Taking Virtual Robots To a Higher Level

VRC Over Under Virtual Skills Challenge Code Explanation by Team 2813A

Table of Contents

A photo of our first time winning a tournament :D

Composition

Overview

The architecture of the virtual robot in our project is designed to mimic the real-life VEX robotics systems while integrating advanced programming concepts for efficiency and control.

Design Patterns

Singleton Pattern

- Purpose: Ensures that a class has only one instance and provides a global point of access to it. This pattern is crucial in our project as the robot, does not have multiple instances of the same component.
- Implementation:
  - Classes like Chassis, Intake, and Robot are designed as singletons.
  - The constructor (__init__) checks if __instance__ is None. If it is, it creates a new instance; otherwise, it raises an exception to prevent multiple instances.

Facade Pattern

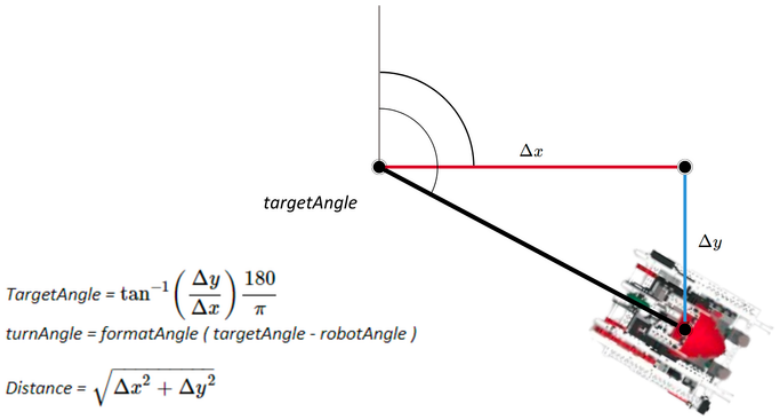
- Purpose: Provides a simplified interface to a complex subsystem. By doing this, it hides the complexity of the system’s components and their interactions, making them easier to use.
- Implementation:
  - The Robot class acts as a facade, wrapping the functionalities of subsystems like Chassis and Intake.
  - It exposes simplified methods for complex actions (e.g., move_to_point, shoot) which internally manage the detailed workings of these subsystems.
  - This approach allows for a clean and simple interface for controlling the robot's movements and actions.

Functions & Sensors

Key Functions

Movement Functions

- move_to_point(x, y):
  - Purpose: Moves the robot to a specific point on the field using GPS coordinates.
  - How It Works: It calculates the distance and angle to the target point and then commands the drivetrain to turn and move the robot accordingly.
  - Usage: Essential for navigating to specific locations on the field for tasks like scoring or collecting triballs.



move_to_point() calculations

- face_coord(x, y, aiming, offset):
  - Purpose: Rotates the robot to face a particular coordinate, with options for aiming and offset adjustments.
  - Functionality: Determines the angle between the robot's current position and the target coordinate, then rotates to align with this angle.

- face_angle(angle):
  - Purpose: Purpose: Orient the robot to face a specific absolute angle relative to the field.
  - Functionality:The robot calculates the difference between its current heading and the desired absolute angle. It then rotates in the most efficient direction to align itself with the specified angle.

Asynchronous Functions

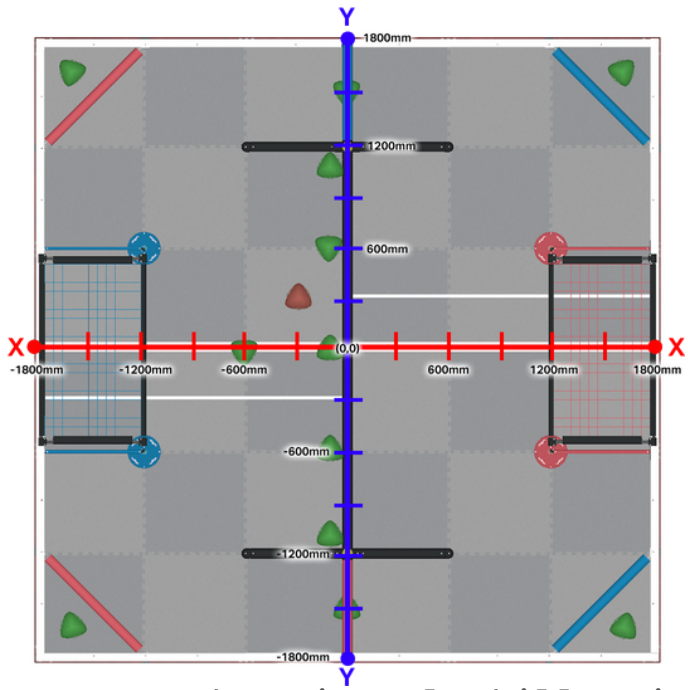
- shoot_while_move_to_point(x, y, d):
  - Purpose: Enables the robot to shoot a triball while moving towards a specific point on the field.
  - Functionality: The robot begins moving towards the target coordinates (x, y). Once it has traveled a specified distance d, the intake motor is activated to simulate shooting. This function is particularly useful for scoring points efficiently as it combines movement and shooting into a single, fluid action.

- intake_ball_at_point(x, y):
  - Purpose: Automates the process of intaking a triball at a specific location on the field.
  - Functionality: The robot navigates to the vicinity of the coordinates (x, y). Upon reaching the approximate location, the intake system is activated to collect the triball. This function is essential for tasks that involve collecting triballs from specific field locations.

Sensor Utilization

GPS for Navigation

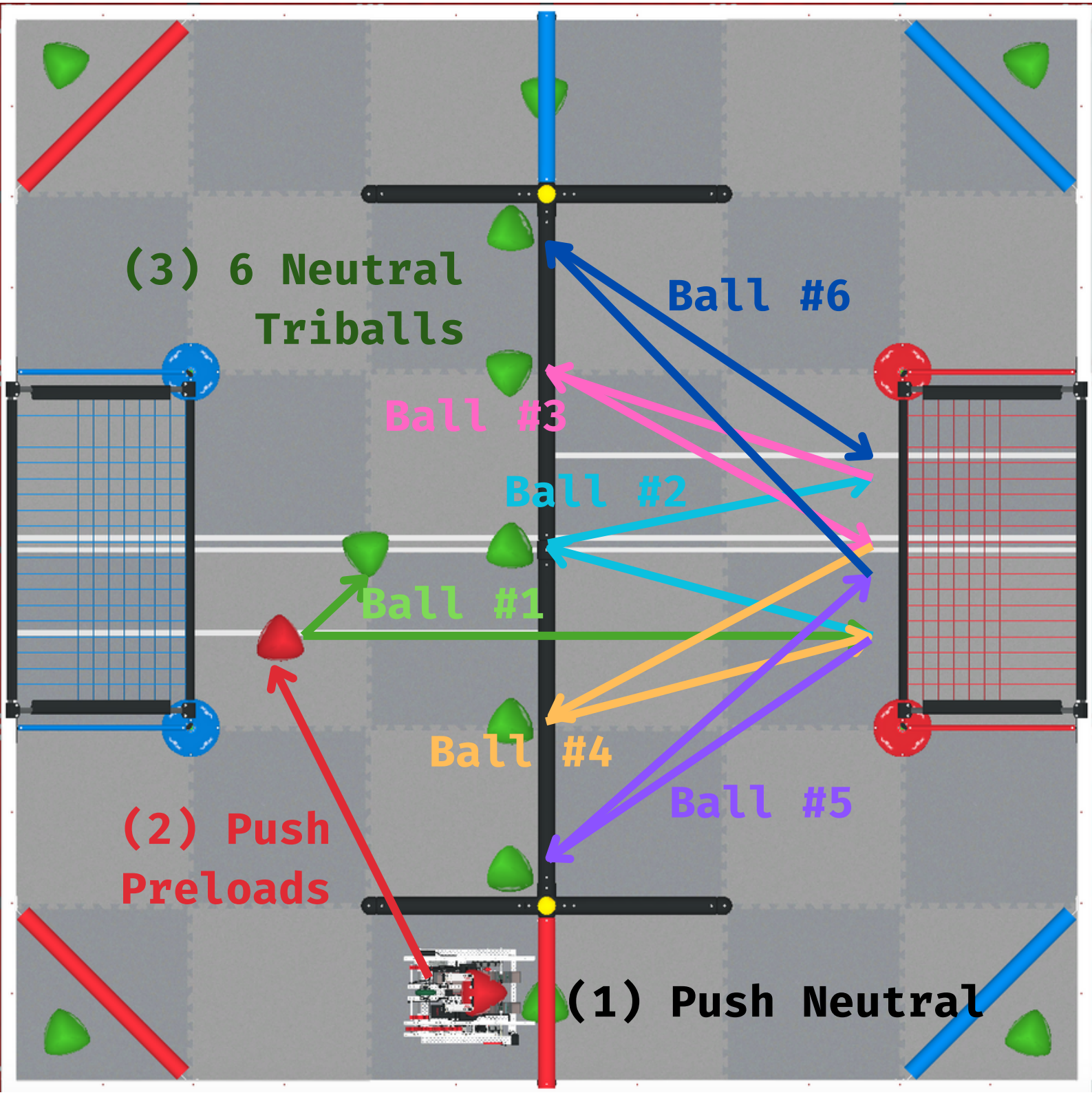
- Functionality: Provides real-time positional data for the robot, critical for accurate movements and strategy execution.



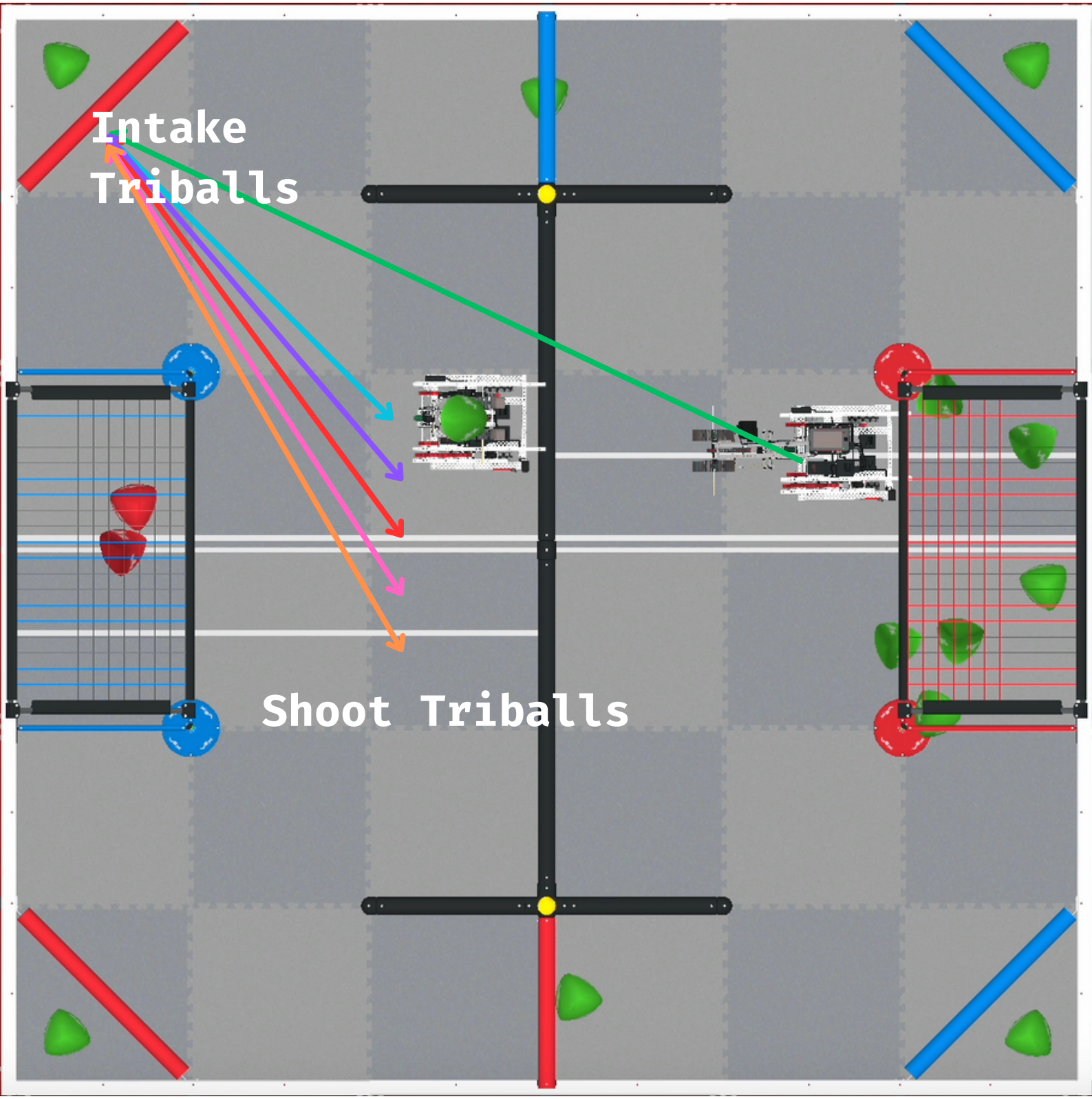
The Over Under Virtual Skills Field

Skills Route

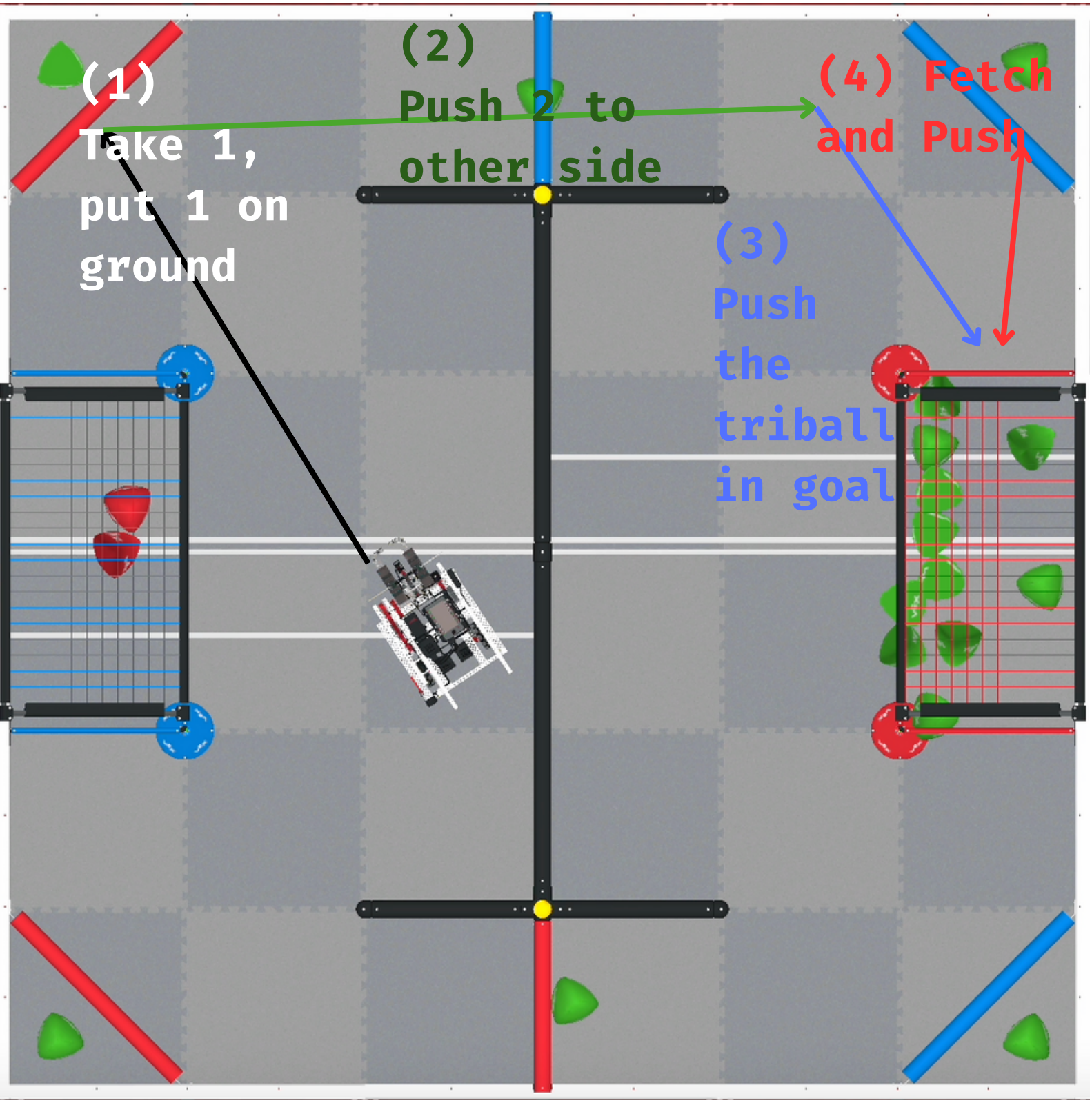
Part 1 2 Preloads + 6 Neutral Triballs on field



Part 2 Matchload 5 Balls from Load Zone 1



Part 3 Take as much as possible from load zones



Final Score: 81

13	81	0	2813A	Azre

Full Code

```python
#region VEXcode Generated Robot Configuration
import math
import random
from vexcode_vrc import *
from vexcode_vrc.events import get_Task_func

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

drivetrain = Drivetrain("drivetrain", 0)
arm_motor = Motor("ArmMotor", 3)
rotation = Rotation("Rotation", 7)
intake_motor = Motor("IntakeMotor", 8)
optical = Optical("Optical", 11)
gps = GPS("GPS", 20)

#endregion VEXcode Generated Robot Configuration
# ------------------------------------------
#
#  Project:      VEXcode Project
# Author:       VEX
# Created:
# Description:  VEXcode VR Python Project
#
# ------------------------------------------

#region helper functions
def format_angle(a):
    sign = 1
    if a < 0:
        sign = -1
    else:
        sign = 1
    positive_a = abs(a)

    mod = positive_a % 360

    if mod <= 180:
        return sign * mod
    else:
        return sign * (mod - 360)
#endregion helper functions

#region classes
class Coord:
    def __init__(self, x, y, theta=0):
        self.x = x
        self.y = y
        self.theta = theta

class Chassis:
    __instance__ = None

    def __init__(self):
        if Chassis.__instance__ is None:
            Chassis.__instance__ = self
        else:
            raise Exception("You cannot create another Chassis class")

    def face_angle(self, angle): # Face an absoulute angle
        drivetrain.set_turn_velocity(100, PERCENT)
        position = Coord(gps.x_position(MM), gps.y_position(MM), gps.heading())

        target_angle = format_angle(angle - position.theta)
        drivetrain.turn_for(RIGHT, target_angle, DEGREES)

    def face_coord(self, x, y, aiming, offset): # Face a coordinate
        drivetrain.set_turn_velocity(100, PERCENT)
        position = Coord(gps.x_position(MM), gps.y_position(MM), gps.heading())

        dx = x - position.x
        dy = y - position.y

        target_angle = 90 - math.atan2(dy, dx) * 180 / math.pi

        if aiming:
            # face coord backwards
            drivetrain.turn_for(RIGHT, float(format_angle(180 + target_angle - position.theta + offset)), DEGREES)
        else:
            # face the coord
            drivetrain.turn_for(RIGHT, float(format_angle(target_angle - position.theta + offset)), DEGREES)

    def move_to_point(self, x, y): # Move to a point
        drivetrain.set_drive_velocity(100, PERCENT)
        position = Coord(gps.x_position(MM), gps.y_position(MM), gps.heading())

        dx = x - position.x
        dy = y - position.y

        dist = (math.sqrt(dx**2 + dy**2))

        target_angle = 90 - math.atan2(dy, dx) * 180 / math.pi

        drivetrain.turn_for(RIGHT, float(format_angle(target_angle - position.theta)), DEGREES)

        drivetrain.drive_for(FORWARD, dist, MM)

    def move_to_point_backward(self, x, y): # Move to a point backward
        drivetrain.set_drive_velocity(100, PERCENT)
        position = Coord(gps.x_position(MM), gps.y_position(MM), gps.heading())

        dx = x - position.x
        dy = y - position.y

        dist = (math.sqrt(dx**2 + dy**2))

        target_angle = 90 - math.atan2(dy, dx) * 180 / math.pi

        drivetrain.turn_for(RIGHT, float(format_angle(180 + target_angle - position.theta)), DEGREES) # +180 degrees compared to move forward

        drivetrain.drive_for(REVERSE, dist, MM)

    def shoot_while_move_to_point(self, x, y, d): # Shoot a triball while moving to a point, start shooting when moved "d" MM
        drivetrain.set_drive_velocity(100, PERCENT)
```

```
        position = Coord(gps.x_position(MM), gps.y_position(MM), gps.heading())

        dx = x - position.x
        dy = y - position.y

        dist = (math.sqrt(dx**2 + dy**2))

        target_angle = 90 - math.atan2(dy, dx) * 180 / math.pi

        drivetrain.turn_for(RIGHT, float(format_angle(target_angle - position.theta)), DEGREES)

        drivetrain.drive_for(FORWARD, d, MM)
        intake_motor.spin(REVERSE)
        drivetrain.drive_for(FORWARD, dist-d, MM)

    def shoot_while_move_to_point_backward(self, x, y, d):
        drivetrain.set_drive_velocity(100, PERCENT)
        position = Coord(gps.x_position(MM), gps.y_position(MM), gps.heading())

        dx = x - position.x
        dy = y - position.y

        dist = (math.sqrt(dx**2 + dy**2))

        target_angle = 90 - math.atan2(dy, dx) * 180 / math.pi

        drivetrain.turn_for(RIGHT, float(format_angle(180 + target_angle - position.theta)), DEGREES)

        drivetrain.drive_for(REVERSE, d, MM)
        intake_motor.spin(REVERSE)
        drivetrain.drive_for(REVERSE, dist-d, MM)

    def intake_ball_at_point(self, x, y): # Intake a triball at a point, position is triball position
        drivetrain.set_drive_velocity(100, PERCENT)
        position = Coord(gps.x_position(MM), gps.y_position(MM), gps.heading())

        dx = x - position.x
        dy = y - position.y

        dist = (math.sqrt(dx**2 + dy**2))

        target_angle = 90 - math.atan2(dy, dx) * 180 / math.pi

        drivetrain.turn_for(RIGHT, float(format_angle(target_angle - position.theta)), DEGREES)
        intake_motor.spin(FORWARD)
        drivetrain.drive_for(FORWARD, dist - 300, MM) # Stop 300 MM before the point since the arm length is 300

class Intake:
    __instance__ = None

    def __init__(self):
        if Intake.__instance__ is None:
            Intake._instance__ = self
        else:
            raise Exception("You cannot create another Intake class")

    def shoot(self, turns): # Shoot a triball by rotating a specific turn
        intake_motor.spin_for(REVERSE, turns, TURNS)

class Robot:
    __instance__ = None

    def __init__(self):
        if Robot.__instance__ is None:
            Robot.__instance__ = self
            self.chassis = Chassis()
            self.intake = Intake()
        else:
            raise Exception("You cannot create another Robot class")

    @staticmethod
    def get_instance():
        if Robot.__instance__ is None:
            Robot()
        return Robot.__instance__

    @staticmethod
    def face_angle(angle):
        Robot.__instance__.chassis.face_angle(angle)

    @staticmethod
    def face_coord(x, y, aiming=False, offset=0):
        Robot.__instance__.chassis.face_coord(x, y, aiming, offset)

    @staticmethod
    def move_to_point(x, y):
        Robot.__instance__.chassis.move_to_point(x, y)

    @staticmethod
    def move_to_point_backward(x, y):
        Robot.__instance__.chassis.move_to_point_backward(x, y)

    @staticmethod
    def shoot_while_move_to_point(x, y, d):
        Robot.__instance__.chassis.shoot_while_move_to_point(x, y, d)

    @staticmethod
    def shoot_while_move_to_point_backward(x, y, d):
        Robot.__instance__.chassis.shoot_while_move_to_point_backward(x, y, d)

    @staticmethod
    def intake_ball_at_point(x, y):
        Robot.__instance__.chassis.intake_ball_at_point(x, y)

    @staticmethod
    def shoot(turns):
        Robot.__instance__.intake.shoot(turns)


#endregion classes

# Add project code in "main"
def main():
    #initalize components
    robot = Robot.get_instance()
```

```
    # Set velocity of all components to 100%
    drivetrain.set_drive_velocity(100, PERCENT)
    drivetrain.set_turn_velocity(100, PERCENT)
    intake_motor.set_velocity(100, PERCENT)
    arm_motor.set_velocity(100, PERCENT)

    # Preloads to Blue Goal
    arm_motor.spin(FORWARD)
    robot.move_to_point_backward(-250, -1500)
    robot.move_to_point(-900, -300)
    robot.shoot_while_move_to_point_backward(-600, -300, 0)
    drivetrain.drive_for(FORWARD, 100, MM)
    wait(0.25, SECONDS)

    # 1st (-600, 0)
    intake_motor.spin(FORWARD)
    robot.face_angle(20)
    wait(0.3, SECONDS)
    robot.face_angle(90)
    robot.shoot_while_move_to_point(600, -300, 200)

    # 2nd (-100, 0)
    robot.intake_ball_at_point(-100, 0)
    robot.shoot_while_move_to_point(700, 200, 0)

    # 3rd (-125, 600)
    robot.intake_ball_at_point(-125, 600)
    robot.shoot_while_move_to_point(700, 0, 100)

    # 4th (-125, -600)
    robot.intake_ball_at_point(-125, -600)
    robot.shoot_while_move_to_point(700, -300, 0)

    # 5th (-125, -1100)
    robot.intake_ball_at_point(-125, -1100)
    robot.shoot_while_move_to_point(800, -200, 200)

    # 6th (-125, 1100)
    robot.intake_ball_at_point(-125, 1100)
    robot.shoot_while_move_to_point(900, 300, 300)

    # Match Load Zone 1 (-1600, 1600)
    for i in range(5):
        robot.intake_ball_at_point(-1600, 1600)
        wait(0.2, SECONDS)
        arm_motor.spin_to_position(0.9, TURNS, wait=False)
        intake_motor.stop()

        robot.move_to_point_backward(-360, 375-150*i) # Move y down 150 every time to revent the triballs from sticking together and not entering the goal
        robot.face_angle(270)
        wait(0.1, SECONDS)

        robot.shoot(1.2)
        arm_motor.spin_to_position(3.35, TURNS, wait=False)

    # take two from matchload zone
    arm_motor.spin(FORWARD)
    wait(0.5, SECONDS)
    robot.face_coord(-1600, 1600)
    robot.intake_ball_at_point(-1600, 1600)
    robot.face_coord(-1600, 1600)
    wait(0.2, SECONDS)
    intake_motor.spin(REVERSE)
    robot.face_angle(80)
    wait(0.4, SECONDS)
    intake_motor.spin(FORWARD)
    robot.face_coord(-1600, 1600)
    wait(0.2, SECONDS)

    # push two triballs to other side
    robot.move_to_point(500, 1500)

    # push one in goal
    intake_motor.spin(REVERSE)
    robot.move_to_point(1500, 850)
    wait(0.2, SECONDS)

    # push blue matchload zone ball in goal
    robot.intake_ball_at_point(1600, 1600)
    robot.face_coord(1600, 1600)
    wait(0.3, SECONDS)
    robot.shoot_while_move_to_point(1500, 900, 50)

    # step back to prevent touching triballs
    robot.move_to_point_backward(1500, 1100)

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

- **2813 VR Skills Template**

https://github.com/hhe1ibeb/2813VRskillsLib

- **Team Member**: Ethan, Elaine, Ellie, Vic, Kyle, Morris
- **Location**: Taipei, Taiwan