CISP280 Project 1 Design Document

Matthew Weaver

# Section 1 – Project Overview

# Build an interactive tic-tac-toe game. The game should have an option for allowing the computer or user to go first. The utility function returns 3 values: 1 for win, -1 for lose, and 0 for tie. The game will have a a menu at the beginning to select player versus player or player versus computer as well as an option to know the rules. Finally the game will have an option to quit.

# Section 2 – High Level Entities/Objects

# This program will consist of four Java classes.

# Class Menus, this class provide all the methods that run the Main menu and all options of that menu. The menus options include choosing a two player game, a player versus computer game, rules information, about option and an exit option. This class will also run the selected game.

# Class AIplayer, this class provides all the variables and methods for the computer to play tic-tac-toe

# Class Humanplayers, this class provides all the variables and methods for two people to play tic-tac-toe

# Class Game, this class is the main class of the program and will run the Main menu from menus class.

# Section 3 – Low Level Design For Each Entity/Object

3.1 Class Menus

3.1.1 Usage

This class is used to display main menu and give the user the ability to choose options from that menu. Depending on the option chosen the menu will display or execute the given option.

3.1.2 Model

|  |
| --- |
| **Menus** |
|  |
| *+ Mainmenu (): void*  *+ Human (): void*  *+ AI (): void*  *+ Rules (): void*  *+ About (): void* |

3.1.3 Methods Descriptions

**Mainmenu**

This method calls the method the user chooses. Those methods include Human (), AI (), Rules () and About().

**Human**

This method calls all the methods necessary to play a human versus human tic-tac-toe game.

Those methods include names (), game pieces (), play order () and game ().

**AI**

This method calls all the methods necessary to play human versus computer tic-tac-toe game. Those methods include names (), game pieces (), play order () and game ().

**Rules**

This method tells the user the rules of the game. It has option to go back to the main menu.

**About**

This method tells the user why the game was created and also has a option to go back to main menu.

3.2 Class HumanPlayers

3.2.1 Usage

This class contains all the methods and variables to create and player versus player tic-tac-toe environment.

3.2.2 Model

|  |
| --- |
| **Humanplayers** |
| -Board: char Array [3] [3]  -player1: String  **-**player2: String  -CurrentPlayer: int  -marker1: char  -marker2: char  -plays: int  -br: bufferedreader |
| + Beginningboard (): void  + changeplayers (): void  + placemarker (int plays): Boolean  + winner (): Boolean  + createboard (): String  + names (): void  + gamepieces(): void  + playorder (): void  + game (): void  + getPlayer1(): String  + setplayer1 (string p1): void  + getPlayer2(): String  + setplayer2 (string p2): void  + getcurrentplayer (): int  + setcurrentplayer (int current): void  + getmarker1(): char  + setmarker1 (char piece1): void  + getmarker2(): char  + setmarker2 (char piece2): void  + getplays (): int  + setplays (int moves): void |

3.2.3 Method Descriptions

**BeginningBoard**

This method sets up the game board with digits one through nine using a two-dimensional for loop. It has a counter to the numbers one through nine on the board. Additionally it sets the current player to the current player and sets plays to zero.

**ChangePlayers**

This method switches the current player to player one to player two or vice versa it also increments the play amount.

**Placemarker**

this method places player one or player to marker on the board depending on who the current player is returns true if successful and false if not. Using a two-dimensional for loop and assigning digits one through nine on the board.

**Winner**

This method uses the standard tic-tac-toe rules to determine if a player has won a game. It does this by using a two-dimensional array and checking if there are three markers of the same player in a row, column or diagonally on the board.

**Createboard**

This method draws gameboard to the screen using a two-dimensional array the

formatting property is used after every three squares.

**Names**

This method gets both players names and set them to player one or player two.

**Gamepieces**

This method asks player one to select either X or O as their marker. As a result marker one is set to the marker player one enters. Player two will automatically get the other letter as their marker. If input is not x or o in either capital or lowercase form player one they will be asked to enter another value. After the markers have been selected the marker associated with each player will display on screen.

**Playorder**

This method determines who will go first. The player that has x for their marker will go first. The player who goes first will display on screen

**Game**

This method plays the tic-tac-toe game. It accomplishes this by declaring a Boolean playing and setting it to true. It also declares to strings one to store the current players name and the square they choose along with an integer to store the players pick. While playing equals true call the beginning gameboard and draw it to the screen. Declare a Boolean variable allowed and set it equal to false this variable will determine if users pick is valid. While there is not a winner and plays are less than nine and while loud equals true indicate whose turn it is and ask them to pick a square the users entry and stored in the string square convert the entry to a character of length one and then convert that character to an integer and store it in pick. Place marker at pick and set this equal to allowed. If entry is not allowed choose another number. Then change players and redraw the board with the space is updated. When a winner is found display winners name if no winner is found display draw. Asked the users if they want to play again if yes start the game over if no take them back to main menu.

**Getplayer1**

This method returns player one's name.

**Setplayer1**

This method sets player one's name to user entry.

**Getplayer2**

This method returns player two 's name

**Setplayer2**

This method sets player two 's name to user entry

**Getcurrentplayer**

This method returns the current player

**Setcurrentplayer**

This method sets player one or player two as the current player

**Getmarke1**

This method returns player one's marker

**Setmarker1**

This method sets marker1 to x or o in either capital or lowercase

**Getmarke2**

This method returns player two marker

**Setmarker2**

This method sets marker2 to x or o in either capital or lowercase

**Getplays**

This method returns the number of plays.

**Setplays**

This method keep track of the number of plays.

3 .3 Class AIplayer

3.3.1 Usage

This class contains all the methods and variables to create and player versus computer tic-tac-toe environment.

3.3.2 Model

|  |
| --- |
| **AIplayer** |
| -Board: char Array [3] [3]  -player1: String  **-**PC: String  -CurrentPlayer: int  -marker1: char  -PC marker: char  -plays: int  -br: bufferedreader |
| + Beginningboard (): void  + changeplayers (): void  + placemarker (int plays): Boolean  + winner (): Boolean  + createboard (): String  + names (): void  + gamepieces(): void  + playorder (): void  + humanchoice (string player, string square, int pick, Boolean allowed): Boolean  + randomstrategy (string player, string square, Boolean allowed): Boolean  + game (): void  + getPlayer1(): String  + setplayer1 (string p1): void  + getPC (): String  + setPC (string p2): void  + getcurrentplayer (): int  + setcurrentplayer (int current): void  + getmarker1(): char  + setmarker1 (char piece1): void  + getPCmarker (): char  + setPCmarker (char piece2): void  + getplays (): int  + setplays (int moves): void |

3.2.3 Method Descriptions

**BeginningBoard**

This method sets up the game board with digits one through nine using a two-dimensional for loop. It has a counter to the numbers one through nine on the board. Additionally it sets the current player to the current player and sets plays to zero.

**ChangePlayers**

This method switches the current player to player one to PC player or vice versa it also increments the play amount.

**Placemarker**

this method places player one or player to marker on the board depending on who the current player is returns true if successful and false if not. Using a two-dimensional for loop and assigning digits one through nine on the board.

**Winner**

This method uses the standard tic-tac-toe rules to determine if a player has won a game. It does this by using a two-dimensional array and checking if there are three markers of the same player in a row, column or diagonally on the board.

**Createboard**

This method draws gameboard to the screen using a two-dimensional array the

formatting property is used after every three squares.

**Names**

This method gets player one's name and set to player one. PC player gets set to computer

**Gamepieces**

This method asks player one to select either X or O as their marker. As a result marker one is set to the marker player one enters. PC player will automatically get the other letter as their marker. If input is not x or o in either capital or lowercase form player one they will be asked to enter another value. After the markers have been selected the marker associated with each player will display on screen.

**Playorder**

This method determines who will go first. The player that has x for their marker will go first. The player who goes first will display on screen

**Humanchoice**

This method handles the person choice in the tic-tac-toe game. It does this by through a while loop and the Boolean variable allowed while allowed equals true the user is asked to enter a value into the computer if that value is between one and nine it is placed in the appropriate position on the board if it is not they asked again to enter a valid input.

**Randomstrategy**

This method handles the computers choice in the tic-tac-toe game. It does this by generating random numbers from 1 to 9 and placing them on the board while the Boolean variable allowed equals true. If the number and generates is already taken it simply generates another.

**Game**

Thismethod plays the game. It does this using two wow loops one making sure a Boolean variable playing stays equal to true at the beginning of the loop draw the beginning board. The while loop makes sure there is no winner and number of plays is less than nine. Getcurrentplayer equals player one then runs the human choice method if current player equals PC that runs the random strategy method. In both cases after running the selected method it switches players and draws the current board. If a winner is found display winner on screen and ask user if they want to play again if yes start loop over if no return to main menu. If no winner is found display draw and ask the user the same question.

**Getplayer1**

This method returns player one's name.

**Setplayer1**

This method sets player one's name to user entry.

**Get PC**

This method returns PCs name

**SetPC**

This method sets PCs name to computer

**Getcurrentplayer**

This method returns the current player

**Setcurrentplayer**

This method sets player one or player two as the current player

**Getmarke1**

This method returns player one's marker

**Setmarker1**

This method sets marker1 to x or o in either capital or lowercase

**GetPCmarker**

This method returns PCs marker

**Set PCsmarker**

This method sets PCmarker to x or o in either capital or lowercase

**Getplays**

This method returns the number of plays.

**Setplays**

This method keep track of the number of plays.

3 .3 Class game (Main program)

3.3.1 Usage

This class contains a method call to call the main menu from the menu class. This is so the program will run.

# Section 4 – Assumptions (If none – say so)

The only assumption is that the person is running this program in a Java in the environment.