AI Lab Record

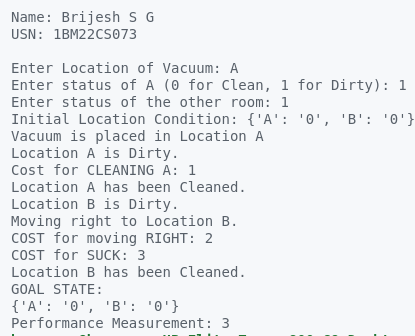
Date: Oct 1, 2024

# **Program Title**: Implement a Vacuum Cleaner Agent

Algorithm - snapshot from observation

Code:

| def vacuum\_world():  goal\_state = {'A': '0', 'B': '0'}  cost = 0  location\_input = input("Enter Location of Vacuum: ")  status\_input = input("Enter status of " + location\_input + " (0 for Clean, 1 for Dirty): ")  status\_input\_complement = input("Enter status of the other room: ")  print("Initial Location Condition: " + str(goal\_state))  if location\_input == 'A':    print("Vacuum is placed in Location A")  if status\_input == '1':  print("Location A is Dirty.")    goal\_state['A'] = '0'  cost += 1  print("Cost for CLEANING A: " + str(cost))  print("Location A has been Cleaned.")  if status\_input\_complement == '1':    print("Location B is Dirty.")  print("Moving right to Location B.")  cost += 1  print("COST for moving RIGHT: " + str(cost))    goal\_state['B'] = '0'  cost += 1  print("COST for SUCK: " + str(cost))  print("Location B has been Cleaned.")  else:  print("No action. Cost: " + str(cost))  print("Location B is already clean.")  else:  print("Location A is already clean.")  if status\_input\_complement == '1':  print("Location B is Dirty.")  print("Moving RIGHT to Location B.")  cost += 1  print("COST for moving RIGHT: " + str(cost))    goal\_state['B'] = '0'  cost += 1  print("COST for SUCK: " + str(cost))  print("Location B has been Cleaned.")  else:  print("No action. Cost: " + str(cost))  print("Location B is already clean.")  else:  print("Vacuum is placed in Location B")    if status\_input == '1':  print("Location B is Dirty.")    goal\_state['B'] = '0'  cost += 1  print("COST for CLEANING B: " + str(cost))  print("Location B has been Cleaned.")  if status\_input\_complement == '1':    print("Location A is Dirty.")  print("Moving LEFT to Location A.")  cost += 1  print("COST for moving LEFT: " + str(cost))    goal\_state['A'] = '0'  cost += 1  print("COST for SUCK: " + str(cost))  print("Location A has been Cleaned.")  else:  print("Location A is already clean.")  else:  print("Location B is already clean.")  if status\_input\_complement == '1':  print("Location A is Dirty.")  print("Moving LEFT to Location A.")  cost += 1  print("COST for moving LEFT: " + str(cost))    goal\_state['A'] = '0'  cost += 1  print("COST for SUCK: " + str(cost))  print("Location A has been Cleaned.")  else:  print("No action. Cost: " + str(cost))  print("Location A is already clean.")    print("GOAL STATE: ")  print(goal\_state)  print("Performance Measurement: " + str(cost))  vacuum\_world() |
| --- |

Snapshot of the output:  


State Space tree-snapshot from observation

Date: Oct 1, 2024

# **Program Title: TicTacToe**

Algorithm - snapshot from observation

Code:

| board={1:' ',2:' ',3:' ',  4:' ',5:' ',6:' ',  7:' ',8:' ',9:' '  }  def printBoard(board):  print(board[1]+'|'+board[2]+'|'+board[3])  print('-+-+-')  print(board[4] + '|' + board[5] + '|' + board[6])  print('-+-+-')  print(board[7] + '|' + board[8] + '|' + board[9])  print('\n')  def spaceFree(pos):  if(board[pos]==' '):  return True  else:  return False  def checkWin():  if(board[1]==board[2] and board[1]==board[3] and board[1]!=' '):  return True  elif(board[4]==board[5] and board[4]==board[6] and board[4]!=' '):  return True  elif(board[7]==board[8] and board[7]==board[9] and board[7]!=' '):  return True  elif (board[1] == board[5] and board[1] == board[9] and board[1] != ' '):  return True  elif (board[3] == board[5] and board[3] == board[7] and board[3] != ' '):  return True  elif (board[1] == board[4] and board[1] == board[7] and board[1] != ' '):  return True  elif (board[2] == board[5] and board[2] == board[8] and board[2] != ' '):  return True  elif (board[3] == board[6] and board[3] == board[9] and board[3] != ' '):  return True  else:  return False  def checkMoveForWin(move):  if (board[1]==board[2] and board[1]==board[3] and board[1] ==move):  return True  elif (board[4]==board[5] and board[4]==board[6] and board[4] ==move):  return True  elif (board[7]==board[8] and board[7]==board[9] and board[7] ==move):  return True  elif (board[1]==board[5] and board[1]==board[9] and board[1] ==move):  return True  elif (board[3]==board[5] and board[3]==board[7] and board[3] ==move):  return True  elif (board[1]==board[4] and board[1]==board[7] and board[1] ==move):  return True  elif (board[2]==board[5] and board[2]==board[8] and board[2] ==move):  return True  elif (board[3]==board[6] and board[3]==board[9] and board[3] ==move):  return True  else:  return False  def checkDraw():  for key in board.keys():  if (board[key]==' '):  return False  return True  def insertLetter(letter, position):  if (spaceFree(position)):  board[position] = letter  printBoard(board)  if (checkDraw()):  print('Draw!')  return  elif (checkWin()):  if (letter == 'X'):  print('Bot wins!')  else:  print('You win!')  return  else:  print('Position taken, please pick a different position.')  position = int(input('Enter new position: '))  insertLetter(letter, position)  return  player = 'O'  bot ='X'  def playerMove():  position=int(input('Enter position for O:'))  insertLetter(player, position)  return  def compMove():  bestScore=-1000  bestMove=0  for key in board.keys():  if (board[key]==' '):  board[key]=bot  score = minimax(board, False)  board[key] = ' '  if (score > bestScore):  bestScore = score  bestMove = key  insertLetter(bot, bestMove)  return  def minimax(board, isMaximizing):  if (checkMoveForWin(bot)):  return 1  elif (checkMoveForWin(player)):  return -1  elif (checkDraw()):  return 0  if isMaximizing:  bestScore = -1000  for key in board.keys():  if board[key] == ' ':  board[key] = bot  score = minimax(board, False)  board[key] = ' '  if (score > bestScore):  bestScore = score  return bestScore  else:  bestScore = 1000  for key in board.keys():  if board[key] == ' ':  board[key] = player  score = minimax(board, True)  board[key] = ' '  if (score < bestScore):  bestScore = score  return bestScore  while not checkWin():  compMove()  playerMove() |
| --- |

Snapshot of the output

State Space tree-snapshot from observation

Date: Oct 1, 2024

Program Title:

Algorithm - snapshot from observation

Code

Snapshot of the output

State Space tree-snapshot from observation