EXTREME TIC-TAC-TOE BOT

Key features of the code:

- Used simple alpha-beta pruning with iterative deepening
- Used zobrist hash to speed up computation
- Implementation of Bonus move
- Used separate update function for the board

Heuristic :

The board is composed of 16 blocks. There are two parts of computation

1. Computing heuristic for the small blocks
2. Computing for the entire board

<u>Small Blocks:</u>

There are 3 entities to take care of : Row, Column and Diamond.
We assign value to a particular block state on the basis of number of X's and O's in each
Of the above 3 entities . In our case , for a single X in a row award 1, for 2 X's award 100 and so on . (Considering X is the maximizing player) and return negative if it is minimizing player's symbol .
For an entity with at least one O and at least one X ignore as that entity cannot make you win the block.
Final Value : Sum over the values of all (4*3 = 12)entities

<u>Big Board:</u>

Similar approach as for smaller. Here

- Ignore if both an X and an O in an entity or a Draw
- If the small block has an X or an O assign a specific value
- otherwise(it is empty) take the sum of values of all the block values of the empty blocks and the specific value for the X's and O's

Bonus Move :

If you win any small-block in game then you get a single bonus move(i.e if you win a small block again in bonus move, you won't get a bonus move again).

As in last stages of game most of the small-blocks are just one or two move ahead from ending so, the significance of bonus move increases. We Implemented the bonus move in our code by maintaining a variable **last_blk_won like this :-**

```
#For Checking that the selected move wins a small block or not
status, blk_won = self.update(board, old_move, prevans,
self.map_symbol[player])

# iF last_blk_won ==  1 implies u already in bonus move state so, you
won't get it again
if blk_won == True :
    self.last_blk_won ^= 1
else:
    self.last_blk_won = 0
```

If your code looks for bonus move even if it is not in that state it may end up selecting a move that can cost your game.
Another important part in the implementation is to check it at the very starting itself because it maybe possible that when the move function is called the bot has already played a move before.

Update function:

As the update function in simulator was checking that the move is valid or not and in our code as we are only calling the update function on valid moves. so, we don't need to check its validity again because of that we made our own update function.

This saves computations in our program and hence our code runs much faster which will eventually helps the minimax function to go more deeper in the tree for finding best move.

Zobrist Hashing:

Zobrist hashing is a hash function we used based on the xor operation . We store a board position of a block on the basis of bits.
We have 3 states for a particular cell :-

- X
- O
- Empty

So, state of a cell can be represented by 2 bits and as there are 16 cells in a small block so total no. of bits used for representation are 2*16 = 32 for a small board.
Thus a block can be represented within 32 bits and storing the values corresponding to these in a dictionary speeds up the code . Note in this case since it perfectly records the game state , there is no need for detecting collisions.