```
1: #Derek Trom
2: #Program Excercise 2
3:
4: .data
5:     p00:     .asciiz    "\nStart Playing A Tic-Tac-Toe Game\n"
6:     p0:      .asciiz    "\nContinue? (Y/N): "
7:     p1:      .asciiz    "\nChoose X or O to start: "
8:     p3:      .asciiz    "\nPlay again? (Y/N): "
9:     p4:      .asciiz    "Hit the spacebar to start the System's next move.\n"
10:     p5:     .asciiz    "My Move is: "
11:     xWins:  .asciiz    "You (X) win the match"
12:     oWins:      .asciiz    "You (O) win the match"
13:     compXWins:  .asciiz    "I (the System X) win!"
14:     compOWins:  .asciiz    "I (the System O) win!"
15:     Prompt: .asciiz    "\nChoose a number 1-9 to play: "
16:     wrong:  .asciiz    "\nIncorrect input try again...\n"
17:     wrongPiece: .asciiz    "\nInvalid piece...\n"
18:     spotTaken:  .asciiz    "\nSpot taken...\n"
19:     computerX:  .asciiz    "I (the System) am first and I pick X\n"
20:     computerO:  .asciiz    "I (the System) am first and I pick O\n"
21:     drawGame:   .asciiz    "It's a Draw!"
22:     board:      .ascii     "\n\n        | |        1|2|3\n       -----       -----"
23:                 .ascii     "\n        | |        4|5|6\n       -----       -----"
24:                 .asciiz    "\n        | |        7|8|9\n"
25:
26:     boardArray: .byte      0,0,0,0,0,0,0,0,0 # used for moves made and checking win
27:     playerTurn: .byte      0  # keeps track of if x or o turn
28:     comTurn:    .byte      0  #if computer turn or not
29:     counter:    .word      0  #game counter to keep track of how many moves made
30:     str1:       .space     2  #space for int input
31:     winCounter: .word      0 #count num wins
32:
33: .text
34: ############## MAIN START ####################
35:     main:
36:         la   $a0, p00 #welcome message
37:         li   $v0, 4   #print load
38:         syscall       #call print
39:
40:
41:         # choose user or system move first using randint function
42:         li $a1,2   #load 2
43:         xor  $a0,$a0,$a0     # get seed number
44:         li   $v0,42    #random number generator
45:         syscall
46:         beq  $a0,$zero,computerChooseXO #if zero computer starts
47: #
48: ############# PLAYER STARTS #################
49: #
50:     startingLoop:
51:         la   $a0, p1      # load player message
52:             li   $v0, 4
53:             syscall           # print player msg
54:
55:         # enter x or o
```

```
56:             li $v0,12 #syscall for byte
57:             syscall
58:
59:             #check for valid x or o
60:             beq $v0,'X',xStarts
61:             beq $v0,'x',xStarts
62:             beq $v0,'O',oStarts
63:             beq $v0,'o',oStarts
64:
65:
66:             la   $a0, wrongPiece  #catch if not x or o
67:             li   $v0, 4
68:             syscall               # print error message
69:             j startingLoop #return to top of the loop
70:
71:
72:     # set tutn to X
73:     xStarts:
74:             li   $t0,'X'
75:             sb   $t0,playerTurn($zero)
76:             li   $t0,'O'
77:             sb   $t0,comTurn($zero)
78:             j play #jump to play game
79:
80:     # set turn to O
81:     oStarts:
82:             li   $t0,'O'
83:             sb   $t0,playerTurn($zero)
84:             li   $t0,'X'
85:             sb   $t0,comTurn($zero)
86:             j  play #jump to play game
87: #
88: ################ PLAY TIC-TAC-TOE #################
89: #
90:     play:
91:
92:             lb   $t0,playerTurn($zero)   # whose turn
93:
94:         # print board
95:             la   $a0, board
96:             li   $v0, 4
97:             syscall
98:             lb   $t0,playerTurn($zero)   # whose turn
99:             lb   $t1,comTurn($zero)
100:            beq  $t0,$t1,systemturn
101:            jal loadConstants
102:            b   storemove
103:
104: #
105: ############# COMPUTER STARTS ###################
106: #
107:    computerChooseXO:
108:                li $a1,2  #number of choices
109:                xor  $a0,$a0,$a0     # generate seed
110:                li   $v0,42  #random number generator
```

```
111:                syscall
112:                beq  $a0,$zero,computerIsX #if 0 computer will be x
113:                li   $t0,'O'    #system will be o else
114:                sb   $t0,comTurn($zero) #load 0 for computer turn
115:                li   $t0,'O'
116:                sb   $t0,playerTurn($zero)
117:                la   $a0, computerO   #print message that computer is 0
118:                li   $v0, 4           # print
119:                syscall
120:                j play
121:    computerIsX:
122:                la   $a0, computerX       # print message computer is x
123:                li   $v0, 4           # print syscal
124:                syscall
125:                li   $t0,'X'
126:                sb   $t0,comTurn($zero)
127:                li   $t0,'X'
128:                sb   $t0,playerTurn($zero)
129:                j play #start playing game
130:
131:    # get computer turn
132:    systemturn:
133:        #ask user to press space for next move
134:            la   $a0, p4       # p4 load
135:            li   $v0, 4           # syscall 4
136:            syscall               # print
137:            #receive input
138:            li   $v0, 12          # specify read string
139:            syscall
140:            li $t9, 32            #load 32 to t9
141:            bne $t9, $v0, notSpace #if not space entered then error
142:            jal computer         # system turn
143:            move $t0,$v0
144:
145:            la   $a0, p5       # p5 message print My move is:
146:            li   $v0, 4           # syscall 4
147:            syscall               # print
148:
149:            move   $a0, $t0       #move computer move to $a0
150:            addi   $a0,$a0,1      #add one to account for array indexing
151:            li   $v0, 1           # print int
152:            syscall
153:
154:            move $v0,$t0      #keep track of turn
155:
156:
157: #
158: ################## STORE PLAYER MOVE ######################
159: #
160:    storemove:
161:            lb  $a0,playerTurn($zero)
162:            sb  $a0,boardArray($v0)      # store move
163:        addi $v0, $v0, 1 #add one to match board place in $v0
164:        # place player
165:            jal offsetAndPlace #place piece on board
```

```
166:
167:        # check for winner
168:            jal   checkWin
169:            beq   $v0,1,winner #if 1 in $v0 its a win
170:
171:        # check for draw
172:            lw    $t0,counter($zero) #load count
173:            addi  $t0,$t0,1 #add 1
174:            sw    $t0,counter($zero) #load to counter
175:            beq  $t0,9,draw #if 9 moves made its a draw
176:        # switch turn
177:            lb    $t0,playerTurn($zero)
178:            beq   $t0,'X',oTurn
179:            li    $t0,'X'
180:            sb    $t0,playerTurn($zero)
181:                jal continue #ask to continue
182:            j     play # continue game
183:    # o's turn
184:    oTurn:
185:            li    $t0,'O'
186:            sb    $t0,playerTurn($zero)
187:            j play     # continue game
188: #
189: ############### CONTINUE? Y/N #################
190: #
191: #ask to continue
192:    continue:
193:        # print board
194:                la   $a0, board
195:                li   $v0, 4
196:                syscall
197:        la   $a0, p0      # load player message
198:            li   $v0, 4
199:            syscall           # print player msg
200:
201:        # enter x or o
202:            li $v0,12 #syscall for byte
203:            syscall
204:
205:            #check for valid y or n
206:            beq $v0,'Y',playYes
207:            beq $v0,'y',playYes
208:            beq $v0,'N',playAgain
209:            beq $v0,'n',playAgain
210:
211:
212:            la   $a0, wrong  #catch if not x or o
213:            li   $v0, 4
214:            syscall               # print error message
215:            j continue #return to top of the loop
216:        playYes:
217:            jr $ra
218:
219: #
220: ####################### DRAW GAME ######################
```

```
221: #
222: #ITS A DRAW
223:    draw:
224:    # print board
225:            la   $a0, board       # first argument for print (array)
226:            li   $v0, 4           # specify Print String service
227:            syscall               # print message
228:
229:            la   $a0, drawGame        # point to drawgame message
230:            li   $v0, 4           # specify Print String service
231:            syscall               # print msg
232:            j playAgain
233: #
234: ################ WINNING GAME #################
235: #
236: #SOMEBODY WON
237:    winner:
238:
239:        # print board
240:            la   $a0, board       # first argument for print (array)
241:            li   $v0, 4           # specify Print String service
242:            syscall               # print message
243:            lb   $s6, playerTurn($zero) #load playerTurn
244:            lb   $s5,comTurn($zero) #load com turn
245:            beq  $s5,$s6,compWins #if they are the same it is computer turn
246:            j    playerWins #else player won
247:         playerWins:
248:        # go to either player won x or o
249:            beq $s6,'X',xwins
250:            beq $s6,'O',owins
251:        compWins:
252:        #go to either computer won x or o
253:            beq $s5,'X',computerXwins     # go to winner
254:            beq $s5,'O',computerOwins
255:
256: #
257: ################# X WINS ###################
258: #
259: #
260:    computerXwins:
261:        #print computer won as x
262:            la   $a0, compXWins        # load xWins message
263:            li   $v0, 4           # syscall 4 to print
264:            syscall               # print
265:            j playAgain      #jump to play again questions
266:    xwins:
267:        #print player won as x
268:            la   $a0, xWins        # load xWins message
269:            li   $v0, 4           # syscall 4 to print
270:            syscall               # print
271:            j playAgain      #jump to play again questions
272:
273: #
274: ################# O WINS ###################
275: #
```

```
276:    computerOwins:
277:        #print computer won as O
278:            la   $a0, compOWins        # load xWins message
279:            li   $v0, 4           # syscall 4 to print
280:            syscall               # print
281:            j playAgain      #jump to play again questions
282:    owins:
283:        #print player won as o
284:            la   $a0, oWins        # load oWins message
285:            li   $v0, 4           # syscall 4 to print
286:            syscall               # print
287:            j playAgain       #jump to play again questions
288:
289: #
290: ################## CHECK FOR WIN ###################
291: #
292: #maybe some redundant code in here but I tried to make it smarter
293:    checkWin:
294:        #check for wins
295:            subu  $sp, $sp, 4     # Decrement the $sp to make space for $ra.
296:            sw    $ra, ($sp)      # Push the return address, $ra to stack
297:
298:            subu  $sp, $sp, 4     # Decrement the $sp to make space for $t1.
299:            sw    $t1, ($sp)      # Push the return address, $t1 to stack
300:
301:            subu  $sp, $sp, 4     # Decrement the $sp to make space for $t2.
302:            sw    $t2, ($sp)      # Push the return address, $t2 to stack
303:
304:            subu  $sp, $sp, 4     # Decrement the $sp to make space for $t3.
305:            sw    $t3, ($sp)      # Push the return address, $t3 to stack
306:
307:            subu  $sp, $sp, 4     # Decrement the $sp to make space for $t4.
308:            sw    $t4, ($sp)      # Push the return address, $t4 to stack
309:
310:            subu  $sp, $sp, 4     # Decrement the $sp to make space for $t5.
311:            sw    $t5, ($sp)      # Push the return address, $t5 to stack
312:
313:            subu  $sp, $sp, 4     # Decrement the $sp to make space for $t6.
314:            sw    $t6, ($sp)      # Push the return address, $t6 to stack
315:
316:            subu  $sp, $sp, 4     # Decrement the $sp to make space for $t7.
317:            sw    $t7, ($sp)      # Push the return address, $t7 to stack
318:
319:            subu  $sp, $sp, 4     # Decrement the $sp to make space for $t8.
320:            sw    $t8, ($sp)      # Push the return address, $t8 to stack
321:
322:            subu  $sp, $sp, 4     # Decrement the $sp to make space for $t9.
323:            sw    $t9, ($sp)      # Push the return address, $t9 to stack
324:
325:
326:            li   $v0,1              # assume turn wins
327:
328:            # get moves in registers to check for winning combinations
329:            lb  $t1,boardArray($zero) #load move array[0]into $t1
330:            lb  $t2,boardArray+1($zero)#load move array[1]into $t2
```

```
331:            lb  $t3,boardArray+2($zero)#load move array[2]into $t3
332:            lb  $t4,boardArray+3($zero)#load move array[3]into $t4
333:            lb  $t5,boardArray+4($zero)#load move array[4]into $t5
334:            lb  $t6,boardArray+5($zero)#load move array[5]into $t6
335:            lb  $t7,boardArray+6($zero)#load move array[6]into $t7
336:            lb  $t8,boardArray+7($zero)#load move array[7]into $t8
337:            lb  $t9,boardArray+8($zero)#load move array[8]into $t9
338:
339:         #rows check
340:
341:    topRow: #win 1,2,3
342:            bne $a0,$t1,topRow2  #if x(88)/o(79) not same
343:            bne $a0,$t3,topRow2  #if x(88)/o(79) not same
344:            bne $a0,$t2,topRow2  #if x(88)/o(79) not same
345:            j popBackToCaller #else it is a win
346:        topRow2: #win 1,2,3
347:            bne $a0,$t1,topRow3  #if x(88)/o(79) not same
348:            bne $a0,$t2,topRow3  #if x(88)/o(79) not same
349:            bne $a0,$t3,topRow3   #if x(88)/o(79) not same
350:            j popBackToCaller #else it is a win
351:        topRow3: #win 1,2,3
352:            bne $a0,$t3,middleRow  #if x(88)/o(79) not same
353:            bne $a0,$t2,middleRow  #if x(88)/o(79) not same
354:            bne $a0,$t1,middleRow   #if x(88)/o(79) not same
355:            j popBackToCaller #else it is a win
356:    middleRow: #win 4,5,6
357:            bne $a0,$t4,middleRow2  #if x(88)/o(79) not same
358:            bne $a0,$t6,middleRow2  #if x(88)/o(79) not same
359:            bne $a0,$t5,middleRow2  #if x(88)/o(79) not same
360:            j popBackToCaller #else it is a win
361:        middleRow2: #win 4,5,6
362:            bne $a0,$t4,middleRow3  #if x(88)/o(79) not same
363:            bne $a0,$t5,middleRow3  #if x(88)/o(79) not same
364:            bne $a0,$t6,middleRow3  #if x(88)/o(79) not same
365:            b popBackToCaller #else it is a win
366:        middleRow3: #win 4,5,6
367:            bne $a0,$t6,bottomRow  #if x(88)/o(79) not same
368:            bne $a0,$t5,bottomRow  #if x(88)/o(79) not same
369:            bne $a0,$t4,bottomRow  #if x(88)/o(79) not same
370:            j popBackToCaller #else it is a win
371:    bottomRow: #win 7,8,9
372:            bne $a0,$t7,bottomRow2  #if x(88)/o(79) not same
373:            bne $a0,$t8,bottomRow2  #if x(88)/o(79) not same
374:            bne $a0,$t9,bottomRow2  #if x(88)/o(79) not same
375:            j popBackToCaller #else it is a win
376:        bottomRow2: #win 7,8,9
377:            bne $a0,$t7,bottomRow3  #if x(88)/o(79) not same
378:            bne $a0,$t9,bottomRow3  #if x(88)/o(79) not same
379:            bne $a0,$t8,bottomRow3  #if x(88)/o(79) not same
380:            j popBackToCaller #else it is a win
381:        bottomRow3: #win 7,8,9
382:            bne $a0,$t9,leftColumn  #if x(88)/o(79) not same
383:            bne $a0,$t8,leftColumn  #if x(88)/o(79) not same
384:            bne $a0,$t7,leftColumn  #if x(88)/o(79) not same
385:            j popBackToCaller #else it is a win
```

```
386:    # columns check
387:    leftColumn: #win 1,4,7
388:            bne $a0,$t1,leftColumn2  #if x(88)/o(79) not same
389:            bne $a0,$t4,leftColumn2  #if x(88)/o(79) not same
390:            bne $a0,$t7,leftColumn2  #if x(88)/o(79) not same
391:            j popBackToCaller #else it is a win
392:        leftColumn2: #win 1,4,7
393:            bne $a0,$t1,leftColumn3  #if x(88)/o(79) not same
394:            bne $a0,$t7,leftColumn3  #if x(88)/o(79) not same
395:            bne $a0,$t4,leftColumn3  #if x(88)/o(79) not same
396:            j popBackToCaller #else it is a win
397:        leftColumn3: #win 1,4,7
398:            bne $a0,$t7,middleColumn  #if x(88)/o(79) not same
399:            bne $a0,$t4,middleColumn  #if x(88)/o(79) not same
400:            bne $a0,$t1,middleColumn  #if x(88)/o(79) not same
401:            j popBackToCaller #else it is a win
402:    middleColumn:#win 2,5,8
403:            bne $a0,$t2,middleColumn2  #if x(88)/o(79) not same
404:            bne $a0,$t5,middleColumn2  #if x(88)/o(79) not same
405:            bne $a0,$t8,middleColumn2  #if x(88)/o(79) not same
406:            j popBackToCaller #else it is a win
407:        middleColumn2:#win 2,5,8
408:            bne $a0,$t2,middleColumn3  #if x(88)/o(79) not same
409:            bne $a0,$t8,middleColumn3  #if x(88)/o(79) not same
410:            bne $a0,$t5,middleColumn3  #if x(88)/o(79) not same
411:            j popBackToCaller #else it is a win
412:        middleColumn3:#win 2,5,8
413:            bne $a0,$t8,rightColumn  #if x(88)/o(79) not same
414:            bne $a0,$t5,rightColumn  #if x(88)/o(79) not same
415:            bne $a0,$t2,rightColumn  #if x(88)/o(79) not same
416:            j popBackToCaller #else it is a win
417:    rightColumn: #win 3,6,9
418:            bne $a0,$t3,rightColumn2  #if x(88)/o(79) not same
419:            bne $a0,$t6,rightColumn2  #if x(88)/o(79) not same
420:            bne $a0,$t9,rightColumn2  #if x(88)/o(79) not same
421:            j popBackToCaller #else it is a win
422:        rightColumn2: #win 3,6,9
423:            bne $a0,$t3,rightColumn3  #if x(88)/o(79) not same
424:            bne $a0,$t9,rightColumn3  #if x(88)/o(79) not same
425:            bne $a0,$t6,rightColumn3  #if x(88)/o(79) not same
426:            j popBackToCaller #else it is a win
427:        rightColumn3: #win 3,6,9
428:            bne $a0,$t9,diagonal1  #if x(88)/o(79) not same
429:            bne $a0,$t6,diagonal1  #if x(88)/o(79) not same
430:            bne $a0,$t3,diagonal1  #if x(88)/o(79) not same
431:            j popBackToCaller #else it is a win
432:
433:    # diagonals
434:    diagonal1: #win 1,5,9
435:            bne $a0,$t1,diagonal12 #if x(88)/o(79) not same
436:            bne $a0,$t5,diagonal12  #if x(88)/o(79) not same
437:            bne $a0,$t9,diagonal12  #if x(88)/o(79) not same
438:            j popBackToCaller
439:        diagonal12: #win 1,5,9
440:            bne $a0,$t1,diagonal123  #if x(88)/o(79) not same
```

```
441:            bne $a0,$t9,diagonal123  #if x(88)/o(79) not same
442:            bne $a0,$t5,diagonal123  #if x(88)/o(79) not same
443:            j popBackToCaller #else a win
444:        diagonal123: #win 1,5,9
445:            bne $a0,$t9,diagonal2  #if x(88)/o(79) not same
446:            bne $a0,$t5,diagonal2  #if x(88)/o(79) not same
447:            bne $a0,$t1,diagonal2  #if x(88)/o(79) not same
448:            j popBackToCaller #else a win
449:
450:    diagonal2: #win 3,5,7
451:            bne $a0,$t3,diagonal22  #if x(88)/o(79) not same
452:            bne $a0,$t5,diagonal22  #if x(88)/o(79) not same
453:            bne $a0,$t7,diagonal22  #if x(88)/o(79) not same
454:            j popBackToCaller #else it is a win
455:        diagonal22: #win 3,5,7
456:            bne $a0,$t3,diagonal223  #if x(88)/o(79) not same
457:            bne $a0,$t7,diagonal223  #if x(88)/o(79) not same
458:            bne $a0,$t5,diagonal223  #if x(88)/o(79) not same
459:            j popBackToCaller #else it is a win
460:        diagonal223: #win 3,5,7
461:            bne $a0,$t7,notWin  #if x(88)/o(79) not same
462:            bne $a0,$t5,notWin  #if x(88)/o(79) not same
463:            bne $a0,$t3,notWin  #if x(88)/o(79) not same
464:            j popBackToCaller #else it is a win
465:
466:    # no winner yet
467:    notWin:
468:            li  $v0, 0 #set $v0 to 0
469:
470:    popBackToCaller:
471:
472:            lw    $t9, ($sp)       # Pop the return address, $t9.
473:            addu  $sp, $sp, 4      # add unsigned 4 to $sp
474:
475:            lw    $t8, ($sp)       # Pop the return address, $t8.
476:            addu  $sp, $sp, 4      # add unsigned 4 to $sp
477:
478:            lw    $t7, ($sp)       # Pop the return address, $t7.
479:            addu  $sp, $sp, 4      # add unsigned 4 to $sp
480:
481:            lw    $t6, ($sp)       # Pop the return address, $t6.
482:            addu  $sp, $sp, 4      # add unsigned 4 to $sp
483:
484:            lw    $t5, ($sp)       # Pop the return address, $t5.
485:            addu  $sp, $sp, 4      # add unsigned 4 to $sp
486:
487:            lw    $t4, ($sp)       # Pop the return address, $t4.
488:            addu  $sp, $sp, 4      # add unsigned 4 to $sp
489:
490:            lw    $t3, ($sp)       # Pop the return address, $t3.
491:            addu  $sp, $sp, 4      # add unsigned 4 to $sp
492:
493:            lw    $t2, ($sp)       # Pop the return address, $t2.
494:            addu  $sp, $sp, 4      # add unsigned 4 to $sp
495:
```

```
496:            lw    $t1, ($sp)       # Pop the return address, $t1.
497:            addu  $sp, $sp, 4      # add unsigned 4 to $sp
498:
499:            lw    $ra, ($sp)       # Pop the return address, $ra.
500:            addu  $sp, $sp, 4      # add unsigned 4 to $sp
501:
502:            jr    $ra #return to caller
503: #
504: ################# PLAY AGAIN? Y/N ###################
505: #
506:    playAgain:
507:            la    $a0, p3        # point to p3
508:            li    $v0, 4         #load syscall 4
509:            syscall
510:
511:            # enter y or n
512:            li    $v0,12
513:            syscall
514:
515:        # validate entry
516:            beq    $v0,'Y',newgame
517:            beq    $v0,'y',newgame
518:            beq    $v0,'N',exit
519:            beq    $v0,'n',exit
520:            la     $a0, wrong    # catch wrong entry message
521:            li     $v0, 4
522:            syscall
523:            j playAgain #jump to top of play again
524:
525: #
526: ################# NEW GAME RESETS ###################
527: #
528:    newgame:
529:            #reset the board and moves array
530:            #replace all move spaces with blank in the board
531:                li     $s0, ' '
532:            li     $s2, 9
533:            sb     $s0, board($s2)
534:            li     $s2, 11
535:            sb     $s0, board($s2)
536:            li     $s2, 13
537:            sb     $s0, board($s2)
538:            li     $s2, 59
539:            sb     $s0, board($s2)
540:            li     $s2, 61
541:            sb     $s0, board($s2)
542:            li     $s2, 63
543:            sb     $s0, board($s2)
544:            li     $s2, 109
545:            sb     $s0, board($s2)
546:            li     $s2, 111
547:            sb     $s0, board($s2)
548:            li     $s2, 113
549:            sb     $s0, board($s2)
550:            # clear moves in board array
```

```
551:            sb     $zero,boardArray($zero)
552:            sb     $zero,boardArray+1($zero)
553:            sb     $zero,boardArray+2($zero)
554:            sb     $zero,boardArray+3($zero)
555:            sb     $zero,boardArray+4($zero)
556:            sb     $zero,boardArray+5($zero)
557:            sb     $zero,boardArray+6($zero)
558:            sb     $zero,boardArray+7($zero)
559:            sb     $zero,boardArray+8($zero)
560:
561:            # clear counter to 0
562:            sw     $zero,counter($zero)
563:
564:            j      main #back to main to restart
565:
566: #
567: ################# GET PLAYER MOVE ##################
568: #
569: #Players move functions and blocks
570:    loadConstants:
571:        li    $t1,'X'
572:            li    $t2,'O'
573:            li    $t8, 49 #used to test if the number is less than 9
574:            li    $s1, 57
575:    getMove:
576:        lb   $t0,playerTurn($zero)    # get turn
577:
578:            la   $a0, Prompt        # first argument for print (array)
579:            li   $v0, 4           # specify Print String service
580:            syscall               # print message
581:
582:            #get integer input
583:        li    $v0, 8  #receive input
584:            la    $a0, str1 #store in str1
585:            li    $a1, 2 #allocate space for input
586:            move  $s7, $a0  #move response to $t7
587:            syscall
588:        lb    $s7, 0($s7)
589:            bgt   $s7, $s1, errorInt  #catch if > 9
590:            blt   $s7, $t8, errorInt  #catch if < 1
591:            subi  $v0, $s7, 48
592:            subi  $v0,$v0,1 # decrement to match moves array
593:            lb    $t5,boardArray($v0)
594:            bne   $t5,$zero,takenSpot #else move cant be used
595:            jr    $ra #return to caller
596:
597: #
598: ####################ERROR BLOCKS##################
599: #
600: # NUMBER, SPACE, AND OTHER ERROR
601:    errorInt:
602:            la   $a0, wrong     # first argument for print (array)
603:            li   $v0, 4           # specify Print String service
604:            syscall               # print message
605:            j loadConstants
```

```
606:         notSpace:
607:            la   $a0, wrong     # first argument for print (array)
608:            li   $v0, 4           # specify Print String service
609:            syscall               # print message
610:            j systemturn
611:
612:    takenSpot:
613:            la   $a0, spotTaken      # first argument for print (array)
614:            li   $v0, 4           # specify Print String service
615:            syscall               # print message
616:            j loadConstants
617:
618: #
619: ###################### COMPUTER MOVE ##########################################
620: #
621: #Computer functions and blocks
622: #Win first, block second, one-step ahead, and random
623:    foundPossibleWin:
624:        j computeReturnAddress
625:    computer:
626:
627:            subu  $sp, $sp, 4     # Decrement the $sp to make space for $ra.
628:            sw    $ra, ($sp)      # Push the return address, $ra.
629:            li    $t1,0              # start from move 0
630:    findWin:
631:                lb    $t2,boardArray($t1)     # check if move open
632:                bne   $t2,$zero,notWinner     # move is open if move == 0
633:                lb    $a0,playerTurn($zero) #simulate player turn
634:                sb    $a0,boardArray($t1) #store the players turn in board
635:                jal   checkWin
636:                 # check if win for player
637:                sb    $zero,boardArray($t1)   # store a zero back in simulated move
638:                beq   $v0,0,notWinner #if $v0 contains 0 that move is not a win
639:                move  $v0,$t1            # winner found
640:                j     computeReturnAddress
641:    notWinner:
642:            # find a blocking move
643:            addu  $t1,$t1,1
644:            blt   $t1,10,findWin #if moves not exhausted go back to top
645:            lb    $a0,playerTurn($zero) #load what piece turn it is
646:            beq   $a0,'X',pieceX #if x switch
647:            li    $a0,'X' #else load x to a0
648:            j     notPieceX #o piece
649:    pieceX:
650:            li    $a0,'O' #load O piece to a0
651:
652:    notPieceX:
653:
654:            li    $t1,0  #restart at 0 to find a blocking move
655:
656:    findBlock:
657:
658:        lb    $t2,boardArray($t1)  # load move to $t2
659:                bne   $t2,$zero,notBlock #branch if != 0 spot taken
660:                sb    $a0,boardArray($t1) #store x or o in $a0
```

```
661:                jal   checkWin #check if it is a win
662:                sb    $zero,boardArray($t1) #if returns as not a win store 0 in move
663:                beq   $v0,0,notBlock  #if is is 0 no blocking move
664:                move  $v0,$t1 #move to $v0
665:                j     computeReturnAddress
666:    notBlock:
667:                addu  $t1,$t1,1 #increment counter
668:                blt   $t1,10,findBlock
669:                li    $t1, 0 #reset to do one step look ahead
670:         findOneAhead:
671:                lb    $t2,boardArray($t1)     # check if move open
672:                bne   $t2,$zero,notOneAhead     # move is open if move == 0
673:                lb    $a0,playerTurn($zero) #simulate player turn
674:                sb    $a0,boardArray($t1) #store the players turn in board
675:                li    $s3, 0 #initialize second counter for lookahead
676:                b     secondMark ##start second mark
677:
678:    secondMark:
679:        #check for a second mark as win
680:        lb    $t2,boardArray($s3)     # check if move open
681:                bne   $t2,$zero,notSecondMark #taken
682:                jal   checkWin #check for win
683:                sb    $zero,boardArray($t1)   # store a zero back in simulated move
684:                beq   $v0,0,notSecondMark #if $v0 contains 0 that move is not a win
685:
686:                b     foundPossibleWin # winner found leave in board and return to cal
ler
687:
688:
689:            notSecondMark:
690:                addu  $s3,$s3,1
691:            blt   $s3,10,secondMark #if moves not exhausted go back to top
692:            j notOneAhead
693:
694:    notOneAhead:
695:            # find a blocking move
696:            addu  $t1,$t1,1
697:            blt   $t1,10,findOneAhead #if moves not exhausted go back to top
698:
699:        # pick a random move
700:                li    $t1,9 #load upper bound
701:                lb    $t0,counter($zero)  # calculate n
702:                sub  $a1,$t1,$t0 #subract t0 and t1
703:                xor  $a0,$a0,$a0     # get random number 0 to n
704:                li   $v0,42 #syscal for random number
705:                syscall
706:
707:                li    $t1,0             # count down random number
708:                move  $t0,$a0           # get random number
709:    # count down random number
710:    randomMove:
711:            lb    $t2,boardArray($t1)    #load array[$t1] to $t2
712:            bne   $t2,$zero,randomTaken    #spot already taken
713:            move  $v0,$t1    #move t1 to v0
714:            beq   $t0,$zero,computeReturnAddress #spot not taken
```

```
715:            subi  $t0,$t0,1 #subtract one from $t0
716:    randomTaken:
717:            addi  $t1,$t1,1 #add 1 to t1
718:            b randomMove  #back to top of randomMove
719:    computeReturnAddress:
720:            lw    $ra, ($sp)       # Pop the return address, $ra.
721:            addu  $sp, $sp, 4      # Increment the $sp.
722:            jr    $ra  #return to original caller
723: #
724: ################## PLACING PIECE ###################
725: #
726:    offsetAndPlace:
727:            #to load into board
728:            lb    $a0,playerTurn($zero) # load turn
729:            move  $t0, $v0   #move number choice to $t0
730:            move  $t1, $v0   #move number choice to $t1
731:            sub   $t0, $t0, 1  #minus 1
732:            div   $t0, $t0, 3  #divide by three
733:            mul   $t0, $t0, 44  #multiply by 44
734:            mul   $t1, $t1, 2   #$t1 X 2
735:            add   $t1, $t1, 7   #add 7 to $t1
736:            add   $t0, $t1, $t0 #add $t1 and $t0
737:            sb    $a0, board($t0)  # Store the marker in the board
738:            jr    $ra # return to caller
739:
740: #
741: ################## EXIT BLOCK ###################
742: #
743:    exit:
744:            li   $v0, 10          # system call for exit
745:            syscall
```