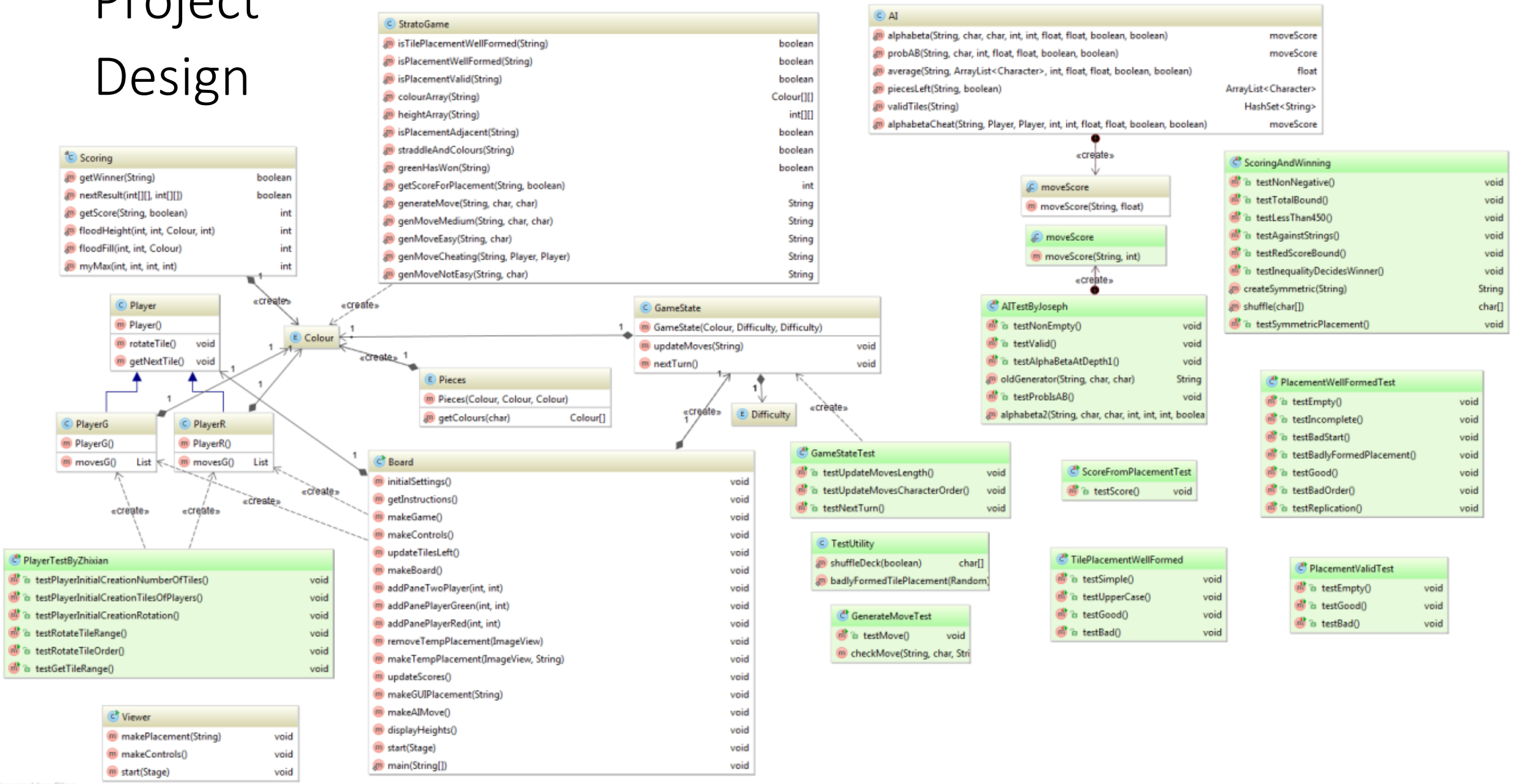
Stratopolis

(Or at least the version by Manal Mohania, Joseph Meltzer, and Zhixian Wu)

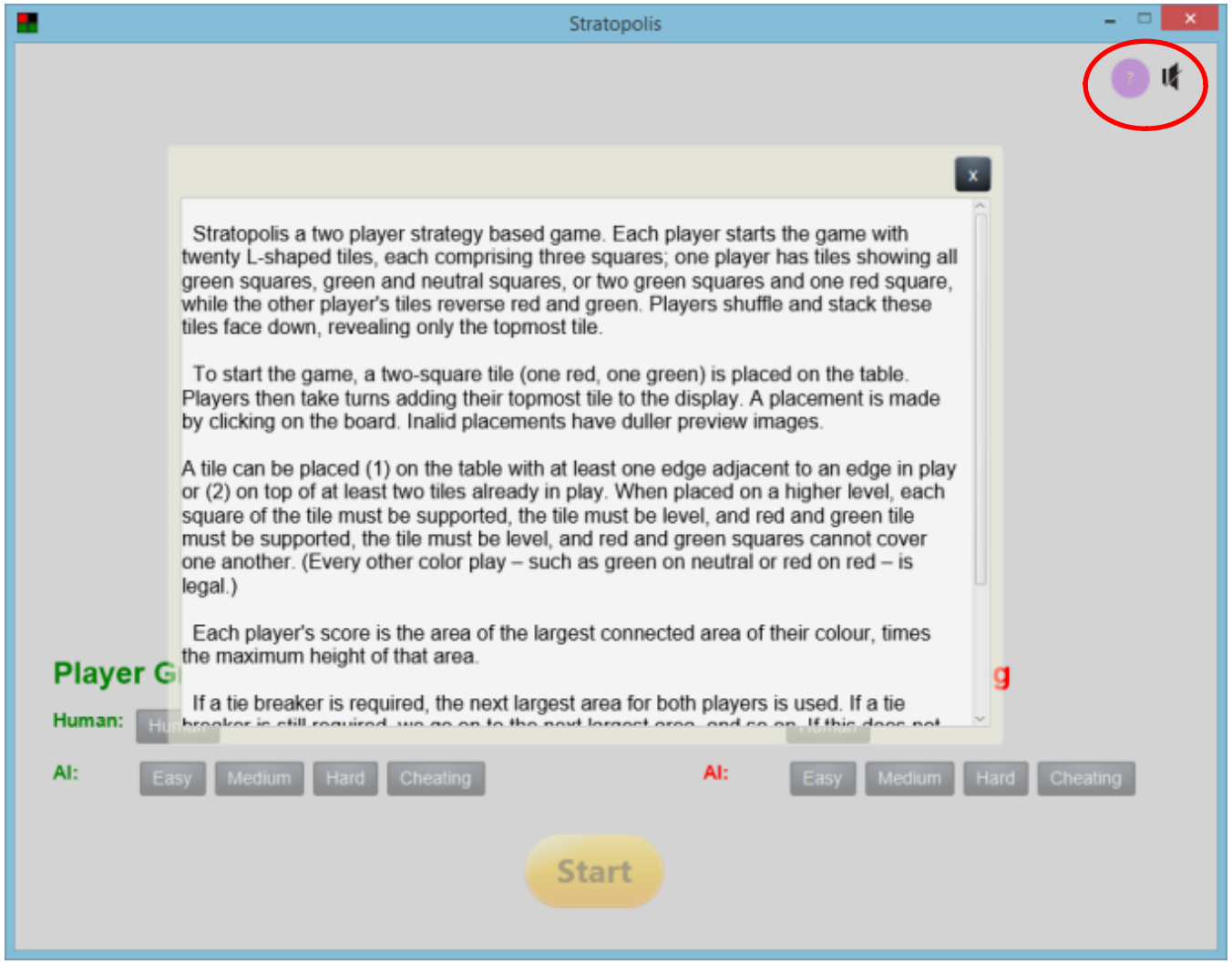
Project Design



Starting Screen

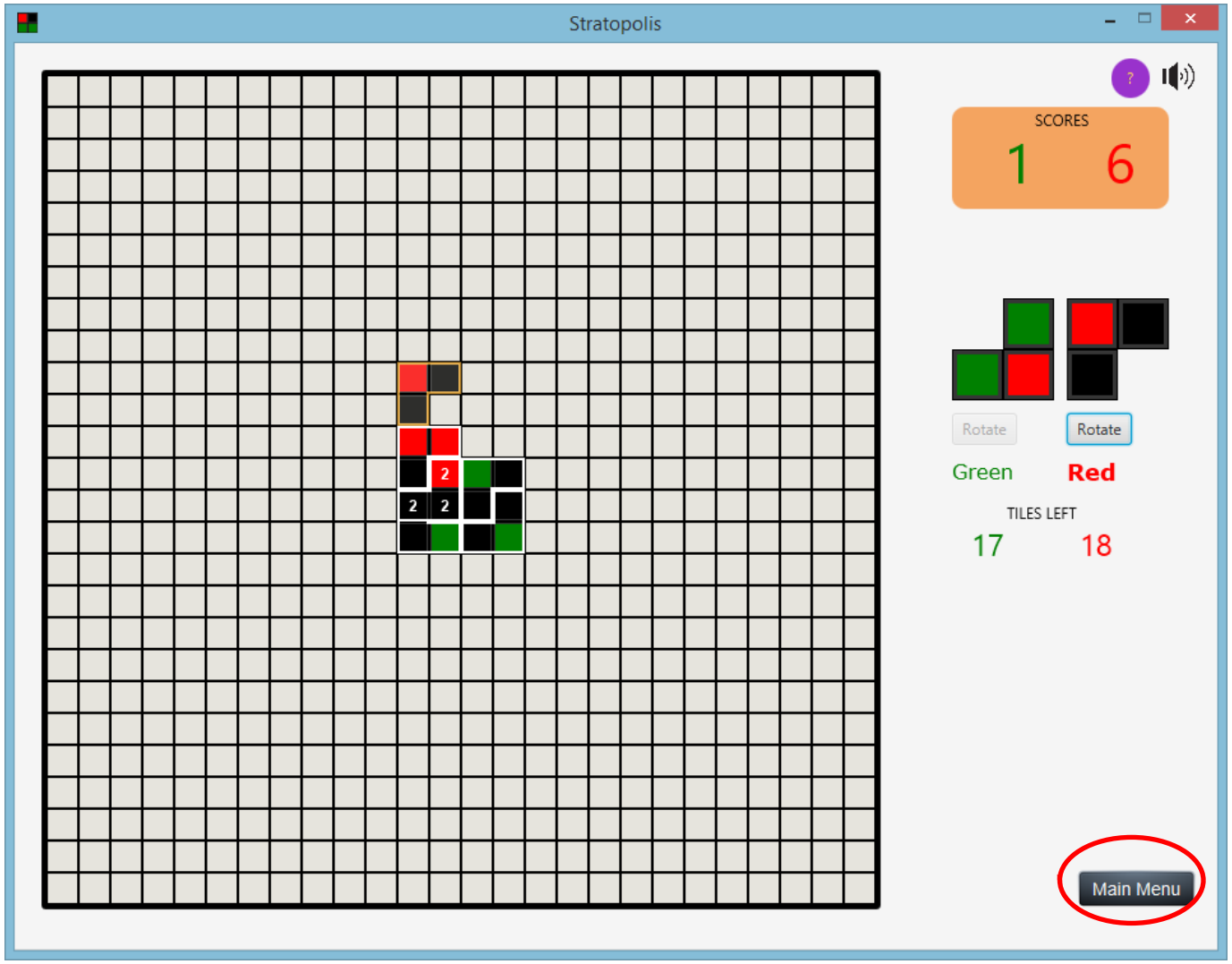
- This is made by calling the function initialSettings(). We put this into a separate function since if you want to restart the game it has to make this screen again.
- You can choose which player is human and which is an AI, the difficulty of the AI, and you can have an AI vs. AI game.

Starting Screen



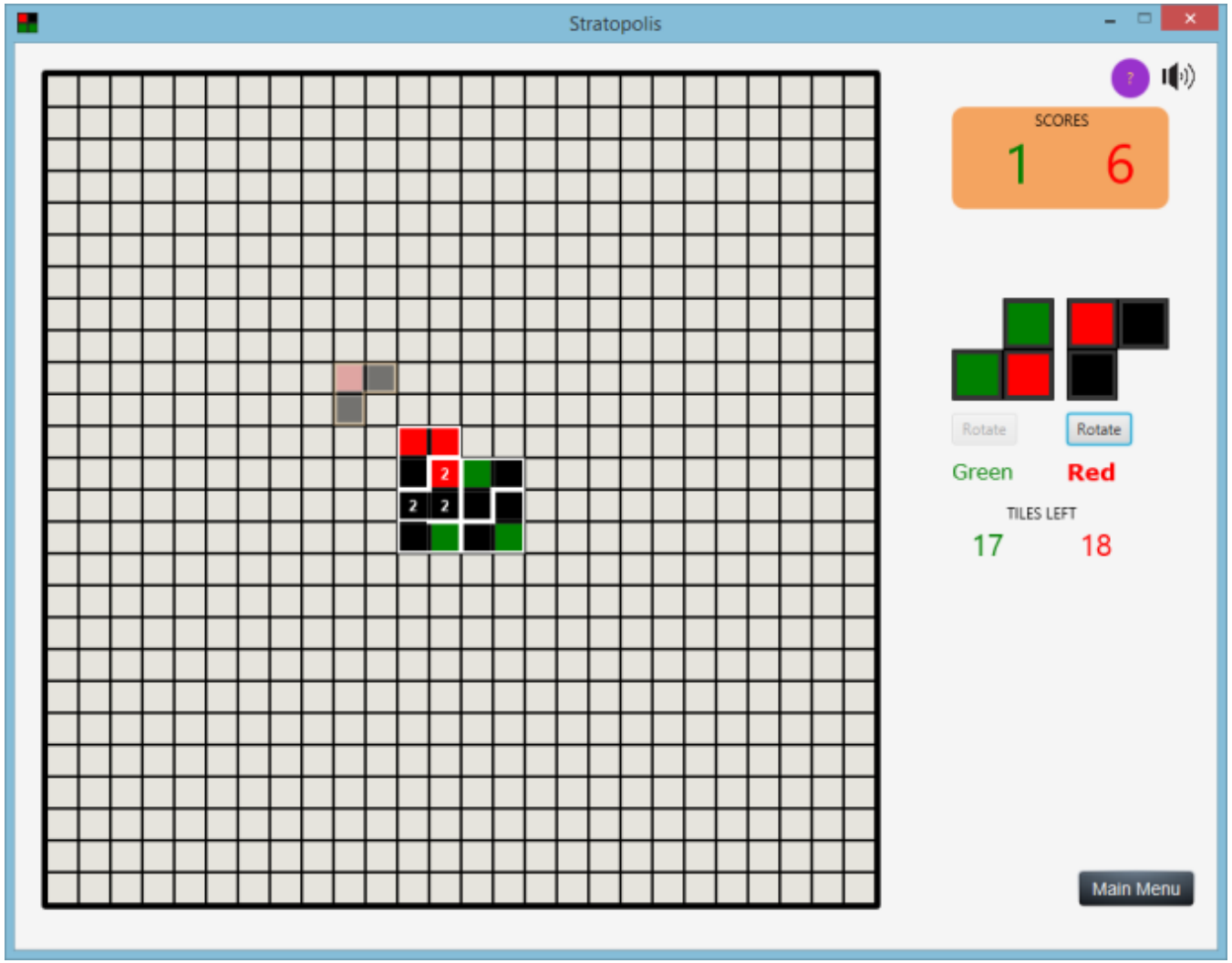
- There's a button for the sound. Next to it is the instructions button, which gives you a pop-up with the instructions, disabling all the buttons when it's up.
- When you click start, the game screen is created a little differently depending on which players are human and which are AI. It also creates both players for each new game.

Human vs. Human



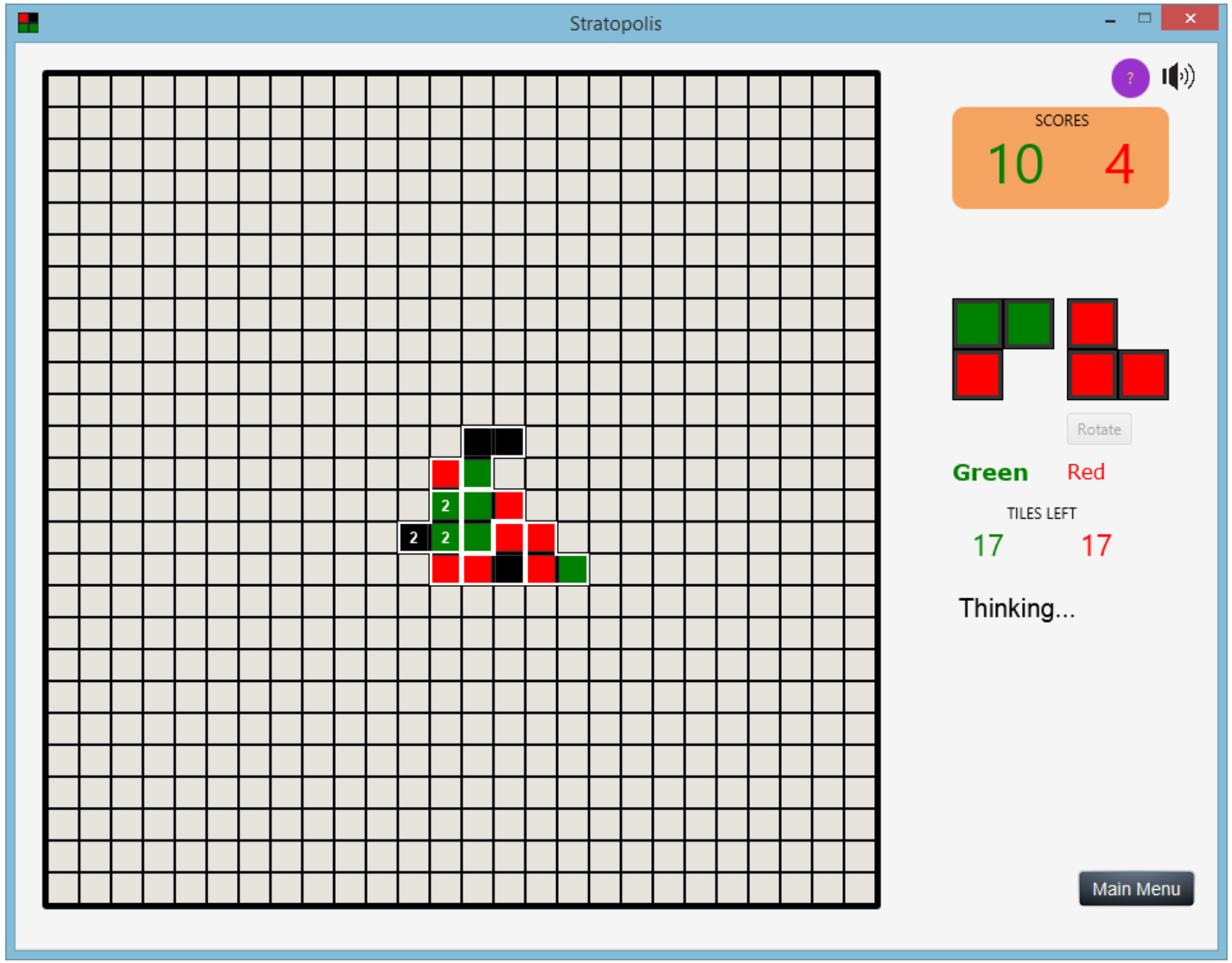
- All the games with at least one human player has 26x26=576 panes that generate previews of the piece to be placed. If the placement is not valid, the preview image is desaturated.
- Only the player whose turn it is has their rotate button working, the other is disabled.
- All games have a ‘Main Menu’ button that takes you back to the start screen.

Human vs. Human



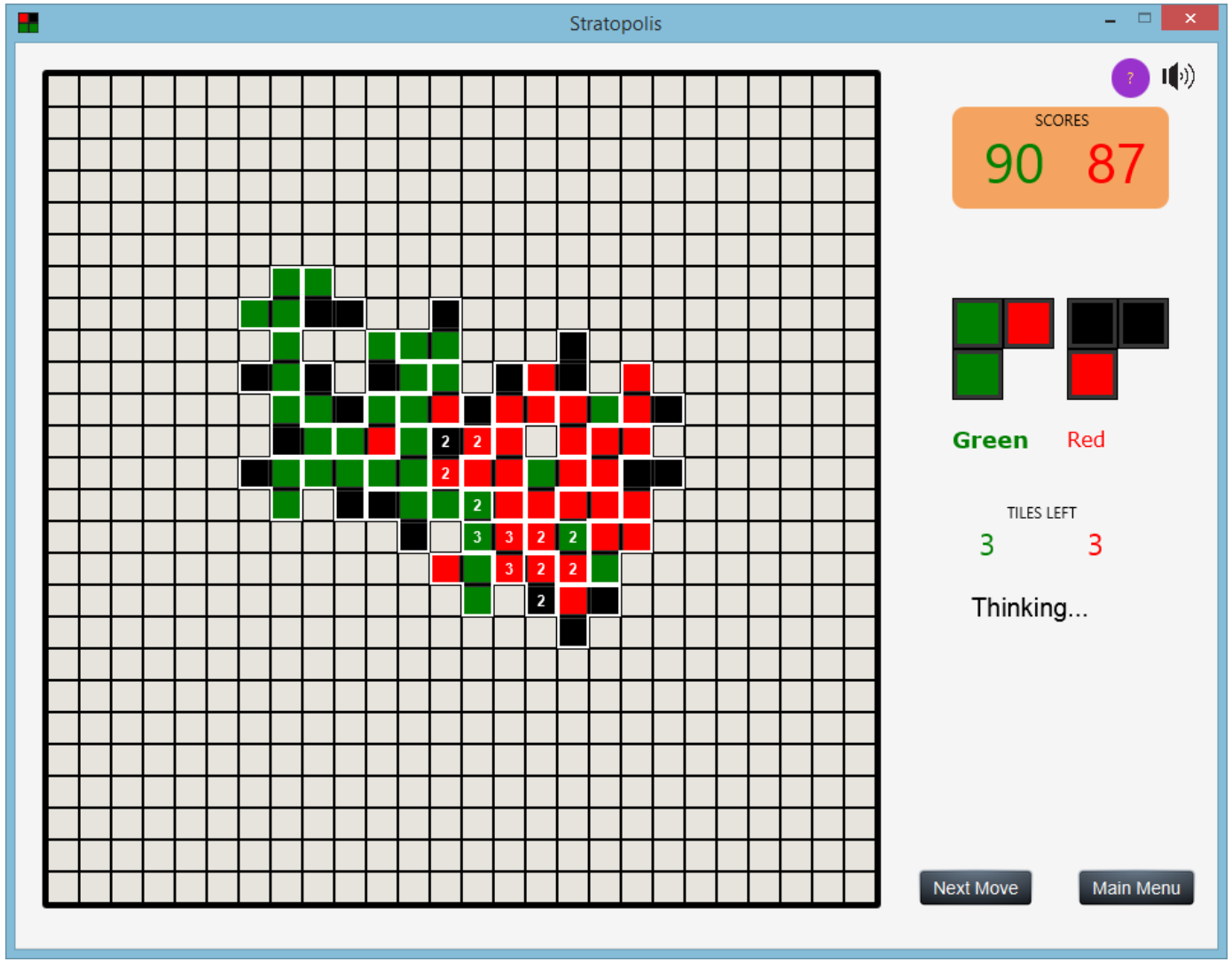
- If you click on the pane, makeGUIPlacement() is called. Every move, be it the initial MMUA, one played by a human, or by an AI, is made using this function
- This function does a lot (in addition to putting an image in the right place), mostly relating to updating various things such as player objects.

Human vs. AI



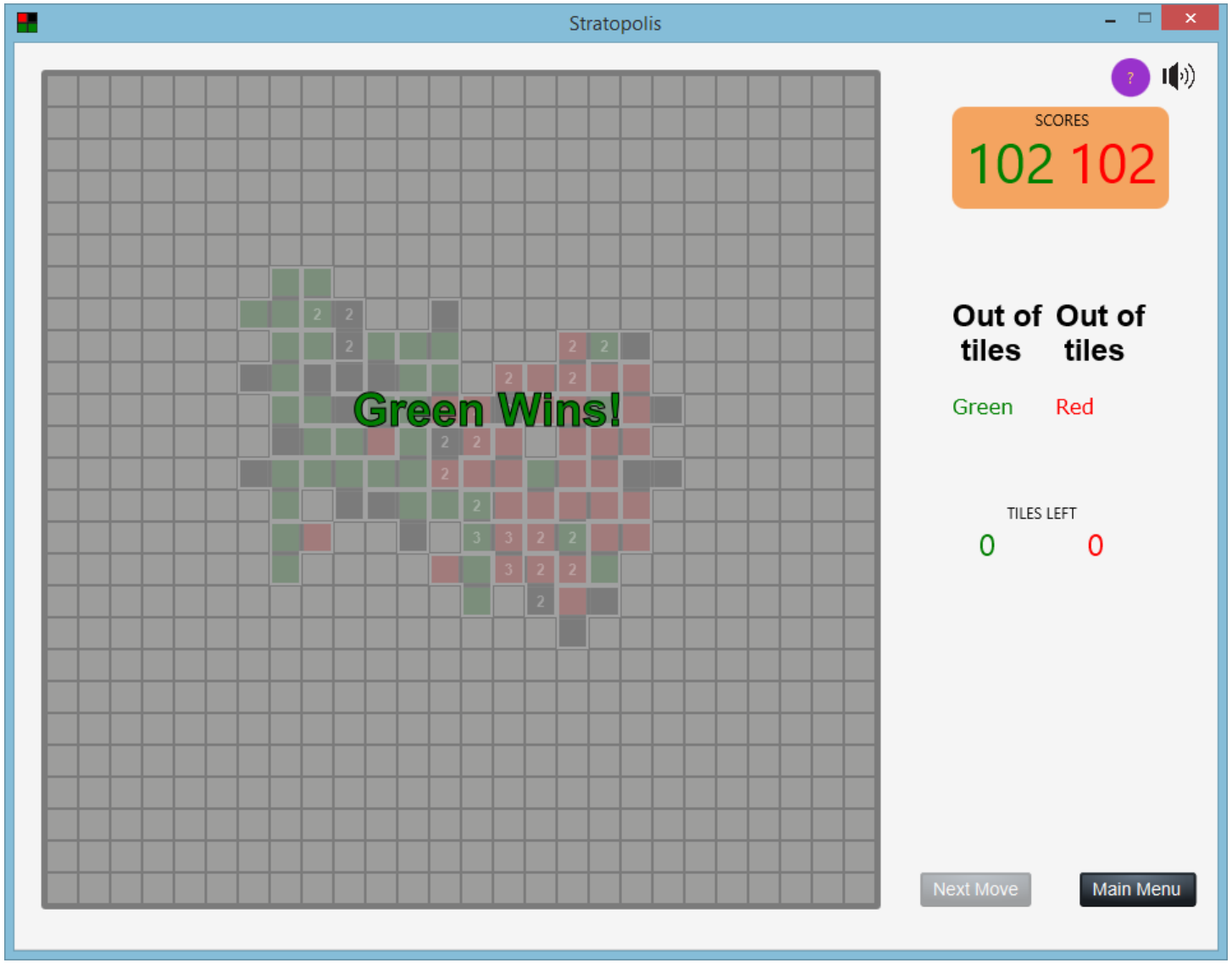
- Here the AI's rotate button is simply omitted.
- And when you click the pane, it makes your move when you press down and displays 'Thinking…' text.
- Then it makes the AI's move when the mouse is released, and removes the 'Thinking…' text when it's done.

AI vs. AI



- Here the clickable panes are not added. There is only a ‘Next Move’ button to advance the game.
- ‘Thinking..’ is still showed when ‘Next Move’ is pressed, and it is removed after the AI makes its move.

Game Over



- Every time makeGUIPlacement() makes its placement, it lastly checks if the game is over.
- If it is over, the winner is calculated and displayed, and the ‘Next Move’ button along with all clickable panes are disabled.

Designing the AI

- At its core, the minimax algorithm
- Methods for refining the search
  - Alpha-Beta pruning
  - Move ordering
  - Search range
- Limitations of Stratopolis: imperfect information
- Probabilistic AI

Designing the AI

- First version, used for ‘Easy’ difficulty
  - Naïve search of depth 1
  - Weak heuristic
  - Perfect information
  - near-Instant search time
- Second version, used for ‘Medium’ difficulty
  - Search depth 2
  - Better heuristic
  - Perfect information
  - Fast search time

Designing the AI

- Third version, used for ‘Hard’ difficulty
  - Search depth 3
  - Altered heuristic
  - Perfect information on first two levels
  - Imperfect information from then on
  - Probabilistic heuristic evaluation
  - Extremely slow search time
    - Third level
    - Probabilistic cases
  - Needed a lot of refining

AI Efficiency

- Alpha-Beta pruning
  - Straightforward to implement
  - Cut-offs could be effective given search width
  - Not enough
- Move ordering
  - Straightforward to implement
  - Could increase number of ‘effective’ cut-offs
  - Eliminate most empty tiles
  - Not as intelligent as it could be
  - Still not quite enough

AI Efficiency

- Limiting the search area
  - Searching only within a ‘bounding box’ of the placed pieces

  - More specific
  - Searching only tiles adjacent to or on a placed piece
  - Noticeable improvement
- Profiling
  - String.charAt
  - isPlacementValid getScore
  - Little self time

The Scoring Algorithm

- The flood fill algorithm was used repetitively to determine the score
  - Begin with the top leftmost square.
    - If it is not of the required colour or a tile does not exist skip it , and move on to the next square
    - If it is of the required colour use the flood fill algorithm to determine the area. Use flood fill to determine the maximum height in the region too.
    - Black was used as the replacement colour to ensure that no square gets counted twice
  - Save the areas and the corresponding heights in a 2D array
  - Identify the maximum area and hence, the score.

Scoring Efficiency

- The only major efficiency concern with the scoring algorithm is that flood fill is used twice over two copies of the 26x26 colour array – once to determine the area and once to determine the corresponding maximum height.

- To improvise, one could probably combine the two different flood fill functions and use a single copy of the 26x26 colour array, and use black as the replacement colour, once both functions have been called.

Winner Determination Algorithm

- Determining the winner is trivial if the final scores of the green and red players are different.
- If they are not, the 2D arrays used to store the areas and the corresponding heights from the scoring section are used.
- The maximum area is identified in each of the two arrays, and if their score is equal, those areas are removed and each of the entries to their right are shifted by one index to the left.
- The function is recursively called with the modified 2D arrays until a winner is identified.
- If at the end, there is no clear winner, a random colour is chosen according to the rules of the game.