**CATCH THE FALLING BOTTLES**

**Aim:** To develop a program that implements the Hill-Climbing algorithm to control a Bin in a game, enabling it to catch the falling Bottles by minimizing the horizontal distance between the Bin and the Bottle.

**Program:**

import time

from pystage.en import Sprite, Stage

stage = Stage()

stage.add\_backdrop('Background')

stage.create\_variable('Bottle Collected')

stage.create\_variable('Bottle Fallen')

stage.show\_variable("Bottle Collected")

stage.set\_monitor\_position("Bottle Collected", -220, -150)

stage.show\_variable("Bottle Fallen")

stage.set\_monitor\_position("Bottle Fallen", 100, -150)

# Create and initialize sprite 'bin'

bin\_sprite = stage.add\_a\_sprite(None)

bin\_sprite.set\_name("Bin")

bin\_sprite.set\_x(122)

bin\_sprite.set\_y(-91.35)

bin\_sprite.go\_to\_back\_layer()

bin\_sprite.go\_forward(3)

bin\_sprite.set\_size\_to(50)

bin\_sprite.add\_costume('bin', center\_x=30, center\_y=15)

# Create and initialize sprite 'bottle'

bottle = stage.add\_a\_sprite(None)

bottle.set\_name("Bottle")

bottle.set\_x(-133)

bottle.set\_y(172)

bottle.go\_to\_back\_layer()

bottle.go\_forward(2)

bottle.set\_size\_to(80)

bottle.add\_costume('bottle', center\_x=31, center\_y=31)

bottle.add\_sound('chomp')

bottle.add\_sound('collect')

# Scratch Blocks for 'bin'

def when\_program\_starts\_1(self):

while True:

# Hill-Climbing logic

current\_distance = abs(self.x\_position() - bottle.x\_position())

neighbor\_left\_distance = abs((self.x\_position() - 10) - bottle.x\_position())

neighbor\_right\_distance = abs((self.x\_position() + 10) - bottle.x\_position())

if neighbor\_left\_distance < current\_distance:

self.change\_x\_by(-10.0) # Move left if it reduces the distance

elif neighbor\_right\_distance < current\_distance:

self.change\_x\_by(10.0) # Move right if it reduces the distance

time.sleep(0.01)

bin\_sprite.when\_program\_starts(when\_program\_starts\_1)

# Scratch Blocks for 'bottle'

def when\_program\_starts\_2(self):

self.set\_variable("Bottle Collected", 0)

self.set\_variable("Bottle Fallen", 0)

while True:

self.change\_y\_by(-8.0)

if (self.y\_position() < -170):

self.go\_to\_random\_position()

self.set\_y(180.0)

if (self.get\_variable("Bottle Fallen") == 10):

self.say\_for\_seconds("You Lost", 2.0)

self.stop\_all()

if (self.get\_variable("Bottle Collected") == 25):

self.say\_for\_seconds("You Win", 2.0)

self.stop\_all()

if self.touching(bin\_sprite) and self.y\_position() == -52.0:

self.change\_variable\_by("Bottle Collected", 1.0)

self.play\_sound\_until\_done("collect")

self.go\_to\_random\_position()

self.set\_y(180.0)

else:

if (self.y\_position() < -160):

self.change\_variable\_by("Bottle Fallen", 1.0)

self.play\_sound\_until\_done("chomp")

self.go\_to\_random\_position()

self.set\_y(180.0)

time.sleep(0.01)

bottle.when\_program\_starts(when\_program\_starts\_2)

stage.play()

**Initial State:**

The initial state includes the positions and statuses of the Bin and Bottle sprites.

Bottle Collected & Bottle Fallen variables are both set to 0. The Bin is positioned at (122, -91.35).

The Bottle starts at a random position at the top of the screen and falls downwards.

Variables initialized: Bottle Collected = 0 and Bottle Fallen = 0

**Goal State:**

Collect 25 Bottles without letting 10 Bottles fall.

**Goal Test:**

The game checks two conditions: If Bottle Collected equals 25 then its the win condition or else

if Bottle Fallen equals 10 then its the lose condition.

**Actions:**

***Bin:*** Move left or right to minimize the distance between itself and the Bottle based on a Hill-Climbing algorithm.

***Bottle:*** Fall vertically down by changing its y-position. Reset to a new random position and y-position when it falls below y = -170. If it touches the Bin, it increases the Bottle Collected variable. If it falls beyond y = -160 without being caught, it increases the Bottle Fallen variable.

**Transition Model:**

**Bin:** The Bin’s movement is determined by a Hill-Climbing strategy, where it moves left or right to minimize its distance from the Bottle.

**Bottle:** The Bottle moves downward and either gets caught by the Bin or falls to the ground. Depending on these conditions, it will either increase the Bottle Collected or Bottle Fallen variable and reset its position.

**Path Cost:** A higher number of Bottles falling (missed by the Bin) could be seen as a higher path cost, leading to a loss if it reaches 10.

**Hill Climbing Algorithm Implementation**

**Objective:**

The objective of the Hill Climbing algorithm in this scenario is to minimize the horizontal distance between the "Bin" and the falling "Bottle". The "Bin" needs to move left or right to better align with the position of the falling Bottle.

**Algorithm Steps:**

1. **Calculate Current Distance:** The distance between the "Bin" and the "Bottle" is calculated using their x-coordinates. This is represented by current distance.
2. **Calculate Neighbour Distances:** The algorithm considers two potential moves for the "Bin":

Move Left: The distance if the "Bin" moves 10 units to the left:

Move Right: The distance if the "Bin" moves 10 units to the right:

1. **Compare Distances:** The algorithm compares these neighbour distances to the current distance:

If the distance to the Bottle decreases by moving left, then the "Bin" moves **left**

If the distance to the Bottle decreases by moving right, then the "Bin" moves **right**

1. **Repeat and Sleep:** This process repeats indefinitely in a while loop, with a small delay between iterations to control the update rate:

**How It Works in the Game:**

***Movement Adjustments:*** The "Bin" continuously evaluates whether moving left or right brings it closer to the falling "Bottle". If either direction reduces the distance, the "Bin" moves in that direction.

***Greedy Approach:*** The algorithm is greedy because it makes a local decision at each step, choosing the direction that immediately minimizes the distance to the Bottle. It does not consider global optimality but rather immediate improvement.

**Agent Type:**

Bin: Reactive agent using Hill-Climbing logic to catch Bottles.

Bottle: Simple reactive agent that falls and interacts with the environment (Bin, ground).

**Performance Measure:** The number of Bottles collected (Bottle Collected). The game ends when 25 Bottles are collected (win) or 10 Bottles are missed (lose).

**Environment:** A 2D grid where the Bin and Bottle exist. The environment is dynamic as the Bottle continuously falls and resets its position.

**Actuators:**

Bin: Controls its horizontal position (x-axis).

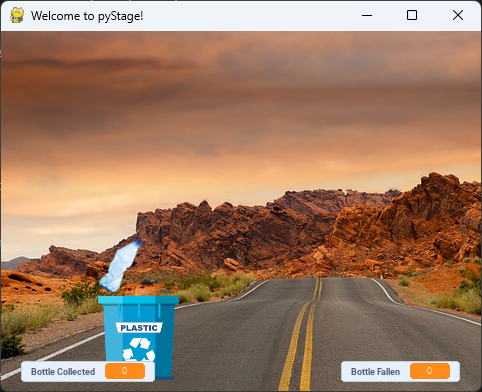
Bottle: Controls its vertical position (y-axis) and can reset to a new random position.

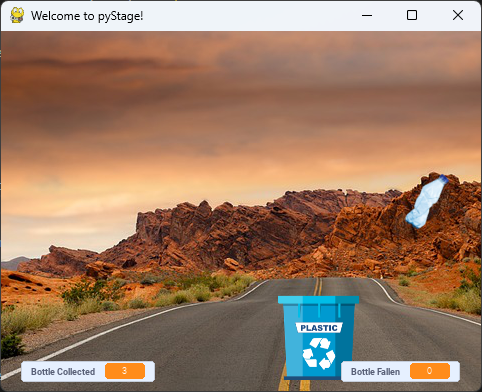
**Sensors:**

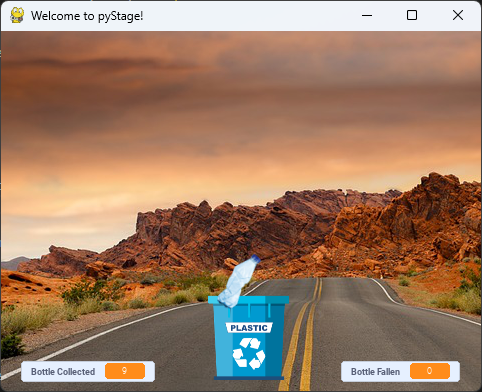
Bin: Detects the Bottle's x-position to decide movement direction.

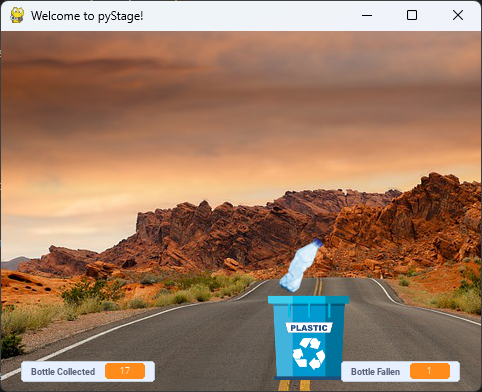
Bottle: Detects whether it is touching the Bin or if it has fallen below a certain y-coordinate to trigger position reset and variable updates.

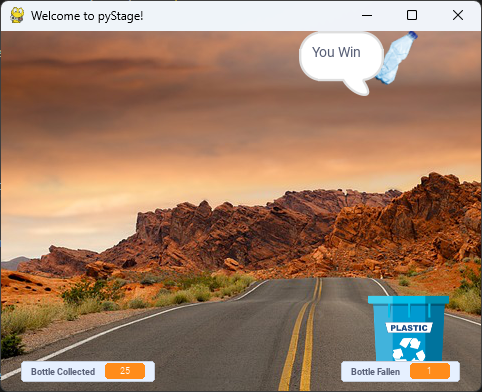
**Output:**

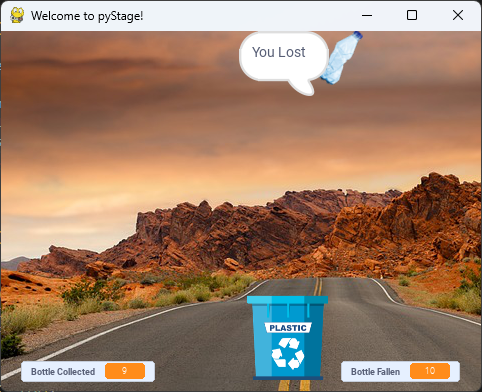












**Conclusion:**

The program to implement the Hill-Climbing algorithm to control a Bin in a game, enabling it to catch the falling Bottles was developed and executed Successfully.