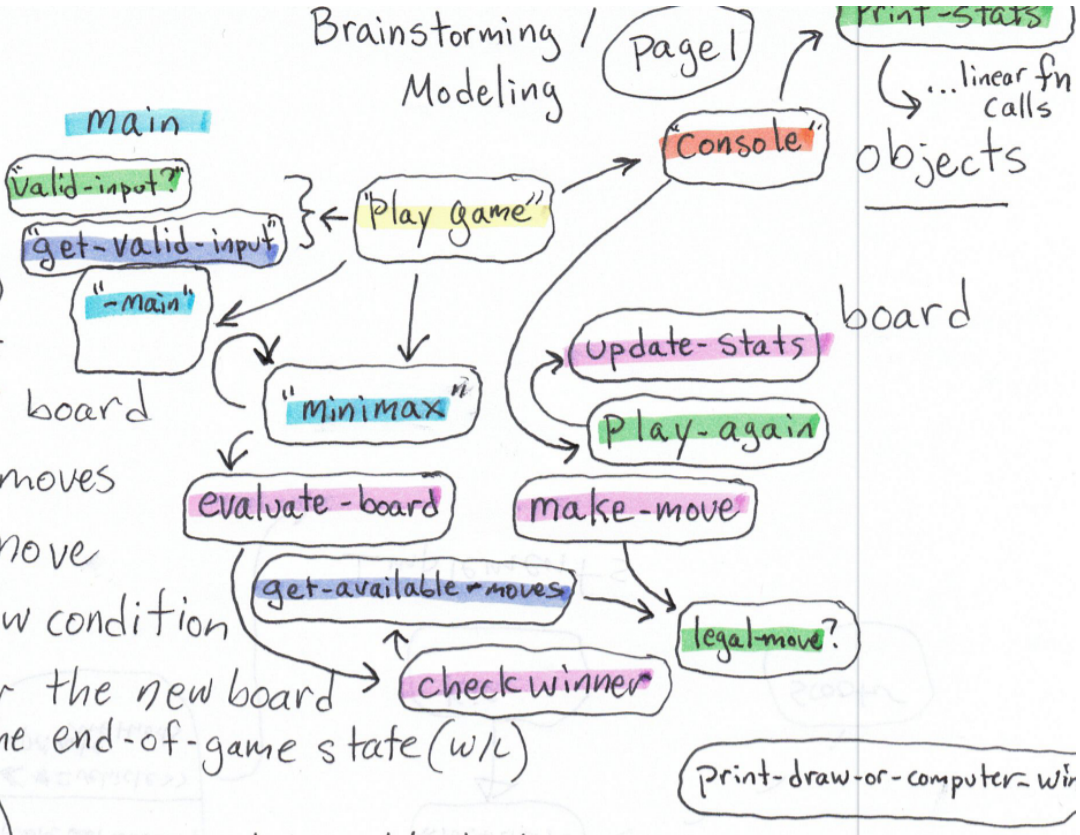
Tic Tac Toe

Brainstorming / Modeling



use cases

print heading
take user input
print current board
Get available moves
Determine best move
Make the move
Evaluate the new condition
Return either the new board state or the end-of-game state (w/c)

actors

~~agents~~

player

cpu]
not
*really
though.

Appropriate tests

data attributes

title
label
draw
score [-1, 0, 1]
win/lose - victory?

records

none
game histor

Right now it doesn't prevent you from playing over your own spot also

Project Description

Lorelai Lyons
11/8/2024

Goal: Make a tic-tac-toe game.

MVPs: The game should run as a console application in a loop until a player is determined a winner, and then exit.

- ↳ It will take a users name
- ↳ Print an updated board each turn
- ↳ Utilize a minimax algorithm to determine the best possible move.
- ↳ "Take that move" and return an updated board.
- ↳ Determine a winner, and exit.
- ↳ Incorporate testing where appropriate, use TDD whenever possible.

Stretch Goals

Use a modern web library to enhance the interactivity of the application

Keep a record of the players wins and losses in local storage

- ↳ optionally include a different stora option or options
- ↳ keep other statisti
  - favorite starting move?
  - avg. num turns

Implement Multiplayer

This implementation of tic-tac-toe is intended to be "clean", and relatively transparant in regards to its inner workings.

The game will be a simple loop incorporating some locally defined variables to keep track of any 'state'.

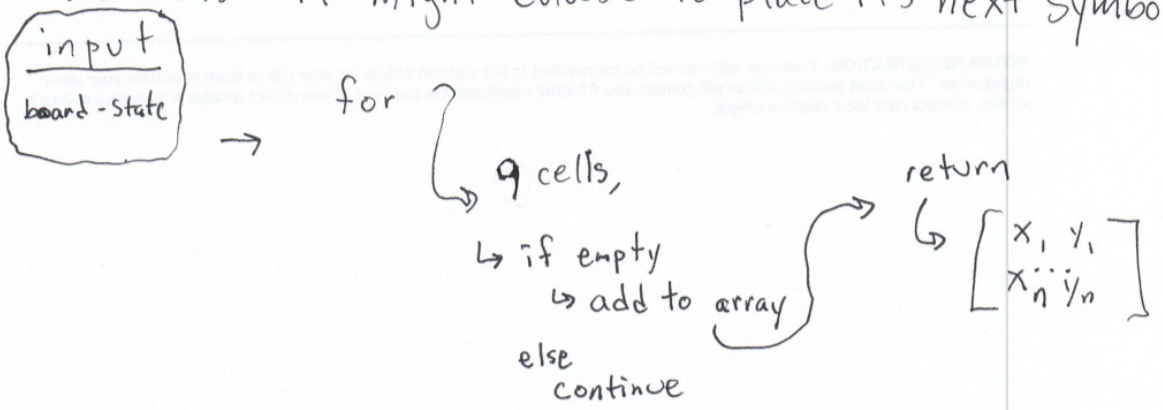
I will demonstrate a simple implementation of this game as a browser application using Helix, which is a modern clojurescript wrapper around React.

(As a stretch Goal)

Page 3

Get_available_moves

This function uses clojure for loops to iterate through the 9 board cells and determine if it is blank using the ~~core~~ clojure.string blank? function, which checks if a string is empty. This function adds all the empty spots into an array of valid "locations" it might choose to place its next symbol.



~~Evaluate_available_moves~~ victory?

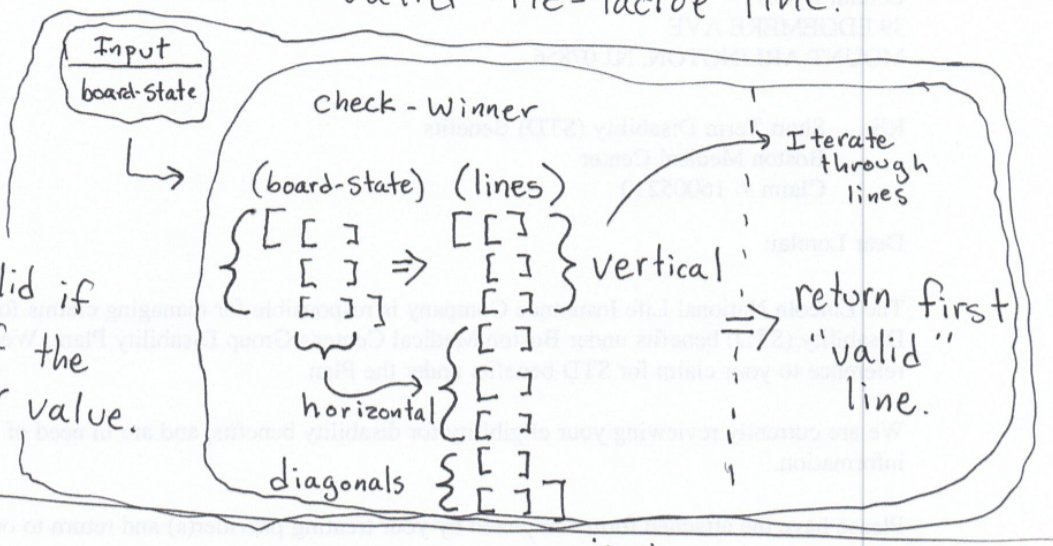
this function takes the board state, and calls another function, check_victory, along with the input value, ~~maybe this is useless...~~

if that function determines a line exist that satisfies a victory condition, this will return that result as a 1 in favor of the player, or a -1 if in favor of the computer.

otherwise it is not a victory yet, and a 0 is passed, indicating we need another turn.

page 4

Check winner constructs the ~~axis~~ upon which the game is being judged. Some is used to return the first instance of a valid tic-tactoe line.



Valid: A line is valid if it contains 3 of the same character value.

Make Move

input
cell/coordinate [x,y]

[[0,0 0,1 0,2]
[1,0 1,1 1,2] =>
[2,0 2,1 2,2]]

if (x, y) = a cell without " "
empty
return INVALI

The board state is a vector of 3 vectors.

↳ Each inner vector has 3 characters in them

↳ The core clojure function blank? can check if these are valid positions by assessing ~~if any of these values have~~ which values are "empty",

↳ If it is a valid spot being compared then the board state is updated accordingly

↳ Else it will return that the choice is invalid, and then return 'control' to the main loop where the player will be prompted for another choic

Main Loop

↳ Print the heading and
Prompt the user for their name

~~↳ Print the board~~

↳ Enter the main loop

↳ print the current board

↳ Has anyone won?

↳ if yes, declare it and end

EXIT

↳ else

(player) ↳ Where would you like to play?

update board

(computer) ↳ Computer's turn

↳ recur until there is a clear victor