Create – Applications From Ideas
Written Response Submission Template

Please see Assessment Overview and Performance Task Directions for Student for the task directions and recommended word counts.

Program Purpose and Development

2a)

Video narration

2b)

Before I had any code that allowed actually playing Tic-Tac-Toe, when I made progress in a function I opened the HTML page, used the browser's console to change any variables or the grid array needed to run that function and then ran it through the console. This was inefficient and annoying, but I soon realized I could create a temporary test() function that would run upon click by a button that would do all this for me; then, all I had to do in the console was examine the result of the function. Later, of course, I was able to remove this function and its button as enough code existed for me to test the program in the course of playing the game. A difficulty I encountered during development was in the winner(player) function, and my work-around is obvious in the boolean tests for the if statements within the function. When I first created this function, I used the boolean "grid[i] == [player, player, player]" to check for a horizontal row win. However, for some reason, the function failed ro recognize this as true even when the array was correct. I could not seem to overcome this problem until I decided to try testing for each item in the row individually with AND operators joining these: "grid[i][0] == player && grid[i][1] == player && grid[i][2] == player". I had this idea partly because testing for vertical wins worked properly, and this was what I used in that instance.

2c)

```
function easyGame() {
    if (gameType == 1 && currentPlayer == 'o' && allowed) {
        document.getElementById('message').innerHTML = "Thinking...";
        setTimeout( //timeout to create illusion of thinking
            function(){
                if (nextWin('o')) { //if AI will win
                    grid[winLocation[0]][winLocation[1]] = 'o';
                    document.getElementById(winLocation[0].toString() + winLocation[1].toString()).src = "o.png";
                } else if (nextWin('x')) { //block player from winning
                    grid[winLocation[0]][winLocation[1]] = 'o';
                    document.getElementById(winLocation[0].toString() + winLocation[1].toString()).src = "o.png";
                } else {
                    //if blocking or winning unnecessary, pick a random spot
                    var x, y;
                    var worked = false;
                    while (!worked) { //pick random spots until one works
                        x = Math.floor(Math.random() * 3);
                        y = Math.floor(Math.random() * 3);
                        if (grid[x][y] == "none") {
                            grid[x][y] = 'o';
                            document.getElementById(x.toString() + y.toString()).src = "o.png";
                            worked = true;
                        } else {
                            worked = false;
                        }
                    }
                }
                //switching turn
                currentPlayer = 'x';
                document.getElementById('message').innerHTML = "Your turn.";
                //check if AI won
                if(winner('o')) {
                    document.getElementById('message').innerHTML = "I win!<br /><button onclick='newGame()'>New Game</button>";
                    oWins++;
                    updateScoreboard()
                    allowed = false;
                } else if (winner('o') == undefined) {
                    document.getElementById('message').innerHTML = "Tie!<br /><button onclick='newGame()'>New Game</button>";
                    allowed = false;
                }
            }, 700
        )
    }
}
```

The function easyGame() is the algorithm that the AI uses on its turn in 1-player games. Assuming it is allowed, the AI uses two smaller algorithms in order to play its turn. First, it runs nextWin, an algorithm that checks whether it is possible for the given player to win on this turn. If the "O" player (the AI) can win, it will complete its row of three, and if the "X" player (the human user) can win, the AI will block them. If neither of these is possible, the AI uses another algorithm where it iterates through picking random spots on the board until one is available, and it finally plays its turn in this spot.

This function is necessary for the 1-player game to work at all. Without the algorithm nextWin(player), the AI would be too easy because it would ignore obvious chances to win or block the human the majority of the time, and without the use of random choices, the AI would not play any turns until it was about to lose.

2d)

```
function winner(player) { //function checks if the given player is the winner
    for (i in grid) {
        //checks rows
        if (grid[i][0] == player && grid[i][1] == player && grid[i][2] == player) {
            return true;
        }
        //checks columns
        if (grid[0][i] == player && grid[1][i] == player && grid[2][i] == player) {
            return true;
        }
    }
    //checks diagonals
    if (grid[0][0] == player && grid[1][1] == player && grid[2][2] == player) {
        return true;
    } else if (grid[0][2] == player && grid[1][1] == player && grid[2][0] == player) {
        return true;
    }
    // If player hasn't won..
    for (a in grid) { //if board isn't full, return false (game isn't over)
        if (grid[a].includes('none')) {
            return false;
        }
    }
    //undefined means tie
    return undefined;
}
```

This function is an abstraction that determines whether the given player has won, and returns a null value if the game has ended without the player winning. This abstraction was useful because checking whether a player has won the game on their turn required much more code than I originally anticipated; it was easier to simply call on the above "winner" program than to use a lengthy boolean using AND and OR operators.