GRIFFITH COLLEGE

Course	BSCO / BSCH
Module	HCI-GUI
Issue Date	01/11/2016
Due Date	Please see Moodle for details.
Assignment Number	2
Assignment Title	UltimateXOs (Tic-Tac-Toe)
Weighting	15% of overall grade

Introduction

In this assignment you are required to produce a working version of UnltimateXOs. The GUI should follow the basic structure of Example20 from the lab book with modifications and aesthetic and functional improvements.

Instructions and Submission Guidelines

1. Please read the instructions below carefully.
2. There are many lines of code that you will need to add to files or subtract from copies of files.
3. Please be sure to incrementally build and test your code to feel like you are making progress.
4. Please use detailed comments where appropriate
5. Please use Example20 to assist in the rapid development of your solution.
6. You must upload your work via Moodle by the due date.
7. You are required to submit a single archive file to Moodle using the file name conventions detailed in the *Assessment, Submissions and Requirements* Lecture.
8. Penalties will be applied if:
    a. You do not submit by the due data
    b. Your submission fail to meet the specified guidelines. These are detailed in the lecture notes
9. The submission file should contain a zipped version of your project and this file with screenshots detailing the successful or unsuccessful completion of the various tasks.

Explanation

Your GUI will consist of **9 embedded** XOBoards as below. The cells of the boards have been colour coded below for your benefit. Where you can place an X or an O will be dictated by your opponent's previous move. The underlined yellow cell in the top left of the top left board could is considered to be the 0,0 cell in the 0,0 XOBoard of the XOUltimateBoard.

0,0,0,0	0,0,0,1	0,0,0,2	0,1,0,0	0,1,0,1	0,1,0,2	0,2,0,0	0,2,0,1	0,2,0,2
0,0,1,0	0,0,1,1	0,0,1,2	0,1,1,0	0,1,1,1	0,1,1,2	0,2,1,0	0,2,1,1	0,2,1,2
0,0,2,0	0,0,2,1	0,0,2,2	0,1,2,0	0,1,2,1	0,1,2,2	0,2,2,0	0,2,2,1	0,2,2,2
1,0,0,0	1,0,0,1	1,0,0,2	1,1,0,0	1,1,0,1	1,1,0,2	1,2,0,0	1,2,0,1	1,2,0,2
1,0,1,0	1,0,1,1	1,0,1,2	1,1,1,0	1,1,1,1	1,1,1,2	1,2,1,0	1,2,1,1	1,2,1,2
1,0,2,0	1,0,2,1	1,0,2,2	1,1,2,0	1,1,2,1	1,1,2,2	1,2,2,0	1,2,2,1	1,2,2,2
2,0,0,0	2,0,0,1	2,0,0,2	2,1,0,0	2,1,0,1	2,1,0,2	2,2,0,0	2,2,0,1	2,2,0,2
2,0,1,0	2,0,1,1	2,0,1,2	2,1,1,0	2,1,1,1	2,1,1,2	2,2,1,0	2,2,1,1	2,2,1,2
2,0,2,0	2,0,2,1	2,0,2,2	2,1,2,0	2,1,2,1	2,1,2,2	2,2,2,0	2,2,2,1	2,2,2,2

Example

- Player 1 starts the game by placing an X in the 0,0 cell of board 8
- Player 2 can only place an O in a free cell of board 0 because of where player 1 placed his X.

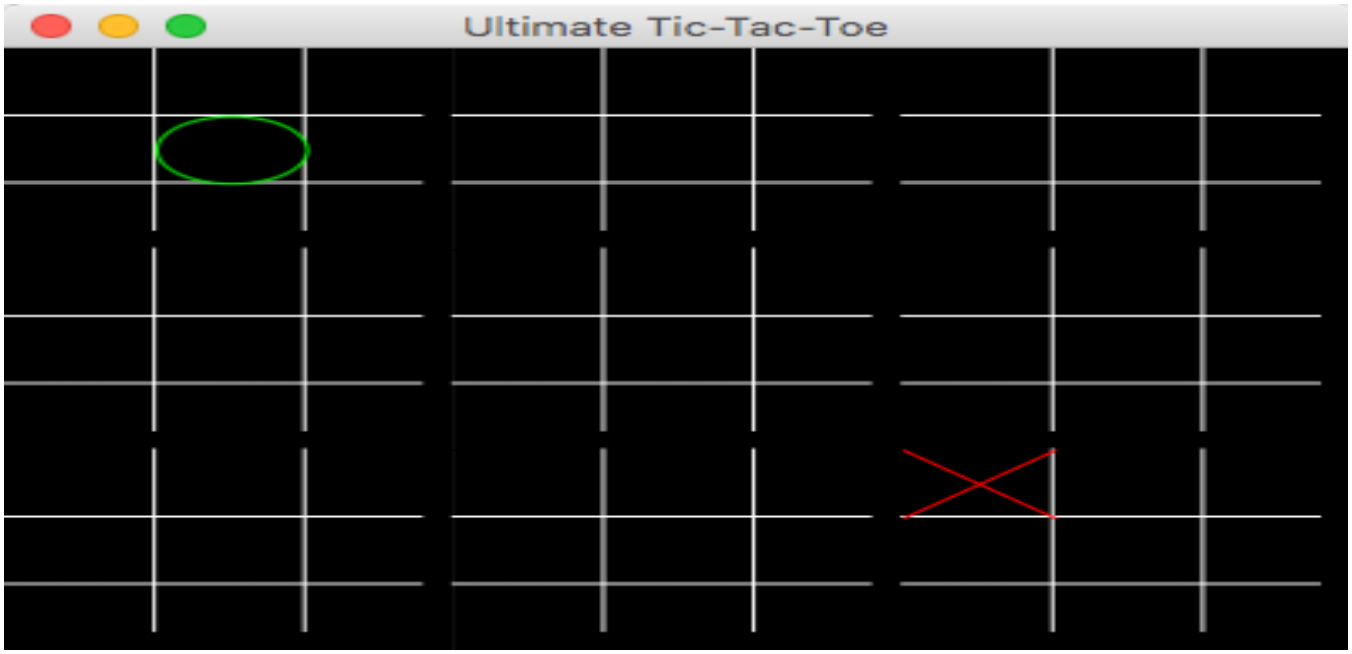
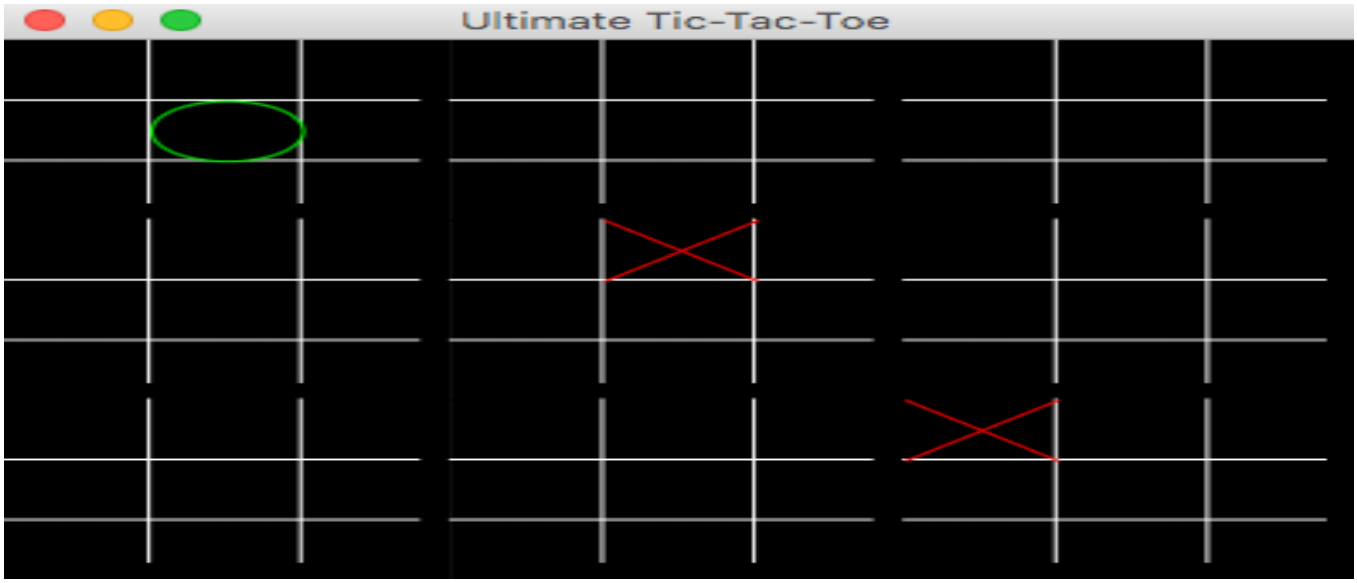


Figure 1 – GUI for UltimateXOs

- Player 1 can now only place an X in a free cell in the central board (because player 2 placed an O in the central position of board 1)



- Play continues in this fashion.
- If a prescribed board is full then a player may choose any available cell on any available board. Following this move the game reverts to the flow above.
- The game finishes when all cells are populated with an X or an O.
- The player who wins the most boards wins the game.

Online Resources

1. Rules: https://www.youtube.com/watch?v=37PC0bGMiTI
2. A winning strategy: https://www.youtube.com/watch?v=weC1pAeh2Do
3. Game Logic: https://mathwithbaddrawings.com/2013/06/16/ultimate-tic-tac-toe/
4. Online Edition: http://ultimatetictactoe.creativitygames.net

Steps to turn Example20 into basic XOUltimateXos

1. Make a completed copy of the Example20 project and rename it to the appropriate submission name as per 1st lecture of the year and make the following edits to the files within it.
2. **Example20.java**
    a. Rename to UltimateXOs.java
    b. Change the title of the primaryStage to something appropriate
3. **XOUltimate.java**
    a. Make this file and copy the contents of XOBoard.java into it.
    b. Delete all references and data related to
        i. Rectangle back
        ii. Line h1,h2,v1,v2
        iii. Translate ch_one, ch_two, cw_one, cw_two
    c. XOUltimate()- Ensure that XOBoards are added to XOUltimate. Ensure that all references mentioned above have been deleted.
    d. reset() – Initalize XOBoards and add them
    e. placePiece() – call placePiece() of the correct XOBoard with localized height and width values as arguments.
    f. getCurrent_player() – add this method
    g. setCurrent_player() – add this method
4. **XOBoard.java**
    a. Delete current_player field.
    b. XOBoard() - accept a reference to an XOUltimate object and store it as private member data.
    c. resize() – ensure the horizontal and vertical lines do not go full width or height
    d. placePiece()- get and set current player in XOUltimate private object using accessor methods written in XOUltimate.java.
5. **CustomControl.java**
    a. Change all references from XOBoard to XOUltimateBoard.
6. **CustomControlSkin.java**
    a. Do not change
7. **GameLogic.java**
    a. This will contain your logic about how to calculate a winner. It will principally focus on the board array in XOUltimateBoard and XOBoard. It will have methods which will be called principally from XOBoard/XOUltimateBoard placePiece(). Some updating of other classes may be necessary.

Successful completion of the steps above will result in the completion of tasks 1 and 2 below.

User Tasks and Marks		
Task No.	**Description**	**Marks**
1	The application should load and present a GUI to the end user similar to that of Figure 1. **Hint:** Follow steps above.	20
2	It should be possible to place an X or O anywhere on the board in turn. **Hint:** Follow steps above.	20
3	Detection of the winner of a single board is determined and displayed. **Hint:** If a move was successfully completed examine the relevant *XOBoard* to see if a winning state is present. If so notify the user and update the *board* array of the *XOUltimateBoard*. Majority of code to appear in *GameLogic*.	15
4	Detection of overall winner is determined and displayed. **Hint:** If a move was successfully completed and the relivent *XOBoard* contains a winning state update the *board* array of the *XOUltimateBoard* and see if it contains a winning state and if so notify the user and end the game. Majority of code to appear in *GameLogic*.	15
5	User is restricted from placing an X or an O in an invalid cell (based on opponents previous move) ***Hint***: Review online resources to understand this requirement. In the *XOBoard* create a *boolean* field called *active* to track if that board can accept a piece or not. In *XOUltimateBoard* create a field to track the most recent move which will be used by *GameLogic* to determine the next prescribed board. The rest of the logic is up to you.	15
6	Design Improvements, any design improvement which is sensible and improves the aesthetics and or functionality of the program. This must be clearly documented.	15

N.B. Be sure to comment what is working and not working for each of the tasks. The boxes should be expanded to contain the content.

Screen Shots – Task 1 (1 image + what is working/not working)

Screen Shots – Task 2 (2 images + what is working/not working)

Screen Shots – Task 3 (1 image + what is working/not working)

Screen Shots – Task 4 (1 image + what is working/not working)

Screen Shots – Task 5 (1 image + what is working/not working)

Screen Shots – Task 6 (2 images and short description + what is working/not working)

Edited by Alex Cronin 2016.10.31 @ 22:15. Version 1