TECHNICAL REPORT

INTRODUCTION

A Tic-Tac-Toe game specially curated for our astronauts heading towards Mars to establish a permanent settlement. We Team Ares have taken up the challenge to entertain our crew with this engaging game.

The program has been written in JavaScript and the gripping UI is a result of HTML and CSS. To create a witty opponent we have used Minimax algorithm along with alpha beta pruning to get to a move swiftly. Besides that our program follows a functional programming paradigm.

Code supported by Google Chrome, Microsoft Edge, Mozilla Firefox, Safari. Try it on your favourite browser.

FEATURES

1.Astronaut VS Spacebot (Human VS AI)

- We provide you with 4 Difficulty levels or competitors to defeat
- Grid size : a variety of boards are available (3x3, 4x4, 5x5). If you want to challenge yourself go beyond 3x3.

2.Astronaut VS Astronaut (Human VS Human)

- Grid size : a variety of boards are available (3x3, 4x4, 5x5). If you want to challenge yourself go beyond 3x3.
- Hint/Lifeguard : if in the course of the game you find yourself stuck don't worry the lifeguard has your back but remember it's just once in 3x3, twice in 4x4 and thrice in 5x5

3. Various Board sizes

- 3X3
- 4X4
- 5X5

4. Various Difficulty levels / Depths

- Lieutenant (level 1)
- Major (level 2)
- Colonel (level 3)
- Marshal (level 4)

5. Music Button : to keep you going

6. Home button : to change the specifications of your game at any point of time

7. Start button : to start game

8. Replay : to start a fresh game on a new board

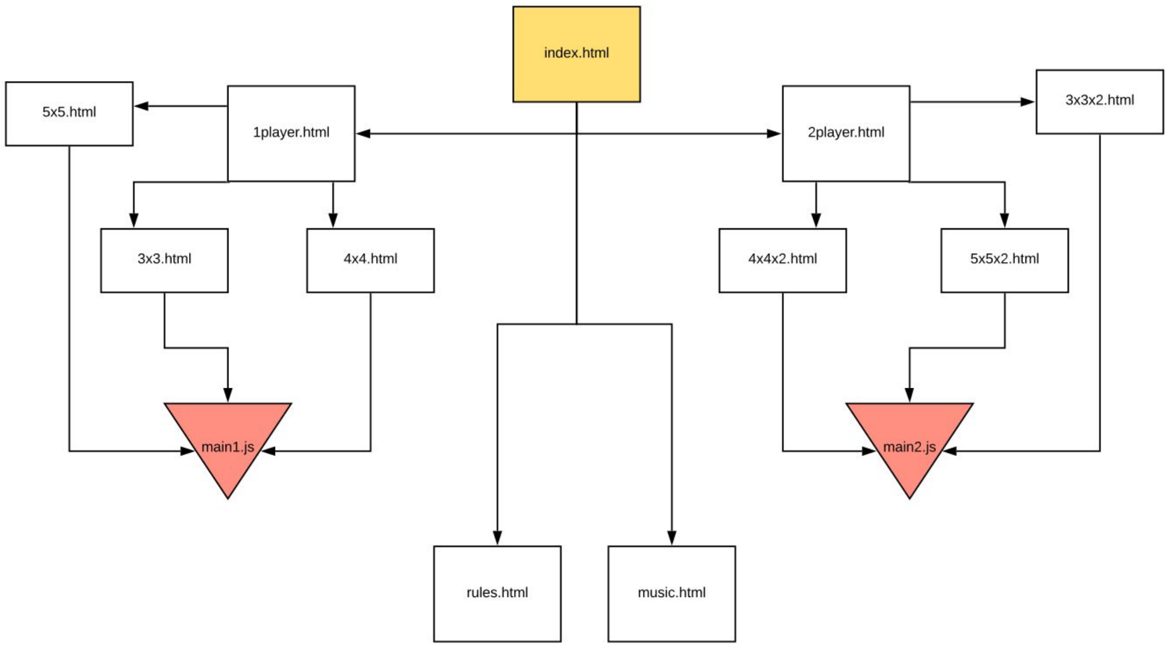
9. Flexibility : to switch to a new grid at any point of time in the game

10. Back button : to return back to previous page

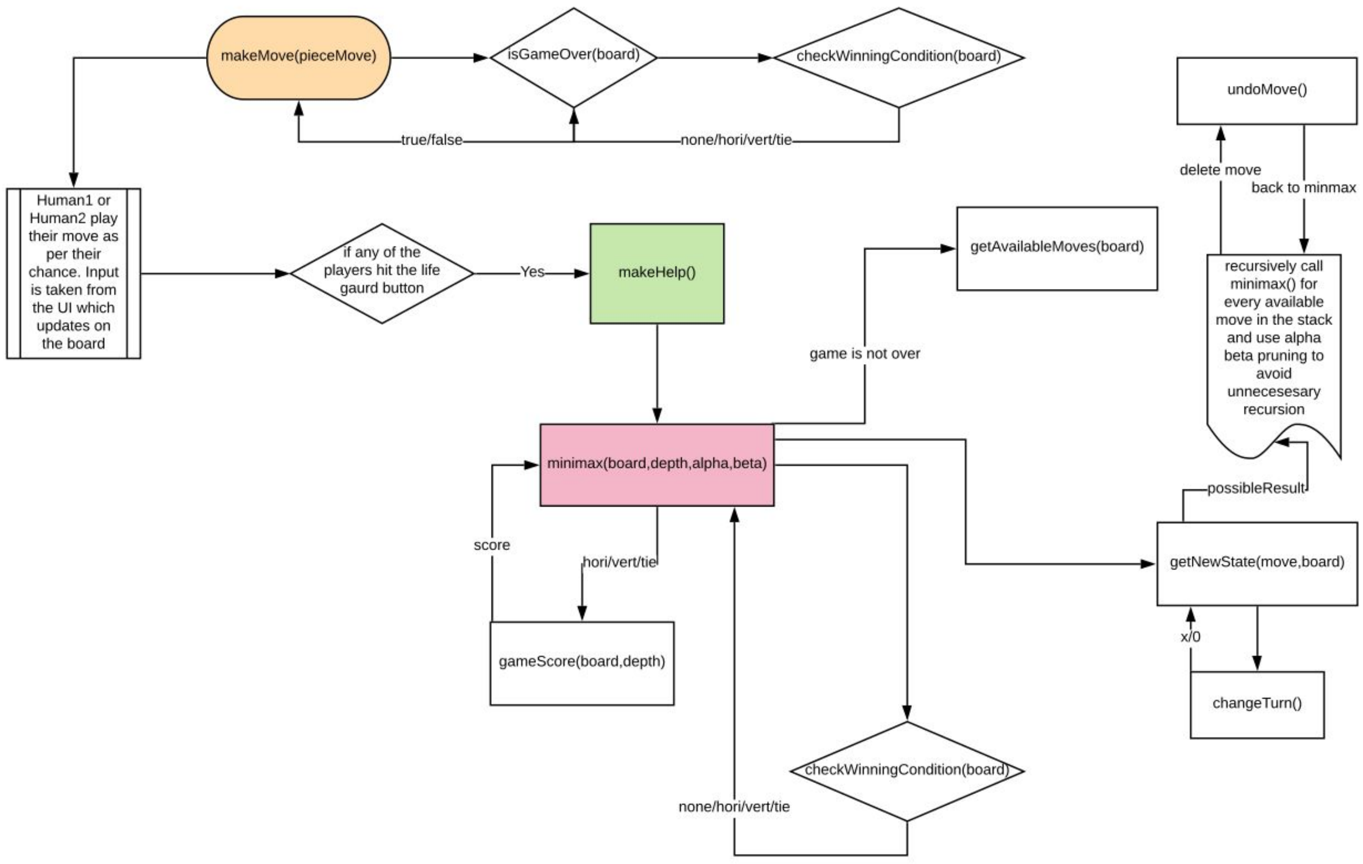
11. Opportunity to choose : who should start the game first so that it is a fair play

WORKFLOW

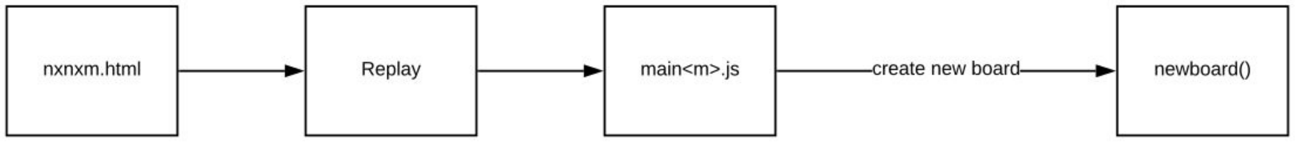
Architectural Design



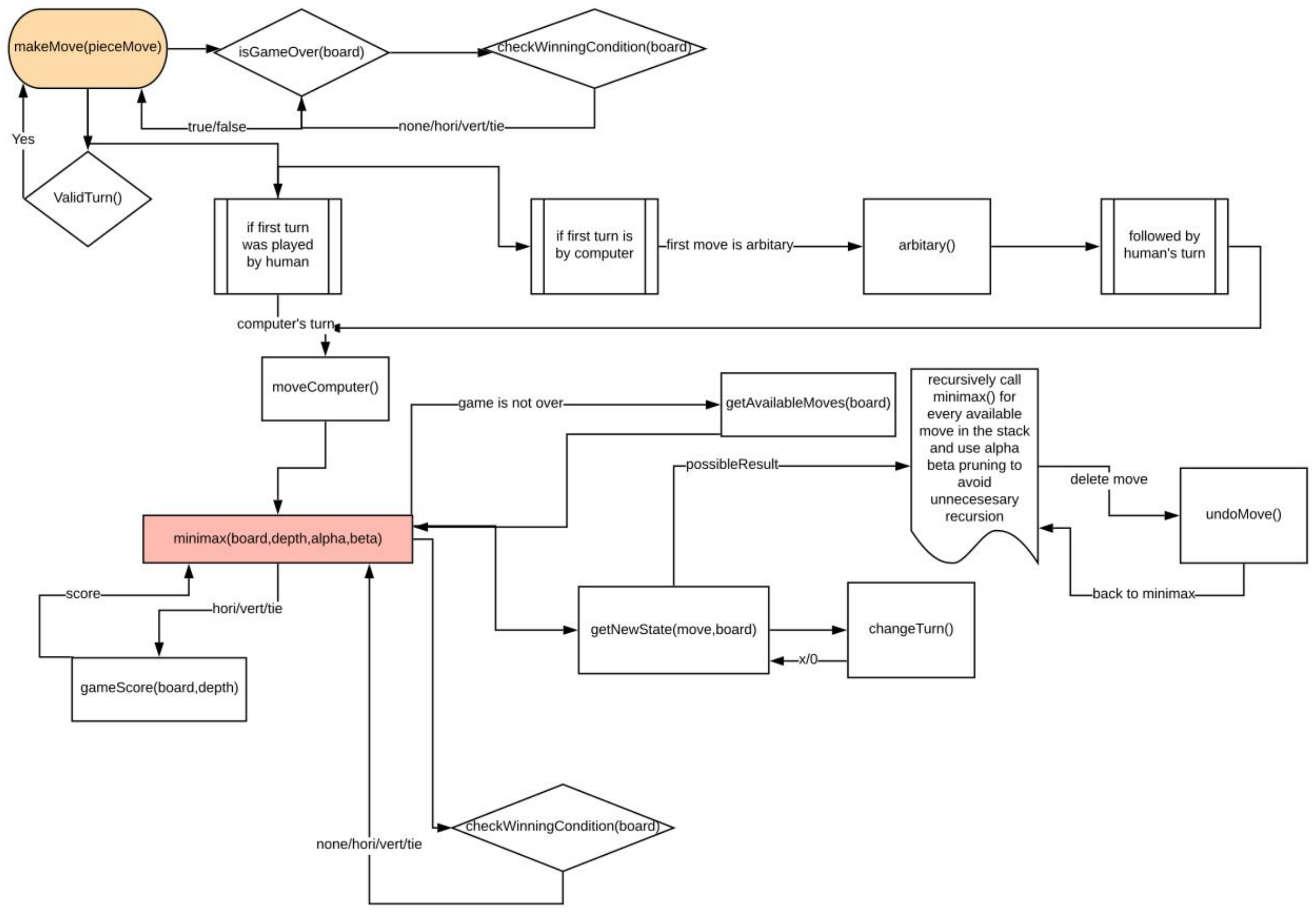
Astronaut vs Astronaut
main2.js workflow


makeMove(pieceMove)
isGameOver(board)
checkWinningCondition(board)
true/false
none/hori/vert/tie
undoMove()
delete move
back to minmax
Human1 or Human2 play their move as per their chance. Input is taken from the UI which updates on the board
if any of the players hit the life gaurd button
Yes
makeHelp()
getAvailableMoves(board)
game is not over
recursively call minimax() for every available move in the stack and use alpha beta pruning to avoid unnecesesary recursion
minimax(board,depth,alpha,beta)
possibleResult
score
hori/vert/tie
getNewState(move,board)
x/0
gameScore(board,depth)
changeTurn()
checkWinningCondition(board)
none/hori/vert/tie

Replay/ new board creation workflow



Astronaut vs Spacebot
Main1.js workflow



Overview of the code snippet

A detailed description of the different functions defined in the code.

validTurn()
Handles the errors of inappropriate entries. Check whether the game is over or not, and in case when game is over , it disables the makeMove function.
Returns isValid value to makeMove().

newboard()
Create and initialize new board based on the size required

arbitrary()
When computer plays the first move then it is called to play any random move

makeMove(pieceMove)
This is the master function that is called by the html page. Checks valid turn, checks if game is ongoing or if it is over, updates the board with the played move, updates the active turn, for 2 player game it hits the html page to take input from UI, and in case of the computer's turn it calls the movecomputer() function

movecomputer()
Takes care of the game played by AI. Calls the minimax(board, depth, alpha, beta) function, updates AI's move on the board, checks if game is over, and displays necessary messages

gameScore(board, depth)
Calculates score of the winner based on the depth at which it won the game by calling the checkWinningCondition(board) function

makeHelp()
This is your life guard. It calls the minimax algorithm and plays the best move for you, but remember there is a limit to the help you get.

minimax(board, depth, alpha, beta)
This is the function that handles all the magic that we see. It handles the game played by AI and serves as the lifeguard to our human players. This function recursively calls itself to find the best move to win the game. It evaluates the suggested move in every step and the alpha beta pruning added in the code makes sure that the best move is reached at the earliest. The difficulty level of the game is taken care of by limiting the depth of recursion of the minimax function.

Reason for applying alpha beta pruning;
When we had not used alpha beta pruning we realised that AI was taking to long to decide on it's move, as a result of which the player's experience was hampered. Alpha beta pruning avoids any unnecessary recursion and gets to the best result at the earliest.

undoMove(board, move)
Deletes the most recent move made and changes the active turn

getNewState(move, board)
To change active turn and update the board

changeTurn()
Just switches the active turn from X to O and vice versa

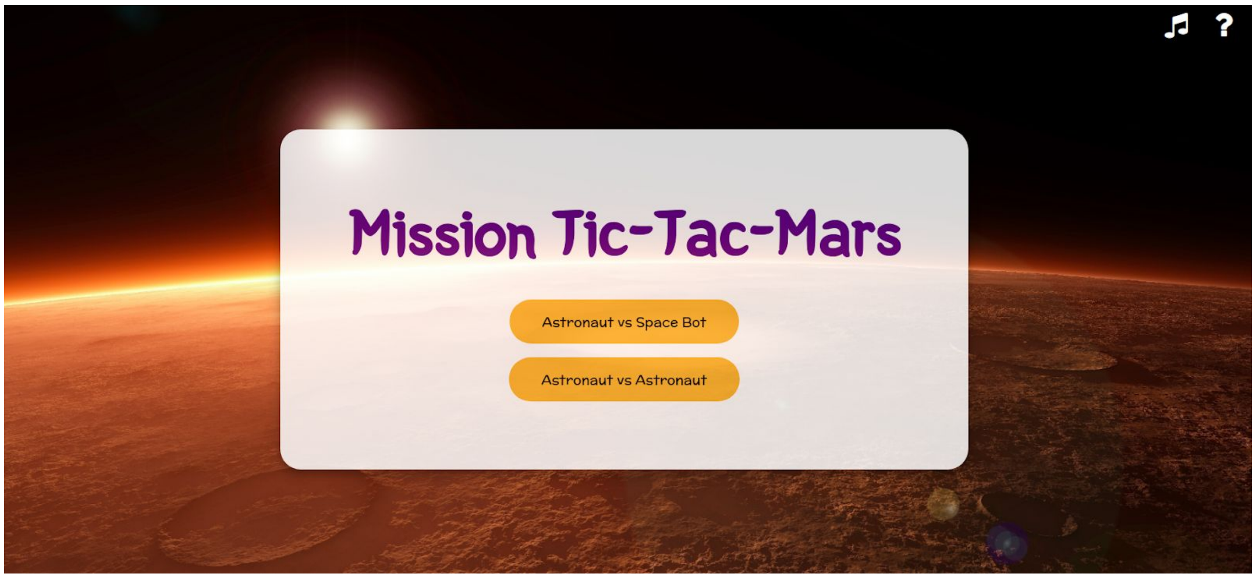
getAvailableMoves(board)
Creates a stack of all the unoccupied cells on the board and passes it to minimax to find the best spot from it

checkWinningCondition(board)
One of the most important functions, it’s job is to see if either of the players have won the game in any possible way : horizontally, vertically, diagonally or if it is a tie or if the game is not over

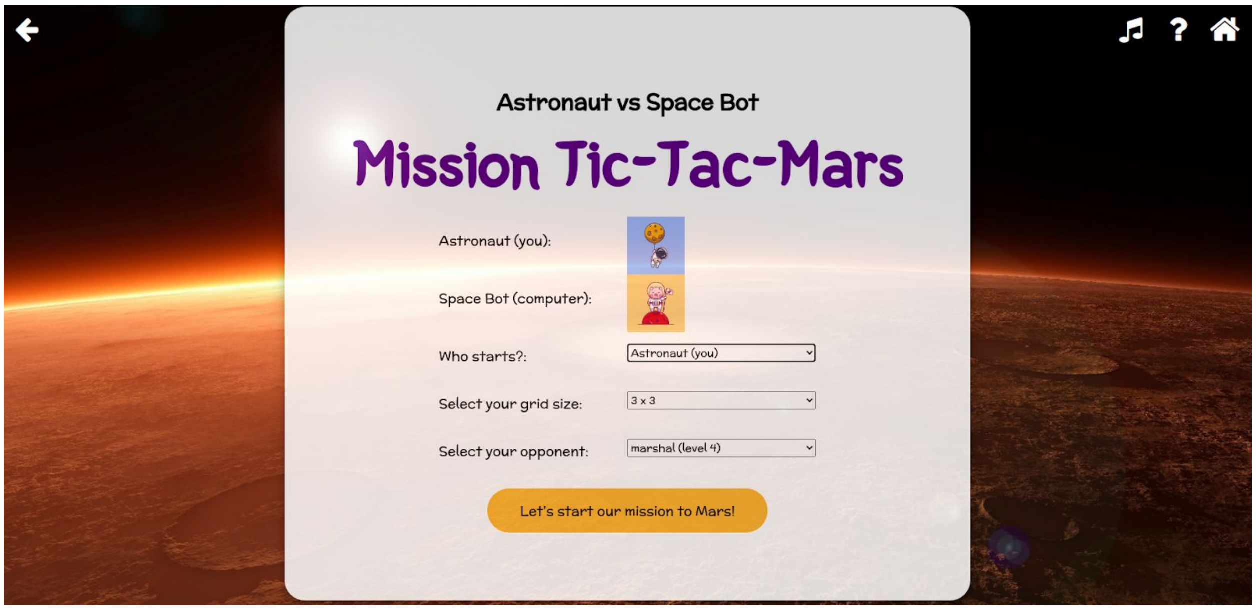
isGameOver(board)
Displays a message to the players based on the state of the game : WON, LOST, TIE, ONGOING

Sneak Peak into the UI

Home page



Select your game specifications



Astronaut vs Space Bot

Mission Tic-Tac-Mars

Astronaut (you):

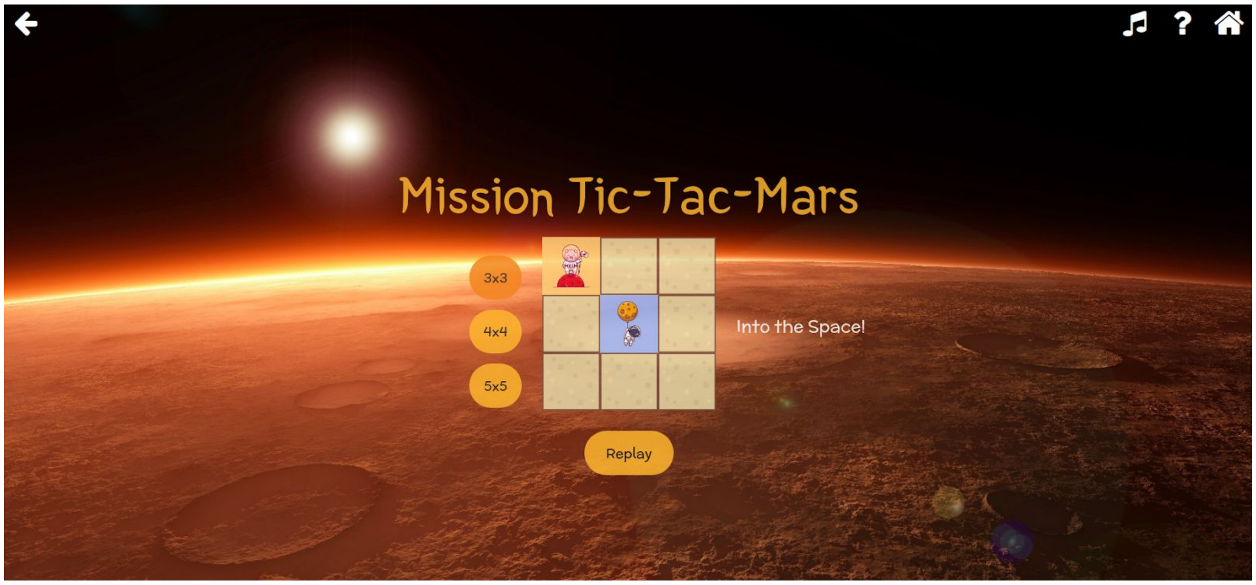
Space Bot (computer):

Who starts?: Astronaut (you)

Select your grid size: 3 x 3

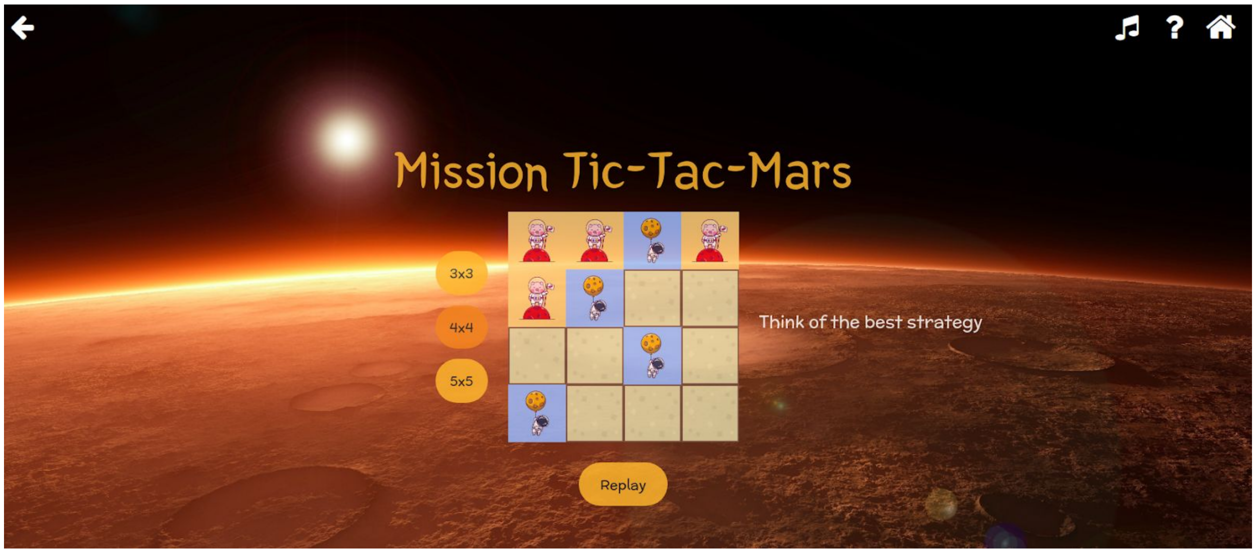
Select your opponent: marshal (level 4)

Game page

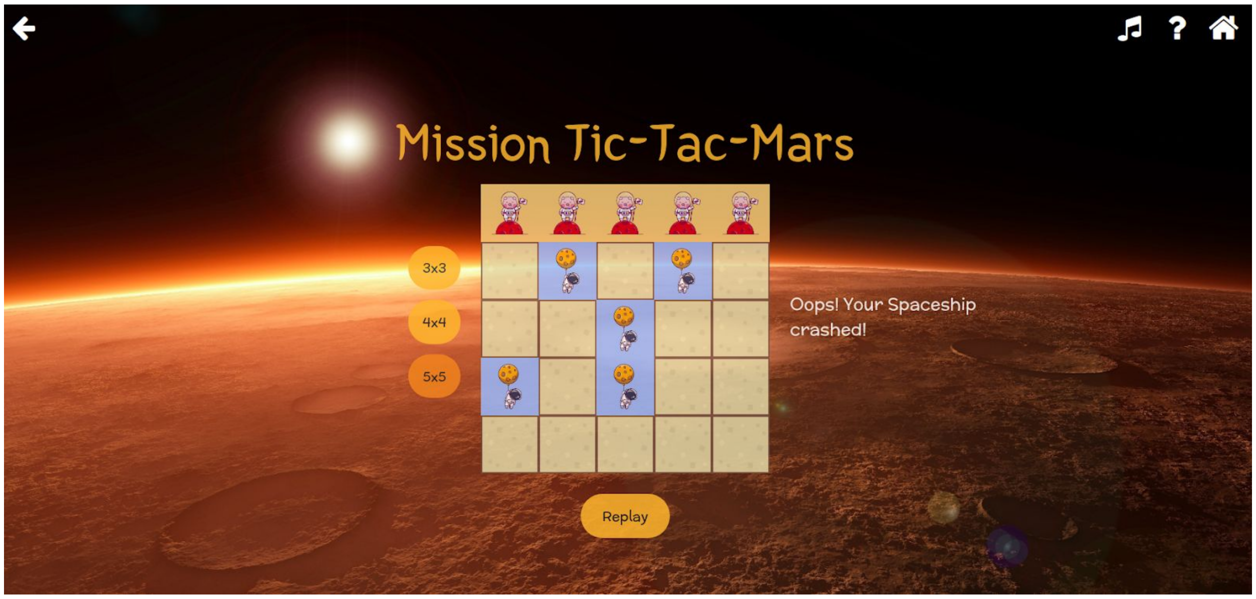


Mission Tic-Tac-Mars

Into the Space!



Think of the best strategy



Oops! Your Spaceship crashed!

Go ahead and create your own game with your customisation.
Hope you land on mars safely, try your luck before it is too late.