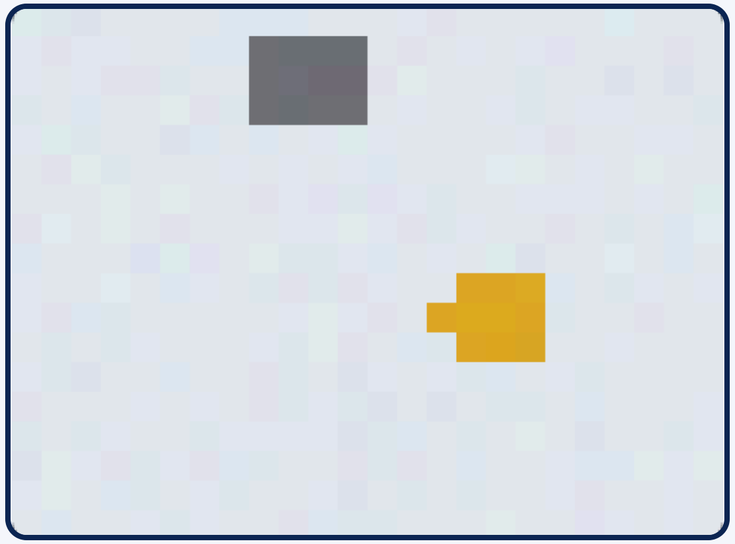
What it sees, and how a program reads it

THE CAMERA · WHAT IT IS

A grid of coloured pixels, looking at the floor ahead



Top view: the orange square is what the camera sees.



The Camera window: the same patch.

One camera at the **front centre**, looking **down** at the floor ahead.

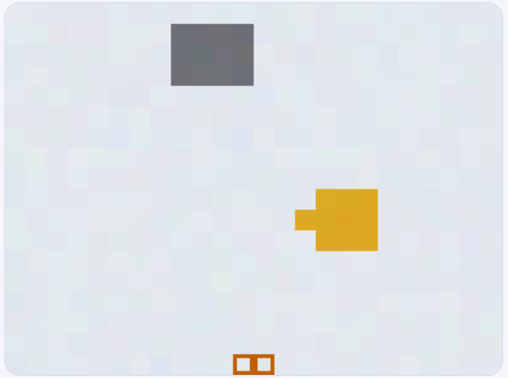
A patch of about **30 × 30 cm**, from 7 to 37 cm in front of the robot’s centre.

Objects, zones, the pond, warning bands: all show up as colours.

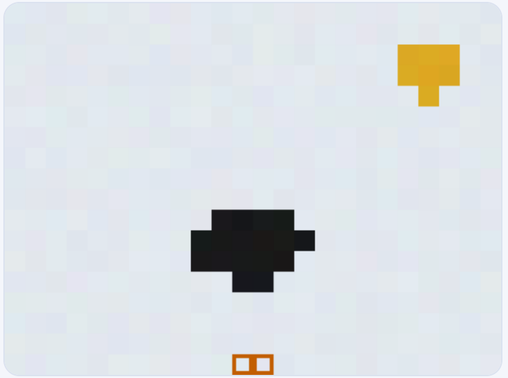
24 × 18 pixels: one colour per square.

THE CAMERA · WHAT IT SEES

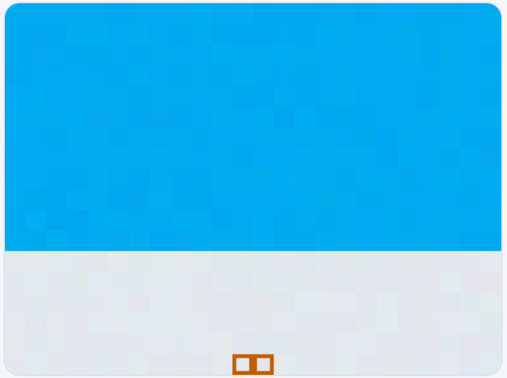
Every surface has its own colour



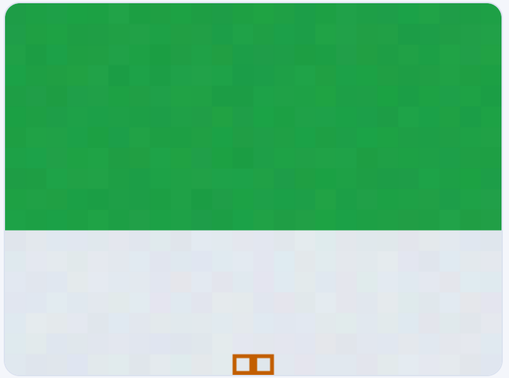
Silver (top) and **gold** (right)



Iron: almost black



Water pond: blue

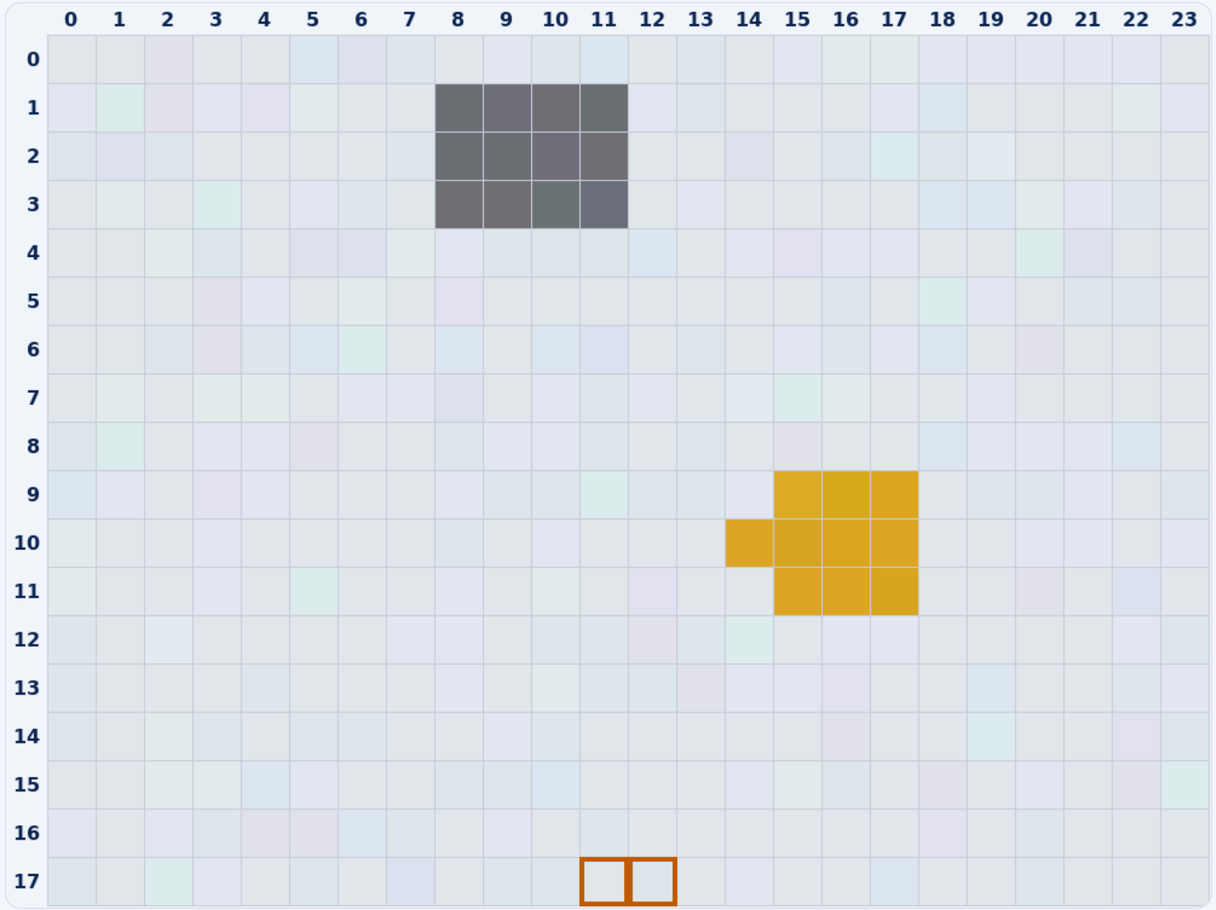


Deposit area: green



Warning band (light red) around an **unsafe area** (dark red).

The two squares outlined in orange at the bottom centre are the **pickup point**: collecting, depositing and traps count there.



THE CAMERA · HOW THE IMAGE IS STORED

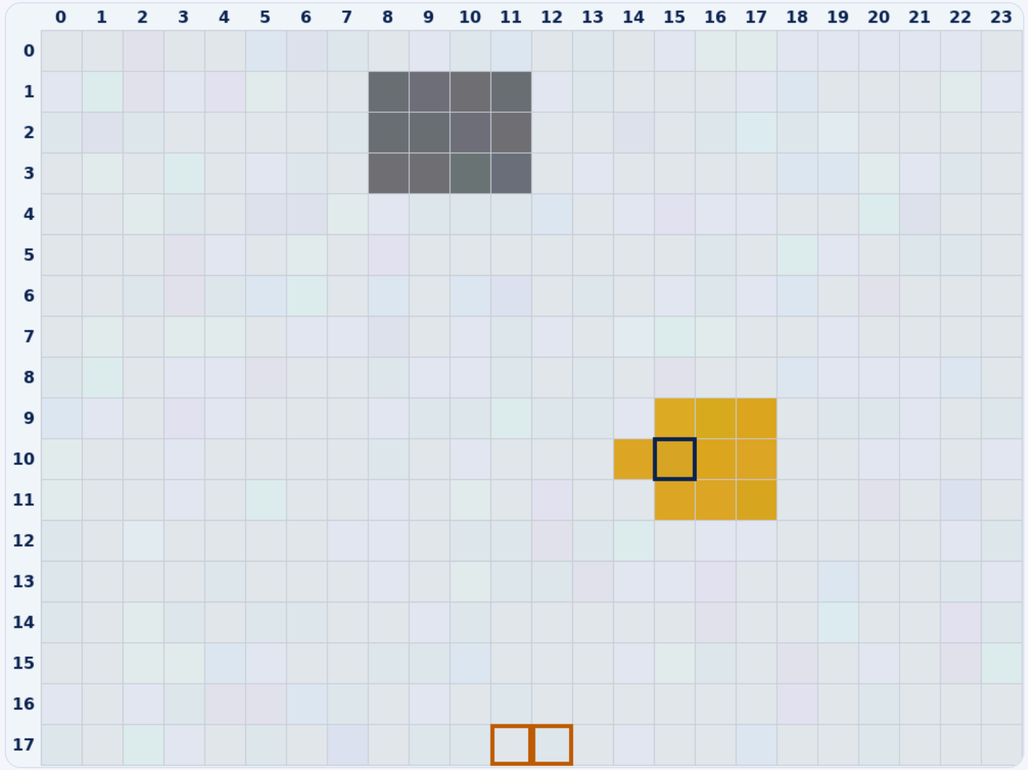
Camera[row][col]

CAM_W = 24 columns: 0 = left … 23 = right.

CAM_H = 18 rows: 0 = far … 17 = just in front.

Camera[row][col] is one pixel: a list of three numbers, **[R, G, B]**.

Pickup point: row **17**, columns **11 and 12**.



THE CAMERA · READING ONE PIXEL

One pixel, three numbers

```
int r = Camera[10][15][0];  // 218  red
int g = Camera[10][15][1];  // 169  green
int b = Camera[10][15][2];  // 35   blue
```

Row first, then column, then the colour: **[0]** red, **[1]** green, **[2]** blue.

Floor at the pickup point: **229, 233, 235**. Lots of everything = almost white.

READING A COLOUR · RGB

Each colour is three numbers, 0 to 255

Surface	Colour	R	G	B	Seen as
Floor		223–229	229–235	235–241	bright in all three
Iron		21–27	22–27	23–29	dark in all three
Silver		107–113	109–115	115–121	middle grey
Gold		217–223	165–171	31–37	R high, G middle, B low
Water pond		0–3	167–173	237–243	R low, G and B high
Deposit area		27–33	157–163	67–73	G high
Warning band		247–253	172–178	172–178	light red

Real values, sampled from the camera in the simulator. Each number changes by about **±3** from pixel to pixel: that is sensor noise.

READING A COLOUR · A COLOUR TEST

Test a range, not one number

```
int gold(int y, int x) {
    int r = Camera[y][x][0]; int g = Camera[y][x][1];
    int b = Camera[y][x][2];
    return r > 195 && g > 135 && g < 200 && b < 80;
}
```

Measured gold: R **217–223** · G **165–171** · B **31–37**

Test: R **> 195** · G **135–200** · B **< 80** → room for noise and light

== 220 would almost never be true: the value moves with noise.

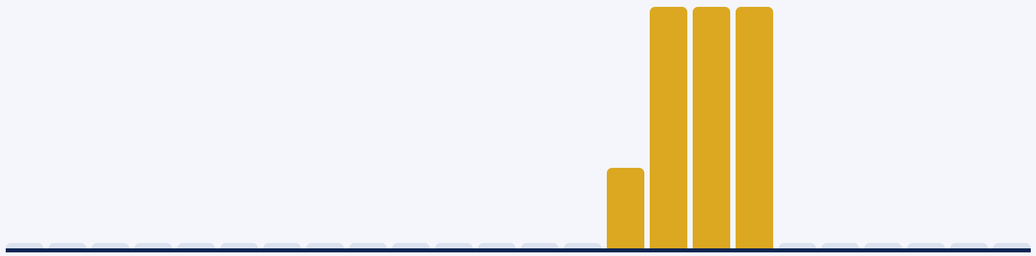
A **range** with a safety margin catches every gold pixel, also when the light changes.

The ranges must **not overlap** with other surfaces: no floor, iron or silver pixel passes this test.

Our first function: **gold(y, x)** answers yes or no for one pixel.

FROM PIXELS TO A DECISION

Where is the gold? Then steer



col 0 **centre 11.5** col 23

Gold pixels per column in this image: columns **14–17**.

Left of centre → turn left

Centre → drive on

Right of centre → turn right

```
float err = col - (CAM_W - 1) / 2.0; // 3.5
WheelLeft = 3 + err * 0.25;          // 3.875
WheelRight = 3 - err * 0.25;         // 2.125: right
```

Nearest gold pixel: row **11**, column **15**. The further from the centre, the harder the turn.

Collect only when **both** pickup pixels (row 17, columns 11 and 12) are gold.

PART 2

The mission: Session 14

Collect 3 gold objects with the camera

Session 14 · Gold hunt

Find gold in the Camera image and collect 3 GOLD objects. Leave iron and silver.

Each gold: +20. 3 gold: +100.

Gold 0/3

Camera tip: Gold pixels are about R > 195, G 135–200, B < 80. Turn towards the column with the most gold pixels, then drive until it is at the bottom centre.

Sessions | Restart session

SESSION 14 · GOLD HUNT

Gold only: leave iron and silver

Collect **3 gold** objects. Iron and silver are refused: **wrong metal**.

Each gold: **+20** · 3 gold: **+100** · total **160**.

Time limit **5:00**.

Medals · Bronze: finish · Silver: under 3:00 · Gold: under 1:45 and perfect.



THE FIELD

Six of each, one block

240 × 180 cm, walls all round, a block in the middle.

6 gold, 6 silver, 6 iron: we need any 3 gold.

The robot does not know where they are. It must **look**.

The camera sees only 30 cm ahead: explore, then steer to what you see.

PART 3

Step by step in C

A gold detector, a finder and a steering rule

STEP 1 · THE PLAN

Four rules, most important first

1. Both pickup pixels are gold → **Collect** (stop, red flashing LED, 3.2 s).
2. Wall or block close ahead (US_Front < 18) → **turn** for 0.4 s.
3. Gold somewhere in the image → **steer** towards the nearest gold pixel.
4. Nothing seen → **search**: drive in a wide curve.

Two helper functions do the camera work: **gold(y, x)** and **nearestGold()**.

STEP 2 · LINES 5–8

The colour test as a function

```
5  int gold(int y, int x) {                    // is this pixel gold?
6      int r = Camera[y][x][0]; int g = Camera[y][x][1]; int b = Camera[y][x][2];
7      return r > 195 && g > 135 && g < 200 && b < 80;
8  }
```

Takes a **row** and a **column**, answers true or false.

The ranges from Part 1: R > 195, G 135–200, B < 80.

Line 3, **int gy = -1;**, is a global variable: every function can read and change it.

STEP 3 · LINES 10–15

Find the nearest gold pixel

```
10  int nearestGold() {                          // column of the nearest gold, or -1
11      for (int y = CAM_H - 1; y >= 0; y--)     // bottom row = closest
12          for (int x = 0; x < CAM_W; x++)
13              if (gold(y, x)) { gy = y; return x; }
14      return -1;
15  }
```

Rows from the **bottom** up: the first hit is the **closest** gold.

return stops both loops at once and gives the column.

No gold anywhere → **-1**.

STEP 4 · LINES 17–27

Collect at the pickup point, turn at walls

```
17  void Game1() {
18      LED_1 = 0;
19      int mid = CAM_W / 2;
20      int col = nearestGold();
21      if (gold(CAM_H - 1, mid - 1) && gold(CAM_H - 1, mid)) {  // Collect
22          WheelLeft = 0; WheelRight = 0;
23          LED_1 = 2; LED_R = 255; LED_G = 0; LED_B = 0;
24          Duration = 3200;
25      } else if (US_Front < 18) {                              // Wall
26          WheelLeft = 3; WheelRight = -3;
27          Duration = 400;
```

Both pickup pixels gold: the object is under the pickup point.

LED_1 = 2, red 255, 0, 0, **Duration = 3200**: collect.

US_Front < 18: turn right for 0.4 s.

STEP 5 · LINES 28–36

Steer to the gold, or search

```
28      } else if (col >= 0) {                                  // SteerToGold
29          float err = col - (CAM_W - 1) / 2.0;
30          float speed = 3;
31          if (gy > CAM_H - 5) speed = 1.5;
32          WheelLeft = speed + err * 0.25; WheelRight = speed - err * 0.25;
33      } else {                                                // Search
34          WheelLeft = 3.5; WheelRight = 2.5;
35      }
36  }
```

err < 0: gold on the left. > 0: on the right.

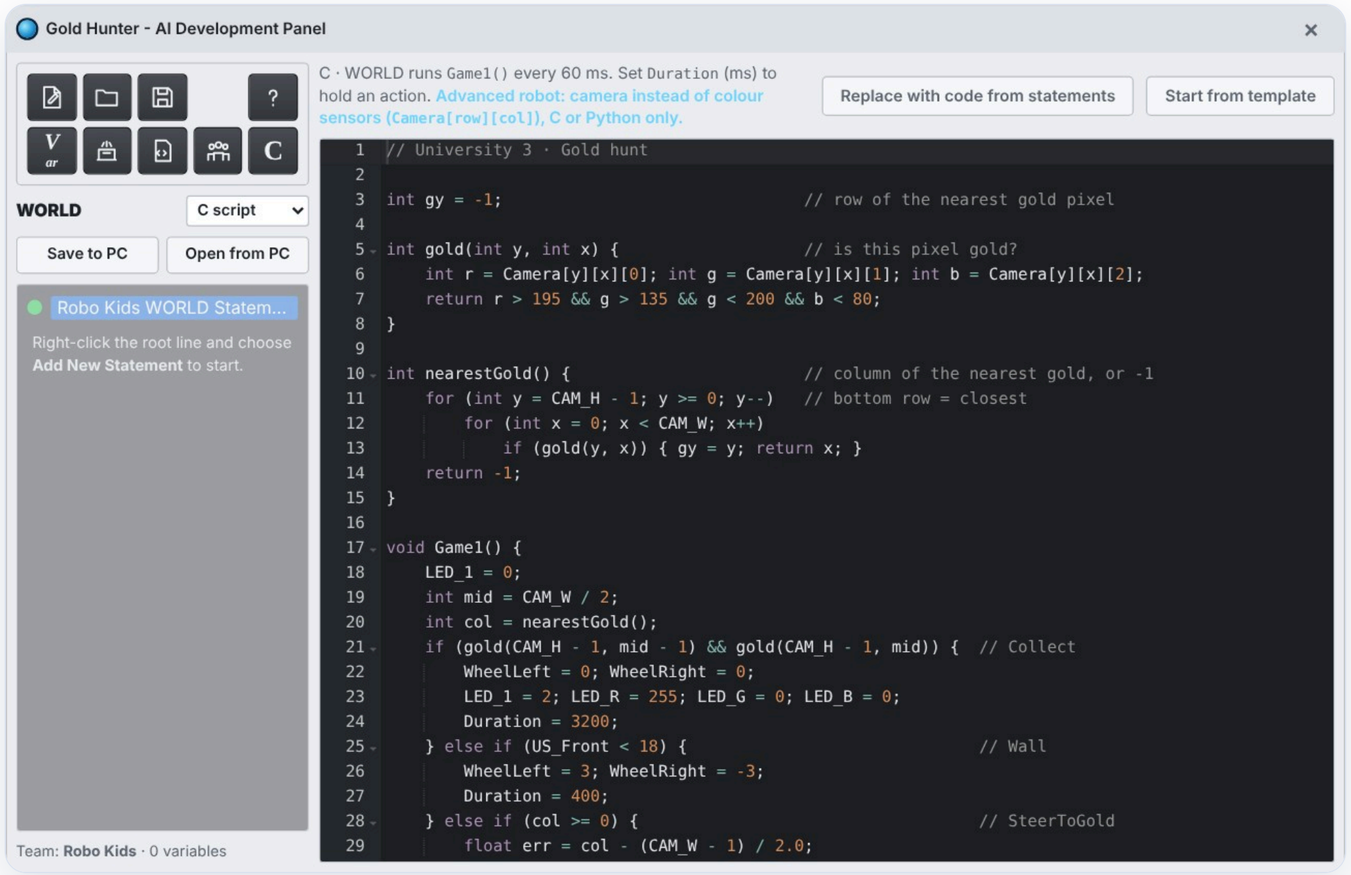
Gold in the last 4 rows → slow down to **1.5** so it does not slip past.

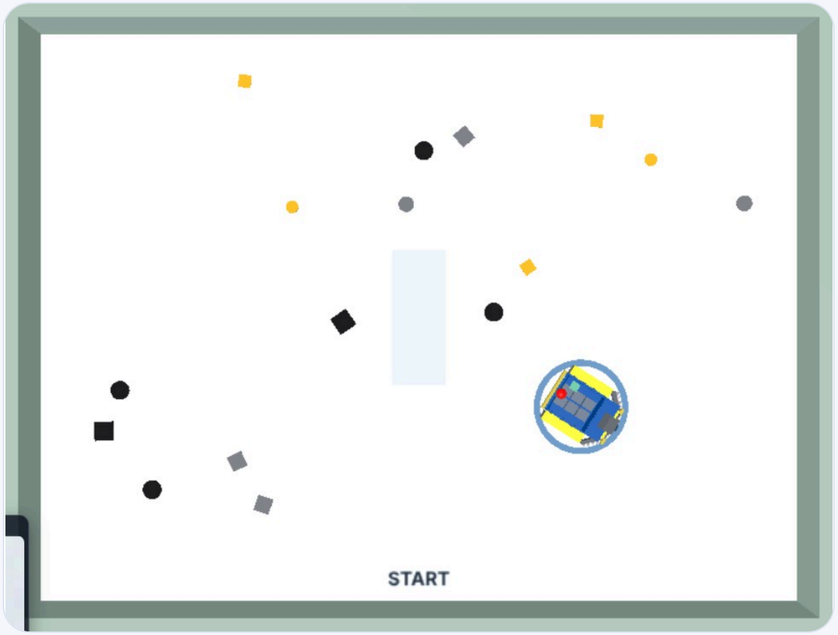
else: a wide curve, 3.5 / 2.5.

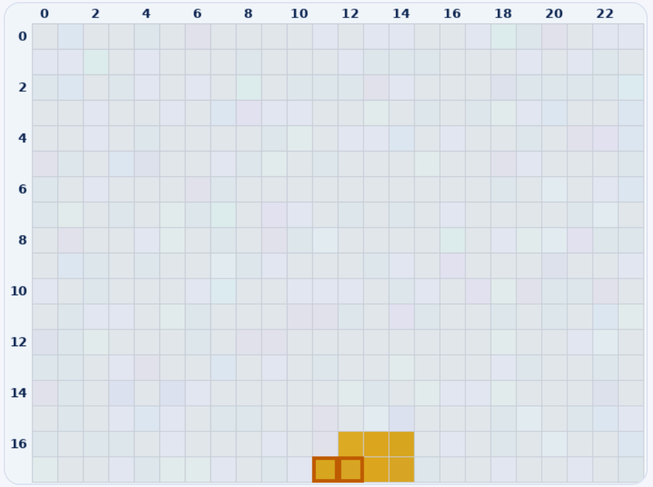
STEP 6 · BUILD AND LOAD

The whole program, built

1. Category **Advanced**, Learning, Session 14.
2. WORLD menu: **C script**.
3. Type the program, name it **Gold Hunter**.
4. **Build**, then **Load** on BLUE.







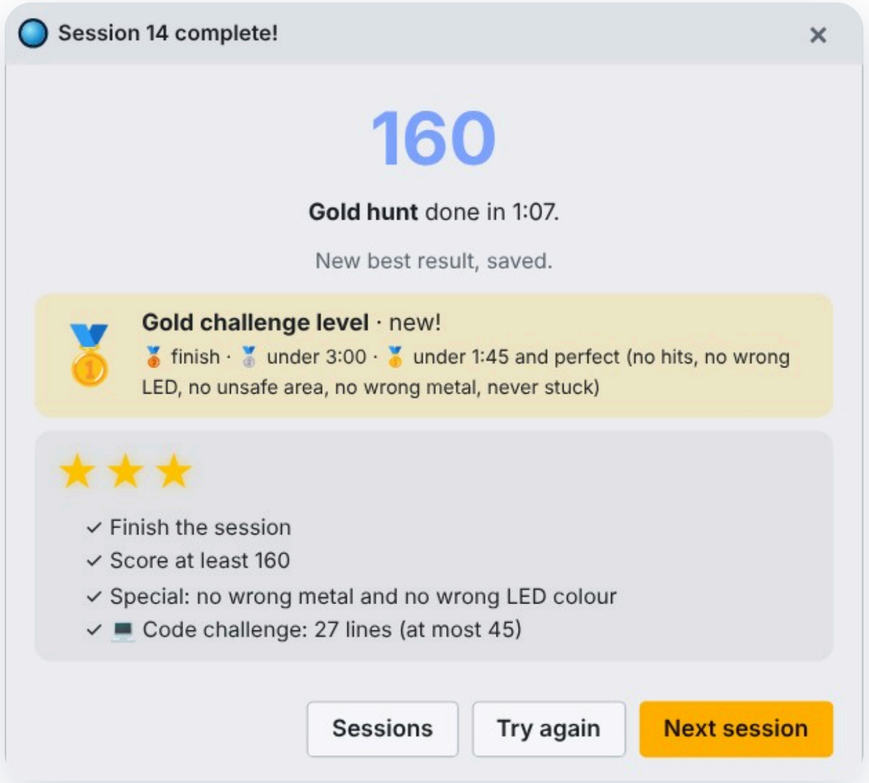
STEP 7 · RUN · 6.3 S

The first gold, at the pickup point

At **6.3 s** the gold sits on the pickup point: both pixels pass the test.

The robot stops and the red LED flashes for 3.2 s: **+20**.

Then it steers to the next gold it sees.



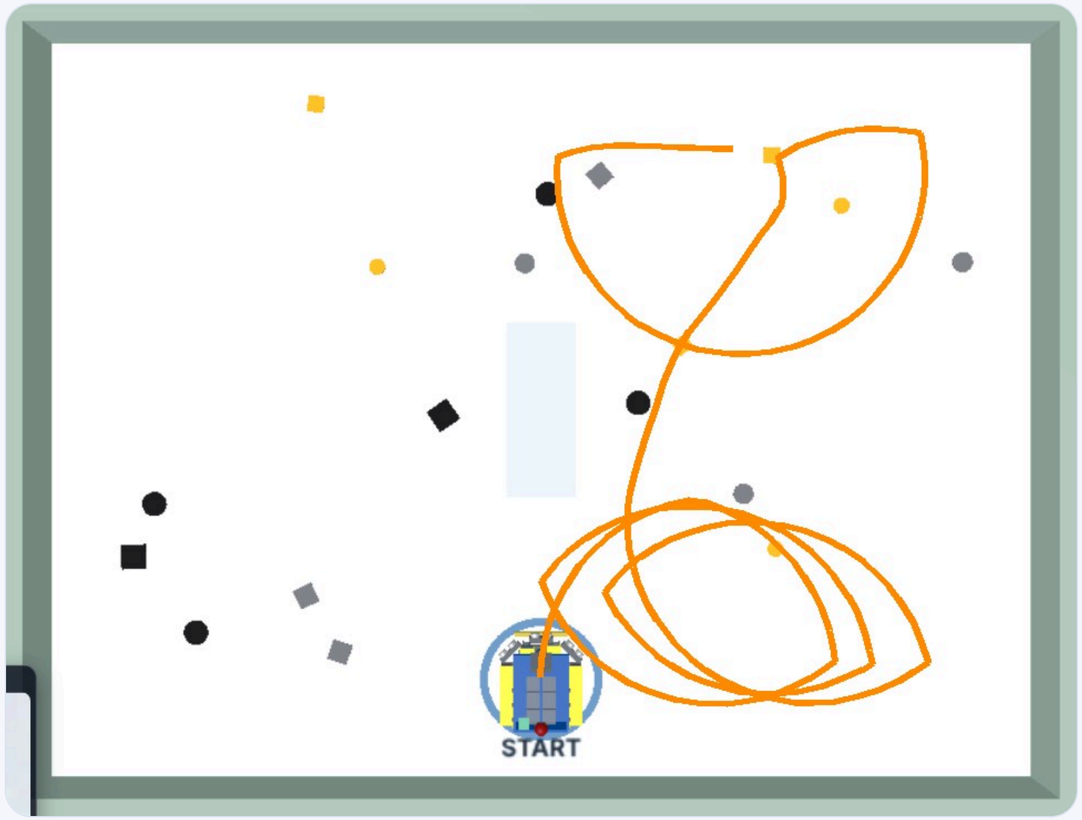
RESULT · TESTED

160 points in 1:07 · Gold

Medals · Bronze: finish · Silver: under 3:00 · Gold: under 1:45 and perfect.

Stars · finish · score 160 · no wrong metal and no wrong LED colour.

Code challenge · at most 45 lines. We used 27.



THE PATH · RECORDED

Explore, see, steer, collect

Wide curves while nothing is in view.

Short, straight runs as soon as gold appears in the camera.

Camera + colour test + steering = **targeted behaviour**.

DID NOT WORK?

Check first, then tune carefully

What you see	What to do
Drives over gold, never stops	Print the pixel: is it inside your ranges? Check row **CAM_H - 1**, columns **mid - 1** and **mid**
Stops on silver or iron	Your ranges are too wide: compare with the RGB table
Circles around a gold object	Turn gain too high, or no slow-down near the bottom rows
Gold slips under the robot	Slow down earlier: **gy > CAM_H - 5**
Build error with a line number	Check semicolons, { } and the three [ ][ ][ ] indexes

NEXT TIME

University 4 · Oil or Rocks

Black letters on the floor: the camera must tell an **O** from an **R** by its shape, not its colour.