Tiffany Hwu
205C Final Project

Efficient Bayesian Route Planning

Introduction

In a search and rescue scenario, a robot must navigate through a disaster area, identifying survivors to be rescued while preserving its own safety. Many tradeoffs must be calculated, such as whether to exploit a known area or explore a new area. Acquired knowledge about the safety levels of certain areas, likelihood of finding survivors, and urgency of attention for each survivor are all factors that must account into the route planning of the robot.

This applied scenario is essentially a foraging task in which beliefs about foraging sites and safety of areas can be modeled by Bayesian inference methods. In this project, I simulate small environments with different arrangements and configurations of foraging sites, and observe the effects of explorative and exploitative agents searching the environment.

Methods

The project was completed in two stages. The first focuses on getting the agent to effectively learn the changing danger levels in an environment. The second stage adds on the element of foraging.

Experiment 1: Inferring the safety levels of different regions of the map

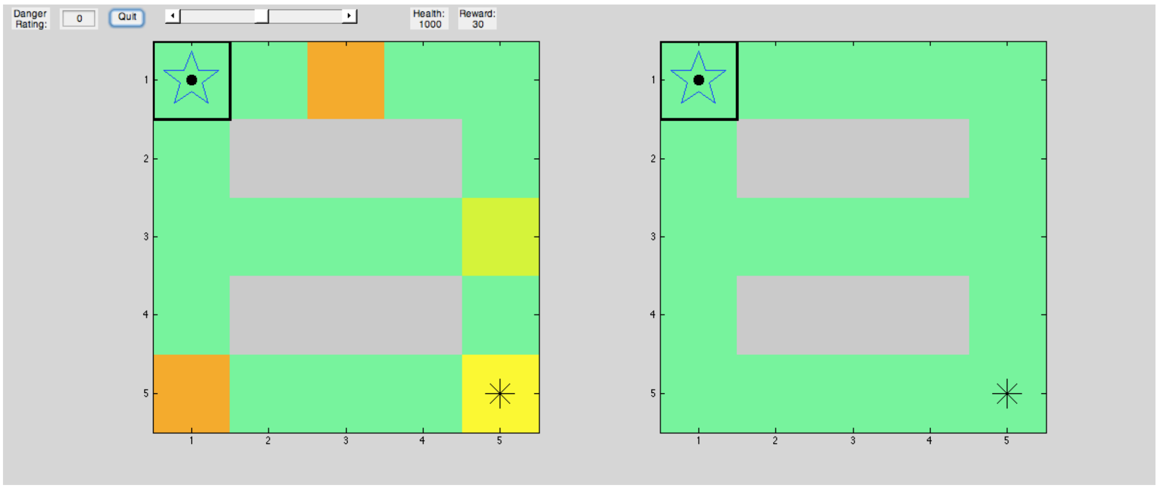


Figure 1

As seen in figure 1, the environment is simulated as a square 2D grid. Gray squares indicate off-limit areas, while any other colored area indicates a part of the valid path. Squares of increasing redness indicate the danger level of the area, a value that ranges 1-100. A random destination, marked by a star, is set for the agent, marked by a dot, who first starts out in the home square, marked by a dark outline and star pattern. The left side of the figure represents the true state of the world, whereas the right side represents the agent's beliefs of the world. As we see in the initial maps, the agent starts out with no knowledge of the dangers on the map, and is already aware of the entire physical layout and able to calculate multiple routes to reach the assigned destination.

Each step in the simulation has a defined sequence of events. First, all possible routes from the current location to the destination are calculated. Then, the accumulated danger of each route is found by summing the danger values of each square involved in the route. The agent takes a weighted sum of the distance and accumulated danger of each route and chooses the ideal route to take. It takes one step along this preferred route. In the case of this small map, the danger rating is multiplied by .1 to achieve a reasonable effect, such that a path of length 3 containing a danger spot of rating 40 would amount to a route desirability score of 7. In the new location, the agent samples the area for danger and updates its beliefs about the danger rating of that square. It also does the same for all neighboring locations of the current square. If the destination is now reached, a new random destination is assigned, and the process starts over. The sequence of random destination points allows the agent to roam the entire environment for our observation.

For each square, the agent has a belief value of the danger level of the square, which can range from 1 to 100. The prior belief of this value is represented as a normal probability distribution the danger levels as μ parameters.

Belief of μ is updated by Bayes rule using maximum a posteriori estimate. The prior is represented as a normal probability distribution as in the following equation. Initially μ is set to 0 and σ to 1.

$$p(\mu) = \frac{1}{\sqrt{2\pi\sigma_0^2}} e^{\frac{-(\mu-\mu_0)^2}{2\sigma_0^2}}$$

Likelihood is also represented as a normal distribution, with x as a new observation of danger rating.

$$p(x|\mu) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{\frac{-(x-\mu)^2}{2\sigma^2}}$$

Hypotheses about μ are updated using Bayes rule.

$$p(\mu|x) \propto p(x|\mu)p(\mu)$$

To calculate the maximum a posteriori estimate, we note that the likelihood is set to a distribution of N(μ,1) while the prior has a distribution of N(μ_0 ,σ_0). The product of two normal distributions results in another normal distribution with parameters μ and σ as

$$\left(\frac{\mu_0}{\sigma_0^2} + \frac{\sum_{i=1}^n x_i}{\sigma^2}\right) \Big/ \left(\frac{1}{\sigma_0^2} + \frac{n}{\sigma^2}\right),$$
$$\left(\frac{1}{\sigma_0^2} + \frac{n}{\sigma^2}\right)^{-1}$$

Thus, the mode of such a distribution is simply a calculation of μ. We can then plug in values to determine the updated belief. Setting σ_0 to 1 for simplicity and n to 1 (a memoryless strategy), we end up with a simple averaging of the old and new belief of danger.

$$\mu_{MAP} = \left(\frac{\mu_0}{\sigma_0^2} + \frac{\sum_{i=1}^n x_i}{\sigma^2}\right) \Big/ \left(\frac{1}{\sigma_0^2} + \frac{n}{\sigma^2}\right) = \left(\frac{\mu_0}{1^2} + \frac{x_i}{1^2}\right) \Big/ \left(\frac{1}{1^2} + \frac{1}{1^2}\right) = \frac{\mu_0 + x_i}{2}$$

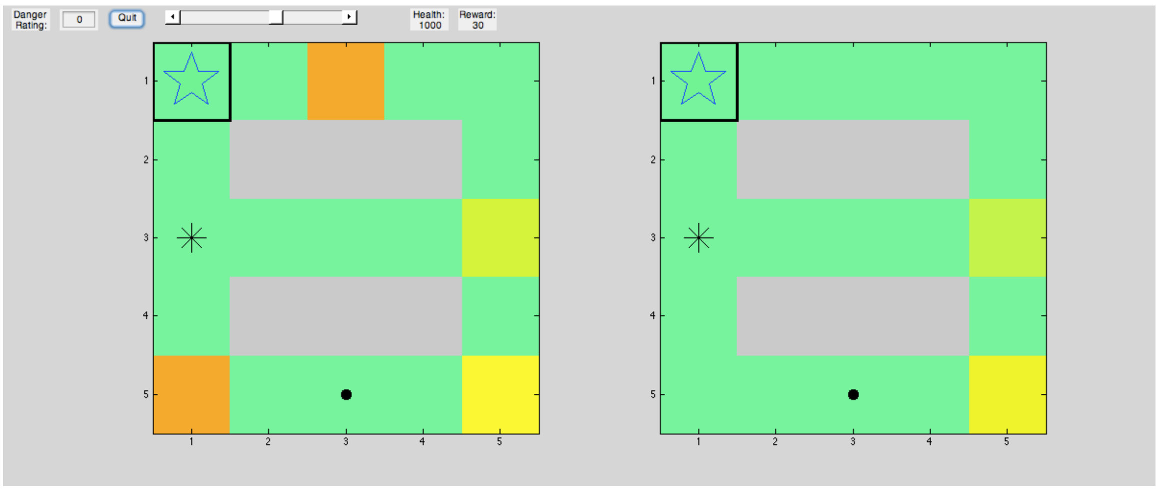


Figure 2

With this simple model of averaging, the agent is able to effectively learn the danger ratings of each area. Figure 2 shows the exploring agent at the initial stages of exploration. It has learned the danger ratings of a few areas.

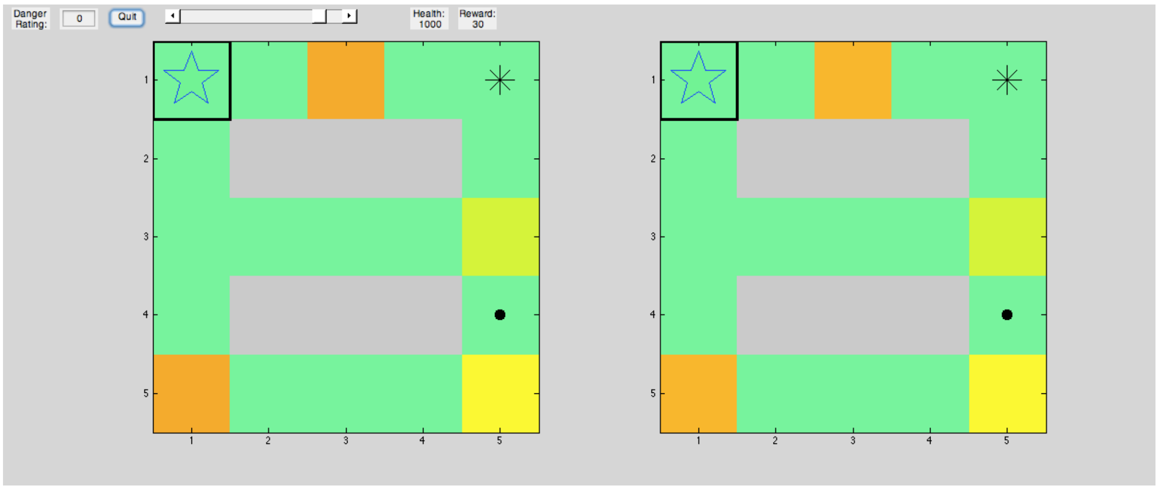


Figure 3

After more time has passed, the entire map is learned.

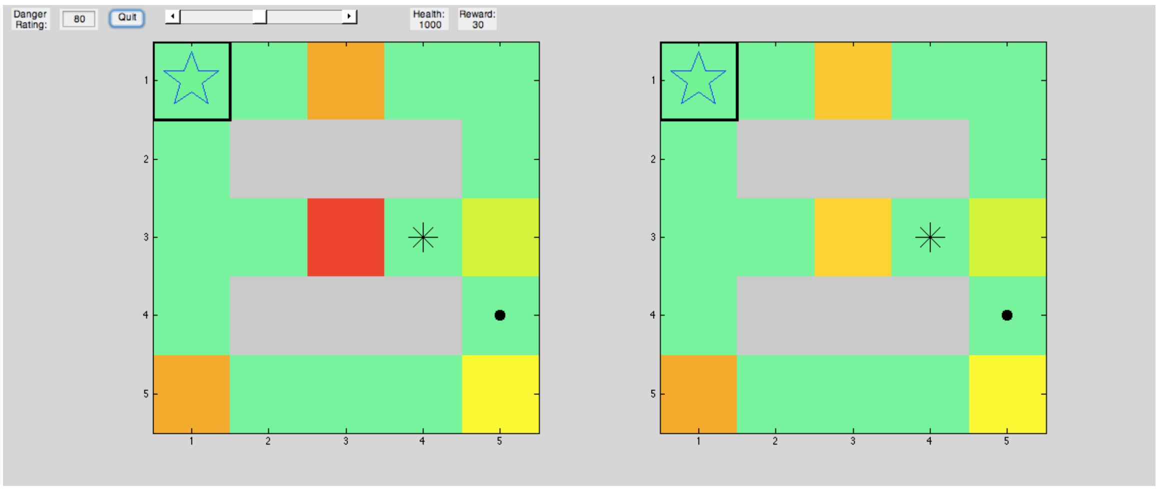


Figure 4

The agent can adapt to new dangers in the environment. The addition of a dangerous spot in the center is immediately detected, and will eventually be completely learned.

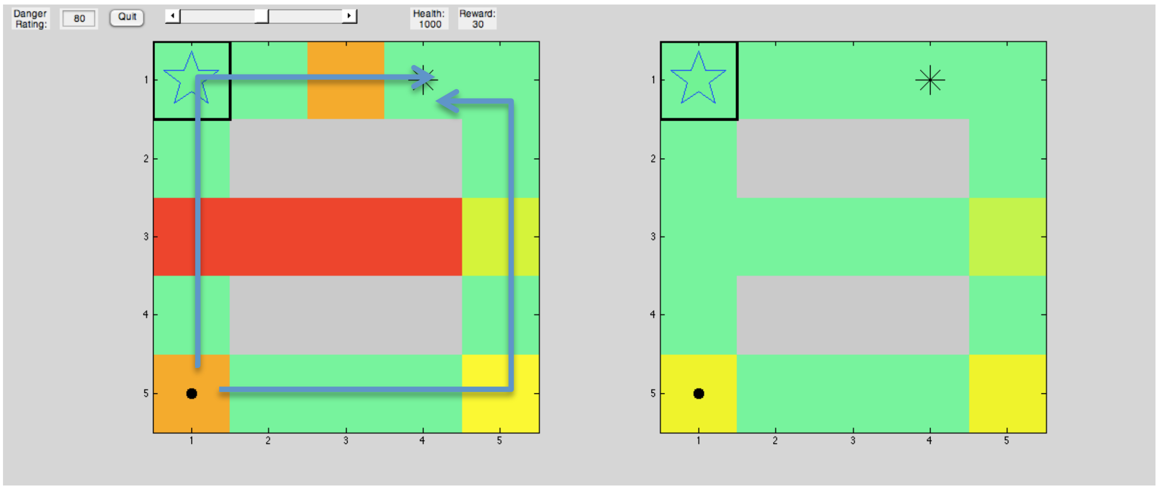

Danger Rating: 80 Health: 1000 Reward: 30

Figure 5

The agent also properly strategizes its route. In Figure 5, there are 2 reasonably short routes from the current location to the destination. After the agent has properly learned the danger mapping, it will take the slightly longer route in order to avoid the crossing the highly dangerous red strip. We can see that it effectively balances between distance and safety.

Experiment 2: Collecting resources on the map

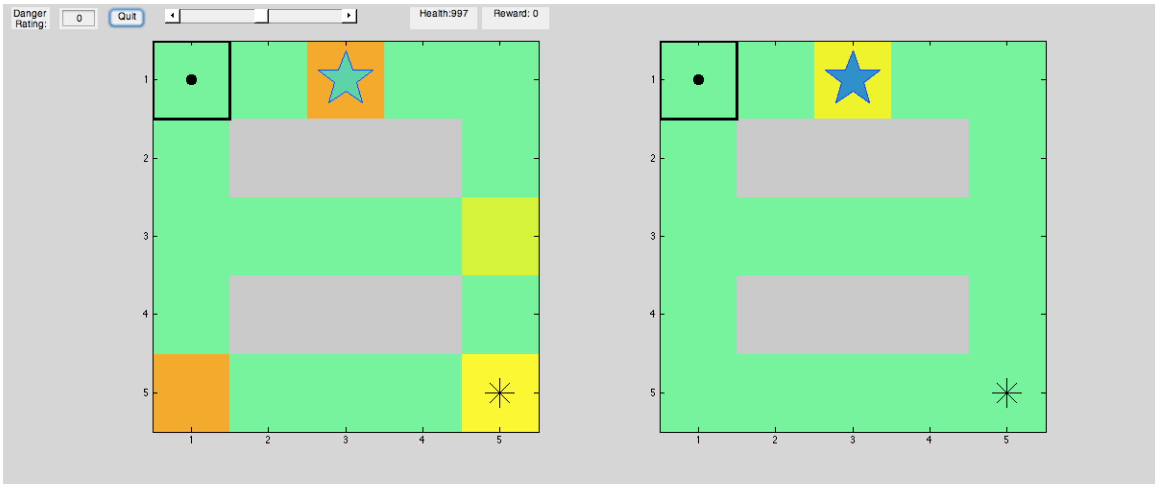

Danger Rating: 0 Health:997 Reward: 0

Figure 6

The next step is to add a foraging component to the search and rescue. The stars on the map now represent resources, with darker shades of blue representing higher reward values from a range of 1-100. Similar to the safety levels, the reward levels are sampled when the agent crosses a resource in its path.

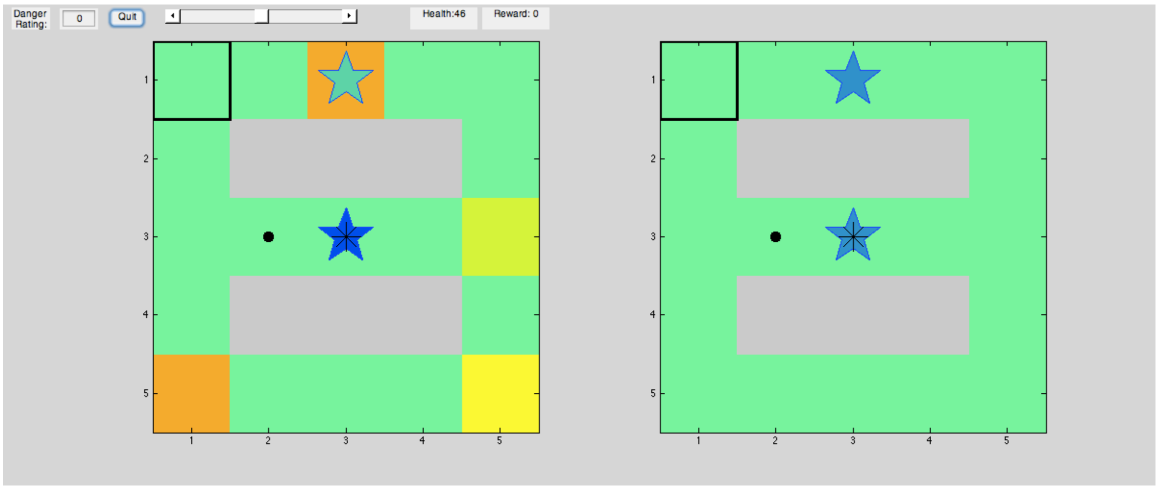


Figure 7

The agent's new goal is to pick its own new destinations, whether it is to go to a new foraging site or to return home. The agent now has a health value that gets decremented by 1 for distance and further decremented by a sampling of the danger in that area. If health reaches below zero, the agent dies and starts over at the home square, with no recollection of past observances. The agent can only restore health by returning to the home square, but must also strive to maximize foraged rewards. An exploration parameter is also added. This parameter directly influences 3 things: the prior belief of the resources having high reward, the minimum health buffer that the agent maintains before heading home, and the selection of destination. An explorative agent will simply pick a random resource to visit, whereas and exploitative agent will pick one that it has seen to provide the greatest reward.

Rather than a normal distribution, the rewards are sampled along an exponential distribution to ensure that all values are positive. The goal of the agent is then to learn the decay rate of λ for the exponential distribution of each square. For ease of computation, the conjugate prior of a gamma distribution is used to represent prior belief of λ.

$$p(\lambda) = \frac{\beta^{\alpha}}{\Gamma(\alpha)}\lambda^{\alpha-1}e^{-\beta\lambda}$$

The likelihood is represented by an exponential distribution

$$p(x|\lambda) = \lambda e^{-\lambda x}$$

Hypotheses of gamma are updated using Bayes rule

$$p(\lambda|x) \propto p(x|\lambda)p(\lambda)$$

Noting that our prior has a distribution of Gamma(α,β) and our likelihood has distribution Exponential(λ), the conjugate pair of distributions results in a posterior of Gamma as well, with α and β calculated as

$$\alpha + n,\ \beta + \sum_{i=1}^{n} x_i.$$

The mode of a Gamma distribution is

$$\frac{\alpha-1}{\beta}, \alpha >= 1 \quad .$$

Therefore, plugging in the α and β of the posterior into the mode equation results in the following:

$$\gamma MAP = \frac{\alpha+n-1}{\beta+\sum\limits_{i=1}^{n} x_i}$$

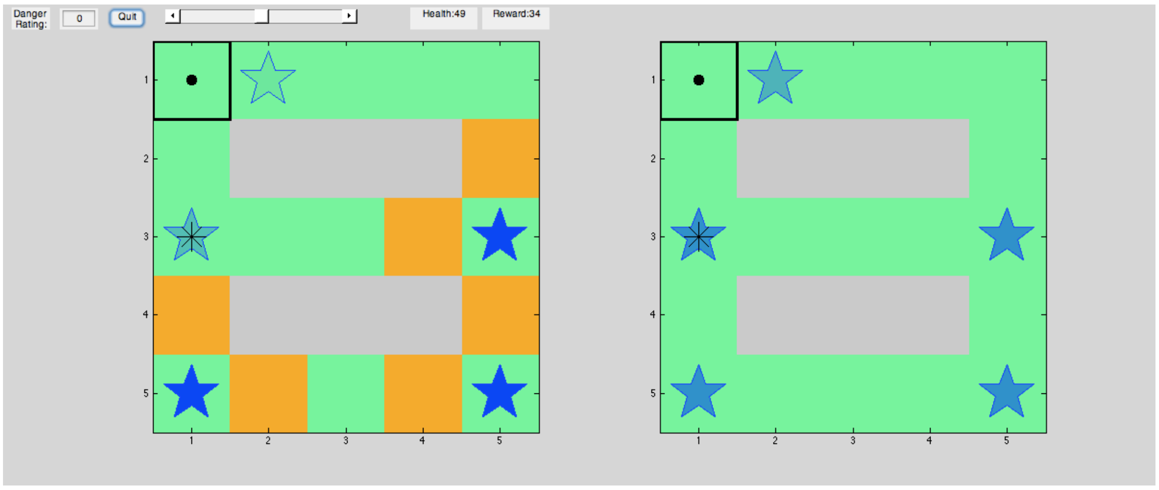
Exploration is a value ranging (0,1) that affects 3 parts of the decision-making process:

1) The initial value of α in the belief. A highly explorative agent will have an α that affects the shape of the distribution such that the mean is very high. The agent thinks all foraging locations are very promising.
2) The safety net value of health that the agent maintains before deciding to head home
3) Picking a destination. A random number is chosen in the range (0,1). If it is less than the exploration value, the agent will pick a random foraging site to head towards rather than just picking a site it knows is good.

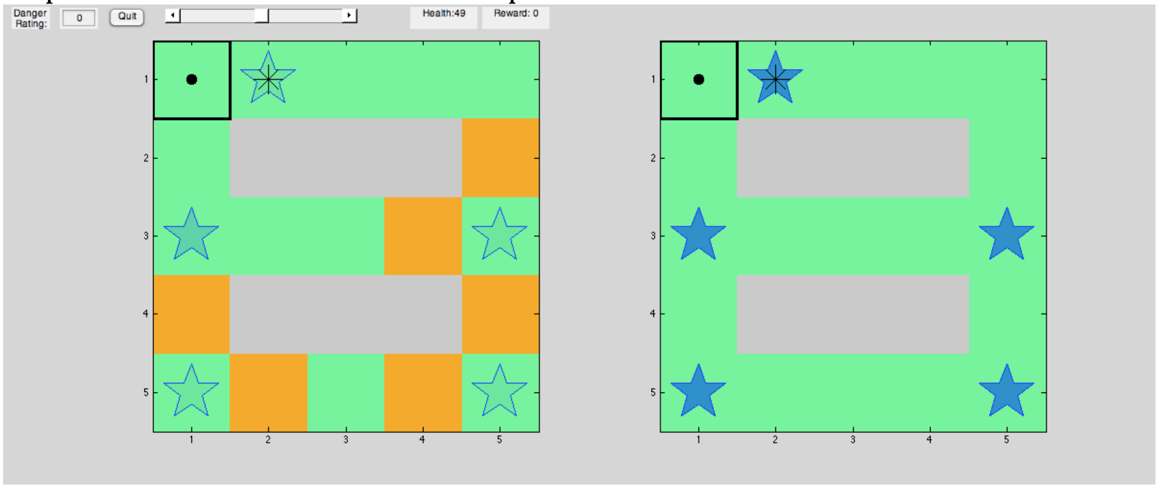
Results

For each map, 5 conditions of exploration parameter values are tested for a fixed and equal amount of time (50 steps). Then, for each map, the number of deaths occurred and reward earned in each life is recorded down.

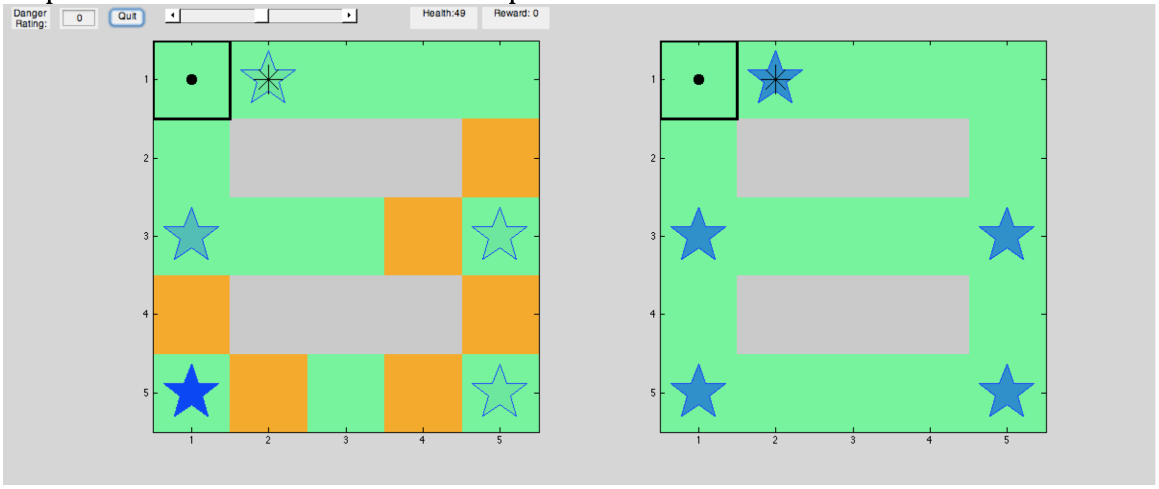
Map 1 is designed for the benefit of a more explorative agent. Foraging spots of very high value are hidden within dangerous areas that an explorative agent would be more likely to reach. Map 2 is designed more to the benefit of an exploitative agent, as the hidden rewards are of very low value. Map 3 is somewhere in between 1 and 2, with some hidden foraging spots worth a lot and some worth a little.



Map 1: Environment beneficial for exploration



Map 2: Environment beneficial for exploitation



Map 3: Something in between

Results for explore map

Exploration	# Lives Spent	Average reward per life
0.1	4	1159.5
0.3	4	**1806.5**
0.5	11	625.4545455
0.7	12	662.6666667
0.9	14	663.3571429

Results for exploit map

Exploration	# Lives Spent	Average reward per life
0.1	1	**1377**
0.3	3	782.6666667
0.5	8	289.5
0.7	11	230.7272727
0.9	9	222.6666667

Results for middle map

Exploration	# Lives Spent	Average reward per life
0.1	3	**1042.666667**
0.3	5	870.4
0.5	10	563.3
0.7	7	**775.1428571**
0.9	13	373.6923077

Average reward per life

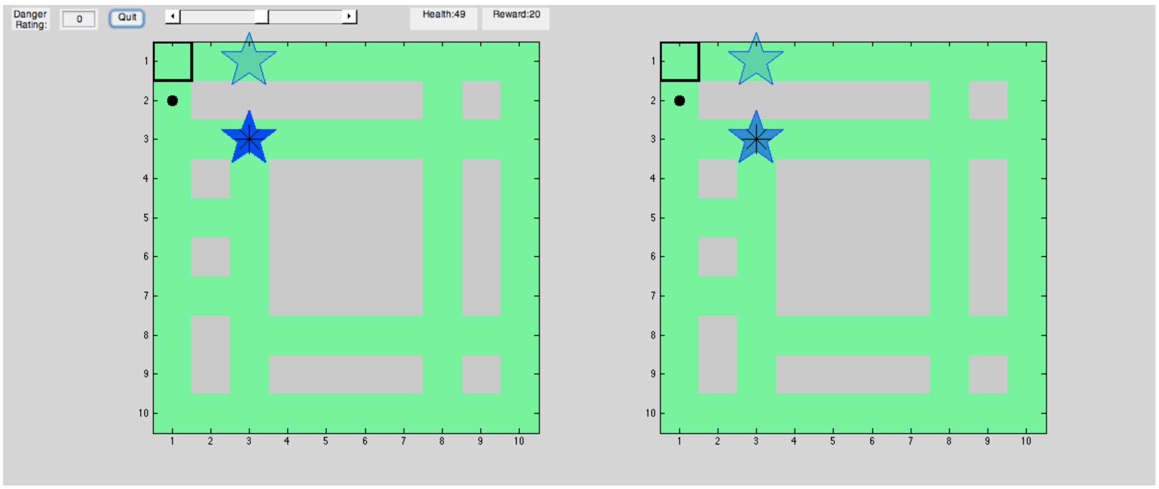
Exploration	Map Explore	Map Exploit	Map Middle
0.1	1159.5	**1377**	**1042.666667**
0.3	**1806.5**	782.6666667	870.4
0.5	625.4545455	289.5	563.3
0.7	662.6666667	230.7272727	**775.1428571**
0.9	663.3571429	222.6666667	373.6923077

Although more rigorous testing would be needed to make any conclusions, these results illustrate the difference between strategies of exploration and exploitation. To be expected, agents that are more explorative die more often. In general among all conditions, the explore map has more deaths and greater rewards while the exploit map has less deaths and less rewards than the middle map. When looking at the average rewards per life, the explore map preferred an exploration level of .3 while the exploit map preferred the lowest level of .1. The middle map also preferred .1 but had a secondary peak at .7.

<u>**Conclusion**</u>

The simulations show that Bayesian methods of updating beliefs about safety and reward are effective in allowing the agent to successfully navigate. To some degree, the arrangement of landscape requires the agent to change the extent to which it should be willing to explore and take risks.

Much more adjustment of the parameters in the simulation is necessary to view the detailed relationships between the exploration parameter and the map.
Another improvement that definitely needs to be addressed is the computational difficulty of calculating routes in a larger map. Originally, I had intended to generate larger maps representing Aldrich Park, but the slowness of route calculation made it unfeasible to run the simulation. Even a map such as the one below was infeasible for use:



Aldrich Map

This could potentially be solved by caching values, converting the graph to a sparse node format, or figuring out how to only calculate the top k desirable routes.

Provided that future simulations are successful, the next logical step is to transfer the model to a real-life environment and robot.

MATLAB Scripts

All scripts containing 'bayesianNavigation' can be used to start the simulation, each one containing a more developed version than the previous. Here I provide only bayesianNavigation3.m and key related functions. Feel free to contact me for the full code.

```
%Route Planning
%Iteration 3: Learns safety and resource values and tries to survive
%Runs route planning simulation

%Parameters: sigma, danger_wt, exploration, beta, start_health
%Variables: real grid, belief grid, location, destination
%0:neutral, >0:safe, <0:unsafe, NaN:not a valid location

function bayesianNavigation3(real_grid,home,real_resources)
    %Set parameters
    sigma = 1; %for safety prior
    beta = 1; %for resource prior
    danger_wt = .5;
    exploration = .5;
    alpha = exploration * 100; %for resource prior
    default_health = 50;

    %Set tracking variable
    rewards = [];

    %ui functions and variables
    speed = .2;
    quit = false;

    function set_real_grid(points)
        points = round(points);

real_grid(points(1,2),points(1,1))=str2double(get(danger_edit,'String')
);
    end
    function set_speed(sp)
        speed = sp;
    end
    function qt()
        quit = true;
        rewards = [rewards reward_counter]
    end

    %Set up figure/UI
    f = figure('Position', [100, 100, 1200, 500]);
    danger_edit =
uicontrol('Parent',f,'Style','edit','Position',[60,475,40,23],'String',
'0');
    danger_text =
uicontrol('Parent',f,'Style','text','Position',[10,475,40,23],'String',
'Danger Rating:');
    qt_ui =
uicontrol('Parent',f,'Style','pushbutton','Position',[110,475,40,23],'S
tring','Quit','Callback',@(~,~)qt());
    spd_ui =
uicontrol('Parent',f,'Style','slider','Position',[170,473,200,23],'Min'
,0,'Max',10,'SliderStep',[1 1]./10,...
        'Value',5,'Callback',@(h,~)set_speed(get(h,'Value')));
    health_text =
uicontrol('Parent',f,'Style','text','Position',[425,475,70,23],'String'
,strcat('Health: ',num2str(default_health)));
    reward_text =
uicontrol('Parent',f,'Style','text','Position',[500,475,70,23],'String'
,'Reward: 0');

    %Set real and belief grid
    orig_belief_grid = real_grid;
    orig_belief_grid(~isnan(real_grid)) = 0;
    belief_grid = orig_belief_grid;
    orig_belief_resources = real_resources(:,1:2);
    orig_belief_resources = [orig_belief_resources repmat([alpha/beta
alpha beta 0 0],size(real_resources,1),1)];
    belief_resources = orig_belief_resources;
    %{
    real_grid = [0 0 50 0 0; 0 NaN NaN NaN 0; 0 0 0 0 20; 0 NaN NaN NaN
0; 50 0 0 0 30];
    orig_belief_grid = [0 0 0 0 0; 0 NaN NaN NaN 0; 0 0 0 0 0; 0 NaN
NaN NaN 0; 0 0 0 0 0];
    belief_grid = orig_belief_grid;
    %Set home, real resources, belief resources
    home = [1 1];
    real_resources = [1 3 20; 3 3 80];
    orig_belief_resources = [1 3 alpha/beta alpha beta 0 0; 3 3
alpha/beta alpha beta 0 0];%need to keep track of mu, alpha, beta,
total observations, n
    belief_resources = orig_belief_resources;
    %}

    %Set health counter
    health_counter = default_health;
    %Set reward counter
    reward_counter = 0;
    %Set current location and destination
    curr_loc = home;
    dest_loc = new_dest2(belief_grid, belief_resources, curr_loc, home,
health_counter, exploration);

    %Loop through time until whenever
    while true
        if quit
            break
        end
        %plot and wait
        s1 = subplot(1,2,1);
        imgsc =
plot_grid(real_grid,curr_loc,dest_loc,home,real_resources);
        ax = imgca;

set(imgsc,'ButtonDownFcn',@(~,~)set_real_grid(get(ax,'CurrentPoint')),'
HitTest','on')
        s2 = subplot(1,2,2);
        plot_grid(belief_grid, curr_loc, dest_loc, home,
belief_resources);
        pause(1-speed/11);

        %change to check whether dead
        if health_counter < 0
            %record results
            rewards = [rewards reward_counter];
            %reset location back to home
            curr_loc = home;
            %restore health
            health_counter = default_health;
            if real_grid(curr_loc(1),curr_loc(2)) > 0
                health_counter = round(health_counter - 1 -
round(safety_observation/10));
            else
                health_counter = health_counter - 1;
            end
            set(health_text, 'String', strcat('Health:
',num2str(health_counter)));
```

```
            %reset rewards
            reward_counter = 0;
            set(reward_text, 'String', strcat('Reward:
',num2str(reward_counter)));
            %reset beliefs
            belief_resources = orig_belief_resources;
            belief_grid = orig_belief_grid;
            %set new dest
            dest_loc = new_dest2(belief_grid, belief_resources,
curr_loc, home, health_counter, exploration);
        else
            if isequal(curr_loc,dest_loc)
                %set new dest
                dest_loc = new_dest2(belief_grid, belief_resources,
curr_loc, home, health_counter, exploration);
            end;
            %restore health if reached home
            if isequal(curr_loc,home)
                health_counter = default_health;
                set(health_text, 'String', strcat('Health:
',num2str(health_counter)));
            end;
            %make observation and update belief grid
            safety_observation = randn +
real_grid(curr_loc(1),curr_loc(2));
            curr_belief = belief_grid(curr_loc(1),curr_loc(2));
            belief_grid(curr_loc(1),curr_loc(2)) =
(curr_belief+safety_observation)/2; %assume sigma = 1
            %make observation and update belief grid for neighbors
            [ neighbors ] = find_neighbors(curr_loc, belief_grid);
            for i = 1:size(neighbors,1)
                safety_neighbor_observation = randn +
real_grid(neighbors(i,1),neighbors(i,2));
                curr_belief =
belief_grid(neighbors(i,1),neighbors(i,2));
                belief_grid(neighbors(i,1),neighbors(i,2)) =
(curr_belief+safety_neighbor_observation)/2; %assume sigma = 1
            end
            %sample resource and update belief_resources
            if sum(ismember(real_resources(:,1:2), curr_loc, 'rows')) >
0
            %member =
sum(builtin('_ismemberoneoutput',real_resources(:,1:2),
curr_loc),2)==2;
            %if sum(member) > 2
                %resource_row = find(member);
                resource_row = find(ismember(real_resources(:,1:2),
curr_loc, 'rows'));
                resource_observation =
exprnd(real_resources(resource_row,3));
                belief_resources(resource_row,6) =
belief_resources(resource_row,6)+resource_observation;
                belief_resources(resource_row,5) =
belief_resources(resource_row,5)+belief_resources(resource_row,6);
                belief_resources(resource_row,4) =
belief_resources(resource_row,4)+belief_resources(resource_row,7);
                belief_resources(resource_row,7) =
belief_resources(resource_row,7) + 1;
```

```
                belief_resources(resource_row,3) =
round(1/(belief_resources(resource_row,4)-
1)/belief_resources(resource_row,5)+belief_resources(resource_row,6));
                reward_counter = round(reward_counter +
resource_observation);
                set(reward_text, 'String', strcat('Reward:
',num2str(reward_counter)));
            end
            %update health counter
            if real_grid(curr_loc(1),curr_loc(2)) > 0
                health_counter = round(health_counter - 1 -
round(safety_observation/10));
            else
                health_counter = health_counter - 1;
            end
            set(health_text, 'String', strcat('Health:
',num2str(health_counter)));
            %calculate routes
            routes = calculate_routes(curr_loc,dest_loc,belief_grid);
            %calculate safety belief of each route
            danger_ratings = zeros(1,size(routes,1));
            for i =1:size(routes,1)
                route = routes{i};
                idx = sub2ind(size(belief_grid), route(:,1),
route(:,2));
                danger_ratings(i) = sum(round(belief_grid(idx)./10));
            end
            %calculate distance of each route
            distances = zeros(1,size(routes,1));
            for i =1:size(routes,1)
                route = routes{i};
                distances(i) = size(route,1);
            end
            %calculate desired route based on safety and distance
            desirability = danger_ratings + distances;
            [b,ind] = sort(desirability,'ascend');
            %take a step in the desired direction
            curr_loc = routes{ind(1)}(2,:);
        end
    end
end

%input: current location, destination location, belief grid
%output: cell array with routes. a route of length n is a matrix of nx2
%with x coordinates in column 1 and y coordinates in column 2.
function [routes] = calculate_routes(curr_loc,dest_loc,belief_grid)
    %call the recursive function
    routes = calculate_routes_recursive(dest_loc,curr_loc,belief_grid);
end


%input: destination location, visited locations (empty at first
call),routes belief grid
%output: cell array with routes. a route of length n is a matrix of nx2
%with x coordinates in column 1 and y coordinates in column 2.
%using algorithm from http://stackoverflow.com/questions/58306/graph-
algorithm-to-find-all-connections-between-two-arbitrary-vertices
```

```
function [routes] = calculate_routes_recursive(dest_loc, visited, 
belief_grid)
    routes = {};
    nodes = find_neighbors(visited(size(visited,1),:),belief_grid);
    for i = 1:size(nodes,1)
        node = nodes(i,:);
        if ismember(visited,node,'rows')
            continue;
        end
        if isequal(node,dest_loc)
            visited = [visited; node];
            routes = [routes;visited];
            visited(size(visited,1),:) = [];
            break;
        end
    end
    for i = 1:size(nodes,1)
        node = nodes(i,:);
        %member = sum(builtin('_ismemberoneoutput',visited,
node),2)==2;
        if sum(ismember(visited,node,'rows')) || isequal(node,dest_loc)
        %if sum(member) || isequal(node,dest_loc)
            continue
        end
        visited = [visited; node];
        routes = [routes; calculate_routes_recursive(dest_loc, visited, 
belief_grid)];
        visited(size(visited,1),:) = [];
    end
end
```