```
1: #CSCI370 - Assignment 1
2: #Grant Haataja
3: .data
4: startMessage: .asciiz "\nStart a one-player tic-tac-toe game.\n"
5: pieceMessage: .asciiz "Pick a piece to start (X/O): "
6: pieceError: .asciiz "\nInvalid piece!  "
7: continueMessage: .asciiz "Continue? (y/n) "
8: continueError: .asciiz "\nInvalid character!   "
9: newGameMessage: .asciiz "\nNew game? (y/n) "
10: nextMove: .asciiz "\nEnter the next move (1-9): "
11: moveError: .asciiz "\nInvalid move!  "
12: X: .asciiz "X"
13: O: .asciiz "O"
14: x: .asciiz "x"
15: o: .asciiz "o"
16: board:
17: .ascii   "\n\n        | |        1|2|3\n       -----       -----"
18: .ascii     "\n        | |        4|5|6\n       -----       -----"
19: .asciiz    "\n        | |        7|8|9\n"
20: ogboard:
21: .ascii   "\n\n        | |        1|2|3\n       -----       -----"
22: .ascii     "\n        | |        4|5|6\n       -----       -----"
23: .asciiz    "\n        | |        7|8|9\n"
24:
25: .text
26: #program begins executing here: this is where the game starts
27: main:
28: #print starting message
29: li $v0, 4
30: la $a0, startMessage
31: syscall
32: #save X and O to temp registers
33: li $t0, 'X'
34: li $t1, 'O'
35: li $t2, 'x'
36: li $t3, 'o'
37:
38: userInput:
39: #tell the user to pick a piece to start
40: li $v0, 4
41: la $a0, pieceMessage
42: syscall
43: #get the user's choice from input
44: li $v0, 12
45: syscall
46: #save user's choice for reference later
47: move $s5, $v0
48: #check to see if the user's choice was valid
49: beq $v0, $t0, step1
50: beq $v0, $t1, step1
51: beq $v0, $t2, step1
```

```
52: beq $v0, $t3, step1
53: #if user's choice is wrong, no branching occurs and we print the appropriate error mes
sage
54: li $v0, 4
55: la $a0, pieceError
56: syscall
57: #jump back to the start of userInput routine to try again
58: j userInput
59:
60: #subroutine to print board
61: printBoard:
62: li $v0, 4
63: la $a0, board
64: syscall
65: jr $ra
66:
67: step1:
68: #jump and link to the printBoard subroutine to print out the unwritten board
69: jal printBoard
70:
71: step2:
72: #save moves 1-9 in temp registers for comparison
73: li $t0, '1'
74: li $t1, '2'
75: li $t2, '3'
76: li $t3, '4'
77: li $t4, '5'
78: li $t5, '6'
79: li $t6, '7'
80: li $t7, '8'
81: li $t8, '9'
82: #prompt the user to choose a move
83: li $v0, 4
84: la $a0, nextMove
85: syscall
86: #get user's move
87: li $v0, 12
88: syscall
89: #check if user entered a valid move
90: beq $v0, $t0, offset
91: beq $v0, $t1, offset
92: beq $v0, $t2, offset
93: beq $v0, $t3, offset
94: beq $v0, $t4, offset
95: beq $v0, $t5, offset
96: beq $v0, $t6, offset
97: beq $v0, $t7, offset
98: beq $v0, $t8, offset
99: #if user's choice is wrong, no branching occurs and we print the appropriate error mes
sage
100: li $v0, 4
```

```
101: la $a0, moveError
102: syscall
103: #jump back to the start of step2 routine to try again
104: j step2
105:
106: #subroutine to find offset and modify board string
107: offset:
108: #correct for using char data types
109: subi $s0, $v0, 48
110: #calculate offset to determine where to save user moves
111: subi $t0, $s0, 1
112: div $t1, $t0, 3
113: mul $t2, $t1, 44
114: addi $t3, $t2, 7
115: mul $t4, $s0, 2
116: add $s1, $t3, $t4
117: #now check if an X or O
118: beq $s5, 'X', writeX
119: beq $s5, 'x', writeX
120: beq $s5, 'O', writeO
121: beq $s5, 'o', writeO
122:
123: #subroutine if the user enters a move that has already been taken
124: wrongChoice:
125: li $v0, 4
126: la $a0, moveError
127: syscall
128: #jump back to the start of step2 routine to try again
129: j step2
130:
131: writeX:
132: lb $t0, board($s1)
133: li $t1, 'X'
134: li $t2, 'O'
135: beq $t0, $t2, wrongChoice
136: beq $t0, $t1, wrongChoice
137: #now commit the user's move to the board string
138: li $t1, 'X'
139: sb $t1, board($s1)
140: li $s5, 'O'
141: j step3
142:
143: writeO:
144: lb $t0, board($s1)
145: li $t1, 'X'
146: li $t2, 'O'
147: beq $t0, $t2, wrongChoice
148: beq $t0, $t1, wrongChoice
149: #now commit the user's move to the board string
150: li $t1, 'O'
151: sb $t1, board($s1)
```

```
152: li $s5, 'X'
153:
154: step3:
155: #jump and link to subroutine to print the modified board
156: jal printBoard
157:
158: continue:
159: #ask the user if they wish to continue
160: li $v0, 4
161: la $a0, continueMessage
162: syscall
163: #get user's choice
164: li $v0, 12
165: syscall
166: beq $v0, 'n' newGame
167: beq $v0, 'N' newGame
168: beq $v0, 'y' step2
169: beq $v0, 'Y' step2
170: #if the user chooses anything else, no branching occurs and we print the error messag
e
171: li $v0, 4
172: la $a0, continueError
173: syscall
174: j continue
175:
176: newGame:
177: #ask the user if they wish to start a new game or exit
178: li $v0, 4
179: la $a0, newGameMessage
180: syscall
181: #get user's choice for starting a new game or not
182: li $v0, 12
183: syscall
184: beq $v0, 'n' exit
185: beq $v0, 'N' exit
186: beq $v0, 'y' cleanUp
187: beq $v0, 'Y' cleanUp
188: #if the user chooses anything else, no branching occurs and we print the error messag
e
189: li $v0, 4
190: la $a0, continueError
191: syscall
192: j newGame
193:
194: #reset the board and jump back to main to start a new game
195: cleanUp:
196: #load addresses of written and unwritten boards, as well as the length of board strin
g
197: la $a0, board
198: la $a1, ogboard
199: la $a2, 128
```

```
200: #store address of written board in temp register
201: move $t0, $a0
202: #store address of unwritten board in temp register
203: move $t1, $a1
204: #store address of the byte just past the end of board string
205: add $t2, $a0, $a2
206:
207: copy:
208: #stop if we reach the end of the board string
209: beq $t0, $t2, finish
210: #otherwise continue copying byte by byte
211: lbu $t3, 0($t1)
212: sb $t3, 0($t0)
213: addi $t0, $t0, 1
214: addi $t1, $t1, 1
215: #stop if we reach a null terminator
216: beq $t3, $zero, main
217: #otherwise loop back up and continue copying
218: j copy
219:
220: finish:
221: addi $t2, $t2, -1
222: sb $zero, 0($t2)
223: #jump to main to begin a fresh game
224: j main
225:
226: #exit the program
227: exit:
228: li $v0, 10
229: syscall
```