The 2D world modelled here is a grid of size 6x6 tiles, that will be used to demonstrates the ability of three different agents in an Environment. The Environment is a 'Shopping Mall' in which an agent, which is a 'Woman' is trying to locate her friend and pick her up. The final target of the agent is to pick up her 'Friend'. The Environment also includes obstacles that make it difficult for the agent to achieve its target. The obstacle in the current world is 'Fire'. For every tile that contains Fire, its adjacent tiles are indicated as 'Heat'.

The placements of the obstacles (Fire) and the goal (Friend) is completely randomised for all the agent-environment instances.

The three different agent types that were implemented are:

- Simple Reflex Agent

- Model Based Agent

- Goal Based Agent

Additionally, a small discussion on how a utility-based agent would perform in the same environment is also included.

The Environment can be randomised with respect to its size,

### The code for the search algorithm was used from the AIMA file search4e.ipynb

def depth\_limited\_search(problem, limit=10):

"Search deepest nodes in the search tree first."

frontier = LIFOQueue([Node(problem.initial)])

result = failure

while frontier:

node = frontier.pop()

if problem.is\_goal(node.state):

return node

elif len(node) >= limit:

result = cutoff

elif not is\_cycle(node):

for child in expand(problem, node):

frontier.append(child)

return result

w4.delete\_thing(Rachel)

class Woman(Agent):

holding = []

killed\_by = ""

direction = Direction("right")

def can\_pick(self, thing):

"""Woman can only pickup friend"""

return thing.\_\_class\_\_ == Friend

Rachel = Woman(program)

w4.add\_thing(Rachel, (1, 1), True)

w4.add\_thing(Friend(), (8,8), True)

DLS = path\_states(depth\_limited\_search(s1,limit = 20))

path3 = list(DLS)

def draw\_grid(world):

global grid

grid[:] = (123, 234, 123)

for x in range(0, len(world)):

for y in range(0, len(world[x])):

if len(world[x][y]):

grid[y, x] = color[world[x][y][-1].\_\_class\_\_.\_\_name\_\_]

def run(steps=50, path = path, delay=1):

"""Run the Environment for given number of time steps,

but update the GUI too."""

for step in range(steps):

global grid, w4

draw\_grid(w4.get\_world()[0])

grid.show()

if w4.is\_done():

break

w4.step(path)

if len(path3) > 1:

path = [list(x) for x in path3][1:]

else:

path = path3

print(path)