Software Training

Task 5 – ROS2

Note: All tasks should be implemented inside a single ROS 2 workspace. You will submit a GitHub repository link containing your workspace with a single package called `turtle_chase`.

The folder structure should look like:

```
ros2_ws (Your ROS 2 workspace)
├─ src (Source folder)
│  └─ turtle_pkg (Package)
│     ├─ launch (Launch files)
│     │  └─ turtle_chase_launch.py
│     └─ turtle_pkg
│        └─ turtle_chase.py
├─ install
├─ build
└─ log
```

Task Description: Turtle Chase Game

You are required to build a simple chase game inside `turtlesim`.

The player controls the default turtle – `turtle1` – using the keyboard, and must move it to collide with randomly spawned turtles ("enemies"). When a collision occurs, the enemy disappears, the score increases, and a new enemy spawns at a random location.

Requirements

1. Create a Python node called `turtle_chase.py` inside the `turtle_pkg` package. This node will handle:
   - Spawning enemy turtles
   - Detecting collisions
   - Respawning enemies
   - Publishing the score
2. Use the following **turtlesim** services:
   - `/spawn`: Creates new turtles at given (x, y) coordinates.
     Take a look at Github Documentation for this service: https://github.com/ros/ros_tutorials/blob/noetic-devel/turtlesim/srv/Spawn.srv
   - `/kill`: Removes turtles by name.
     Take a look at Github Documentation for this service: https://github.com/ros/ros_tutorials/blob/noetic-devel/turtlesim/srv/Kill.srv

3. When the game starts:
   - Spawn 3 **turtles** – `enemy1`, `enemy2`, `enemy3` – at random positions.
   - Subscribe to the player's pose `/turtle1/pose`
   - Subscribe to each enemy's pose `/enemy`X`/pose`.X is the number given to the enemy turtle.
4. Implement a **collision detection mechanism** using a timer callback:
   - If the distance between the player turtle and an enemy is < 0.05, count it as a hit.
   - On collision:
     - Remove the enemy turtle using `/kill`.
     - Respawn it at a new random location using `/spawn`.
     - Increase the score by 1.
5. Publish the score on topic `/score` using the `std_msgs/msg/Int32` message type. Verify the score using: `ros2 topic echo /score`
6. Create a launch file called `turtle_chase_launch.py` that starts:
   - `turtlesim_node` (simulation environment)
   - `turtle_pkg` (the game logic node)
7. Run `turtle_teleop_key` (for controlling the player turtle with arrow keys)

Deliverables

- **ROS 2 node** `turtle_chase.py` implementing enemy spawning, collision detection, and score publishing.
- **Launch file** `turtle_chase_launch.py` to start turtlesim node and `turtle_pkg` nodes.
- **Screenshot(s)** of the turtlesim window showing multiple turtles.
- **Screenshot** of terminal output showing the `/score` topic updating.
- Video Recording of the whole game showing how it works

Notes on Implementation

1. **You'll have some callback functions:**
   - `player_callback(msg:Pose)`: receives `/turtle1/pose`.
   - `enemy_callback(msg:Pose)`: receives enemy poses and appends it to a dictionary called `enemy_positions` – used as follows: `enemy_positions[name]=msg`
   - `check_collisions()`: timer callback to check for collisions.

2. check_collision function should look something like this:

```python
def check_collisions(self):
    #if player node (turtle1) isn't available exit the function
    for name, pose in list(self.enemy_positions.items()):
        # find distance between player and enemy
        # if dist < 0.05 then
        # log that enemeyX was hit
                # Update the score and publish it
                # kill_enemy('enemeyX')
                # spawn_enemy('enemeyX') -> the new spawned turtle enemy would have the same number as the one you just killed

```

3. **Client function for request spawning a new turtle and killing the one that got hit**

   - spawn_enemy(name): calls /spawn to create a turtle.
   - kill_enemy(name): calls /kill to remove a turtle.

4. You can also implement a function called find_distance(pose1: Pose,pose2: Pose) to calculate the distance between 2 turtles given their current Position (pose1, pose2).
   Pose is a message type that describes a point in space (x, y, z) so to calculate the distance between 2 points in space (x_1, y_1), (x_2, y_2)
   Recall – equation for distance between two points: $\sqrt{(y_2 - y_1)^2 + (x_2 - x_1)^2}$

END OF DOCUMENT