AI Assignment

threshold(150).

% Room locations

room(r117, c117).

room(r101, c101).

room(r103, c103).

room(r105, c105).

room(r107, c107).

room(r109, c109).

room(r111, c111).

room(r113, c113).

room(r115, c115).

room(r119, c119).

room(r121, c121).

room(r123, c123).

room(r125, c125).

room(r127, c127).

room(r129, c129).

room(r131, c131).

room(canteen, cc118).

room(stairs, cs1).

% Door connection room to corridor

door(c117, r117).

door(cs1, stairs).

door(r101, c101).

door(c103, r103).

door(c105, r105).

door(c107, r107).

door(c109, r109).

door(c111, r111).

door(c113, r113).

door(c115, r115).

door(cc118, canteen).

door(c119, r119).

door(c121, r121).

door(c123, r123).

door(c125, r125).

door(c127, r127).

door(c129, r129).

door(c131, r131).

% door statuses

doorStatus(c0, open).

doorStatus(c117, open).

doorStatus(cs1, closed).

doorStatus(c101, closed).

doorStatus(c103, closed).

doorStatus(c105, closed).

doorStatus(c107, closed).

doorStatus(c109, closed).

doorStatus(c111, closed).

doorStatus(c113, closed).

doorStatus(c115, closed).

doorStatus(cc118, closed).

doorStatus(c119, closed).

doorStatus(c121, closed).

doorStatus(c123, closed).

doorStatus(c125, closed).

doorStatus(c127, closed).

doorStatus(c129, closed).

doorStatus(c131, closed).

% South Corridor

connected(c0,cs1,open).

connected(cs1,c101,open).

connected(c101,c103,open).

connected(c103,c105,open).

connected(c105,c107,open).

connected(c107,c109,open).

connected(c109,c111,open).

% West Corridor

connected(c109,c113,open).

connected(c113,c115,open).

connected(c115,c117,open).

connected(c117,c118,open).

connected(c118,cc118,open).

% North corridor

connected(c118,c119,open).

connected(c119,c121,open).

connected(c121,c123,open).

connected(c125,c127,open).

connected(c127,c129,open).

connected(c129,c131,open).

% East Corridor

connected(c131,c132,open).

connected(c132,c133,open).

connected(c133,c0,open).

% Master Key location

key\_location(r117).

robot\_location(c0).

robot\_power(150).

% Task: get\_key

task(get\_key, []).

% Task: deliver\_coffee\_to\_paul

task(deliver\_coffee\_to\_paul, [canteen, r101]).

door\_status(Pos, Status) :-

robot\_location(RobotLoc), % Get the current robot location

door(Pos, OtherPos), !,

door\_status\_check(Pos, OtherPos, RobotLoc, Status).

door\_status(Pos, Status) :-

robot\_location(RobotLoc), % Get the current robot location

door(OtherPos, Pos), !,

door\_status\_check(OtherPos, Pos, RobotLoc, Status).

door\_status(\_, closed). % Default case for any undefined doors

door\_status(r117, open) :- key\_location(\_). % Specific case for r117

door\_status(\_, closed). % Default case

% Helper predicate for checking door status

door\_status\_check(Pos1, Pos2, RobotLoc, open) :-

door(Pos1, Pos2),

(doorStatus(Pos1, open); doorStatus(Pos2, open); (robot\_can\_open\_doors, robot\_location(RobotLoc))).

% predicate to determine if the robot can open doors

robot\_can\_open\_doors :-

robot\_power(Power),

threshold(Threshold),

Power >= Threshold.

connected\_to(Pos, Pos, \_, \_) :-

write('Reached destination: '), write(Pos), nl, !. % Base case

connected\_to(Pos1, Pos2, Visited, Depth) :-

Depth > 0,

NewDepth is Depth - 1,

(door(Pos1, NextPos); door(NextPos, Pos1)),

\+ member(NextPos, Visited),

door\_status\_check(Pos1, NextPos, DoorStatus),

write('Checking connection from '), write(Pos1), write(' to '), write(NextPos),

write(' with door status '), write(DoorStatus), nl,

DoorStatus = open,

connected\_to(NextPos, Pos2, [NextPos | Visited], NewDepth).

move(RobotLoc, NewLoc, Cost) :-

write('Trying to move from '), write(RobotLoc), write(' to '), write(NewLoc), nl,

door\_status(RobotLoc, DoorStatus),

connected\_to\_with\_limit(RobotLoc, NewLoc, [], 100),

door\_status(NewLoc, NextDoorStatus),

write('Door status from '), write(RobotLoc), write(' is '), write(DoorStatus), nl,

write('Door status to '), write(NewLoc), write(' is '), write(NextDoorStatus), nl,

DoorStatus = open,

NextDoorStatus = open,

calculate\_cost(RobotLoc, NewLoc, Cost),

deduct\_power(Cost),

retract(robot\_location(RobotLoc)),

assert(robot\_location(NewLoc)),

write('Moved to '), write(NewLoc), nl,

format('Calculated cost: ~w~n', [Cost]).

connected\_to\_with\_limit(X, Y, Visited, Depth) :-

write('Checking path from '), write(X), write(' to '), write(Y), write(' with depth '), write(Depth), nl,

(connected\_to(X, Y, Visited, Depth) ->

write('Path found from '), write(X), write(' to '), write(Y), nl;

write('No path found from '), write(X), write(' to '), write(Y), nl, fail).

calculate\_cost(Pos1, \_, Cost) :- door\_status(Pos1, open), Cost is 1.

calculate\_cost(Pos1, \_, Cost) :- door\_status(Pos1, closed), Cost is 2.

deduct\_power(Cost) :-

robot\_power(CurrentPower),

NewPower is CurrentPower - Cost,

retract(robot\_power(CurrentPower)),

assert(robot\_power(NewPower)),

write('Power deducted: '), write(Cost), nl,

write('Current robot charge: '), write(NewPower), nl.

test\_move\_c0\_to\_cs1 :-

robot\_location(c0),

robot\_power(Power),

write('Initial robot location: c0'), nl,

write('Initial robot power: '), write(Power), nl,

move(c0, cs1, Cost),

write('Move cost: '), write(Cost), nl,

robot\_location(NewLocation),

write('New robot location: '), write(NewLocation), nl.

start\_task([Task | RestTasks]) :-

robot\_location(InitialLocation),

robot\_power(Charge),

write('Executing start\_task...'), nl,

write('Initial location: '), write(InitialLocation), nl,

write('Robot charge: '), write(Charge), nl,

write('Executing task: '), write(Task), nl,

perform\_tasks([Task | RestTasks], InitialLocation).

perform\_tasks([], CurrentLocation) :-

write('Final location: '), write(CurrentLocation), nl.

perform\_tasks([Task | RestTasks], CurrentLocation) :-

execute\_task(Task, CurrentLocation, NewLocation),

perform\_tasks(RestTasks, NewLocation).

execute\_task(get\_key, CurrentLocation, NewLocation) :-

write('Executing get\_key task from '), write(CurrentLocation), nl,

key\_location(KeyLoc),

write('Key is at '), write(KeyLoc), nl,

move(CurrentLocation, KeyLoc, \_),

NewLocation = KeyLoc,

write('Got the key!'), nl.

execute\_task(deliver\_coffee\_to\_paul, CurrentLocation, NewLocation) :-

task(deliver\_coffee\_to\_paul, [Destination | \_]),

move(CurrentLocation, Destination, \_),

NewLocation = Destination,

write('Coffee delivered to Paul at '), write(Destination), nl.