BALASUBRAMANIYAN Sankari

HEXAPAWN IMPLEMENTATION

METHOD 1

NAÏVE VERSION

1). A main class Hexapawn, Configuration class are created.

2). Inside Configuration class two parameterised constructors are created to create an empty hexapawn board (2D array of type String) and to created a flattened String which holds the configuration of the board.

3). Initially when the main function, the input (size of the board and its initial configuration is taken from the user).

4). A function addLine() is created to convert the string input (configuration) to characters and add them to the board.

5). Two main function Successor() and naïve_version() is created to find the successive movements and to calculate its pertaining value.

6). Successor()

- For the black movement, first the left diagonal is checked for a white pawn: If it exists, the white pawn is removed and black is placed at its position and this particular movement is added to the resultant

- Then, the right diagonal is checked (considering the corner constraints), if there exist a white pawn, that pawn is removed and black is placed at its position and that particular configuration is added to the resultant.

- The possibility of black pawn's forward movement is checked and if its possible, the movement is added to the resultant configuration.

- Similarly during the white pawns turn, the motion is checked exactly like it was done for black pawn.

- So hence the successor function returns three possible black pawn move and three possible white pawn move for all the pawns present in the board.

7). Naïve_function()

- First checks for the presence of white pawn in black line – white won and returns 0

- Checks for the presence of black pawn in white line – If yes, black won and returns 0

- An Arraylist Successor is created to call the successor function and calculate the present configuration's successive motion.

- If the length of the Arraylist is zero, there is no successive motion and hence zero is returned.

- Now every pawns motion in the successors Arraylist, its own successive movements with other black and white is calculated by recursively calling naïve_function again and the value is stored.

- Two variables MaxPosi and MaxNega is maintained to calculate the value of each successive motions. MaxPosi stores the maximum Positive value and MaxNega stores the maximum Negative value

- In the end the value of the configuration is returned.

METHOD 2

DYNAMIC PROGRAMMING

HashMap (called memo) is used to store the configuration and its value, so we don't have to calculate the values of the configurations that already exist. To access a value of the configuration or to insert a value of the configuration in the HashMAp, it only takes O(1) time complexity.

- **A function equal() is defined to create equal contract and compare the private attributes of the class.**
- **Hashcode is implemented with prime value as 31 to avoid or minimise Hashcollision.**
- **Two main function in this method are the Successor() and the Dynamic_version().**
- **Successor is implemented as it was stated before for naïve version**
- **In Dynamic_version(), we check if the current successor value is already present as a key in the HashMap. If it does, then the value of the configuration is returned and if not the value is calculated by recursively calling the dynamic_version function again.**
- **In the end the value of the configuration is returned.**