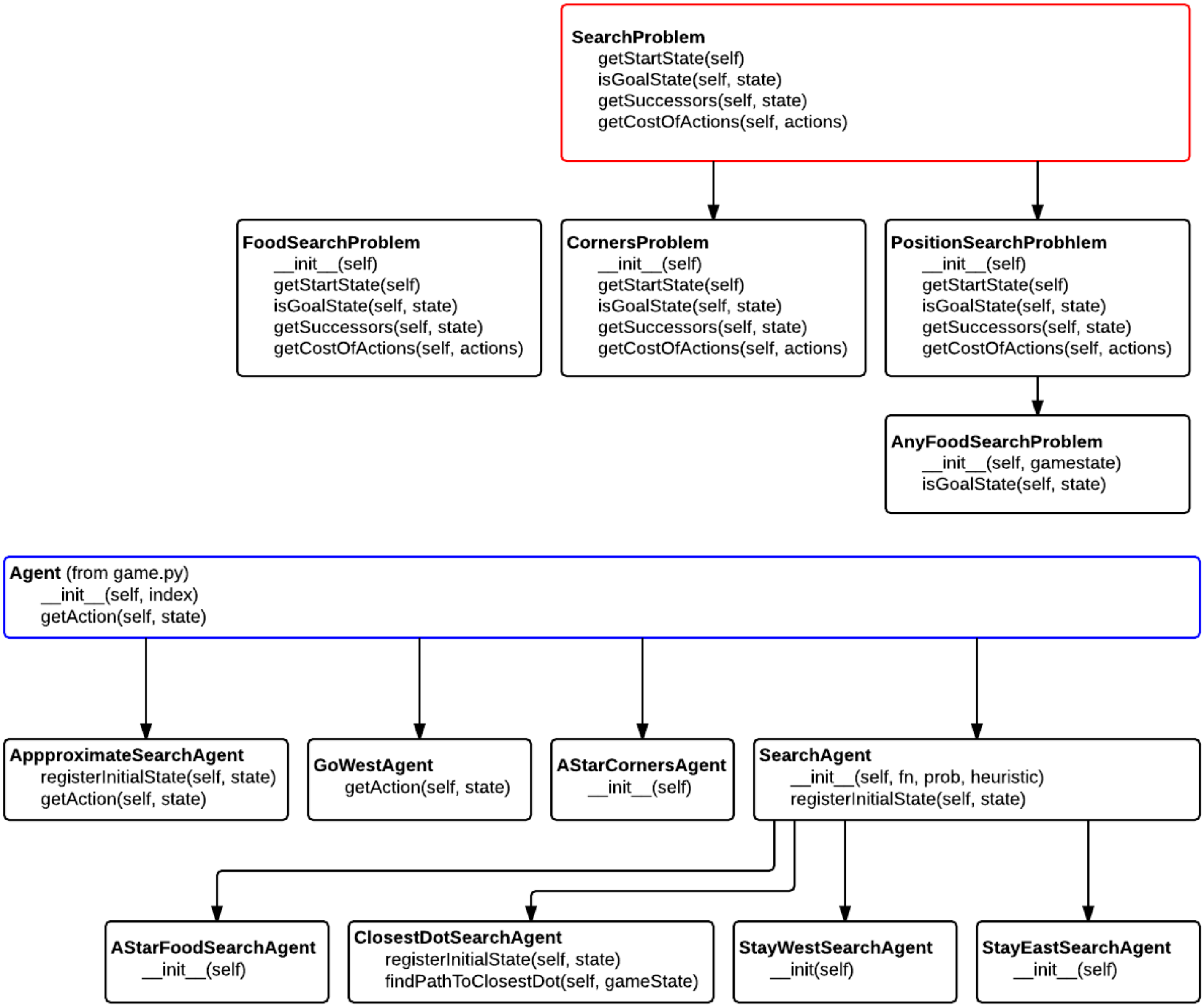
CSSE413-02
Adam Michael

Structure of `search.py` and `searchAgents.py`

Classes from `search.py` are red, classes from `searchAgent.py` are in black. Classes from other files are in blue.



Descriptions of class procedures

SearchProblem.getStartState – returns the initial state of the problem

SearchProblem.isGoalState – returns true if and only if a given state is a valid goal state

SearchProblem.getSuccessors – returns a list of triples of state, action and stepCost where the state is the successor o the current state, the action is the action rqeuired to get there and stepCost is the cost of reaching that successor from the current state

SearchProblem.getCostOfActions – input is a list of valid actions and output is the total cost of that sequence of actions

Agent.getAction – input is a GameState and output is an action from Directions.{North, South, East, West, Stop}

SearchAgent.registerInitialState – creates a new search problem with an initial state and finds a path of actions

SearchAgent.getAction – returns the next action in the path of actions that was determined in registerInitialState

PositionSearchProblem.getStartState – self-explanatory

PositionSearchProblem.isGoalState – checks if has reached goal state and takes appropriate display actions if so

PositionSearchProblem.getSuccessors – checks position of walls to determine successor list

CornersProblem.getStartState – returns the start state of the corners state space as opposed to the full Pacman state space

CornersProblem.isGoalState – determines whether a given state is a valid goal state of the corners state space

CornersProblem.getSuccessors – returns a list of triples of states, actions and costs for the current state in the statespace. For this implentation the cost is always 1.

PositionSearchProblem.getCostOfActions – uses a cost function to calculate and sum the cost for each action in the list of movement actions

FoodSearchProblem.getStartState – self-explanatory

FoodSearchProblem.isGoalState – returns true if and only if there are no more food dots in the game

FoodSearchProblem.getSuccessors – same as other getSuccessor implementations

FoodSearchProblem.costOfActions – uses distances as cost

GoWestAgent.getAction – returns the west direction if possible, else-wise returns stop.

ApproximateSearchAgent.registerInitialState – self-explanatory

ApproximateSearchAgent.getAction – same as others

Descriptions of other procedures

manhattanHeuristic – returns the minimum distance between two points as traveled by moving up or down

euclideanHeuristic – returns the distance between two points using a straight line

cornersHeuristic – returns a lower bound on the shortest path from the state to a problem goal