**BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE, PILANI**

Batch No. :

**DEPARTMENT OF COMPUTER SCIENCE AND INFORMATION SYSTEMS**

**Artificial Intelligence (BITS F444/ CS F407)**

**I Semester 2017-18**

**Programming Assignment-2**

**Coding Details**

**(October 3, 2017)**

*Instruction: Type the details precisely and neatly*

1. ID 2015A1PS0524P

Name ANUP BHUTADA

1. Mention the names of Submitted files :
   1. <align3.py>
   2. <align3\_alphabeta.py>
   3. <anup\_driver.py>
   4. <2015A1PS0524P.docx>
2. Total number of submitted files: **4**
3. Name of the folder : **anup\_2015A1PS0524P\_assign2**
4. Have you checked that all the files you are submitting have your name in the top?(yes/no) - **YES**
5. Have you checked that all the files you are submitting are in the folder as specified in 4 (and no subfolder exists)?(yes/no) - **YES**
6. Problem formulation
   1. State representation:

The state has been represented in the form of a class that has the following attributes:

redpos: Stores a list of positions on the 4x4 grid where the user has made his/her moves.

bluepos: Stores a list of positions on the 4x4 grid where the machine has made its moves.

Dimen: Stores a tuple representing the size of the environment (rows, columns).

goal: Stored a boolean value indicating whether the state is a goal state.

* 1. Pseudo code of your successor function

def nextStates(self, Player):

**max\_in\_arr** = generate an array storing the number of tiles occupied in each column

for **ind** in **max\_ind\_arr**:

**x** = column index

**y** = **ind** + 1

next\_pos = using max\_ind\_arr create a list of tuples (**x**, **y**) that indicate the possible next moves

next\_states = []

for **x** in next\_pos:

new\_state = inherit all the attributes of the parent node and add a new position **x** to it.

next\_states.append(new\_state)

return next\_states

* 1. Terminal states generation process:

The game keeps growing until a terminal state is reached. The terminal state is generated when entire grid has been filled with tokens of either players, or when either one of them have won the game by aligning 3 tokens of their respective colours.

* 1. Data structure to store terminal states:

The structure of the node is the same as that of a state described in 7-a with the ‘goal’ attribute updated to ‘True’.

* 1. Method to access terminal states and corresponding utility values

The MINMAX function returns a tuple that has the following information stored at its indices

0: utility value of the most preferred move

1: list containing all the nodes leading from the current state up to the most preferred terminal state.

So the terminal state can be accessed by calling the second element and the utility value can be accessed by calling the first element in the returned tuple.

1. Minimax Technique details
   1. Node structure:

The structure of the node is the same as that of a state described in 7-a.

* 1. Method to ensure the correctness of terminal test (describe in maximum 4 lines):

If a state is a goal state (having 3 tokens of same colour placed adjacently), the ‘goal’ attribute is updated to ‘True’

* 1. Total number of nodes generated to play one game: **7312303**
  2. Write the statistics here as asked

R1 = **7312903** R2 = **452 bits** R3 = **16**

R4 = **1364.75 seconds** R5 = **0.00535**

* 1. Code status (implemented fully/ partially/ not done): **Implemented fully**

1. Alpha Beta technique details:
   1. Explain the logic used for pruning (in maximum four lines):

Suppose that a max node is being explored. This node will have four children min nodes atmost. Each of these min nodes is recursively explored and its utility value is compared with alpha. If the utility value is greater than alpha, the value of alpha is updated. Simultaneously, the utility values of the children of the min nodes are also compared to alpha, and if any of the child nodes has a value lesser than alpha, no further child nodes of that min node are explored.

This same principle can be extended for exploring a min node and updating beta values.

* 1. Total number of nodes generated to play one game: **3175939**
  2. Write the statistics here as asked

R6 = **3175939** R7 = **0.565** R8 = **760.27**

1. Code status (implemented fully/ partially/ not done): **implemented fully**

1. Comparative analysis

Fill in the following information based of 10 independent games

|  |  |  |
| --- | --- | --- |
|  | Minimax Algorithm | Alpha Beta Pruning |
| Average number of nodes created | **7312903** | **3175939** |
| Average time taken | **1364.75 seconds** | **760.27 seconds** |
| Number of times machine wins (player M) | **10** | **10** |

1. GUI details
   1. Created the GUI (yes/ No): **YES**
   2. Have created it according to the specifications?(yes/No): **YES**
   3. Which module of Python is used for creating graphics? - **TURTLE**
   4. Is this under the standard Python library or not? – **YES,** IT IS UNDER THE STANDARD PYTHON LIBRARY
   5. If not, why?
2. Graphics details:
   1. Is turtle graphics working fine for displaying the board and coins? - **YES**
   2. How have you calibrated the board and accepted human input to play the game?

The human input is taken through the terminal that is running the program. A prompt is made to appear in the terminal asking the user to enter the index of the column in which the user intends to make his/her next move. The input given by the user is processed and the GUI is updated accordingly.

* 1. How are you showing the base line?

The baseline in the GUI is indicated by the topmost row of the 4x4 grid.

* 1. How are you showing the move of the machine?

The moves played by the machine are indicated by blue coloured circular tokens placed on the corresponding position where the machine has played on the 4x4 grid.

* 1. How are you showing the move of the human player?

The moves played by the machine are indicated by red coloured circular tokens placed on the corresponding position where the user has played on the 4x4 grid.

1. Compilation Details:
   1. Code Compiles (Yes/ No):\_\_**YES**\_\_\_\_\_\_\_\_\_\_\_\_
   2. Mention the .py files that do not compile:\_\_\_\_\_**NONE**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_
   3. Any specific function that does not compile:\_\_\_\_**NONE**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_
   4. Ensured the compatibility of your code with the specified Python version(yes/no)\_\_\_**YES**\_\_\_\_\_\_\_\_\_
   5. Instructions for compilation of your files mentioning the multi file compilation process used by you (We may use the replica of these for compiling your files while evaluating your code) –

Both the files for minmax and alphabeta algorithm are capable of running independently. The ‘option.py’ file combines the functionality of both the files and lets the user call any one of these algorithms by entering the corresponding option.

1. Driver Details: Does it take care of the options specified earlier(yes/no):\_\_\_\_**YES**\_\_\_\_\_\_\_
2. Execution status (describe in maximum 2 lines):

The program is being executed as expected with proper results and outputs.

1. Declaration: I, \_**ANUP BHUTADA**\_\_ (name) declare that I have put my genuine efforts in creating the python code for the given programming assignment and have submitted only the code developed by me. I have not copied any piece of code from any source. If the code is found plagiarized in any form or degree, I understand that a disciplinary action as per the institute rules will be taken against me and I will accept the penalty as decided by the department of Computer Science and Information Systems, BITS, Pilani.

ID\_\_**2015A1PS0524P**\_\_\_\_\_\_\_\_\_\_\_\_ Name:\_\_\_\_ **ANUP BHUTADA** \_\_\_\_\_\_\_

Date: **October 3, 2017**

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Should not exceed three pages