#scoretetris: Number of points scored for a 'tetris' (ie eliminating 4 lines at once).

#speedup: The game will get twice as fast for every n lines.

#colour: Set this to 0 if you want to compile without colours (eg to play over Nifty Telnet on MacOS or the built-in telnet client on Windows)

```
.macro sys_newselect
Tetris.s Wed Mar 09 12:56:37 2005 1
         xor %eax, %eax #smaller than writing to %eax directly
         mov $142, %al #new sys_select
         xor %ebx, %ebx
         mov $1, %bl #highest is 0 (stdin), plus 1
         bts $0, -128(%esp)
         lea -128(%esp), %ecx #Take some stack for the fd_set struct
         xor %edx, %edx #writefds
         xor %esi, %esi #exceptfds
         xor %edi, %edi #ignore remainder
         int 0x80
.endm
```

Notes:
- Uses VT100 terminal codes to position the cursor and to draw coloured text.
- I also assume that this requires a > 2.0 Linux kernel which supports
  *sys_newselect* (also uses *sys_write*, *sys_read*, *sys_nanosleep*, *sys_exit*,
  *sys_times*) I have only tested it with 2.0.x, 2.2.x and 2.4.x
  *kernels. I'm pretty sure that this will work with 386+ processors.
- 
  - [NM] note that in the sample Makefile below, the entry point is specified
  - via the -e option to *ld*; note also the definition of various
  - quantities via *-defsym*; this is similar to a #define, but supplied
  - externally to calling "as", on the command line; GCC has the same
  - thing with -D

# Example Makefile:
```
NAME = tetris
ENTRY = _tetris
SYMS = -defsym instructionsX=12
SYMS = -defsym instructionsY=19
SYMS = -defsym width=10
SYMS = -defsym xoffset=3
SYMS = -defsym height=16
SYMS = -defsym yoffset=2
SYMS = -defsym wait=50
SYMS = -defsym scoredrop=2
SYMS = -defsym scorelockin=3
SYMS = -defsym scoreline=100
SYMS = -defsym scoretetris=1000
SYMS = -defsym speedup=10
SYMS = -defsym colour=1
SYMS = -defsym instructionsX, instructionsY: The offset of the instructions from the
first Esc, then [r;cH where r and c are given in 2-character string
of the top, left hand corner of the screen.

wait: Controls the speed of the game. Lower number equals faster play.

scorelockin: Number of points scored for 'locking' a piece in.

scoreline: Number of points scored for eliminating a line (for 1-3 lines)

# Explanation of the symbols which can be changed to suit personal taste:
# InstructionsX, InstructionsY: The offset of the instructions from the
top, left hand corner of the screen.
# width, height: The dimensions of the playing area
# xoffset, yoffset: The offset of the playing area from the top, left hand
corner of the screen.
# *scorelockin*: Number of points scored for 'locking' a piece in.
# *scoreline*: Number of points scored for eliminating a line (for 1-3 lines)
```

```
# format, e.g. the string "12" for row 12; that's a total of 8 bytes; 137: # the part of the .data segment labeled vt100_position contains these 8 138: # bytes; in order to move the cursor, we must fill in the r and c 139: # portions of these bytes, and then write the 8 bytes to the screen; the 140: # macro twodigits below converts r or c (it's called twice from 141: # mvaddstr, once for r and then for c) from a number to a 2-character 142: # string, and places it into vt100_position; the last two arguments 143: # specify where to write within vt100_position (for row or column); the 144: # inc instruction is apparently there because the VT100 scheme starts 145: # numbering at 1 instead of 0 146: 147: #Write the chars equivalent to 'source' into vt100_position 148: .macro twodigits source first second 149:         mov  %source, %ax 150:         inc %ax 151:         movb $10, %bl 152:         divb %bl 153:         add $0x30,%al 154:         movb %al, vt100_position+first 155:         add $0x30, %ah 156:         movb %ah, vt100_position+second 157: .endm 158: 159: # \[NM\] in the \[n\]curses package, the macro mvaddstr moves the cursor 160: # (thus the "mv" in the name) to row y, column x, and adds (thus 161: # "addstr") to the screen at that point, replacing what was there 162: 163: # before, and then advancing the cursor to the point just after the 164: # string that was written; in the variant here, the numbers y and x must 165: # be converted to strings, as explained in my comments for the macro 166: # twodigits 167: 168: #Named in honour of the n\[curses function 169: .macro mvaddstr y x string length 170:         twodigits y, 2, 3 171:         twodigits x, 5, 6 172:         mov $vt100_position, %ecx 173:         xor %edx, %edx 174:         add $0x30, %dx 175:         movb %al, (%eax, %ecx) 176:         .if colour 177:         cmpb $0x30, (%ebx, %eax) 178: .else 179:         cmpb $0x0, (%ebx, %eax) 180: .endif 181:         .if 4-from 182:                 jnz collisionTest_over 183:         .endif 184:         .if to-from 185:                 collisionLoop (from+1), to 186:         .endif 187: .endm 188: 189: #Make real use of gas macros 205: .macro macroLoop from=1, to=4 206:         storeLoop "\(from+1)", \to 207:         .endm 208: 209: # [NM] in the [n]curses package, the macro mvaddstr moves the cursor 210: # (thus the "mv" in the name) to row y, column x, and adds (thus 211: # "addstr") to the screen at that point, replacing what was there 212: 213: # before, and then advancing the cursor to the point just after the 214: # string that was written; in the variant here, the numbers y and x must 215: # be converted to strings, as explained in my comments for the macro 216: # twodigits 217: 218: #Named in honour of the n\[curses function 219: .macro mvaddstr y x string length 220:         twodigits y, 2, 3 221:         twodigits x, 5, 6 222:         mov $vt100_position, %ecx 223:         xor %edx, %edx 224:         add $0x30, %dx 225:         movb %al, (%eax, %ecx) 226:         .if colour 227:         cmpb $0x30, (%ebx, %eax) 228: .else 229:         cmpb $0x0, (%ebx, %eax) 230: .endif 231:         .if 4-from 232:                 jnz collisionTest_over 233:         .endif 234:         .if to-from 235:                 collisionLoop "\(from+1)", \to 236:         .endif 237:         .endm 238: 239: #Make real use of gas macros 240: .macro macroLoop from=1, to=4 241:         storeLoop "\(from+1)", \to 242:         .endm 243: 244: # [NM] in the [n]curses package, the macro mvaddstr moves the cursor 245: # (thus the "mv" in the name) to row y, column x, and adds (thus 246: # "addstr") to the screen at that point, replacing what was there 247: 248: # before, and then advancing the cursor to the point just after the 249: # string that was written; in the variant here, the numbers y and x must 250: # be converted to strings, as explained in my comments for the macro 251: # twodigits 252: 253: #Named in honour of the n\[curses function 254: .macro mvaddstr y x string length 255:         twodigits y, 2, 3 256:         twodigits x, 5, 6 257:         mov $vt100_position, %ecx 258:         xor %edx, %edx 259:         add $0x30, %dx 260:         movb %al, (%eax, %ecx) 261:         .if colour 262:         cmpb $0x30, (%ebx, %eax) 263: .else 264:         cmpb $0x0, (%ebx, %eax) 265: .endif 266:         .if 4-from 267:                 jnz collisionTest_over 268:         .endif 269:         .if to-from 270:                 collisionLoop "\(from+1)", \to 271:         .endif 272: .endm 273: 274: # [NM] in the [n]curses package, the macro mvaddstr moves the cursor 275: # (thus the "mv" in the name) to row y, column x, and adds (thus 276: # "addstr") to the screen at that point, replacing what was there 277: 278: # before, and then advancing the cursor to the point just after the 279: # string that was written; in the variant here, the numbers y and x must 280: # be converted to strings, as explained in my comments for the macro 281: # twodigits 282: 283: #Named in honour of the n\[curses function 284: .macro mvaddstr y x string length 285:         twodigits y, 2, 3 286:         twodigits x, 5, 6 287:         mov $vt100_position, %ecx 288:         xor %edx, %edx 289:         add $0x30, %dx 290:         movb %al, (%eax, %ecx) 291:         .if colour 292:         cmpb $0x30, (%ebx, %eax) 293: .else 294:         cmpb $0x0, (%ebx, %eax) 295: .endif 296:         .if 4-from 297:                 jnz collisionTest_over 298:         .endif 299:         .if to-from 300:                 collisionLoop "\(from+1)", \to 301:         .endif 302: .endm
.ascii "  
.exitstring:
.ascii "User exitted"
newline:
.ascii "\n"
loserstring:
.ascii "Loser
creditstring:
.ascii "Tetris in 3k, by dburr@ug.cs.usyd.edu.au"
score:
.hword 0
timeout:
.long 0
.vt100_position:
.byte 0x1b
.ascii "[12;13H"
.if colour
.vt100_colour: #The proper english way of spelling the word!
.byte 0x1b
.ascii "[44m"
.else
.vt100_invert:
.byte 0x1b
.ascii "[07m"
endif
.vt100_clear:
.byte 0x1b
.ascii "[2J"
.vt100_cursor:
.byte 0x1b
.ascii "[?25l"
yposition:
.byte 0
xposition:
.hword 2
.sleepcount:
.byte 0
.shapeStarts:
.byte 2, 3, 5, 7, 11, 15, 19
.shapeIndex: #This data contains the positions of the blocks in each shape

@.hword 0b0000110100000001 #0<<14|1<<12|2<<10|2<<8|0<<6|0<<4|0<<2 1
@.hword 0b0000100000000100 #0<<14|1<<12|0<<10|0<<8|0<<6|0<<4|0<<2 2
@.hword 0b0000001100000101 #0<<14|1<<12|2<<10|2<<8|0<<6|0<<4|1<<2 1
@.hword 0b0000101000000110 #1<<14|1<<12|1<<10|0<<8|0<<6|0<<4|2<<2 2
@.hword 0b000000110100000001 #0<<14|1<<12|2<<10|2<<8|0<<6|0<<4|1<<2 2
@.hword 0b0000000010000101 #0<<14|0<<12|1<<10|1<<8|0<<6|1<<4|0<<2 1
@.hword 0b0000000010000110 #0<<14|0<<12|1<<10|1<<8|0<<6|0<<4|2<<2 2
@.hword 0b0000000000000001 #0<<14|0<<12|1<<10|0<<8|0<<6|0<<4|0<<2 1
@.hword 0b0000000000000010 #0<<14|0<<12|1<<10|0<<8|0<<6|1<<4|0<<2 2
@.hword 0b0000000000000001 #0<<14|0<<12|1<<10|1<<8|0<<6|1<<4|0<<2 1
@.hword 0b0000000000000001 #0<<14|0<<12|1<<10|0<<8|0<<6|0<<4|0<<2 1
@.hword 0b0000000000000001 #0<<14|0<<12|1<<10|0<<8|0<<6|0<<4|0<<2 1
@.hword 0b0000000000000101 #0<<14|0<<12|1<<10|2<<8|1<<6|0<<4|0<<2 1
@.hword 0b0000000000000010 #0<<14|0<<12|1<<10|0<<8|0<<6|0<<4|2<<2 2
@.hword 0b0000000000000001 #0<<14|0<<12|1<<10|0<<8|0<<6|0<<4|0<<2 1
@.hword 0b0000000000000110 #0<<14|0<<12|1<<10|0<<8|0<<6|0<<4|2<<2 2
@.hword 0b0000000000000101 #0<<14|0<<12|1<<10|0<<8|0<<6|0<<4|1<<2 1
@.hword 0b0000000000000010 #0<<14|0<<12|1<<10|1<<8|0<<6|1<<4|0<<2 1
@.hword 0b0000000000000001 #0<<14|0<<12|1<<10|0<<8|0<<6|1<<4|0<<2 1
@.hword 0b0000000000000001 #0<<14|0<<12|1<<10|0<<8|0<<6|0<<4|0<<2 1
@.hword 0b0000000000000010 #0<<14|0<<12|1<<10|0<<8|0<<6|0<<4|1<<2 1
@.hword 0b0000000000000001 #0<<14|0<<12|1<<10|0<<8|0<<6|0<<4|0<<2 1

.linesgone:
.hword 0 #number of lines eliminated so far in the game
.currentwait:
.byte wait #gets smaller as the game gets faster
.buffer:
.byte 0, 0 #for arrow keys we read two
.screen:
.fill width*height
.lastrand:
.long 0
.globl _tetris
.text
Return a 4-bit number in %al that is no greater than %cl
.rand:
movl lastrand, %eax
mov %eax, %ebx
imul $1664525, %eax
add $1013904223, %eax
shr $10, %eax
xor %ebx, %eax
movl %eax, lastrand
andb $0x7, %al
cmp %al, %cl
jb rand
ret
.sys_write:
oxor %ebx, %eax
mov %eax, %ebx
add $013904223, %eax
shr $10, %eax
xor %ebx, %eax
movl %eax, lastrand
andb $0x7, %al
cmp %al, %cl
jn rand
ret
#Requires the string to write in %ecx, length in %edx
.sys_write:
xor %eax, %eax
mov $%, %al
sys_write:
xor %eax, %eax
mov $4, %al
sys_write:
mov $4, %al
sys_write:
409:         #Requires the length to read in %edx
410:         sys_read:
411:             xor %eax, %eax
412:             mov $3, %al #sys_read
413:             xor %ebx, %ebx #fd stdin
414:             mov $buffer, %ecx #buffer
415:             int $0x80
416:             ret
417:418:         coords:
419:             xor %edx, %edx
420:             xor %eax, %eax
421:             mov blockType, %al
422:             test %al, %al
423:             jz coords_noIndex
424:             mov shapeStarts-1(%eax), %dl
425: coords_noIndex:
426:                 add rotation, %dl #because each entry is 2 bytes
427:                 pop %eax
428:                 pushw shapeIndex(%edx)
429:                 jmp *%eax
430:431:         storeLoop
432:         #Write the block into the screen array at xposition, yposition
433:         #There are 4 squares in the current piece. Test the lines which these
434:         #occupy to see if they are part of a complete line. If so, remove, redraw
435:         #Also adds to the score and speeds the game up if necessary
436:         elimline:
437:             mov yposition, %dl
438:             xor %eax, %eax
439:             mov %dl, %al #%al will contain the ypositions to test
440:             add $4, %dl
441:             cmpb $height-1, %dl
442:             jl elimline_skip
443:             mov $height-1, %dl #%dl contains one more than the last value to test
444: elimline_skip:
445:             xor %ebx, %ebx
446:             mov $width, %bl
447:             imul %eax, %ebx
448:             add $screen, %ebx #ebx contains the start of the line
449:             xor %ecx, %ecx
450:             elimline_test:
451:                 inc %cl #%ecx contains the x position to test
452:                 cmpb $0x30, (%ecx, %ebx) #test this for each position in line
453:                 cmpb $0, (%ecx, %ebx) #test this for each position in line
454: elimline_end:
455:                 je elimline_linedone #ie: don't eliminate this line
456:                 cmpb $width-2, %cl
457:                 jne elimline_test
458:                 inc %dh
459:                 add $width, %ebx
460:                 elimline_loop:
461:                 dec %ebx
462:                 movb -width(%ebx), %cl
463:                 movb %cl, (%ebx)
464:                 cmp $screen+width, %ebx
465:                 jne elimline_loop
466: elimline_linedone:
467:                 inc %al
468:                 cmp %al, %dl
469:                 jne elimline_skip
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 $4, %dl
478:                 mov $kdh, %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 %ecx
502:                 movb -width(%ecx), %cl
503:                 movb %cl, (%ecx)
504:                 cmp $screen+width, %ecx
505:                 jne elimline_loop
506: elimline_linedone:
507:                 inc %al
508:                 cmp $1, %al
509:                 jne elimline_skip
510:                 mov $speedup, %bl
511:                 add $4, %dl #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:                 mov linesgone, %ax
523:                 mov $speedup, %bl
524:                 div %bl
525:                 mov $1, %dl
526:                 addw %dx, linesgone
527:                 mov linesgone, %ax
528:                 div %bl
529:                 cmp $1, %dl
530:                 je elimline_tetris
531:                 shrb $1, currentwait
532: elimline_samespeed:
533:                 test %cl, %cl
534:                 jz elimline_noredraw
535:                 call redraw
536: elimline_noredraw:
537:                 mov $0, %position
538:                 mov $2, xposition
539:                 mov $0, sleepcount
540:                 jmp playgame
541:542: #Draw the current blockType at xposition,yposition (offset from xoffset, yoffset). Will be coloured depending on %cl. Update score
543: 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: #Write the number in %ax
566: numbertostring:
567: mov $10, %bx
568: mov $stringbuffer+5, %ecx
569: numbertostring_loop:
570: dec %ecx
571: xor %dx, %dx
572: div %bx
573: add $0x30, %dl
574: movb %dl, (%ecx)
575: test %ax, %ax
576: jnz numbertostring_loop
577: mov $stringbuffer+5, %edx
578: sub %ecx, %edx
579: call sys_write
580: call sys_write:
581: ret
582: #Requires the y coord in %al, x coord in %bx, val to colour in %cl
583: drawBlock:
584: xor %ah, %ah
585: add $0x30, %dx
586: add $0x30, %dx
587: push %ax
588: push %bx
589: .if colour
590: movb %cl, vt100_colour+3
591: mov $vt100_colour, %ecx
592: .else
593: test %cl, %cl
594: jz drawblock_out
595: sub $0x30, %cl
596: .endif
597: drawblock:
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 %ecx
606: pop %edx
607: shl $1, %ecx
608: mvaddstr %ax, %ecx, $blankstring, $2
609: ret
610: #Redraw the playing area (doesn’t update score)
611: redraw:
612: 563: inc %ax
613: xor %edx, %edx
614: jmp gameover:
615: # [NM] here we must restore the screen, and especially the keyboard; to
616: see why, try playing the game but killing it with ctrl-C; you’ll see
617: # that the keyboard suddenly becomes inoperable in that window (to
618: # restore it, hit ctrl-j then “reset” then ctrl-j again)
619: #Restore the screen
620: #Requires the y coord in %ax, x coord in %bx, val to colour in %cl
621: #Redraw the playing area (doesn’t update score)
# more thing: the author does a write to the screen so
# often (duh!) that he has collected the code to do so into a
# subroutine, which he has named sys_write, but which is simply
# a call to the usual OS function write(), OS call #4:

sys_write:
  #       xor %eax, %eax
  #       mov $4, %al #sys_write
  #       xor %ebx, %ebx
  #       mov $1, %bl #stdout
  #       int $0x80
  #       ret

# Test the shape for any collision. If collision, then the zero flag will
# [NM], so, here is the code to clear the screen:

mov $vt100_clear, %ecx
xor %edx, %edx
or $10,-48(%esp) #c_lflag |= (ICANON|ECHO)
collisionTest:
  xor %edx, %edx
  call coords
  mov $4, %dl
  call sys_write
  mov $screen, %ebx

# these lines initialize the cursor position
mov $vt100_cursor, %ecx
collisionLoop
  xor %edx, %edx
  pop %dx
  mov $6, %dl
  ret

# [NM] in the old days, before VT100 really became standard,
# would work on any terminal type; for example, if you were
# writing a text editor, like vi, you certainly would not want
# to have to write a different version for each terminal type;
# so, UNIX developers wrote the package named "curses" (get the
# pun?); the programmer would simply call functions in this
# package, and those functions would worry about how to make a
# certain operation (e.g. cursor up one line) work; they would
# do this by lookups in a database of all known terminal types
# and their various cursor-movement codes, but the point is that
# a programmer writing, say, vi could program cursor movements
# without knowing what kind of terminal the user would use;
# the author of Tetris here has written his own functions like that
# (linking in curses from the C library would make the game too
# big) and has even used the same macro names, e.g. mvaddstr
# Digital Equipment Corporation's VT100 terminal type became a
# standard, and today almost all terminal windows (e.g. xterm)
# emulate a VT100 terminal

# [NM] the next 4 lines of code write the code for clearing (i.e.
# blanking out) a VT100 screen; if you check earlier in the file,
# you'll see that vt100_clear is this:

mov $screen,%ebx
tetris_yloop:
  movb $0x31, (%ebx) #red for the playing arena
  movb $0xff, width-1(%ebx)
  ;byte 0x1b
  movb $0x31, width-1(%ebx)
  .ascii "[2J"
  .byte 0xff
  .byte 0xff
  .byte 0xff
  .byte 0xff
  .byte 0xff
  .byte 0xff
  .byte 0xff

# [NM] the next 3 lines (2 macro calls and an andb) unset
# canonical mode, so keyboard input is instant, i.e. no waiting
# for the user to hit the Enter key; the echo is also unset

setterm
andb $245,-48(%esp) #c_lflag &= ~ICANON|ECHO
setterm

# of tutorials on ncurses on the Web
# there needed to be a way for people to write programs which
# would work on any terminal type; for example, if you were
# linking in curses from the C library would make the game too
# big) and has even used the same macro names, e.g. mvaddstr
mvaddstr $instructionsY, $instructionsX, $namestring, $15
mvaddstr $instructionsY+1, $instructionsX, $quitstring, $31
xor %al,%al

# [NM] in the old days, cursor movement on a terminal was done
# by printing a certain sequence of bytes to the screen; it was
# different from each brand/model of terminal, but eventually
# Digital Equipment Corporation's VT100 terminal type became a
# standard, and today almost all terminal windows (e.g. xterm)
# emulate a VT100 terminal

# [NM] that first byte is the ASCII code for ESC (the Escape key),
# which is a preface for all the VT100 cursor-movement codes; in
# other words, to clear a VT100 screen, one sends ESC[2J to the
# screen; try it yourself, by compiling and running this C
# program:

mv $4, %dl
ret

sys_exit

or $10,-48(%esp) #c_lflag |= (ICANON|ECHO)
setterm
xor %eax, %eax
mov $4, %al #sys_write
xor %eax, %eax
mov $4, %dl
int $0x80
ret

mov $1, %bl #stdout
int 0x80
ret

mov $vt100_clear, %ecx
xor %edx, %edx
or $10,-48(%esp) #c_lflag |= (ICANON|ECHO)
collisionTest:
  xor %edx, %edx
  call coords
  mov $4, %dl
  call sys_write
  mov $screen, %ebx

# these lines initialize the cursor position
mov $vt100_cursor, %ecx
xor %edx, %edx
collisionLoop
  xor %edx, %edx
  pop %dx
  mov $6, %dl
  ret

# [NM] in the old days, before VT100 really became standard,
# would work on any terminal type; for example, if you were
# writing a text editor, like vi, you certainly would not want
# to have to write a different version for each terminal type;
# so, UNIX developers wrote the package named "curses" (get the
# pun?); the programmer would simply call functions in this
# package, and those functions would worry about how to make a
# certain operation (e.g. cursor up one line) work; they would
# do this by lookups in a database of all known terminal types
# and their various cursor-movement codes, but the point is that
# a programmer writing, say, vi could program cursor movements
# without knowing what kind of terminal the user would use;
# the author of Tetris here has written his own functions like that
# (linking in curses from the C library would make the game too
# big) and has even used the same macro names, e.g. mvaddstr
mvaddstr $instructionsY, $instructionsX, $namestring, $15
mvaddstr $instructionsY+1, $instructionsX, $quitstring, $31
xor %al,%al
mov $screen,%ebx
mov $0x31, (%ebx) #red for the playing arena
mov $0x31, width-1(%ebx)

Tetris.s

; Move $1, %cl
; yloop_inner:
.if colour
    movb $0x30, (%ebx, %ecx) #init to black
.else
    movb $0, (%ebx, %ecx)
.endif
    inc %cl
    cmpb $width-1, %cl
    jl yloop_inner
.add $width, %ebx
    inc %al
    cmpb $height-1, %al
    jl yloop
.add $width*(height-1)+screen, %ebx
    xor %eax, %eax
    xloop:
.if colour
    movb $0x31, (%eax,%ebx)
.else
    movb $0xff, (%eax,%ebx)
.endif
    inc %al
    cmpb $width, %al
    jl xloop
.call redraw
.playgame:
    mov $6, %cl  #7 shapes
    call rand
    movb %al, blockType
    call numberrots
    dec %cl
    call rand
    movb %al, rotation
.if colour
    mov $6, %cl
    call rand
    add $0x31, %al
    movb %al, currentcolour
    mov %al, %cl
.else
    movb $0xff, %cl
.endif
    call drawShape
    call collisionTest
    jnz gameover
.playgame_keyloop:
    sys_nanosleep $250000
    incb sleepcount
    movb $0x31, %al
    mov $al, currentcolour
    mov %al, %cl
.endif
    call drawShape
    incb yposition
    call collisionTest
.playgame_xloop:
.if colour
    mov $0x31, (%eax,%ebx) #init to black
.else
    xor %cl, %cl
.endif
    call drawShape
    incb yposition
    call collisionTest
.endif
    mov $6, %cl  #7 shapes
    call rand
    movb %al, blockType
    call numberrots
    dec %cl
    call rand
    movb %al, rotation
.if colour
    mov $6, %cl
    call rand
    add $0x31, %al
    movb %al, currentcolour
    mov %al, %cl
.else
    movb $0xff, %cl
.endif
    call drawShape
    call collisionTest
    jnz gameover
.menu:
    xor %edx, %edx
    mov $1, %dl
    call sys_read
    cmpb $'q', buffer
    je gameover
.if colour
    mov $0x30, %cl
.else
    xor %cl, %cl
.endif
    call drawShape
    cmpb $' ', buffer
    jne menu_checkarrow
    jmp move_rotation
.menu_checkarrow:
    cmpb $0x1b, buffer #check for arrow key
    jne menu_checkdone
    xor %edx, %edx
    mov rotation, %al
    push %ax
    inc %al
    call numberrots
    divb %cl
    movb %ah, rotation
    pop %cx
    mov %cl, rotation
    jmp menu_checkdone
.menu_leftarrow:
    decw xposition
    call collisionTest
    jz menu_checkdone
    incw xposition
    jmp menu_checkdone
.menu_rightarrow:
    incw xposition
    call collisionTest
    jz menu_checkdone
    decw xposition
    jmp menu_checkdone
.menu_downarrow:
    incb yposition
    call collisionTest
    jz menu_checkdone
    decb yposition
    jmp menu_checkdone
.menu_slept:
    .if colour
        mov $0x30, %cl #black to overwrite
.else
        xor %cl, %cl
.endif
        call drawShape
        incb yposition
        call collisionTest
951:         jnz storePiece
952:         movb $0, sleepcount
953:         movb $0, sleepcount
954:         movb $0, sleepcount
955:         movb $0, sleepcount
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