

```

1:
2: # Note: Additional comments added by NM, marked by "NM".
3:
4: # Note that the sample Makefile in the comments shows that the entry
5: # point to the program is _tetrtris (see the ld line).
6:
7: # Note that the author has used a lot of macros here.
8:
9: # Read "main" first, by going to the label _tetrtris. That's where the
10: # "NM" comments are.
11:
12: #Tetrtris in x86 assembly
13: #Copyright 2000,2001 dburr@ug.cs.usyd.edu.au, under the terms of the GPL
14: #For the latest version, see http://bordello.2y.net:8000/programs/tetrtris
15: #
16: #Notes:
17: #Uses VT100 terminal codes to position the cursor and to draw coloured text.
18: #I also assume that this requires a > 2.0 Linux kernel which supports
19: #sys_newselect (also uses sys_write, sys_read, sys_nanosleep, sys_exit,
20: #sys_times and sys_ioctl). I have only tested it with 2.0.x, 2.2.x and 2.4.x
21: #kernels. I'm pretty sure that this will work with 386+ processors.
22: #
23: #Example Makefile:
24: #--start--
25: #NAME = tetrtris
26: #ENTRY = _tetrtris
27: #
28: #SYMS = -defsym instructionsX=12
29: #SYMS += -defsym instructionsY=19
30: #SYMS += -defsym width=10
31: #SYMS += -defsym xoffset=3
32: #SYMS += -defsym height=16
33: #SYMS += -defsym yoffset=2
34: #SYMS += -defsym wait=50
35: #SYMS += -defsym scoredrop=2
36: #SYMS += -defsym scorelockin=3
37: #SYMS += -defsym scoreline=100
38: #SYMS += -defsym scoretetris=1000
39: #SYMS += -defsym speedup=10
40: #SYMS += -defsym colour=1
41: #
42: #$(NAME).o: $(NAME).o
43: # ld -s -o $@ -m elf_i386 $^ -e $(ENTRY)
44: #
45: #$(NAME).o: $(NAME).s
46: # as -o $@ $^ $(SYMS)
47: #
48: #clean:
49: # rm -f $(NAME).o $(NAME)
50: #--end--
51: #Explanation of the symbols which can be changed to suit personal taste:
52: #instructionsX, instructionsY: The offset of the instructions from the
53: #top, left hand corner of the screen.
54: #width, height: The dimensions of the playing area
55: #xoffset, yoffset: The offset of the playing area from the top, left hand
56: #corner of the screen.
57: #wait: Controls the speed of the game. Lower number equals faster play.
58: #scoredrop: Number of points scored for 'dropping' a piece (with spacebar).
59: #scorelockin: Number of points scored for 'locking' a piece in.
60: #scoreline: Number of points scored for eliminating a line (for 1-3 lines)
61: #scoretetris: Number of points scored for a 'tetris' (ie eliminating 4 lines
62: #at once).
63: #speedup: The game will get twice as fast for every n lines.
64: #colour: Set this to 0 if you want to compile without colours (eg to
65: #play over Nifty Telnet on MacOS or the built-in telnet client on Windows)
66:
67: .macro sys_newselect
68:     xor %eax, %eax #smaller than writing to %eax directly
69:     mov $142, %al #new sys_select
70:     xor %ebx, %ebx
71:     mov $1, %bl #highest is 0 (stdin), plus 1
72:     bts $0, -128(%esp)
73:     lea -128(%esp), %ecx #Take some stack for the fd_set struct
74:     xor %edx, %edx #writefds
75:     xor %esi, %esi #exceptfds
76:     mov $timeout, %edi
77:     int $0x80
78: .endm
79:
80: .macro sys_nanosleep length
81:     xor %eax, %eax
82:     mov $162, %al #sys_nanosleep
83:     movl $0, -8(%esp) #seconds
84:     movl \length, -4(%esp) #nanoseconds
85:     lea -8(%esp), %ebx
86:     xor %ecx, %ecx #ignore remainder
87:     int $0x80
88: .endm
89:
90: .macro sys_exit
91:     xor %eax, %eax
92:     mov $1, %al #sys_exit
93:     xor %ebx,%ebx #with 0 status
94:     int $0x80
95: .endm
96:
97: .macro sys_times
98:     xor %eax, %eax
99:     mov $43, %al #sys_times
100:    xor %ebx,%ebx #NULL
101:    int $0x80
102: .endm
103:
104: .macro getterm
105:    lea -60(%esp), %edx #big enough for a termios struct
106:    mov $0x5401, %ecx #TCGETS
107:    call sys_ioctl
108: .endm
109:
110: .macro setterm #No need to set %edx again
111:    mov $0x5403, %ecx #TCSETSW
112:    call sys_ioctl
113: .endm
114:
115: #Write the chars equivalent to 'source' into vt100_position
116: .macro twodigits source first second
117:     mov \source, %ax
118:     inc %ax
119:     movb $10, %bl
120:     divb %bl
121:     add $0x30,%al
122:     movb %al, vt100_position+\first
123:     add $0x30, %ah
124:     movb %ah, vt100_position+\second
125: .endm
126:
127: #Named in honour of the nurses function
128: .macro mvaddstr y x string length
129:     twodigits \y, 2, 3
130:     twodigits \x, 5, 6
131:     mov %vt100_position, %ecx
132:     xor %edx, %edx
133:     mov $8, %dl
134:     call sys_write
135:     mov \string, %ecx
136:     xor %edx, %edx

```

```
137:     mov \length, %dl
138:     call sys_write
139: .endm
140:
141: #Mask off the bits for the n'th block
142: .macro bitMask n
143:     .if 4-\n
144:         shr $8-2*\n, %dx
145:     .endif
146:     and $0x303, %dx #Lower two bits of each
147:     mov yposition, %al
148:     add %dl, %al
149: .endm
150:
151: #Put the y location of the n'th block in %ax, x location in %bx
152: .macro screenoffset n
153:     bitMask \n
154:     mov xposition, %bx
155:     add %dh, %bl
156: .endm
157:
158: #Make %ax the offset of the n'th block from the start of the screen array
159: #where %dx is the piece in question
160: .macro pieceoffset n
161:     bitMask \n
162:     imul $width, %ax
163:     add xposition, %ax
164:     shr $8, %dx
165:     add %dx, %ax
166: .endm
167:
168: #Make real use of gas macros
169: .macro storeLoop from=1, to=4
170:     .if 4-\from
171:         movw (%esp), %dx
172:     .else
173:         pop %dx
174:     .endif
175:     pieceoffset \from
176:     movb %bl, (%eax, %ecx)
177:     .if \to-\from
178:         storeLoop "(\from+1)", \to
179:     .endif
180: .endm
181:
182: .macro collisionLoop from=1, to=4
183:     .if 1-\from
184:         movw (%esp), %dx
185:     .endif
186:     pieceoffset \from
187:     .if colour
188:         cmpb $0x30, (%ebx, %eax)
189:     .else
190:         cmpb $0x0, (%ebx, %eax)
191:     .endif
192:     .if 4-\from
193:         jnz collisionTest_over
194:     .endif
195:     .if \to-\from
196:         collisionLoop "(\from+1)", \to
197:     .endif
198: .endm
199:
200: .macro drawLoop from=1, to=4
201:     .if 1-\from
202:         movw (%esp), %cx
203:     .endif
204:     .if 4-\from
205:         mov 2(%esp), %dx
206:     .else
207:         pop %cx
208:         pop %dx
209:     .endif
210:     screenoffset \from
211:     call drawblock
212:     .if \to-\from
213:         drawLoop "(\from+1)", \to
214:     .endif
215: .endm
216:
217: .data
218:
219: quitstring:
220:     .ascii "'q' to quit, arrow keys to move"
221:
222: scorestring:
223:     .ascii "Score: "
224:
225: linestring:
226:     .ascii "Lines: "
227:
228: namestring:
229:     .ascii "Daniel's Tetris"
230:
231: blankstring:
232:     .ascii " "
233:
234: exitstring:
235:     .ascii "User exited"
236:
237: newline:
238:     .ascii "\n" #Also used after the previous string
239:
240: loserstring:
241:     .ascii "Loser\n"
242:
243: creditstring:
244:     .ascii "Tetris in 3k, by dburr@ug.cs.usyd.edu.au\n"
245:
246: score:
247:     .hword 0
248:
249: timeout:
250:     .long 0
251:     .long 0 #No delay while checking stdin
252:
253: vt100_position:
254:     .byte 0x1b
255:     .ascii "[12;13H"
256:
257: .if colour
258: vt100_colour: #The proper english way of spelling the word!
259:     .byte 0x1b
260:     .ascii "[44m"
261: .else
262: vt100_invert:
263:     .byte 0x1b
264:     .ascii "[07m"
265: .endif
266:
267: vt100_clear:
268:     .byte 0x1b
269:     .ascii "[2J"
270:
271: vt100_cursor:
272:     .byte 0x1b
```

```

273: .ascii "[?25l"
274:
275: yposition:
276: .byte 0
277:
278: xposition:
279: .hword 2
280:
281: sleepcount:
282: .byte 0
283:
284: shapeStarts:
285: .byte 2, 3, 5, 7, 11, 15, 19
286:
287: shapeIndex: #This data contains the positions of the blocks in each shape
288: #Each requires 16 bits: x1<<14|x2<<12|x3<<10|x4<<8|y1<<6|y2<<4|y3<<2|y4
289: .hword 0b0000010100010110 #0<<14|0<<12|1<<10|1<<8|0<<6|1<<4|1<<2|2
290: .hword 0b0100100100010001 #1<<14|0<<12|2<<10|1<<8|0<<6|1<<4|0<<2|1
291: .hword 0b0000010100010100 #0<<14|0<<12|1<<10|1<<8|0<<6|1<<4|1<<2|0
292: .hword 0b0000000000011011 #0<<14|0<<12|0<<10|0<<8|0<<6|1<<4|2<<2|3
293: .hword 0b0001101100000000 #0<<14|1<<12|2<<10|3<<8|0<<6|0<<4|0<<2|0
294: .hword 0b0001011000000101 #0<<14|1<<12|1<<10|2<<8|0<<6|0<<4|1<<2|1
295: .hword 0b0100010000010110 #1<<14|0<<12|1<<10|0<<8|0<<6|1<<4|1<<2|2
296: .hword 0b0001011001000101 #0<<14|1<<12|1<<10|2<<8|1<<6|0<<4|1<<2|1
297: .hword 0b0100010100010110 #1<<14|0<<12|1<<10|1<<8|0<<6|1<<4|1<<2|2
298: .hword 0b0001100100000001 #0<<14|1<<12|2<<10|1<<8|0<<6|0<<4|0<<2|1
299: .hword 0b0000000100011001 #0<<14|0<<12|0<<10|1<<8|0<<6|1<<4|2<<2|1
300: .hword 0b0001101000000001 #0<<14|1<<12|2<<10|2<<8|0<<6|0<<4|0<<2|1
301: .hword 0b0001000000000110 #0<<14|1<<12|0<<10|0<<8|0<<6|0<<4|1<<2|2
302: .hword 0b0000011000010101 #0<<14|0<<12|1<<10|2<<8|0<<6|1<<4|1<<2|1
303: .hword 0b0101010000011010 #1<<14|1<<12|1<<10|0<<8|0<<6|1<<4|2<<2|2
304: .hword 0b0001010100000110 #0<<14|1<<12|1<<10|1<<8|0<<6|0<<4|1<<2|2
305: .hword 0b0001100000000001 #0<<14|1<<12|2<<10|0<<8|0<<6|0<<4|0<<2|1
306: .hword 0b0000000100011010 #0<<14|0<<12|0<<10|1<<8|0<<6|1<<4|2<<2|2
307: .hword 0b1000011000010101 #2<<14|0<<12|1<<10|2<<8|0<<6|1<<4|1<<2|1
308:
309: linesgone:
310: .hword 0 #number of lines eliminated so far in the game
311:
312: currentwait:
313: .byte wait #gets smaller as the game gets faster
314:
315: .bss
316:
317: buffer:
318: .byte 0, 0 #for arrow keys we read two
319:
320: rotation:
321: .byte 0 #overwrite with a random rotation
322:
323: blockType:
324: .byte 0 #overwrite with a random block type
325:
326: .if colour
327: currentcolour:
328: .byte 0 #overwrite with random colour
329: .endif
330:
331: stringbuffer:
332: .fill 5
333:
334: screen:
335: .fill width*height
336:
337: lastrand:
338: .long 0
339:
340: .globl _tetris
341: .text
342:
343: #Return a 4-bit number in %al that is no greater than %cl
344: rand:
345: movl lastrand, %eax
346: mov %eax, %ebx
347: imul $1664525, %eax
348: add $1013904223, %eax
349: shr $10, %eax
350: xor %ebx, %eax
351: movl %eax, lastrand
352: andb $0x7, %al
353: cmp %al, %cl
354: jb rand
355: ret
356:
357: #Requires the string to write in %ecx, length in %edx
358: sys_write:
359: xor %eax, %eax
360: mov $4, %al #sys_write
361: xor %ebx, %ebx
362: mov $1, %bl #stdout
363: int $0x80
364: ret
365:
366: #Requires the length to read in %edx
367: sys_read:
368: xor %eax, %eax
369: mov $3, %al #sys_read
370: xor %ebx, %ebx #fd stdin
371: mov $buffer, %ecx #buffer
372: int $0x80
373: ret
374:
375: #Requires the number of the ioctl in %ecx, address for termios struct in %edx
376: sys_ioctl:
377: xor %eax, %eax
378: mov $54, %al #sys_ioctl
379: xor %ebx, %ebx #stdin
380: int $0x80
381: ret
382:
383: #Take the current entry from the shapeIndex and push it on the stack
384: coords:
385: xor %edx, %edx
386: xor %eax, %eax
387: mov blockType, %al
388: test %al, %al
389: jz coords_noIndex
390: mov shapeStarts-1(%eax), %dl
391: coords_noIndex:
392: add rotation, %dl
393: shl $1, %dl #because each entry is 2 bytes
394: pop %eax
395: pushw shapeIndex(%edx)
396: jmp *%eax
397:
398: #Write the block into the screen array at xposition, yposition
399: storePiece:
400: addw $scorelockin, score
401: decb yposition
402: .if colour
403: movb currentcolour, %cl
404: .else
405: movb $0xff, %cl
406: .endif
407: call drawShape
408: mov yposition, %al

```

```
409: test %al, %al
410: jz gameover
411:
412: call coords
413: xor %eax, %eax
414: .if colour
415: mov currentcolour, %bl
416: .else
417: mov $0xff, %bl
418: .endif
419: mov $screen, %ecx
420: storeLoop
421:
422: #There are 4 squares in the current piece. Test the lines which these
423: #occupy to see if they are part of a complete line. If so, remove, redraw
424: #Also adds to the score and speeds the game up if necessary
425: elimline:
426: mov yposition, %dl
427: xor %eax, %eax
428: mov %dl, %al #%al will contain the ypositions to test
429: add $4, %dl
430: xor %dh, %dh #number of lines eliminated in %dh
431: cmpb $height-1, %dl
432: jl elimline_skip
433: mov $height-1, %dl #%dl contains one more than the last value to test
434: elimline_skip:
435: xor %ebx, %ebx
436: mov $width, %bl
437: imul %eax, %ebx
438: add $screen, %ebx #ebx contains the start of the line
439: xor %ecx, %ecx
440: elimline_test:
441: inc %cl #%ecx contains the x position to test
442: .if colour
443: cmpb $0x30, (%ecx, %ebx) #test this for each position in line
444: .else
445: cmpb $0, (%ecx, %ebx) #test this for each position in line
446: .endif
447: je elimline_linedone #ie: don't eliminate this line
448: cmpb $width-2, %cl
449: jne elimline_test
450: inc %dh
451: add $width, %ebx
452: elimline_loop:
453: dec %ebx
454: movb -width(%ebx), %cl
455: movb %cl, (%ebx)
456: cmp $screen+width, %ebx
457: jne elimline_loop
458: elimline_linedone:
459: inc %al
460: cmp %al, %dl
461: jne elimline_skip
462:
463: mov %dh, %ch #for testing linesgone later
464: cmpb $4, %dh
465: je elimline_tetris
466: shr $8, %dx
467: imul $scoreline, %dx
468: addw %dx, score
469: jmp elimline_finished
470: elimline_tetris:
471: addw $scoretetris, score
472: elimline_finished:
473: shr $8, %cx
474: movw linesgone, %ax
475: mov $speedup, %bl
476: div %bl

477: mov %al, %dl
478: addw %cx, linesgone
479: movw linesgone, %ax
480: div %bl
481: cmp %al, %dl
482: je elimline_samespeed
483: shrb $1, currentwait
484: elimline_samespeed:
485: test %cl, %cl
486: jz elimline_noredraw
487: call redraw
488: elimline_noredraw:
489: movb $0, yposition
490: movw $2, xposition
491: movb $0, sleepcount
492: jmp playgame
493:
494: #Draw the current blockType at xposition,yposition (offset from xoffset,
495: #yoffset). Will be coloured depending on %cl. Update score
496: drawShape:
497: call coords
498: push %cx
499: drawLoop
500: .if colour
501: movb $0x30, vt100_colour+3
502: mov $vt100_colour, %ecx
503: .else
504: movb $0x30, vt100_invert+3
505: mov $vt100_invert, %ecx
506: .endif
507: xor %edx, %edx
508: mov $5, %dl
509: call sys_write
510: mvaddstr $instructionsY+2, $instructionsX, $scorestring, $7
511: mov score, %ax
512: call numbertostring
513: mvaddstr $instructionsY+3, $instructionsX, $linestring, $7
514: mov linesgone, %ax
515: call numbertostring
516: ret
517:
518: #Write the number in %ax
519: numbertostring:
520: mov $10, %bx
521: mov $stringbuffer+5, %ecx
522: numbertostring_loop:
523: dec %ecx
524: xor %dx, %dx
525: div %bx
526: add $0x30, %dl
527: movb %dl, (%ecx)
528: test %ax, %ax
529: jnz numbertostring_loop
530: mov $stringbuffer+5, %edx
531: sub %ecx, %edx
532: call sys_write
533: ret
534:
535: #Requires the y coord in %al, x coord in %bx, val to colour in %cl
536: drawblock:
537: xor %ah, %ah
538: add $xoffset, %bx
539: add $yoffset, %ax
540: push %ax
541: push %bx
542: .if colour
543: movb %cl, vt100_colour+3
544: mov $vt100_colour, %ecx
```

```

545: .else
546:     test %cl, %cl
547:     jz drawblock_out
548:     sub $0xf8, %cl
549: drawblock_out:
550:     add $0x30, %cl
551:     movb %cl, vt100_invert+3
552:     mov $vt100_invert, %ecx
553: .endif
554:     xor %edx, %edx
555:     mov $5, %dl
556:     call sys_write
557:     pop %cx
558:     pop %ax
559:     shl $1, %cx
560:     mvaddstr %ax, %cx, $blankstring, $2
561:     ret
562:
563: #Redraw the playing area (doesn't update score)
564: redraw:
565:     xor %ax, %ax #y
566: redraw_outer:
567:     xor %ebx, %ebx #x
568: redraw_inner:
569:     push %ebx
570:     push %ax
571:
572:     xor %ecx, %ecx
573:     mov $width, %cl
574:     imul %eax, %ecx
575:     mov screen(%ebx, %ecx), %cx
576:
577:     call drawblock
578:
579:     pop %ax
580:     pop %ebx
581:
582:     inc %bl
583:     cmpb $width, %bl
584:     jl redraw_inner
585:
586:     inc %ax
587:     cmpb $height, %al
588:     jl redraw_outer
589:     ret
590: gameover:
591:
592: # [NM] here we must restore the screen, and especially the keyboard; to
593: # see why, try playing the game but killing it with ctrl-C; you'll see
594: # that the keyboard suddenly becomes inoperable in that window (to
595: # restore it, hit ctrl-j then "reset" then ctrl-j again)
596: .if colour
597:     movw $0x3030, vt100_colour+2
598:     mov $vt100_colour, %ecx
599: .else
600:     movb $0x30, vt100_invert+3
601:     mov $vt100_invert, %ecx
602: .endif
603:     xor %edx, %edx
604:     mov $5, %dl
605:     call sys_write
606:     movb $'h', vt100_cursor+5
607:     mov $vt100_cursor, %ecx
608:     xor %edx, %edx
609:     mov $6, %dl
610:     call sys_write
611:     cmpb $'q', buffer
612:     jne gameover_loser
613:
614:     mvaddstr $instructionsY+4, $0, $exitstring, $13
615:     jmp gameover_quit
616: gameover_loser:
617:     mvaddstr $instructionsY+4, $0, $loserstring, $6
618: gameover_quit:
619:     mov $scorestring, %ecx
620:     xor %edx, %edx
621:     mov $7, %dl
622:     call sys_write
623:     mov score, %ax
624:     call numbertostring
625:     mov $newline, %ecx
626:     xor %edx, %edx
627:     mov $1, %dl
628:     call sys_write
629:     mov $creditstring, %ecx
630:     xor %edx, %edx
631:     mov $41, %dl
632:     call sys_write
633:     getterm
634:     or $10, -48(%esp) #c_lflag |= (ICANON|ECHO)
635:     setterm
636:     sys_exit
637:
638: #Test the shape for any collision. If collision, then the zero flag will
639: #NOT be set
640: collisionTest:
641:     call coords
642:     xor %eax, %eax
643:     mov $screen, %ebx
644:     collisionLoop
645: collisionTest_over:
646:     pop %dx
647:     ret
648:
649: #Writes the number of rotations of blockType into %cl
650: numberrots:
651:     xor %ebx, %ebx
652:     mov blockType, %bl
653:     test %bl, %bl
654:     jz numberrots_zeroshape
655:     add $shapeStarts, %ebx
656:     mov (%ebx), %cl
657:     sub -1(%ebx), %cl
658:     jmp numberrots_done
659: numberrots_zeroshape:
660:     mov shapeStarts, %cl
661: numberrots_done:
662:     ret
663:
664: _tetris:
665:     # [NM] the next 3 lines (2 macro calls and an andb) unset
666:     # canonical mode, so keyboard input is instant, i.e. no waiting
667:     # for the user to hit the Enter key; the echo is also unset
668:     getterm
669:     andb $245, -48(%esp) #c_lflag &= ~(ICANON|ECHO)
670:     setterm
671:
672:     sys_times
673:     mov %eax, lastrand #seed the randomizer
674:
675:     # [NM] in the old days, cursor movement on a terminal was done
676:     # by printing a certain sequence of bytes to the screen; it was
677:     # different from each brand/model of terminal, but eventually
678:     # Digital Equipment Corporation's VT100 terminal type became a
679:     # standard, and today almost all terminal windows (e.g. xterm)
680:     # emulate a VT100 terminal

```

```

681: # [NM] the next 4 lines of code write the code for clearing (i.e.
682: # blanking out) a VT100 screen; if you check earlier in the file,
683: # you'll see that vt100_clear is this:
684:
685: # vt100_clear:
686: # .byte 0x1b
687: # .ascii "[2J"
688:
689: # [NM] that first byte is the ASCII code for ESC (the Escape key),
690: # which is a preface for all the VT100 cursor-movement codes; in
691: # other words, to clear a VT100 screen, one sends ESC[2J to the
692: # screen; try it yourself, by compiling and running this C
693: # program:
694:
695: # main()
696: #
697: # { int esc = 27;
698: #
699: # printf("%c[2J",esc);
700: #
701: # }
702:
703: # [NM] one more thing: the author does a write to the screen so
704: # often (duh!) that he has collected the code to do so into a
705: # subroutine, which he has named sys_write, but which is simply
706: # a call to the usual OS function write(), OS call #4:
707:
708: # sys_write:
709: #     xor %eax, %eax
710: #     mov $4, %al #sys_write
711: #     xor %ebx, %ebx
712: #     mov $1, %bl #stdout
713: #     int $0x80
714: #     ret
715:
716: # [NM], so, here is the code to clear the screen:
717: mov $vt100_clear, %ecx
718: xor %edx, %edx
719: mov $4, %dl
720: call sys_write
721:
722: # [NM] these lines initialize the cursor position
723: mov $vt100_cursor, %ecx
724: xor %edx, %edx
725: mov $6, %dl
726: call sys_write
727:
728: .if colour
729:     mov $vt100_colour, %ecx
730: .else
731:     mov $vt100_invert, %ecx
732: .endif
733: xor %edx, %edx
734: mov $5, %dl
735: call sys_write
736:
737: # [NM] in the old days, before VT100 really became standard,
738: # there needed to be a way for people to write programs which
739: # would work on any terminal type; for example, if you were
740: # writing a text editor, like vi, you certainly would not want
741: # to have to write a different version for each terminal type;
742: # so, UNIX developers wrote the package named "curses" (get the
743: # pun?); the programmer would simply call functions in this
744: # package, and those functions would worry about how to make a
745: # certain operation (e.g. cursor up one line) work; they would
746: # do this by lookups in a database of all known terminal types
747: # and their various cursor-movement codes, but the point is that
748: # a programmer writing, say, vi could program cursor movements
749:
750: # without knowing what kind of terminal the user would use; the
751: # author of Tetris here has written his own functions like that
752: # (linking in curses from the C library would make the game too
753: # big) and has even used the same macro names, e.g. mvaddstr
754:
755: mvaddstr $instructionsY, $instructionsX, $namestring, $15
756: mvaddstr $instructionsY+1, $instructionsX, $quitstring, $31
757:
758: # [NM] see one more set of NM comments at "gameover" above
759:
760:     xor %al,%al
761:     mov $screen,%ebx
762: tetris_yloop:
763: .if colour
764:     movb $0x31, (%ebx) #red for the playing arena
765:     movb $0x31, width-1(%ebx)
766: .else
767:     movb $0xff, (%ebx)
768:     movb $0xff, width-1(%ebx)
769: .endif
770:     xor %ecx, %ecx
771:     mov $1, %cl
772: tetris_yloop_inner:
773: .if colour
774:     movb $0x30, (%ebx, %ecx) #init to black
775: .else
776:     movb $0, (%ebx, %ecx)
777: .endif
778:     inc %cl
779:     cmpb $width-1, %cl
780:     jl tetris_yloop_inner
781:
782:     add $width, %ebx
783:     inc %al
784:     cmpb $height-1, %al
785:     jl tetris_yloop
786:
787:     mov $width*(height-1)+screen, %ebx
788:     xor %eax, %eax
789: tetris_xloop:
790: .if colour
791:     movb $0x31, (%eax,%ebx)
792: .else
793:     movb $0xff, (%eax,%ebx)
794: .endif
795:     inc %al
796:     cmpb $width, %al
797:     jl tetris_xloop
798:
799:     call redraw
800:
801: playgame:
802:     mov $6, %cl #7 shapes
803:     call rand
804:     movb %al, blockType
805:     call numberrots
806:     dec %cl
807:     call rand
808:     movb %al, rotation
809: .if colour
810:     mov $6, %cl
811:     call rand
812:     add $0x31, %al
813:     mov %al, currentcolour
814:     mov %al, %cl
815: .else
816:     movb $0xff, %cl

```

```
817: .endif
818:     call drawShape
819:     call collisionTest
820:     jnz gameover
821:
822: playgame_keyloop:
823:     sys_nanosleep $250000
824:     incb sleepcount
825:     movb currentwait, %cl
826:     cmpb %cl, sleepcount
827:     je playgame_slept
828:     sys_newselect
829:     test %eax, %eax
830:     jz playgame_keyloop
831: playgame_checkkey:
832:     xor %edx, %edx
833:     mov $1, %dl
834:     call sys_read
835:     cmpb '$q', buffer
836:     je gameover
837: .if colour
838:     mov $0x30, %cl
839: .else
840:     xor %cl, %cl
841: .endif
842:     call drawShape
843:     cmpb '$ ', buffer
844:     jne playgame_checkarrow
845: playgame_droploop:
846:     incb yposition
847:     call collisionTest
848:     jz playgame_droploop
849:     addw $scoredrop, score
850:     jmp storePiece
851: playgame_checkarrow:
852:     cmpb $0x1b, buffer #check for arrow key
853:     jne playgame_checkdone
854:     xor %edx, %edx
855:     mov $2, %dl
856:     call sys_read
857:     movb buffer+1, %al
858:     cmpb '$D', %al #Left arrow
859:     je playgame_leftarrow
860:     cmpb '$C', %al #Right Arrow
861:     je playgame_rightarrow
862:     cmpb '$B', %al #Down Arrow
863:     je playgame_downarrow
864:     cmpb '$A', %al #Up Arrow
865:     jne playgame_checkdone
866:     xor %ah, %ah
867:     mov rotation, %al
868:     push %ax
869:     inc %al
870:     call numberrots
871:     divb %cl
872:     movb %ah, rotation
873:     call collisionTest
874:     jz playgame_checkdone
875:     pop %cx
876:     mov %cl, rotation
877:     jmp playgame_checkdone
878: playgame_leftarrow:
879:     decw xposition
880:     call collisionTest
881:     jz playgame_checkdone
882:     incw xposition
883:     jmp playgame_checkdone
884: playgame_rightarrow:
885:     incw xposition
886:     call collisionTest
887:     jz playgame_checkdone
888:     decw xposition
889:     jmp playgame_checkdone
890: playgame_downarrow:
891:     incb yposition
892:     call collisionTest
893:     jz playgame_checkdone
894:     decb yposition
895:     jmp playgame_checkdone
896:
897: playgame_slept:
898: .if colour
899:     mov $0x30, %cl #black to overwrite
900: .else
901:     xor %cl, %cl
902: .endif
903:     call drawShape
904:     incb yposition
905:     call collisionTest
906:     jnz storePiece
907:     movb $0, sleepcount
908:
909: playgame_checkdone:
910: .if colour
911:     movb currentcolour, %cl
912: .else
913:     movb $0xff, %cl
914: .endif
915:     call drawShape
916:     jmp playgame_keyloop
```