MINI-PROJECT – 1
(2020-2021)

Build and Deploy an application for a Game (Rock,Paper and Scissor)

MID-TERM REPORT

Institute of Engineering &Technology

Submitted by-
Shantanu Saraswat
(181500651)

Supervised By:-
Priya Agrawal Mam

Technical Trainer

Contents

Abstract

Rock paper scissors (also known by other permutations such as **scissors paper rock**, **scissors paper stone**, **paper rock scissors**, or **ro-sham-bo**) is a hand game usually played between two people, in which each player simultaneously forms one of three shapes with an outstretched hand. These shapes are "rock" (a closed fist), "paper" (a flat hand), and "scissors" (a fist with the index finger and middle finger extended, forming a V). "Scissors" is identical to the two-fingered V sign (also indicating "victory" or "peace") except that it is pointed horizontally instead of being held upright in the air. A simultaneous, zero-sum game, it has only two possible outcomes: a draw, or a win for one player and a loss for the other.

A player who decides to play rock will beat another player who has chosen scissors ("rock crushes scissors" or sometimes "blunts scissors"[1]), but will lose to one who has played paper ("paper covers rock"); a play of paper will lose to a play of scissors ("scissors cuts paper"). If both players choose the same shape, the game is tied and is usually immediately replayed to break the tie. The type of game originated in China and spread with increased contact with East Asia, while developing different variants in signs over time. Other names for the game in the English-speaking world include **roshambo** and other orderings of the three items, with "rock" sometimes being called "stone".

Rock paper scissors is often used as a fair choosing method between two people, similar to coin flipping, drawing straws, or throwing dice in order to settle a dispute or make an unbiased group decision. Unlike truly random selection methods, however, rock paper scissors can be played with a degree of skill by recognizing and exploiting non-random behavior in opponents.

Introduction

In our project we address the problem of how to implement a stateful multi-player game while minimizing trust in third-parties. By “stateful” we mean that there is some meaningful state that persists between separate interactions, as opposed to each interaction being selfcontained, as in chess, tic-tac-toe, etc. A typical example of a stateful game is poker, since each hand affects the money balances held by each player, which persist between hands. Most games that exist today prevent cheating either by centralization — requiring that all interactions be routed through a trusted third-party solely responsible for maintaining the game state; or else by obfuscating the software running on each player’s device to prevent them from illegally altering the state. In our project we seek a way to use cryptographic technology to alleviate these constraints. Our project implements a minimal game of this type as a proof-of-concept.

About This Game

As alluded to previous sections, we incentivize proper behavior in the case of one player prematurely ending, or aborting, the game. Ishai et al [2], show that in the case of a malicious majority, it is possible for the parties in the MPC protocol to know who aborts. Conversely, parties, such as the voters, who do not take place in this computation, cannot ascertain who the aborting party is. Within these results, we attempt to motivate ‘proper’ behavior within reason. Our protocol allows a user who believes the game has been aborted to post this fact to the ledger. The player includes the proof of the game so far in this notification which is stored in the users’ account state. The opposing player now has a set amount of time to refute the abort claim. At this point, there are two possibilities:

1. The accused player rebuts the abort claim and posts their own move to the ledger to continue the game,

2. Or the accused player fails to post and the skill involved comes out in favor of the accuser Due to the fact that each transaction with the ledger has a set cost in stake, it is far cheaper to continue game play through player-to-player interaction — not to mention more timely. In gameplay through the ledger,

players must wait for the ledger to be resolved before they can make the next move. It is worth noting, that while the voters do not know which player truly aborted, the players themselves do know. That being said, the honest player can blacklist the malicious player for future games.

How This Game Works

Rock Paper Scissors Protocol Secure An existing implementation by Thofmann [7], called Rock Paper Scissors Protocol Secure (RPSPS), is based on the same underlying premise as ours; namely, the use of a decentralized system to help support two-party games of RPS in which state is maintained between games. RPSPS uses Bitcoin as opposed to our proof-of-stake ledger system. The format of the transactions between players follows a similar pattern to ours, using commitments to ensure the integrity of moves. However, despite these similarities, their adversarial model diverges significantly from ours. We consider malicious players in addition to their malicious distributed third-party (the “voters” in our system). Thus, they do not deal with the case of a malicious player replaying games or aborting mid-game. There is also no method for verifying the outcome of a game; the loser of an encounter has little incentive to actually hand over their stake.

Multiparty Computation (MPC) with a Malicious Majority While the setting we are considering is very similar to that of MPC — two mutually distrusting parties who wish to compute a joint output — the security in our model is fundamentally 3 different. Of the main security properties of MPC, we are concerned primarily with the fairness of the outcome. We are not concerned with hiding the other player’s input; in fact, we want players to be able to tell if they have won or not, so that they can verify the skill level changes determined by the outcome of the game. Even though we are only concerned with the fairness of the protocol, we can still apply results from MPC in the case where we have a malicious majority (in a two party interaction, even one malicious player is considered a majority.

Security Argument

We allow for the following limited malicious activity from players:

• Malicious parties (either the players themselves, or a third party who has access to the transcript) replaying a game transcript to the voters.

• Either player quitting mid-encounter to prevent the game from reaching a conclusion.

• Either player attempting to change their move after seeing the other's move
• Man-in-the-middle attacks between the two players.

• A malicious party forging an encounter (or parts thereof) to post to the voters.

Architecture

The game ecosystem as a whole may be conceptualized as a number of players who periodically interact with each other and with a "ledger" that represents the global state, available in common to all players. The principle guiding our implementation is that each communication should involve as few parties as possible. Communication, when it does occur, goes over a peer-to-peer network for voters, and from point to point in the case of players. Players communicate with each other to play out encounters, which they then submit to the voters for verification and "disposition" (i.e. the process of updating the ledger to reflect the outcome of the encounter). Accordingly, in our design we have taken pains to ensure that gameplay only requires the players to interact with the ledger at the beginning and end of an encounter. In particular, 1The winner gains (and the loser loses) an amount of points equal to 1000 1+e W−L 1000 , where W, L are the skill ratings of the winner and loser before the encounter, respectively. 2 the players need not record their individual moves in the ledger as they make them; instead, the moves are assembled by the players themselves into a encounter proof transcript that they submit to the ledger at the end. Additionally, at the start, the players must query the ledger in order to record the beginning of their encounter; this prevents players from being in more than one encounter at the same time.

Features:-

1. Provides a intresting layout and GUI, so that players will not get bored .
2. Simple, fast and convenient for playing this luck based game.
3. Greater customer satisfaction!!

Hardware Requirements:-

. Latest Configuration Laptop

Software Requirements:-

1. Visual Studio(Version 1.48)
2. Tools:-
    a. HTML5
    b. CSS3
    c. JAVASCRIPT
    d. BOOTSTRAP
    e. NOTEJS OR MONGODB

Objective:-

Rock paper scissors is often used as a fair choosing method between two people, similar to coin flipping, drawing straws, or throwing dice in order to settle a dispute or make an unbiased group decision. Unlike truly random selection methods, however, rock paper scissors can be played with a degree of skill by recognizing and exploiting non-random behavior in opponents.

Implementation Details:-

We created three individual components as part of our software prototype: relay, voter, and client. For more information about the implementation and the code refer to Section. Relay instantiates a TCP server that voters connect to. By keeping a list of connected voters, it can relay a message received from any voter to all other voters to simulate an idealized lossless peer-to-peer network. In the current implementation, a client does not directly join the voter network, but instead posts and receives message from voters through a separate HTTP server interface on the relay. Voter is a game proof verifier that records and maintains the global game state in a SQLite database. A voter upon receiving game proof from clients through relay's HTTP interface, 10 will validate the proof according to predefined rules and broadcast its validation result via relay to other voters. A voter also receives validation results from other voters. Once a consensus is reached, the SQLite database representing the ledger is updated accordingly. A voter must have a valid account already existing on the ledger to do so. All components take advantage of asynchronous network programming featured in Python Tornado library. It abstracts low-level TCP networking API in a event-based paradigm and uses advanced Python coroutine feature to support concurrent access. This greatly eases the implementation of communications between components. We use PyCrypto for the cryptographic primitives, including RSA encryption and digital signatures. For hash functions, we use Python's built-in hashlib module. Since no implementation for a commitment scheme was readily available, we created our simple commitment scheme. In our design, a user concatenates

the value to be committed with a 20-byte randomly generated padding, and computes the SHA256 hash of the concatenated string as the commitment. To reveal the commitment, a user sends its committed value and previously generated padding to the verifier who computes the correct commitment and checks if it matches the previously received commitment. We represent users in our system using their unique public-keys. In addition, similar to the Bitcoin network, we use a 48-character long account ID generated from the trailing 36 bytes of the hash of a user's public key encoded in DER format. This significantly reduces the length of an account ID, while preserving the one-to-one mapping between users and account IDs. However, when users sign a message, they must include their public key along with the signature since the verifier may not necessarily have the signer's public key for verification. The verifier must also check if the signer's account ID corresponds to the received public key.

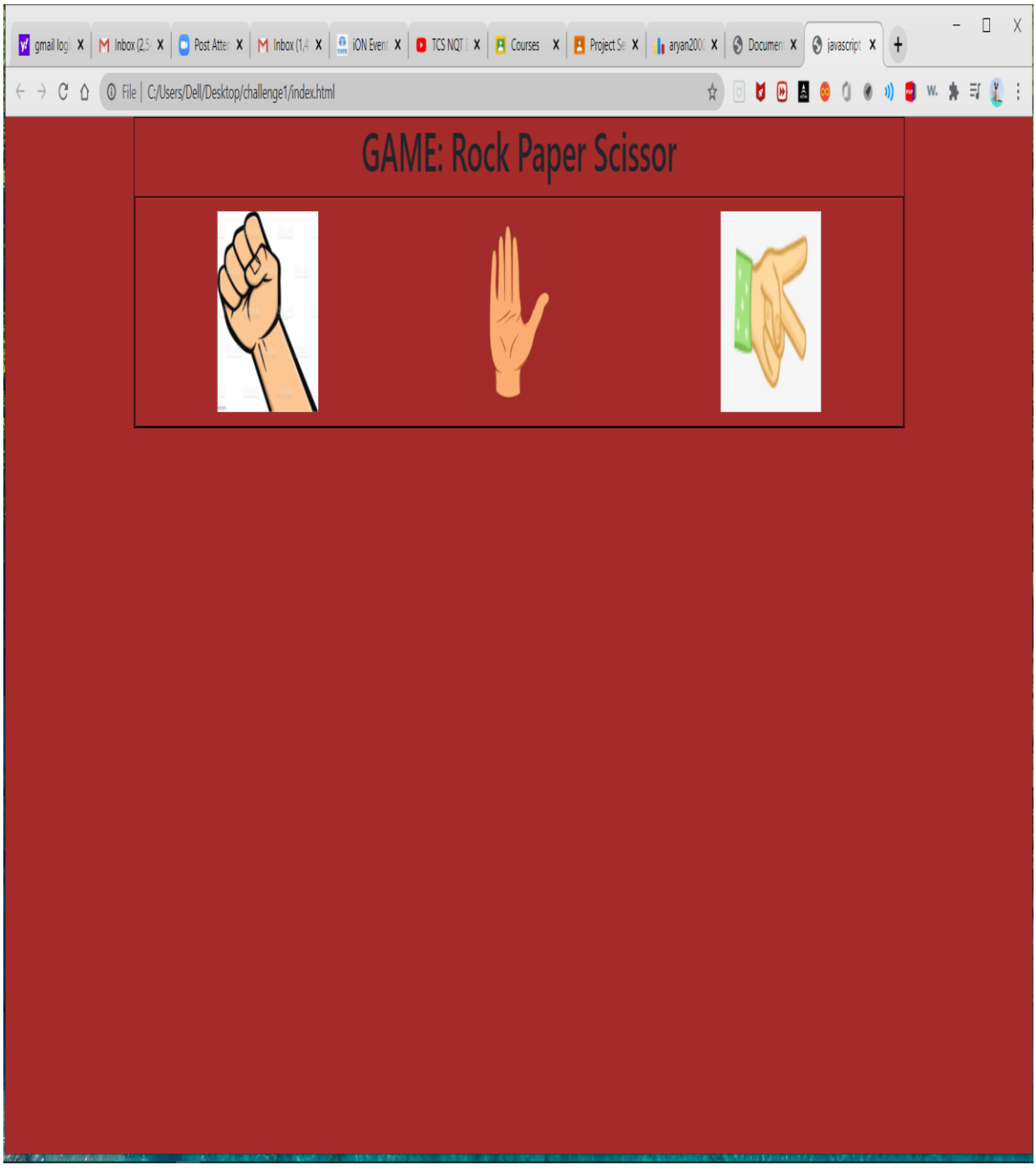
Progres

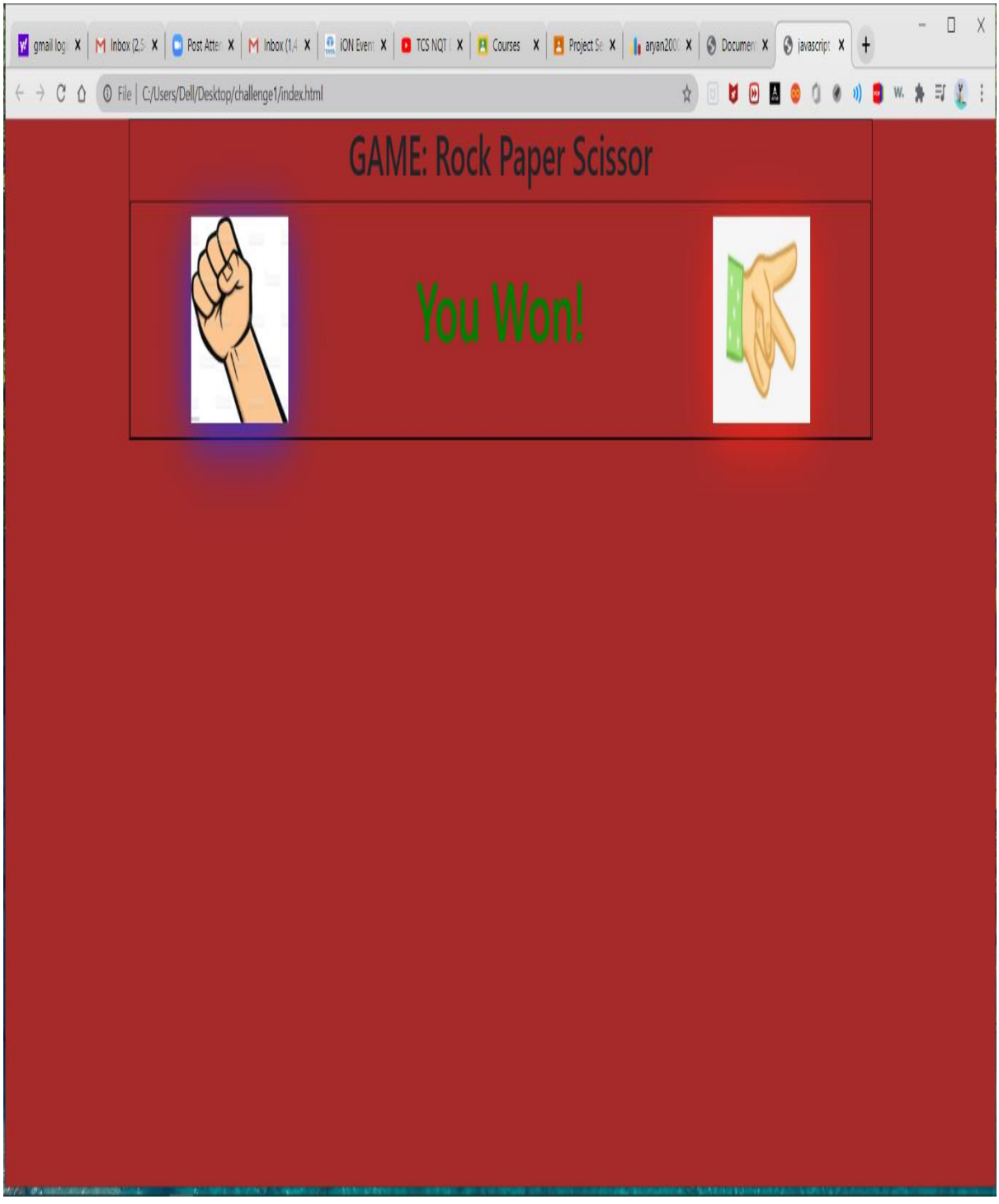
My project development is on running and almost 60-70% project is completed till now .

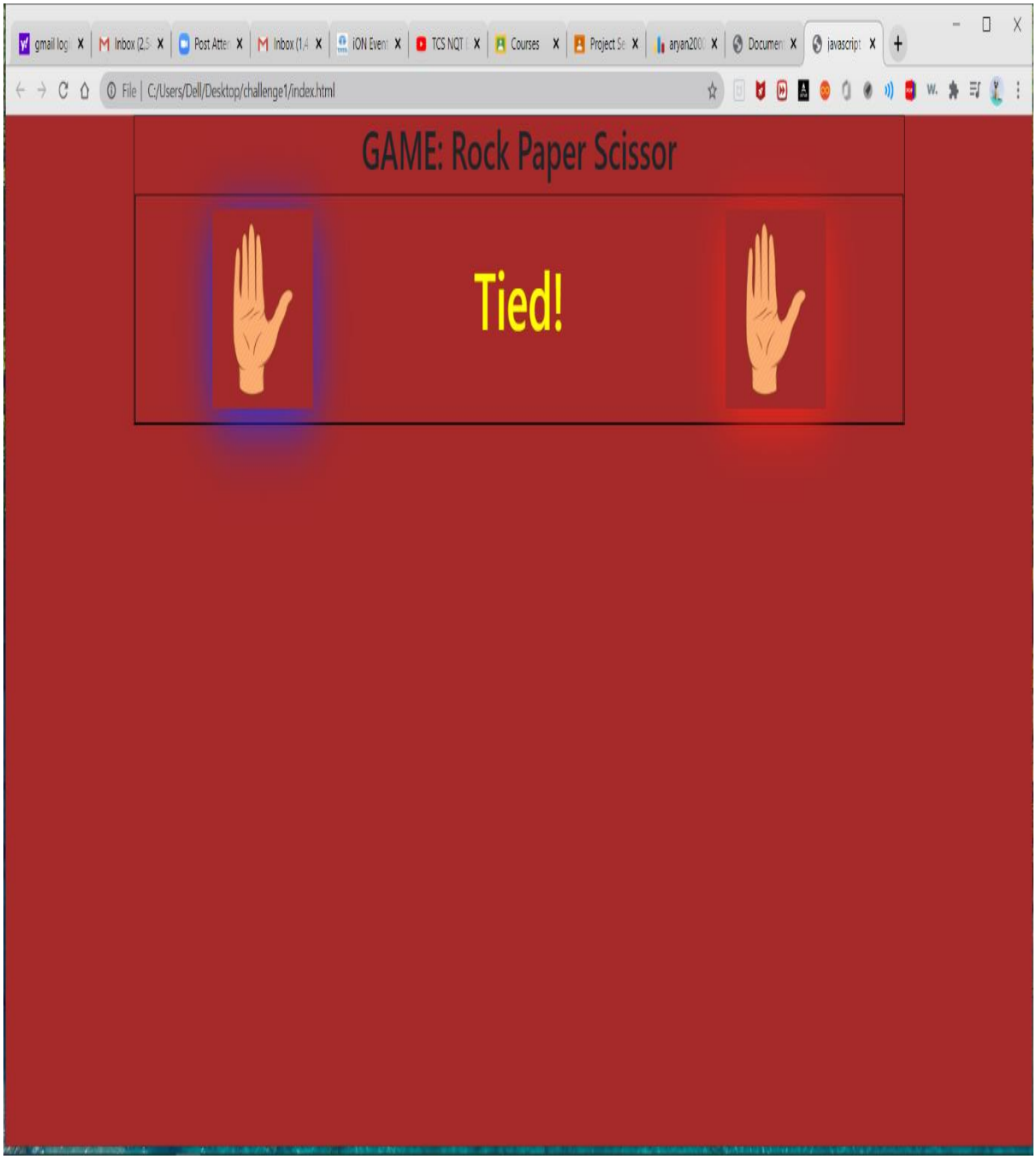
The project for this game is more efficient and reliable for users to play such a luck based game .

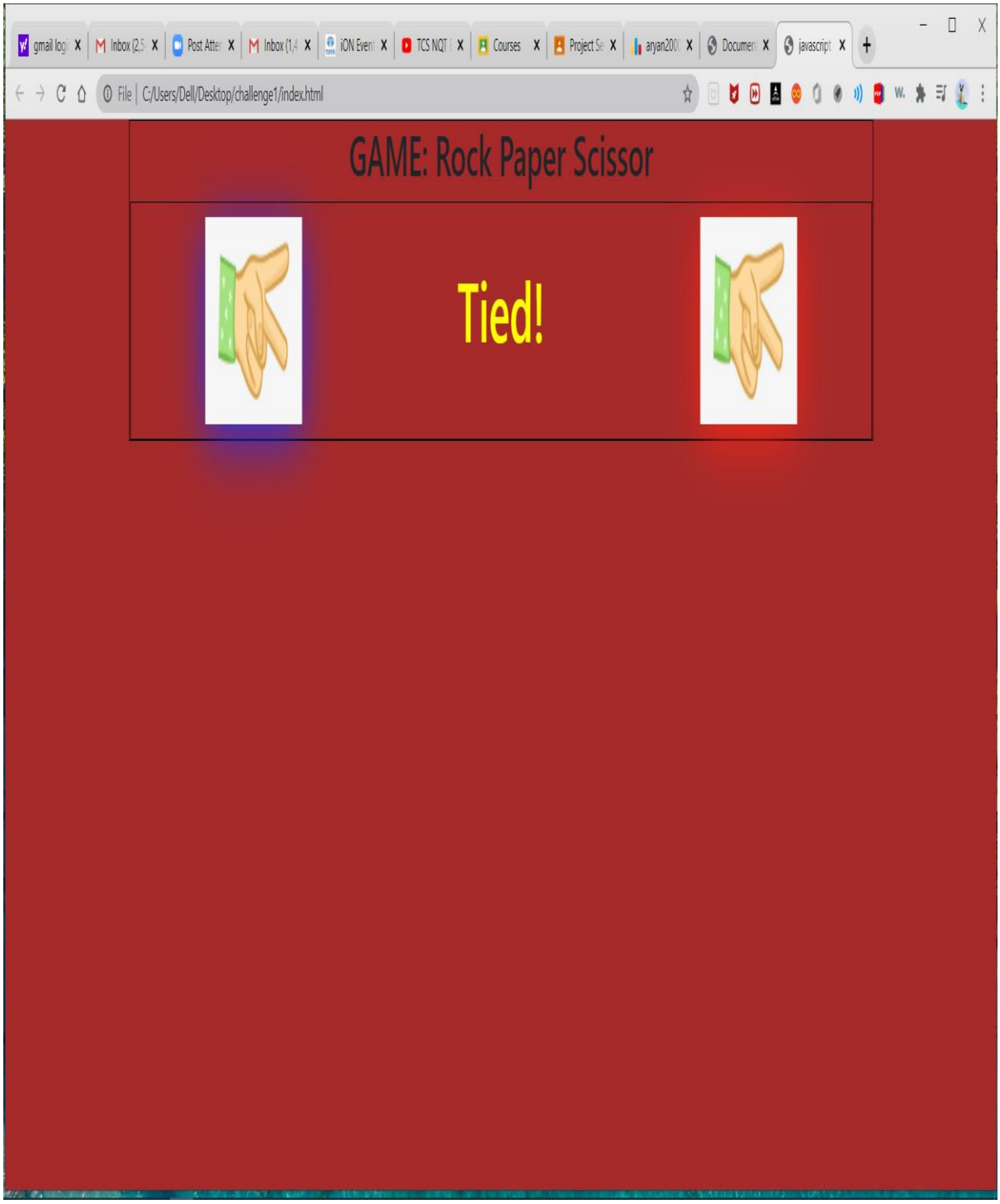
The project will deploy at least 10-15 days from now onwards.

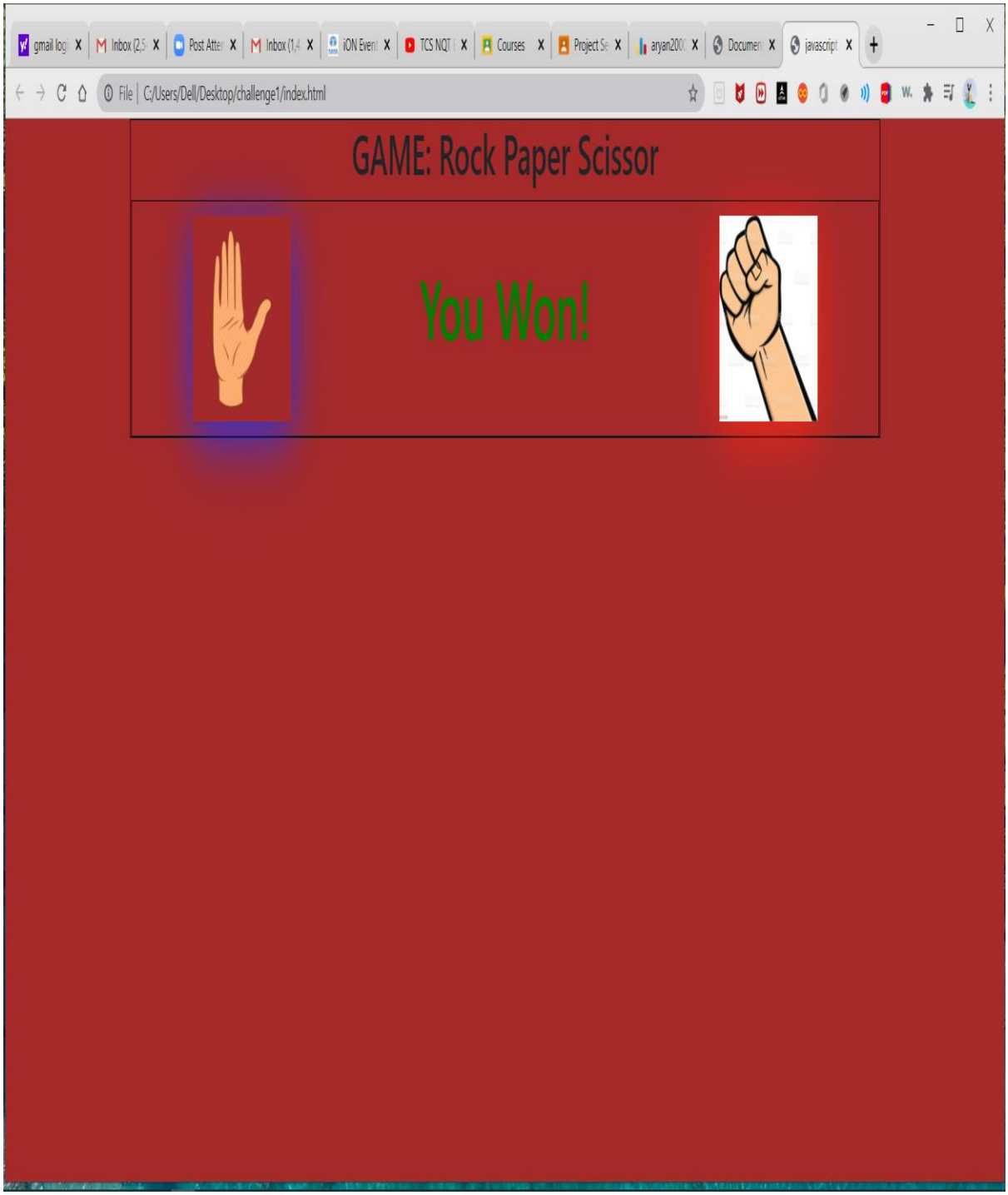
Screenshots



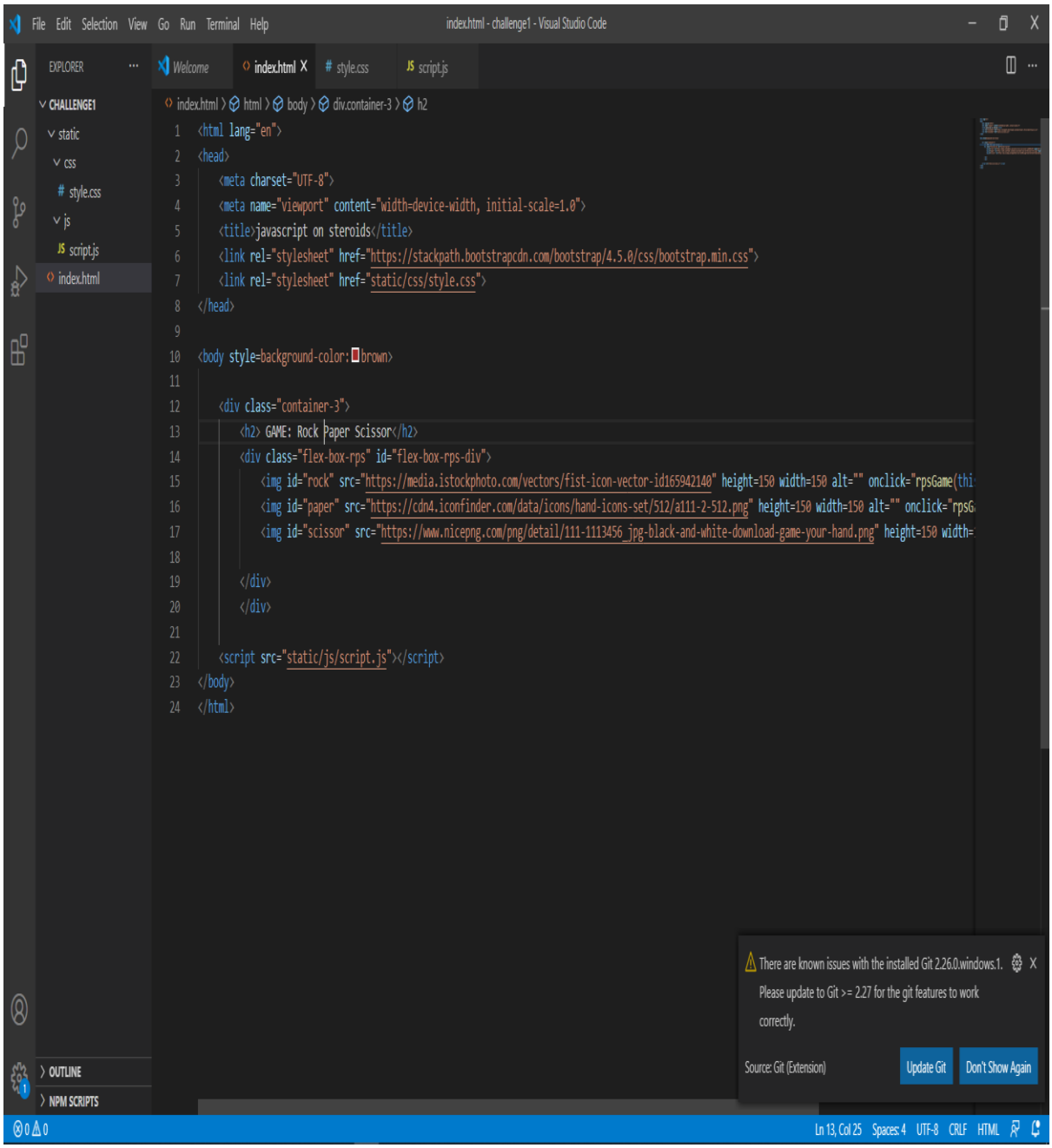

gmail log
Inbox (2,5
Post Atter
Inbox (1,4
iON Event
TCS NQT
Courses
Project Se
aryan200
Documen
javascript
File | C:/Users/Dell/Desktop/challenge1/index.html
GAME: Rock Paper Scissor
You Won!


gmail log
Inbox (2,5
Post Atter
Inbox (1,4
iON Event
TCS NQT
Courses
Project Se
aryan200
Documen
javascript
File | C:/Users/Dell/Desktop/challenge1/index.html
GAME: Rock Paper Scissor
Tied!


File | C:/Users/Dell/Desktop/challenge1/index.html
GAME: Rock Paper Scissor
Tied!


File | C:/Users/Dell/Desktop/challenge1/index.html
GAME: Rock Paper Scissor
You Won!

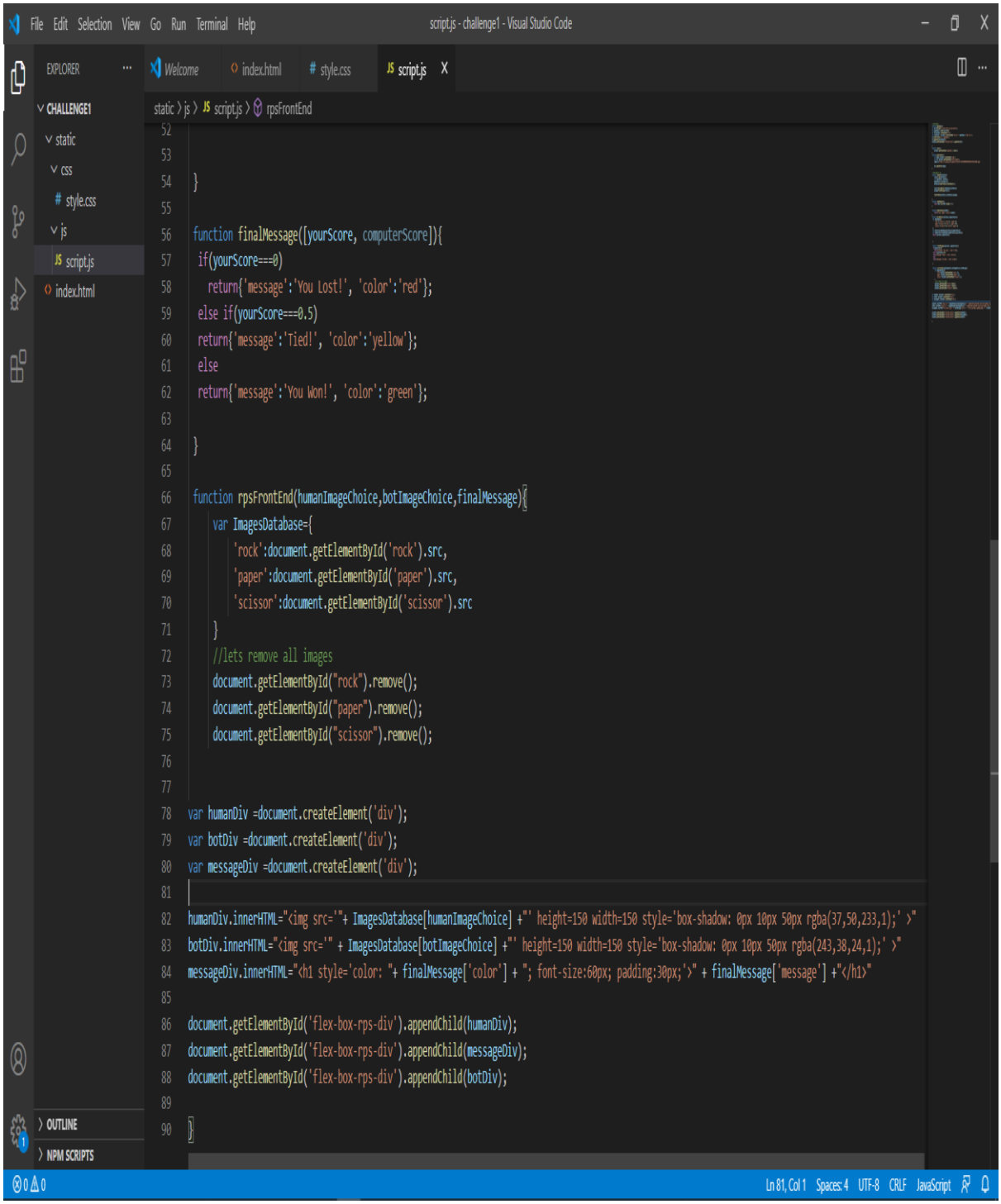
Code


File Edit Selection View Go Run Terminal Help
index.html - challenge1 - Visual Studio Code
EXPLORER
CHALLENGE1
static
css
style.css
js
script.js
index.html
Welcome
index.html X
style.css
script.js
index.html > html > body > div.container-3 > h2
<html lang="en">
<head>
<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<title>javascript on steroids</title>
<link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.5.0/css/bootstrap.min.css">
<link rel="stylesheet" href="static/css/style.css">
</head>
<body style=background-color:brown>

<h2> GAME: Rock Paper Scissor</h2>

<img id="rock" src="https://media.istockphoto.com/vectors/fist-icon-vector-id165942140" height=150 width=150 alt="" onclick="rpsGame(thi
<img id="paper" src="https://cdn4.iconfinder.com/data/icons/hand-icons-set/512/a111-2-512.png" height=150 width=150 alt="" onclick="rpsG
<img id="scissor" src="https://www.nicepng.com/png/detail/111-1113456_jpg-black-and-white-download-game-your-hand.png" height=150 width=

<script src="static/js/script.js"></script>
</body>
</html>
There are known issues with the installed Git 2.26.0.windows.1.
Please update to Git >= 2.27 for the git features to work correctly.
Source: Git (Extension)
Update Git
Don't Show Again
OUTLINE
NPM SCRIPTS
Ln 13, Col 25 Spaces: 4 UTF-8 CRLF HTML


File Edit Selection View Go Run Terminal Help
script.js - challenge1 - Visual Studio Code
EXPLORER
CHALLENGE1
static
css
style.css
js
script.js
index.html
Welcome
index.html
style.css
script.js
static > js > script.js > rpsFrontEnd
}
function finalMessage([yourScore, computerScore]){
if(yourScore===0)
return{'message':'You Lost!', 'color':'red'};
else if(yourScore===0.5)
return{'message':'Tied!', 'color':'yellow'};
else
return{'message':'You Won!', 'color':'green'};
}
function rpsFrontEnd(humanImageChoice,botImageChoice,finalMessage){
var ImagesDatabase={
'rock':document.getElementById('rock').src,
'paper':document.getElementById('paper').src,
'scissor':document.getElementById('scissor').src
}
//lets remove all images
document.getElementById("rock").remove();
document.getElementById("paper").remove();
document.getElementById("scissor").remove();
var humanDiv =document.createElement('div');
var botDiv =document.createElement('div');
var messageDiv =document.createElement('div');
humanDiv.innerHTML="<img src='"+ ImagesDatabase[humanImageChoice] +"' height=150 width=150 style='box-shadow: 0px 10px 50px rgba(37,50,233,1);' >"
botDiv.innerHTML="<img src='" + ImagesDatabase[botImageChoice] +"' height=150 width=150 style='box-shadow: 0px 10px 50px rgba(243,38,24,1);' >"
messageDiv.innerHTML="<h1 style='color: "+ finalMessage['color'] + "; font-size:60px; padding:30px;'>" + finalMessage['message'] +"</h1>"
document.getElementById('flex-box-rps-div').appendChild(humanDiv);
document.getElementById('flex-box-rps-div').appendChild(messageDiv);
document.getElementById('flex-box-rps-div').appendChild(botDiv);
}
OUTLINE
NPM SCRIPTS
Ln 81, Col 1 Spaces: 4 UTF-8 CRLF JavaScript

References:-

https://www.youtube.com/watch?v=Qqx_wzMmFeA

https://www.w3schools.com/js/DEFAULT.asp

https://en.wikipedia.org/wiki/JavaScript

“JavaScript: The Good Parts” by Douglas Crockford