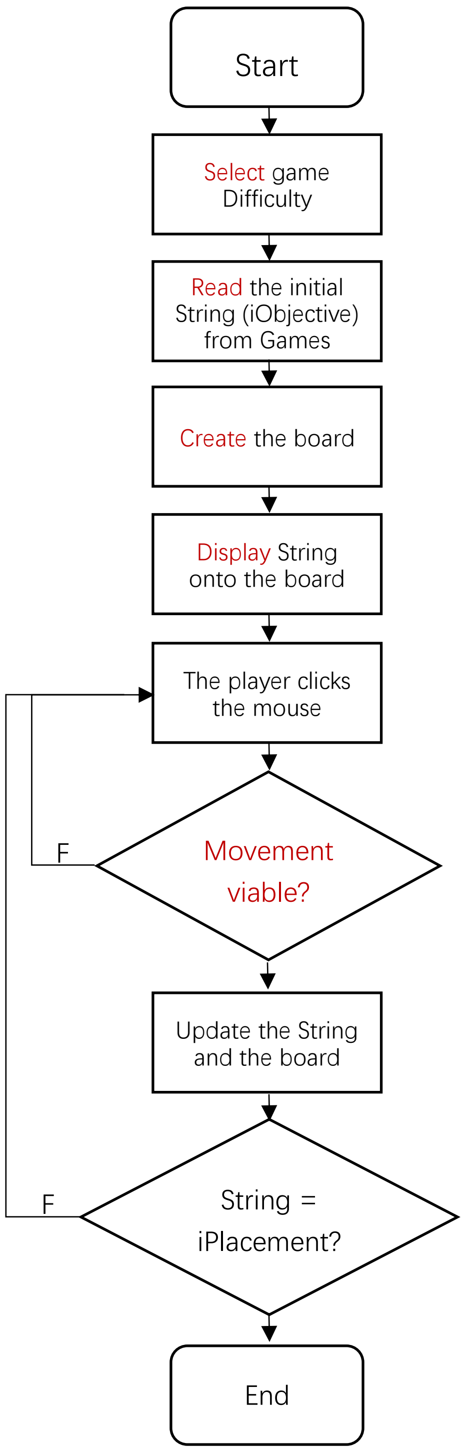
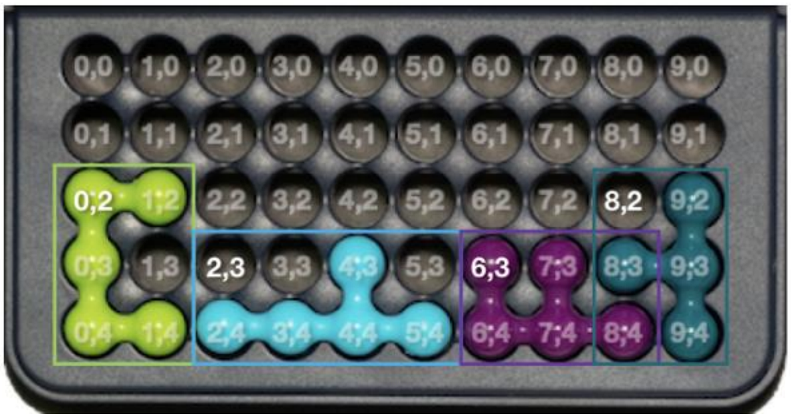
1. Flow chart of game operation



Start
Select game Difficulty
Read the initial String (iObjective) from Games
Create the board
Display String onto the board
The player clicks the mouse
Movement viable?
F
Update the String and the board
String = iPlacement?
F
End

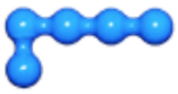
2. Main logical strategy

The key question of the game is how to determine if the placement of a piece is viable. Our strategy is to represent every position on the board with a Boolean, two-dimensional array Board[5][10], for the following statement, the Boolean array is:



F	F	F	F	F	F	F	F	F	F
F	F	F	F	F	F	F	F	F	F
T	T	F	F	F	F	F	F	F	T
T	F	F	F	T	F	T	T	T	T
T	T	T	T	T	T	T	T	T	T

Meanwhile, we need to create a new type Spotsposition[4][4] to describe the spot that a piece will occupy. These field will be stored in a class Piece as the piece are in the rotations of N, and the two flips of a piece is stored separately in two different instances. For example, the following piece is stored as two instances:



Piece b

T	T	T	T
T	F	F	F
F	F	F	F
F	F	F	F

Piece B

T	T	T	T
F	T	F	T
F	F	F	F
F	F	F	F

When we need to judge if a movement is viable, we can simply read the String, find the rotation and upper left spot of the piece and then check if any true spot conflict with the true spot currently in the Board[][].

3. The classes that need to be included

```
/*
 * This enumeration type represents the four cardinal directions of a piece
*/
public enum Direction {
        NORTH('N'), EAST('E'), SOUTH('S'), WEST('W');
         final private char symbol;

        Direction(char symbol) {}
        public static Direction fromChar(char direction){}
        public char toChar(){}
/**
 * Given an direction,
 * return the next direction as piece rotates 90 degree.
 * e.g. N->E->S->W->N
 * @param current direction.
 * @return the next `Direction`.
 */
        public Rotate(Direction Dir)(){} //Let the piece turn 90 degree
}

/*
 * This class define a piece
*/

public class Piece {

private final char pieceName;
public final boolean Spotsposition[][] = new boolean[4][4];
private Direction orientation = Direction.NORTH; //Default orientation is North
private int LeftPos = -1, UpperPos = -1; //This field record theUpLeft position of the piece if
it has been put on the board, otherwise it will be -1

public Piece(PieceName pieceName, boolean Spotsposition[][]){}

/**
 * Given the new direction and position of the piece,
 * update the orientation and LeftPos, UpperPos field of the piece.
 *
 * @param the orientation and position that want the piece put into.
 * @return void.
 */
public void ChangeDirandPos(String PieceStatement){}
public char getPieceName(){}
public char getOrientation(){}
```

```
/**
 * Calculate the Boolean table according to the orientation
 * and the Spotsposition array of the piece
 * The method includes rotation and translation of the
 * array’s valu
 *
 * @param None.
 * @return A 4*4 boolean array that describe the piece
 * in the current orientation.
 */
public boolean[][] GetCurrentPos(){}
/**
 * Check if the piece has been used in another flip
 * e.g. For piece B, if b is already in the String, return
 * false
 * @param the current String that describe piece on the board.
 * @return the answer.
 */
public boolean IsPiecetaken(String CurrentOnboard){}

/**
 * Transfer a piece's statment to String such as b73E
 *
 * @param: null.
 * @return the 4 char String that include position, name and orientation.
 */
public String PiecetoString(){}
}

/**
 * This class provides the text interface for the IQ Fit Game
 * There are some other methods and field that we need to add to this class
*/
public class FitGame {
public boolean Board[][] = new Boolean[5][10]

/**
 * Add a new piece on the board, this will include checking
 * if the placement viable. If so, the method will update
 * the board[][] as well as update the string
 *
 * @param: the current String that describe piece on the
 *         board, the piece that want to add
 * @return The new string, it will not change if the placement
 *          is not viable.
 */
public String AddtoBoard(String currentString, Piece PieceName){}

/**
 * Move a piece from the board, this will include surch
 * the piece from String. If find the piece, the method will update
 * the board[][] as well as update the string
 *
 * @param: the current String that describe piece on the
 *         board, the piece that want to move away
 * @return The new string, it will not change if the placement
 *          is not viable.
 */
public String MovefromBoard(String currentString, Piece PieceName){}

/**
 * Read the current string and update the board
 * This method is prepared for initialize the game
 * @param: the current String that describe piece on the
 *         board
 * @return void
 */
public void StringToBoard(String currentString, Piece PieceName){}

}
```