UNATRON main loop

SETUP $24  Set page "0" to $24
ADDCHQ ECU  $2FA2  Subroutine for adding another character to the character list or "c-list".
ANTISH ECU  $3088  Subroutine for erasing a char. from the screen at its current location.
NEWLOC ECU  $3074  Subroutine for computing char's new loc from current loc and char's vector.
NYTSET ECU  $3017  Subroutine for moving cursor from shape data when writing shape.
OKMOY ECU  $2FE3  Subroutine for checking to see if proposed new loc for char is already occupied.
REALCO ECU  $2FCB  Subroutine for translating screen loc into memory loc and bit set.
VCOUNT ECU  $30F4  Subroutine for multiplying vector in VOUT by two.
WRITSH ECU  $30B1  Subroutine for writing shape - shape addr in STS;H, real loc in RLGC and RBIT.
BGVYSH ECU  $2F58  Subroutine for building restore shape for a write-restore operation.
DNVYEC ECU  $2EC1  Subroutine for generating vector from char to AIM loc on screen.
KEVYEC ECU  $2ED4  Subroutine for generating vector from char to players character.
RNDEVE ECU  $2F13  Subroutine for generating a random vector.
SHPADR ECU  $2EC5  Subroutine for looking at shape's addr from the shape table.
KBDWAII ECU  $2E71  Subroutine cycles and waits for fire button to be pushed.
CCISP ECU  $2EB8  Subroutine for counting the number of player's characters left.
TALLY ECU  $2CC4  TALLY displays scoring.
CREEC ECU  $2454  # of chain reactions in this round.
CCMH ECU  $2459  # of computer hits in this round.
YURM ECU  $2458  # of player's hits in this round.
TCREAC ECU  $2456  Total number of chain reactions since start of game.
TCOMH ECU  $2454  Total number of computer hits since start of game.
TCRMS ECU  $2452  Total number of player's hits since start of game.
ATRCI ECU  $2450  Parameter controls strength of CURC's attraction to AIM location.
MENL ECU  $2451  # of player's characters left in game.
ETHAI ECU  $2450  Wait counter to slow game down near the end of round.
PSCL ECU  $2447  LSB of proposed screen location.
PSCLR ECU  $2446  Screen coordinate, real value of a point on screen; $128 by Y + X.
RBIT ECU  $2443  Bit set; selects pixel 2, 3 or 4 in a byte defined by RLOC.
RLOC ECU  $2444  Actual byte location of pixel on the screen.
STSH ECU  $2433  Where addr of the start of a shape is stored after calling SHPADR.
TMP2 ECU  $2431  Work storage.
NUMN ECU  $2410  # Number of neutrons released when a mine is exploded.
TLOC ECU  $2441  Working temporary for variable RLGC.
TBIT ECU  $2440  Working temporary for variable REIT.
VOUT ECU  $2430  LSB of vector generated by vector generating routines.
VCUT ECU  $243C  Vector generated by vector generating routines.
RMPK ECU  $2438  Proposed Shape; where # of shapes to be ADDCHQed is stored.
TPLY ECU  $2432  Work storage.
CURC ECU  $2434  Shape # of atom currently reactive.
RAWY ECU  $2439  # a non-zero value here will cause CURC to run away from player's char.
TEMX ECU  $2433  Temp storage usually used for X register.
TVEC ECU  $2437  Temp vector storage.
STVE ECU  $2435  Temp vector storage.
RDN0 ECU  $242F  Working storage.
RMP3 ECU  $2430  Working storage.
STBS ECU  $2402  # Addr of start of screen borders layout.
RDN2 ECU  $242E  Working storage.
RDN3 ECU  $2429  Working storage.
RDN4 ECU  $242C  Working storage.
RDN5 ECU  $2428  Working storage.
RDN6 ECU  $242B  Number of holes, makers to appear on the screen this round.
MAXH ECU  $241A  # Number of holes, makers currently on the screen.
NUMM ECU  $242A  Number of holes, makers currently on the screen.
NUMM ECU  $2429  Number of mines currently on the screen.
MAXM ECU  $2415  # Max # of mines that can appear on screen this round.
MANST ECU $2404 * Screen loc of where player's character starts.
SPST ECU $2406 * Screen loc of where first atom starts
SETPTR ECU $2400 * Addr of next set of overlaid data.
MSCR ECU $2413 * Screen loc of where mine's appear.
MNFH ECU $2419 * Number of mines that must appear before a hole appears.
MINH ECU $2416 * Relative chance of a mine appearing.
MINPL ECU $2417 * The shortest number of cycles a mine will wait before chasing player.
MINSF ECU $2418 * The longest number of cycles a mine will wait before chasing player.
MINR ECU $244F Real memory addr where a mine will appear.
MINB ECU $244E Bit set where a mine will appear.
GCM ECU $2412 * Relative chance of computer gun firing.
G2L ECU $2411 * Length of shot originating from gun #2.
G2V ECU $240F * Vector for shot originating from gun #2.
G2S ECU $240D * Screen loc of where gun #2 appears.
G2R ECU $244C Real memory addr of where gun #2 appears.
G1L ECU $244B * Bit set of G1L gun that appears.
G1V ECU $240C * Length of shot originating from gun #1.
G1R ECU $244A * Vector for shot originating from gun #1.
G1S ECU $2408 * Screen loc of where gun #1 appears.
G1B ECU $2448 * Bit set of where gun #1 appears.
ORG $2500 Main routine starts here.

START LDA $24 Value for direct page register.
TFR A,DP Set direct page register.
STA 65478 "0" setting video page.
STA 65481 "1"
STA 65482 "0"
STA 65483 "1"
STA 65487 "1"
STA 65488 "0"
STA 65472 "0" setting video mode G3C.
STA 65474 "C"
STA 65477 "1"
LDA $255 Value for video control register - gives black background and one of two color sets.
STA 65314 Poke the value into the video control register. Video parameters are not reset from here on in.
LCD $220C Hard coded address of the first set of data to be overlaid at from $2400 to $2429
LDA $201R Store the $220C so that the first set of overlaid data is read starting from that point.
LDS $27FF Going to move the hole out of the way. No data has been stored below $800.
LDA $04 Give the player four characters to spend. This will be incremented to five as soon as score is put up.
STA MENL Store it.
LDD $0 Zero out D register and use it to zero out point totals.
STD TYNM Player's (your) total number of hits is assigned C.
STD TCMH Computer's total number of hits is assigned D.
STD TCREA Total number of chain reactions is assigned D.
CLR YURM Clear player's hits for this round.
CLR CREAC Clear number of chain reactions for this round.
CLR CCMH Clear number of computer's hits for this round.

* One time initialization of the game has been completed.
* You will retrieve $29 bytes of overlaid data starting at SETPTR.
* Data is written from $2400 to $2428. Note $24 has been set to
* page zero in the direct page register. Data in the $29 bytes
* controls the difficulty of this round, screen layout etc.

ZSTART LDX SETPTR Start of a round: Get addr of next set of overlaid data.
CLRA 
There are $29 bytes (some unused) of overlaid data. The A register is going to be used to count to $29.
LDY #124CG  ; Where overlaid data will go. The 529 bytes include all parameters which change the game from round to round
LDX #0    ; Get a byte of the data.
STY      ; Put the byte down.
INCX      ; Increment the counter.
CPA #529  ; Have all 529 bytes been transferred?
BEQ FP2   ; If so, branch out of this loop.
BRA FP1   ; A register still not equal to $29, continue loop.

; The character c-list starts at 12547 and extends to 13311.
; Video ram runs from 13312 to 16383. In one big sweep both
; will be initialized to zeros.

LDX #12547  ; This is the addr of the start of the character c-list. The display ram is adjacent to the c-list. Clear bot
CMPX #16383  ; Check to see if the end of the video ram has been reached.
BGE C201    ; If so, exit loop, Character c-list and video ram are cleared.
CLR $X+     ; Zero out byte, X points to next byte.
BRA C200    ; Continue clearing c-list and video ram.

JSL TALLY   ; Now things are happening! Put up scoreboard.
JSR KEDWAI  ; What for fire button to be pushed.

; The first item and the player are now set up. The screen locations
; where both will start are extracted from the data overlaid above
; and translated into real video ram locations and bit sets.
; The 1st item and the player are added to the character c-list.
; Note the player's character is done first. This assures the
; player will always hold the first position in the c-list.

LDA #56    ; This is the shape the player will start out with. It looks like this: V
STA PSHFP   ; Store it as the proposed shape so ADDCHC can pick it up.
JSL ADDCHC  ; Add player to c-list w/shape = 06, screen loc = PSCR, real loc = RLOC, and bit set = RBIT.
LDAarelco ; Get screen address of where atom #1 starts.
STA PSCR    ; Store it as proposed screen location.
JSL REALCO  ; Translate into real coordinates.
LDA #18    ; Shape #18 is the first atom in it's normal (as opposed to wobble) state.
STA PSHFP   ; Store it as proposed shape.
STA CURC    ; Store it as the current character - the one the player is chasing, the one that can fission.
JSL ADDCHC  ; Add the character to the c-list.

; For the rest of the round the computer guns will be firing and
; mines will appear. The screen positions where these originate
; are read from the overlaid data section, translated into video
; ram locations and bit sets and stores so they can be fetched
; whenever a mine is to be born or a gun is to fire. Calculating
; this information once at the beginning of a round saves time later.

LDA PSCR    ; Get the address of the screen location where the mines are born.
STA PSCR    ; Store it as proposed screen location.
JSL REALCO  ; Translate the screen location into a video ram location and bit set.
LDA RLOC    ; Get the video ram location.
STA MNFR    ; Store it here. It will stay here and need not be recalculated whenever a mine is born.
LDA RBIT    ; Get the bit set just calculated.
STA MNFB    ; Store it so it need not be recalculated again this round.
LCD G1S Get the screen location where gun #1 appears.
STO PSCR Store as proposed location.
JSP REALCO Translate the screen location into a video ram location and bit set.
LCD RLOC Get the video ram location.
STO G1R Store it so it need not be recalculated this round.
LDA REIT Get the bit set.
STA G1B Store it so it need not be recalculated.
LDO G2S Get the screen location where gun #2 will appear.
STO PSCR Store it as proposed location so REALCO can get it.
JSP REALCO Translate to real coordinates.
LDO RLOC Get the video ram address just generated.
LDA REIT Get the bit set.
STA G2B Store it so it need not be recalculated.

CLR EMAI Clear the end of round wait counter so that the game runs at normal speed.
CLR NNUM Clear the number of hokkers counter (because there are none of course).
LDA #07 An attraction parameter. The lower the value, the harder CURC will try to reach AIM; the screen location.
STA ATRCT #07 is a moderate value, it is lowered later so that the end of the round isn’t spent chasing 3 or 4 dots.

The round has been set up. All actions from here in take place inside a given round.

RESTART CLR NNUM Every time the player is hit by a mine, or whenever a new round starts, mines are deleted from the c-list.
C202 LDX #13312 Load X register with the start of the video ram. The screen is going to be cleared.
CLPX #66383 Check to see if the screen has been cleared.
BGT XXX If it hasn’t, branch out of loop.
CLR /X+ Clear a byte of video ram and increment X.
BRA C202 Continue with the loop.

The border layout will now be drawn on the screen.
The layout is made up of two shapes, #108 and #110.
In the first two screens shape #108 is a horizontal brick and
#110 is a vertical brick. The location STBO points to the start
of a list of screen locations where the shapes are to be drawn.
Shape #108 is drawn at all addresses plucked from the list
until a negative address is encountered. Shape #110 is then drawn
at the next bunch of addresses plucked from the list until
a second negative number is encountered.

XXX LDA #108 The screen layout is made of two shapes. #108 is the first. The shape’s address must be looked up.
LDA SHPADR Shape number in A is address of the shape is returned to STSH.
JSP STBD The screen address for each piece of the layout (borders) is in a table pointed to by STSH.
BGT XXX2 Get two bytes from the table. Increment X two bytes.
BRA XXX8 If the word loaded into D is < C, this signals that all occurrences (if any) of #108 have been seen. Branch.
STO PSCR Store the bytes as a proposed screen address.
JSP REALCO Translate the address into real coordinates.
JSP WRTHSP write the shape pointed to by STSH in video ram at RLOC with bit set RBIT.
BRA XXX8 Continue stepping through the table.

XXX LDA #110 This is the second shape in the screen layout.
JSP SHPADR The shape addr is looked up repetitively. Wasteful, but time is cheap at this point.
LDA /X+ Get the screen addr of the shape to be put up. X points to next shape.
BGT XXX4 If the screen addr located into D < 0 (i.e., end of data) exit loop.
STO PSCR Store the addr as proposed screen location.
JSP REALCO Translate it into real coordinates.
JSP WRTHSP write shape pointed to by STSH at RLOC with bit set RBIT.
BRA XXX8 Continue looping through layout table.
* The mines and chain reaction neutrons are deleted from the screen.
* This saves the player from repeatedly being attacked by the same
  mine or from losing too many points to chain reactions. If this
  is the first pass through this section this round then there
  won't be any mines or neutrons in the c-list anyway. If it is not
  the first pass through, the player has just been hit by a mine.

X4X4 LCX #12547 Set X to point to start of c-list. Will loop through and delete mines and neutrons.

X4X5 CMPX #13312 Check to see if at the end of the character c-list.

BGE LCP If so branch out of this loop to main LOOP.

LCA X Get the shape number of the character from the c-list.

CMPA #42 Check to see if its a neutron from chain reaction.

BEQ XXXX If it is, branch below where it will be deleted.

CMPPA #106 Check to see if character from c-list is a mine.

BEQ XXXX If it is, branch below where it will be deleted.

BRA XXXX It is neither mine nor neutron. Branch below but skip the deletion part of the deal.

X4X7 CLR X The mine or neutron is deleted.

X4X6 LEAX 9/X The X register is increased by 9 to point to the next character in the c-list.

BRA XXXX Branch back up and continue looking for mines and neutrons.

* This completes the screen setup. The rest is normal operation of the game.

* This is the main loop of the game.

* First we will check the player's joystick pots and decide
  1) What vector to give the player's character (49 possibilities).
  2) What the player's shot will take if fired.
  3) What the player's character will look like based on the
     direction it is moving.

* The player can move at three speeds in a given direction. The speed
  is determined by how displaced the joystick is from the center.

* The player's shot vector is also prepared. It is twice the player's
  vector when the player's vector is non-zero. Otherwise it is left
  unmodified.

LOOP JSR [$ADD] Simple the joystick pots with ROM routine whose address is at $ADD.

LCX #12547 Load the X register with the start of the c-list. Player's character ALWAYS occupies the first position.

CLR $FF20 The d/a converter is at $FF20. It has just been used in sampling the joysticks and must be cleared for sound

LCA #1BC Load A with value for routine output from d/a converter to TV sound modulator.

STA $FF23 Store it at this PIA.

LDC #C The player's vector and shape and shot vector are now going to be decided.

STO TYEC Clear temporary vector location.

STO VCUT Clear vector location.

STP PSHP Set proposed shape # to C, values will be added to this to decide the final shape #.

CMPPA $158B Get vertical joystick reading. Check for upward movement first. There are three speeds each direction.

CMPPA #18 Up slowly/medium or fast - all less than or equal to 18 on joystick.

BGT C70 If greater than 18, check for downward movement indicated by joystick.

LDC #06 Shape for player's character facing up.

STP PSHP Store as proposed shape.

CMPPA #6 Joystick value of < 6 indicates quick upward movement.

BGE UT1 Value greater than or equal - check for medium speed, branch

LDO $FFDC It is used quickly. Vector = $FFDC = -256.

STO TYEC Store as temporary vector for player's character.

STD VCU For shot always twice player's character speed. Multiply by two.

STD VCUT Store as vector for player's shot.

BRA C71 Branch to check horizontal movement.
U01 CMPA #12 Medium speed indicated by a value less than 12 and greater than 5.
BGE LC2 If value is greater than or equal to 12 slow speed must be called for. Branch.
LDD #FF8C Vector for medium speed upward = FF80 = -128.
STD TVEC Store as temporary vector for player’s character.
CLRA VOUT Preparing vector for shot = twice player’s speed. Clearing B makes value = FF00 = -256.
STD VOUT Store as vector for player’s shot.
BRA C71 Branch to check for horizontal movement.
LDD #FF8C Slow speed. This will be vector for shot = -128.
STD VOUT Store as vector for player’s shot.
INC RND3 This location used as an odd/even counter for slow player vectors.
LDA RND3 Every other cycle player will move upward. Effectively half speed.
ANDA #C1 If non-zero result from this give player vector of -128, else leave = 0.
BEQ C71 Result was zero, branch.
LDD #FF80 Vector equal to -128.
STD TVEC Store as temporary vector for player’s character.
BRA C71 Go do horizontal component.
C70 CMPA #45 If ended up here there was no upward movement given to player. Check for downward.
BNE C71 Set downward movement to 1. Branch.
LDB #12 If joystick value < 45, no downward movement called for. Go check for horizontal.
STD PSHF Store the 12 as proposed shape number.
CMPA #57 A value greater than 57 calls for fast downward movement.
BNE UC3 If value is less than or equal, branch to check for medium speed.
LDD #256 Vector for fast downward movement.
STD TVEC Store as temporary vector for player’s character.
LSLA Multiply by two so that shot moves at 512.
STD VOUT Store shot’s vector as vector.
BRA C71 Branch to check for horizontal movement.
CMPA #51 Check for medium speed.
BLT UC4 A value less than 51 calls for slow speed. Branch.
LDD #128 Vector for medium speed.
STD TVEC Store as temporary vector for player’s character.
LDD #256 Vector for shot = twice player’s speed.
STD VOUT Store as vector for player’s shot.
BRA C71 Branch to check for horizontal movement.
LDC #128 Slow speed. 128 is vector for shot.
STD VOUT Store as vector for player’s shot.
INC RND3 Increment odd/even counter.
LDA RND3 Get odd/even counter.
ANDA #C1 Non-zero result here and character moves down with vector of 128, else no movement downward.
BEQ C71 If result is zero, go check for horizontal movement.
LDD #128 Result not = 0, give player vector for downward movement.
STD TVEC Store as vector for player’s character.
C71 LCA SDIFSA Horizontal movement section. Get result for joystick sample.
CMPA #18 A value greater than 18 indicates no leftward movement.
BGE C72 If no leftward movement, branch to check for rightward movement.
INC PSHF Adding two to player shape. If there was no vertical movement, PSHF will equal 2.
INC PSHF If there was upward movement, PSHF will = 8. If downward PSHF will = 14. Takes care of diagonal shapes.
CMPA #C6 Check to see if fast leftward movement is called for.
BGE UC5 As, then check for medium speed.
LDD TVEC Fast. Get player’s vector generated so far.
SUBA #C2 Add two pixel leftward displacement.
STD TVEC Store as temporary vector for player’s character.
LDC VOUT Get player’s shot vector generated so far.
SUBA #C4 Give twice player’s leftward displacement.
STD VOUT Store as vector for player’s shot.
BRA C71 Exit player vector generating section.

Urantron listing page: 06
W05 CMPA #12 Check for medium speed.
LDT UC6
LDC TVEC Medium speed. Get player's vector generated so far.
LDC #01 Give one pixel leftward displacement.
STD TVEC Store as temporary vector for player's character.
LDD VCUT Get player's shot vector generated so far.
SUBD #6 Give twice player's leftward displacement.
STD VCUT Store as vector for player's shot.
BRA C73 Exit vector generating section.
LDD VCUT If get here, slow speed is called for. Get shot vector previously calculated.
SUBD #01 Give leftward displacement.
STD VCUT Store as vector for player's shot.
INC RND2 Increment horizontal odd/even counter.
LDA RND2 Get odd/even counter.
ANDA #01 If result here = 0, no movement this cycle.
BEQ C73 Branch out of vector generating section if result = 0.
LDC TVEC Get player's vector generated so far.
SUBD #01 Give leftward displacement.
STD TVEC Store as vector for player's character.
BRA C73 Exit vector generating section.
C72 CMPA #45 If get here, no leftward movement called for, check for rightward movement. Value < 45 - no horiz movement.
BLE C73 No rightward movement called for. Branch and exit vector generating section.
LOD H Multiply 04 and old shape gives shape facing correct direction.
ADD Ph Add to shape already stored away.
STD PSM Store as proposed shape.
CMPA #57 Check for fast rightward motion.
BLE UC7 Value less than 58, no fast movement. Branch to check for medium speed.
LDC TVEC Fast rightward movement. Get player's vector already generated.
ADDH #6 Add two pixel rightward displacement.
STD TVEC Store as temporary vector for player's character.
LDD VCUT Get player's shot vector generated so far.
ADD #6 Add twice player's displacement.
STD VCUT Store as vector for player's shot.
BRA C73 Exit vector generating section.
W07 CMPA #51 Check for medium rightward speed. Value < 51 implies medium slow speed.
BLT U08 If less than 51, branch and do slow speed.
LDC TVEC Get player's vector calculated previously.
ADDH #1 Give slow rightward displacement.
STD TVEC Store as temporary vector for player's character.
LDC VCUT Get player's shot vector calculated before.
ADD #01 Give twice player's rightward displacement.
STD VCUT Store as vector for player's shot.
BRA C73 Exit vector generating section.
U08 LDD VCUT Slow speed. Get player's shot vector previously calculated.
ADDH #1 Add rightward displacement.
STD VCUT Store as vector.
INC RND2 Increment horizontal odd/even counter.
LDA RND2 Get odd/even counter.
ANDA #01 If result from this = zero, no movement this cycle.
BEQ C73 Result = 0, branch.
LDC TVEC Result not = 0. Get player's vector calculated so far.
ADDH #01 Add rightward displacement.
STD TVEC Store as temporary vector for player's character.
LDC TVEC Get player's vector calculated above.
BEQ LLS If the joystick was in the middle, the vector = 0, player's character does not move. Branch.
STD 12554 Vector not = 0. Store in c-list where player's vector is always stored (7x1).
LDA VGET
  Get player's shot vector calculated above. Can be sure vector not = 0 because player's vector not = C.
STD $VGET
  Store shot vector here. Note: if player's vector had been zero, shot vector wouldn't be modified.
JST &X
  Get shape used for player in the last round.
JSR $YAPDR
  Look up the shape in the shape table.
JSR $NATISH
  Erase the shape from the screen.
JSR $NEWLOC
  Compute a new proposed location from the player's vector and old screen location.
JSR $REALLOC
  Translate the proposed location into real coordinates.
LCA $PSP
  Get shape number determined for player when vector was calculated.
JSR $YAPDR
  Look up the shape's starting address.
JSR $QKMOV
  Check to see if the new shape will fit at the new screen location.
BEQ C28
  If the new shape will fit, return code = 0, branch.

LCD $VGET
  Will not fit at the new loc. Generate random vector. Try to make player bounce off of whatever is in the wa
STD 12554
  Store in c-list at location where player's vector is.
JSR $NEWLOC
  Generate new proposed screen loc from the new vector.
JSR $REALLOC
  Translate into real coordinates.
JSR $QKMOV
  Check to see if the new location is unoccupied.
BEQ C28
  If the new location is free, it is ok to move. Branch.
LCD 12551
  Still can't move. Give up. Get player's real screen location (4/X).
STD RLOC
  Store as real screen loc so we can write the old shape back.
LCA 12553
  Get player's old bit set (6/X).
STA RBIT
  Store as bit set so we can write the old shape back.
LCA 12547
  Get shape number for player's previous character (6/X).
JSR $YAPDR
  Look up the shapes address.
JSR $WRTSP
  Write the old shape back. We have given up on the player this round. Cannot move him (or her).
C28
  Leave this section. Go check fire button.
JSR $WRTSP
  If get here was able to write new shape. Write the shape.
LDA $PSH
  Get the number of the shape just written.
STA 12547
  Store in c-list as player's shape (6/X).
LCD $PSR
  Get the new screen location generated and written on.
STD 12548
  Store in c-list as player's screen location (6/X).
LDD RLOC
  Get real value of location just written to.
STD 12551
  Store in c-list as player's real location (6/X).
LCA RBIT
  Set bit set of location just written to.
STA 12553
  Store in c-list as player's old bit set (6/X).
BRA SHOT
  Information recorded with player's character. Branch to check fire button.
LCA 12547
  If get here, joystick was in the middle. Shape not moved, but must rewrite in case a holemaker ste it.
JSR $YAPDR
  Look up player's old character shape address.
STD RLOC
  Store as real location.
LCA 12553
  Get player's old bit set.
STA RBIT
  Store as bit set.
JSR $WRTSP
  Write player's old shape back.

* The joystick button will be checked to see if a shot is to be fired

SHOT
LCA 65280
  Fire button is memory mapped. Read value.
CMPA #255
  If value = 255, button not pushed.
BEQ G01
  Branch if not pushed.
CMPA #127
  If value = 127, fire button not pushed.
BEQ CC1
  Branch if not pushed.
TST RD0S
  This is the button pushed last cycle flag. If it is true, button was already pushed.
BEQ LLM
  Branch if button already pushed.
JNC RD0S
  Set button pushed last cycle flag.
LDA $VGET
  Get vector calculated for player's shot.
STD WOUT
  Store as vector.
LDD 12548 Get player's screen location. Shot must originate from where player's character is.
STD PSCR Store as proposed screen location. RBIT and RLOC still hold player's location.
STA #28 Player's shot first shape number = 48. Shape number changes as shot proceeds.
STA PSCR Store as proposed shape.
LDA #31 This is a counter for shot shape changes. Every eight cycles shape of player's shot will change.
STA TIP Store in space normally reserved for wobble byte.
JSR ADDCHQ Add the player's shot to the character c-list.
BRA LLM Branch around line below.
CLR RAD5 This line merely resets the button pushed last round flag. Only get here if button was not pushed.

* This next section checks to see if a holemaker can be added
* to the c-list. The decision is based on the number of mines
* already in the c-list and the number of holemakers allowed
* vs the number already in the c-list.

LLM
LDA RAD1 Get "random" number.
 ANDA NLMM Take the number <= than the number of mines on the screen.
 CMPA #16FM Check against the number of mines necessary before a holemaker can appear.
 BGT TIP If the result is <= 0, there will be no holemakers born this round. Branch.
 LDA NLMM Get the number of holemakers already living.
 CMPA MAXM Compare it with the maximum number allowed.
 BEQ TIP If the maximum number of holemakers allowed already exits, branch.
 LDA #102 Shape number for a holemaker.
 STA PSHP Store as proposed shape.
 LDD 12557 This is the screen location of the second character in the c-list.
 STD PSCR Store as proposed screen location. Chose second character's loc only because holemaker must start somewhere
 LDD 12560 Get second character's real memory location.
 STD RLOC Store as real location.
 LDA 12562 Get second character's bit set.
 STA RBIT Store as bit set.
 JSR RADDVC Get a random vector.
 JSR ADDCHQ Add the holemaker to the character c-list.
 INC NLMM Increment the holemaker count for this round. Done adding holemaker.

* In this section a "random" number is generated and checked
* against a parameter (MINCH) to see if a mine can be added
* to the c-list. No more than the maximum number allowed can
* be added.

TIP
LDA RAD1 Adding mines to the c-list. Get random number.
 ORA RK4 Or with a second random number.
 CMPA MINCH Compare with the chance of getting a mine this round.
 BMI LYM If the number is higher than MINCH, don't add a mine. Branch.
 LDA NLMM Get number of mines presently alive on the screen.
 CMPA MAXM Compare with the maximum allowed this round.
 BGT CHAR If we have reached the maximum number of mines allowed, don't add a mine. Branch.
 LDA #106 Character number for a mine.
 STA PSHP Store as proposed shape.
 LDD MSCR Get starting screen location for mines this round.
 STD PSCR Store as proposed screen location.
 STD MINT Get real video ram location for mine calculated when setting up this round.
 STD RLOC Store as real location.
 LDA #10B Get bit set for mine calculated when setting up this round.
 STA RBIT Store as bit set.
 LCA #2T Bubble byte. Mine's control whether the mine is actively chasing the player or bouncing (see mine section.
JSR ADDCHC Add the mine to the character c-list.
IND NUMH Increment the number of mines counter.
LDA #FF Going to make a noise here to tell of the birth of a mine.
TIE DECA Noise is a saw tooth wave of 253 decreasing periods.
BEQ CHARS If A is 0, done with noise, branch.
TFR A,B noise.
TIE STB $FF20 Store noise to A/D converter.
TIE CECB noise.
TIE BRE TIE More noise.
TIE BRA TIE More noise.

* Values are tested here to see if the computer guns will fire.

LV3 CMPA GCM Here we see if guns fire. If mine was born, this was skipped. Compare contents of A with gun chance.
BHI CHARS If A was higher, no guns are fired, branch.
LDD G1S Get gun #1's screen location for this round.
STD PSCR Store as screen location.
LDD G1R Get gun #1's video ram loc as calculated when setting up this round.
STD RLOC Store as real loc.
LDA G1B Get gun #1's bit set as calculated when setting up this round.
STA RBIT Store as bit set.
LDD G1V Get gun #1's vector for this round.
STD VCUT Store as vector.
LDA G1L Get length of gun #1's shot for this round.
STA TPP1 Store as wobble byte. For guns, wobble byte is decremented to zero and then gun’s shot deleted from c-list.
LDA #96 Shape number for computer gun shot.
STA PSHP Store as proposed shape.
JSR ADDCHQ Add shot to the character c-list.
LDA #136 Shape number for the gun itself. Going to write it on the screen whenever gun fires.
JSR SMPADR Look up the shapes address.
JSR WRITSH write the shape.
LDD G2S Get gun #2's screen location.
STD PSCR Store as screen location.
LDD G2R Get gun #2's video ram loc as calculated when set up of round.
STD RLOC Store as real location.
LDA G2B Get gun #2's bit set as calculated in set up of round.
STA RBIT Store as bit set.
LDD G2V Get gun #2's vector.
STD VCUT Store as vector.
LDA G2L Get gun #2's shot length.
STA TPP1 Store as wobble byte.
LDA #96 Computer shot shape number.
STA PSHP Store as proposed shape.
JSR ADDCHQ Add computer shot to the c-list.
LDA #138 Shape number for gun #2.
JSR SMPADR Look up shape address.
JSR WRITSH write the gun.

* The loop below is entered once for every location in the c-list.
* Way up above the X register was set to 12547, the player's location in the c-list. The rest of the characters are now checked.
* By looping through nine bytes at a time each c-list entry is sampled. If the entry (shape number) is zero the slot is considered empty. Other numbers correspond to other characters:

ENTRY = CURC ;Process current atom
ENTRY = #106 ;Process mine
ENTRY = #102
ENTRY > #42
ALL ELSE

Each discriminating step occurs in the order given.

LEAX 9,X
Add nine to X so we will be addressing the next character.

CMPX #13312
Check to see if the end of the c-list has been reached.

LBGE LOOP
If it hasn't branch way back up to LOOP.

LCA EAH
Get the end of round wait counter.

BEQ N22 If it equals C (which it does till the last atom is up) branch.

N22 DEC
Decrement wait value.

BNE N22 If not done waiting branch back up.

N22 LDA RAD4
Gonna make the sound of the explosion. If there has been one recently. RAD4 is set to SFF decrmted for sound.

BEQ GC8 If RAD4 = 0 don't make any noise branch.

CUDA #2
Going to make sound for two cycles every second cycle. Check to see if its time.

BEQ QDX No, not this cycle but branch to decrement counter anyway.

LDA RAD1
Get that other random number.

EORA 2X
On an XOR with 2nd byte of character's screen loc. Trying to make a pseudo random noise.

ANDA RAD4
Add more confusion to the byte.

CCMA Add more confusion to the byte.

QDX DEC RAD4
Decrement RAD4 so that the sound eventually dies out.

QPS STA $FF20 Store to the A/D converter. The zeros every two rounds are stored here too so the sound is dynamic.

IN X
Don't with the noise. Get the character number from the c-list.

BEQ CHARB A character number of zero means no character, branch up.

CMPO CURC Compare the character number with the current atom.

BEQ CHASED If the character is CURC, branch to process CURC.

CMPO #106 Is the character a mine?

LBQG MINE If it is branch to process a mine.

CMPO #102 Is the character a holemaker?

LBQG MCLE If it is branch to process holemaker.

CMPO #42 All characters greater than or equal to 42 are shots of some sort at this point. Branch to do neutrons.

LBQG NEUT Branch if neutrons.

BRA C30 The only characters not selected yet are blinking, bothersome atoms, branch to process them.

(* Next section for current atom. Depending on random samples
* current atom (CURC) may run away from player's character
* (if shots have been fired) or head for location between
* computer guns (AIM) or possibly just continue with present
* vector.

CHASED TST RAWY If get here, are processing current atom. Check run-away byte. (Incremented below by neutrons.)

BLE VC1 If not greater than zero, branch

DEC RAWY Decrement run away byte so it'll eventually get to zero and CURC can stop running.

EEM RAWY Increment again.

LDA A0 Get the random number.

ANDA #07 Skip making CURC run away if not = C.

JSR KEWVEC Generate vector towards player.

LDS #0 Clear out D register. Going to make a vector away from player by subtracting vector from O.

SUBD VCUW Subtract generated vector.

STO 7X Store as this CURC's vector.

BRA C30 Branch to process movement.

VC1 INC A0 In the section it is determined if CURC character we are processing will head for AIM.

LDA A0 Number A0 has been modified by line above. Get random number.

ANDA ATRC	Round number by the attraction parameter.

BNE C30 If the result was not zero, branch to process character's movement.
JSR 0NVEC Generate vector towards AIM location.
LDD VCUT Get vector generated.
STD 7, X Store as CURC character’s vector.

* Both CURC and inactive atoms are processed the same in this
* section. Atom is moved according to vector if possible. Wobbling
* effect is taken care of.

LDA 3, X Move the movement of all atoms is processed. Get wobble byte for character from c-list.
BEQ C31 If wobble byte 0 there is no wobble, don’t bother processing the wobble branch.
LSL A Multiply the wobble by two. This will be clear in a second.

ANDA #C2 Clean out all bits except the second. The result of this is added to the shape # to get # of wobble.

AADA /X The antishape written will correspond to either the wobble or normal depending on which was written last.
JSR SPADR Look up the shape’s address.
JSR AATISH Write the antishape to erase it from the screen.
JSR NEMLOC Generate new screen loc from shapes old loc and vector.
JSR REALCO Translate into real coordinates.
LCA 3, X Get wobble byte again. Going to check whether wobble or normal shape for this char is to be written next.
BEQ C32 If wobble byte = 0 next shape will be normal by default.
DECA Decrement the value for the wobble.
LSLA Multiply by two.
AADA /X #C2 Remove all but second bit.
AADA /X Add result to shape number to get wobble number if bit was set or keep shape number if not.
JSR SPADR Look up the shape’s address.
JSR ORMOV Check to see if the shape can be written at the location pointed to by RLOC and RBIT.
BEQ C33 If it can, branch.
JSR RNDVEC If gets here, shape wouldn’t fit at new location. Generate random vector.
LCA VCUT Get vector generated.
STD 7, X Store in c-list as character’s vector.
JSR NEMLOC Generate a new location from new vector and character’s old position.
JSR REALCO Translate into real coordinates.
JSR ORMOV Check to see if new location is clear for character to move into.
BEQ C35 If it is branch.
LDD 4, X It wasn’t better to give up this cycle. Get shape’s old real location.
STD RLOC Store as real loc.
LDA 6, X Get shape’s old bit set.
STA RBIT Store as bit set.
LDA 3, X Get shape’s wobble byte (note it was never modified).
BEQ C34 If wobble byte = 0, then without a doubt character was in it’s normal state before.
LSLA Wobble byte not = 0. Multiply by two.
ANDA #C2 Clear all but second bit.

ANDA /X Add character’s normal state shape number.
JSR SPADR Look up resulting shape’s address.
JSR WRTSPW Write the old shape of the character back where we found it.
LDD /X We weren’t able to move so apparently shape hit something. Reset wobble to shape wobbles for a while.
CRA 3, X Result of this op is assigned to the char’s wobble byte. Note bit 1 is preserved, thus is old wobble state.
STA 3, X Store in c-list as character’s wobble byte.

LRA CHARA All done with inoble atom, branch to process more characters.
ENE C35A If it was already wobbling, skip next instruction else must set wobbling w/normal state being first.
INC 3, X How the char’s wobble byte = 1. Has to be odd so that after decrementing below, correct antishape used next cyc
LDA /X Get shape number. This’ll be or’d with wobble byte and stored as wobble byte.
ORA 3, X OR with wobble byte, note: can always be sure shape number is even.
STA 3, X Store result as wobble byte.

TST 3, X Get here, handle movement of all atoms whether they bounced or not. Check wobble byte.
BEQ C36 If wobble = 0 branch.
C36  CE  3,X  have done everything with a decremented value of wobble byte. Time to decrement the byte itself.
JSR  WRTHSP  Finally! Write the shape (normal or wobble) determined for the atom.
LDD  PSCHR  Get the screen loc used.
STD  1,X  Store in c-list as character's screen location.
LDD  RLOC  Get real video ram location used.
STD  4,X  Store in c-list as character's video ram location.
LDA  RBIT  Get bit set used to write the shape.
STA  6,X  Store in c-list as character's bit set.
LBSA  CHARS  All done with atom, branch to get another character to work on.

* Neutrons and shots of all types are processed in the next section.
* If a shot hits something control is transferred to the BOMB section.

NEUT  STA  5820  If get here character is either computer shot, player shot or chain reaction shot. Make a noise.
STA  TIPS  Store the character (shape) number here. Will need it later as X register may be modified.
CMPA  #66  Check to see if it's a computer shot.
BEQ  NV1  If it is, branch and stuff below for one, don't want computer shots causing atoms to blow away.
CMPA  #112  This is the number of chat screen released when a mine explodes. It is temporarily inert.
BEQ  CCM1  If shot=112 branch below. It is explained there.
INC  RAMY  Player's shots and chain reactions increment the RUN AWAY parameter.
JSR  SNMPOR  For player's shots and chain reactions look up the old shape's address.
DEC  3,X  Dec wobble byte. If result=0, character is deleted.
BNE  701  Not = 0, branch.
CLR  3,X  Clear character from c-list.
LBSA  CHARS  Character deleted, go do another.
C701  LDA  3,X  The player's shot change every 7 cycles. Will see if dealing w a player's shot if so--is it time to change?
CMPA  #48  Check against first shape # of player's shot.
BLT  C70X  Less than, it must be a player reaction neutron, shape = 42 branch.
CMPA  #06  Check against computer's shot #.
BEQ  C70X  Shot is computer's shot, branch.
LDA  #3  Have isolated shot to be one of the eight forms of the player's shot. 03 is a mask.
BNE  C70X  Result not = 0, leave player's shape # the same this cycle.
LDA  3,X  Result=0, time to change player's shot's shape. Get present shape #.
ACDA  #06  Add 6 to get next shape #. Note: antibomb is at X+2 and bomb is at X+4.
STA  3,X  Store as player's shot new shape #.
JSR  SNMPOR  Look up address of new shape.
C70X  JSR  NEWLOC  Character not deleted. Calculate new screen location.
JSR  REALCO  Translate into real coordinates.
JSR  OKMOV  Check to see if it is clear to move into the new location.
BNE  BCBM  If it isn't, blow up! Branch.
JSR  WRTHSP  Write the shape at the new location.
LDD  PSCHR  Get screen loc used.
STD  1,X  Store in c-list as character's screen loc.
LDD  RLOC  Get real loc used.
STD  4,X  Store in c-list as character's real loc.
LDA  RBIT  Get bit set used.
STA  6,X  Store in c-list as character's bit set.
LBSA  CHARS  Branch to process new character.

* Exploding mines splinter into computer shots, however the shots are inert for a short time so that they may disperse from the point of the explosion. If this were not so most of the shots would bump into one another and explode right there. The following section decides if it is time for the
* shots to become active.

* COMM

DEC 3X

BEQ C2Q

JSR SMPADR

JSR AKTISH

JSR NEWLCC

JSR REALCO

JSR CRMGV

LBBNE CHARS

LDA C70X

LDA #96

STA SX

LDA #50

LDA SX

STA 3X

LIBRA CHARS

* The next section takes care of the explosion which occurs when
* some kind of shot hits an object on the screen. The shape number
* for the shot is used to determine what kind of explosion
* to use. Say the shot or neutron's shape number is Q. The overlay
* shape number is then Q+2 and the explosion shape number is
* Q+3. The overlay shape is read from the screen, the explosion
* written and the overlay restored.
* This section also contains the code which decides if an atom
* has been fissioned or a mine destroyed.

* BOMB

CLR SX

STX TEMX

BRA DCAR

LDA TPP3

ADD A4

JSR SMPADR

JSR WRTSHP

LDA TPP3

ADD A2

JSR SMPADR

LDA TPP3

JSR WRTSHP

LDA #255

STA $FF20

STA RN0

LDA #0

STA $FF20

LDA $128

STA $FF20

LDA #255

STA $FF20

LDA TLOC

LDB TEMX

LDD TPP3

STA S1J

STA S1K

STA S1L

STA S1M

DCAR LCD TLOC

SUSD A13312

LSLA

LSLB

INCA

FF1

FF1 LSLA

Unatron listing page: 14
LSB  FF2
   Multiply LSB.

INC   T0IT
   Propagate carry into MSB.

ADD   T0IT
   Add bit set from point of collision into LSB. Now D contains the screen loc for where the shot struck.

LDA  #12567
   Load X with start of character c-list.

LDA  CURC
   Load A with CURC’s number for comparison below.

LDA  #106
   Load B with the character number for a mine.

LEAX  9,x
   Make X point to the next character in the c-list.

CMPX  #3132
   Are we at the end of the c-list?

CMPI   C52
   If so, neither CURC nor MINE was involved in the explosion. Branch to finish up this section.

BEQ   FF5
   If it is, branch to see if it was close enough to the point of impact to be blown up.

CMPA  x
   Check to see if character in c-list is a mine.

BEQ  FF5
   If it is, branch to see if it was close enough to the point of impact to be blown up. Branch.

BRK  FF4
   It is neither. A and B still hold values for CURC and MINE, branch part way up.

LDA  1,x
   Are we looking at either MINE or CURC in c-list? Get char’s screen loc from c-list.

SUBA  PSCR
   Subtract screen loc calculated above. Am going to do a crude distance calculation.

STA  RND4
   Gonna get all bits $FFED and higher. Effectively the Y coordinate of the subtraction. Store LSB for later.

LSL B  Shift left. Want to get Y axis difference into one byte.
   Shift the $80 bit off the left end.

INC   T0IT
   Propagate carry in LSB. Now have Y axis difference in the A register.

BGT  FF7
   Checking to see if difference is negative.

NEG   A
   If not, branch around next instruction.

CMPA  #04
   Check to see if Y displacement is within 4 pixels. Note: this is a crude rectangular distance func.

BGT  FF3
   If Y displacement > 4 pixels, branch back up and continue checking the character c-list.

LEAX  RND4
   Get here, Y within range. Gonna check X coordinate displacement.

TPR  A
   Make a working copy of X displacement.

ANDA  #4F
   Zero out high bit.

ANDB  #40
   Have X bit signed X displacement. Maximum distance=64 pixels. Check 7th bit to see if negative.

BEQ  FF8
   If not, skip next instructions.

CRA  #80
   Propagate negative into 8th bit so byte is a proper negative.

NEG A
   Take two’s complement to get X-axis difference.

CMPA  #03
   Check to see if X displacement is within 3 pixels of shot’s point of impact.

BGT  FF3
   If not, branch up to check more characters in the c-list.

LCA  #106
   The character was hit! Get character number from the c-list.

BNE  T29
   Was it a mine?

JMP  ShPA
   Get mine’s shape address.

JMP  ANTISH
   Erase mine from the screen.

CLR x
   Delete the mine from the c-list.

DEC  NUMM
   Decrement the number of active mines counter.

LCA  NUMN
   Get the number of computer neutrons to be released from the explosion of the mine.

STA  RND4
   Store away. Will use as a loop counter.

LDA  #112
   Get inert computer shot #. Must make inert so that shots disperse without blowing up on each oth.

STA  PSHP
   Store as proposed shape.

LDD  1,x
   Get deleted mine’s screen location.

STO  PSCR
   Store as screen loc. Will make inert shots originate from where mine used to be.

LDD  4,x
   Get deleted mine’s real location.

STO  RLOC
   Store as real loc.

LDA  6,x
   Get deleted mine’s bit set.

STA  RBIT
   Store as bit set.

LCA  #5
   This will be stored as inert shot’s wobble. In five cycles shot will become active computer shot.

Unatron listing page: 15
T2T DEC  RAND4 Decrement loop counter.
LDRQ  C52  If equal to zero, we are done here. Branch below.
LDSR  RANDVE Set a vector for the insert shot.
BRA  T2T Branch up to see if more are to be added.
T29 LCA  3X Get here. CURC is in explosion. Get char's wobble byte so can write correct antishape.
BEQ  C702 If wobble byte = 0 then char is in its normal state. (as opposed to wobble) Branch.
LSLA  X Shift wobble byte left to multiply by two.
ANDA  #C2 Clear out all but second bit.
ACOA  X Add to character's shape # to get correct state.
JSR  SPADP Look up shape in table.
JSR  ANISH Erase the shape from the screen.
CLR  X Erase the character from the c-list.
LDD  1X Get deleted char's screen loc. Going to make fission occur where character was.
STD  PSR Store as screen loc.
LDD  4X Get character's real loc from c-list.
STD  RLX Store as real loc.
LDA  6X Get character's bit set from c-list.
STA  RBIT Store as bit set.
LCA  #42 First will add chain reaction neutrons to c-list if needed.
STI  PSHP Store character # for chain reaction neutrons to the c-list.
STA  TMAP Store as wobble byte too. This is decremented each cycle, so the neutron's max life is 42 cycles.
LDA  CLRC Gonna check how many if any neutrons to add based on CURC's number. Lower nos. release none.
CMPA  #22 Check to see if CURC is less than or equal to 22 (second atom).
BLE  C46X If it is, no neutrons added. Branch.
JSR  RANDVE At least one to be added. Generate a vector.
JSR  ADDCMG Add the chain reaction neutron to the c-list.
LDA  CLRC Gonna check CURC again.
CMPA  #26 If CURC <= 26 (fourth atom) will not add any more neutrons.
BLE  C46X Less than or equal17 Branch.
JSR  RANDVE Generate vector for second neutron to be added.
JSR  ADDCMG Add character to c-list.
LDA  CLRC Get CURC. Gonna see if we reached the last atom for this round yet.
ADDA  #04 Four + CURC gives number of next atom (if there is one).
CMPA  #42 If the result is 42, we are on the last atom and no more are to be added this round.
BNE  C46 If not the last, branch.
LDA  EWAI Just deleted an occurrence of atom #6. There are no atoms added. Must make busy wait to slow things down.
CMPA  #27 Just got end of round wait counter. If equals 27, i.e., 5 atoms left, will change attraction parameter.
BNE  PIP Not equal 25, branch around next instructions.
LDA  AC2 New value of ATTRACT. How last five atoms will jump around less.
STA  ATRCT Store as attraction parameter.
FUP INC  EWAI Increment the end of round wait counter. Program will count to EWAI every loop to slow game as screen clear.
CLRA  PSHP Rejoining instructions above. Store A as proposed shape.
C46 STA  RANDVE Get a vector for the character.
CLR  TMAP Clear the wobble byte to be added to the c-list with the character.
JSR  ADDCMG Add new atom to c-list.
JSR  RANDVE Must add one more. Get another vector.
JSR  ADDCMG Add another atom to the c-list.
LDB  TMAP Get shape number of shot that caused all this. Must give owner of the shot credit for the hit.
CMPB  #42 was it a chain reaction neutron?
BNE  TA1 If not branch.
LCA  CREAC Get chain reactions this round counter.
ACOA  #C1 Add one to it for the hit.
C4A STA  CREAC Put in proper 3C0 form.
STA  CREAC Return it where it came from.
TA1
BRA TA3  Branch below.
CMP2 #S6  Did a computer hit cause the explosion?
BNE TA2  If not, branch below. It would have to have been player's shot.
LDA CCMH  If it was computer shot, get computer hits this round counter.
ADDA #C1  Add one to give credit for the hit.
CPLA CCMH  Put in proper BCD form (it is easier to print later).
STA CCMH  Store as computer hit counter.
BRA TA3  Branch below.

TA2
LDA YLHR  Get your hits counter.
ADDA #C1  Give yourself credit for the hit.
CPLA  Put in proper BCD format.
STA YLHR  Put it back.

TA3
LDY #T2547  Must check to see if there is a new CURC after the fission.
LDA CRUC  Get value of current atom.
LEAY 9A  Step through character c-list.
CMPY #13312  Reach the end of the c-list.
BGE C250  If so, it is time to change CURC. Branch.
CMPA Y  Check CURC against character in c-list.
BEQ C52  If equal, there is at least one occurrence of CURC left. Branch out of here, leave CURC alone.
BRA C51  Keep checking. Branch up to look at next character in the c-list.
C250 ADDA #G4  If get here, it is time to change value of CURC. Add 4 to get number of next atom.
BRA C51  Check again. Branch up to look at next character in the c-list.
CMFA #E2  If = 42 which is > 38 = atom #6, then this round is over.
LBEQ ZSTART  Branch to start a new round.
STA CRUC  Store new value of CURC.

C52
LDA TPP3  Rendezvous in DCAR. Get # of shot that caused the explosion.
ADDFA #C2  Add two to get number of overlay for the explosion.
JSR SHPAID  Look up the overlay shape's address.
JSR BEVYSH  Build the overlay at location of explosion.
JBR DRET  Branch way up and finish this explosion mess up.

* Control is transferred to the following section whenever the player
* is struck by a mine.

DEAD
DEC MENL  Player's character hit by mine, decrement number of player's "men" left.
LDA #46  Gonna draw explosion shape on the screen.
JSR SHPAID  Look up shape's address in table.
JSR WRTSH  Write the shape at the player's location.
CLRA  Gonna make a whining noise.
LVL DECAC  A triangle wave of increasing amplitude and decreasing frequency for 255 cycles.
BEQ BR9  When A=0, sound is finished.
TFR A B  Move A to B where it'll be incremented and sent to the D/A converter.
INCB  Increment like I said.
BNE LV2  Rot = 0, still building this cycle of the wave.
BRA LV1  Get to here, this triangle built. Branch up.
BR9
LDA #112  Gone with noise. Gonna make pieces fly off the player's dead character.
STA TPP3  Go store the shape number of the first piece here temporarily.
BR1
LDA TPP3  Store the 112 here. Only reason is so can be sure RND4 does not equal 0. Explained below.
ACDDA #C4  Add 4 to get the number of the next explosion piece to be added.
CMFA #136  If the piece to be added is 136, five pieces have already been added.
BEQ BR2  Exit this section if all five pieces have been added.
STA TPP3  Store the new shape number temporarily.
STA PSHP  Store it as proposed shape.
TFR A B  Make a working copy. Are composing wobble byte- the byte says how many cycles the piece will live.
OAR #200  Make it live fairly long.
STB TPP1 Store as proposed wobble byte.
JSR RNDVEC Generate a random vector.
JSR ADDCHG Add the character to the c-list.
BRA BR1 Branch up to see if more pieces are to be added.

BR2 LDX #12547 Gonna loop through the c-list and process only the pieces. Everything else in the c-list remains intact.

BR3 LEAX 9,X Make X point to the next character in the c-list.
CLRB Need a zero.
STB 3FF20 Store it as sound.
CMPX #13312 Cone looping through c-list?
BNE BR4 If not branch and loop some more.
TST RAD4 Serving as a flag to keep going when all pieces have been deleted. If = 0, all been deleted.
BEQ BR5 All deleted, branch below.
CLR RAD4 Reset flag for all deleted. Will be set true if a piece is found in the c-list.
BRA BR2 Branch to start loop over.

BR4 LDA >X Get the character X pointed to by X.
BEQ BR3 If zero, keep looping. No character there.
CMPA #116 Compare character number with the lowest of the pieces.
BLO BR3 If it’s less, it’s not one of the pieces. Leave it alone and branch up.
ACDA #22 It is one of the pieces. Add two to get the number of it’s overlay shape.
JSR SMPADR Find the shape’s address.
LDA 4,X Get the piece’s real loc.
STD RLOC Store as real loc.
LDA 6,X Get the piece’s bit set.
STA REIT Store at bit set.
JSR WRTSHP Write the overlay shape. (It was composed last cycle.)
DEC 2X Decrement the piece’s wobble byte.
BNE BR6 If not = 0, shape gets to live at least one more cycle. Branch around next instructions.
CLR >X Delete the piece from the c-list.
BRA BR3 Branch to loop.

BR6 INC RND4 Set the flag that says there are pieces still left in the c-list.
LDA #15 Gonna make a noise.
ANDA 3,X Value sent to A/D converter is piece’s wobble byte’s low 4 bits.
STA 3FF20 Store to A/D converter.
BNE BR3 If result from last op = 0, will move the piece. Else leave alone. (this way pieces move slowly). Branch.
JSR NEWLOC Gonna move piece. Find new location.
JSR REALCO Translate to real coordinates.
JSR ECYYP Build an overlay of piece at new coordinates. note: shape’s address found above.
LDA >X Get # of the piece. Have been working with the overlay till now.
JSR SMPADR Find the shape’s address.
JSR WRTSHP Write the shape on the screen.
LDA RLOC Get the generated real address.
STD 4,X Store in c-list as shape’s real address.
LDA RBIT Get the bit set used.
STA 6,X Store in the c-list as the shape’s bit set.
LDD PSCR Get the screen loc generated.
STD 1,X Store in the c-list as the shape’s screen address.
LDA 3,X Gonna make some noise. Get character’s bit set.
LDB 2,X Get 2nd byte of character’s screen loc.
MUL Scramble the two. (We have plenty of time at this point.)
STA 3FF20 Store the MSB to the A/D converter.
BFA BR3 Branch up to loop more.

BR5 TST MENL Cone with the pieces. See how many “men” the player has left.
BNE SSS If there is at least one, the game isn’t over yet. Branch below.
JSR TALLY Player is done for. Put up the score.

SSS JSR CCISP Whether it’s the end or not, put up the number of scores.
JSR KBDWAI Wait for the player to read it and push the fire button.
LBRA  RESTART Branch way way up.

* Following section processes a holemaker.
* If the holemaker hits something on the screen & jagged black
  hole is written where the collision occurred.

HOLE
JSR  $4A
JSR  ANTIH
JSR  NWLOC
JSR  REALLC
JSR  CKMDV
BEQ  LLA
JSR  RNDVEC
LDD  VCUT
STD  7X
LDA  #104
JSR  WRT
LDD  LOC
STD  RLOC
LDA  TBIT
STA  RBIT
JSR  WRT
LBRA  CRAS
LLA
JSR  WRT
LDD  RLOC
STD  4X
LDD  PSCR
STD  1X
LDA  RBIT
STA  6X
LBRA  CRAS

* Section for processing mines.
* Mines continually plot a course for the player's character
  until they bump into something other than the player.
  The mines will then wander aimlessly for a number
  of cycles specified in the wobble byte ($3/4X). The number
  of cycles spent wandering is calculated between two parameters
  specified in the overlaid data.

MINE
JSR  $4A
JSR  ANTIH
TST  3X
BEQ  T23
JSR  NWVEC
LDD  VCUT
STD  7X
JSR  NWLOC
JSR  REALLC
JSR  CKMDV
BEQ  T22
LDD  TLOC
CMPD  12551
BNE  T74
CMPA  12553

Unetron listing page: 19
BAE T24 Not the same didn’t hit the player dead on, branch.
JSR WRTSHP Hit the player! Write the mine on the screen.
JSR DEAD Player is dead! Go make sure he/she knows it.
ECRA RAD1 In the process of making up a new wobble byte for the mine.
AAPA MINSPH Limit the number of cycles the mine will wander aimlessly by MINSPH.
CR0 MINPH And make sure it wanders at least MINSPH cycles.
STA 3,X Store in the c-list as the mine’s new wobble byte.
JSR RADVEC Generate a random vector for the mine.
LDD VGUT Get the vector.
STD 7,X Store in c-list as the mine’s new vector.
DEC 3,X Get here, processing all wandering mines.
JSR NEWLOC Generate a new location from mine’s vector and old location.
JSR REALCO Translate into a real location.
JSR QKMOV Check to see if new move is ok.
BEQ T22 If it is, branch.
JSR RADVEC No, try again. Generate another random vector.
LDD VOUT Get generated vector.
STD 7,X Store in the c-list as the mine’s vector.
LDD 4,X Get the mines real loc from the c-list. Gonna write the mine back where it was.
STD RLOC Store as real loc.
LDA 6,X Store the mine’s bit set.
STA RBIT Store as bit set.
JSR WRTSHP Write the mine back where it was found in the first place.
LBRA CHARS Branch to process more characters.

T22 JSR WRTSHP New move was ok, write the mine at the new location.
LDD RLOC Get the real loc generated.
STD 4,X Store in the c-list as the mine’s real loc.
LDD PSCR Get the screen loc generated.
STD 1,X Store in the c-list as the mine’s screen loc.
LDA RBIT Get the bit set.
STA 6,X Store in the c-list as the mine’s bit set.
LBRA CHARS Branch to process more characters.
SETUP $24  Direct page register loaded w/$24 in main program.
CRG  $2FA2  Link by hand.
PSC2  ECU  $2447  LSB of screen location.
PSR2  ECU  $2446  Screen location.
RB1T  EQU  $2443  Bit set.
RLOCC  ECU  $2449  Real video ram location.
STSH  ECU  $243E  Start of shape construction instructions.
TLOC  EQU  $2441  Working temporary for real vidram location.
VOUT  EQU  $243D  Second byte of vector.
VC2C  EQU  $243C  Vector.
PSHP  EQU  $243B  Shape number.
TMP1  ECU  $2432  Working storage.
CURC  EQU  $243A  Current fissionable character.
RAWY  EQU  $2439  Run away parameter.
TEM1  EQU  $2433  Working storage.
TVEC  EQU  $2437  Working vector temporary storage.
RND1  EQU  $242F  Working temporary.
STB0  EQU  $242D  Start of screen layout border.
RND2  EQU  $242E  Working temporary.
RND3  EQU  $242C  Working temporary.
RND4  EQU  $242C  Working temporary.
RND5  EQU  $2428  Working temporary.

* ADDCHQ - Subroutine traverses character c-list looking for
  * the first unfilled spot. When found, the following assignments
  * are made:    PSHP -> /x    Character shape number
  *              PSCH -> 1/y    Screen location
  *              LCD1 -> /z    Blobble byte
  *              RLOCC -> 4/y    Real vidram location
  *              RB1T -> 6/y    Bit set
  *              VOLT -> 7/y    Vector
  * If there is no space in the c-list the character is not added
  *
ADDCHQ LDY #12358  Start of character c-list = 9.
C90 LEAY 9,y  Add nine to point to next character in c-list.
CMPY #13312  At the end of the c-list?
BGE C91 Yes, no character added, branch to exit.
BNE C90 Occupied, branch up to loop more.

C90 LCA PSHP Get proposed shape number.
STA /y Store in c-list as new shape.
LDC PSCH Get proposed screen location.
STD 1/y Store in c-list as new character's.
LDA TMP1 Get blobble byte.
STA 3/y Store in c-list as character's.
LDC RLOCC Get real screen location.
STD 4/y Store in the c-list as the character's.
LCA RB1T Get bit set.
STA 6/y Store in the c-list as the character's.
LDA VOUT Get generated vector.
STD 7/y Store in the c-list as the character's.
C91 RTS Return.

* REALCO - Subroutine takes screen loc from PSCH and translates
* into real coordinates. Vidram location is stored in RLCC.
and bit set is stored in RBIT.

**REALCO**

LEB PSC2 Get 2nd byte of screen location.

ANDB #C3 Clear all bit last 2 bits - this makes up the bit set.

STB RBIT Store as bit set.

LCD PSCR Get screen loc again.

LSRB Divide by two. Gonna get vidram location.

LSRA Divide by two. Gonna get vidram location.

BCC CC1 If no carry generated, skip next instruction.

ADDB #180 Propagate the carry into the low byte.

CC1

LSRB Divide by two again.

LSRA Divide by two again.

BCC X12 If no carry generated, skip next instruction.

ADDB #180 Propagate carry into the low byte.

X12

ADDO #13312 how have real loc assuming vidram is on page 0. Add 13312 to get page 34.

STB RLOC Store as real loc.

RTS Return.

**OKMOV** Subroutine traces out shape onto the screen to see

if the space is unoccupied. No pixels are written, only checked

for coincidence. Input is present position of cursor indicated

by RLOC and RBIT. STSH is accessed to get address of start of

shape writing instructions. When and if it is determined that

one of the pixels to be written would coincide with something

on the screen already the condition code register is cleared

and the subroutine returns. If there is no coincidence the

zero flag is set before return.

OKMOV

LEB RLOC Checking to see if spot on screen is clear for a shape to move into. Get video ram location.

STD TLOC Store in working temp.

LSRA RBIT Get bit set.

STA TBIT Store in working temp.

LCD STSH Load index register with first instruction for drawing the shape.

EC2

LCA #CC Get shape construction instruction.

BGE CC3 If it is less than zero branch around next instruction. An instruction less than zero means end of shape inst.

LCA #504 Get here it is ok to move shape into RLOC with bit set RBIT. Gonna set zero flag in CC to say so.

TFR A/CC Set zero flag.

RTS Return.

CC2

ANDA #140 Check instruction byte to see if pixel being written.

BEQ CC4 Result = C, no pixel being written. Branch to bottom.

LDA #1C0 Gonna check pixel at TLOC with bit set TBIT. If there is already a lit pixel there - not OKMOV. $CO is mask.

LDB TBIT Get bit set. Will shift mask right TBIT times.

BEQ CC5 If TBIT = 0, no shifting to be done, branch.

CC3

LSRA Shift mask right.

LSRA Shift mask right.

EC3C Increment shift counter.

CC5

ANDA #1C0 And the mask with the vidram location.

BEQ CC4 If = C branch to move cursor and continue checking if the move is ok.

CLRA If the pixel wasn't blank then the move is not ok. Gonna set CC to say so.

TFR A/CC Zero flag in cond code is not true - move was not ok.

RTS Return. Note: TLOC anc TBIT contain location of coincidence. This is used for the origin of explosions.

CC4

JSR AXTSET Move 'cursor'.

BRA CC2 Branch up to check next instruction byte.

**NXTSET** - Moves cursor pointed to by TLOC anc TBIT, temporaries
* for RLOC and RBIT. The cursor bits of the scope instruction
* currently pointed to by the Y register are consulted to find
* where next to move the cursor.

**NXTSET**

`LEA #320` Subroutine for moving cursor. (see text for exact details.) Get bit 3 - if set cursor moves left.

`ANDA #Y` Check bit 3 of instruction byte.

`BEQ CC7` Not set, branch below.

`ECX TBIT` Move cursor left, decrement bit set.

`BGE C10` If bit set greater than or = C, then TLOC is still correct. If not, (=1) then must correct TBIT and TLOC.

`LCA #C3` If result above was -1 then new bit set = 3 and TLOC = TLOC + 1.

`STA TBIT` Store as new temporary bit set.

`LDE TLOC` Get vidram loc.

`SUBC #C1` Subtract 1 to move one byte left.

`STD TLOC` Put it back.

`BRA C10` Move left, assuming now that no need to check for moving to the right. Branch to check vertical movement.

**CC7**

`LCA #31C` Eit 4 tells if cursor moves right. $10 is a mask.

`ANDA #Y` Mask instruction byte.

`BEQ C10` If result=0, don't move cursor to the right. Branch to check if cursor is to be moved up or down.

`INC TBIT` Cursor moves right. Increment bit set.

`LCA #C4` Gonna compare with 4 (<@>0 and vidram loc+1).

`ANDA TBIT` Equal to four?

`BEQ C10` If not, branch to check vertical movement.

`CLR TBIT` Get here, set bit set=0.

`LDX TLOC` Gonna increment vidram loc.

`ADDC #C1` Increment.

`STD TLOC` Put it back.

**C10**

`LDA #308` Mask to check for upward movement of cursor.

`ANDA #Y` Mask instruction byte.

`BEQ C10` If result=0, branch to check for downward movement.

`LDE TLOC` Get vidram loc.

`SUBC #32` Move upward one line (32x=128).

`CMPD #13312` Is new loc below start of vidram?

`BGE C14` If not, skip next instruction.

`ADDC #3072` Cause wrap-around.

`BRA C14` Branch to bottom.

**C11**

`LCA #304` Mask for downward movement.

`ANDA #Y` Mask out instruction byte.

`BEQ CC9` If result=0, no downward movement, skip below.

`LDE TLOC` Get vidram loc.

`ADDC #32` Move down one line.

`CMPD #16383` Did that move it off the end?

`BLE C14` If not, branch around next instruction.

`SUBC #3072` Cause wrap-around.

**C14**

`STD TLOC` Store vidram loc.

**C09**

`LEAY 1/Y` Make Y point to next instruction byte.

`RTS` Return

* NEWLOC - Extracts character's vector (7 X) and screen loc (7 X)
* and adds them together. Result is new screen loc stored in PSCR.

**NEWLOC**

`LDE 1/X` Set character's screen loc.

`ADDC 7/X` Add character's vector.

`BGE C15` If still > 0 (still on the screen maybe) branch.

`ADDC #12268` Below zero not defined. Add 12268 (# of pixels on the screen) to cause wrap around.

`BRA C16` Skip next instructions.

**C15**

`CMPD #12287` Off the end of the screen?
BLE C16 If not, branch to exit.
SUBD #12288 Cause wrap-around.
C16 STD PSCR Store new screen loc.
RTS Exit.

* ANTISH - Subroutine takes character's real vidram loc and bit
* set directly from the c-list and uses them as the starting
* cursor. The shape instructions pointed to by STSH are used
* to write the shape in black thereby erasing it from the screen
* completely.

ANTISH LCD 4yx Get character's vidram loc.
STD TLOC Store in working temporary.
LDA 3yx Get character's bit set.
STA TBIT Store in working temporary.
LDY STSH Get the shape's address, put in index register.
C17 LDA Y Get instruction byte.
BGE C18 If not < 0, not end of shape. Return.
RTS Byte < 0, end of shape. Return.
C18 ANDA #80 Check to see if pixel written.
BEQ C19 If not, branch below.
LDA #30 Yes, $30 is a pixel mask.
LCE TBIT Get the bit set.
BEQ C21 If = 0, no shifting of mask need be done.
C20 LSRA Shift mask right.
LSRA Shift mask right.
CECB Increment counter.
BNE C20 If not = 0, not done shifting. Branch up.
C21 CCMA Invert the mask.
ANDA [TLOC] Anding causes all bits except masked bits to remain the same.
STA TLOC Store it back into vidram. Pixel is now blacked out.
C19 JSR NITSET Get next cursor location.
BRA C17 Branch up to do more.

* WRTSHP - Subroutine takes RLOC and RBIT as starting cursor.
* Shape whose instructions are pointed to by STSH is written
* on the screen.

WRTSHP LDD RLOC Get vidram loc.
STD TLOC Store in working temporary.
LDA RBIT Get bit set.
STA TBIT Store in working temporary.
LDY STSH Put the shape's address in the index register.
C22 LDA Y Get instruction byte.
BGE C23 If not < 0, not done. Branch around next instruction.
RTS Return.
C23 TPR A,B Make 2 copies of instruction byte.
ANDA #140 Check to see if pixel being written.
BEQ C24 If not, branch to bottom.
ANDB #30 Pixel being written. Isolate pixel in B.
LCA #3 Shift left 3 minus bit set times to get pixel in correct location in byte.
SUBA TBIT Make 3 minus bit set.
STA TVEC Put here for counting.
LCA #3 Pixel mask.
C300 TST TVEC See if need to shift.
BEQ C301 If = 0, no shifting to be done.
<table>
<thead>
<tr>
<th>C302</th>
<th>LSLA</th>
<th>Shift mask left.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>LSLA</td>
<td>Shift mask left.</td>
</tr>
<tr>
<td></td>
<td>LSLB</td>
<td>Shift pixel left.</td>
</tr>
<tr>
<td></td>
<td>LSLB</td>
<td>Shift pixel left.</td>
</tr>
<tr>
<td>DECE</td>
<td>TWECE</td>
<td>Decrement the counter.</td>
</tr>
<tr>
<td>BNE</td>
<td>C302</td>
<td>Not done, branch up to do more.</td>
</tr>
<tr>
<td>C301</td>
<td>CCMA</td>
<td>Cone shifting, invert mask.</td>
</tr>
<tr>
<td>STA</td>
<td>[TLOC]</td>
<td>Put it back.</td>
</tr>
<tr>
<td>CRB</td>
<td>[TLOC]</td>
<td>Add in shape's pixel.</td>
</tr>
<tr>
<td>STB</td>
<td>[TLOC]</td>
<td>Put it back. Pixel is now written!</td>
</tr>
<tr>
<td>C24</td>
<td>JSR</td>
<td>NXTSET Move cursor.</td>
</tr>
<tr>
<td>BRA</td>
<td>C2Z</td>
<td>Branch up to do more.</td>
</tr>
<tr>
<td>VMULT4</td>
<td>LSL</td>
<td>VCU1 For vector multiplication. Not currently used.</td>
</tr>
<tr>
<td></td>
<td>LSL</td>
<td>VCU1 Multiply vector LSB.</td>
</tr>
<tr>
<td></td>
<td>BCC</td>
<td>V11 If carry clear skip next instruction.</td>
</tr>
<tr>
<td>INC</td>
<td>VCU1</td>
<td>Propagate carry into MSB.</td>
</tr>
<tr>
<td>V11</td>
<td>RTS</td>
<td>Return.</td>
</tr>
<tr>
<td>END</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
SHPADR - The shape number is passed in the A register,
the address of the shape is looked up in the shape table
and the result is stored in STSH.

SHPADR TFR A→B Subroutine passed shape # through A→B looks up shape's addr and stores in STSH.
CLRA Clean out high byte.
ADD STBL Add to the address of the shape table to get address of the address of the shape.
TFR D→Y Gonna do a sort of indirect address.
LDD A→Y Get value at the offset in the shape table. This is the shape's address.
STD STSH Store as start of shape.
RTS Return.

OWNVEC - generates a vector towards the AIM location.
- Subroutine swaps player's screen position out of the c-list and
  replaces it with the AIM location and calls NEWVEC.
- NEWVEC thinks it is generating a vector towards the player.
- Upon return from NEWVEC the character's screen is restored
  to the c-list.

OWNVEC LDD 12542 Generate a vector towards AIM. Gonna store player's loc away for a minute.
STD TEMX Store it away. Gonna call a sub that generates a vector towards player with a fake player loc.
LDD AIM Get the point on the screen where CURC is heading.
STD 12548 Store it as player's screen loc for a minute.
JMP NEWVEC Call sub that generates a vector towards the player.
LDD TEMX Vector generated. Gonna put player's screen loc back where it was, no one will ever know we messed with it.
STD 12548 Store in player's screen loc in c-list.
RTS Go home.

* NEWVEC - subroutine generates a vector towards the player's character
* from whatever character the X register is addressing in
* the c-list. Result is stored in VOUT.

NEWVEC
LDD 1/X For generating a vector towards player, get the character's screen loc from the c-list.
ANOB #$80 Clear all bits that describe the X coordinate of the character.
STD TVEC Store that away temporarily.
LDD 12548 How get the player's screen loc from the c-list.
ANOB #$80 Clear all bits that describe the X coordinate.
SUBB TVEC Subtract the player's Y coordinate from the character's.
CMPD #C Are the on the same Y coordinate?
BLE CE0 Same coord or player has higher Y coord branch around next instructions.
LDD #256 Making up vector, give positive displacement of 2 pixels on Y axis.
BRA CB1 Branch to take care of X axis.
BEQ CB1 Branch if same Y coord.
LDD #$FFC0 Player has lower Y coord than character. Vector will have -2 pixel Y axis displacement.
C81 STD VOUT Store the result of the Y coord comparison.
ANDB #$7F Isolate bits that describe X coord.
STB TVEC Store away for a nanosecond.
LDB 12549 Get player's LSB of screen loc.
ANDB #$7F Isolate bits that describe X coord.
SUBB TVEC Subtract this from character's X coord.
BNE C82 If not the same X coord branch.
RTS If the same, all done making up vector, return.

C82 BGT CE3 If player is to the right, branch below.
LDD VCUT Get vector generated so far.
SUBB #C2 Player is to the left. Subtract 2 to give 2 pixel leftward displacement.
STD VCUT Store as resulting vector.
RTS Return.
C83 LDD VCUT Player is to the right. Get vector generated so far.
ADD #02 Give 2 pixel rightward displacement.
STD VCUT Store as resulting vector.
RTS Return.

* RNDVEC - Generates a 'random' vector. Result stored in VOUT.

RNDVEC
LDA 12554 Generating a 'random' vector out of whatever is laying around.
INC RAO1 Change this.
ACDA RAO1 Modify that.
ECRA 2/X Scramble it up in the pan.
STA TVEC And put it here for a minute.
CLR VCUT Gonna make the vector in pieces, want to start with a zero vector.
CLR VCUT2 Clear second byte of vector.
ANDB #C1 Gonna build parts of the vector by checking the bits of the number we just made up.
BNE VC1 If the 8th bit=0 branch.
LDD #0FF8C Not = 0, give -1 pixel Y displacement.
STD VCUT Store Y displacement of vector.
BRA VC2 Go see about X displacement.
VO1 LDA TVEC Get the number generated above.
ANDA #C2 Check the 7th bit.
BEQ VC2 If it equals 0, the vector will have no vertical displacement at all. Branch.
LCD #128 Give vector +1 pixel Y displacement.
STD VOUT Store as vector.

LDA TVEC Give X displacement.
ANDA #C4 Check 6th bit.
BEQ VC3 If = 0, no leftward X displacement. Branch.
LCD VOUT Get vector generated so far.
SUBB #C1 Give leftward displacement.
STD VOUT Store as vector.
BRA VC4 All done, branch to bottom.

LDA TVEC Check for rightward displacement.
ANDA #C8 Check 5th bit.
BEQ VC4 If = 0, no X displacement. Branch.
LDD VOUT Get vector generated so far.
ADD #C1 Give rightward displacement.
STD VOUT Store as vector.

LDD VOUT Cont want 0 vectors, gonna check it. Get vector.
BEQ RNDVEC If = 0, go up and start over.
RTS Non-zero vector generated. Return.

* ROVYSH - Builds an overlay shape from the pixels on the screen
* into the instructions pointed to by STSH. Building starts where
* the cursor is positioned, RLOC and RBIT. The shape instructions
* are actually modified in the pixel bits.

ROVYSH LCD RLOC Sub builds restore overlay for shape. Shape's addr in STSH, real loc in RLOC and bit set in RBIT.
STD TLOC Store real vidram loc here for working storage.
LDA RBIT Get bit set.
STA TBIT Store as temporary bit set.

LDD STSH Get the overlay shape's addr into the index reg.
LOA >>Y Get first instruction byte for overlay.
BGE YC1 If not less than zero, there is more to do. Branch.
RTS Instruction less than zero then done, return.

ANDA #$40 Check to see if a pixel is written.
BEQ YC2 If result is zero, it is only a cursor movement instruction. Branch below.
LCA #$C0 This is a pixel mask.
LCB TBIT Get the bit set. Will shift pixel mask right TBIT times.
BEQ YC3 If bit set = 0, skip next shift instructions.

TC4 LSR A Shift right.
SRA A Shift right.
DEC B Decrement bit set count.
BNE YC4 If 8 not equal to zero, more shifting to do. Branch up.

BEQ YC7 If = 0, will merely clear pixel in instruction byte. Branch to do that.
LDB #C3 Set non-zero, gonna shift pixel value all the way to the right to build instruction byte.
SUBB TBIT Subtract the bit set from the number 3.
BEQ YC5 If result = 0, pixel is all the way to the right, branch.

TC6 LSR A Shift right.
SRA A Shift right.
DEC B Decrement the shift counter.
BNE YC6 If not done shifting, branch up.

TC5 LCB >>Y Get the instruction byte again.
ANDB #$FC Clear pixel out of the byte.
STB >>Y Put it back.
ORA >>Y Or the pixel we just shifted into the instruction byte.
STA ,Y  Store result as new shape instruction.
BRA YC2  Branch down.

YC2  LCA #$FC  Pixel was = 0 = black, load a mask into A.
      ANDA ,Y  Get all but pixel from instruction byte.
      STA ,Y  Store as instruction byte. Pixel = 0 = black.

YC2  JSR NXTSET  Call subroutine to move "cursor".
      BRA YCO  Branch up to do more.
SETDP 524 Direct page register loaded w/$24 in main program.
START ECU 52500 The beginning of everything.
REALCO ECU 52FCB Subroutine for translating a screen loc into a real vidram loc and bit set.
WRTHP ECU 530B8 Subroutine for writing a shape on the screen.
SHPADR ECU 52EB5 Subroutine for fetching shape instruction address.
CREAC ECU 5245A Number of chain reactions.
CMCH ECU 52459 Number of computer hits.
YLH ECU 52458 Number of your hits.
TCREAC ECU 52456 Total for chain reactions.
TCOMH ECU 52454 Total for computer.
TVURH ECU 52452 Total for player.
MENL ECU 52451 Player's men left.
PSCH ECU 52446 Proposed screen location.
TMP2 ECU 52431 Working temporary.
TMP1 ECU 52432 Working temporary.
TEMX ECU 52433 Working temporary.
RND1 ECU 5242F Working temporary.
RMP ECU 52430 Working temporary.
RND2 ECU 5242E Working temporary.
RND3 ECU 52420 Working temporary.
RND4 ECU 5242C Working temporary.
RND5 ECU 52428 Working temporary.
ME1 ECU 52458 Address of "Spare!".
ME2 ECU 5245C Address of "game over".
ME3 ECU 5245F Address of "This Round!"
ME4 ECU 52461 Address of "Chain Reaction:"
ME5 ECU 52463 Address of "Computer Hits"
ME6 ECU 52465 Address of "Your Hits"
ME7 ECU 52467 Address of "Reaction Total"
ME8 ECU 52469 Address of "Computer Total"
ME9 ECU 5246A Address of "Your Total"
ME10 ECU 5246D Address of "Your High Score"
ME11 ECU 5246F Address of "00"
MEEX ECU 52471 Address where text string for numbers to be displayed is built.
YURMS ECU 52473 Your high score stored here.
TYURL ECU 52453 Low byte of your score.
TCOMH ECU 52455 Low byte of computer score.
TREAC ECU 52457 Low byte of chain reaction score.
SETPTR ECU 52400 * Address of next set of overlaid data.
CRG ECU 52CC4 Start here.

* TALLY - Subroutine puts up the scoreboard and the updated scores. Also writes "game over" when applicable.

TALLY LCD #1537 For putting up score board. 1537 is screen loc where the message "This Round!" will go.
STD PSCR Store as screen loc.
LDX ME3 Get the start of the message into the index pointer.
JSR WRTHES write it on the screen.
LCD #2817 Screen loc where "Chain Reactions" will go.
STD PSCR Store as screen loc.
LDX ME4 Get the start of the message into the index pointer.
JSR WRTHES write it on the screen.
LCD #4097 Screen loc where "Computer Hits" will go.
STD PSCR Store as screen loc.
LDX ME5 Get the start of the message into the index pointer.
JSR WRTHES write it on the screen.
LOD    #5377 Screen loc where "Your Hits" will appear.
STD    PSCR Store as screen loc.
LDX    ME6 Get the start of the message into the index pointer.
JSR    WRTMES write it on the screen.
LOD    #7041 Screen loc where "Reaction Total" will go.
STD    PSCR Store as screen loc.
LDX    ME7 Get the start of the message into the index pointer.
JSR    WRTMES write it on the screen.
LOD    #3321 Screen loc where "Computer Total" will go.
STD    PSCR Store as screen loc.
LDX    ME8 Get the start of the message into the index pointer.
JSR    WRTMES write it on the screen.
LOD    #6601 Screen loc where "Your Total" will go.
STD    PSCR Store as screen loc.
LDX    ME9 Get the start of the message into the index pointer.
JSR    WRTMES write it on the screen.
LOD    #10881 Screen loc where "Your High Score" will appear.
STD    PSCR Store as screen loc.
LDX    ME10 Get the start of the message into the index pointer.
JSR    WRTMES write it on the screen.
LOD    #7148 Screen loc where "00" will go.
STD    PSCR Store as screen loc.
LDX    ME11 Get the start of the message into the index pointer.
JSR    WRTMES write it on the screen.
LOD    #8428 Screen loc where "00" will go.
STD    PSCR Store as screen loc.
LDX    ME11 Get the start of the message into the index pointer.
JSR    WRTMES write it on the screen.
LOD    #9708 Another "00".
STD    PSCR Store as screen loc.
LDX    ME11 Get the start of the message into the index pointer.
JSR    WRTMES write it on the screen.
LOD    #10988 Screen loc where another "00" will go.
STD    PSCR Store as screen loc.
LDX    ME11 Get the start of the message into the index pointer.
JSR    WRTMES write it on the screen.
CLRA Clear out high byte of D register.
LDB    CFEAC Get # of chain reactions. Gonna build a text string.
JSR    BCBUFF Build string for WRTMES from D register.
LOD    #2919 Where # of chain reactions will go.
STD    PSCR Store as screen loc.
LDX    ME11 Get start address of number into X register.
JSR    WRTMES write the number on the screen.
CLRA Clear out high byte of D register.
LDB    CCNH Get # of computer hits.
JSR    BCBUFF Turn it into a text string for WRTMES.
LOD    #4199 Where # of computer hits will go.
STD    PSCR Store as screen loc.
LDX    ME11 Get start address of number into the X register.
JSR    WRTMES write the number on the screen.
CLRA Clear out high byte of D register.
LDB    YLRH Get # of your hits this round.
JSR    BCBUFF Turn it into a text string for WRTMES.
LOD    #3479 Screen loc where # of player's hits will appear on the screen.
STD    PSCR Store as screen loc.
LDX    ME11 Get start address of number into the X register.
JSR  WRTME5
LDB  TCREAC
LDA  TREACL
ACDA  CREAC
JSR  BFIX
ACDA  CREAC
JSR  BFIX
ACDA  CREAC
JSR  BFIX
LDB  A:B
STD  TCREAC
JSR  BCBUFF
STD  #132
STD  PSCR
LDX  ME0X
JSR  WRTME5
LCA  #6
STA  TMP3
LDA  TCOMM
LDA  TCOMHL
UFX  TMP3
ECX  UFF
BEQ  UGG
BRA  UFF
EXG  A:B
STD  TCOMM
JSR  BCBUFF
LDD  #8A
STD  PSCR
LDX  ME0X
JSR  WRTME5
LDB  TYURH
LDA  TYURHL
ADDA  YURM
JSR  BFIX
ADDA  YLRH
JSR  BFIX
EXG  A:B
STO  TYURH
JSR  BCBUFF
LDD  #692
STD  PSCR
LDX  ME0X
JSR  WRTME5
TST  MENL
NE  LC4
LCD  TYURH
CMPD  YLRS
BLO  LC4
STD  YLRH
JSR  BCBUFF
LDD  #10972
STD  PSCR
LDX  ME0X

Unetron listing page: 32
JSR WRTHES
Write the number on the screen.

TST MENL
Check player's men left.

BEO LO5
If there are none left, branch around next instructions.

LDD SETPTR
Time to see if player gets a free ball. Happens every 2nd round, dependent on SETPTR's low byte.

ANDB #C1
Because SETPTR is odd, free ball every 2nd round.

BEO XCU
If equal to zero no free ball this round. Branch around next instructions.

INC MENL
Give player a free ball.

XQU
JSR CCISP
Put up number of spares.

CLR CREAC
Zero out chain reaction count for next round.

CLR COMH
Zero out computer hit count for next round.

CLR YURM
Zero out your hit count for next round.

JSR KBDDAI
Subroutine waits for the fire button.

RTS
Go home.

LO5
LDD #41
Screen loc where "game over" goes.

STD PSRC
Store as screen loc.

LDX ME2
Get address of "game over" into X register.

JSR WRTHES
Write it on the screen.

JSR KBDDAI
Wait for the fire button.

PULS X
At present we are in a subroutine. This will pull two bytes off the hardware stack and <more>

* WRTHES - puts a string of characters up on the screen (always
* text in this application). Leading zeros are stripped off of
* the text. The start of the string is pointed to by the X register;
* the end is denoted by a shape number of zero. Each shape in the
* string is placed on the screen 5 spaces apart.

* WRTHES
CLR RND4
Sub for writing messages. RND4 is a flag to wipe out leading zeros. (see below)

MCO
LDA ->
Get byte of message.

BGE MC1
In this case if byte >= 0, done with message.

CMPA #140
Is it a "0" (text zero that is).

BEQ MC10
If so, branch so as not to set RND4.

 INC RND4
Make RND4 non-zero.

M10
TST RND4
If flag is set, a character not = 0 has been encountered.

BNE M11
Not equal to zero, skip next instruction.

LOA #208
A leading zero or substitute for a blank. (blank = 208)

M11
JSR SHPADDR
Look up the character shape's address.

JSR REALEC
Translate the value in PSCR into bit set and vidram location.

JSR WRTHSP
Write the character.

LDD PSRC
Gonna move to the right. Get the screen loc.

ADDD #5
Move to the right 5 pixels.

STD PSRC
Put it back for the next character.

BRA M00
Branch up to do more.

M01
RTS
Done. Return.

* BCBUFF - A bcd number is passed in the D register and turned
* into the corresponding text string for display by WRTHES.
* The resulting 4 byte string is written at MEEX.

* BCBUFF
STO TEMX
For turning a BCD number into text. D reg holds the number. Store it here.

LDX MEEX
Get address of location where will build the string.

ANDA #1FO
Isolate first BCD digit of number.

LSRA
Shift right. Must get digit so we can add to 140 (="0") and end up with a shape number.

LSRA
Shift right again. Note: shape number must be even.

LSRA
Shift again.

ACDA #140
Add to number for "0" to offset.
STA /X+  Store in area where string is being built.
LCA TEMX  Get the original BCD number again.
ANDA #30F  Get 2nd digit.
LSLA  Shift left to get an even number.
AEOA #14C  Add offset of “0”.  Note: 140=”0”, 142=”1”, 144=”2” ...
STA /X+  Store in area where string is being built.
LCD TEMX  Get original BCD number again.
TFR B/A  Make a copy of the low byte...
STA TEMX  ... and put it here where we can get it easily.
ANDA #30C  Isolate 3rd BCD digit.
LSRA  Move right with first digit.
LSRA  Shift right.
LSRA  Shift left to get an even number.
ADDX #140  Add offset of character “0”.
STA /X+  Move where string is being built.
LDA TEMX  Get byte stored away a nano-second ago.
ANDA #30F  Isolate 4th (last) digit.
LSLA  Shift left to get an even number.
AEOA #140  Add the offset.
STA /X+  Start where string is being built (last character at MEXK)
CLRA  Need a zero.
STA /X  Store in string to signal end of string.
RTS  Go home.

* KBDWAI - Goes into a busy wait and returns when the fire button is pushed.

MBDWAI  LDA 65280  Get location where fire button is mapped.
CMPS #255  If = 255, button not being pushed.
BEQ KBDWAI  Not being pushed then branch up.
CMPS #127  If = 127, button not being pushed.
BEQ KBDWAI  Not being pushed then branch up.
RTS  Button was pushed.  Return.

* BFIX - Performs the DAA instruction on the whole D register (it is only defined for A). Subroutine is called with high byte and low byte of D already in reverse order.

BFI X  DAA  Routine for doing 16 bit BCD arithmetic. Decimal adjust whatever is in A.
BCC TU1  If no carry, branch to bottom to return.
EXG A>B  Switch A and B so can use DAA Instruction on the lower byte.
ADDA #C1  Add carry into byte.
CA D  Decimal adjust.
EXG A>B  Put the bytes back into reversed order.
TU1  RTS  Return.

* CDISP - If the player has any spares left this routine is called to display them in the upper left hand corner of the screen.

CDISP LCD #129  Where message “Spar” will go.
STD PSCR  Store as screen loc.
LDX ME1  Address where message “Spar” starts.
JSR WRTMES  Write the message.
LDD #55D  Location where the shape for player’s character will be written to display spares.
STO PSCR  Store as screen loc.