PROJECT 2: TEAM 13
Team Members: Ayushri Jain

The program begins by utilizing a launch file to initiate its execution. It starts by creating a LogoDrawer node that takes in a parameter named 'num_turtles'. Subsequently, the program generates 'num_turtles' instances of the TurtleController node, and these nodes are executed concurrently.

Each TurtleController node has its own publisher, subscriber, and client functionalities. These nodes control individual turtle robots to draw specified lines in parallel, coordinating their movements to achieve the desired drawing. The code implements logic for turtle control, including spawning, killing, and navigating the turtles to designated positions along the specified lines. This allows for synchronized and distributed drawing by multiple turtle robots.

- **line_length Function:** It calculates and returns the Euclidean distance between two points representing the endpoints of a line segment. It takes a single argument line, which is a list containing two tuples, each representing the (x, y, theta) coordinates of the line's endpoints.
- **divide_lines_into_parts Function**: It divides a list of lines into parts based on the number of turtles specified in the LogoDrawer class. It ensures an approximately equal distribution of lines among the turtles while handling any remaining lines. It takes two arguments: lines (the list of lines to be divided) and n (the number of turtles). The function calculates the number of lines each turtle should handle (equal_division) and distributes them accordingly. Remaining lines, if any, are assigned to the last turtle. The function returns a list of parts, where each part is a sublist of lines assigned to a specific turtle.
- **cmd_publisher**: It is an instance of the Twist message publisher associated with each TurtleController node. It publishes linear and angular velocity commands to control the turtle's movement.
- **pose_subscriber**: It is a subscription to the Pose topic specific to each turtle. It listens for pose updates, which provide information about the turtle's current position and orientation.
- **pen_client**: It is a client used to interact with the SetPen service of each turtle. It allows the code to control whether the turtle's pen is down (drawing) or up (not drawing).
- **timer**: It is a timer callback associated with each TurtleController node. It periodically triggers the timer_callback method at a specified interval (0.1 seconds) to control the turtle's movement.
- **current_index**: It keeps track of the current line segment that the turtle is working on. It is incremented as the turtle completes drawing a line and moves on to the next one.
- **spawn_turtle Function**: It is responsible for spawning a turtle at a predefined position. It uses the Spawn service to create a new turtle and set its initial position and orientation. The function is called when a new TurtleController node is initialized.

- **pose_callback Function**: It is called whenever a pose update is received by the pose_subscriber. It updates the pose attribute of the TurtleController node with the latest pose information.
- **timer_callback Function**: It is periodically triggered by the timer. It contains the logic for controlling the turtle's movement to reach its designated start and end points, drawing lines in the process. It also manages when the turtle should start, stop, or switch to a new line segment, based on its current state and goals.

Design Choices:

- **Parameterization:** The program allows specifying the number of turtles (robots) to be used as a parameter. This enables flexibility in the number of robots involved in the drawing task.
- **Parallel Execution:** Each turtle is represented as a separate instance of the TurtleController class, enabling parallel execution and individual control for each robot.
- **Modular Approach:** The code employs a modular approach, dividing tasks into classes and functions for better organization and maintainability.

Handling Multi-Robot Aspect:

- **Publishers and Subscribers:** Each TurtleController node manages its own publishers for issuing movement commands (cmd_publisher), subscribers for receiving pose updates (pose_subscriber), and clients for interacting with services like pen control (pen_client). This isolation ensures that each robot operates independently.
- **Timer Callback:** The timer_callback function controls the movement and behavior of each individual robot. It checks the robot's current state and goals, enabling synchronized but independent actions for each robot.

Managing Individual Robot Movement:

- **Timers:** Timers are used to periodically trigger the timer_callback function for each turtle. This function determines whether a turtle should move, turn, or pause based on its current state and goals.
- **Pose Updates:** Pose updates received through subscribers (pose_subscriber) allow the program to keep track of each turtle's position and orientation.
- **Services:** Services, such as SetPen, are used to control when a turtle's pen should be up (not drawing) or down (drawing).

Specific Values:

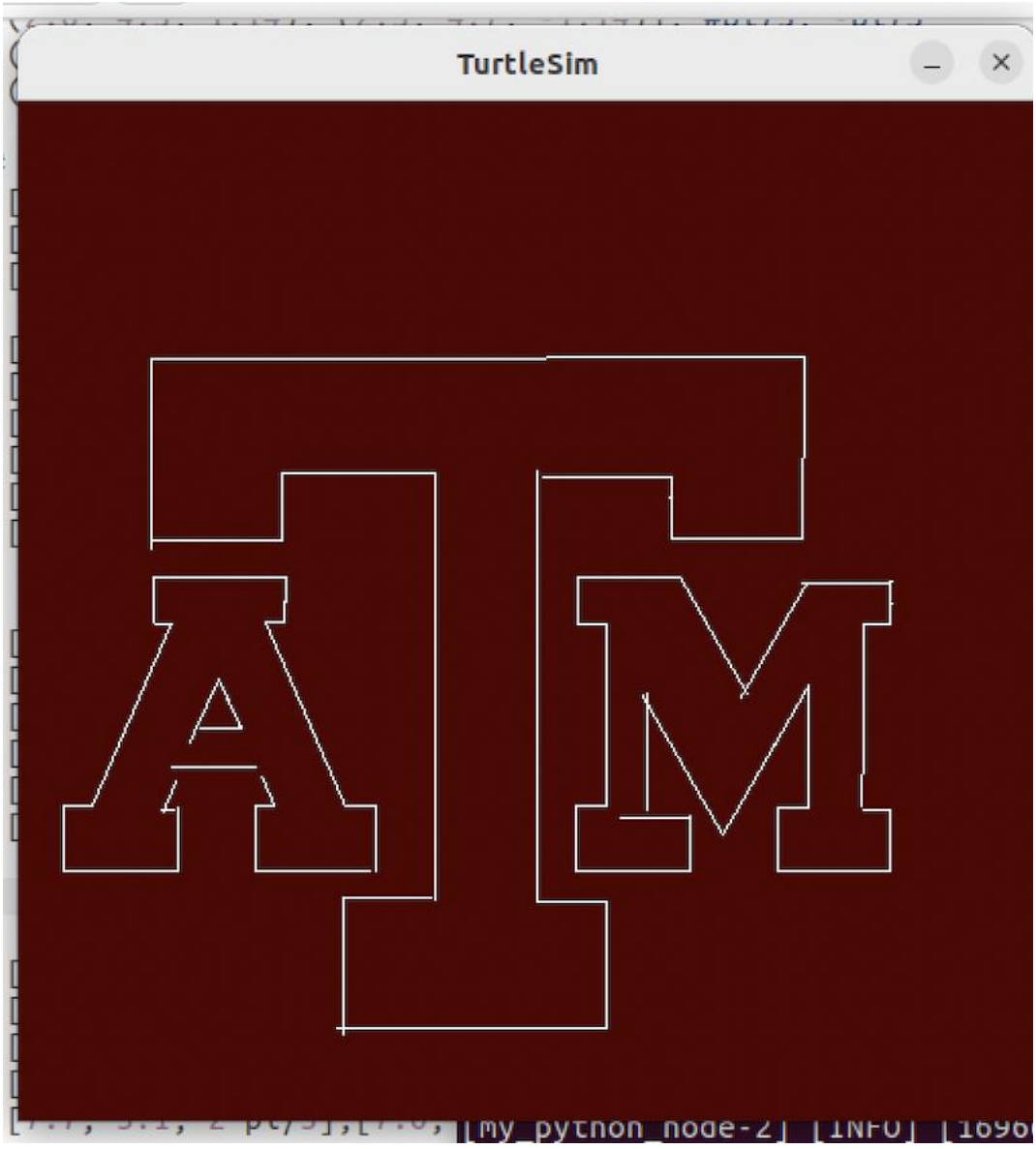
Specific numbers and values in the code include time intervals, distances, and angles.

- The timer interval for timer_callback is set to 0.1 seconds.
- The program checks for distance thresholds (0.01) to determine when a turtle should move.
- Angle thresholds (0.001) are used to control turtle orientation.

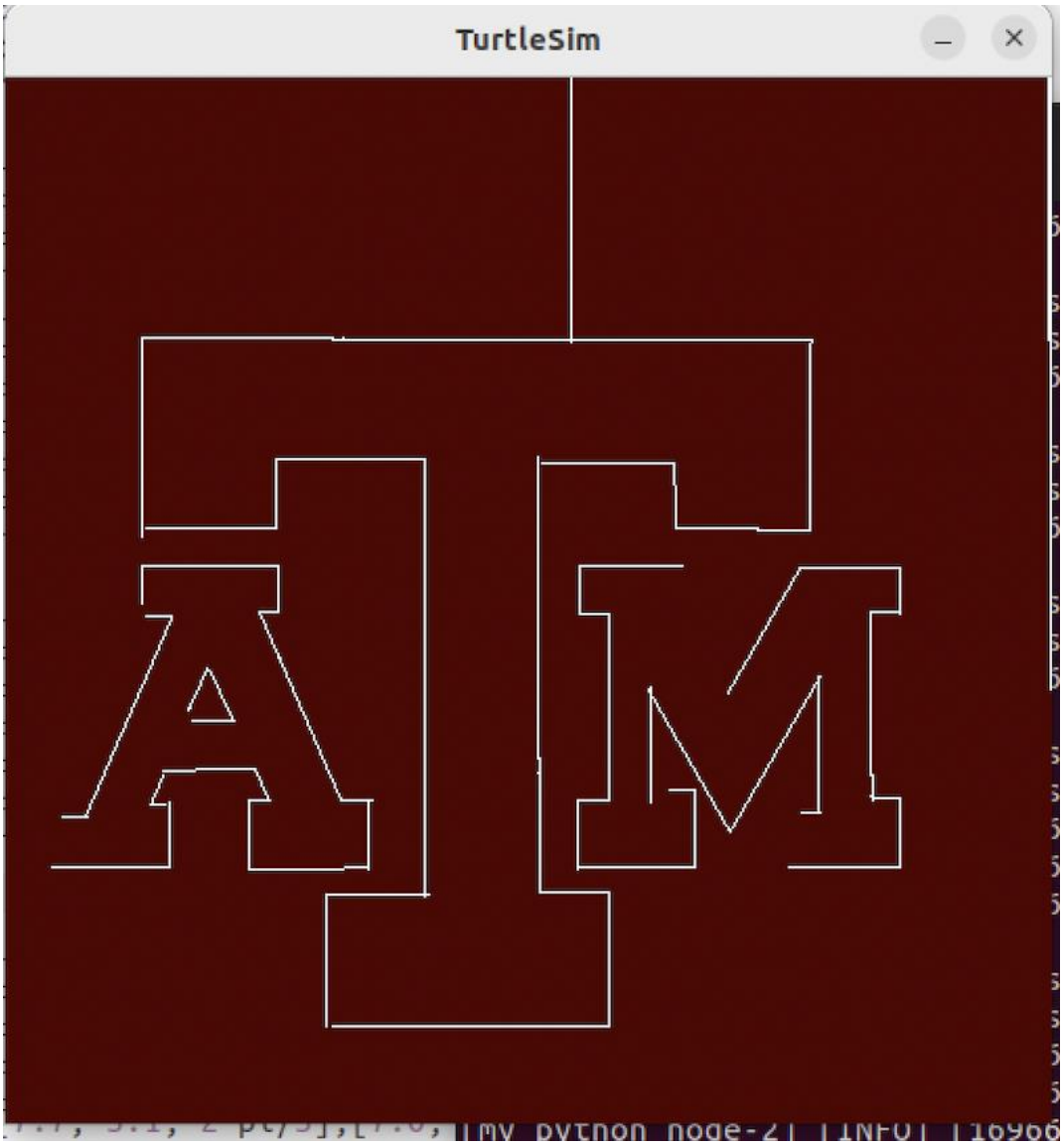
- The initial turtle positions and orientations are specified ((5.544445, 5.544445, initial_angle)).
- These values were likely chosen through experimentation and testing to achieve the desired movement behavior and drawing accuracy.

Question. Did the results meet your expectations?
Answer. There is high uncertainty in turtle's movements. Sometimes the logo is drawn nicely, sometimes not. Sometimes, the turtles go out of bounds. I put a lot of efforts to make it work.

With two turtles:



With ten turtles:


TurtleSim