SYST10199 Web Programming

Notes May 5, 2019

Problem: design a tic tac toe game

NOTE: design does not include implementation (sometimes, you do “proof of concept”)

Description

Tic tac toe is a two-player game where the players take turns to select an available cell in a three-by-three grid with the aim to win the game by placing their marker in a row, a column, or a diagonal.

Algorithm: first pass

1. Start game
2. x marks an empty cell // turn 1, 8 cells left
3. check if winner // no need to check [x]
4. x passes marker to o
5. o marks an empty cell // turn 2, 7 cells left
6. check if winner // no need to check [xo]
7. o passes marker to x
8. x marks an empty cell // turn 3, 6 cells left
9. check if winner // no need to check [xox]
10. x passes marker to o
11. o marks an empty cell // turn 4, 5 cells left
12. check if winner // no need to check [xoxo]
13. o passes marker to x
14. x marks an empty cell // turn 5, 4 cells left
15. check if winner // three x’s can be in a row, col, or diag [xoxox];
    // if yes, declare winner and end game.
16. x passes marker to o // if no, continue
17. o marks an empty cell // turn 6, 3 cells left
18. check if winner // if yes, declare winner and end game.
19. o passes marker to x // if no, continue
20. x marks an empty cell // turn 7, 2 cells left
21. check if winner // if yes, declare winner and end game.
22. x passes marker to o // if no, continue
23. o marks an empty cell // turn 8, 1 cell left
24. check if winner // if yes, declare winner and end game.
25. o passes marker to x // if no, continue

26. x marks an empty cell // turn 9, 0 cells left
27. check if winner // if yes, declare winner and end game.
    // if no winner declare a TIE and end game.
28. end game // start new game?

Notes

- if we replace, x and o with current player, three steps repeat
- **avoid references to implementation components, such as array, method, data types, etc. at this stage. You are writing for a human being's readability, not the computer.**

Algorithm: second pass

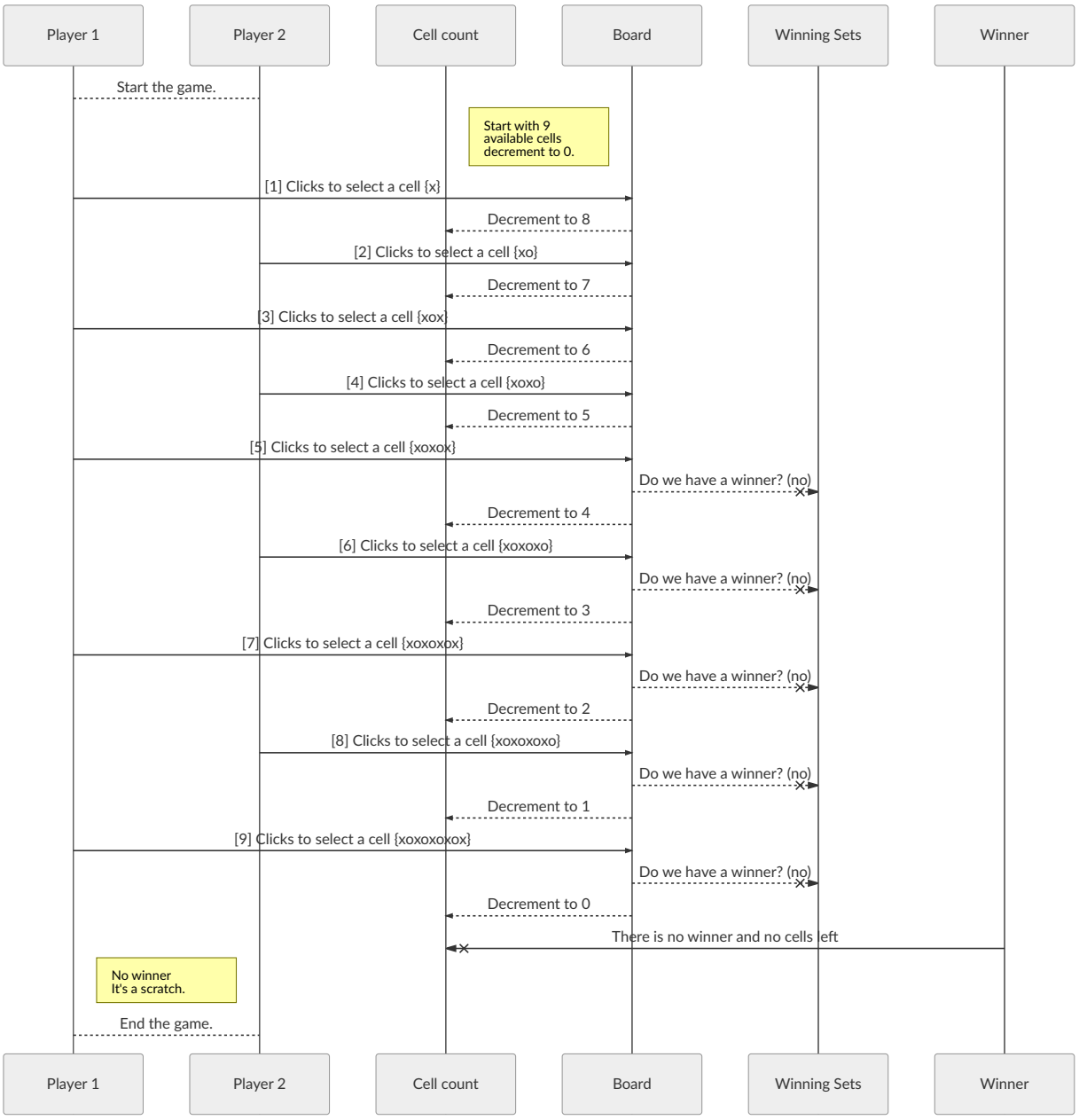
1. Start game
2. current player marks an empty cell // keep an eye on cell availability
3. we check if we have a winner // TODO: add condition, start checking when turn > 4
4. if yes, declare a winner and end the game.
5. if no empty cells left and no winner, declare a tie and end the game.
6. flip to other player, return to 2 // NOTE: game must end in step 4 or 5 only!!! we go to this step only if the game is still on

Algorithm: third pass

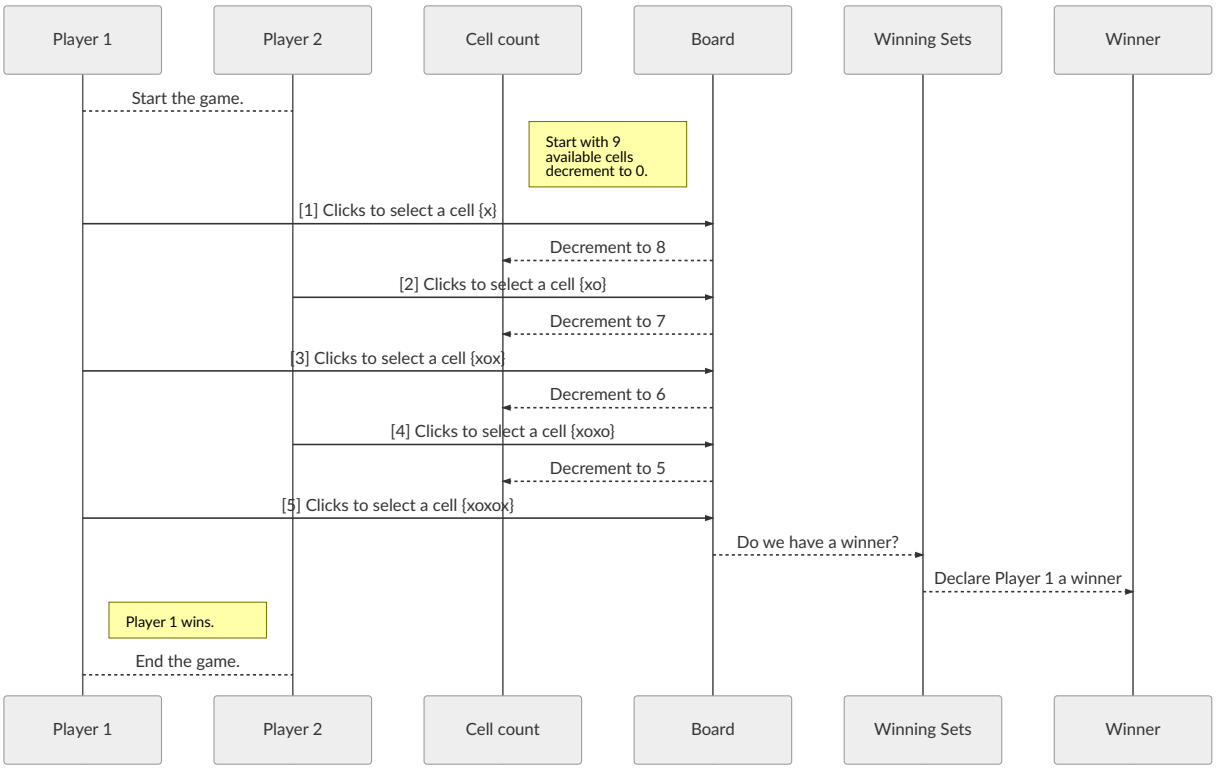
- manually run through the game, adding details as they emerge
- a common approach is to write test cases; take each test case and walk through your code; think of a typical case and boundary cases (ie. x wins with a column, x wins with a row, x wins with a diagonal, o wins with ..., there is a tie,...).
- During every walk through, write down your observations. After the walk through add to or modify your algorithm. Keep in mind that if you make changes to your algorithm, you need to retest all test cases.
- Ideally, test all possibilities - which might not be feasible, later learn to use tools that automate testing.

Algorithm: sequence for a scratch

Once you have written out your algorithm and tested your logic and completeness by walking through cases, you can use one of many formal representations, such as a sequence diagram. For example, here is a sequence diagram for a scratch (draw, tie):



Sequence diagram: example of X winning on turn 5



Testing and Documentation

Use your Technical Communication (Lannon, Klepp, Kelly) **textbook** from The Art of Technical Communication course to help you improve your problem solving and documentation skills. For example, **Chapter 19** describes, in detail, how to develop analytical reports. Those details encompass the process of solving problems, what questions to ask, what to do with those details, etc.