PRESENTATION 7 · LEARNING WITH PYTHON

# One of each

Iron, silver and gold, and a robot that counts what it collects using variables.

**TheBotCup 2026 · AiBotCup Simulator**



RECAP · PRESENTATION 6

# Four branches found all the iron

**IronLeft**
dark on the left → Collect

**IronRight**
dark on the right → Collect

**Ahead**
wall close → turn left

**Forward**
otherwise drive

That robot collected **everything** it found. Today that is not good enough: it must **choose**.

PART 1

# The mission: Session 7

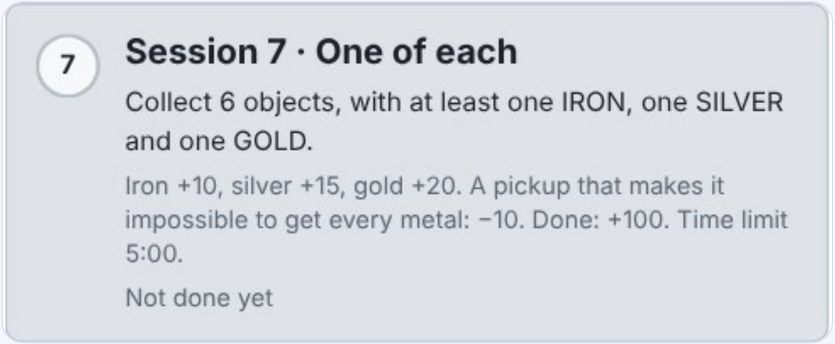
## Six objects, every metal at least once

SESSION 7 · ONE OF EACH

# Collect 6, with at least one of each metal

| What happens | Points |
|---|---|
| Iron · silver · gold | +10 · +15 · +20 |
| All 6 done, every metal | +100 |
| A pickup that makes it impossible | −10, refused |

Time limit **5:00**. The robot holds at most **6** objects.






THE FIELD

# Three metals, mixed up

- **7 iron** · black
- **6 silver** · grey
- **6 gold** · yellow

Same field as Session 6: block in the middle, START at the bottom.

COLOUR SENSORS · FROM PRESENTATION 6

# Tell the three metals apart

| Metal | R | G | B |
|---|---|---|---|
| **Iron** | 0 – 60 | 0 – 60 | 0 – 60 |
| **Silver** | 90 – 135 | 92 – 137 | 98 – 142 |
| **Gold** | 195 – 255 | 135 – 200 | 0 – 80 |
| Floor | about 226 | about 232 | about 238 |

These ranges come from the simulator’s own tip. Floor never fits any of them.

PART 2

# The main problem: counting

## How many do I have already?

THE PROBLEM

# Six places, and they can fill up wrong




Gold is still missing and only **one place** is left. If the next object is not gold, the mission becomes impossible: the simulator **refuses it, −10**.

The robot must know **how many of each** it already has.

WE TESTED IT · NO COUNTING

# “Collect everything” lost points

## What it did

Picked up 3 iron and 2 silver first. Then **one pickup was refused** because gold was still missing.

## Score

It finished, but with **170** instead of 180: a −10 for the refused pickup.

On another field, or with other luck, it could be worse. The robot needs a **memory**.

THE IDEA

# To count, the robot needs a number it can remember and change.

Sensors tell what is happening **now**. A counter remembers what happened **before**. In programming that is called a **variable**.

PART 3

# Variables

## The robot’s memory

WHAT IS A VARIABLE?

# A labelled box that holds one number

It has a **name** (Iron) and a **value** (0, 1, 2...).

The program can **read** the value and **change** it.

It **keeps** its value every 60 ms tick, until you change it.



HOW IT WORKS · THE COUNTER

# Read it in a condition, change it in an action

**READ · in a condition**

```
Iron < 2
```

True while Iron is 0 or 1.

**CHANGE · in a branch**

```
Iron = Iron + 1
```

“New Iron is old Iron plus one.”

**Iron:** 0 → 1 → 2 … then **Iron < 2** is false: no more iron.

MAKE VARIABLES IN PYTHON

# One line at the very top

```
# Session 7 · One of each

Iron = 0; Silver = 0; Gold = 0                    # our counters
```

**Outside** game1(): this line runs **once**, when the program is loaded.

**Start at 0**: no metal collected yet.

**Names**: letters, digits and _, starting with a letter. Not a sensor name.

USE VARIABLES IN A BRANCH

# and Iron < 2, then Iron = Iron + 1

```
20  def game1():
21      LED_1 = 0
22      if isIron(CSLeft_R, CSLeft_G, CSLeft_B) and Iron < 2:           # IronLeft
23          collect(); Iron = Iron + 1
24      elif isIron(CSRight_R, CSRight_G, CSRight_B) and Iron < 2:      # IronRight
25          collect(); Iron = Iron + 1
```

**Read**: **and Iron < 2**: the sensors say iron and we still want iron.

**Change**: **Iron = Iron + 1** right after **collect()**.

OUR RULE

# Collect a metal only while its counter is below 2

**2 + 2 + 2 = 6**

Six objects can only happen as two of each, so there is **always one of each** and no pickup is ever refused.

PART 4

# Step by step in Python

## Three counters, eight branches

STEP 1 · THE PLAN

# Three counters, then 8 branches

| Branch | Condition | Then | Counter |
|---|---|---|---|
| **1 IronLeft** | isIron(left) and Iron < 2 | collect() | Iron + 1 |
| **2 IronRight** | isIron(right) and Iron < 2 | collect() | Iron + 1 |
| **3 SilverLeft** | isSilver(left) and Silver < 2 | collect() | Silver + 1 |
| **4 SilverRight** | isSilver(right) and Silver < 2 | collect() | Silver + 1 |
| **5 GoldLeft** | isGold(left) and Gold < 2 | collect() | Gold + 1 |
| **6 GoldRight** | isGold(right) and Gold < 2 | collect() | Gold + 1 |
| **7 Ahead** | US_Front <= 20 | −2 / 2, 700 ms | — |
| **8 Forward** | else | 4.5 / 4.5 | — |

STEP 2 · LINES 5–12

# Three functions that answer yes or no

```
5  def isIron(R, G, B):
6      return R <= 60 and G <= 60 and B <= 60
7
8  def isSilver(R, G, B):
9      return 90 <= R <= 135 and 92 <= G <= 137 and 98 <= B <= 142
10
11 def isGold(R, G, B):
12     return R >= 195 and 135 <= G <= 200 and B <= 80
```

**return** sends the answer back: true or false.

**R, G, B**: the three colour numbers we pass in, left or right.

**Ranges** from the colour table: iron dark, silver grey, gold yellow.

STEP 3 · LINES 14–18

# collect(): the same as Presentation 6

```
14  def collect():
15      WheelLeft = 0; WheelRight = 0
16      LED_1 = 2                                  # flashing
17      LED_R = 255; LED_G = 0; LED_B = 0          # red
18      Duration = 3200                            # hold 3.2 seconds
```

**Wheels 0**: stop on the metal.

**LED_1 = 2**, red: the collect signal.

**Duration = 3200**: hold 3.2 s.

STEP 4 · LINES 20–25 AND 26–33

# Six collect branches, one counter each

```
26      elif isSilver(CSLeft_R, CSLeft_G, CSLeft_B) and Silver < 2:     # SilverLeft
27          collect(); Silver = Silver + 1
28      elif isSilver(CSRight_R, CSRight_G, CSRight_B) and Silver < 2:  # SilverRight
29          collect(); Silver = Silver + 1
30      elif isGold(CSLeft_R, CSLeft_G, CSLeft_B) and Gold < 2:         # GoldLeft
31          collect(); Gold = Gold + 1
32      elif isGold(CSRight_R, CSRight_G, CSRight_B) and Gold < 2:      # GoldRight
33          collect(); Gold = Gold + 1
```

**IronLeft, IronRight** (lines 20–25): same counter, Iron.

**Silver** and **Gold**: same pattern, own counter.

**LED_1 = 0** first: the LED is off unless we collect.

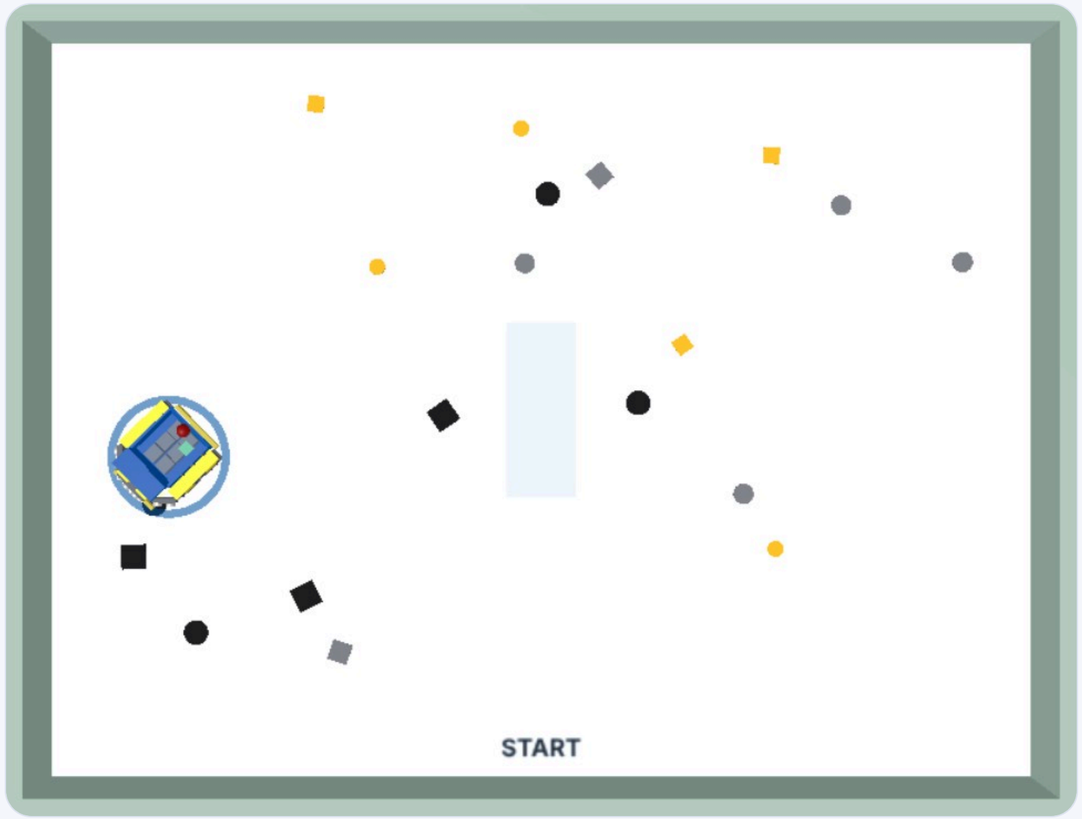
STEP 5 · LINES 34–38

# Ahead and Forward, as in Presentation 6

```
34      elif US_Front <= 20:                                    # Ahead
35          WheelLeft = -2; WheelRight = 2
36          Duration = 700
37      else:                                                   # Forward
38          WheelLeft = 4.5; WheelRight = 4.5
```

**Collect branches stay on top**.

A metal the robot no longer wants matches no branch above, so **Forward drives over it**.




STEP 7 · RUN

# The first catch at 7.3 s

CSLeft reads **21, 25, 28**: isIron is true, and Iron is still 0.

**IronLeft** runs: collect(), then **Iron = 1**.

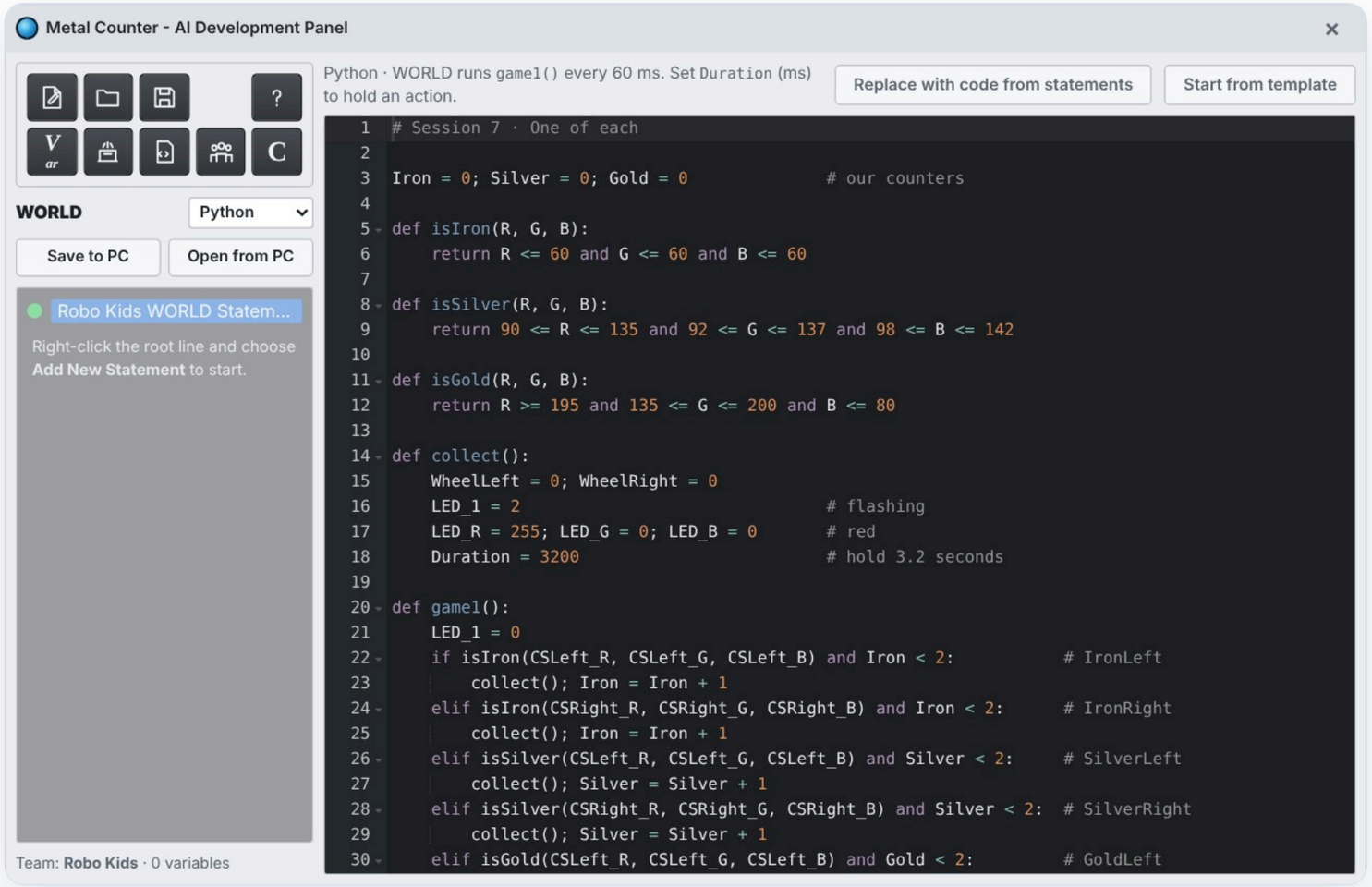
At the end the counters read **2, 2, 2**.

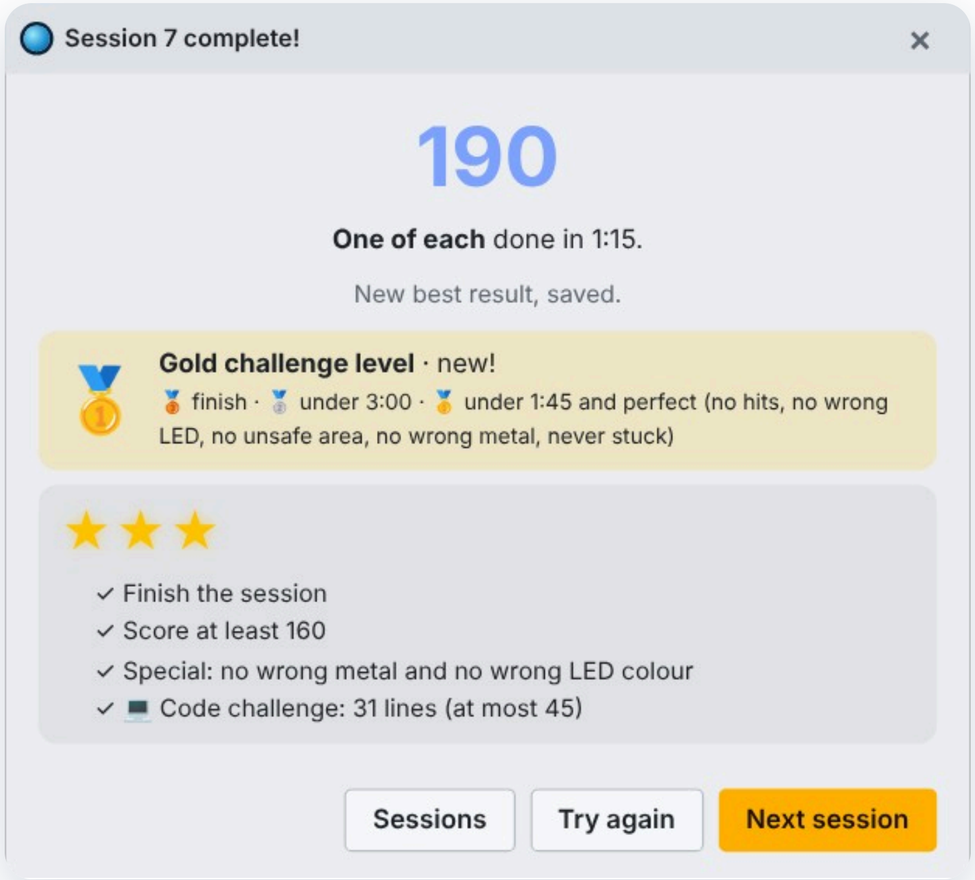
No pickup is refused: the counters stop at 2.

STEP 6 · BUILD AND LOAD

# The whole program, built

1. WORLD menu: **Python**.
2. Type the program, name it **Metal Counter**.
3. Press **Build**, then **OK**.
4. Press **Load** on the BLUE side.





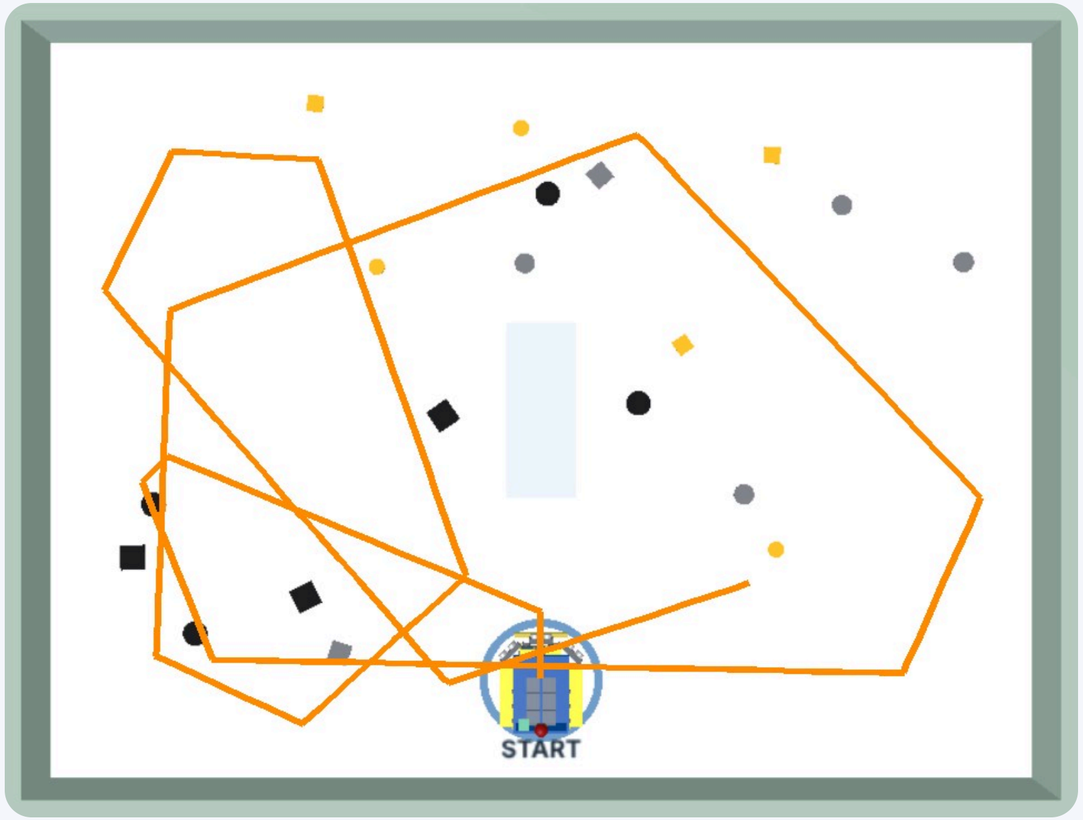

RESULT · TESTED

# 190 points in 1:15 · Gold

**Score** · 2×10 + 2×15 + 2×20 = 90, plus 100 = 190.

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

**Stars** · no wrong metal · code challenge: 31 of at most 45 lines.




THE PATH · RECORDED

# Two of each, then done

Order in our run: iron, silver, silver, iron, gold, gold.

Once a counter reaches 2, that metal is ignored.

The counters let the robot **choose**.

DID NOT WORK?

# Check first, then tune carefully

| What you see | What to do |
|---|---|
| Build error with a line number | Colon : after def / if / elif / else, and indentation |
| Counter never changes | **Iron = Iron + 1** inside the branch, after collect() |
| Counter is always 0 again | The counters must be at the top, **outside** game1() |
| Collects a third of one metal | The **and Iron < 2** part is missing |
| Never collects silver or gold | Check the ranges in isSilver and isGold |
| Old program still on the robot | Build again, then Load on BLUE |

NEXT TIME

# Presentation 8 · Two of each

Exactly two of every metal, and the simulator’s **own built-in counters**.