Student Name: 33001A VR Skills Challenge

Assignment:

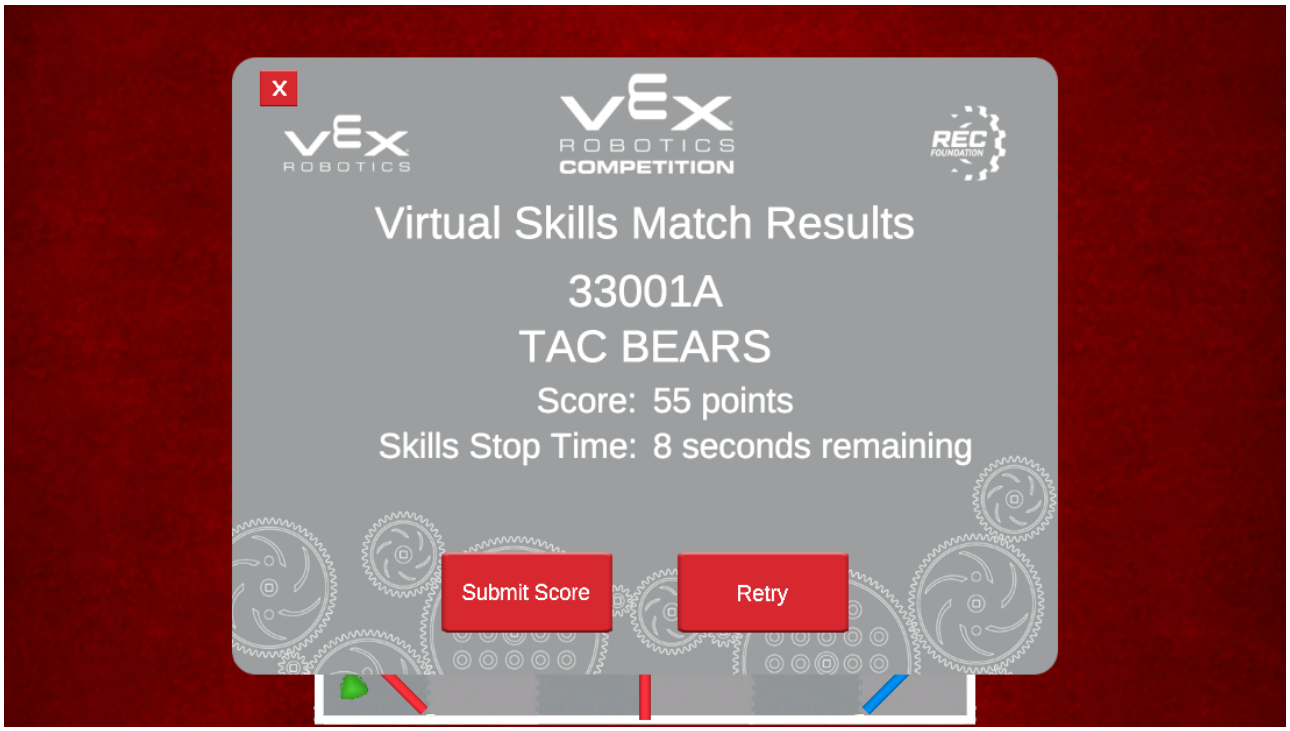
Notes:

Playground: VRC Virtual Skills - Over Under

Project Name: VEXcode_Project

Project Type: Python

Date: Wed Jan 31 2024



```
1      #region VEXcode Generated Robot Configuration
2      import math
3      import random
4      from vexcode_vrc import *
5      from vexcode_vrc.events import get_Task_func
6
7      # Brain should be defined by default
8      brain=Brain()
9
10     drivetrain = Drivetrain("drivetrain", 0)
11     arm_motor = Motor("ArmMotor", 3)
12     rotation = Rotation("Rotation", 7)
13     intake_motor = Motor("IntakeMotor", 8)
14     optical = Optical("Optical", 11)
15     gps = GPS("GPS", 20)
16
17     #endregion VEXcode Generated Robot Configuration
18     import math
19     import random
20     from vexcode_vrc import *
21     from vexcode_vrc.events import get_Task_func
22
23     brain=Brain()
24
25     drivetrain = Drivetrain("drivetrain", 0)
26     arm_motor = Motor("ArmMotor", 3)
27     rotation = Rotation("Rotation", 7)
28     intake_motor = Motor("IntakeMotor", 8)
29     optical = Optical("Optical", 11)
30     gps = GPS("GPS", 20)
31
32     #endregion VEXcode Generated Robot Configuration
33     # ----------------------------------------
34     #
35     #   Project:      VEXcode Project
36     #   Author:       33001A
37     #   Created:
38     #   Description:  VEXcode VR Python Project
39     #
40     # ----------------------------------------
41
42
43     #PID stands for Proportional Integral Derivative Controller
44     #It is a control mechanism to minimize uncertainty and error in movement
45     #The proportional term decides how fast the controller is going to approach the de
  sired value
46     #The integral term gathers the cumulative error and adds it to the output to accou
  nt for the residual error
47     #The derivative term estimates how the error is going to change in the future and
  adds it to the controller by getting the rate of change
48     #Ultimately, it creates a control mechanism that approaches a value more smoothly
  and correctly
49
```

```
50      #Pid controller class initialization
51
52      class pidController:
53          #Constructor
54          def __init__(self, kP, kI, kD, dt):
55              #Proportional constant
56              self.kP = kP
57              #Integral constant
58              self.kI = kI
59              #Derivative constant
60              self.kD = kD
61              #Time difference between each pid update
62              self.dt = dt
63              #Integral term (since we want to preserve the value of the
64              #integral after each PID method call due to the integral being
65              #cumulative, we declare it as a class attribute unlike the proportional an
    d derivative terms)
66              self.integral = 0
67              #Last error value in the last call of the PID method
68              self.last_error = 0
69              #Initial velocity is set at 100%
70              self.vel = 100
71
72          #The main PID method
73          def pid(self, desired_velocity):
74              #finding the error value by subtracting current velocity from the desired
    velocity
75              error = desired_velocity - drivetrain.velocity(PERCENT)
76
77              #Multiplying the error by the proportional constant to decide how aggressi
    vely
78              #the controller is going to approach the error
79              proportional_error = error * self.kP
80
81              #Finding the integral of the error by multiplying the error with dt to fin
    d the
82              #cumulative error (the area between the desired velocity and pid velocity
    graph)
83              self.integral += error * self.dt
84
85              #Finding the rate of change in the error to find the derivative term
86              derivative = (error - self.last_error)/self.dt
87
88              #Saving the last error
89              self.last_error = error
90
91              #Returning the addition of all terms multiplied by constants and the curre
    nt velocity to output to the drivetrain
92              return drivetrain.velocity(PERCENT) + (proportional_error) + (self.integra
    l * self.kI) + (derivative * self.kD)
93
94          #PID resetting method for each new movement
95          def reset_pid(self):
96              self.integral = 0
```

```
97              self.last_error = 0
98
99      #Initializing PID
100     #The constants were tuned by trial and error to find the best values without oscil
  lation
101     pid = pidController(0.5, 0.3, 0.025, 0.1)
102
103     #size of a field tile in inches
104     tile_size = 24
105
106     #Implementing basic odomentry by storing coordinates (in inches),
107     #with initial value set according to starting location H (position (0, 0) is the c
  enter of the field)
108     position_x = tile_size * 1.5
109     position_y = tile_size * -2.5
110
111     #Turns the robot to the direction of (x, y), where x and y are measured in tiles
112     def turn_to(x, y):
113         #The difference between the robot's y position and the desired y position
114         dy = tile_size * y - position_y
115
116         #The difference between the robot's x position and the desired x position
117         dx = tile_size * x - position_x
118
119         #The angle (in degrees) between the desired direction and a line turned 90 deg
  rees clockwise to the heading
120         angle_to_turn = math.degrees(math.atan2(dy, dx))
121         #The function math.atan2(x, y) inverts angles correctly in all cases.
122         #For example, math.atan(-1 / 1) = math.atan(1 / -1) but the angle between (0,
  0)
123         #and (-1, 1) isn't equal to that between (0, 0) and (1, -1)
124         #A detailed explanation is given in https://en.wikipedia.org/wiki/Atan2
125
126         #The method turn_to_heading(theta, unit) turns the robot towards the heading t
  heta
127         #Since angle_to_turn is the angle between the desired direction and a line tur
  ned 90
128         #degrees clockwise to the heading, 90 - angle_to_turn is the heading correspon
  ding to (x, y)
129         drivetrain.turn_to_heading(90 - angle_to_turn, DEGREES)
130
131     #Turns the robot to (x, y) where x, y is measured in tiles using stored coordinate
  s, moves to (x, y) for a specified percentage and then updates coordinate values
132     def move_to(x, y):
133         #Globalizing position_x and position_y to globally update the values
134         global position_x
135         global position_y
136
137         #Turning to direction of (x, y)
138         turn_to(x, y)
139
140         #Resetting PID values before moving
141         pid.reset_pid()
142
```

```
143         #Length determined by the distance formula
144         drivetrain.drive_for(FORWARD, math.sqrt(pow(tile_size * x - position_x, 2) + p
  ow(tile_size * y - position_y, 2)), INCHES)
145 
146         #Updating stored position values
147         position_x = tile_size * x
148         position_y = tile_size * y
149 
150     #Moves the robot back x inches and updates the stored coordinate values
151     #Unifies both backwards and forward movement since negative x
152     def move_for(x):
153         #Globalizing position_x and position_y to globally update the values
154         global position_x
155         global position_y
156 
157         #Resetting PID values before moving
158         pid.reset_pid()
159 
160         #Moving backwards x inches
161         drivetrain.drive_for(FORWARD, x, INCHES)
162 
163         #Updating stored position values
164         position_x += x * math.sin(math.radians(drivetrain.heading(DEGREES)))
165         position_y += x * math.cos(math.radians(drivetrain.heading(DEGREES)))
166 
167     #Thread that continuously updates the drivetrain's velocity in the background usin
  g the PID
168     def pid_thread():
169         while True:
170             drivetrain.set_drive_velocity(pid.pid(pid.vel), PERCENT)
171             #Updates occur every dt seconds
172             wait(pid.dt, SECONDS)
173 
174     #Function that spins the intake motor forward until the optical sensor detects an
  object
175     def pick_up():
176         intake_motor.spin(FORWARD)
177         wait(100, MSEC)
178         #Spins the intake motor 15 miliseconds longer each loop until it detects an ob
  ject
179         while not optical.is_near_object():
180             wait(15, MSEC)
181             #To force an update in the optical sensor
182             #Without it, the optical sensor can get stuck into thinking there is alway
  s an object close by
183             move_for(0.001)
184 
185         #Stops intake motor
186         intake_motor.stop()
187 
188     #Function that spins the intake motor forward until the optical sensor detects an
  object
189     def drop():
190         intake_motor.spin(REVERSE)
```

```
191         wait(100, MSEC)
192         #Spins the intake motor 10 miliseconds longer each loop until it detects an ob
  ject
193         while optical.is_near_object():
194             wait(10, MSEC)
195 
196         #Stops intake motor
197         intake_motor.stop()
198 
199 
200 
201     #Main movement method
202     #Throughout the whole main method keep in mind that move_to both turns and moves t
  owards the desired location, the turn_to functions are mostly to drop and pick up triba
  lls
203     def main():
204         #Setting the drivetrain turn velocity to 100%
205         drivetrain.set_turn_velocity(100, PERCENT)
206 
207         #Setting the arm_motor velocity to max
208         arm_motor.set_velocity(100, PERCENT)
209 
210         #Extending the arm
211         arm_motor.spin_for(FORWARD, 5, TURNS)
212 
213         #Moving forward for a small amount to not touch the back wall before turning
214         move_for(2)
215 
216         #Moving to the front of the goal to drop preload
217         move_to(2, -1.5)
218 
219         #Preload drop
220         drop()
221 
222         #Backpedal to not hit goalpost
223         move_for(-12)
224 
225         #Go near the second triball
226         move_to(2.4, -2.4)
227 
228         #Turn and pick up the second triball
229         turn_to(3, -3)
230         pick_up()
231 
232         #Go near the goal
233         move_for(-12)
234         move_to(2.25, -1.5)
235 
236         #2nd triball drop
237         drop()
238 
239         #Backpedaling to not hit goalpost
240         move_for(-6)
```

```
241
242         #Go to and pick up 3rd triball
243         move_to(1.5, -2.65)
244         move_to(0.5, -2.65)
245         pick_up()
246
247         #Going back to the goal
248         move_for(-20)
249
250         #Repositioning backwards to prevent throwing off stored position values by con
  tacting the walls
251         drivetrain.turn_for(LEFT, 30, DEGREES)
252         move_for(-12)
253         move_to(1.5, -1.20)
254         drivetrain.turn_to_heading(60, DEGREES)
255
256         #3rd triball drop
257         drop()
258
259         #Backpedal to not hit goalpost
260         move_for(-6)
261
262         #Moving to the 4th triball and going back to the goal
263         move_to(0.4, -1.75)
264         turn_to(-0.25, -1.9)
265         pick_up()
266         drivetrain.turn_for(LEFT, 10, DEGREES)
267         move_for(-16)
268         move_to(1.65, -0.85)
269
270         #4th triball drop
271         drop()
272
273         #Maneuvering to the 5th triball and going back to the goal
274         move_for(-12)
275         move_to(0.4, -1.25)
276         turn_to(-0.5, -1.25)
277         pick_up()
278         move_for(-12)
279         move_to(1.5, -0.5)
280
281         #5th triball drop
282         drop()
283
284         #6th triball pick up
285         move_to(0.4, -0.25)
286         turn_to(-0.4, -0.25)
287         pick_up()
288         move_for(-12)
289         move_to(1.5, 0)
290         turn_to(2, 0)
291
292         #6th triball drop
```

```
293         drop()
294 
295         #7th triball pick up
296         move_to(0.4, 0.75)
297         turn_to(-0.2, 0.75)
298         pick_up()
299         move_for(-12)
300         move_to(1.5, 1)
301         turn_to(3, 0)
302 
303         #7th triball drop
304         drop()
305 
306         #8th triball pick up
307         move_to(0.4, 1.35)
308         pick_up()
309         move_for(-16)
310         move_to(1.5, 0.5)
311 
312         #8th triball drop
313         drop()
314 
315         #intermediate step to not ram into wall
316         move_to(1.5, 1.5)
317         move_to(2.45, 2.3)
318         drivetrain.turn_for(RIGHT, 10, DEGREES)
319 
320         #9th triball pick up
321         pick_up()
322         move_for(-12)
323         move_to(2.3, 1.5)
324 
325         #9th triball drop
326         drop()
327 
328         #10th triball pick up
329         move_for(-12)
330         move_to(0.75, 2.5)
331         turn_to(0, 2.5)
332         pick_up()
333         move_for(-20)
334         drivetrain.turn_for(RIGHT, 45, DEGREES)
335         move_for(-6)
336         move_to(2.5, 1.35)
337 
338         #10th triball drop
339         drop()
340 
341         #11th triball pick up
342         move_for(-30)
343         move_to(-1.5, 2.5)
344 
345         #Slowing down to prevent ramming into the triball
```

```
346         pid.vel = 70
347 
348         #Using intake_motor.spin(FORWARD) because using pick_up() doesn't work while m
  oving
349         intake_motor.spin(FORWARD)
350         move_to(-2.25, 1.5)
351         drivetrain.turn_to_heading(180, DEGREES)
352 
353         #11th triball drop
354         drop()
355 
356     #running both threads
357     vr_thread(pid_thread)
358     vr_thread(main)
359 
```