Tic-Tac-Toe Game Design Document

Note: This design document will primarily focus more on the GUI aspects of design rather than other aspects that have already been implemented in the previous project, such as AI.

Section 1: Purpose:

The purpose of this project is to work and improve on our version of a 3-dimensional 4x4x4 game of tic-tac-toe and thoroughly test its functionality as a JavaFX application equipped with a user-friendly graphical-user-interface. Our team of developers has been working endlessly to ensure the user gains the best experience possible when playing this game. While this project is very closely related to the previous, they do differ in their goals. The goal of the previous project was to develop a working game and AI that could effectively and efficiently keep the game moving. This project, however, is focused on developing a convenient and easy-to-use visually-interactive environment in which to play said game. As a group, we were able to develop, test, and deliver, a working product while managing a tight deadline and quickly gaining the skills necessary to be successful. The goal was to manage many different moving parts of project development. Not only was this good practice in program development and keeping in mind the concept of computer human interaction, it is now also a fun game that anyone can run and enjoy on a JVM, and whether they can open a terminal or not, the intent is to provide players with a sense of pride and accomplishment for executing and playing a fun game of tic-tac-toe.

Section 2: Definition of Higher Level Entities:

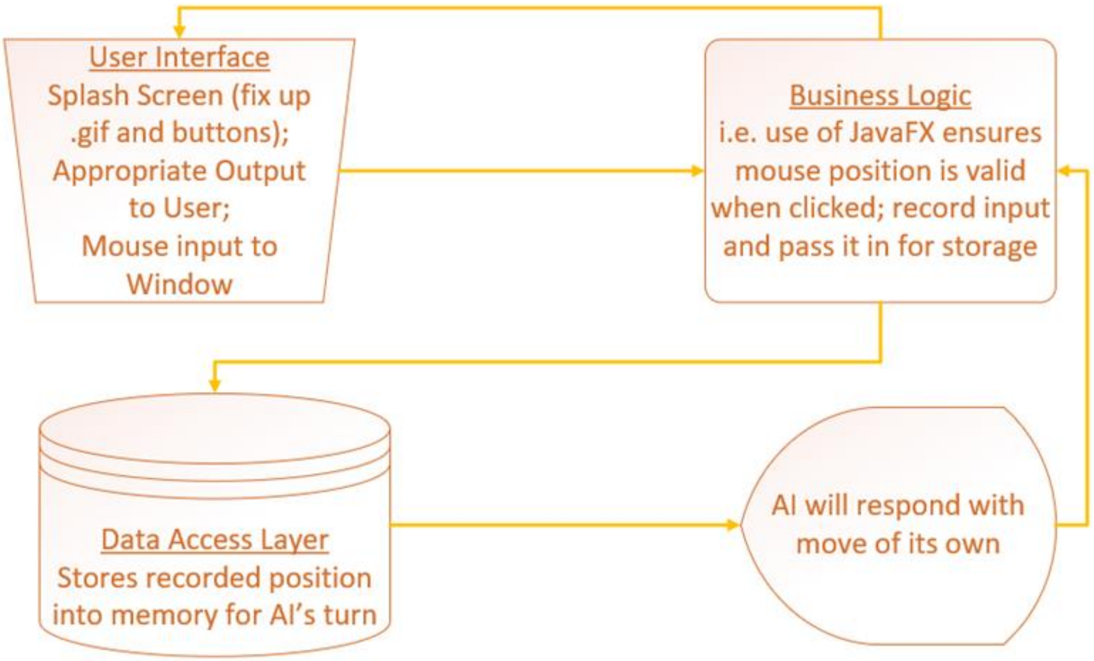
1) GUI:

- The application starts off in a windowed screen and has a user interface at the start of the game that first displays a splash screen that contains the title of the game, our team number, team names, and a continue button.
- The program is able to read from a stored file for the high scores and also increment the scores whenever the player wins a game.
- After getting past the splash screen, the user is shown simple instructions on how to play the game, and can click the continue button to transition to the game screen.

- From there after pressing the continue button, the game screen is shown, which consists of a high-score table in the top left corner of the screen, an instructions table in the top right part of the screen, an exit button at the very top-right edge of the screen, and finally, our game table, which contains our game board buttons (64).

2) Back-End:

- The back-end, as with the last project, is responsible for handling all stored data, the game board, and all the functions used to handle game logic and flow. All logic needed for data management will be kept internally to be able to communicate with JavaFX API what to do with the GUI in response.
- Most of the project 3 backend code was designed with portability in mind so that C++ -> Java conversion would be simple.
- This document will not extensively cover this, as nothing of it has changed since project 3, except for some tweaks when passing the information into the JavaFX code instead of the ncurses graphics library we used last time.


User Interface
Splash Screen (fix up .gif and buttons); Appropriate Output to User; Mouse input to Window
Business Logic
i.e. use of JavaFX ensures mouse position is valid when clicked; record input and pass it in for storage
Data Access Layer
Stores recorded position into memory for AI's turn
AI will respond with move of its own

3) Game-Play:

- The actual game play consists of the user attempting to beat the AI.
- We were able to improve the user experience when playing this game; the user now does not have to use the arrow keys to make a move but will instead make a move by navigating with a mouse and clicking on a cell.
- Use of the game board by clicking its buttons to place a move is extremely straightforward and intuitive, as a human's natural reaction when they see a button is to click it, so of course when they see 64 buttons appear on screen they will want to click them.
- All buttons that already contain either an X or an O piece are disabled and visually faded out. This lets the player know visually that they cannot move to that spot as the spot is already occupied, this to, is a more straightforward approach than displaying a "invalid move, try again" message to the screen everytime for the player to read.
- The scores are displayed on the same game screen so the user can be reminded of his or her competition, if any.

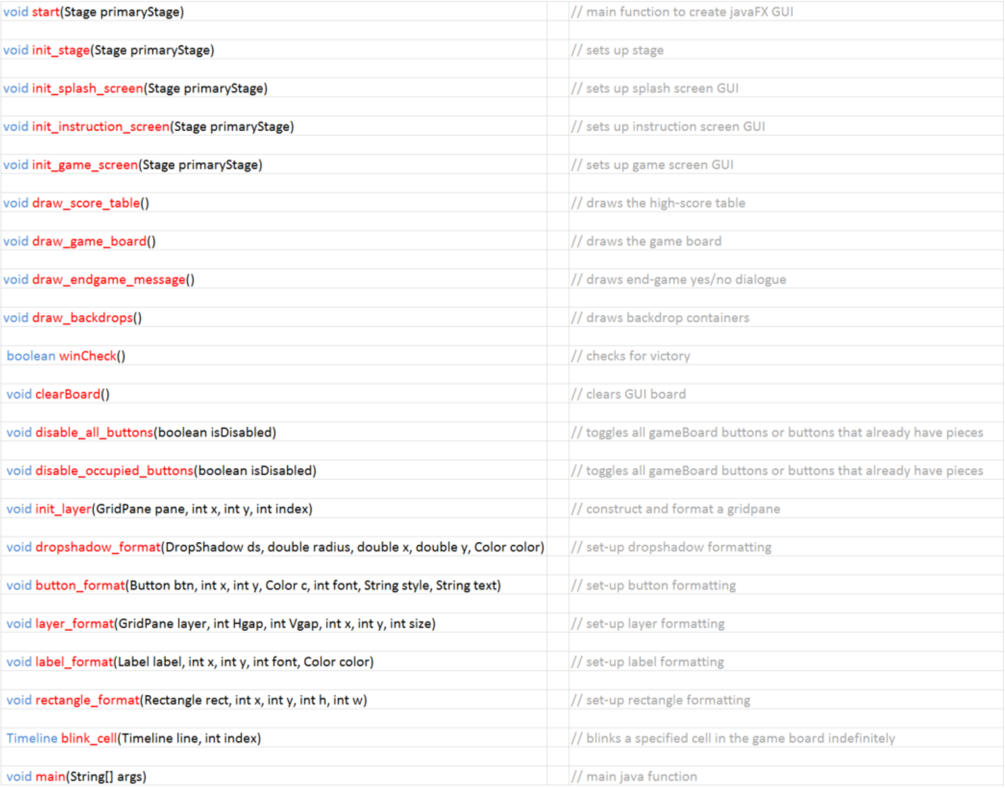
Section 3: Low Level Design:

1) GUI:

- **Usage:**
    - The user will have access to a graphical user interface and be able to navigate and interact using the mouse. JavaFX is the graphics library that is going to help handle the user’s input and, in turn, give output to the user through the GUI. This will allow us to depart from the text-based user interface from last project and get closer towards a standalone full-fledged application with a clean GUI.
    - JavaFX features a class called Button, which is how most of the transitional and input/output functionality of our program is structured. Inside of Button is a method called setOnAction() which can be called, and using a simple lambda expression, we can use it to do anything we want when the action is performed on the button; the particular action we are interested in is a mouse click.
    - The splash screen has a single button that displays the text to the user “Click to Continue”, and this leads to the use of the JavaFX classes Stage and Scene.

- The Stage is the actual window, and the Scene is what is inside the window. We created separate scenes for the splash screen, instruction screen, and game screen.

- **Configuration:**
  - The user will be required to keep the window centered at their screen at a constant width of 800 pixels and a constant height of 600 pixels. No other special configuration is needed for the rest of the game.

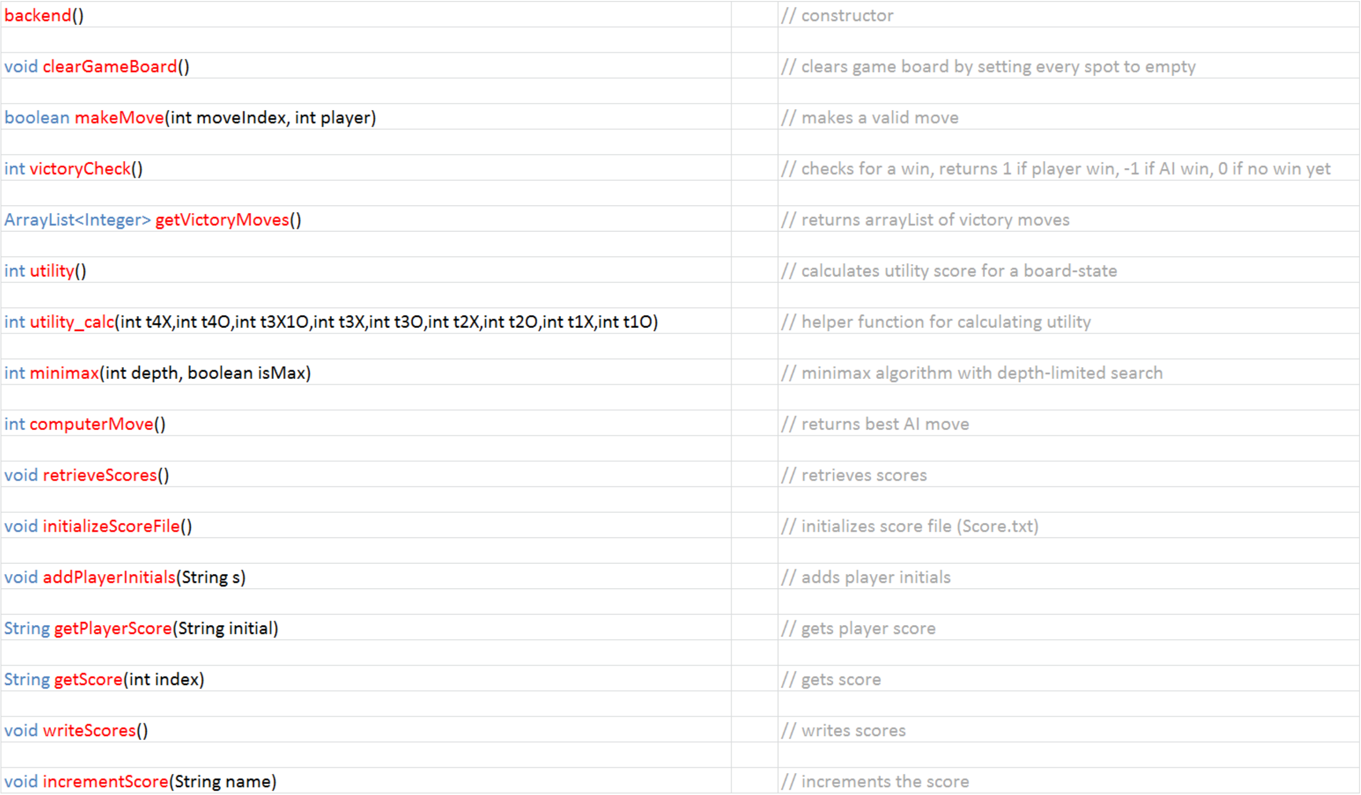


void start(Stage primaryStage)	// main function to create javaFX GUI
void init_stage(Stage primaryStage)	// sets up stage
void init_splash_screen(Stage primaryStage)	// sets up splash screen GUI
void init_instruction_screen(Stage primaryStage)	// sets up instruction screen GUI
void init_game_screen(Stage primaryStage)	// sets up game screen GUI
void draw_score_table()	// draws the high-score table
void draw_game_board()	// draws the game board
void draw_endgame_message()	// draws end-game yes/no dialogue
void draw_backdrops()	// draws backdrop containers
boolean winCheck()	// checks for victory
void clearBoard()	// clears GUI board
void disable_all_buttons(boolean isDisabled)	// toggles all gameBoard buttons or buttons that already have pieces
void disable_occupied_buttons(boolean isDisabled)	// toggles all gameBoard buttons or buttons that already have pieces
void init_layer(GridPane pane, int x, int y, int index)	// construct and format a gridpane
void dropshadow_format(DropShadow ds, double radius, double x, double y, Color color)	// set-up dropshadow formatting
void button_format(Button btn, int x, int y, Color c, int font, String style, String text)	// set-up button formatting
void layer_format(GridPane layer, int Hgap, int Vgap, int x, int y, int size)	// set-up layer formatting
void label_format(Label label, int x, int y, int font, Color color)	// set-up label formatting
void rectangle_format(Rectangle rect, int x, int y, int h, int w)	// set-up rectangle formatting
Timeline blink_cell(Timeline line, int index)	// blinks a specified cell in the game board indefinitely
void main(String[] args)	// main java function

2) Back-End:

- **Usage:**
  - All input will be passed into the data layer to be stored in the game's history.
  - All of the computer's decisions will be made well within ten seconds using a minimax algorithm and the utility function of a typical zero-sum game.
  - The game board, as with the last ncurses project, continues to also be stored internally as a one dimensional array of length 64, where 0, 1 and -1 correspond to an empty space, X piece (player), and O piece (AI), respectively.

- **Configuration:**
  - The game play will not need any special configuration, other than the fact that the intermediate logic will be to ensure the user has made a valid move and has not attempted an illegal command.



backend()	// constructor
void clearGameBoard()	// clears game board by setting every spot to empty
boolean makeMove(int moveIndex, int player)	// makes a valid move
int victoryCheck()	// checks for a win, returns 1 if player win, -1 if AI win, 0 if no win yet
ArrayList<Integer> getVictoryMoves()	// returns arrayList of victory moves
int utility()	// calculates utility score for a board-state
int utility_calc(int t4X,int t4O,int t3X1O,int t3X,int t3O,int t2X,int t2O,int t1X,int t1O)	// helper function for calculating utility
int minimax(int depth, boolean isMax)	// minimax algorithm with depth-limited search
int computerMove()	// returns best AI move
void retrieveScores()	// retrieves scores
void initializeScoreFile()	// initializes score file (Score.txt)
void addPlayerInitials(String s)	// adds player initials
String getPlayerScore(String initial)	// gets player score
String getScore(int index)	// gets score
void writeScores()	// writes scores
void incrementScore(String name)	// increments the score

3) Game-Play:

- **Usage:**
  - Each button, in principle, has its own way of calling the method that handles the press of a button. The can be viewed of as an “interrupt” and “handling of an interrupt” from an operating system standpoint.
  - Luckily, Java easily incorporates multi-threading and does not require too much complication.

Section 4: Benefits, Assumptions, Risks/Issues:

1) Benefits:

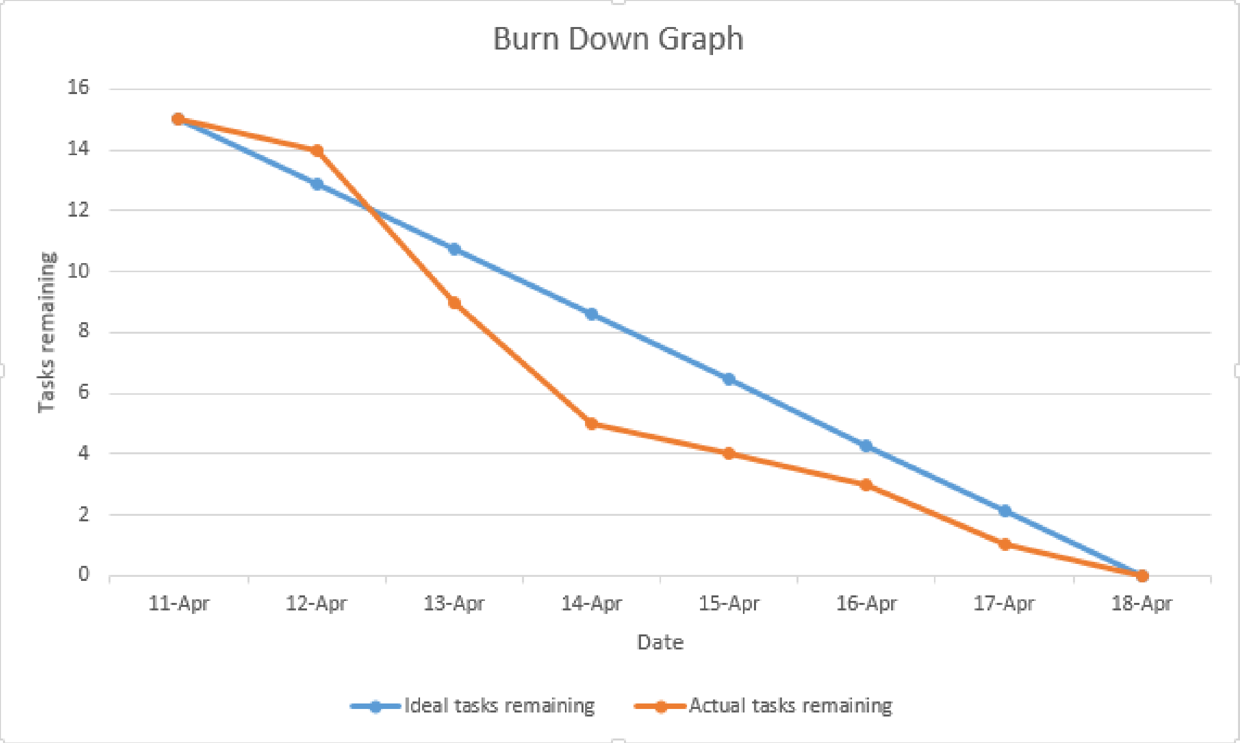
- By utilizing the JavaFX libraries instead of the ncurses library in c++, this gives us a quick development process that is more streamlined and easier for the user to view and comprehend.
- By imposing strict limits on the users input, checking if the desired position is already occupied by a move and preventing out of range choices, we limit the chance for bugs.
- By choosing a flat rate depth to stop the AI from searching on the min max tree, we can guarantee a relatively consistent run time for the AI min max tree creation/traversal whilst still maintaining extreme foresight.
- By limiting class interactions/data mingling (classes request the information they need from other classes only if that request passes the requested classes check) helps prevent data corruption among the many shared data management methods.

2) Assumptions:

- The user has the ability to run/have access to the JavaFX library and will have the necessary JRE downloaded.
- The high_scores file to be read/written will be in a plain text (.txt) file and newline delimited.
- User input for the initials will be as expected. User will enter 3 characters A-Z for their initials.
- High_scores file will be in the correct directory path.
- User will not modify the high_scores file on their local machine outside of the game.

3) Risks/Issues:

- Memory issues are the largest concern, since hard limiting the AI to a certain depth based on our test with the TAMU compute servers, we are invariably allowing for issues to occur when other users attempt to run our program on more devices that don't share the same level of available memory (ex: raspberry pi).
- There are a few known combination of moves that can currently beat the AI, such as if you get into a situation where you have two possible winning lines, the AI will have to pick to block one of these and leave the other open for you to win. For this project we only emphasized effort on the GUI portion of the project as required by rubric, the AI is not the primary focus in this case.



Ncurses Evaulations:

For the Ncurses evaluation we used a basic numeric scale shown in lecture to objectify qualitative feedback regarding the GUI's navigation, overall ease of understanding, and play-ability. A high score of 5/5 denotes "of high quality or ease" whereas a low score of 1/5 denotes "of poor quality or difficulty."

Peer Review #1:

- Ncurses GUI made it easy to navigate the game screen: ***4/5***
- Ncurses GUI made it easy to understand the instructions and play the game: ***5/5***
- Ncurses GUI overall aesthetic: ***4/5***
- Problems overall:
    - It was confusing to see where the other player had moved and I kept losing track of my own moves as well as his (the AI's).

Peer Review #2:

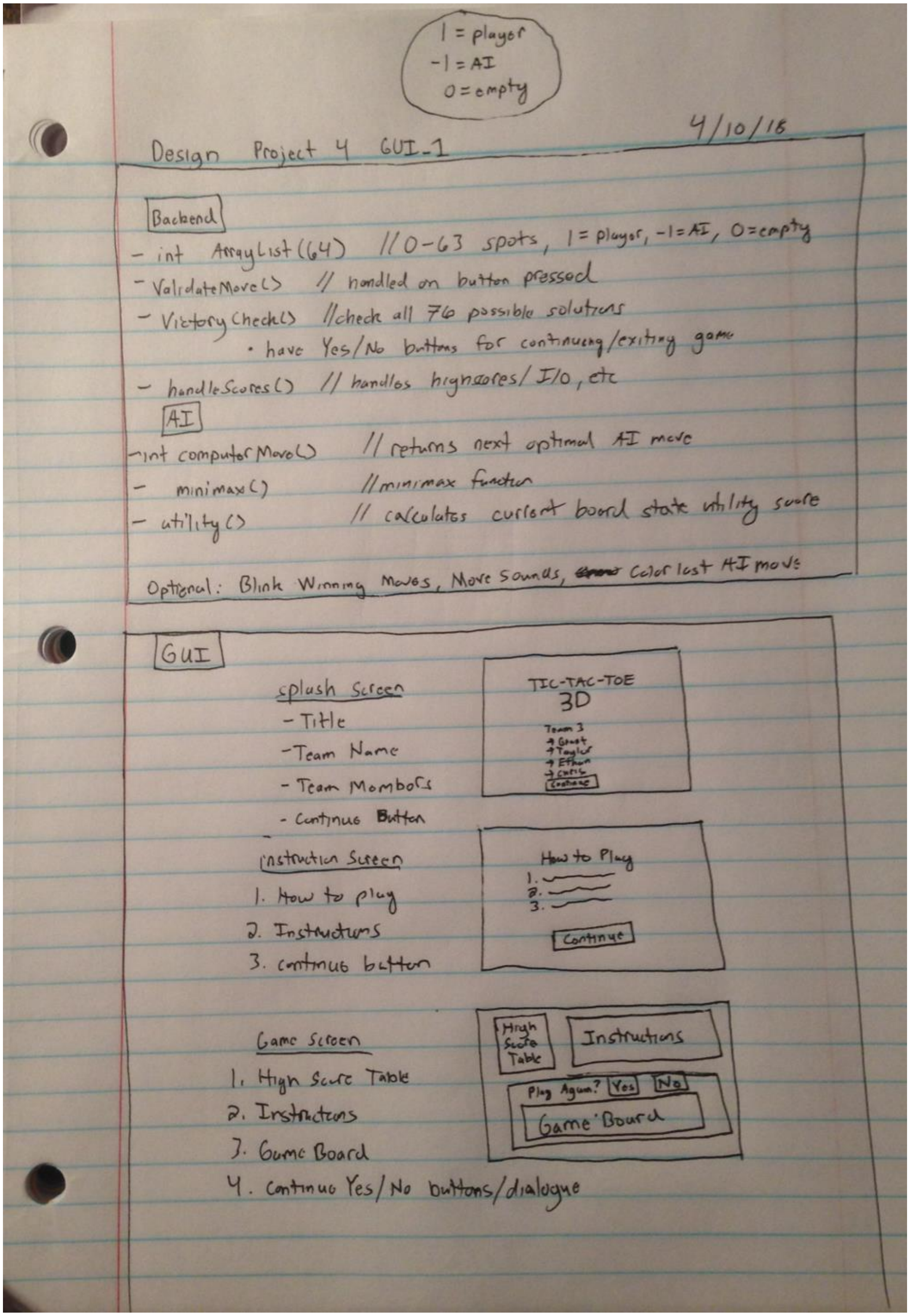
- Ncurses GUI made it easy to navigate the game screen: ***4/5***
- Ncurses GUI made it easy to understand the instructions and play the game: ***3/5***
- Ncurses GUI overall aesthetic: ***2/5***
- Problems overall:
    - The game screen was very hard to read and understand, it was confusing to tell which move went where because of the lack of color. I did not understand at first how the game worked but I eventually caught on.

Peer Review #3:

- Ncurses GUI made it easy to navigate the game screen: ***3/5***
- Ncurses GUI made it easy to understand the instructions and play the game: ***3/5***
- Ncurses GUI overall aesthetic: ***5/5***
- Problems overall:
    - It was different than any other games that I had played so it was confusing at first, the game board did not help in clarifying what was going on.

With these three Peer Reviews, it is apparent that the Ncurses GUI suffers from a lack of aesthetic, clear navigability, and clearness of instruction. GUI-1 paid respects to these insights by taking a step forward in the right direction by making a pleasing interactive aesthetic with the streamlined color layout and sleek buttons while offering straightforward navigability through a few transitional and direct accesses to the game instructions at all times during game-play.

Overall Design Structure:



1 = playor
-1 = AI
0 = empty

Design Project 4 GUI-1 4/10/18

Backend

- int ArrayList(64) //0-63 spots, 1 = playor, -1 = AI, 0 = empty
- ValidateMove() // handled on button pressed
- VictoryCheck() //check all 76 possible solutions
    - have Yes/No buttons for continuing/exiting game
- handleScores() // handlos highscores/I/O, etc

AI

- int computerMove() // returns next optimal AI move
- minimax() //minimax function
- utility() // calculates current board state utility score

Optional: Blink Winning Moves, Move Sounds, Color last AI move

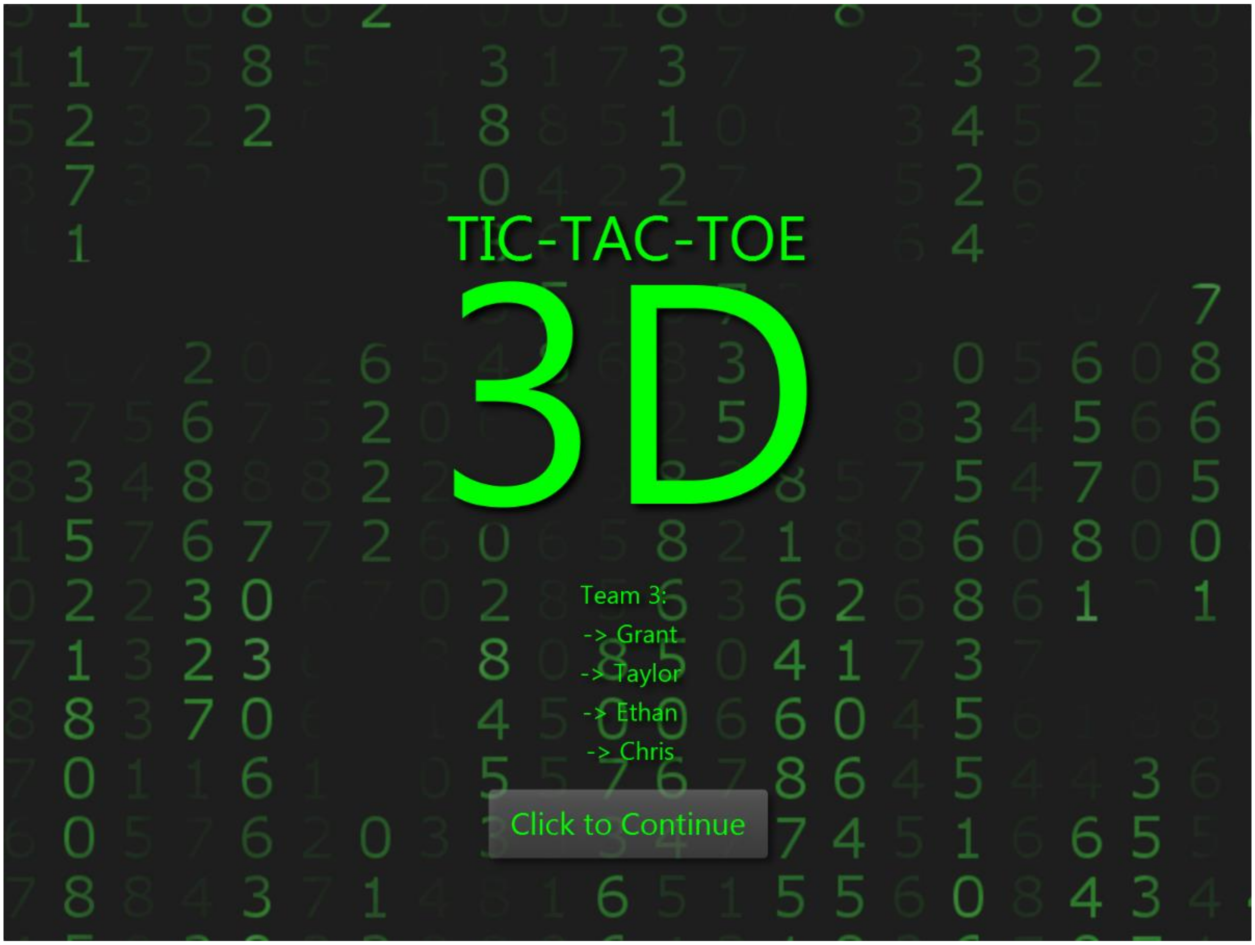
GUI

splash Screen
- Title
- Team Name
- Team Members
- Continue Button

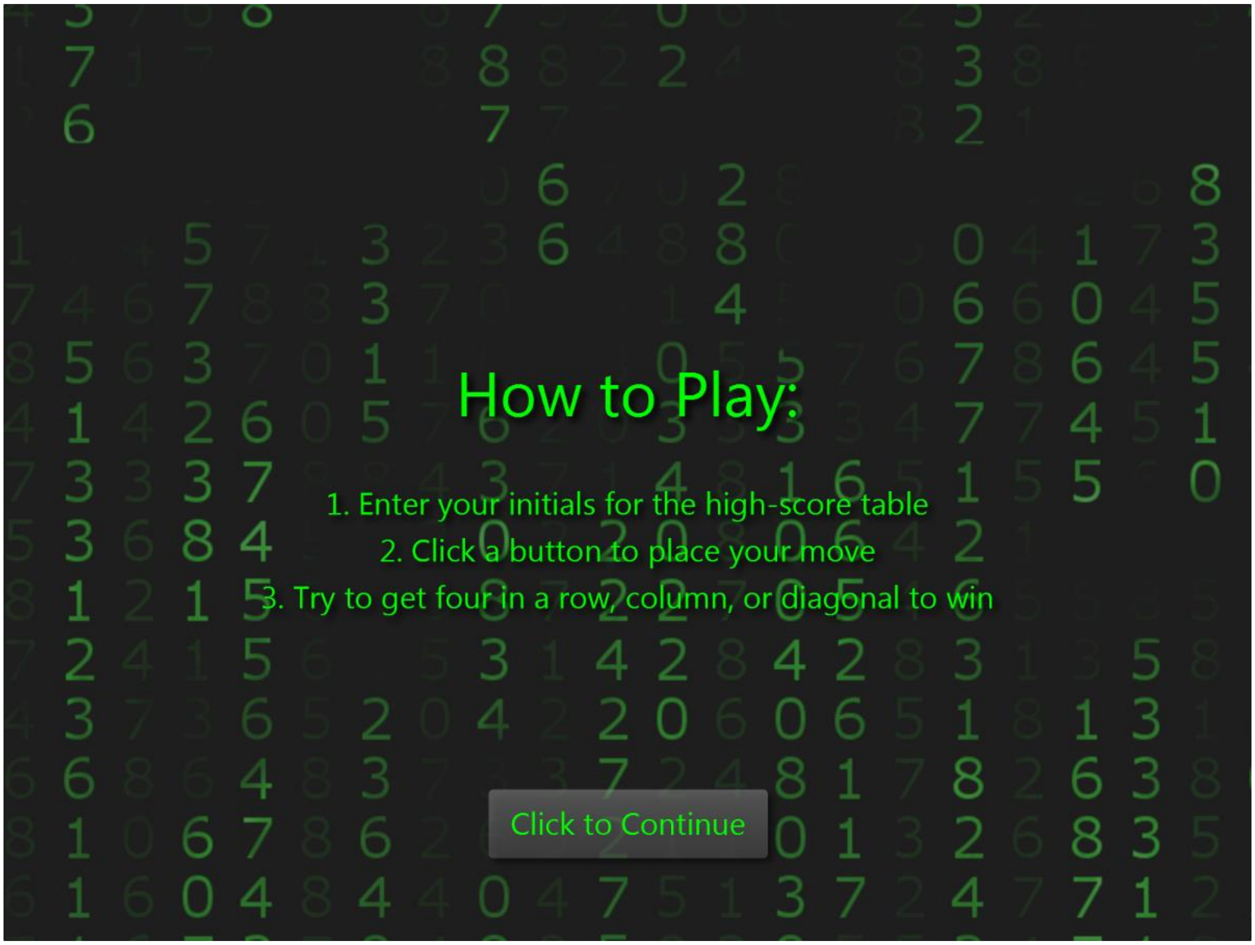
instruction Screen
1. How to play
2. Instructions
3. continue button

Game Screen
1. High Score Table
2. Instructions
3. Game Board
4. Continue Yes/No buttons/dialogue

Splash Screen:



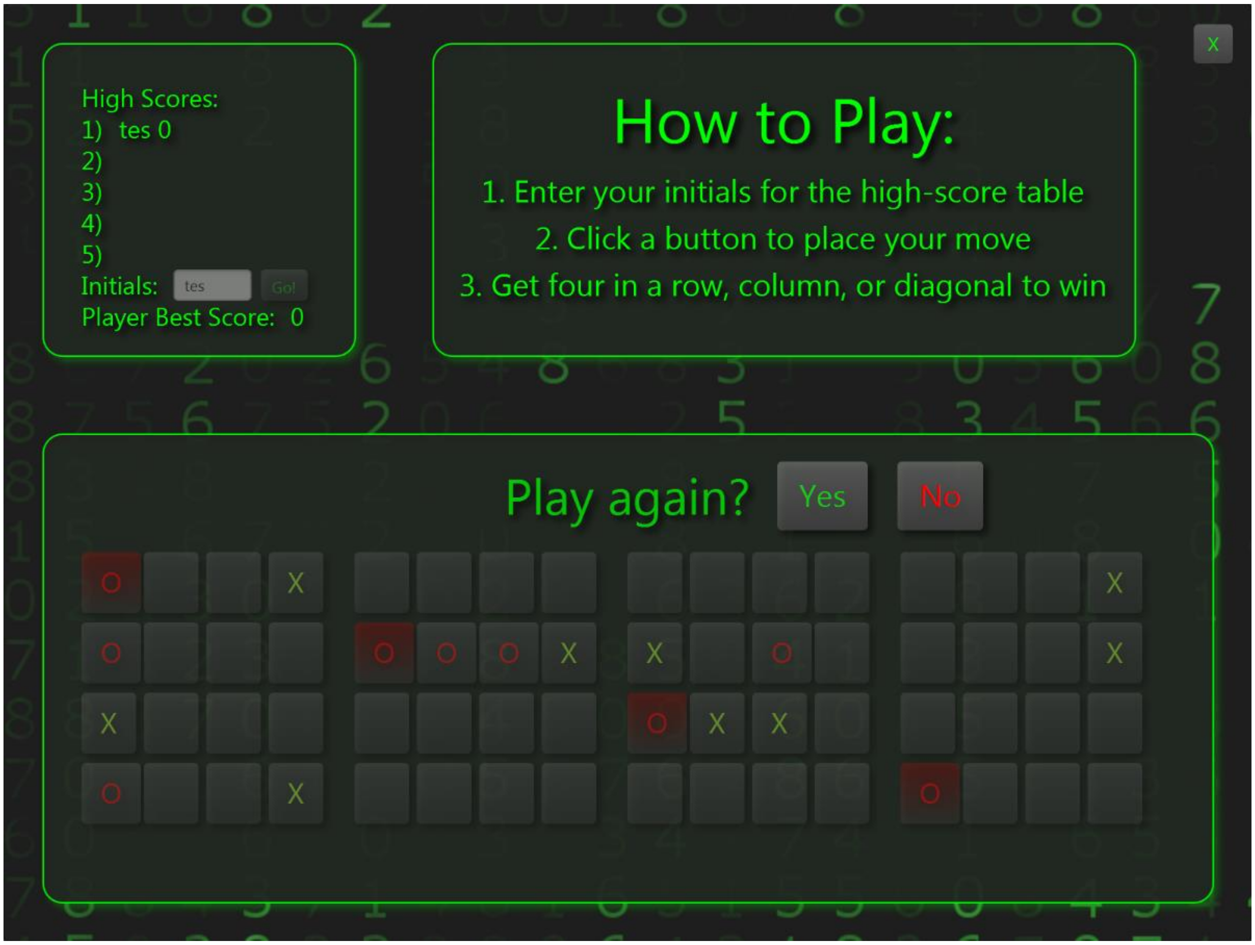
Instructions Page:



How to Play:

1. Enter your initials for the high-score table
2. Click a button to place your move
3. Try to get four in a row, column, or diagonal to win

Game Page:



High Scores:
1) tes 0
2)
3)
4)
5)
Initials: tes Go!
Player Best Score: 0

How to Play:

1. Enter your initials for the high-score table
2. Click a button to place your move
3. Get four in a row, column, or diagonal to win

Play again? Yes No

O			X
O			
X			
O			X

O	O	O	X

X		O	
O	X	X	

			X
			X
O			