CISC 440 - Artificial Intelligence & Robotics
Spring 2020

Assignment 06: Due Tuesday, 4/21 by 11:59pm

For this assignment, you will continue "StudentWorld" by using "choco-solver" to implement a planner that identifies a series of moves to reach a target cell. Begin by forking and cloning the assignment as described in the "Assignment Setup and Submission" document in Canvas. You will need to finish the implementation of the StudentWorldConstraintPlanner class.

StudentWorld

StudentWorld PEAS task environment description:

Performance measure
- Escaping through the exit door in the fewest number of moves without being consumed by a student.

Environment
- 5x5 grid of cells with 1 player starting in the upper left cell (0,0), 5 randomly selected cells containing students, and 1 randomly selected door. Neither a student nor the door can be placed in (0,0), (0,1), or (1,0). The student and the door cannot be placed in the same cell.

Actuators
- The player can move up, right, down, or left.

Sensors
- All cells adjacent to a student will smell. All cells adjacent to the door will glow. The size of the grid is known. The CSPPlayer knows which cells have been visited.

StudentWorldConstraintPlanner (studentworld)

An instance of this class will be created by the agent when it has already identified the next cell to visit and needs to know how to get there. This class will use two lists of IntVars to represent the sequence of moves and cells required to reach the target cell from the current cell.

moves : List<IntVar>

This instance variable stores IntVar representations of the moves needed to reach the cell indicated by the instance variables **targetRow** and **targetCol**. All IntVars in this list should store a value in the range [0,3], where 0 represents up, 1 represents right, 2 represents down, and 3 represents left. You will need to access and/or update this list in the createVariables and createConstraints methods (or in methods called by those methods).

playerCells : List<IntVar>

This instance variable stores IntVar representations of the cell that contains the player at each time step on the path from the current cell to the target cell. All IntVars in this list should store a value in the range [0,24], where cells in the top row have values 0-4, cells in the second row have values 5-9, ..., and cells in the bottom row have values 20-24. If n moves are needed to reach the cell indicated by the instance variables **targetRow** and **targetCol**, then index 0 in

playerCells should store an IntVar representation of the player's current cell and index n-1 in **playerCells** should store an IntVar representation of the target cell. You will need to access and/or update this list in the createVariables and createConstraints methods (or in methods called by those methods).

createVariables(numMoves : int) : void

This method should populate the **playerCells** and **moves** instance variables with new IntVar variables. It should only create enough variables for the specified number of moves and required number of cells to indicate player locations from start to finish.

createConstraints(numMoves : int) : void

This method should create constraints involving IntVar variables in the **playerCells** and **moves** instance variables. My solution calls multiple functions to create different categories of constraints. It should only create constraints for the specified number of moves.

Some of these constraints will relate something at time *t* to time *t*-1. For example, if the player is in (2,4) and was previously in (2,3), then the player must have moved right to get to (2,4).

This method should also create constraints involving the first and last IntVars in **playerCells**, since it knows the player's current cell and it knows the target cell.

PROVIDED: getCellNum(row : int, col : int) : int

This provided helper method takes the row and column number of a cell and returns its corresponding IntVar value (0-24).

PROVIDED: getMoveActions() : List<Action>

This provided helper method is called by getShortestPath() to convert the list of move representations in the **moves** instance variable to a list of Actions.

PROVIDED: getShortestPath() : List<Action>

This provided method is called by the agent to get the list of moves to reach the target cell. It repeatedly uses the constraint solver to see if a path from the current cell to the goal cell can be found with n moves, then n+1 moves, then n+2 moves, until it finds a path, which it returns.

Odds and Ends

- Get the player's current cell by calling the getCell() method on the **player** instance variable.
- My solution uses model.and() and model.or() to create compound conditions within model.ifThen() statements.

Your StudentWorldConstraintPlanner.java file will be used along with the other provided files when grading. Your grade be determined largely by the number of tests passed in StudentWorldConstraintSolverTest.java. While these tests are a good indicator of correctness, they do not guarantee a fully correct implementation. In other words, passing all tests may not translate to a 100% on the assignment. Submit your assignment by pushing your code to your gitlab repository fork by the deadline.