CITS4211 Artificial Intelligence

**Project – Tetris**

**Report**

LIM, Kian Aik (Johnathan) (20687818)

TSE, Tai Loi (Tyler) (20893048)

05/2013

# Overview

To the requirement of the project, our quest is to develop a tetris player. As to our knowledge, the default requirement of this tetris game is allowed to have at least: i) One buffer size; ii) 11 blocks of width and semi-infinite height; iii) 7 unique tetrominoes and; iv) 4 rotations (except “I”, “O”, “S”, “Z” blocks). With such default setting the possible state can reach up to a really gigantic number. So to achieve such program we had come to consensus that we will need to implement a search algorithm based on a few sets of well-set heuristics.

During the game play, our agent will assign the block into every single space. Possible state will range from 9 (“O” block) to 32 (“L”, “J” and “T” block) with each tetromino, and same with the buffer block as well.

With all the possible state laid out, the agent will use 4 sets of heuristics to determine the optimal dropping spot. The four sets of heuristics are i) Total Height; ii) Flatness; iii) Covered holes and; iv) Rows Cleared. Assigning marks for each heuristics and to let the search algorithm to take use of.

# Tetris Board implementation

We had separated the tetris board into two classes: i) board and; ii) block. Each do separate job handling the whole board and complementing each other.

For the board class, it handles the board as a stack ADT storing it locally and with the method popBoard it will return a 2D-array of Boolean. The reason of using Boolean rather than char or int is because of the size, as most virtual machine will have a smaller size storing Boolean, and the handling of the array can also be simplify, as every call to the element will return the actual Boolean rather than making a comparable (e.g. int==1). The returning of popBoard is optimized to handle the adding of blocks into the board, it will only pop lines until each column contains at least a true, else it will pop till the bottom. It will also add four line of false on top of the array, which will act as a buffer if block “I” need to be place in rotation 0 or 2 on the highest block.

|  |  |  |
| --- | --- | --- |
| Macintosh HD:Users:JohnathanMBP:Downloads:board.png |  | Macintosh HD:Users:JohnathanMBP:Downloads:poped.png |
| The original board | popBoard | Board being popped out to a 2D Boolean |

In the pushBoard of the implementation, the program will ignore the blank buffer space and also line that has been clear and the whole processed board will be push back into the Stack.

|  |  |  |
| --- | --- | --- |
| Macintosh HD:Users:JohnathanMBP:Downloads:played.png |  | Macintosh HD:Users:JohnathanMBP:Downloads:playprocessed.png |
| The played board | pushBoard | Board being pushed back to the stack |

# Tetris Board implementation (continue)

For the block class, we have implemented it with switch. Such function can reduce the time to iterate through library of tetrominoes and just return what ever it is needed.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Block ID | Block name | Rotation 0  0° | Rotation 1  90°CW | Rotation 2  180°CW | Rotation 3  270°CW |
| **1** | **I** | |  |  | | --- | --- | |  | 0 | | 0 |  | | 1 |  | | 2 |  | | 3 |  | | |  |  |  |  |  | | --- | --- | --- | --- | --- | |  | 0 | 1 | 2 | 3 | | 0 |  |  |  |  | | |  |  | | --- | --- | |  | 0 | | 0 |  | | 1 |  | | 2 |  | | 3 |  | | |  |  |  |  |  | | --- | --- | --- | --- | --- | |  | 0 | 1 | 2 | 3 | | 0 |  |  |  |  | |
| **2** | **O** | |  |  |  | | --- | --- | --- | |  | 0 | 1 | | 0 |  |  | | 1 |  |  | | |  |  |  | | --- | --- | --- | |  | 0 | 1 | | 0 |  |  | | 1 |  |  | | |  |  |  | | --- | --- | --- | |  | 0 | 1 | | 0 |  |  | | 1 |  |  | | |  |  |  | | --- | --- | --- | |  | 0 | 1 | | 0 |  |  | | 1 |  |  | |
| **3** | **T** | |  |  |  | | --- | --- | --- | |  | 0 | 1 | | 0 |  |  | | 1 |  |  | | 2 |  |  | | |  |  |  |  | | --- | --- | --- | --- | |  | 0 | 1 | 2 | | 0 |  |  |  | | 1 |  |  |  | | |  |  |  | | --- | --- | --- | |  | 0 | 1 | | 0 |  |  | | 1 |  |  | | 2 |  |  | | |  |  |  |  | | --- | --- | --- | --- | |  | 0 | 1 | 2 | | 0 |  |  |  | | 1 |  |  |  | |
| **4** | **J** | |  |  |  | | --- | --- | --- | |  | 0 | 1 | | 0 |  |  | | 1 |  |  | | 2 |  |  | | |  |  |  |  | | --- | --- | --- | --- | |  | 0 | 1 | 2 | | 0 |  |  |  | | 1 |  |  |  | | |  |  |  | | --- | --- | --- | |  | 0 | 1 | | 0 |  |  | | 1 |  |  | | 2 |  |  | | |  |  |  |  | | --- | --- | --- | --- | |  | 0 | 1 | 2 | | 0 |  |  |  | | 1 |  |  |  | |
| **5** | **L** | |  |  |  | | --- | --- | --- | |  | 0 | 1 | | 0 |  |  | | 1 |  |  | | 2 |  |  | | |  |  |  |  | | --- | --- | --- | --- | |  | 0 | 1 | 2 | | 0 |  |  |  | | 1 |  |  |  | | |  |  |  | | --- | --- | --- | |  | 0 | 1 | | 0 |  |  | | 1 |  |  | | 2 |  |  | | |  |  |  |  | | --- | --- | --- | --- | |  | 0 | 1 | 2 | | 0 |  |  |  | | 1 |  |  |  | |
| **6** | **S** | |  |  |  | | --- | --- | --- | |  | 0 | 1 | | 0 |  |  | | 1 |  |  | | 2 |  |  | | |  |  |  |  | | --- | --- | --- | --- | |  | 0 | 1 | 2 | | 0 |  |  |  | | 1 |  |  |  | | |  |  |  | | --- | --- | --- | |  | 0 | 1 | | 0 |  |  | | 1 |  |  | | 2 |  |  | | |  |  |  |  | | --- | --- | --- | --- | |  | 0 | 1 | 2 | | 0 |  |  |  | | 1 |  |  |  | |
| **7** | **Z** | |  |  |  | | --- | --- | --- | |  | 0 | 1 | | 0 |  |  | | 1 |  |  | | 2 |  |  | | |  |  |  |  | | --- | --- | --- | --- | |  | 0 | 1 | 2 | | 0 |  |  |  | | 1 |  |  |  | | |  |  |  | | --- | --- | --- | |  | 0 | 1 | | 0 |  |  | | 1 |  |  | | 2 |  |  | | |  |  |  |  | | --- | --- | --- | --- | |  | 0 | 1 | 2 | | 0 |  |  |  | | 1 |  |  |  | |

# Tetris Board implementation (continue)

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  |  | | --- | --- | --- | --- | |  |  |  |  | |  |  |  |  | |  |  |  |  | |  |  |  |  | |  | |  | | --- | |  | |  | |  | |  | |
| |  |  |  |  | | --- | --- | --- | --- | |  |  |  |  | |  |  |  |  | |  |  |  |  | |  |  |  |  | |  | |  |  | | --- | --- | |  |  | |  |  | |
| |  |  |  |  | | --- | --- | --- | --- | |  |  |  |  | |  |  |  |  | |  |  |  |  | |  |  |  |  | |  | |  |  | | --- | --- | |  |  | |  |  | |  |  | |
| |  |  |  |  | | --- | --- | --- | --- | |  |  |  |  | |  |  |  |  | |  |  |  |  | |  |  |  |  | |  | |  |  | | --- | --- | |  |  | |  |  | |  |  | |
| |  |  |  |  | | --- | --- | --- | --- | |  |  |  |  | |  |  |  |  | |  |  |  |  | |  |  |  |  | |  | |  |  | | --- | --- | |  |  | |  |  | |  |  | |
| |  |  |  |  | | --- | --- | --- | --- | |  |  |  |  | |  |  |  |  | |  |  |  |  | |  |  |  |  | |  | |  |  | | --- | --- | |  |  | |  |  | |  |  | |
| |  |  |  |  | | --- | --- | --- | --- | |  |  |  |  | |  |  |  |  | |  |  |  |  | |  |  |  |  | |  | |  |  | | --- | --- | |  |  | |  |  | |  |  | |

To handle the block we have choose to return a 2D-array of Boolean of the exact size of the block instead of the tradition way of a 4 by 4 array, such handling is to reduce the size of each block, and reduce the time to search for a reference point of the block

# Program Code

# Structure and Design of the Program

# Experimental and Theoretical analysis

# Progress Documentation

Thought process:

To evaluate the position of the piece to be drop, we first have to list up the conditions we need to consider to put on the board. The four conditions we have to evaluate:

1. Total Height
2. Flatness
3. Covered Holes
4. Rows Clear

First code we need to build is the board physics and all pieces physics.

Building Each Pieces of tetris size and rotation

Using java J unit test class to test out each of the block.

We choose to use Boolean array instead of integer array because Boolean has less size than integer that may allow the system to run faster.

Building a class called line that represented each line of the tetris board

Building board & the ADT to store the line