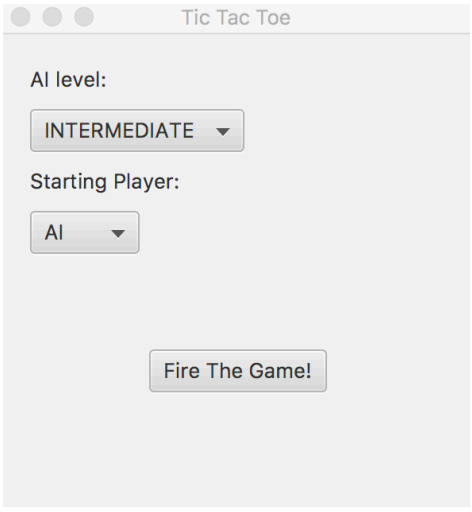
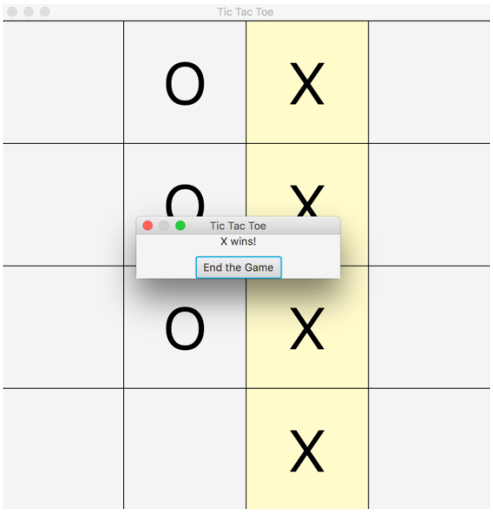
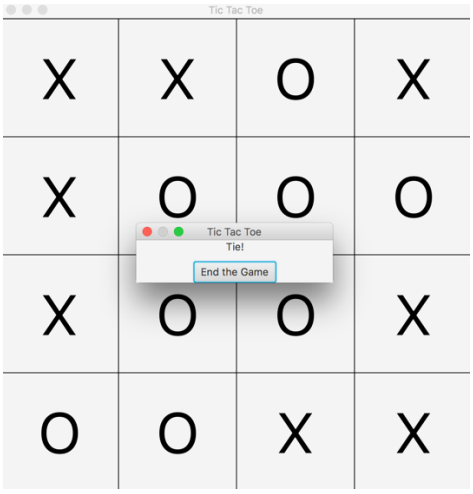
Tic Tac Toe

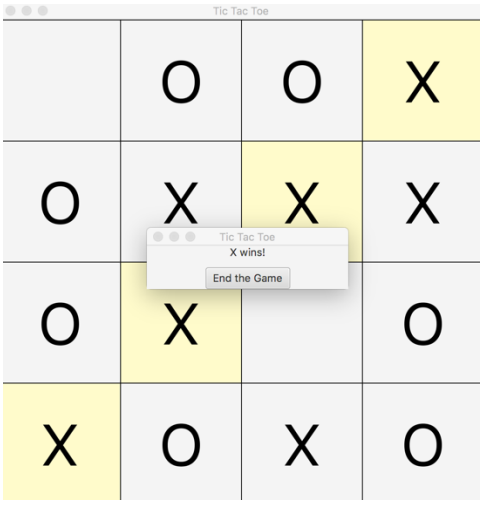
1. Introduction

Tic-tac-toe is a game for two players, ‘X’ and ‘O’, who take turns marking the spaces in a 4 * 4 grid. In this project, Human player uses ‘O’ and Artificial Intelligence(AI) player uses ‘X’. AI is implemented with ALPHA-BETA search algorithm. When the game launched, user could choose the AI level and starting player. The goal of user is placing 4 ‘O’s that lines up vertically, horizontally or diagonally.









2. Source Code

Source code are zipped together with this document and it could be found at my GitHub page as well (https://github.com/BoogieJay/TicTacToe-AI). Below are some important designing aspects of Tic Tac Toe.

2.1 Minimax with Alpha-Beta Pruning.

When very time AI decides where to place the next move, the function minimax will be called. Then the getMaxValue and getMinValue will be called in turns. My code is implemented according to the pseudo code below.

```
function ALPHA-BETA-SEARCH(state) returns an action
    v ← MAX-VALUE(state, min_utility, max_utility)
    return the action in ACTIONS(state) with value v

function MAX-VALUE(state, α, β) returns a utility value
    if TERMINAL-TEST(state) then return UTILITY(state)
    v ← -∞
    for the next a in ACTIONS(state) do
        v ← MAX(v, MIN-VALUE(RESULT(state, a), α, β))
        if v ≥ β then return v
        α ← MAX(α, v)
    return v

function MIN-VALUE(state, α, β) returns a utility value
    if TERMINAL-TEST(state) then return UTILITY(state)
    v ← +∞
    for the next a in ACTIONS(state) do
        v ← MIN(v, MAX-VALUE(RESULT(state, a), α, β))
        if v ≤ α then return v
        β ← MIN(β, v)
    return v
```

2.2 AI level design.

In the project, one of the levels among 'EASY', 'INTERMEDIATE' AND 'DIFFICULT' could be chosen. They are implemented according to different evaluation function and depth of searching.

The evaluation function is Eval = $C_1 * X_3 + C_2 * X_2 + C_3 * X_1 – (C_1 * O_3 + C_2 * O_2 + C_3 * O_1)$. And level constants C_1, C_2, C_3 are assigned differently according to certain level.

	(C_1, C_2, C_3)	Search Depth
EASY	(1, 3, 6)	1
INTERMEDIATE	(1, 0, 0)	1
DIFFICULT	(6, 3, 1)	Infinite if no cutoff occurs

X_3: three X and one Empty cell X_2: two X and two Empty cells X_1: one X and three Empty cells
O_3: three O and one Empty cell O_2: two O and two Empty cells O_1: one O and three Empty cells

As for the performance of different levels, 'EASY' level AI will prefer more X_3 than X_2 than X_1, 'INTERMEDIATE' level AI will prefer more X_3 but is blind to X_2 and X_1, 'DIFFICULT' level AI will follow the ALPHA-BETA search algorithm but with cutoff.

My implementation:

```java
/**
 * return factors according to current level
 * @return factors
 */
private int[] levelFactor() {
    switch (level) {
        case EASY: return new int[]{1, 3, 6};
        case INTERMEDIATE: return new int[]{1, 0, 0};
        case DIFFICULT: return new int[]{6, 3, 1};
        default: return new int[]{6, 3, 1};
    }
}
```

(C_1, C_2, C_3)

```java
/**
 * Level class represents three AI levels that human could choose to play with.
 * Each level also has a depth value to limit the maximum search depth while searching
 *
 * Created by BoogieJay
 * 4/24/17.
 */
public enum Level {
    DIFFICULT(Integer.MAX_VALUE), INTERMEDIATE(1), EASY(1);

    private final int depth;

    Level(int depth) { this.depth = depth; }

    public int getDepth() { return this.depth; }
}
```

Depth

2.3 Cutoff design.

If the search in Alpha-Beta algorithm exceeds 1 second, the rest search will be run as Alpha-Beta algorithm with limited depth 7. In such case, the maximum running time for search could be controlled within 4 seconds.

My implementation:

```java
/**
 * check if time duration reach the CUT_OFF_START_TIME
 * if reached, set statType.needCutOff = true
 * also return the time passed so far from beginning.
 *
 * @return the time passed so far
 */
private int checkTimeOut() {
    int currentSecond = LocalDateTime.now().getSecond();
    int diff = 0;
    diff = currentSecond - startSecond;
    if (currentSecond < startSecond) {
        diff += 60;
    }
    if (diff > CUT_OFF_START_TIME) {
        statType.setCutOff(true);
    }
    return diff;
}
```

check timeout

```
// if the cutoff already happened and the rest search depth deeper than CUT_OFF_LEVEL, just set it to CUT_OFF_LEVEL
if (statType.getNeedCutOff() && currentLevel > CUT_OFF_LEVEL) {
    currentLevel = CUT_OFF_LEVEL;
}
```

if time out, limited the rest search depth no larger than 7

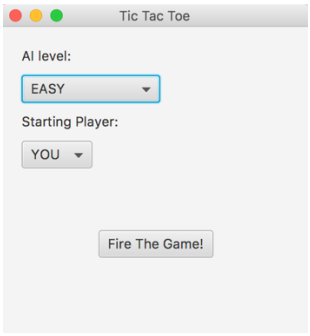
2.4 User Interface

The user interface is designed by using JavaFX. The project has three windows which are welcome window, board window and alert window.

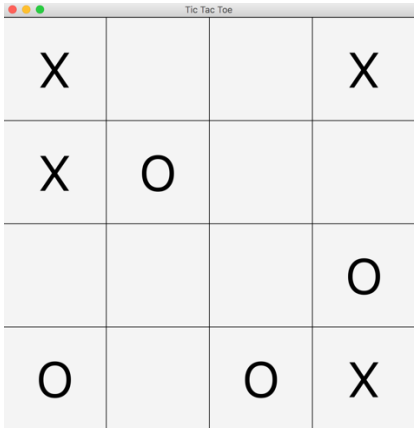
Welcome window: retrieve information of AI level and Starting player

Board window: show the board for playing

Alert window: show the result and close the program



Welcome Window



Board Window

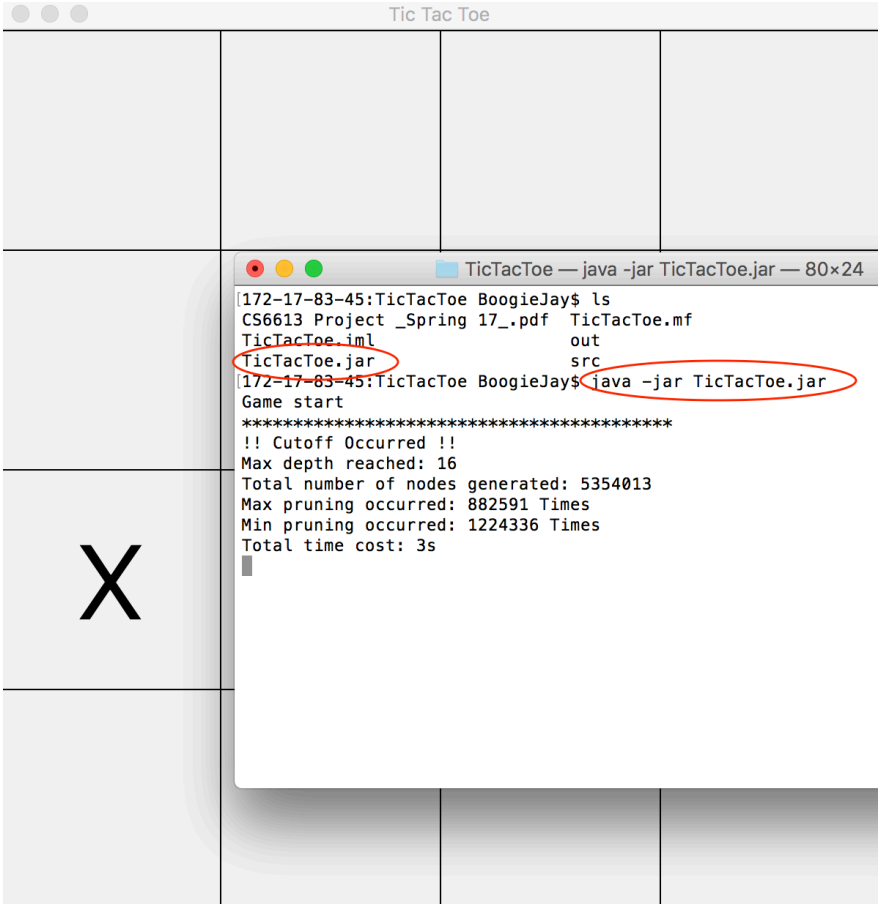
Alert Window

3. Run the Tic Tac Toe

The easiest way to run this game is just **double clicking** the TicTacToe.jar file.

If you want to run this game along with statistic information as required, you could first make sure that you are in the **folder containing the TicTacToe.jar file**, then input the following line into your command line

Java –jar TicTacToe.jar



Statistic information shown at the console

4. Conclusion

The Tic Tac Toe game shows how Alpha-Beta algorithm gives the computer the 'intelligence' to compete with human. It also reveals how basic AI algorithm works. The game performs well overall, but it seems cost too much time at the first and second steps when Hard level is chosen.