TETRIS AI PROJECT

In tetris you have blocks of four (*tetrominoes*) falling from the top of the board. The player moves and rotates the blocks and stacks them up .

when a row is filled entirely with blocks (the row with the red outline below), you get a *clear*; that entire row is removed and the rest of the board is shifted down (often with a series of beeping noises and a small increase to your score) .

If the blocks don't get cleared and they stack to the top of the board, you lose. So ideally you want to fill as many lines as possible and avoid stacking the blocks up.

Objective of the game :-

The levels don't stack up , so that the game window is full . because then the game will be over .

Strategy Parameters / state variables :-

We lose when the blocks get stacked up so high that we can't put in new blocks, right? So it makes sense to avoid piling up large numbers of blocks in high positions in the first place, or to **penalize height .**

So for each position the computer can calculate a *height penalty* — for each block the computer adds a number depending on how high it is. Then when the AI tries to decide where to put the next block, it 'knows' that blocks piled up really high is a bad thing and tries to avoid it .

A row is only considered a clear if the entire row is filled — if there's even a single hole in the row, it doesn't get removed. Not good. So it makes sense to give a negative score to positions with holes in them — to **penalize holes**.

If we define a *blockade* as any block that's directly above a hole, we should **penalize blockades .**

Well, a hole only stops being a hole if there are no more blocks above it, so stacking more blocks above holes would only make it harder to remove the hole.

Hueristic Function Used :-

Breadth First Search

1. Look at the current block and the next block and simulate ALL possible combinations (positions and rotations) of the two blocks.
2. Calculate a score for each of the positions.
3. Move the block to the position with the highest score and repeat.

Hueristic Function Used :-

Score = A * Sum of Heights
+ B * Number of Clears
+ C * Number of Holes
+ D * Number of Blockades

Where as Defined in Strategy , Sum of Heights , Number of clears , Number of holes , Number of blockades are **State variables .**

Where A, B, C, and D are *weights* that we decide — how important is each of the factors. I initially came up with some pretty arbitrary values:

- -0.03 for the height multiplier
- -7.5 per hole
- -3.5 per blockade
- +8.0 per clear
- +3.0 for each edge touching another block
- +2.5 for each edge touching the wall
- +5.0 for each edge touching the floor

Using Genetic Algorithm

A *genetic* algorithm is just a searching heuristic; it derives its ideas from evolution, where nature creates complex and sophisticated organisms by making *random* changes to the DNA.

After running the genetic algorithm for about ten generations, I picked a candidate that was scoring decently:

- -3.71 for the height multiplier
- -4.79 per hole
- +1.4 per blockade
- -1.87 per clear
- +4.8 for each edge touching another block
- +3.22 for each edge touching the wall
- +3.68 for each edge touching the floor

OUTPUT