Method

The program written is split into four main parts: running the experiment; the Markov Decision Process; reading an inputted map; and integration with Answer Set Prolog. The program takes as input a map, a start coordinate, and an end coordinate. As output the robot provides a map through each ‘room’ in the map, as well as if there was an error in mapping the rooms.

The map reading function reads an input map and splits it into individual rooms which are each assigned a number. ‘Doors’ (holes in the wall of the room) are then identified and used to assess how rooms are connected to other rooms. The function used for this experiment was only able to read rectangular rooms and horizontal doors which were not directly at the edge of the room. Rooms and doors could be of any size, assuming these limitations were adhered to.

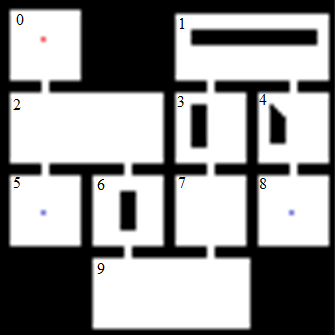
During initialisation of each room, a Markov Decision Process is used to create plans for each room of the map. Plans are created equal to the number of doors, with each plan using the centre of one of the room’s doors as a goal state. To calculate a plan, the MDP takes as input an array corresponding to the room, reward values for wall collision, reaching the goal state, and movement, and a value corresponding to probability of movement (in the case of this experiment, failure to move implies staying on the same spot). For each individual point on the map all surrounding points (up, down, left, and right of the point) are assigned a value equal to the initial value of the point (on first iteration, all points are zero) and all surrounding points, each of these multiplied by the movement probability. If the point is the goal point, the goal state reward is added. If any surrounding points are the goal state, reward is added equivalent to the goal state reward multiplied by the movement probability. This code is ran iteratively to receive a plan. All plans are created initially, and then one extra plan is created for the room in which the goal point is located, where the goal coordinate is set as the goal state.

For route planning a one-step look ahead method is utilised to assess expected reward of each movement. To perform this the function takes as input the plan, the start coordinate, and coordinates of any terminal states (in the case of this experiment, the only terminal state used was the goal state).To determine which direction to go, the value of the tile in a direction and all surrounding tiles, including the initial point are assessed. If a potential movement would be onto a wall tile, no value is added. The value for each direction is divided by the number of values added (normally by four, by if one tile would be onto a wall it would be divided by three instead) to normalise expected reward. The movement with the highest expected reward is then chosen to be the next location. Importantly, a location that has previously been selected can not be selected again, to prevent loops happening. As such, the next location can only move to any space not currently in the planned route. The process will run iteratively until either a route is found to the goal state, or the plan fails due to no more points being available for the route to select next. As a different MDP is constructed for each room on the map, even if one room fails to find a route, other rooms may still be able to solve a route.

To decided which rooms need to be routed through, Answer Set Programming is utilised. A template ASP file is used and rules are added too it based on the connections of the room. These rules are of the form ‘holds(adjacent(x1,x2),0).’. This rule represents that room x1 and x2 are adjacent and this holds true at time step 0. For the purpose of this experiment, rooms adjacent at time step 0 are assumed to always be adjacent. These adjacency rules are created automatically for each connection on the map. Importantly, rules must be available for adjacency between x1 and x2, and adjacency between x2 and x1, otherwise movement can only happen in the direction specified. Two other rules are added to the ASP file, which are the start room and the goal room. These are of the form: ‘holds(at(r,x1),0).’, and ‘goal(I) :- holds(at(r,x2),I).’. The first rule means robot is at x1 at time 0. The second rule means the goal is achieved at time I, if robot is at x2 at time I. The ASP file is then run to try and solve the optimal route. To optimise solving times, a maximum number of timesteps allowed is used. This starts at 1 and increases by 1 until a solution is found. If no solution is found after this number is larger than the number of rooms in the map, an error is thrown. Assuming a route is found, the robot then uses the path planning for each room described above to determine a route to the final goal state.

To run the experiment, required inputs are a map, a start coordinate, a goal coordinate, rewards for each action (as described above) and number of iterations for the MDP. For performing the experiment, the independent variables used will be the possible reward values, the goal point, and the number of iterations of the MDP. The ‘hit wall’ reward value will be set as -100, -10, 0, or 10. The ‘goal’ reward value will be set as 1, 10, 100, 1000, or 10000. The ‘movement’ reward value will be set as -100, -10, -1, 0, or 1. There will be two different possible goal points, as demonstrated in figure 1. The number of MDP iterations will be set as 1, 10, 100, and 1000

Figure 1. The red dot represents the start point. Blue dots represent the two possible goal points. The map here is the map used for the experiment. The numbers in the top left represent the room number, as is categorised by the program.



The dependent variable used will be the total length of the route for each combination of independent variables. This will be used to determine the optimal combination of reward values, as well as number of MDP iterations. This will be determined by seeing which combination has the lowest path for both potential paths. This is because it is important to determine whether the best plan is the same for both possible routes.