GOALS 5 SUBMISSION

(i) a code specification/API/flowchart, as you developed at the Prep meeting, plus any changes.

DriveSystem.py

- **Motor Class**
  - Class: Controls one motor using two PWM GPIO pins.
    - __init__(self, io, pinA, pinB)
      - Sets up pins and initializes PWM.
    - setLevel(self, level: float)
      - Sets motor speed and direction.
    - stop(self)
      - Stops motor movement.

- **DriveSystem Class**

  - Class: Controls both left and right motors for movement.
    - __init__(self, io, left_pins: tuple[int, int], right_pins: tuple[int, int])
      - Creates left and right Motor objects.
    - drive(self, mode: str)
      - Drives with a named movement mode.
    - stop(self)
      - Stops both motors.

Sense.py

- **IR Class**

  - Class: Reads a single IR sensor.
    - __init__(self, io, pin: int)
      - Sets up one GPIO pin for input.
    - read(self) → int
      - Returns 1 if tape detected, 0 otherwise.

- **LineSensor Class**

  - Class: Reads three IR sensors as a unit.
    - __init__(self, io, pin_L: int, pin_M: int, pin_R: int)
      - Assigns pins for left, middle, right sensors.
    - read(self) → tuple[int, int, int]

        - Returns a 3-tuple of IR values.

AngleSensor.py

- **AngleSensor Class**
    - __init__(self, io)
        - Initializes the sensor interface (e.g., pigpio, I2C).
        - Sets up internal state for heading tracking and optional offset calibration.
    - readadc(self, address)
        - Reads an 8-bit integer (0–255) from the specified ADC channel (0 or 1).
    - read_angle(self) → float
        - Computes and returns the heading angle in degrees based on scaled ADC voltages (range: –180° to +180°).

---

street_behaviors.py

- **Behaviors Class**
    - __init__(self, io, drive, sensor, AngleSensor)
        - Initializes motor, sensor, and angle tracking interfaces. Sets time constants and state filters.
    - follow_line(self) → str
        - Follows a black tape line using IR feedback.
        - Returns "intersection" if triple detection, "end" if all sensors lose tape, or continues tracking.
    - pull_forward(self) → boolean
        - Drives forward briefly to ensure the robot fully enters an intersection before stopping.
        - **Change from last week: here we use an end detector (copy/pasted code) to check if there is a street in front of the intersection. If pull forward returns true then there is a street in front and it should be marked as unexplored if the function returns false then the street in front is marked as unknown.**
    - turning_behavior(self, direction: str) → int
        - Executes a left or right turn using angle feedback from AngleSensor.
        - Waits to leave and re-detect the tape before stopping.
        - Returns the net turn amount in 45° steps (±1, ±2, etc).

MapBuilding.py

- **STATUS (Enum)**

- Defines possible states for any street direction:
    - UNKNOWN: no information yet
    - NONEXISTENT: confirmed no street
    - UNEXPLORED: street detected but not yet driven
    - DEADEND: street ends without reaching another intersection
    - CONNECTED: street successfully traversed both directions

- **heading_to_delta (dict)**
    - Maps heading index (0–7) to (dx, dy) movement deltas

- **Class Intersection**
    - __init__(self, x, y)
        - Initializes a grid location with 8-directional streets set to UNKNOWN
        - Initializes cost to infinity and direction to None

- **Class Map**
    - **Attributes:**
        - x, y, heading: current robot pose
        - intersections: dictionary of discovered intersections
        - goal: destination coordinates
    - **Methods:**
        - pose(self)
            - Returns current pose as (x, y, heading)
        - calcmove(self)
            - Advances robot pose one unit in the direction of heading
        - calcturn(self, turn_amount)
            - Adjusts heading by turn_amount (modulo 8)
        - calcuturn(self)
            - Performs a U-turn (adds 4 to heading)
        - getintersection(self, x, y)
            - Returns existing or creates new Intersection at (x, y)
        - setstreet(self, x, y, heading, status)
            - Sets street status in both forward and reverse directions
        - markturn(self, turn_amount)

        - Updates intersection to mark skipped directions as NONEXISTENT, and facing direction as UNEXPLORED if it was UNKNOWN
    - update_connection(self)
        - Moves forward and marks current and next street directions as CONNECTED
    - markdeadend(self, behaviors)
        - Marks the current heading as a DEADEND, performs a U-turn, and attempts to return to previous intersection
    - showwithrobot(self)
        - Plots map and adds an arrow indicating current robot pose
    - show(self)
        - Plots intersections and their street statuses using matplotlib
    - dijkstra(self, xgoal, ygoal)
        - Implements Dijkstra's algorithm for pathfinding
        - Assigns higher cost to diagonal streets (cost √2 vs 1 for cardinal directions)
    - cleargoal(self)
        - Resets the goal and all intersection costs and directions
    - step_toward_goal(self, behaviors)
        - If heading matches direction to goal:
            - Follows line forward
            - Updates connection
            - Marks street as UNEXPLORED if a new one is found, else as DEADEND
        - Otherwise, turns toward goal and updates map accordingly

Brain.py

- Main Method
    - Function: prompt_and_load_map()
        - Prompts user for a .pickle filename
        - Tries to load and return a saved Map object
        - If loading fails, returns a new blank Map
        - Prompts for robot's current pose (x, y, heading)

Main Execution Block (__main__)

- Initialization
    - Connects to pigpio daemon
    - Instantiates:
        - DriveSystem
        - LineSensor
        - AngleSensor
        - Behaviors

- Prompts user to start with a blank map or load from file

User-Controlled Commands

- "left"/"right"
  - Performs turn using behaviors.turning_behavior

Updates heading and marks turn in map:

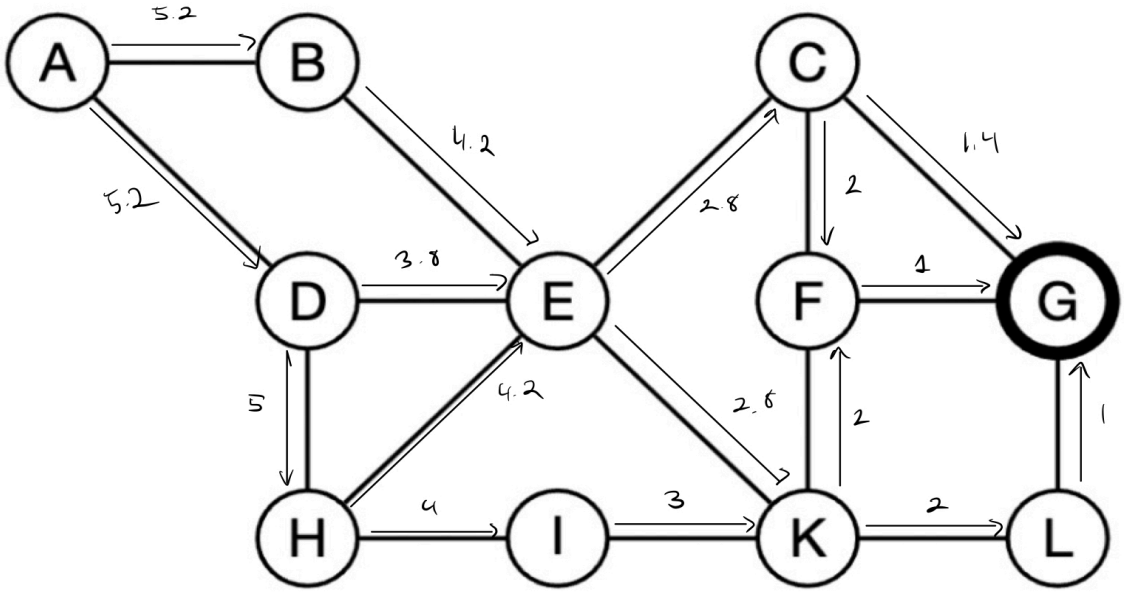
- "straight"
  - Follows line with behaviors.follow_line()
  - If result is:
    - "intersection":
      - Executes pull_forward() to fully enter intersection
      - Updates connection in map
      - Marks street as UNKNOWN if another street is detected, DEADEND otherwise
    - "end":
      - Calls map.markdeadend(behaviors)
    - Other: prints warning

- "goto"
  - Prompts for goal coordinates
  - Runs map.dijkstra() to compute path
- "save"
  - Prompts for filename and saves map to .pkl
- "load"
  - Re-runs prompt_and_load_map()

ii) the results of the by-hand Dijkstra challenge

Node Being Processed	**All Neighbors**	**Neighbors dropped**	**Resulting onDeck queue**
			G_0
G_0	$C_{1.4G}$,F_{1G},L_{1G}		$C_{1.4G}$,F_{1G},L_{1G}
$C_{1.4G}$	$E_{2.8C}$,$F_{2.4C}$,$G_{2.8C}$	$F_{2.4C}$,$G_{2.8C,}$	F_{1G},$L_{1G,}$$E_{2.8C}$
F_{1G}	C_{2F},G_{2F},K_{2F}	$C_{2F,}$$K_{2F}$	$L_{1G,}$$E_{2.8C,}$$K_{2F,}$
L_{1G}	$G_{2L,}$$K_{2L}$	$G_{2L,}$$K_{2L}$	$E_{2.8C,}$$K_{2F,}$
$E_{2.8C}$	$B_{4.2E,}$$C_{4.2E,}$$D_{3.8E,}$$H_{4.2E,}$$K_4$	$C_{4.2E,}$$K_{4.2E,}$	$K_{2F,}$$B_{4.2E,}$$D_{3.8E}$$H_{4.2E}$

	.2E,		
$K_{2F,}$	$E_{3.4K,}F_{3K,}I_{3K,}L_{3K}$	$E_{3.4K,}F_{3K,}L_{3K}$	$B_{4.2E,}D_{3.8E}H_{4.2E,,}I_{3K}$
$B_{4.2E,}$	$A_{5.2B,}E_{5.6B}$	$E_{5.6B}$	$D_{3.8E}H_{4.2E,}I_{3K,}A_{5.2B}$
$D_{3.8E}$	$A_{5.2D,}E_{4.8D,}H_{4.8D}$	$A_{5.2D,}E_{4.8D,}H_{4.8D}$	$H_{4.2E,}I_{3K,}A_{5.2B}$
$H_{4.2E}$	$D_{5.2H,}E_{5.6H,}I_{5.2H}$	$D_{5.2H,}E_{5.6H,}I_{5.2H}$	$I_{3K,}A_{5.2B}$
I_{3K}	$H_{4I,}K_{4I,}$	$K_{4I,}$	$A_{5.2B,}H_{4I,}$
$A_{5.2B}$	$B_{6.2A,}D_{6.6A,}$	$B_{6.2A,}D_{6.6A,}$	$H_{4I,}$
$H_{4I,}$	$D_{5H,}E_{5.4H,}I_{5H}$	$D_{5H,}E_{5.4H,}I_{5H}$	



(iii) a flowchart for the brain (allowing manual and autonomous driving).