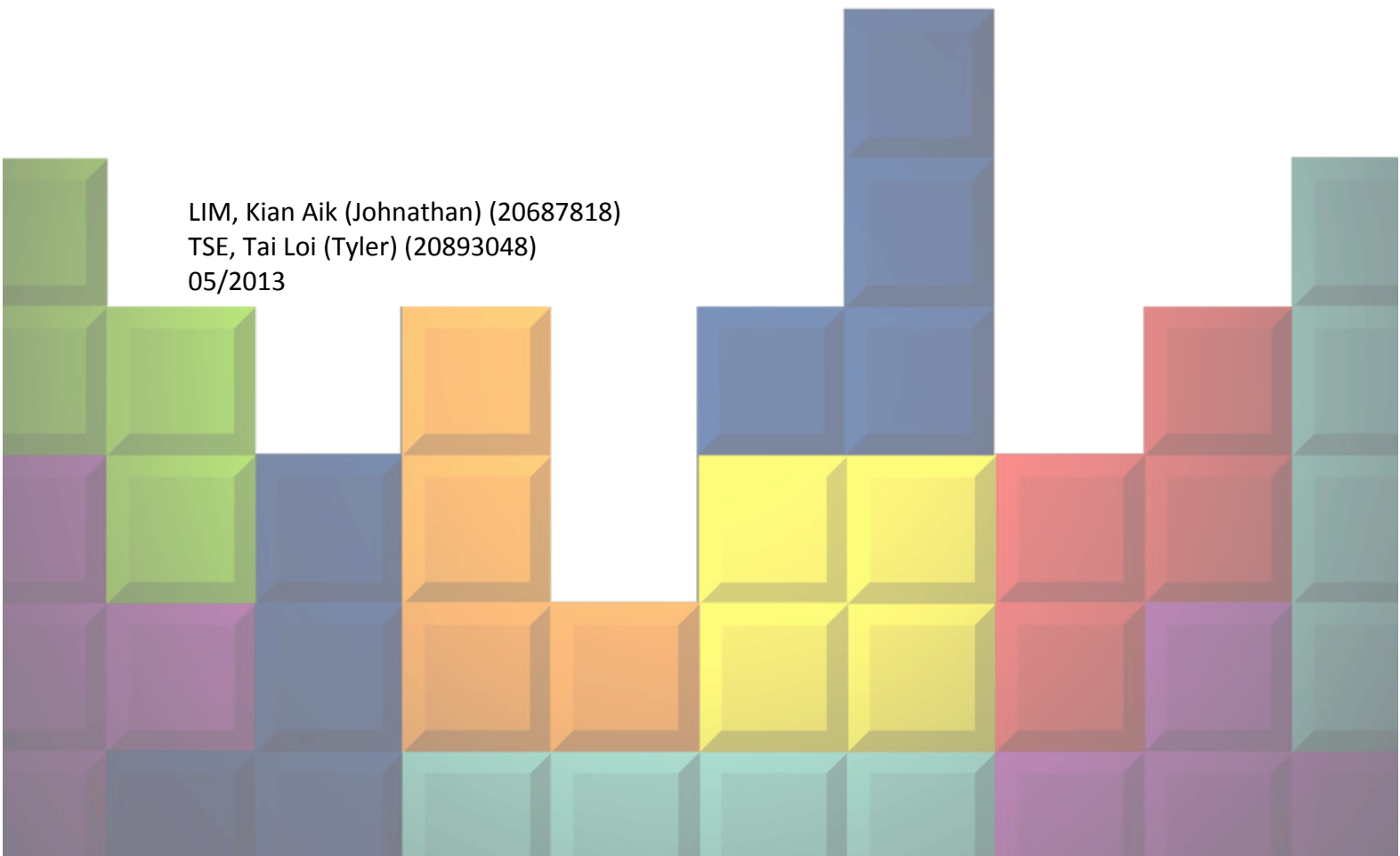


# CITS4211 Artificial Intelligence

## Project – Tetris

## Report

LIM, Kian Aik (Johnathan) (20687818)  
TSE, Tai Loi (Tyler) (20893048)  
05/2013

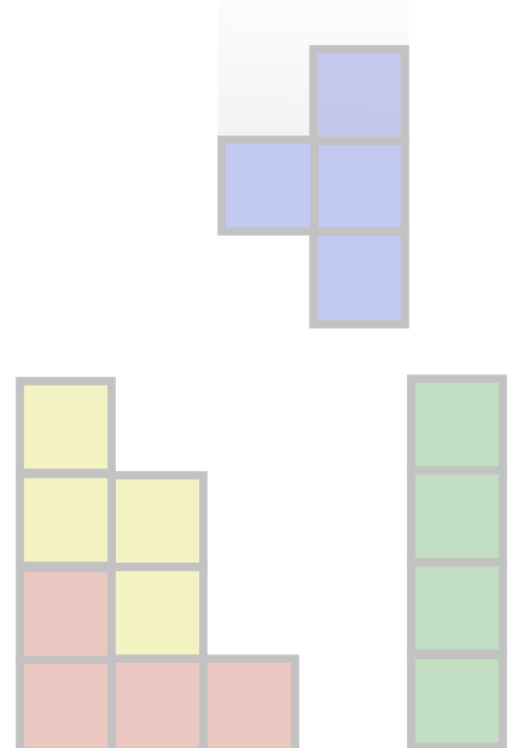


## Overview

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 quite a big number. Although it might reach over 100 states for the inputted block and buffered block to be placed, however the agent will still able to brute force through the best solution with sets of well-defined heuristic. 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-defined heuristics.

During the game play, our agent will assign the block into every single space. Possible state will range from 10 (“O” block) to 33 (“L”, “J” and “T” block) with each tetromino, and same with the buffer block as well. With the number of

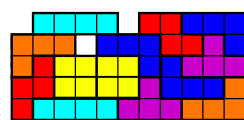
With all the possible state laid out, the agent will use 4 sets of heuristics to determine the optimal dropping spot. The four sets of heuristics are i) Total Height; ii) Flatness; iii) Covered holes; iv) Rows Cleared and; v) On the Left or Right Most. Assigning marks for each heuristics and to let the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

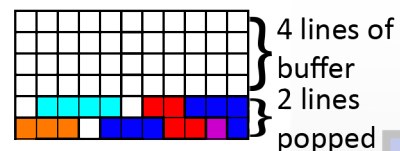
For the board class, it handles the board as a stack ADT storing it locally and with the method `popBoard` it will return a 2D-array of Boolean. The reason to use stack is because of the First-In-Last-Out feature and can stack limitless line of array to the data structure, which fits the requirement for a semi-infinite height. And for 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.



The original board

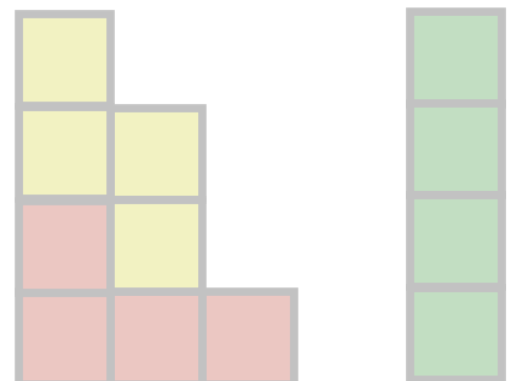
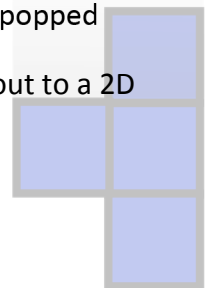


`popBoard`



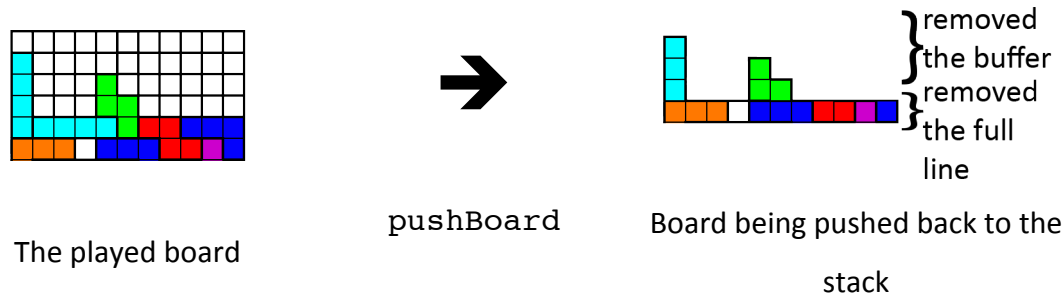
Board being popped out to a 2D

Boolean


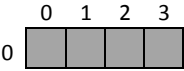

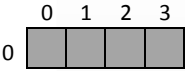
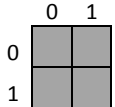
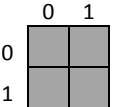
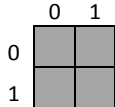
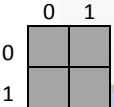
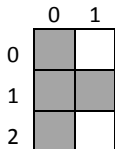
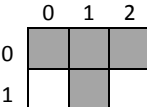
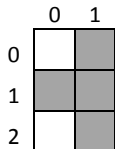
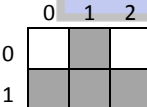
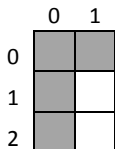
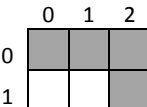
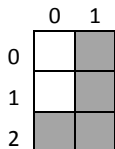
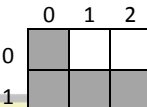


## Tetris Board implementation (continue)

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.

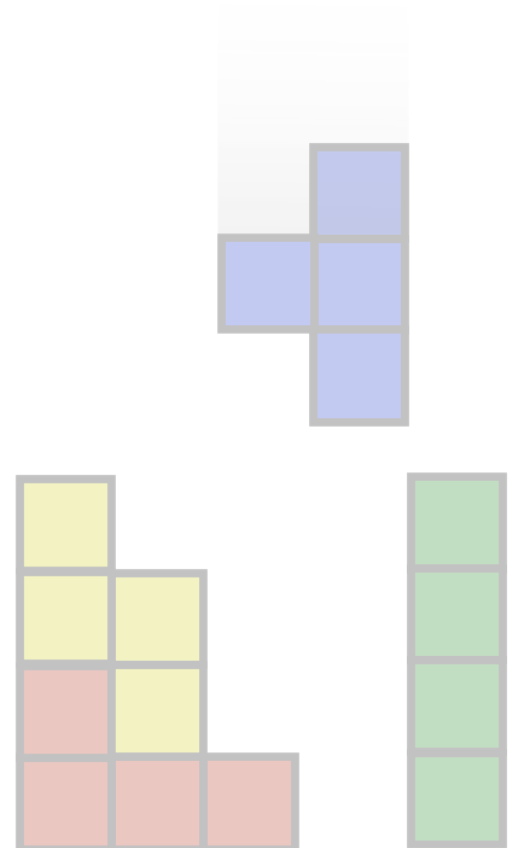


For the `block` class, we have implemented it with `switch`. Such function can reduce the time to iterate through library of tetrominoes as it is only  $O(n)$  and just return what ever it is needed.

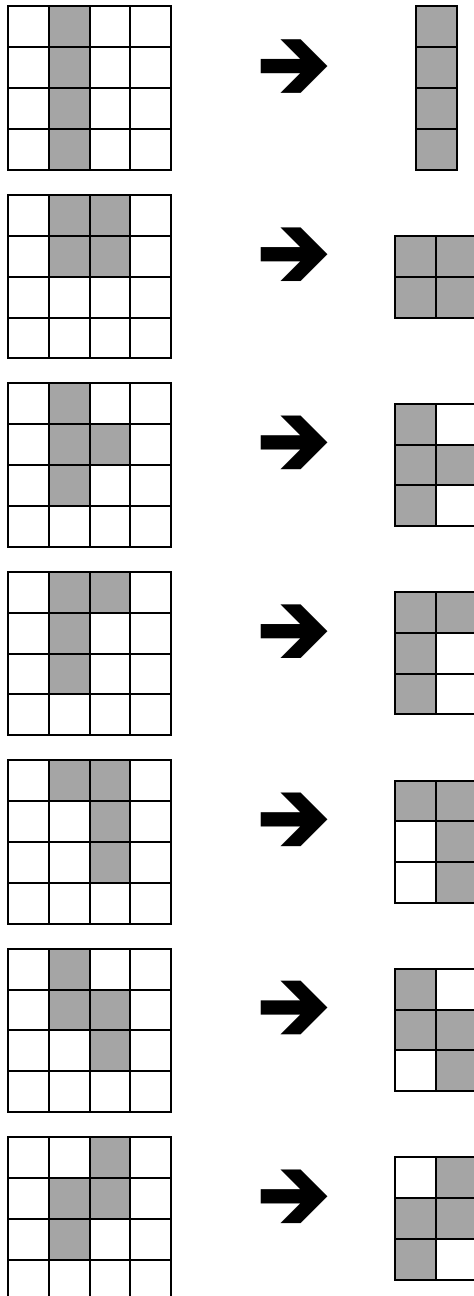
Block ID	Block name	Rotation 0 0°	Rotation 1 90°CW	Rotation 2 180°CW	Rotation 3 270°CW
1	I				
2	O				
3	T				
4	J				

## Tetris Board implementation (continue)

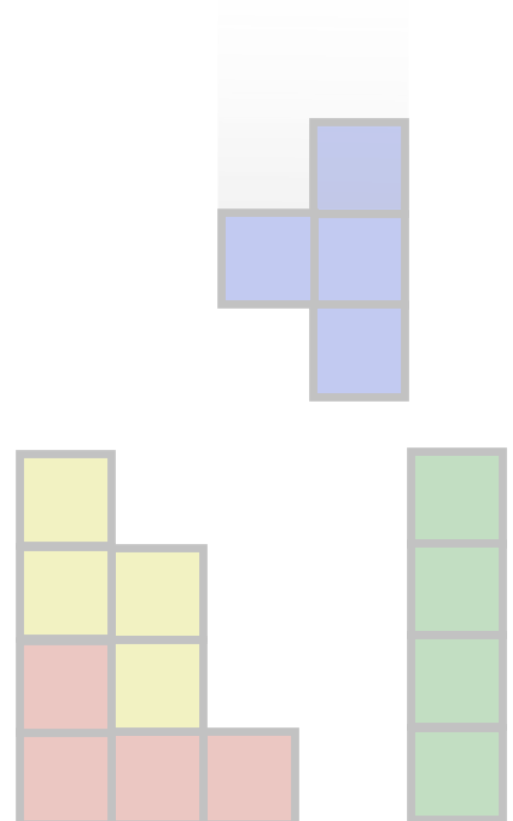
<b>5</b>	<b>L</b>	0 1			0 1 2				0 1			0 1 2		
		0							0					
		1							1					
		2							2					
<b>6</b>	<b>S</b>	0 1			0 1 2				0 1			0 1 2		
		0							0					
		1							1					
		2							2					
<b>7</b>	<b>Z</b>	0 1			0 1 2				0 1			0 1 2		
		0							0					
		1							1					
		2							2					



## Tetris Board implementation (continue)

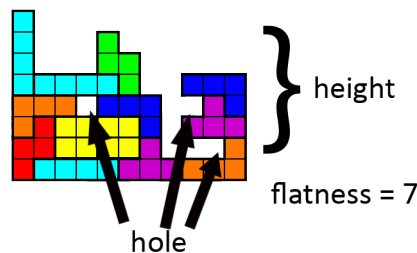


To handle the block we have choose to return a 2D-array of Boolean of the exact size of the block instead of the tradition way of a 4 by 4 array, such handling is to reduce the size of each block, and reduce the time to search for a reference point of the block. But the tradeoff of such implementation is the need to implement a different code per rotation of block, however this can ensure the  $O(n)$  complexity of such conversion.



## Implementation of Artificial Intelligence

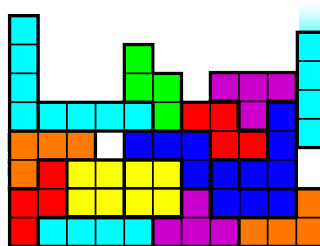
We determine whether each block to be placed in the board with the five heuristic and with each assigning mark due to the placement of the block:



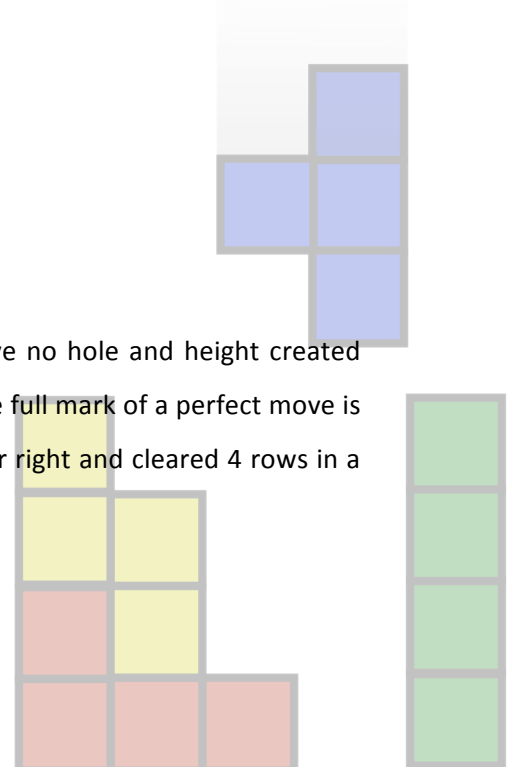
1. Number of holes (`getHole`)
2. Highest height and lowest height difference (`getFlatness`)
3. Height of block (`getHeight`)
4. Cleared rows (`getCleared`)
5. Block will be placed on the left or right most (`isOnRight/isOnLeft`)

Score will be assigning due to the placement of the block and the equation will be as below:

$$-((getHole + getFlatness) \times getHeight \times 2) + getCleared \times 50 + isOnRight + isOnLeft$$



The best case is place a block on the far right or left and to have no hole and height created while having row(s) cleared. So with such scenario, we assume the full mark of a perfect move is 201 marks. This is, a block "I" being placed vertically on the left or right and cleared 4 rows in a move.

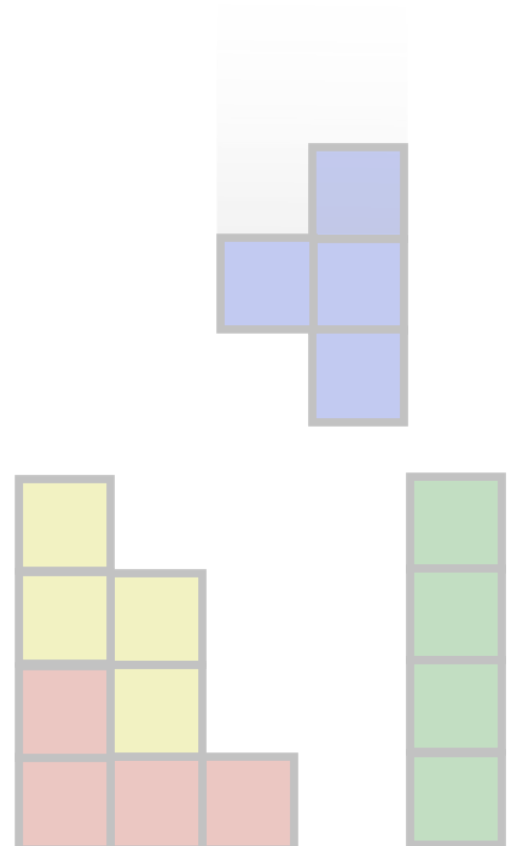




# Implementation of Artificial Intelligence

(continue)

With 201 marks in set, the agent will search through all the possible moves and with prioritized rules.





## 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

