```
  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 _tetris (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 _tetris NEAR THE END OF THE
 10: # FILE.  THAT IS THE ENTRY POINT OF THE PROGRAM, I.E. THE PLACE AT WHICH
 11: # EXECUTION STARTS.
 12:
 13: #Tetris in x86 assembly
 14: #Copyright 2000,2001 dburr@ug.cs.usyd.edu.au, under the terms of the GPL
 15: #For the latest version, see http://bordello.2y.net:8000/programs/tetris
 16: #
 17: #Notes:
 18: #Uses VT100 terminal codes to position the cursor and to draw coloured text.
 19: #I also assume that this requires a > 2.0 Linux kernel which supports
 20: #sys_newselect (also uses sys_write, sys_read, sys_nanosleep, sys_exit,
 21: #sys_times and sys_ioctl).  I have only tested it with 2.0.x, 2.2.x and 2.4.x
 22: #kernels.  I'm pretty sure that this will work with 386+ processors.
 23: #
 24:
 25: # [NM] note that in the sample Makefile below, the entry point is specified
 26: # via the -e option to ld; note also the definition of various
 27: # quantities via -defsym; this is similar to a #define, but supplied
 28: # externally to calling "as", on the command line; GCC has the same
 29: # thing with -D
 30:
 31: #Example Makefile:
 32: #--start--
 33: #NAME = tetris
 34: #ENTRY = _tetris
 35: #
 36: #SYMS =  -defsym instructionsX=12
 37: #SYMS += -defsym instructionsY=19
 38: #SYMS += -defsym width=10
 39: #SYMS += -defsym xoffset=3
 40: #SYMS += -defsym height=16
 41: #SYMS += -defsym yoffset=2
 42: #SYMS += -defsym wait=50
 43: #SYMS += -defsym scoredrop=2
 44: #SYMS += -defsym scorelockin=3
 45: #SYMS += -defsym scoreline=100
 46: #SYMS += -defsym scoretetris=1000
 47: #SYMS += -defsym speedup=10
 48: #SYMS += -defsym colour=1
 49: #
 50: #$(NAME): $(NAME).o
 51: #       ld -s -o $@ -m elf_i386 $^ -e $(ENTRY)
 52: #
 53: #$(NAME).o: $(NAME).s
 54: #       as -o $@ $^ $(SYMS)
 55: #
 56: #clean:
 57: #       rm -f $(NAME).o $(NAME)
 58: #--end--
 59: #Explanation of the symbols which can be changed to suit personal taste:
 60: #instructionsX, instructionsY: The offset of the instructions from the
 61: #top, left hand corner of the screen.
 62: #width, height: The dimensions of the playing area
 63: #xoffset, yoffset: The offset of the playing area from the top, left hand
 64: #corner of the screen.
 65: #wait: Controls the speed of the game.  Lower number equals faster play.
 66: #scoredrop: Number of points scored for 'dropping' a piece (with spacebar).
 67: #scorelockin: Number of points scored for 'locking' a piece in.
 68: #scoreline: Number of points scored for eliminating a line (for 1-3 lines)
 69: #scoretetris: Number of points scored for a 'tetris' (ie eliminating 4 lines
 70: #at once).
 71: #speedup: The game will get twice as fast for every n lines.
 72: #colour: Set this to 0 if you want to compile without colours (eg to
 73: #play over Nifty Telnet on MacOS or the built-in telnet client on Windows)
 74:
 75: .macro sys_newselect
 76:         xor %eax, %eax #smaller than writing to %eax directly
 77:         mov $142, %al #new sys_select
 78:         xor %ebx, %ebx
 79:         mov $1, %bl #highest is 0 (stdin), plus 1
 80:         bts $0, -128(%esp)
 81:         lea -128(%esp), %ecx #Take some stack for the fd_set struct
 82:         xor %edx, %edx #writefds
 83:         xor %esi, %esi #exceptfds
 84:         mov $timeout, %edi
 85:         int $0x80
 86: .endm
 87:
 88: .macro sys_nanosleep length
 89:         xor %eax, %eax
 90:         mov $162, %al #sys_nanosleep
 91:         movl $0, -8(%esp) #seconds
 92:         movl \length, -4(%esp) #nanoseconds
 93:         lea -8(%esp), %ebx
 94:         xor %ecx, %ecx #ignore remainder
 95:         int $0x80
 96: .endm
 97:
 98: .macro sys_exit
 99:         xor %eax, %eax
100:         mov $1, %al #sys_exit
101:         xor %ebx,%ebx #with 0 status
102:         int $0x80
103: .endm
104:
105: .macro sys_times
106:         xor %eax, %eax
107:         mov $43, %al #sys_times
108:         xor %ebx,%ebx #NULL
109:         int $0x80
110: .endm
111:
112: # [NM] one of the OS services is service number 54, ioctl; this is
113: # generally called from C/C++ instead of assembly language, but as with
114: # any system call, it can be done from assembly language; in the case of
115: # a terminal window, what this service does is allow the programmer to
116: # access characteristics of the screen, e.g. move the cursor, blank out
117: # the screen, etc.; all the current information (e.g. current position
118: # of the cursor) is recorded in a termios struct, maintained by the OS
119: # (type "man termios" to see what's in it; also "man ioctl"); to change
120: # something, e.g. move the cursor, one first gets a copy of the current
121: # struct, then changes the relevant field in it, and writes the copy
122: # back to the OS' original struct; the change is instantaneous
123:
124: .macro getterm
125:         lea -60(%esp), %edx #big enough for a termios struct
126:         mov $0x5401, %ecx #TCGETS
127:         call sys_ioctl
128: .endm
129:
130: .macro setterm #No need to set %edx again
131:         mov $0x5403, %ecx #TCSETSW
132:         call sys_ioctl
133: .endm
134:
135: # [NM] the VT100 code to move the cursor to row r, column c consists of
136: # first Esc, then [r;cH where r and c are given in 2-character string
```

```
# format, e.g. the string "12" for row 12; that's a total of 8 bytes;
# the part of the .data segment labeled vt100_position contains these 8
# bytes; in order to move the cursor, we must fill in the r and c
# portions of these bytes, and then write the 8 bytes to the screen; the
# macro twodigits below converts r or c (it's called twice from
# mvaddstr, once for r and then for c) from a number to a 2-character
# string, and places it into vt100_position; the last two arguments
# specify where to write within vt100_position (for row or column); the
# inc instruction is apparently there because the VT100 scheme starts
# numbering at 1 instead of 0

#Write the chars equivalent to 'source' into vt100_position
.macro twodigits source first second
        mov \source, %ax
        inc %ax
        movb $10, %bl
        divb %bl
        add $0x30,%al
        movb %al, vt100_position+\first
        add $0x30, %ah
        movb %ah, vt100_position+\second
.endm

# [NM] in the [n]curses package, the macro mvaddstr moves the cursor
# (thus the "mv" in the name) to row y, column x, and adds (thus
# "addstr") to the screen at that point, replacing what was there
# before, and then advancing the cursor to the point just after the
# string that was written; in the variant here, the numbers y and x must
# be converted to strings, as explained in my comments for the macro
# twodigits

#Named in honour of the ncurses function
.macro mvaddstr y x string length
        twodigits \y, 2, 3
        twodigits \x, 5, 6
        mov $vt100_position, %ecx
        xor %edx, %edx
        mov $8, %dl
        call sys_write
        mov \string, %ecx
        xor %edx, %edx
        mov \length, %dl
        call sys_write
.endm

#Mask off the bits for the n'th block
.macro bitMask n
        .if 4-\n
                shr $8-2*\n, %dx
        .endif
        and $0x303, %dx #Lower two bits of each
        mov yposition, %al
        add %dl, %al
.endm

#Put the y location of the n'th block in %ax, x location in %bx
.macro screenoffset n
        bitMask \n
        mov xposition, %bx
        add %dh, %bl
.endm

#Make %ax the offset of the n'th block from the start of the screen array
#where %dx is the piece in question
.macro pieceoffset n
        bitMask \n
        imul $width, %ax
        add xposition, %ax
        shr $8, %dx
        add %dx, %ax
.endm

#Make real use of gas macros
.macro storeLoop from=1, to=4
        .if 4-\from
                movw (%esp), %dx
        .else
                pop %dx
        .endif
        pieceoffset \from
        movb %bl, (%eax, %ecx)
        .if \to-\from
                storeLoop "(\from+1)", \to
        .endif
.endm

.macro collisionLoop from=1, to=4
        .if 1-\from
                movw (%esp), %dx
        .endif
        pieceoffset \from
.if colour
        cmpb $0x30, (%ebx, %eax)
.else
        cmpb $0x0, (%ebx, %eax)
.endif
        .if 4-\from
                jnz collisionTest_over
        .endif
        .if \to-\from
                collisionLoop "(\from+1)", \to
        .endif
.endm

.macro drawLoop from=1, to=4
        .if 1-\from
                movw (%esp), %cx
        .endif
        .if 4-\from
                mov 2(%esp), %dx
        .else
                pop %cx
                pop %dx
        .endif
        screenoffset \from
        call drawblock
        .if \to-\from
                drawLoop "(\from+1)", \to
        .endif
.endm

.data

quitstring:
        .ascii "'q' to quit, arrow keys to move"

scorestring:
        .ascii "Score: "

linestring:
        .ascii "Lines: "

namestring:
        .ascii "Daniel's Tetris"

blankstring:
```

```
273:         .ascii "  "
274:
275: exitstring:
276:         .ascii "User exitted"
277:
278: newline:
279:         .ascii "\n" #Also used after the previous string
280:
281: loserstring:
282:         .ascii "Loser\n"
283:
284: creditstring:
285:         .ascii "Tetris in 3k, by dburr@ug.cs.usyd.edu.au\n"
286:
287: score:
288:         .hword 0
289:
290: timeout:
291:         .long 0
292:         .long 0 #No delay while checking stdin
293:
294: vt100_position:
295:         .byte 0x1b
296:         .ascii "[12;13H"
297:
298: .if colour
299: vt100_colour: #The proper english way of spelling the word!
300:         .byte 0x1b
301:         .ascii "[44m"
302: .else
303: vt100_invert:
304:         .byte 0x1b
305:         .ascii "[07m"
306: .endif
307:
308: vt100_clear:
309:         .byte 0x1b
310:         .ascii "[2J"
311:
312: vt100_cursor:
313:         .byte 0x1b
314:         .ascii "[?25l"
315:
316: yposition:
317:         .byte 0
318:
319: xposition:
320:         .hword 2
321:
322: sleepcount:
323:         .byte 0
324:
325: shapeStarts:
326:         .byte 2, 3, 5, 7, 11, 15, 19
327:
328: shapeIndex: #This data contains the positions of the blocks in each shape
329: #Each requires 16 bits: x1<<14|x2<<12|x3<<10|x4<<8|y1<<6|y2<<4|y3<<2|y4
330:         .hword 0b0000010100010110 #0<<14|0<<12|1<<10|1<<8|0<<6|1<<4|1<<2|2
331:         .hword 0b0100100100010001 #1<<14|0<<12|2<<10|1<<8|0<<6|1<<4|0<<2|1
332:         .hword 0b0000010100010100 #0<<14|0<<12|1<<10|1<<8|0<<6|1<<4|1<<2|0
333:         .hword 0b0000000000011011 #0<<14|0<<12|0<<10|0<<8|0<<6|1<<4|2<<2|3
334:         .hword 0b0001101100000000 #0<<14|1<<12|2<<10|3<<8|0<<6|0<<4|0<<2|0
335:         .hword 0b0001011000000101 #0<<14|1<<12|1<<10|2<<8|0<<6|0<<4|1<<2|1
336:         .hword 0b0100010000010110 #1<<14|0<<12|1<<10|0<<8|0<<6|1<<4|1<<2|2
337:         .hword 0b0001011001000101 #0<<14|1<<12|1<<10|2<<8|1<<6|0<<4|1<<2|1
338:         .hword 0b0100010100010110 #1<<14|0<<12|1<<10|1<<8|0<<6|1<<4|1<<2|2
339:         .hword 0b0001100100000001 #0<<14|1<<12|2<<10|1<<8|0<<6|0<<4|0<<2|1
340:         .hword 0b0000000100011001 #0<<14|0<<12|0<<10|1<<8|0<<6|1<<4|2<<2|1
341:         .hword 0b0001101000000001 #0<<14|1<<12|2<<10|2<<8|0<<6|0<<4|0<<2|1
342:         .hword 0b0001000000000110 #0<<14|1<<12|0<<10|0<<8|0<<6|0<<4|1<<2|2
343:         .hword 0b0000011000010101 #0<<14|0<<12|1<<10|2<<8|0<<6|1<<4|1<<2|1
344:         .hword 0b0101010000011010 #1<<14|1<<12|1<<10|0<<8|0<<6|1<<4|2<<2|2
345:         .hword 0b0001010100000110 #0<<14|1<<12|1<<10|1<<8|0<<6|0<<4|1<<2|2
346:         .hword 0b0001100000000001 #0<<14|1<<12|2<<10|0<<8|0<<6|0<<4|0<<2|1
347:         .hword 0b0000000100011010 #0<<14|0<<12|0<<10|1<<8|0<<6|1<<4|2<<2|2
348:         .hword 0b1000011000010101 #2<<14|0<<12|1<<10|2<<8|0<<6|1<<4|1<<2|1
349:
350: linesgone:
351:         .hword 0 #number of lines eliminated so far in the game
352:
353: currentwait:
354:         .byte wait #gets smaller as the game gets faster
355:
356: .bss
357:
358: buffer:
359:         .byte 0, 0 #for arrow keys we read two
360:
361: rotation:
362:         .byte 0 #overwrite with a random rotation
363:
364: blockType:
365:         .byte 0 #overwrite with a random block type
366:
367: .if colour
368: currentcolour:
369:         .byte 0 #overwrite with random colour
370: .endif
371:
372: stringbuffer:
373:         .fill 5
374:
375: screen:
376:         .fill width*height
377:
378: lastrand:
379:         .long 0
380:
381: .globl _tetris
382: .text
383:
384: # [NM] most random number generators work something like the following;
385: # the next random number, call it n, is generated from the last one, say
386: # m, by multiplying m by a huge fixed number, adding another huge fixed
387: # number, and then doing a mod operation by another huge fixed number;
388: # years of research have found very good choices for those huge fixed
389: # numbers; the algorithm below is similar
390:
391: #Return a 4-bit number in %al that is no greater than %cl
392: rand:
393:         movl lastrand, %eax
394:         mov %eax, %ebx
395:         imul $1664525, %eax
396:         add $1013904223, %eax
397:         shr $10, %eax
398:         xor %ebx, %eax
399:         movl %eax, lastrand
400:         andb $0x7, %al
401:         cmp %al, %cl
402:         jb rand
403:         ret
404:
405: #Requires the string to write in %ecx, length in %edx
406: sys_write:
407:         xor %eax, %eax
408:         mov $4, %al #sys_write
```

```
409:         xor %ebx, %ebx
410:         mov $1, %bl #stdout
411:         int $0x80
412:         ret
413:
414: #Requires the length to read in %edx
415: sys_read:
416:         xor %eax, %eax
417:         mov $3, %al #sys_read
418:         xor %ebx, %ebx #fd stdin
419:         mov $buffer, %ecx #buffer
420:         int $0x80
421:         ret
422:
423: #Requires the number of the ioctl in %ecx, address for termios struct in %edx
424: sys_ioctl:
425:         xor %eax, %eax
426:         mov $54, %al #sys_ioctl
427:         xor %ebx, %ebx #stdin
428:         int $0x80
429:         ret
430:
431: #Take the current entry from the shapeIndex and push it on the stack
432: coords:
433:         xor %edx, %edx
434:         xor %eax, %eax
435:         mov blockType, %al
436:         test %al, %al
437:         jz coords_noIndex
438:         mov shapeStarts-1(%eax), %dl
439: coords_noIndex:
440:         add rotation, %dl
441:         shl $1, %dl #because each entry is 2 bytes
442:         pop %eax
443:         pushw shapeIndex(%edx)
444:         jmp *%eax
445:
446: #Write the block into the screen array at xposition, yposition
447: storePiece:
448:         addw $scorelockin, score
449:         decb yposition
450: .if colour
451:         movb currentcolour, %cl
452: .else
453:         movb $0xff, %cl
454: .endif
455:         call drawShape
456:         mov yposition, %al
457:         test %al, %al
458:         jz gameover
459:
460:         call coords
461:         xor %eax, %eax
462: .if colour
463:         mov currentcolour, %bl
464: .else
465:         mov $0xff, %bl
466: .endif
467:         mov $screen, %ecx
468:         storeLoop
469:
470: #There are 4 squares in the current piece.  Test the lines which these
471: #occupy to see if they are part of a complete line.  If so, remove, redraw
472: #Also adds to the score and speeds the game up if necessary
473: elimline:
474:         mov yposition, %dl
475:         xor %eax, %eax
476:         mov %dl, %al #%al will contain the ypositions to test
477:         add $4, %dl
478:         xor %dh, %dh #number of lines eliminated in %dh
479:         cmpb $height-1, %dl
480:         jl elimline_skip
481:         mov $height-1, %dl #%dl contains one more than the last value to test
482: elimline_skip:
483:         xor %ebx, %ebx
484:         mov $width, %bl
485:         imul %eax, %ebx
486:         add $screen, %ebx #ebx contains the start of the line
487:         xor %ecx, %ecx
488: elimline_test:
489:         inc %cl #%ecx contains the x position to test
490: .if colour
491:         cmpb $0x30, (%ecx, %ebx) #test this for each position in line
492: .else
493:         cmpb $0, (%ecx, %ebx) #test this for each position in line
494: .endif
495:         je elimline_linedone #ie: don't eliminate this line
496:         cmpb $width-2, %cl
497:         jne elimline_test
498:         inc %dh
499:         add $width, %ebx
500: elimline_loop:
501:         dec %ebx
502:         movb -width(%ebx), %cl
503:         movb %cl, (%ebx)
504:         cmp $screen+width, %ebx
505:         jne elimline_loop
506: elimline_linedone:
507:         inc %al
508:         cmp %al, %dl
509:         jne elimline_skip
510:
511:         mov %dh, %ch #for testing linesgone later
512:         cmpb $4, %dh
513:         je elimline_tetris
514:         shr $8, %dx
515:         imul $scoreline, %dx
516:         addw %dx, score
517:         jmp elimline_finished
518: elimline_tetris:
519:         addw $scoretetris, score
520: elimline_finished:
521:         shr $8, %cx
522:         movw linesgone, %ax
523:         mov $speedup, %bl
524:         div %bl
525:         mov %al, %dl
526:         addw %cx, linesgone
527:         movw linesgone, %ax
528:         div %bl
529:         cmp %al, %dl
530:         je elimline_samespeed
531:         shrb $1, currentwait
532: elimline_samespeed:
533:         test %cl, %cl
534:         jz elimline_noredraw
535:         call redraw
536: elimline_noredraw:
537:         movb $0, yposition
538:         movw $2, xposition
539:         movb $0, sleepcount
540:         jmp playgame
541:
542: #Draw the current blockType at xposition,yposition (offset from xoffset,
543: #yoffset).  Will be coloured depending on %cl. Update score
544: drawShape:
```

```
545:         call coords
546:         push %cx
547:         drawLoop
548: .if colour
549:         movb $0x30, vt100_colour+3
550:         mov $vt100_colour, %ecx
551: .else
552:         movb $0x30, vt100_invert+3
553:         mov $vt100_invert, %ecx
554: .endif
555:         xor %edx, %edx
556:         mov $5, %dl
557:         call sys_write
558:         mvaddstr $instructionsY+2, $instructionsX, $scorestring, $7
559:         mov score, %ax
560:         call numbertostring
561:         mvaddstr $instructionsY+3, $instructionsX, $linestring, $7
562:         mov linesgone, %ax
563:         call numbertostring
564:         ret
565:
566: #Write the number in %ax
567: numbertostring:
568:         mov $10, %bx
569:         mov $stringbuffer+5, %ecx
570: numbertostring_loop:
571:         dec %ecx
572:         xor %dx,%dx
573:         div %bx
574:         add $0x30, %dl
575:         movb %dl, (%ecx)
576:         test %ax,%ax
577:         jnz numbertostring_loop
578:         mov $stringbuffer+5, %edx
579:         sub %ecx, %edx
580:         call sys_write
581:         ret
582:
583: #Requires the y coord in %al, x coord in %bx, val to colour in %cl
584: drawblock:
585:         xor %ah, %ah
586:         add $xoffset,%bx
587:         add $yoffset,%ax
588:         push %ax
589:         push %bx
590: .if colour
591:         movb %cl, vt100_colour+3
592:         mov $vt100_colour, %ecx
593: .else
594:         test %cl, %cl
595:         jz drawblock_out
596:         sub $0xf8, %cl
597: drawblock_out:
598:         add $0x30, %cl
599:         movb %cl, vt100_invert+3
600:         mov $vt100_invert, %ecx
601: .endif
602:         xor %edx, %edx
603:         mov $5, %dl
604:         call sys_write
605:         pop %cx
606:         pop %ax
607:         shl $1, %cx
608:         mvaddstr %ax, %cx, $blankstring, $2
609:         ret
610:
611: #Redraw the playing area (doesn't update score)
612: redraw:
613:         xor %ax, %ax #y
614: redraw_outer:
615:                 xor %ebx, %ebx #x
616: redraw_inner:
617:                 push %ebx
618:                 push %ax
619:
620:                 xor %ecx, %ecx
621:                 mov $width, %cl
622:                 imul %eax, %ecx
623:                 mov screen(%ebx, %ecx), %cx
624:
625:                 call drawblock
626:
627:                 pop %ax
628:                 pop %ebx
629:
630:                 inc %bl
631:                 cmpb $width, %bl
632:                 jl redraw_inner
633:         inc %ax
634:         cmpb $height, %al
635:         jl redraw_outer
636:         ret
637:
638: gameover:
639:
640: # [NM] here we must restore the screen, and especially the keyboard; to
641: # see why, try playing the game but killing it with ctrl-C; you'll see
642: # that the keyboard suddenly becomes inoperable in that window (to
643: # restore it, hit ctrl-j then "reset" then ctrl-j again)
644: .if colour
645:         movw $0x3030, vt100_colour+2
646:         mov $vt100_colour, %ecx
647: .else
648:         movb $0x30, vt100_invert+3
649:         mov $vt100_invert, %ecx
650: .endif
651:         xor %edx, %edx
652:         mov $5, %dl
653:         call sys_write
654:         movb $'h',vt100_cursor+5
655:         mov $vt100_cursor, %ecx
656:         xor %edx, %edx
657:         mov $6, %dl
658:         call sys_write
659:         cmpb $'q',buffer
660:         jne gameover_loser
661:         mvaddstr $instructionsY+4, $0, $exitstring, $13
662:         jmp gameover_quit
663: gameover_loser:
664:         mvaddstr $instructionsY+4, $0, $loserstring, $6
665: gameover_quit:
666:         mov $scorestring, %ecx
667:         xor %edx, %edx
668:         mov $7, %dl
669:         call sys_write
670:         mov score, %ax
671:         call numbertostring
672:         mov $newline, %ecx
673:         xor %edx, %edx
674:         mov $1, %dl
675:         call sys_write
676:         mov $creditstring, %ecx
677:         xor %edx, %edx
678:         mov $41, %dl
679:         call sys_write
680:         getterm
```

```
681:         or $10,-48(%esp) #c_lflag |= (ICANON|ECHO)
682:         setterm
683:         sys_exit
684:
685: #Test the shape for any collision.  If collision, then the zero flag will
686: #NOT be set
687: collisionTest:
688:         call coords
689:         xor %eax, %eax
690:         mov $screen, %ebx
691:         collisionLoop
692: collisionTest_over:
693:         pop %dx
694:         ret
695:
696: #Writes the number of rotations of blockType into %cl
697: numberrots:
698:         xor %ebx, %ebx
699:         mov blockType, %bl
700:         test %bl,%bl
701:         jz numberrots_zeroshape
702:         add $shapeStarts, %ebx
703:         mov (%ebx), %cl
704:         sub -1(%ebx), %cl
705:         jmp numberrots_done
706: numberrots_zeroshape:
707:         mov shapeStarts, %cl
708: numberrots_done:
709:         ret
710:
711: _tetris:
712:         # [NM] the next 3 lines (2 macro calls and an andb) unset
713:         # canonical mode, so keyboard input is instant, i.e. no waiting
714:         # for the user to hit the Enter key; the echo is also unset
715:         getterm
716:         andb $245,-48(%esp) #c_lflag &= ~(ICANON|ECHO)
717:         setterm
718:
719:         sys_times
720:         mov %eax, lastrand #seed the randomizer
721:
722:         # [NM] in the old days, cursor movement on a terminal was done
723:         # by printing a certain sequence of bytes to the screen; it was
724:         # different from each brand/model of terminal, but eventually
725:         # Digital Equipment Corporation’s VT100 terminal type became a
726:         # standard, and today almost all terminal windows (e.g. xterm)
727:         # emulate a VT100 terminal
728:
729:         # [NM] the next 4 lines of code write the code for clearing (i.e.
730:         # blanking out) a VT100 screen; if you check earlier in the file,
731:         # you’ll see that vt100_clear is this:
732:
733:         # vt100_clear:
734:         #   .byte 0x1b
735:         #   .ascii "[2J"
736:
737:         # [NM] that first byte is the ASCII code for ESC (the Escape key),
738:         # which is a preface for all the VT100 cursor-movement codes; in
739:         # other words, to clear a VT100 screen, one sends ESC[2J to the
740:         # screen; try it yourself, by compiling and running this C
741:         # program:
742:
743:         # main()
744:         #
745:         # { char esc = 27;  // ASCII code for ESC
746:         #
747:         # printf("%c[2J",esc);
748:         #
749:         # }
750:
751:         # [NM] one more thing: the author does a write to the screen so
752:         # often (duh!) that he has collected the code to do so into a
753:         # subroutine, which he has named sys_write, but which is simply
754:         # a call to the usual OS function write(), OS call #4:
755:
756:         # sys_write:
757:         #       xor %eax, %eax
758:         #       mov $4, %al #sys_write
759:         #       xor %ebx, %ebx
760:         #       mov $1, %bl #stdout
761:         #       int $0x80
762:         #       ret
763:
764:         # [NM], so, here is the code to clear the screen:
765:         mov $vt100_clear, %ecx
766:         xor %edx, %edx
767:         mov $4, %dl
768:         call sys_write
769:
770:         # [NM] these lines initialize the cursor position
771:         mov $vt100_cursor, %ecx
772:         xor %edx, %edx
773:         mov $6, %dl
774:         call sys_write
775:
776: .if colour
777:         mov $vt100_colour, %ecx
778: .else
779:         mov $vt100_invert, %ecx
780: .endif
781:         xor %edx, %edx
782:         mov $5, %dl
783:         call sys_write
784:
785:         # [NM] in the old days, before VT100 really became standard,
786:         # there needed to be a way for people to write programs which
787:         # would work on any terminal type; for example, if you were
788:         # writing a text editor, like vi, you certainly would not want
789:         # to have to write a different version for each terminal type;
790:         # so, UNIX developers wrote the package named "curses" (get the
791:         # pun?); the programmer would simply call functions in this
792:         # package, and those functions would worry about how to make a
793:         # certain operation (e.g. cursor up one line) work; they would
794:         # do this by lookups in a database of all known terminal types
795:         # and their various cursor-movement codes, but the point is that
796:         # a programmer writing, say, vi could program cursor movements
797:         # without knowing what kind of terminal the user would use; the
798:         # author of Tetris here has written his own functions like that
799:         # (linking in curses from the C library would make the game too
800:         # big) and has even used the same macro names, e.g. mvaddstr
801:         # (see more comments on mvaddstr at its definition above); there
802:         # are lots of tutorials on ncurses on the Web
803:
804:         mvaddstr $instructionsY, $instructionsX, $namestring, $15
805:         mvaddstr $instructionsY+1, $instructionsX, $quitstring, $31
806:
807:         xor %al,%al
808:         mov $screen,%ebx
809: tetris_yloop:
810: .if colour
811:         movb $0x31, (%ebx) #red for the playing arena
812:         movb $0x31, width-1(%ebx)
813: .else
814:         movb $0xff, (%ebx)
815:         movb $0xff, width-1(%ebx)
816: .endif
```

```
817:         xor %ecx, %ecx
818:         mov $1, %cl
819: tetris_yloop_inner:
820: .if colour
821:                 movb $0x30, (%ebx, %ecx) #init to black
822: .else
823:                 movb $0, (%ebx, %ecx)
824: .endif
825:                 inc %cl
826:                 cmpb $width-1, %cl
827:                 jl tetris_yloop_inner
828:
829:         add $width, %ebx
830:         inc %al
831:         cmpb $height-1, %al
832:         jl tetris_yloop
833:
834:         mov $width*(height-1)+screen, %ebx
835:         xor %eax, %eax
836: tetris_xloop:
837: .if colour
838:         movb $0x31, (%eax,%ebx)
839: .else
840:         movb $0xff, (%eax,%ebx)
841: .endif
842:         inc %al
843:         cmpb $width, %al
844:         jl tetris_xloop
845:
846:         call redraw
847:
848: playgame:
849:         mov $6, %cl #7 shapes
850:         call rand
851:         movb %al, blockType
852:         call numberrots
853:         dec %cl
854:         call rand
855:         movb %al, rotation
856: .if colour
857:         mov $6, %cl
858:         call rand
859:         add $0x31, %al
860:         mov %al, currentcolour
861:         mov %al, %cl
862: .else
863:         movb $0xff, %cl
864: .endif
865:         call drawShape
866:         call collisionTest
867:         jnz gameover
868:
869: playgame_keyloop:
870:         sys_nanosleep $250000
871:         incb sleepcount
872:         movb currentwait, %cl
873:         cmpb %cl, sleepcount
874:         je playgame_slept
875:         sys_newselect
876:         test %eax, %eax
877:         jz playgame_keyloop
878: playgame_checkkey:
879:         xor %edx, %edx
880:         mov $1, %dl
881:         call sys_read
882:         cmpb $'q', buffer
883:         je gameover
884: .if colour
885:         mov $0x30, %cl
886: .else
887:         xor %cl, %cl
888: .endif
889:         call drawShape
890:         cmpb $' ', buffer
891:         jne playgame_checkarrow
892: playgame_droploop:
893:         incb yposition
894:         call collisionTest
895:         jz playgame_droploop
896:         addw $scoredrop, score
897:         jmp storePiece
898: playgame_checkarrow:
899:         cmpb $0x1b, buffer #check for arrow key
900:         jne playgame_checkdone
901:         xor %edx, %edx
902:         mov $2, %dl
903:         call sys_read
904:         movb buffer+1, %al
905:         cmpb $'D', %al #Left arrow
906:         je playgame_leftarrow
907:         cmpb $'C', %al #Right Arrow
908:         je playgame_rightarrow
909:         cmpb $'B', %al #Down Arrow
910:         je playgame_downarrow
911:         cmpb $'A', %al #Up Arrow
912:         jne playgame_checkdone
913:         xor %ah, %ah
914:         mov rotation, %al
915:         push %ax
916:         inc %al
917:         call numberrots
918:         divb %cl
919:         movb %ah, rotation
920:         call collisionTest
921:         jz playgame_checkdone
922:         pop %cx
923:         mov %cl, rotation
924:         jmp playgame_checkdone
925: playgame_leftarrow:
926:         decw xposition
927:         call collisionTest
928:         jz playgame_checkdone
929:         incw xposition
930:         jmp playgame_checkdone
931: playgame_rightarrow:
932:         incw xposition
933:         call collisionTest
934:         jz playgame_checkdone
935:         decw xposition
936:         jmp playgame_checkdone
937: playgame_downarrow:
938:         incb yposition
939:         call collisionTest
940:         jz playgame_checkdone
941:         decb yposition
942:         jmp playgame_checkdone
943:
944: playgame_slept:
945: .if colour
946:         mov $0x30, %cl #black to overwrite
947: .else
948:         xor %cl, %cl
949: .endif
950:         call drawShape
951:         incb yposition
952:         call collisionTest
```

```
  953:         jnz storePiece
  954:         movb $0, sleepcount
  955:
  956: playgame_checkdone:
  957: .if colour
  958:         movb currentcolour, %cl
  959: .else
  960:         movb $0xff, %cl
  961: .endif
  962:         call drawShape
  963:         jmp playgame_keyloop
```