AI Course Project

23.04.2022

—

CS-415-AI-C

Sir Muhammad Shakeel
Abdulbasit Janjua
201980014

Overview

I am assigned with the task to **code the project using Python programming language** to solve the CSP. I have used the simpleai library as discussed and used in the course labs.

I have attached the Python (.py) file in the submitted zip folder.

Specifications:

Code	Explanation
from **simpleai.search** import CspProblem, backtrack, min_conflicts, MOST_CONSTRAINED_VARIABLE, HIGHEST_DEGREE_VARIABLE, LEAST_CONSTRAINING_VALUE	In this piece of code I just imported the simpleai module of AI having CSP Problem solver, LCV solvers and MCV solvers etc.
gaming_domains = ['SC','GTA','COD','MK','TO','RL','DS']	In this piece of code I make the **python list** having gaming domains.
def **cons_kiran**(variables, values): return values[0] == 'SC' or values[0] == 'GTA' or values[0] == 'COD'	It is the constraint method for variable **kiran.** And the constraint is kiran must supervise games which are **SC, GTA, COD.** It returns the value of the constraint which is **equal** to SC or GTA or COD.
def **cons_naila**(variables, values): return values[0] != 'RL'	It is the constraint method for variable **naila.** And the constraint is naila cannot supervise a game which is **RL.** It returns the value of the constraint which is **not equal** to RL.

def **cons_faisal**(variables, values):
  if values[0] == 'MK' or values[0] == 'TO' or values[0] == 'RL':
  return False
  else:
  return True
	It is the constraint method for variable **faisal.** And the constraint is faisal cannot supervise games which are **MK, TO, RL.**
	It uses the if else condition in the if part the value is equal to MK, TO and RL which returns false and the else part returns true having any other domain.

def **cons_daud**(variables, values): return values[0] != 'TO'	It is the constraint method for variable **daud.** And the constraint is daud cannot supervise a game which is **TO.** It returns the value of the constraint which is **not equal** TO.
def **diffrent_game**(variables,values): return values[0]!=values[1]	It is the constraint method for variables having **different games.** It returns the not equal comparison of value(0) and value(1).
def **choosed_games**(variables,values): return gaming_domains.index(values[0])<gaming_domains.index(values[1])	It is the constraint method for games which are chosen by the parent must not chosen by child nodes (person depends on others) shown in the list**.**
lcv_variables = ('Marium', 'Daud', 'Faisal', 'Jameela' , 'Kiran' , 'Baber' , 'Naila')	In this piece of code I make the **python tuple** having variable names.

domains = {
  'Baber': ['RL'],
  'Daud': ['SC' , 'GTA', 'COD' , 'MK' , 'TO' , 'RL' , 'DS'],
  'Faisal': ['SC' , 'GTA', 'COD' , 'MK' , 'TO' , 'RL' , 'DS'],
  'Jameela': ['SC' , 'GTA', 'COD' , 'MK' , 'TO' , 'RL' , 'DS'],
  'Kiran': ['SC' , 'GTA', 'COD' , 'MK' , 'TO' , 'RL' , 'DS'],
  'Marium': ['COD' , 'TO', 'DS'],
  'Naila': ['SC' , 'GTA', 'COD' , 'MK' , 'TO' , 'RL' , 'DS']
  }
	In this piece of code I make the **python sets** having domains.

constraints = [(('Kiran'), cons_kiran), (('Naila'), cons_naila), (('Faisal'), cons_faisal), (('Daud'), cons_daud), (('Naila','Marium'),choosed_games), (('Kiran','Daud'),choosed_games), (('Naila','Jameela'),choosed_games), (('Daud','Faisal'),choosed_games), (('Baber','Naila'),diffrent_game), (('Daud','Jameela'),diffrent_game)]	In this piece of code I made the **python list** for variables and applied the constraint methods.
unary_Constraints = [(('Kiran'), cons_kiran), (('Faisal'), cons_faisal), (('Daud'), cons_daud), (('Naila'), cons_naila),]	In this piece of code I made the **python list** for unary variables and applied the constraint methods.
problem_simpleSolution = CspProblem(variables, domains, constraints) problem_partA = CspProblem(variables, domains, unary_Constraints) problem_partC = CspProblem(lcv_variables, domains, constraints)	In this piece of code I define the **CSP Problems** works on firstly variables secondly domains and then constraint methods.
print('\nSolution:\n:', backtrack(problem_simpleSolution)) print('\nUnary Constraint:', backtrack(problem_partA)) print('\nWhen MRV heuristic is applied then baber will assigned first ') print('\nWhen LCV heuristic with Marium as first variable is applied: ' , backtrack(problem_partC, value_heuristic=LEAST_CONSTRAINING_VALUE))	In this piece of code I am just printing the solution
.....End of Code and Explanation.....	