# AI Programming: Assignment 4

Eirik Vågeskar

October 27, 2015

This report describes a Python implementation of an expectimax system for playing Gabriele Cirulli's 2048 game. It is partially inspired by Lee Yiyuan's *2048 AI – The Intelligent Bot* which can be found at
`https://codemyroad.wordpress.com/2014/05/14/2048-ai-the-intelligent-bot/`

The expectimax system is a subclass of the file which implements the game itself. A brief description of the game implementation will be given as well as a description of the expectimax system.

## 1 Game Implementation

The game is implemented as a class called *PowerBoard*. The name was chosen because the tiles in 2048 are powers of two (and due to Python naming restrictions preventing the use of class names starting with numbers). Its constructor takes the dimensions of the board as a tuple and constructs an empty board with the given dimensions.

The representation of the game board is a one dimensional list. Manipulation of entries in a one-dimensional list is slightly faster compared to manipulating a two-dimensional list. Getting and setting values at certain coordinates is handled through helper functions. These are utilized by the sliding and random-tile methods.

## 2 Expectimax Implementation

The expectimax implementation is found in the class *PowerBoardState*, which represents one state of the board during search. It is a subclass of PowerBoard. It inherits all of its superclass's fields and adds the field *recursion_depth*. This is an integer which represents how deep a recursive expectimax starting from this node can go. When this reaches 0, the recursion stops and a heuristic score is returned.

### 2.1 Decision Method

At the heart of the class is the method *decision*, which is used to make a decision about which way the tiles should be slided, given the current state of the board. When a user (or an agent) wants to get a decision about an ongoing PowerBoard game, a PowerBoardState must

be constructed and its board replaced with that of the PowerBoard game. The recursion depth must be set to the desired recursion depth, and after that, *decision* can be called.

When *decision* is called, the PowerBoardState instance creates one child state for each of the available sliding directions through the *move_with_deep_copy* method. It then proceeds to call each child's *expectimax_score* method, which generates a number representing its corresponding move direction's desirability. The direction of the child with the highest score is returned as the decision.

### 2.2 Expectimax Score Method

The *expectimax_score* function returns the average desirability score for the state on which it is called, given that a random tile appears in any open space on the board. The output of the function can be viewed as the value of the state's chance node in an expectimax tree. The function can be represented as follows:

$$\text{score}(\boldsymbol{A}) = \sum_{\boldsymbol{A'} \in \boldsymbol{A}} P(\boldsymbol{A'}, \boldsymbol{A}) \times \begin{cases} \text{terminal\_score(A')} & \text{if } \boldsymbol{A'} \text{ is terminal} \\ \max_{d \in D_{\boldsymbol{A'}}} \text{score}(\text{move}(\boldsymbol{A'}, d)) & \text{otherwise} \end{cases}$$

$\boldsymbol{A}$ is the current state of the board. $S_{\boldsymbol{A}}$ is the set of states that can be generated by randomly spawning a tile on any of the open spaces of $\boldsymbol{A}$. $D_{\boldsymbol{A}}$ are the directions in which it is possible to move given $\boldsymbol{A}$. $P(\boldsymbol{A'}, \boldsymbol{A}$ is the chance of reaching $\boldsymbol{A'}$ from $\boldsymbol{A}$, i.e. the chance of a tile spawning in one exact space multiplied by the chance of the exact value appearing.

A state is terminal if it has a recursion depth of 0 (decremented by 1 for each move) or if the game is over (all spaces full, no more tile merges possible). In that case, the state's terminal score is found, which is the heuristic function of the system.

### 2.3 Terminal Score – The Heuristic Function

In order to win a game of 2048, it is important that the high valued pieces are kept close to a wall, preferably in a corner. This is because the larger pieces are merged less frequently than smaller pieces. A large piece in the middle of the board will block opportunities to combine pieces with lower values.

One way to instruct a system about which positions are desirable is weighting the positions of the board. A higher weight designates a space in which it is desirable to have a piece with a higher value. Thus, the larger the piece in that position, the higher the perceived desirability of that state. This is the principle used by the heuristic function of this implementation, called *terminal_score*.

The weighting of the tiles in the implementation, represented by weighting matrix $\boldsymbol{W}$, looks as follows:

$$\boldsymbol{W} = \begin{pmatrix} r^0 & r^1 & r^2 & r^3 \\ r^7 & r^6 & r^5 & r^4 \\ r^8 & r^9 & r^{10} & r^{11} \\ r^{15} & r^{14} & r^{13} & r^{12} \end{pmatrix}$$

It consists of powers of a radix $r$ organized into an undulating shape. The reasoning behind the organization is: The most desirable place for the highest valued piece is in a corner. The most desirable position for the next largest piece is next to it, and the same reasoning goes for the third largest piece in relation to the next largest, and so on. Because we want to change the state of the board as little as possible when merging, the snake shape is a better alternative than another organization which places $r^i$ next to both $r^{i-1}$ and $r^{i+1}$, e.g. a spiral. (Note: I have found that $r = 4$ yields the best results.)

When *terminal_score* is run, it does a pointwise multiplication (denoted by $\circ$) of the values of the board with each possible permutation of the weighting matrix. These permutations consist of any matrix which can be achieved by number of rotations or transpositions of $\boldsymbol{W}$. The set of permutations is denoted by $P_{\boldsymbol{W}}$. The maximum of these pointwise products reflect the desirability of the state and is the heuristic value.

$$\text{terminal\_score}(\boldsymbol{A}) = \max_{\boldsymbol{W'} \in P_{\boldsymbol{W}}} \boldsymbol{W'} \circ \boldsymbol{A}$$

### 2.4 Variable recursion depth

To decrease running time while still achieving the 2048 tile consistently, the system has a variable recursion depth. This is a function of the number of remaining open spaces (few open spaces yields a higher recursion depth). The recursion depth is usually 1, but increases to 2 when five or fewer open spaces remain. When there are two or fewer open spaces, it increases to 3.

In case a move should merge so many tiles that a large number of open spaces become available, the recursion depth is adjusted down appropriately.

## 3 Closing remark

The game is played by an agent, found in the file *Agent2048.py*.