Braitenberg Vehicles Project

Controls

- Left mouse button: add a robot at the current mouse position.
- Right mosue button: add a light source at the current mouse position
- Spacebar: toggle behavior of all active robots.

Command-Line Arguments

- `-r <num>`: Initialize world with `num` number of robots.
- `-l <num>`: Initialize world with `num` number of light sources.
- `--attract`: Make robots initially attracted to light sources.
- `--avoid`: Make robots initially avoid light sources.

Robot Class

This class contains the necessary methods to implement a basic Braitenberg vehicle. It has variable dimensions and can be attracted to or avoid light sources.

Data Members

- `int length, width`: The dimensions of the robot. These dimensions determine sensor positions, which in turn affect the robot's mobility.
- `int color`: The robot's ARGB color value (used for SDL rendering).
- `int heading`: The robot's orientation in degrees, with 0 defined as facing to the right edge of the window.
- `(vector<vector<double>>) lightMap`: A reference to the world's pre-calculated light values (2D grid of floating point values).

Methods

- `Robot(int x, int y, int length, int width, vector<vector<double>> lightMap, bool attract)`: Constructs a robot at the given coordinates with the given dimensions. *attract* sets robot behavior (i.e. attracted to or repelled by light sources).
- `int getHeading/Color/Length/Width()`: Standard getter functions; return robot properties used in rendering.

- `ivec getPos/Sensor1Pos/Sensor2Pos()`: Getter functions which return an integer column vector containing the robot's position (center coordinates) or the position of the requested light sensor.

- `vec readLightValues/getWheelValues()`: Getter functions which return a floating point column vector containing light sensor values (based on current sensor position) or wheel

values (based on current sensor reading and k-matrix).

- `void updateHeading()`: Applies a change to the robot's current heading in the following manner:
    1. Fetch the current wheel values and calculate their difference.
    2. Shift the heading by adding the difference times a constant value (can be positive or negative).
- `void move()`: Calculates the robot's new orientation (see above) and shifts the robot by calculating the horizontal and vertical components of the robot's heading times a constant magnitude (for simplicity). Note that the robot's position is calculated modulo the window dimensions, so robots wrap around screen boundaries.
- `int mod(int a, int b)`: Private helper function used for position calculations. Calculates `a mod b` (`a % b` returns negative results when `a` or `b` is negative).

World Class

This class contains and manages robots and light sources. It also draws robots and light sources to a window using SDL2.

Data Members

- `vector<vector<double>>` *lightMap*: The reference light map used for all robot sensor readings. Any new robot is given a handle to this light map upon its creation. Additionally, this structure is updated (recalculated) any time a new light source is added to the world.
- `vector<pair<int, int>>` *lightPositions*: Container holding the coordinates of all active light sources. This container is iterated over when the world's light intensity is calculated.
- `vector<Robot*> robots`: Structure used to track and update active robots.
- `double maxLightVal`: Contains the max light intensity of the world (100). Used as a reference for light rendering calculations.
- `bool attract`: Specifies robot behavior (can be toggled at any time).
- `SDL_Window* mainWindow`: 800x600 window used to draw robots and lights.
- `SDL_Texture* windowTexture`: Used as the main rendering target.
- `SDL_Renderer* renderer`: Responsible for updating `windowTexture` and drawing it in `mainWindow`.
- `uint* lightBuffer`: Contains a visual representation of the current light map values. Drawn on `windowTexture`.
- `SDL_Event event`: Used to capture SDL events (for handling keyboard/mouse interaction).

Methods

- `World(vector<pair<int, int>> lightPositions, bool attract)`: Creates a world initially containing lights specified in `lightPositions` and one robot with behavior specified by `attract`.
- `vector<vector<double>> getLightMap()`: returns a copy of the world's current light values.
- `void addRobot(int x, int y, int length, int width)`: inserts a robot with the given

dimensions at the given coordinates. The newly-created robot is added to the `robots` container.

- `void addLight(int x, int y)`: Adds a light source to the world at the given position. After the light is added, `lightMap` is recalculated.
- `void calcLightValues()`: Updates `lightMap` by updating each point light value by summing the Euclidean distance to each light source and dividing 100 over the sum.
- `void updateWorld()`: Moves each robot in `robots` and redraws the screen.
- `void initVideo()`: Initializes the SDL window and rendering.
- `void renderLightMap()`: Draws the light-map (and robots) on the screen.
- `void renderRobot(Robot* robot)`: Draws a pixel at `robot's` coordinates, and a pixel at each of `robot`'s sensor positions.
- `void setRobotAttraction(bool attract)`: Toggles behavior of all robots in the world.
- `void handleEvents()`: Polls and handles SDL mouse/keyboard events