Project Requirements and Background

Gurmehar S. Cheema, Drew Dudash, Chris Rosengrant, Lily Tran

Objective:

Team 11's project will simulate a scenario with an arson setting fires in a town and a firetruck reporting to put out said fires. The arson will set five fires while the firetruck will traverse the town in order to find and put out the fires.

Game board:

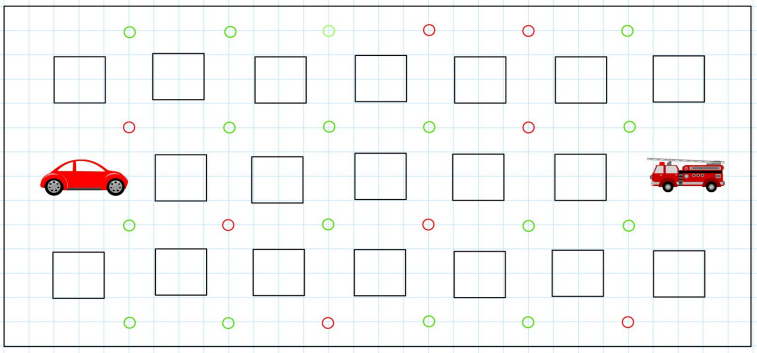


Figure 1: Concept of Game board

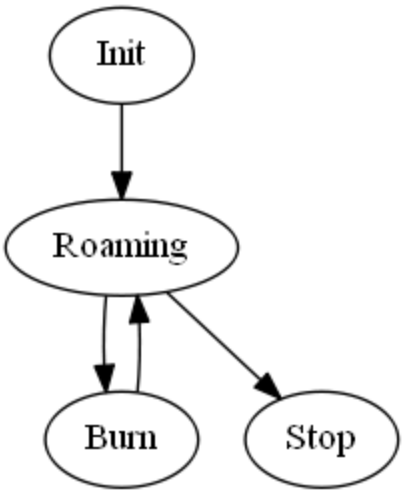
The game board for this scenario is very important. Since we are simulating the response of a firetruck in a particular town, the game board is set in stone as can be seen in *Figure 1*. This static positioning of obstacles (buildings), the two rovers (the firetruck and the arson), and the markers is used to model the static conditions of the city. The overall size of the gameboard is 2 meters by one meter. The buildings on this game board will be represented by the red brick cardboard boxes in the embedded lab. To represent roads, we used the empty space between the cardboard boxes. These roads will have a specified width of 18 inches. These roads will not have a specified direction for which traffic can traverse them. This will allow the rovers to traverse either way they would need and not be restricted by road laws. The tokens, represented by red and green circles in *Figure 1*, will be orientated to represent a center of an intersection. Since a Fire station will report to an intersection of two streets in an urban situation, these markers will represent the locations for which a fire could be set. For exact orientation of these markers, they will be placed 9 inches from every building if one was to measure diagonally from the corner of a building. The fire truck will be located on the far right side of the game board between two buildings to its left and right, one building directly in front of it, and nothing behind. The fire truck will start 9 inches from the building in front of it, and 9 inches from the buildings to its left and right.

Concept of Operation:

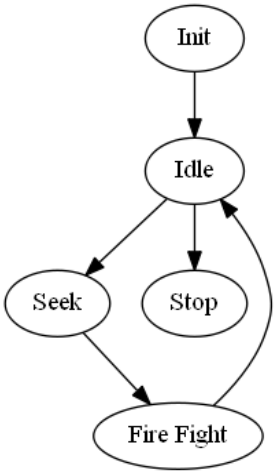
For this project, two Zumo Rovers will simulate fire fighting behavior. The first rover, the arsonist, will move around the game board to set fires. The second rover, the firefighter, will move around and put out fires. In order to set a fire, the arsonist will drive over a marker on the game board. A green marker represents a location which is firesafe and therefore the arsonist cannot set on fire. On the other hand, a red marker represents a location which can be lit on fire. The firefighter will poll the server until a location is available, then it will seek out the fire. The detailed behavior of the setup parameters, the detailed rover behavior, and the server protocol will be described in detail below.

Arsonist:

The arsonist begins in a setup state. In the setup state, the rover will initialize sensors and read in configuration data from the server. The configuration data tells the rover how many buildings to set on fire before it may shutdown. Once the setup is done, the arsonist will move into the roaming state. In the roaming state, the arsonist rover will move forwards and turn 90 degrees to its left and start to drive. As the rover drives, it will have a sensor facing directly down looking for a marker on the floor. When the rover finds either a red or a green marker, it will update the server as a notification that it had made it to the next street intersection. Along with this notification, the rover will have to check which color the marker is. If the marker is green, the rover will continue in the roaming state and keep looking for a red marker. If the rover detects a red marker is directly under the sensor, the rover will transition into the burn state. An important condition that the rover must also consider when finding a red marker is if it had already been to it. What this means is that before the rover can transition into the burn state, the arson must check against previous locations it has been to in order to check that this red marker is indeed unique. Once in the burn state, the arsonist rover will stop, to simulate the driver is setting the fire, and then after 5 seconds the arsonist rover will transition back into the roaming state. During the time the rover is stopped, the rover will send a message to the server with the location and update that specified marker had been set on fire. The rover will flash a red light while in the burn state. Once the rover moves from the burn state to the roaming state, the cycle will begin again. In order for the arsonist to transition into the stop state, the rover must come out of a burn state and check a conditional of how many unique markers it has been to. If that conditional proves to be true, the rover will then move into the stop state. This means that the arson will stop all movement and remain where it is.



Firefighter:



The firefighter will start in a initialization state where it will read configuration information from the server and initialize sensors. The configuration data is the amount of fires the fire fighter must put out before it can stop. This is nearly identical to the arsonist's init behavior. The major distinctions is that the firefighter will be oriented in the opposite direction as the arsonist at the start. After initializing, the firefighter automatically moves into the idle state. While in the idle state, the firefighter is waiting for the arsonist to reach a red marker and start a fire. When the arsonist emerges from its burn state and updates the server to a particular marker being set on fire, the fire truck will transition into the seek state. Once in the seek state, the fire truck will start by calculating the best route based on the location of the fire and how many markers it will need to cross in order to get to the fire. After calculating, the firetruck will then proceed out of the fire station until it will either take a 90 degree left or right turn depending on the routing calculation. The firetruck will also have a sensor on the front looking for red markers on the ground. Everytime a marker, no matter what color, is found on the path, the rover will send an update to the server that it has reached a new marker. This will keep the UI updated and will also assist with the rover traversing the city. Once at the location of the first fire that the arson set, the fire truck will transition into the fire fight state. In this state, the firetruck will come to a complete stop on top of the marker. After three seconds, the rover will transition back into the idle state. This would mean that the fire truck will remain at stop until the arson has reached another red marker and set another fire.

Traffic Light:

A traffic light system with a light, camera, and sensors will be suspended over one of the intersections. First, a general overview. Then, a detailed discussion of the configuration values. Finally, any other considerations.

The traffic light will oscillate between red and green over a configurable period. When the traffic light is green, a green light will be on, and nothing else observable will happen. When the traffic light is red, a red light will be on. While red, the firefighter, but not the arsonist, will route around the traffic light's intersection when attempting to locate the fire. If a rover moves through the intersection and is detected by the traffic light's sensors, a picture will be taken of the rover and sent to the external server. The rover must be within four inches of the center of the light to go off. The picture will not be a stream of pixel data. The picture will be a base64 encoded pixel BMP image with a configurable size. The image will be stored with the name of the rover closest to the intersection at the time the picture is taken. The name of the rover will only be accurate if the one rover is nine inches or more farther from the light than the other. The image must be uploaded before the camera is fired again. Cooldown details appear in the configuration section.

A small client for downloading the latest image from the database to a local machine will be made available. The image client will print a message if there are no images to retrieve. The image client is not expected to be able to run more than once a second. The image client may take upto but no more than 5 seconds to retrieve the image.

On startup, the traffic light system will download the following configuration values from the server.

- The period to oscillate in seconds. The observed period must be accurate to 0.5 seconds. The period to oscillate must be at least 2 seconds long.
- The cooldown period between pictures. The cool down exists to prevent the traffic light system from continuously going off. The observed period must be accurate to 0.5 seconds. The period must be at least 2 seconds long.
- The position of the camera on the grid. The correct position is required for the traffic light to work correctly. The position is discrete, so there is no need for margins.
- The width and height of the camera image in pixels. This width and height must be the same. Only 300x300 pixels is guaranteed to work.

Finally, the traffic light must be no more than 14 inches above the floor and no less than 12 inches above to work correctly.

Sound player:

The sound player's init state gets data from the server every 62.5 ms. The specific data it will get is the amount of fires and if the firefighter is in the same location of the fire. If there's at least one fire or the firefighter is in the same location it will go into the play state. In the play state it will check which data was true (are fires lit or a fire is being extinguished). If fires are lit then it will play the siren sound 5 seconds after getting the data. If a fire is being extinguished it will play the water sound 2 seconds after getting to the fire location. If there are fires and the fire is being extinguished, then the siren sound will stop and the water sound will play (water sound takes priority). The play state will go back to the init state when there are no more fires lit and the process will restart.

Server Description:

Pic32 boards will query the servers for configuration values they require to do their operations.

Previous Projects:

One project we found that is similar to ours is the project from Team Eight in Fall 2017. In order to keep track of objects the team used a pixicam to monitor their game board and keep track of their game objects. The data the team used from the pixicam included x and y coordinates, angles, width, height, and color code. The most important information that they used were the coordinates of the objects. In order to keep track of objects, the pixicam calculates the hues and

saturations of each pixel to recognize similar objects, however the hue of an object can easily be affected by light and exposure. As a result, the team decided to use different color codes that were placed on top of each of their game objects. The savior was yellow green yellow, the saboteur was blue yellow green, and the object being rescued was blue and yellow This solution allowed the pixicam to always distinguish between the three objects and allow data tracking for each object much easier.

Another similar design to our project can be seen in obstacle avoidance robots. Ultrasonic sensors are used for obstacle detection. As the robot moves it transmits ultrasonic waves and those waves are reflected when an object is ahead. However, as demonstrated in class no sensor is a perfect to match all of our needs. Obstacle avoidance robots also use other sensors such as an IR sensor for obstacle detection and proximity sensor for path detection. Similar to the ultrasonic sensor, an IR sensor detects objects by emitting a short ultrasonic burst and listening for an echo. The sensor then uses this echo to determine the relative location of object. The proximity sensors are used to determine if the robot is following the detected path. If the sensor detects that the object is starting to stray from the path it will adjust the robot's position accordingly, so that the robot remains on the correct path.

The next similar design to our project are Amazon's warehouse drones. The idea behind the drones is to deliver an Amazon shoppers' items from the warehouse once they place an order. The warehouse also serves as the location for restocking and refueling of the drones and Amazon products. All processes including order processing, delivery, refueling, and restocking are handled through a connection to a server that both the drones and warehouse are connected to. This design allows for automation of tasks since the server keeps track of any actions that needs to be done and notifies the appropriate object so that tasks can be completed.

List of References

J. Titcomb, "Amazon's flying warehouses to dispatch drone deliveries from the sky," *The Telegraph*, 29-Dec-2016. [Online]. Available: http://www.telegraph.co.uk/technology/2016/12/29/amazons-flying-warehouses-dispatch-drone-deliveries-sky/. [Accessed: 23-Jan-2018].

T. Agarwal, "Obstacle Avoidance Robotic Vehicle Using Ultrasonic Sensor for Obstacle Detection," *ElProCus - Electronic Projects for Engineering Students*, 29-Mar-2016. [Online]. Available: https://www.elprocus.com/obstacle-avoidance-robotic-vehicle/. [Accessed: 23-Jan-2018].

"Virginia Tech - ECE 4534 - Fall 2017 - Team 8 Final Video," *YouTube*, 14-Dec-2017. [Online]. Available: https://www.youtube.com/watch?v=5O6XlXXuigo. [Accessed: 22-Jan-2018].