Reykjavík University

Software Engineering

T-303-HUGB

Late-term Assignment

Group: [PLACEHOLDER]
Andri Már Sigurðsson
Georg Styrmir Rúnarsson
Hafdís Erla Helgadóttir
Hinrik Már Hreinsson
Karl Valdimar Kristinsson

Teacher: Hannes Pétursson

1 Introduction

The following is a design report for a game of Tic-Tac-Toe, which is being created as an assignment in a Software Engineering course at Reykjavík University. The game is written in Java, and is supposed to be extremely simple to use with a minimal user interface. Each individual member will be free to choose their own IDE to write their code in, with Eclipse and IntelliJ being the most probable choices.

For instructions on how to get the project to build on a fresh machine, see the accompanying Development Manual. For instructions on how to get the project to run, see the Administration Manual.

Prototypes (or "mockups") for how the game should look can be seen in Chapter 5, and descriptions of the rules of the game and the proposed AI can be seen in chapters 2 and 3 respectively. In chapter 4 we list our Coding Conventions and other methodologies for the project.

The last chapter shows a list of requirements for the project. This part will be revisited shortly before the deadline for a retrospective on what requirements were implemented and which ones were not.

2 The rules of the game

The rules of Tic-tac-toe are very simple, as described by **Wikipedia**:

> *Tic-tac-toe (or Noughts and crosses, Xs and Os) is a paper-and-pencil game for two players, X and O, who take turns marking the spaces in a 3x3 grid. The player who succeeds in placing three respective marks in a horizontal, vertical, or diagonal row wins the game.*

These simple rules should be easy to implement in Java, either for two human players or a single human player versus an Artificial "Intelligence", described in the next chapter.

3 The AI

Since the game should be playable against a computer opponent, we will have to implement some sort of Artificial "Intelligence". The quotation marks around *Intelligence* are to signify that it will not be an actual intelligence but rather a simple program which marks empty squares in the grid either at random or according to simple conditions.

For our AI, we are planning to have two difficulty-levels:

- Easy: The AI simply marks squares at random with no thought towards winning or preventing the opponent from winning. This level of difficulty would be good for children or others who are simply trying to learn the rules.
- Hard: The AI should actively try to get three in a row, or prevent the opponent from winning (i.e. if the opponent has marked two squares in a row, the AI should try to mark

the third). This level of difficulty should be considerably greater than the easier one, and the player should have to use strategies like *forking* to win. This "AI" should not be a perfect player, however.

4 Coding conventions

Since the project is very small in scope, we decided upon some very simple coding conventions which should ensure consistency in the code without getting in the way or being obtuse. The most important of these are:

- Pascal Casing for classes: Every word in the class name starts with a capital letter (ex: ExampleClass).
- Camel Casing for functions and variables: The first word in the variable or function name starts with a non-capital letter, with each subsequent word starting with a capital letter (ex: exampleFunction(), exampleVariable).
- Curly braces should open in the same line as the control sequence and close in their own line:

```
while(true){
   exampleVariable++;
}
```

5 Prototypes

The following figures show what the final look of the game should be. The first figure shows the title screen where the user is given the option of playing against a human player or a computer-controlled opponent. Figure 2 then shows the two difficulty levels available after deciding to play against a computer opponent. The last figure shows how the game itself should look during and after play. With a scoreboard to the side, a "New Game"-button at the bottom and a red line showing how the results of the game were derived (which three marks are in a horizontal, vertical or diagonal row), this User Interface should be sufficient to display all relevant information without being cluttered.

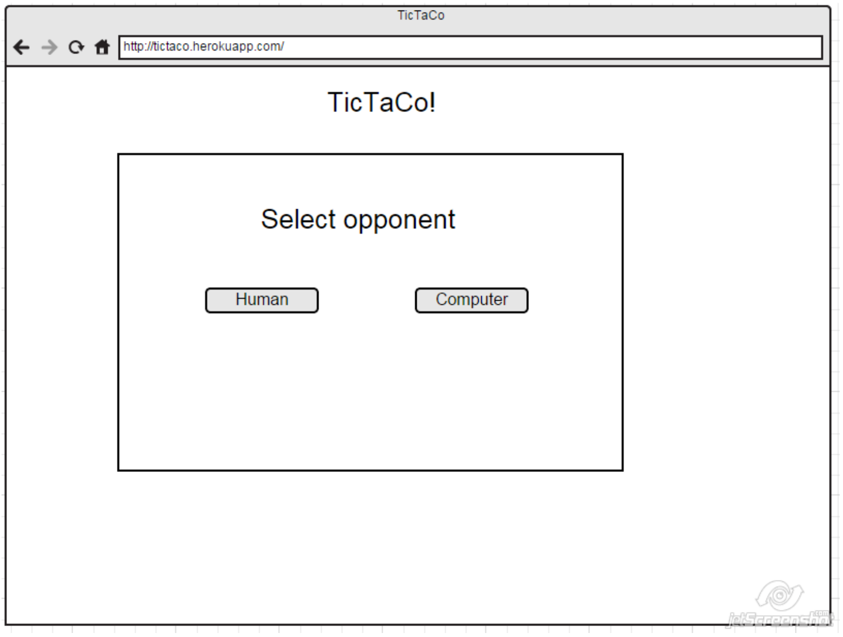

Figure 1: What the user sees when opening the page

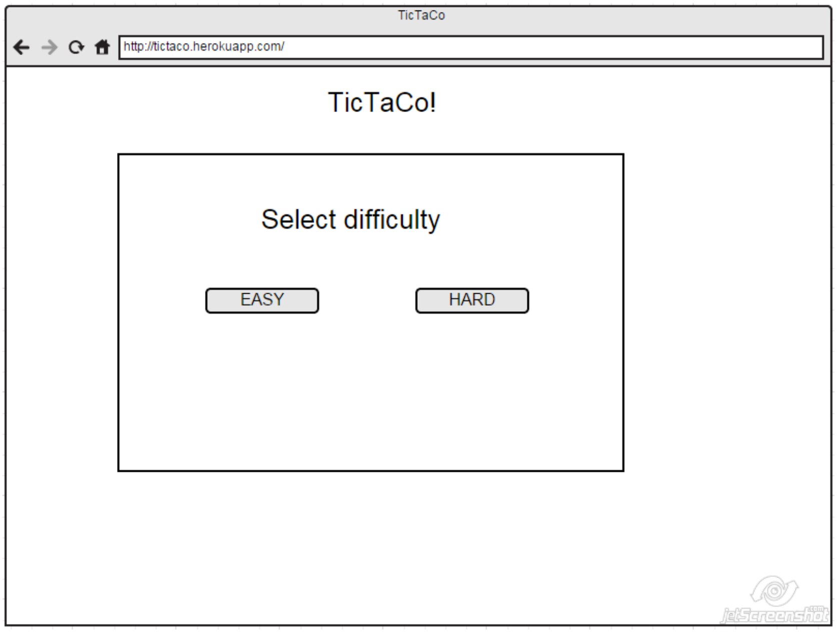

Figure 2: After choosing a computer opponent

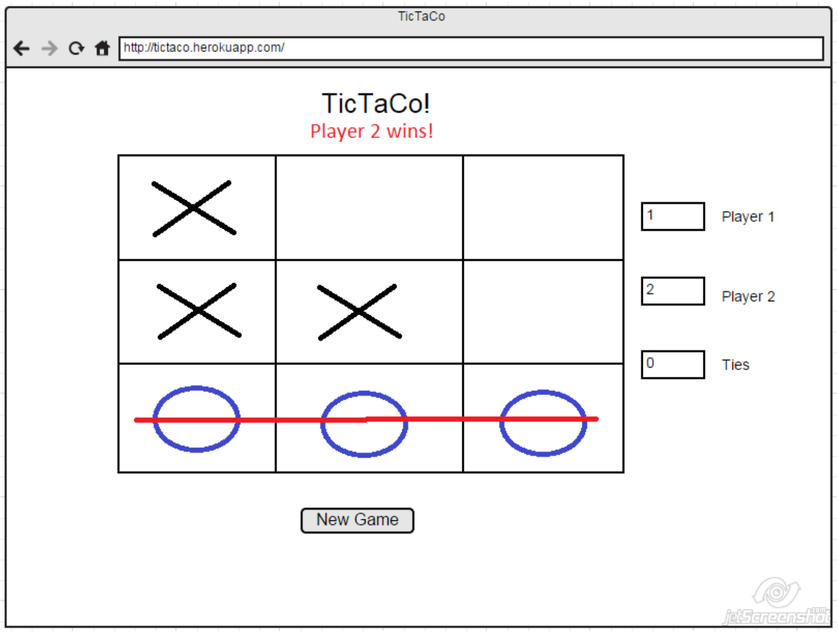

Figure 3: After a game

6 Requirements

Requirement	**Priority**	**Implemented?**
The user should be able to play the game Tic-tac-toe according to the normal rules	A	Yes
The user should be able to play against a computer-controlled opponent	A	Yes
The game should be hosted online	A	Yes
The user should be able to play against a human opponent	B	No
The user should be able to select the difficulty of the computer opponent	B	Yes
The game should display the current score; how many games each player has won and how many ties there have been	B	Yes

Table 1: Functional Requirements