2024 VEX CODE VR SKILLS CHALLENGE

ELEMENTARY SCHOOL DIVISION

Team 15A Crescent Crushers

Team Members:

Noah

Aliya

Krish

Final Score: 83 points

Time Remaining: 2 seconds

Crescent Elementary School **|** Anaheim CA **|** January 2024

```python
# region VEXcode Generated Robot Configuration
import math
import random
from vexcode_viqc import *

# Brain should be defined by default
brain = Brain()

drivetrain = Drivetrain("drivetrain", 0)
intake_bumper = Bumper("IntakeBumper", 3)
front_optical = Optical("FrontOptical", 4)
intake_motor_group = Motor("IntakeMotorGroup", 5)
arm_motor_group = Motor("ArmMotorGroup", 6)
front_distance = Distance("FrontDistance", 9)

# endregion VEXcode Generated Robot Configuration
# ------------------------------------------
#
#   Project:      VEXcode Autonomous VR
#   Author:       Team 15A Crescent Crushers
#   Created:      12-29-2023
#   Description:  VEXcode VR Python Project, https://vr.vex.com/
#   File Name:    15A_Crescent_Crushers.vrpython
#   Notes/Credit: Thanks to VEX for some of the good ideas and concepts came from the VEX activity web site
:
#   education.vex.com/stemlabs//cs/vr-activity-labs/viqrc-virtual-skills-full-volume/pick-it-score-it
# ------------------------------------------

# Constants
# Numbers of degrees to spin the intake to pickup block
INTAKE_ROTATE = 90
# Number of degrees to spin the intake to score block in goal
OUTTAKE_ROTATE = 130
# Number of degrees to raise arm to score block in goal
ARM_MOTOR_OUTTAKE_LEVEL = 315
# Number of degrees to raise arm to "pluck" a block from a flower
ARM_MOTOR_PLUCK_LEVEL = 200
# Number of degrees to raise arm to "scoop" up blocks
ARM_MOTOR_INTAKE_LEVEL = 10
# Length of bot from the center to the end of intake
BOT_LENGTH = 264
# Length of bot from the center to the end of intake
BOT_LENGTH_HALF = BOT_LENGTH * .5
# The additional distance bot travels to goal to score blocks in millimeters
GOAL_DIST = 10


# Description: Drives the robot forward a certain heading and distance
# Input:       blockHeading = heading robot moves towards
#              blockDistance = distance robot moves
# Output:      none
def drive_forward(blockHeading, blockDistance):
    # Turn to block
    drivetrain.turn_to_heading(blockHeading, DEGREES)
    # How far we need to drive to get to block
    drivetrain.drive_for(FORWARD, blockDistance, MM)


# Description: Drives the robot backward a certain distance
# Input:       blockDistance = distance robot moves
# Output:      none
def drive_reverse(blockDistance):
    # how far we need to drive to get to block
    drivetrain.drive_for(REVERSE, blockDistance, MM)


# Description: Spins intake to collect blocks. Spins forward until intake bumber sensor is pressed.
# Input:       none
# Output:      none
def intakeBlock():
    while not intake_bumper.pressing():
        intake_motor_group.spin(FORWARD)
```

```python
        wait(.005, SECONDS)


# Description: Sets robot arm to "pluck" level and drives forward at same time (wait=False)
#              Moving arm while driving at same time , saves run time.
#              Pluck is different then Scoop, it gets blocks from inside flowers without bumping other
blocks
# Input:       blockHeading = heading robot moves towards
#              blockDistance = distance robot moves
# Output: none
def setupForIntakePluck(blockHeading, blockDistance):
    # put arm to the right level to pick up
    arm_motor_group.spin_to_position(ARM_MOTOR_PLUCK_LEVEL, DEGREES, wait=False)
    drive_forward(blockHeading, blockDistance)
    arm_motor_group.spin_to_position(ARM_MOTOR_INTAKE_LEVEL, DEGREES, wait=False)


# Description: Sets robot arm to "scoop" level and drives forward at same time (wait=False)
#              Moving arm while driving at same time , saves run time.
# Input:       blockHeading = heading robot moves towards
#              blockDistance = distance robot moves
# Output:      none
def setupForIntakeScoop(blockHeading, blockDistance):
    # put arm to the right level to pick up
    arm_motor_group.spin_to_position(ARM_MOTOR_INTAKE_LEVEL, DEGREES, wait=False)
    drive_forward(blockHeading, blockDistance)


# Description: Sets robot arm to outtake level and drives forward at same time (wait=False)
#              Moving arm while driving at same time , saves run time.
# Input: blockHeading = heading robot moves towards
#        blockDistance = distance robot moves
#        driveType = after picking up block drive forward or reverse to goal
# Output: none
def setupForOuttake(blockHeading, blockDistance, driveType, armLevel=ARM_MOTOR_OUTTAKE_LEVEL):
    # Raising arm to level to dump into goal(no wait)
    arm_motor_group.spin_to_position(armLevel, DEGREES, wait=False)
    if driveType == "f":
        drive_forward(blockHeading, blockDistance)
    else:
        drive_reverse(blockDistance)


# Description: Spins intake to score blocks. Spins in reverse until optical sensor doesn't see block.
# Input:       none
# Output:      none
def outtakeBlock():
    # spinning intake in reverse to dump the block when the arm isn't moving up
    while arm_motor_group.is_spinning() and front_optical.is_near_object():
        wait(.005, SECONDS)
    while front_optical.is_near_object():
        intake_motor_group.spin_for(REVERSE, OUTTAKE_ROTATE, DEGREES)
        wait(.005, SECONDS)


# Description: Calculates distance between two points using the pythagorean theorem.(a^2 + b^2 = c^2)
# Input:       (x1,y1) position of robot
#              (x2,y2) position of destination (block or goal)
# Output:      Distance in millimeters
def calculate_distance(x1, y1, x2, y2):
    return math.sqrt(math.pow(x2 - x1, 2) + math.pow(y2 - y1, 2))


# Description: Using trigonometry equation of a right angle. (a^2 + b^2 = c^2)
#              sin theta = opp/hyp
#              cos theta = adj/hyp
#              tan theta = opp/adj
# Input:       (x1,y1) position of robot
#              (x2,y2) position of destination (block or goal)
# Output:      heading in degrees
def calculate_heading(x1, y1, x2, y2):
    if x1 == x2:
```

```
        if y2 > y1:
            hdg = 0
        else:
            hdg = 180

    else:
        theta = math.atan((y2 - y1) / (x2 - x1)) * (180 / math.pi)
        if x2 > x1:
            hdg = 90 - theta
        else:
            hdg = 270 - theta
    return hdg


# Description: Using trigonometry equation of a right angle. (a^2 + b^2 = c^2)
#              sin theta = opp/hyp
#              cos theta = adj/hyp
#              tan theta = opp/adj
# Input:       (x1,y1) position of robot
#              (x2,y2) position of destination (block or goal)
# Output:      heading in degrees
def full_park(drive_heading, drive_distance):
    # turn lineup towards supply zone for reverse driving
    drivetrain.turn_to_heading(drive_heading + 180, DEGREES)
    # start moving arm forward without waiting to save time
    arm_motor_group.spin_for(FORWARD, 800, DEGREES, wait=False)
    # drive back into supply zone
    drivetrain.drive_for(REVERSE, drive_distance, MM)
    # straighten out robot to full park
    drivetrain.turn_to_heading(180, DEGREES)
    drivetrain.drive_for(REVERSE, 60, MM, wait=False)
    # full park flip backwards
    arm_motor_group.spin_for(FORWARD, 500, DEGREES)
    arm_motor_group.spin_for(REVERSE, 1500, DEGREES)


# Add project code in "main"
# Description: Our main function that runs our program,
#              it uses a grid map, control lists, loops, and function calls.
# Grid Map    : Our program sets up a coordinate grid on the playground where the initial starting position
#              at the lower left
#              goal two position is x=0, y=36 (millimeters). We know that the playground grid is set up
#              using squares that are each 300 x 300 (mm).
# Controls    : The program uses a big control list that holds data letting the robot know where to start
#              picking up blocks, where to take them to score and what type of way to pickup blocks.
# Loops       : We used a for loop to loop through each row in our list.
# Functions   : Inside the control loop we make calls to different functions that provide the robot actions.
def main():
    # Set Drivetrain and Motor Velocities
    drivetrain.set_drive_velocity(100, PERCENT)
    drivetrain.set_turn_velocity(100, PERCENT)
    intake_motor_group.set_velocity(100, PERCENT)
    arm_motor_group.set_velocity(100, PERCENT)

    # A list containing rows of data that have the control details for robot to perform different actions.
    # Example of row contains:
    # [x pos blk, y pos blk, x pos goal, y pos goal, heading at goal, intake collection type (scoop/pickup
), drive type after collection]
    # [300      , 900      , -70       , 1270      , -45,            , "s
"                                   , "f"                        ]
    control_list = [
        [-300, 600, -70, 1270, 315, "p", "f"],  # purple 2, goal 3
        [300, 1500, -70, 1270, 315, "p", "r"],  # purple 3, goal 3
        [900, 1500, -70, 1270, 315, "p", "r"],  # purple 4, goal 3
        [300, 900, -70, 1270, -45, "s", "r"],  # purple 1, goal 3
        [300, 1200, -125, 175, -135, "s", "f"],  # red 1, goal 2
        [600, 300, -125, 175, -140, "s", "f"],  # red 2, goal 2
        [600, 600, 1600, 200, 140, "s", "f"],  # purple 1, goal 1
        [900, 900, 900, 1200, 0, "sd", "f"],  # scoop purple 3, bump red 3
        [900, 1200, 1590, 210, 135, "d", "f"],  # dump/score purple 3, goal 1
        [1900, 880, 999, 999, 999, "fp", "r"]]  # collection zone purple 4 / full park
```

```python
    # initial robot position at lower left goal 2
    curr_x = 0
    curr_y = 36

    # loop through each row in control list
    for row in control_list:
        # calculate distance and heading from current position of robot to the block
        # subtract out robot distance from body to intake of robot from the block
        drive_distance = calculate_distance(curr_x, curr_y, row[0], row[1]) - BOT_LENGTH
        drive_heading = calculate_heading(curr_x, curr_y, row[0], row[1])

        # pickup a block either using a scroop or pluck (for flowers)
        if row[5] == "s" or row[5] == "sd":
            setupForIntakeScoop(drive_heading, drive_distance)
        if row[5] == "p":
            setupForIntakePluck(drive_heading, drive_distance)
        if row[5] == "fp":
            full_park(drive_heading, drive_distance)

        # spin intake to collect block
        intakeBlock()

        # calculate new x and y position after collecting block
        curr_x = curr_x + drive_distance * math.sin(math.radians(drive_heading))
        curr_y = curr_y + drive_distance * math.cos(math.radians(drive_heading))

        # calculate distance and heading from current position of robot to the goal
        drive_distance = calculate_distance(curr_x, curr_y, row[2], row[3])
        drive_heading = calculate_heading(curr_x, curr_y, row[2], row[3])

        # go to score or to bump block
        if row[5] != "sd":
            # get distance to goal with added goal distance over the container
            drive_distance = drive_distance + GOAL_DIST
            # raise arm to score, drive forward or backward to goal
            setupForOuttake(drive_heading, drive_distance, row[6])
            # turn to same outtake heading at the goal
            drivetrain.turn_to_heading(row[4], DEGREES)
            # score block
            outtakeBlock()
        else:
            # sd = scoop and drive/bump after picking up a block
            drive_distance = drive_distance - BOT_LENGTH
            # bump red block off peg
            setupForIntakeScoop(drive_heading, drive_distance)

            # calculate new x and y position after scoring/bumping block
        curr_x = curr_x + drive_distance * math.sin(math.radians(drive_heading))
        curr_y = curr_y + drive_distance * math.cos(math.radians(drive_heading))


# VR threads — Do not delete
vr_thread(main)
```