AI Programming (IT-3105) Project Module # 4:
Using Minimax with Alpha-Beta Pruning to play 2048

Purpose:

- Learn the basic elements of adversarial search in AI.
- Gain hands-on experience with one of AI's most popular adversarial search methods: Minimax with alpha-beta pruning.

1 Assignment Overview

You will implement Minimax with alpha-beta pruning from scratch in the language of your choice. It is a very simple algorithm that is well-described in most AI textbooks. Consult the course' lecture notes on "Adversarial Search" for additional assistance.

You will apply the algorithm to the (quite popular) 2048 game, which is a sliding tile puzzle that poses a significant challenge to both humans and computers, as opposed to AI Classics such as the (sliding-tile) 8-puzzle, which most intelligences (natural and artificial) can easily conquer.

Your grade will be based, almost entirely, on the level of play that your system achieves.

2 Introduction

Like most good games, 2048 has a small set of simple rules that are easy to understand and remember, but despite this simplicity of description, it presents a complex challenge. Winning is far from easy.

As shown in Figure 1, the board is a 4 x 4 square, with a variable number of tiles, all of which are marked with a power of 2. The player has a choice of 4 moves: left, right, up, and down. Each such move causes EVERY tile on the board to slide as far as possible in the chosen direction.

In addition, when two tiles of equal value collide, they merge into a new tile whose value is the sum of the values of the original tiles: a doubling. For example, when two 4-tiles collide, they produce one 8-tile. The new tile takes the position of the original tile that is farthest along in the direction of the move. So if all tiles slide to the left, and two 4's collide, then the 8-tile replaces the leftmost of the 4-tiles, while the other 4-tile vanishes from the board (see Figure 1).

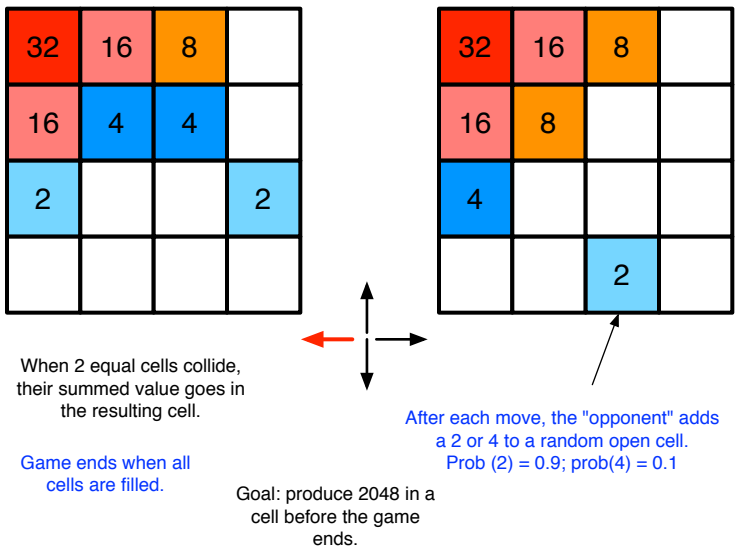

Figure 1: Typical 2048 scenario. (Left) The board just prior to a leftward slide. (Right) The board just after a leftward slide followed by the addition of a new (2) tile near the grid's bottom right corner. Notice that in the 2nd row from the top, the two 4-tiles collide to produce an 8-tile; and in the 3rd row from the top, two 2-tiles collide to produce a 4-tile.

As shown in Figure 2, when two collisions occur in a row or column, then the new tiles both slide as far as possible and become neighbors. However, these new tiles do not immediately collide again; that would require a second slide move. For example, either a left or right slide on the next move (of Figure 2) will cause the two 4's in the 3rd row to collide, while an up or down move would merge the two 4's in column 3 and the two 8's in column 4.

After each slide move (and the ensuing collisions and mergers), a new tile (of value 2 or 4) is added to a randomly-chosen open cell of the board. Rules of the game vary slightly, but for this assignment, the probability distribution for these two choices will be the following:

- With a probability of 0.9, choose a 2.
- With a probability of 0.1, choose a 4.

The choice of the open cell to fill must be **completely random**: a uniform distribution over all the open cells. The placement cannot be biased in any way, as this can easily favor (or block) a win.

Winning in 2048 entails achieving a single tile with the value of – you guessed it — 2048.

The beginning of each game is a board with a single, randomly-chosen, cell housing a 2 (or 4), while all other cells are empty. The choice of 2 or 4 is governed by the same probability distribution as given above: 0.9 for a 2, and 0.1 for a 4. The game ends when the board fills with tiles, so a winning strategy needs to (among many other things), achieve plenty of mergers from the chosen slides.

At first glance, the game of 2048 does not appear adversarial at all: only one person moves tiles. However, the constant interjection of new tiles (termed *spawning* by 2048 aficianados) adds a considerable complexity to decision making, since an unfortunate placement can easily ruin a promising tile arrangement. For example,

in Figure 2, notice that the spawning of a 2-tile occurs in the upper left corner, which was recently vacated by a 32-tile. One key heuristic in 2048 is to keep the highest tile in a corner. This spawned 2-tile blocks any immediate attempts to return 32 to a corner, and, in general, a highly *exposed* high-valued tiles tend to get in the way of other mergers (of lower-valued tiles). Similarly, a small-valued tile, if stuck in the corner behind several large tiles, has little chance of merging with newly-spawned 2s and 4s. Establishing this structure, of more exposure for the smaller tiles and less for the larger tiles, seems to be an important aspect of many winning strategies.

Since 2048 was created as recently as March 2014 (by Italian web developer Gabriele Cirulli), it does not have a long history of play or analysis. However, there is plenty of information about it scattered across the internet, with Youtube videos being some of the best sources of strategies. Many of those discussed on the web can directly translate into heuristic functions for Minimax. You are wise to do a little research. Random play won't get you anywhere near a 2048-tile.

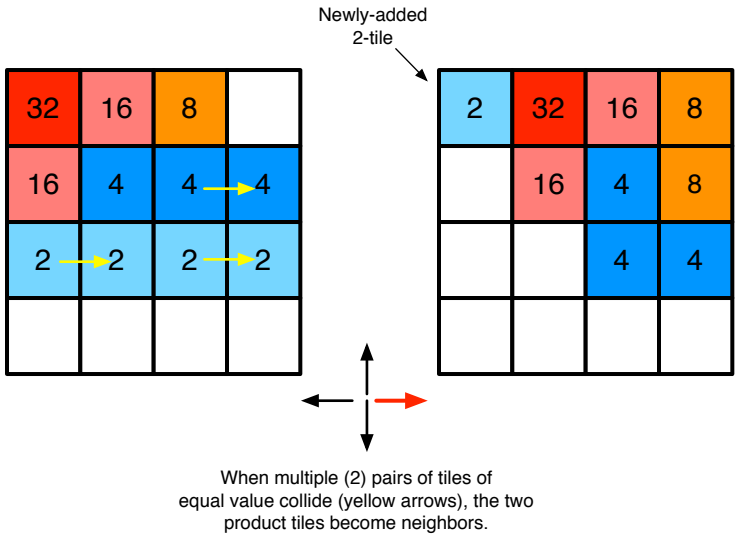

Figure 2: A 2048 scenario in which a rightward slide produces 3 different mergers.

3 Minimax with Alpha-Beta Pruning for 2048

As shown in Figure 3, the application of Minimax to 2048 involves the standard tree with alternating layers of max and min nodes. Each max node considers all (4) move options, while each min node considers all possible placements of the spawned tile (a 2 or 4). The heuristic function is only applied to the bottom layer of the tree, as is standard for Minimax.

There are many alternatives for the heuristic function, and coming up with a good one is probably the biggest challenge of this assignment. Remember that the heuristic function is the main source of *intelligence* in your system. Typical factors to consider include:

1. The location of the (current) largest tile on the board. Is it in a corner?
2. The number of free cells. It's always good to have many, since a nearly-full board could be dangerously close to a premature end-of-game.

As discussed in detailed accounts of Minimax, a large tree of this type is generated **every time** the main player makes a move. The use of alpha-beta pruning helps to reduce the size of this tree, and thus speed decision-making, without reducing the quality of the move choices or the ultimate success of the player. Alpha-beta pruning is a completely context-free algorithm that can be taken right from the textbook and applied directly to most problems, including 2048. See the lecture notes for further details on both standard Minimax and Minimax with alpha-beta pruning.

Although an appropriate search depth will depend, to some degree, on the sophistry of your heuristic function, it should not be necessary to go beyond a depth of 5 or 6. Even a depth of 3 can give reasonable performance; but for most heuristics, you will probably notice an improvement in going from 3 to 4 or 5. The spawning operation produces a high enough branching factor to make the transitions among these depths quite noticeable, in terms of runtime. So don't expect to search at depths beyond 6 or 7, depending upon your computer resources.

To keep the branching factor down, you may elect to use the following shortcut. If there are C open cells, then, for perfect lookahead, your system would need to consider 2C cases: the placement of a 2 or a 4 in each of those cells. The shortcut is to consider only C cases, where the choice of tile for each case is the (biased stochastic) choice of a 2 or 4, with the probability distribution being the standard: {0.9, 0.1 } used during game-time spawning. You may need to experiment with both the C and 2C options, but it is certainly possible to build a winning strategy using the C approach.

3.1 Basic Coding of the Game

The sliding and merging of tiles involves relatively straightforward (though somewhat non-standard) 2-dimensional array operations. You will need to program these **perfectly** in order to play a proper version of the game. One mistake can easily make the game too simple or nearly impossible. Any such bugs will yield a "0" for your demo score, so be 100 % sure that you are simulating the proper game of 2048!!

It is wise to program your simulator and game visualizer (below) – and then use the latter to verify the correctness of the former – before moving on to Minimax and alpha-beta pruning.

3.2 Visualization

For this assignment, you must design (or download) a graphic interface that allows the viewing of **every** state of a game in progress (not just the final state). This interface **must** use color-coding of cells, with some sensible color scheme, such as bright red for higher-valued tiles and light blue for lower-valued tiles (as used in the figures of this document). Color-coding is essential for this assignment, since you will be displaying games run at high speeds – too fast for us to read the numbers. With proper color-coding, we should be able to assess the basic structure of the board and the essence of the board transitions.

If you are coding in a language with traditionally fast run-time performance (such as C++), then your GUI should include a delay option that permits runs at a speed slow enough for us to evaluate progress.

You do **not** need to visualize the Minimax tree in any way, only the moves made as a result of the information in the tree. Remember that Minimax produces a huge tree as the basis for a single move.

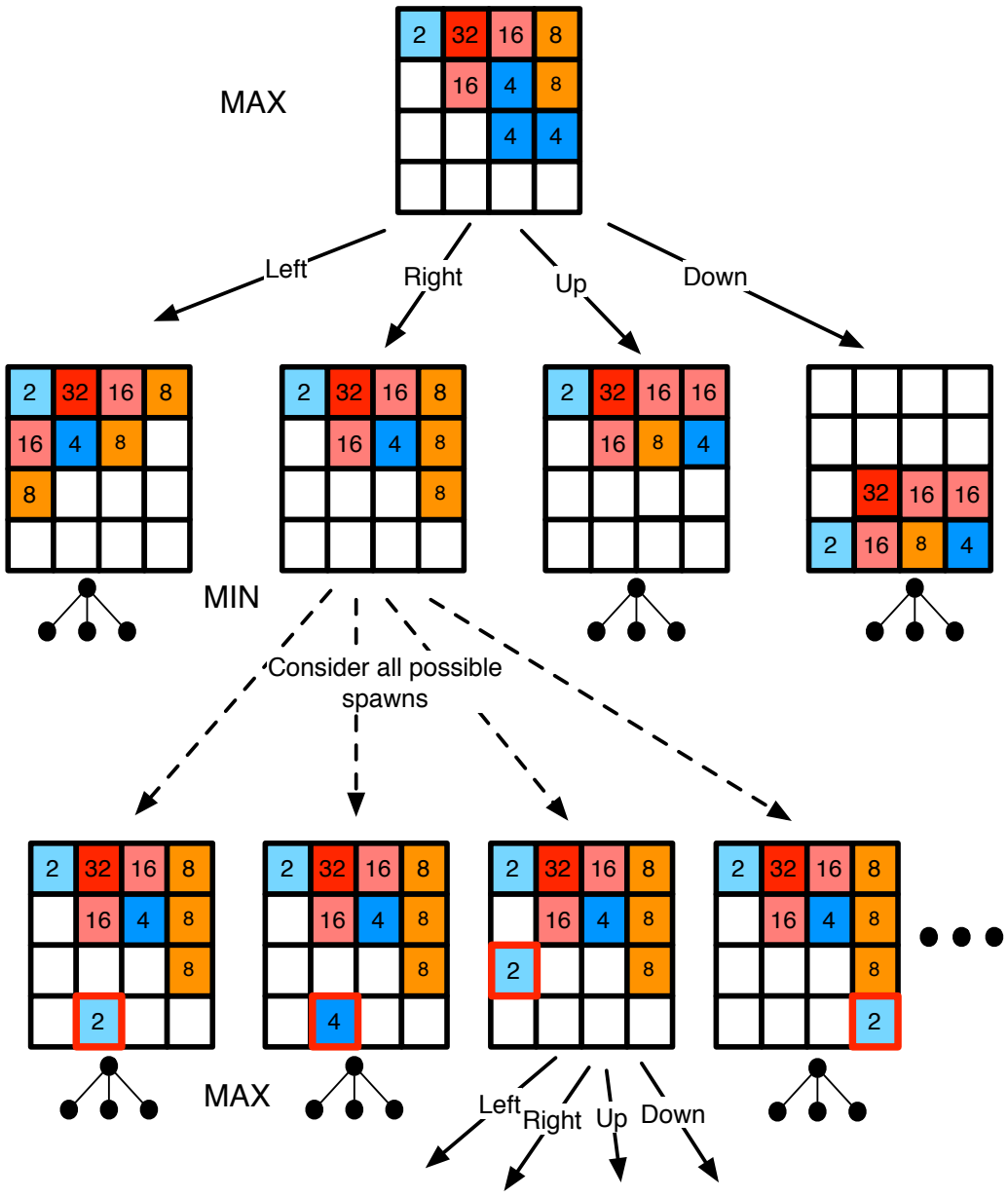

Figure 3: The top of a Minimax search tree for 2048. The main player considers all 4 of its move options, and then, from the resulting states, it must consider all possible spawning events (of a 2 or 4 in any of the open cells) by the opponent. The heuristic function is only applied to nodes at the bottom of the tree (not shown).

4 Deliverables

1. A 3-page report (**4 points**) that:
   - Describes the overall structure and key elements of your Minimax-with-alpha-beta-pruning system.
   - Clearly documents (in full detail) your heuristic function. This must NOT be presented as code, but rather as a mathematical expression where every symbol is clearly defined.
2. The demo of your 2048 player. You will be given a **maximum of 4 runs** of your system during the demo session. The highest tile that you achieve in all of those runs will be the basis of your demo score. If the maximum value is M, then your demo score will be calculated as:

$$3 \times log_2(\frac{M}{16}) \tag{1}$$

   The maximum number of demo points is 21. Producing a tile of 4096 or greater will earn you our eternal admiration, but no extra points.

Important As mentioned earlier, getting the details of 2048 correct is essential for this assignment. That includes the spawning operation, which **must** produce 2's in 90% of the cases, and 4's in the remaining 10 % of cases, and **must** place the spawned piece at a random open location, with equal probability for each of the open cells.

A zip file containing your report along with the commented code must be uploaded to It's Learning prior to the demo session in which this project module is evaluated. You will not get explicit credit for the code, but it is crucial that we have the code online in the event that you decide to register a formal complaint about your grade (for the entire course).

The 25 total points for this module are 25 of the 100 points that are available for the entire semester.

The due date for this module is the 2nd demo session.

References