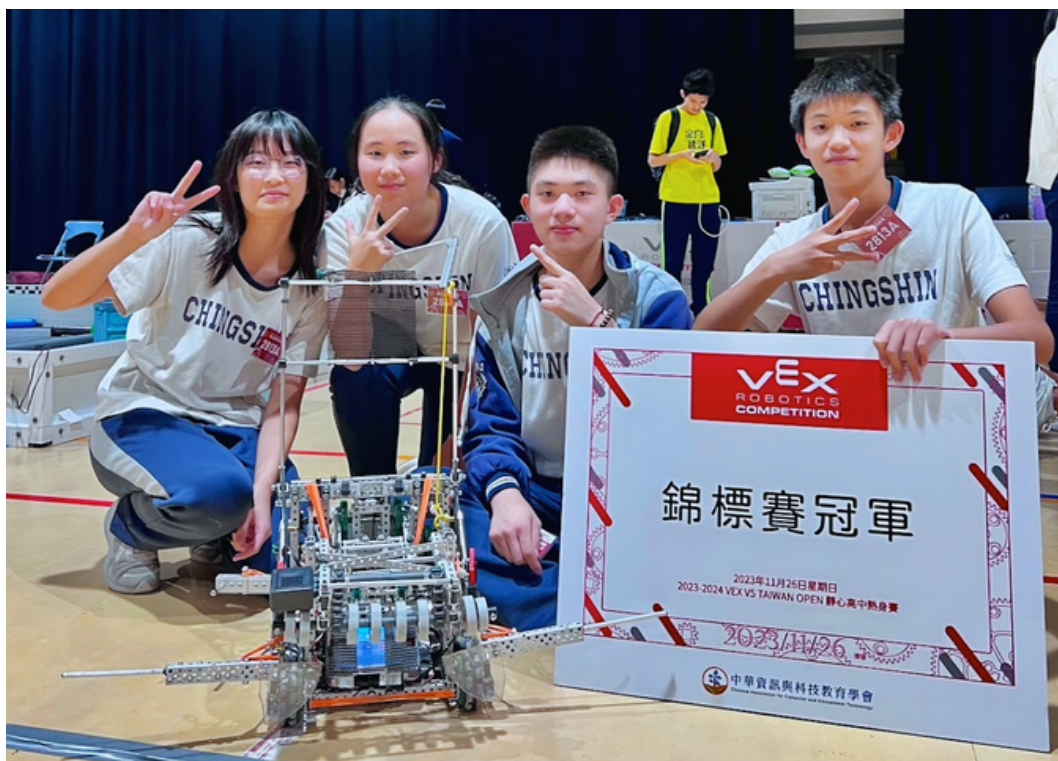
The background of the slide is a virtual reality (VR) environment. At the top, there is a red grid floor with two red circular bases. In the center, a virtual robot is shown from a top-down perspective, tilted at an angle. At the bottom, there is a blue grid floor with two blue circular bases. The robot is a complex assembly of white and grey parts, with a green sensor or camera mounted on top. The text is overlaid on this scene.

Taking Virtual Robots To a Higher Level

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

Table of Contents

Composition	3
Functions and Sensors	4
Skills Route	6
Full Code	9



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

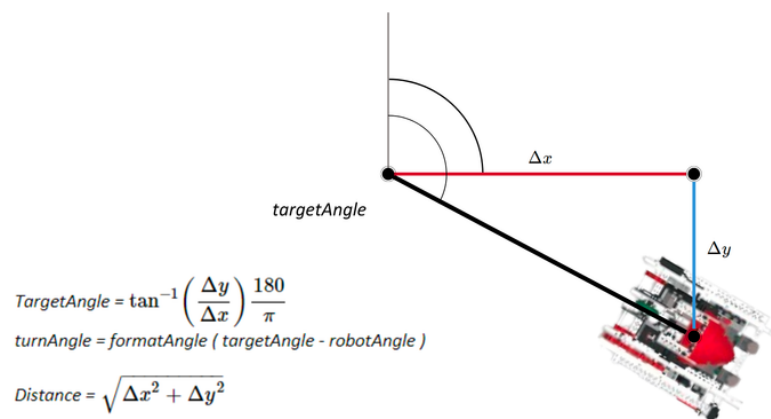
- **Purpose:** Provides a simplified interface to a complex subsystem. By doing this, it hides the complexity of the system's components and their interactions, making them easier to use.
- **Implementation:**
 - The Robot class acts as a facade, wrapping the functionalities of subsystems like Chassis 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: 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.

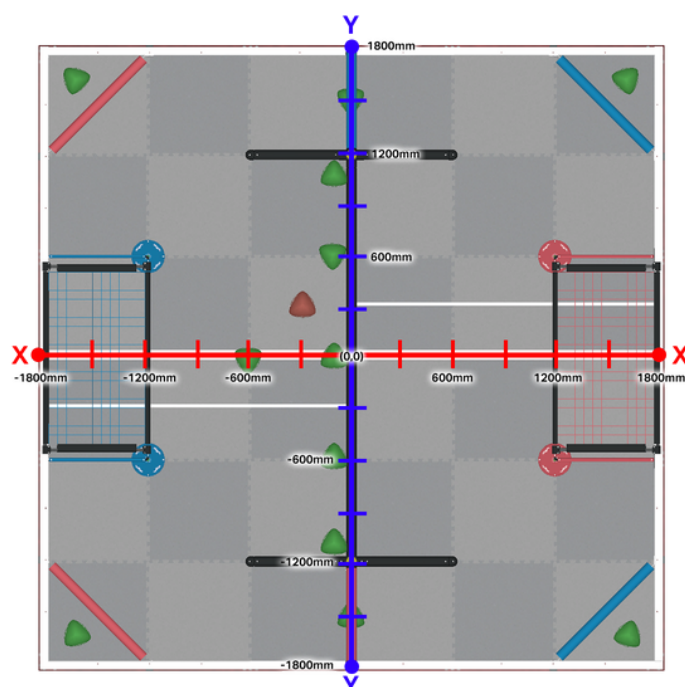
Asynchronous Functions

- `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

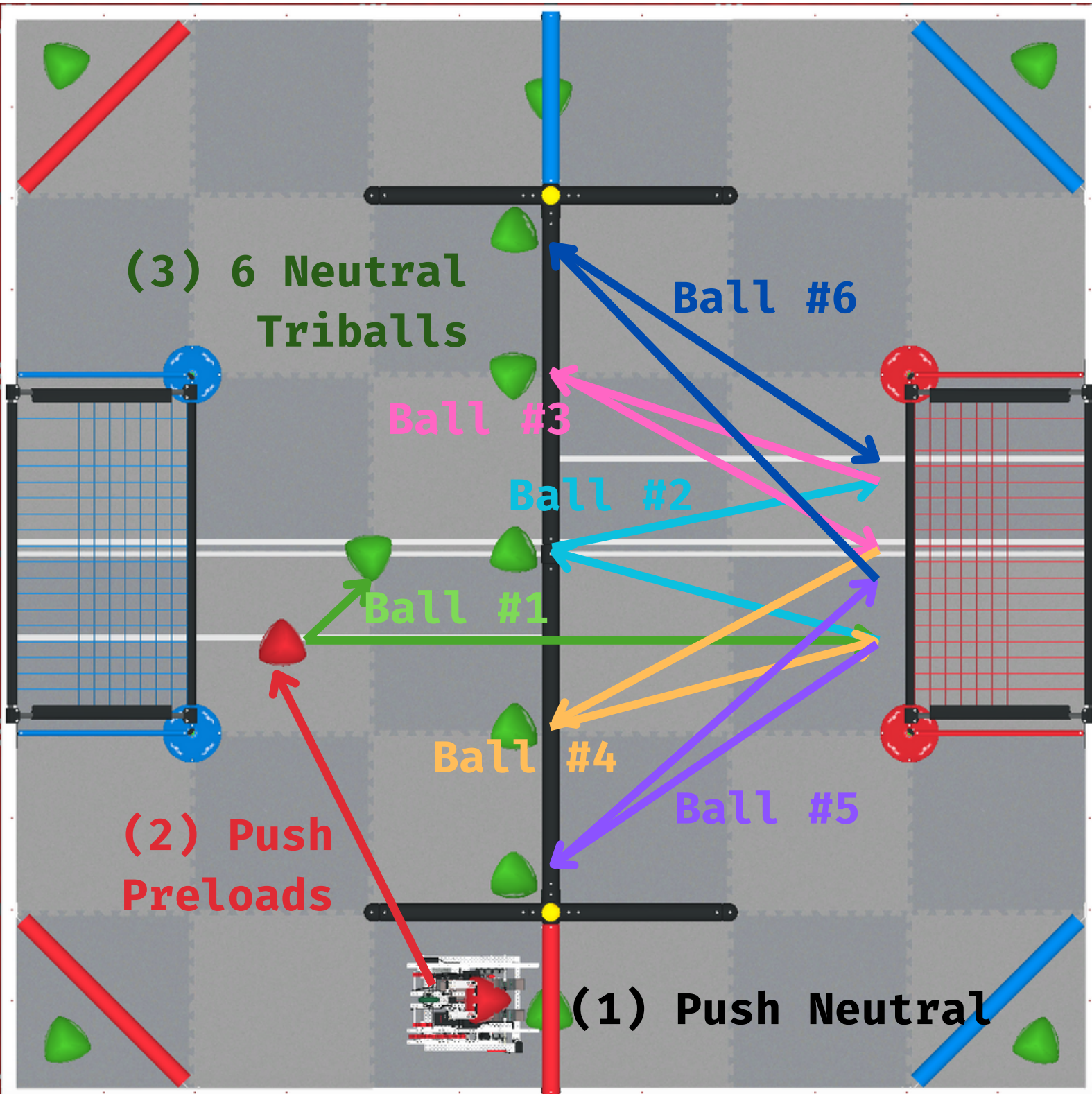
- Functionality: Provides real-time positional data for the robot, critical for accurate movements and strategy execution.



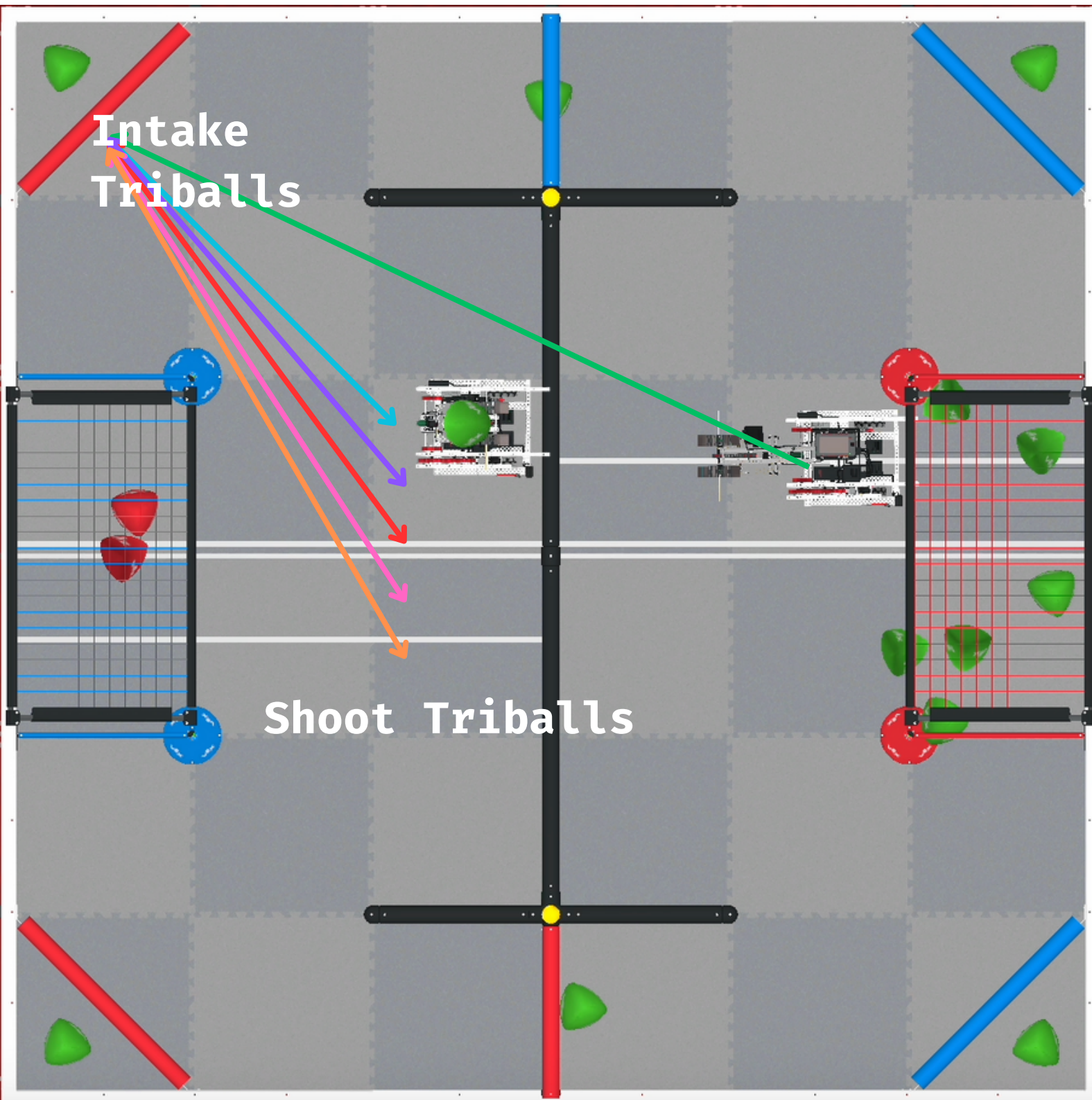
The Over Under Virtual Skills Field

Skills Route

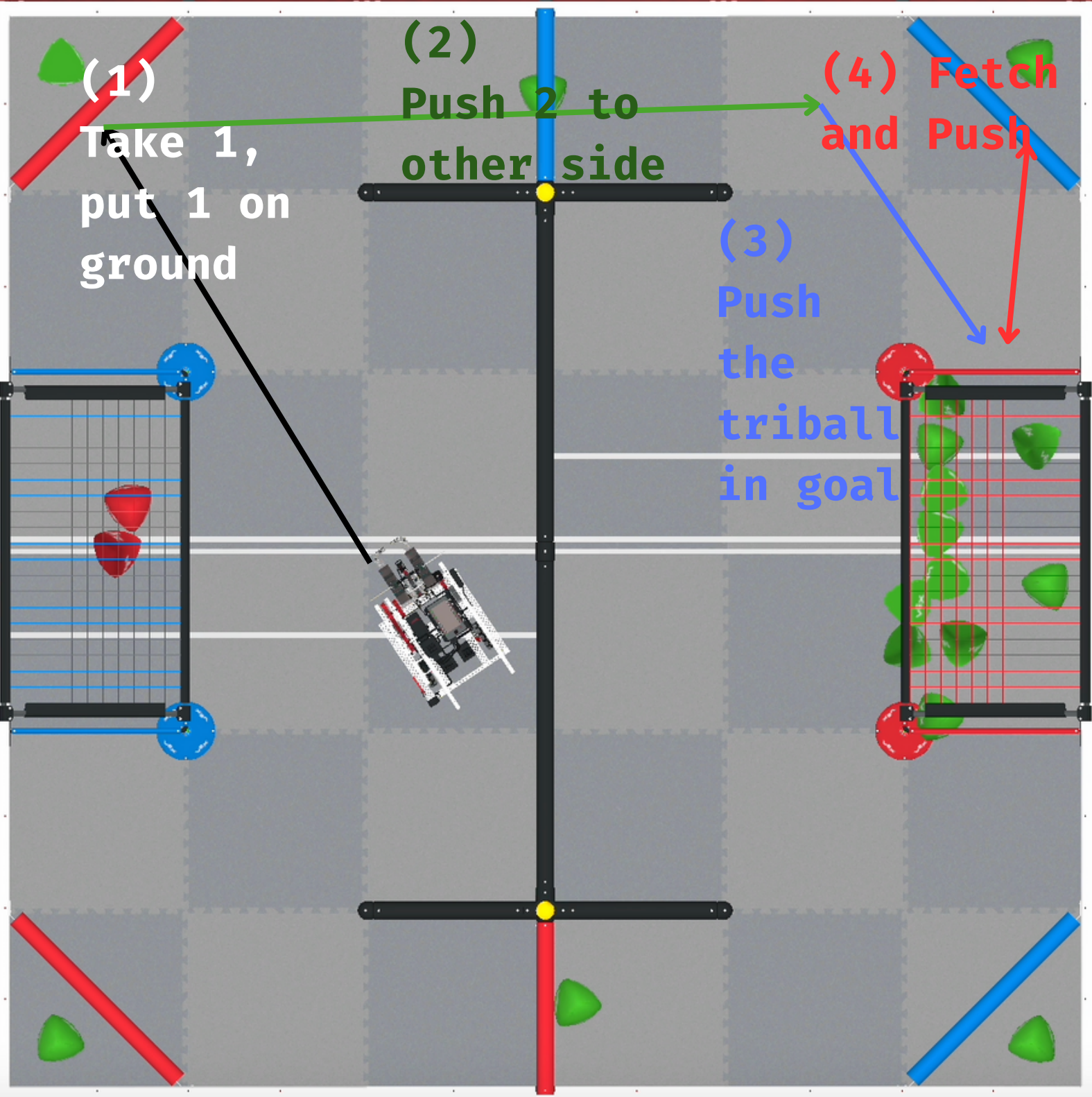
Part 1 2 **Preloads** + 6 **Neutral** Triballs on field



Part 2 Matchload 5 Balls from Load Zone 1



Part 3 Take as much as possible from load zones



Final Score: 81

Full Code

```
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 # -----
19 #
20 # Project:      VEXcode Project
21 # Author:      VEX
22 # Created:
23 # Description: VEXcode VR Python Project
24 #
25 # -----
26
27 #region helper functions
28 def format_angle(a):
29     sign = 1
30     if a < 0:
31         sign = -1
32     else:
33         sign = 1
34     positive_a = abs(a)
35
36     mod = positive_a % 360
37
38     if mod <= 180:
39         return sign * mod
40     else:
41         return sign * (mod - 360)
42 #endregion helper functions
43
44 #region classes
45 class Coord:
46     def __init__(self, x, y, theta=0):
47         self.x = x
48         self.y = y
49         self.theta = theta
50
51 class Chassis:
52     __instance__ = None
53
54     def __init__(self):
55         if Chassis.__instance__ is None:
56             Chassis.__instance__ = self
57         else:
58             raise Exception("You cannot create another Chassis class")
59
60     def face_angle(self, angle): # Face an absolute angle
61         drivetrain.set_turn_velocity(100, PERCENT)
62         position = Coord(gps.x_position(MM), gps.y_position(MM), gps.heading())
63
64         target_angle = format_angle(angle - position.theta)
65         drivetrain.turn_for(RIGHT, target_angle, DEGREES)
66
67     def face_coord(self, x, y, aiming, offset): # Face a coordinate
68         drivetrain.set_turn_velocity(100, PERCENT)
69         position = Coord(gps.x_position(MM), gps.y_position(MM), gps.heading())
70
71         dx = x - position.x
72         dy = y - position.y
73
74         target_angle = 90 - math.atan2(dy, dx) * 180 / math.pi
75
76         if aiming:
77             # face coord backwards
78             drivetrain.turn_for(RIGHT, float(format_angle(180 + target_angle - position.theta + offset)), DEGREES)
79         else:
80             # face the coord
81             drivetrain.turn_for(RIGHT, float(format_angle(target_angle - position.theta + offset)), DEGREES)
82
83     def move_to_point(self, x, y): # Move to a point
84         drivetrain.set_drive_velocity(100, PERCENT)
85         position = Coord(gps.x_position(MM), gps.y_position(MM), gps.heading())
86
87         dx = x - position.x
88         dy = y - position.y
89
90         dist = (math.sqrt(dx**2 + dy**2))
91
92         target_angle = 90 - math.atan2(dy, dx) * 180 / math.pi
93
94         drivetrain.turn_for(RIGHT, float(format_angle(target_angle - position.theta)), DEGREES)
95
96         drivetrain.drive_for(FORWARD, dist, MM)
97
98     def move_to_point_backward(self, x, y): # Move to a point backward
99         drivetrain.set_drive_velocity(100, PERCENT)
100        position = Coord(gps.x_position(MM), gps.y_position(MM), gps.heading())
101
102        dx = x - position.x
103        dy = y - position.y
104
105        dist = (math.sqrt(dx**2 + dy**2))
106
107        target_angle = 90 - math.atan2(dy, dx) * 180 / math.pi
108
109        drivetrain.turn_for(RIGHT, float(format_angle(180 + target_angle - position.theta)), DEGREES) # +180 degrees compared to move forward
110
111        drivetrain.drive_for(REVERSE, dist, MM)
112
113     def shoot_while_move_to_point(self, x, y, d): # Shoot a triball while moving to a point, start shooting when moved "d" MM
114         drivetrain.set_drive_velocity(100, PERCENT)
```

```

115     position = Coord(gps.x_position(MM), gps.y_position(MM), gps.heading())
116
117     dx = x - position.x
118     dy = y - position.y
119
120     dist = (math.sqrt(dx**2 + dy**2))
121
122     target_angle = 90 - math.atan2(dy, dx) * 180 / math.pi
123
124     drivetrain.turn_for(RIGHT, float(format_angle(target_angle - position.theta)), DEGREES)
125
126     drivetrain.drive_for(FORWARD, d, MM)
127     intake_motor.spin(REVERSE)
128     drivetrain.drive_for(FORWARD, dist-d, MM)
129
130 def shoot_while_move_to_point_backward(self, x, y, d):
131     drivetrain.set_drive_velocity(100, PERCENT)
132     position = Coord(gps.x_position(MM), gps.y_position(MM), gps.heading())
133
134     dx = x - position.x
135     dy = y - position.y
136
137     dist = (math.sqrt(dx**2 + dy**2))
138
139     target_angle = 90 - math.atan2(dy, dx) * 180 / math.pi
140
141     drivetrain.turn_for(RIGHT, float(format_angle(180 + target_angle - position.theta)), DEGREES)
142
143     drivetrain.drive_for(REVERSE, d, MM)
144     intake_motor.spin(REVERSE)
145     drivetrain.drive_for(REVERSE, dist-d, MM)
146
147 def intake_ball_at_point(self, x, y): # Intake a triball at a point, position is triball position
148     drivetrain.set_drive_velocity(100, PERCENT)
149     position = Coord(gps.x_position(MM), gps.y_position(MM), gps.heading())
150
151     dx = x - position.x
152     dy = y - position.y
153
154     dist = (math.sqrt(dx**2 + dy**2))
155
156     target_angle = 90 - math.atan2(dy, dx) * 180 / math.pi
157
158     drivetrain.turn_for(RIGHT, float(format_angle(target_angle - position.theta)), DEGREES)
159     intake_motor.spin(FORWARD)
160     drivetrain.drive_for(FORWARD, dist - 300, MM) # Stop 300 MM before the point since the arm length is 300
161
162 class Intake:
163     __instance__ = None
164
165     def __init__(self):
166         if Intake.__instance__ is None:
167             Intake.__instance__ = self
168         else:
169             raise Exception("You cannot create another Intake class")
170
171     def shoot(self, turns): # Shoot a triball by rotating a specific turn
172         intake_motor.spin_for(REVERSE, turns, TURNS)
173
174 class Robot:
175     __instance__ = None
176
177     def __init__(self):
178         if Robot.__instance__ is None:
179             Robot.__instance__ = self
180             self.chassis = Chassis()
181             self.intake = Intake()
182         else:
183             raise Exception("You cannot create another Robot class")
184
185     @staticmethod
186     def get_instance():
187         if Robot.__instance__ is None:
188             Robot()
189         return Robot.__instance__
190
191     @staticmethod
192     def face_angle(angle):
193         Robot.__instance__.chassis.face_angle(angle)
194
195     @staticmethod
196     def face_coord(x, y, aiming=False, offset=0):
197         Robot.__instance__.chassis.face_coord(x, y, aiming, offset)
198
199     @staticmethod
200     def move_to_point(x, y):
201         Robot.__instance__.chassis.move_to_point(x, y)
202
203     @staticmethod
204     def move_to_point_backward(x, y):
205         Robot.__instance__.chassis.move_to_point_backward(x, y)
206
207     @staticmethod
208     def shoot_while_move_to_point(x, y, d):
209         Robot.__instance__.chassis.shoot_while_move_to_point(x, y, d)
210
211     @staticmethod
212     def shoot_while_move_to_point_backward(x, y, d):
213         Robot.__instance__.chassis.shoot_while_move_to_point_backward(x, y, d)
214
215     @staticmethod
216     def intake_ball_at_point(x, y):
217         Robot.__instance__.chassis.intake_ball_at_point(x, y)
218
219     @staticmethod
220     def shoot(turns):
221         Robot.__instance__.intake.shoot(turns)
222
223
224 #endregion classes
225
226 # Add project code in "main"
227 def main():
228     #initialize components
229     robot = Robot.get_instance()
230

```

```

230
231 # Set velocity of all components to 100%
232 drivetrain.set_drive_velocity(100, PERCENT)
233 drivetrain.set_turn_velocity(100, PERCENT)
234 intake_motor.set_velocity(100, PERCENT)
235 arm_motor.set_velocity(100, PERCENT)
236
237 # Preloads to Blue Goal
238 arm_motor.spin(FORWARD)
239 robot.move_to_point_backward(-250, -1500)
240 robot.move_to_point(-900, -300)
241 robot.shoot_while_move_to_point_backward(-600, -300, 0)
242 drivetrain.drive_for(FORWARD, 100, MM)
243 wait(0.25, SECONDS)
244
245 # 1st (-600, 0)
246 intake_motor.spin(FORWARD)
247 robot.face_angle(20)
248 wait(0.3, SECONDS)
249 robot.face_angle(90)
250 robot.shoot_while_move_to_point(600, -300, 200)
251
252 # 2nd (-100, 0)
253 robot.intake_ball_at_point(-100, 0)
254 robot.shoot_while_move_to_point(700, 200, 0)
255
256 # 3rd (-125, 600)
257 robot.intake_ball_at_point(-125, 600)
258 robot.shoot_while_move_to_point(700, 0, 100)
259
260 # 4th (-125, -600)
261 robot.intake_ball_at_point(-125, -600)
262 robot.shoot_while_move_to_point(700, -300, 0)
263
264 # 5th (-125, -1100)
265 robot.intake_ball_at_point(-125, -1100)
266 robot.shoot_while_move_to_point(800, -200, 200)
267
268 # 6th (-125, 1100)
269 robot.intake_ball_at_point(-125, 1100)
270 robot.shoot_while_move_to_point(900, 300, 300)
271
272 # Match Load Zone 1 (-1600, 1600)
273 for i in range(5):
274     robot.intake_ball_at_point(-1600, 1600)
275     wait(0.2, SECONDS)
276     arm_motor.spin_to_position(0.9, TURNS, wait=False)
277     intake_motor.stop()
278
279     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
280     robot.face_angle(270)
281     wait(0.1, SECONDS)
282
283     robot.shoot(1.2)
284     arm_motor.spin_to_position(3.35, TURNS, wait=False)
285
286 # take two from matchload zone
287 arm_motor.spin(FORWARD)
288 wait(0.5, SECONDS)
289 robot.face_coord(-1600, 1600)
290 robot.intake_ball_at_point(-1600, 1600)
291 robot.face_coord(-1600, 1600)
292 wait(0.2, SECONDS)
293 intake_motor.spin(REVERSE)
294 robot.face_angle(80)
295 wait(0.4, SECONDS)
296 intake_motor.spin(FORWARD)
297 robot.face_coord(-1600, 1600)
298 wait(0.2, SECONDS)
299
300 # push two triballs to other side
301 robot.move_to_point(500, 1500)
302
303 # push one in goal
304 intake_motor.spin(REVERSE)
305 robot.move_to_point(1500, 850)
306 wait(0.2, SECONDS)
307
308 # push blue matchload zone ball in goal
309 robot.intake_ball_at_point(1600, 1600)
310 robot.face_coord(1600, 1600)
311 wait(0.3, SECONDS)
312 robot.shoot_while_move_to_point(1500, 900, 50)
313
314 # step back to prevent touching triballs
315 robot.move_to_point_backward(1500, 1100)
316
317 # VR threads - Do not delete
318 vr_thread(main)

```

- **2813 VR Skills Template**

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

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