Assignment 1 Written Report

In the back2back piece model, I used two classes to represent the board and the pieces, respectively.

Each piece is represented by a two-dimensional list corresponding to the rows and columns of a piece. The depth of a given piece is recorded inside the list with 0 representing an empty part, 1 representing a flat part and 2 representing a taller part. For example, [[2,1,2],[0,0,1]] would correspond to the blue piece with one high part, one flat part and one high part again on the first row and only one flat part on the end of the second row. The piece class also records the color and side, front or back, of the piece. The front side is defined as placing the piece in a way such that the uneven side of the piece is facing the front side of the board. The back side or the flipped side indicates that the uneven side of the piece is facing the back side of the board. All pieces are initialized to be in the front state. The pieces could be flipped to a back state or a front state, which is indicated by a 1 or 0 correspondingly in the program. When a placement attempt is unsuccessful, the piece will always be left at the front state. When a piece is placed from the back side of the board, the side state of the piece will be 1 (the back side), but when the board is cleared, all pieces will return to the front state or 0 state. The pieces could be rotated for any number of times before being placed on the board. Each rotation rotates the piece clockwise for 90 degrees. There is no class variable to keep track of the rotation orientation of the piece. We will rotate any given piece as needed to place it on the board.

The board is represented by a 3-dimensional list. The first two dimensions correspond to the five rows and six columns of the board. The third dimension is a list of two strings showing the occupancy state of the board at the position specified by the row and column -- the first string shows the occupancy of the board from the front, i.e., if there is a piece sticking out viewing from the front side; the second string shows the occupancy from the back. When the current grid is empty, the innermost list is initialized as [‘emp’, ’emp’]. If a grid is occupied, the board will record the color abbreviation of the piece placed at the spot, for example, ‘p’ indicating the purple piece. The two strings end with a sign ‘+’ or ‘-’ indicating how the piece is inserted. For example, [‘pk1+’, ‘pk1+’] means one of the pink

pieces is placed at the current grid from the front side and the pink piece has the maximal depth such that it protrudes from both sides of the board. [‘emp’, ‘p-’] means that the current grid is occupied by a purple piece with a shorter depth so that it only protrudes from the back, and the piece is inserted from the back side. [‘y+’, ‘o-‘] would mean that a yellow piece with a shorter depth is placed from the front side and an orange piece with the same depth is placed from the back side into the same grid on the board.

The board class keeps track of the available pieces. When we try to place a used piece onto the board again, an error message “Not available, the piece has been used,” will be reported. The board also records a list of flipped pieces, i.e., the pieces being placed from the back side. When the board is cleared, all the flipped pieces will be returned to their original state facing the front.

The board cannot be flipped. When we need to place a piece from the back side, we can flip the piece and insert from the back. The placement is unsuccessful if any part of the piece is placed outside the 5X6 board. In that case, an IndexError will be raised and an error message “Illegal Position: off the edge,” will be reported. If we try to place a piece on top of a protruding part of another piece, the placement is also unsuccessful and the error message “Illegal Position: overlap not allowed at this position,” is reported.

Two example outputs are shown at the end of the report in Figure 1 and 2.

From this assignment, I realized the difficulties to convert a real world three-dimensional problem into a model that the computer could understand, because of the difference between how we perceive a real world object and how the code represents the object. This assignment gave me an opportunity to practice the ideas in problem identification and efficient knowledge representation. We need to identify the problem and the components that are crucial for us to solve the problems and ignore other properties that exist in the real world content but would be unnecessary and complicated for the program. For example, in the Back2Back puzzle, I could set up the board however I want when I play it physically. Nevertheless, in my program, the board will always be a five by six two dimensional arrays that cannot be rotated or flipped. We also need to be able to describe a problem accurately and store the information efficiently. For example, in my board representation, the two dimensional array contains another inner array to represent

the occupancy of any given grid. This might be potentially inefficient for more convoluted operations later, yet I could not think of a better way to represent all the information we need. This assignment served as a good lesson to show me that there is a long way to go and to learn in problem representations.

A possible heuristic function is f(n)=18-h(n), where h(n) is the number of empty spot left after putting any number of pieces in any 3X3 area containing the piece under consideration. One open side front or back will count as one open spot. In other words, after we put the piece currently under consideration with any orientation into the board from the front or the back side, we then consider any 3X3 area around the newly put in piece. The new piece has to be in the 3X3 area. Then we can try our best to fill the 3X3 area with all remaining unused pieces. After that we count the number of empty spot that could not be filled in. When all spots in the 3X3 area could be filled in, the heuristic function will be 18, the maximum, which is the most promising condition. When f(n) is less than 18, it means that there is at least one spot that could never be filled if we place the current piece this way. The lower the f(n) value, the less promising the current placement attempt would be. In this function, we do not distinguish how many additional pieces we used to fill the 3X3 area in the hope that we would not overestimate the cost of the current placement attempt.

Figure 1. Example Text Interface Output 1

```
Pieces initialized

2 Sides Board Occupancy Display :      --format-- (front,back), '+'placing from front side of
 the board, '-'placing from back of the board

[emp,emp]   [emp,emp]   [emp,emp]   [emp,emp]   [emp,emp]   [emp,emp]
[emp,emp]   [emp,emp]   [emp,emp]   [emp,emp]   [emp,emp]   [emp,emp]
[emp,emp]   [emp,emp]   [emp,emp]   [emp,emp]   [emp,emp]   [emp,emp]
[emp,emp]   [emp,emp]   [emp,emp]   [emp,emp]   [emp,emp]   [emp,emp]
[emp,emp]   [emp,emp]   [emp,emp]   [emp,emp]   [emp,emp]   [emp,emp]
Placement succeeded: piece pk1

Placement succeeded: piece pk2

Illegal Position: overlap not allowed at this position.
Placement failed: piece y Try Again.

Illegal Position: off the edge.
Placement failed: piece o Try Again.

Illegal Position: off the edge.
Placement failed: piece p Try Again.

Not availble: this piece pk1  has been used.

2 Sides Board Occupancy Display :      --format-- (front,back), '+'placing from front side of
 the board, '-'placing from back of the board

[emp,emp]   [emp,emp]   [emp,emp]   [emp,emp]   [emp,emp]    [emp,emp]
[emp,emp]   [emp,emp]   [emp,emp]   [emp,emp]   [pk2-,pk2-]   [pk2-,pk2-]
[emp,emp]   [emp,emp]   [emp,emp]   [emp,emp]   [emp,emp]    [pk1+,pk2-]
[emp,emp]   [emp,emp]   [emp,emp]   [emp,emp]   [pk1+,pk1+]   [pk1+,pk1+]
[emp,emp]   [emp,emp]   [emp,emp]   [emp,emp]   [emp,emp]    [emp,emp]
```

Figure 2. Example Text Interface Output 2

```
/Library/Frameworks/Python.framework/Versions/2.7/bin/python2.7 "/Users/mac/Desktop
/CSC 339 AI/hw1/Board.py"
Pieces initialized

2 Sides Board Occupancy Display :      --format-- (front,back), '+'placing from front side of
 the board, '-'placing from back of the board

[emp,emp]    [emp,emp]    [emp,emp]    [emp,emp]    [emp,emp]    [emp,emp]
[emp,emp]    [emp,emp]    [emp,emp]    [emp,emp]    [emp,emp]    [emp,emp]
[emp,emp]    [emp,emp]    [emp,emp]    [emp,emp]    [emp,emp]    [emp,emp]
[emp,emp]    [emp,emp]    [emp,emp]    [emp,emp]    [emp,emp]    [emp,emp]
[emp,emp]    [emp,emp]    [emp,emp]    [emp,emp]    [emp,emp]    [emp,emp]
Placement succeeded: piece  lb

Placement succeeded: piece  dg

Placement succeeded: piece  pk1

Placement succeeded: piece  pk2


2 Sides Board Occupancy Display :      --format-- (front,back), '+'placing from front side of
 the board, '-'placing from back of the board

[lb-,lb-]    [emp,emp]    [emp,emp]    [emp,emp]    [emp,emp]    [emp,emp]
[emp,lb-]    [emp,emp]    [emp,emp]    [emp,emp]    [pk2-,pk2-]    [pk2-,pk2-]
[lb-,lb-]    [dg-,dg-]    [emp,emp]    [emp,emp]    [emp,emp]    [pk1+,pk2-]
[dg-,dg-]    [emp,dg-]    [emp,emp]    [emp,emp]    [pk1+,pk1+]    [pk1+,pk1+]
[emp,emp]    [emp,emp]    [emp,emp]    [emp,emp]    [emp,emp]    [emp,emp]

Process finished with exit code 0
```