COMP 4200/ COMP5430

BERKELEY SEARCH Q5: Corners Problem.

```
class CornersProblem(search.SearchProblem):
    """
    This search problem finds paths through all four corners of a layout.

    You must select a suitable state space and successor function
    """

    def __init__(self, startingGameState):
        """
        Stores the walls, pacman's starting position and corners.
        """
        self.walls = startingGameState.getWalls()
        self.startingPosition = startingGameState.getPacmanPosition()
        top, right = self.walls.height-2, self.walls.width-2
        self.corners = ((1,1), (1,top), (right, 1), (right, top))
        for corner in self.corners:
            if not startingGameState.hasFood(*corner):
                print 'Warning: no food in corner ' + str(corner)
        self._expanded = 0 # DO NOT CHANGE; Number of search nodes expanded
        # Please add any code here which you would like to use
        # in initializing the problem
        "*** YOUR CODE HERE ***"
```

Task is to do a search that visits all four corners of the maze.
What is needed for problem representation?

COMP 4200/ COMP5430

BERKELEY SEARCH Q6: Corners Problem Heuristic

What are some consistent heuristics for corners problem (visiting all four corners)?

In other words: given Pacman's position and knowing which corners remain to visit, what is a quick estimate of how many moves *must* be required to finish the solution? (optimistic best possible case)

BERKELEY SEARCH Q7: Eating All The Dots

```
def foodHeuristic(state, problem):
  """
  Your heuristic for the FoodSearchProblem goes here.
  The state is a tuple ( pacmanPosition, foodGrid ) where foodGrid is a Grid
  (see game.py) of either True or False. You can call foodGrid.asList() to get
  a list of food coordinates instead.

  If you want access to info like walls, capsules, etc., you can query the
  problem.  For example, problem.walls gives you a Grid of where the walls
  are.

  If you want to *store* information to be reused in other calls to the
  heuristic, there is a dictionary called problem.heuristicInfo that you can
  use.
  """
  position, foodGrid = state
  "*** YOUR CODE HERE ***"
  return 0
```

What are some consistent heuristics for finding all of the food?

In other words—what is a quick estimate of the minimum number of moves that must be required to solve the problem?