UNATRON

UNATRON

Documentation contents:

Tape contents:

1. UNATRON
2. BIRD13X
3. FWRK12X

PLAYING UNATRON

Connect the right joystick to your computer and power-up. Rewind the cassette to the beginning and set your recorder to 'play'. Enter CLOADM <cr>. UNATRON and all of it's data tables are loaded into memory. When the computer responds 'OK' enter EXEC <cr> to start the game.

There are two ways to play it. You can try to see as many screens as possible or you can play for a score. Either way, you will find UNATRON a challenge which will take time to learn.

ATOMS: As you start a round of UNATRON you and one large atom are alone on the screen. As you might have guessed, your job is to split the atom. The atom splits and two totally different atoms result. This will keep up until there are 32 small atoms. They will disappear when hit. You cannot split any atoms of a given type until all atoms of the previous type are gone.

NEUTRONS: As the atoms progress to smaller types they give out neutrons when fissioned. A neighboring atom, if hit by a neutron, will split and also release neutrons. You can see how chain reactions start. When an atom splits from a hit with a neutron, you don't receive any points. The credit instead goes to a chain reaction total.

COMPUTER GUNS: On each of the different screens you will notice two stationary objects appear and emit a stream of particles. These are the computer's guns. The particles are the computer's shots. A fission caused by the computer is credited to the computer. That gives us a total of three scoring parties; you, chain reactions and the computer.

MINES: You may think of the mines as ions if you wish. They have a negative charge, you have a positive one. Mines have a great affinity towards you. In fact, interaction between you and a mine is explosive. If you are hit you lose one of your spares. The game will pause and show you how many you have left. That number includes the one you are about to use. The game is over when you lose your last spare. The best protection is to shoot them. There is a catch though. You don't receive any points and the pieces that splinter away as the mine explodes are more computer shots like the ones coming from the guns. The birth of a mine is noted by a sharp whining sound and the appearance of a cross shaped object. Mines are always born at the exact same spot for a given round.

HOLEMAKER: After a number of mines are born, a holemaker will appear. It aimlessly wanders around cutting jagged holes into the walls. The holes can sometimes be a help, sometimes a hinderance. In the later rounds you actually have to depend on the holemaker to free you. You cannot kill the holemaker.

SCREENS: There are nine screens in this version of UNATRON. In general, they become more difficult for you as the game progresses. After the ninth screen (if you make it that far) play returns to screen 4.

THINGS TO NOTE: Only one type of atom is reactive at a time. These atoms will travel erraticly across the screen. They are attracted to the point between the computer's guns. They are also smart enough to run away from you while you are firing. The transition from one type of atom to another will cause atoms to swarm around the computer's guns.

After you are hit by a mine the screen is re-drawn and all the mines and neutrons are removed.

The shots you fire move in the same direction as you and at twice your speed. If you are moving too quickly you can fire shots right over an object immediately in front of you.

Every atom you fission adds 200 points to your score. Chain reactions and computer hits receive 300 and 600 points respectively.

If you are hit by someone's shot there will be an explosion but you won't be damaged.

An atom or mine need not be hit directly by somebody's shot. It's involvement in the explosion is sufficient to cause a reaction.

Starting with the third round, for every second one you complete you are given another spare.

Whenever there has been a pause in the game, pressing the fire button will move you back into play.

A round is over when all of the 63 possible fissions have taken place.

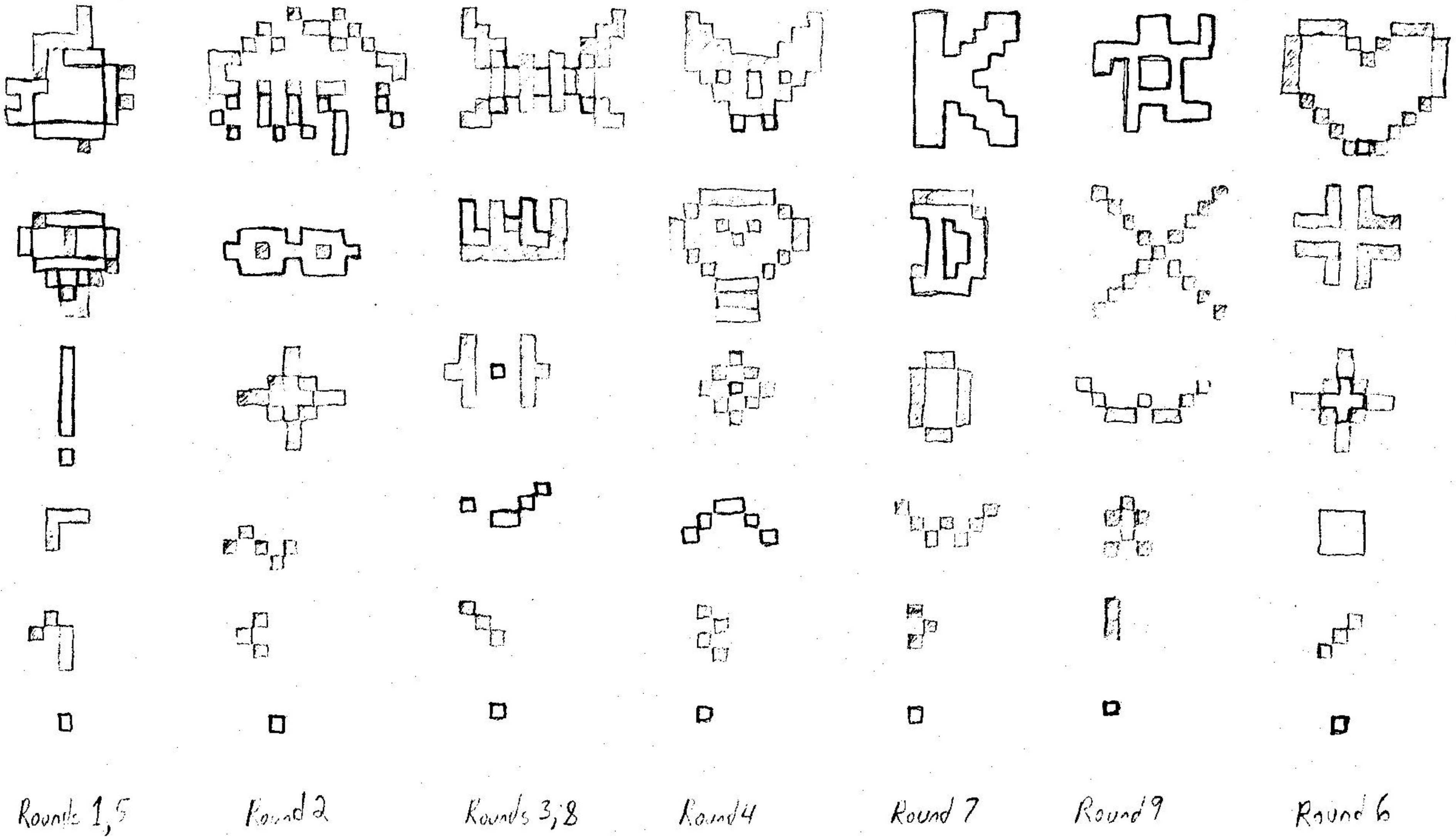
THE SCOREBOARD: The scoreboard shows how each of the 63 fissions for this round were distributed. Also shown are the scores for the game so far and the player's high score. When the last spare is used up, the scoreboard is put up and play is over.

OTHER GAMES: Two other programs are included on the tape with UNATRON. To run them, UNATRON must be loaded FIRST as if it were to be played. Typing CLOADM <cr> again will load the first first of the two programs. A third CLOADM <cr> will load the second. These other programs use all the same graphics routines, graphics data and data structures as UNATRON. The main loop of UNATRON is merely replaced with something else. The two other programs are:

1)BIRD13X - The object of the game is to jump off one of the mountains onto a bird and fly it all the way to the top of the screen. If you fall off, you go splat.

2) FWRK12X - Pushing the fire button gives a fireworks display. If you time it wrong, the firework blows up on the ground.

Atoms



Rough sketches of atoms.

1.0 UNATRON graphics

Managing a dynamic display usually requires that information about the display be in some form other than the display itself. Indeed, for much graphic software the display is no more than a window through which the user can view the models behind it in a meaningful form. In a memory mapped display, such as we are using with this machine, the option of using the display to obtain information about itself is open to us. UNATRON manages the screen by consulting both the actual video ram and remotely maintained data for each character.

1.1 Video addressing

A video addressing scheme or schemes common to both the main program and the graphics handling routines is essential for coordination of the game. The requirements for each differs however.

Working with the main program it is convenient to be able to reference individual points on the screen with the highest resolution possible, i.e. a pixel. The number of pixels equals the number of rows times the number of columns. In mode G3C, as used in UNATRON, this comes out to be 128 X 96 = 12288. So if we want to talk about the fifth pixel in the fourth row we can find it's ordinal location by 128 X 4 + 5 =517. Note rows and pixels number starting from zero.

Video ram is not arranged by discrete pixels. One byte may contain up to eight of them depending on the mode being used. Routines reading and writing video ram would then have to translate a pixel number into a byte number and an offset into the byte whenever locating an individual display point. Referring back to mode G3C and pixel 517, if the start of video ram is known, the actual pixel can be located in memory. Mode G3C gives us four pixels per byte. The start of the video ram is set by UNATRON to 13312 decimal. Here is the formula for translating a pixel number into a video ram location:

Video ram location = INTEGER (pixel number/4) + 13312
Offset into video ram byte = MOD4 (pixel number)

Pixel 517 has a video ram location of 129 + 13312 =13441 and an offset of 1 from the leftmost pixel in the byte.

In UNATRON the pixel number is called the 'screen location'. The video ram byte is called the 'real location' and the offset is called the 'bit set'. A bit set number of 0 indicates the leftmost pixel (XX000000) and 3 indicates the rightmost (000000XX).

In summary, a screen location refers to an individual point on the screen numbered from 0 to 12287. A real location addresses an actual byte of video ram and a bit set describes which of the four pixels in that byte is being referenced.

UNATRON maintains and works with both addressing schemes for all of it's active and inactive characters. Having two methods of keeping track of each takes a little extra memory but yields a savings in time in that one need be calculated from the other only once per move. The instances where a screen location is used verses a real location and bit set are well defined.

Managment of a character's movement is done via the generation of a vector for the character. A vector describes what direction the character is moving in terms of it's screen location. For instance, given that every 128 pixels constitutes a row, a vector of 128 (0080) would cause the character to be displaced one pixel in the downward direction. A vector of -1 (FFFF) would likewise move it one pixel to the left. It follows that -1 + 128 = 127 (007F) would yield a vector to move the character diagonally. Deciding the new location is merely a matter of adding the character's vector to it's screen location. Screen locations also have purpose when generating vectors between characters and points on the screen and in determining the distance between them.

Where the graphics routines are concerned, the real locations and bit sets are used exclusively. The real location allows easy access to the actual video ram byte and the bit is used in preparing masks for operation on that byte.

Figure 1.1 samples the relationship between screen

screen location	real location	bit set
0	13312	0
1	13312	1
2	13312	2
3	13312	3
4	13313	0
5	13313	1
...	...	...

Figure 1.1 - Relationship between screen locations and real locations plus bit sets.

locations and real locations plus bit sets.

1.2 Character shape instrucions

The notion of a cursor applies well to the job of writing, reading, erasing and checking for coincidence of shapes on the screen. A cursor has a unique location at any given time. We are used to seeing the cursor when interacting with a computer but can imagine it's existance even if it is not visible. The shapes in UNATRON are constructed via a cursor.

Character designs stored in memory describe what a shape will look like one instruction at a time. Each of these instructions takes one byte, each describes what to draw and how to move the cursor. The number of these required to construct a shape depends on the size of the shape. Encoded in the instruction bytes are bit patterns which describe what the pixel to be written looks like (if one is to be written)

```
Bit #     .-0-.-1-.-2-.-3-.-4-.-5-.-6-.-7-.
          |   |   |   |   |   |   |   |   |
          | E | W | L | R | U | D | p   p |
          | N | R | E | I | P | O | i   i |
          | D | I | F | G |   | W | x   x |
          |   | T | T | H |   | N | e   e |
          |   | E |   | T |   |   | l   l |
```

Figure 1.2 - Instruction bits for shape construction

and where next to move the cursor.

Bit 0, when set, signals that we have reached the end of the instructions for writing this shape. Using one byte arithmetic this condition is easy to check for. If bit 0 is set, considered as an integer, then the byte is negative, hence end of shape.

Bit 1 tells whether or not a pixel is to be written. It is wholely possible to move the cursor without setting a pixel.

Bits 2 and 3 control the horizontal movement of the cursor. If bit 2 is set, the cursor is placed left one pixel. If bit 3 is set it is moved to the right. Movement of the cursor always occurs after writing a pixel if one is to be written.

Bits 4 and 5 control the vertical movement. When 4 is set the cursor is moved up one pixel and of course bit 5 moves it down. Combinations of vertical and horizontal movement are completely legal.

Bits 6 and 7 are the pixel itself. If bit 1 is not set then these two are ignored. Otherwise the value (00 - 11) is written at the point on the screen where the cursor is positioned.

Consider the following instructions:

```
 47 = 0100 0111  write pixel 11, move down
 04 = 0000 0100  move down
 62 = 0110 0010  write pixel 10, move left
 55 = 0101 0101  write pixel 01, move down and right
 62 = 0110 0010  write pixel 10, move left
 43 = 0100 0011  write pixel 11, dont move cursor
 FF = 1111 1111  end of shape
```

These instructions describe a dot with a multi-colored rectangle under it. Jumping ahead a little, the shape your character starts with is #6. (I happen to know, its in the source and Appendix A). Shape number 6 looks like a V. The memory address of the instructions to draw this shape is

$101E. If ycu type in the seven bytes given above at that address and start up the game, lo and behold a dot with a rectangle uncer it!

1.3 Graphics subroutines

Before writing any shapes on the screen we have to dispence with some prerquisites. First we have to set where it is to be written, that is to say, where to position the cursor initially. Memory locations RLCC and RBIT are used to hold the real video ram location and bit set of a point on the screen. As far as the graphics routines are concerned, setting the values there sets the location of the cursor on upon entry tc the routines.

Secondly, we have to somehow tell those routines where to find the instructions for the construction of the shape. Each rounc accesses one of seven shape tables. Each of the tables contains consecutive two byte addresses of the starting points of shape instructions stored in memory. The 16 bit word at SHTBL contains the location of the shape table being accessed this round. For arçument say SHTBL is equal to $8C0. If we wanted to know the address of the instrucions for drawing shape #6 wouldn't it be convenient to be able to add the 6 to SHTBL = $800 + 6 = $806 and look it up there? Well, thats exactly how it is done. There is a subroutine, SHPADR, which does this process and puts the address in a location STSH (start of shape). We need only pass it the number of the shape we want to look up.

Before returning to discussion of the graphics routines let me just say a few things abcut shape numbers. Shape numbers are always even. This is because the addresses in the shape table are two bytes long. Asking for shape number 3, for instance, would cause a lookup which would return part of the address for shape number 2 anc part for shape number

```
-  Shape number:
S    0     |    2     |    4     |    6     |
H----------|----------|----------|----------|
T msb:lsb  | msb:lsb  | msb:lsb  | msb:lsb  |
B----------|----------|----------|----------|
L               ^ shape #3^
                | illegal |
```

Figure 1.3 - Shape numbers must be even

4. Each shape table can contain a maximum of 128 addresses for cistinct shapes. There is no reason why different shape numbers can't point to the same instructions. In fact, in UNATRCN they often do.

We have now set up the cursor and located the instructions for the shape we want to write on the screen. What next? Write the shape by calling a subroutine named WRTSHP.

In all there are four subroutines which access video ram. Each handles a distinct task. WRTSHP, as explained above, draws a shape on the screen.

The characters of UNATRON have the appearance of being solid. When one collides with another or bumps into a wall the physical presence of the impeding object seems real. If prior to writing a shape we take the instructions for constructing it and trace them out invisibly we can determine if it would be drawn on top of at least one pixel which was already lit. The result of such an operation could be used to determine whether the character should bounce, explode or any number of things depending on it's nature. The subroutine OKMOV does all the handling for this. It's required setup is the same as for WRTSHP. The condition code register is modified upon return to give the status of the check for coincidence.

Sometimes we want to temporarily write on top of something which is already on the screen without destroying it. Take for example an explosion. We want to write what looks like a blast and then erase it. If the explosion occured on top of a wall the erasure would take a piece of the wall with it. For this reason UNATRON maintains what are called Overlay Shapes. Overlay shapes are no different from any other shape in their instructions, rather in the way they are used. The subroutine BOVYSH reads pixels from the screen in a pattern described by the cursor movements in the shape instructions and stores them in the pixel bits of each. If we wanted to draw a blast and then restore the screen we would need an overlay shape with the same cursor movements as the shape of the blast itself. The overlay shape would need to be recorded from the screen with a call to BOVYSH, the blast written by WRTSHP and the overlay written in the same spot with another call to WRTSHP. The subroutine BOVYSH uses the same setup data as WRTSHP and OKMOV.

The last of the four graphics subroutines directly accessing video ram is called ANTISH. Prior to moving a character or if one is destroyed it must be erased. ANTISH takes the shape instructions and writes them pixel for pixel in black. (In other words, the antishape is written) Like the other three routines ANTISH gets the shape instruction address from STSH. The location of the cursor is not taken from RLOC and RBIT however. The point on the screen where the antishape is to be written is extracted from the c-list. The c-list is explained in detail in section 2.1 but a short explanation here coulon't hurt.

Every living character has information about it's position and status stored along with it in one master list. The video real video ram location and bit set for each can be found there. These describe the cursor location where the character is written and are directly accessed in erasing it.

In summary for this section, there are four subroutines which access the video ram exclusively. WRTSHP draws a shape on the screen. OKMOV checks for coincidence. BOVYSH builds an overlay shape from the video ram. ANITSH erases active characters from the screen.

2.0 Character identities

UNATRCN has the capability of keeping track of up to 85 disctinct active characters at one time. How a character of a given type acts and reacts is hard coded but the number and distribution is a matter of what happens during play. Data for each character is maintained and updated each cycle of the program.

2.1 Character list

To keep track of animate characters UNATRON maintains a character list or c-list for short. It is located just below the video ram from 12547 to 13311. Every nine bytes of the list constitutes space for an entry into the list. I will explain what each of the nine bytes is used for and return to discussion of the list as a whole later.

Characters are identified by a one byte even number ranging from 2 to 128. The main loop uses this number to determine how a character should react, i.e. holemakers shoulc eat walls, mines should attack etc. The number for a character often tells what shape it is. That is to say, for character #44 draw shape #44. Simple enough.

Cf the rine bytes of each c-list entry, byte 0 is the shape number.

For each character, the screen location it presently holds is stored in bytes 1-2.

Byte 3 is called the wobble byte. It is described in more detail in section 3.3 but briefly, it is used by the main program to keep track of special features of the character.

Bytes 4-5 contain the character's present video ram address.

Byte 6 contains the bit set for the video ram address in bytes 4-5.

Bytes 7-8 hold the character's present vector. The vector describes what direction it is moving with respect to it's screen location.

To locate a character or to scan the c-list we start at the beginning, 12547, and step through nine bytes at a time. The program is set up so that the first entry is ALWAYS the player's character. Often the player's vector, screen location etc. is accessed by address as opposed to addressing offset to an index register as with other c-list entries. An entry with shape number C is reguarded as a hole in the list and may be filled in by any subsequent entries.

The c-list is available to the main loop and the vector generating routines. Characters are added by a call to a subroutine ADDCHQ. The components to be added, i.e. shape

```
---------------------------------------------
| 0 || 1    2 || 3 || 4    5 || 6 || 7    8 |
  ^       ^      ^       ^      ^       ^
  |       |      |       |      |       |_VECTOR
  |       |      |       |      |_BIT SET
  |       |      |       |_REAL LOC
  |       |      |_WOBBLE
  |       |_SCREEN LOC
  |_SHAPE NUMBER
```

Figure 2.1 - Nine bytes of a c-list entry

number etc., are picked up from the local equates as shown in figure 2.2. Deletion is simply a matter of setting the

```
PSHP -> ,X   ;character shape number
PSCR -> 1,X  ;screen location
TMP1 -> 3,X  ;wobble byte
RLOC -> 4,X  ;real vidram location
RBIT -> 6,X  ;bit set
VOUT -> 7,X  ;vector
```

Figure 2.2 - Clist assignments from equates

c-list entry's shape number equal to 0.

2.2 Who's who

Appendix A is a list of the shape numbers and a description of which shape corresponds to each. As you can see, more than one shape may belong to an individual character. Many of the shape numbers are never entered into the c-list to become active characters though any even numbered character is legal. If the shape number for the letter "a" were added to the queue for example, it would be treated like it was someone's shot. The shape on the screen would be an "a" and it would move until it hit something. The subset of active shapes that is used you will recognize from the game. The difference in their actions comes from how the program discriminates them one from another.

In the main program, just under the label CHARS, is where the discrimination takes place. These are the categories characters are lumped into:

1) CURC - current character or reactive atom. The equate CURC holds the character number for the reactive atom. If the entry we are looking at is an occurance of CURC, actions may be taken to make it either run from the player or head towards the computer's guns (AIM).

2) Mines - mines are always character #106

3) Holemakers - always number 1C2

4) Anything >= 42 is either someone's shot or a neutron, etc. The common denominator is that when it hits something it explodes. Say the character is number X (X>=42). Shape number X+4 will be the explosion and X+2 will be the overlay for the explosion. This is true for all explosions of characters >= 42.

5) All else - everything else is considered a non-reactive atom. It will move in one direction until it hits something, recieve a new vector and start a period of wobbling between shape X and shape X+2.

3.0 Animation

Most movement in UNATRON is linear and has a conditional attached where the movement is blocked. The methodology for moving all characters is the same, the motivations are different.

3.1 Vectors

Let's say you are at spot X0. You want to move to spot X1 and after that X2. Presumably X1 and X2 are in a straight line path away from you and the same distance apart as you are from X1. The point X1 is some number of pixels, say A, away from you in the horizontal direction and B pixels away in the vertical direction. It is simple now to generate a constant which when added to X0 gives X1 and when added to X1 gives X2, i.e. a vector. At a resolution of 128X96 (G3C) the vector we want is A + 128 X B. There is no reason why the vector cannot be less than zero.

There are some practical considerations; A vector of too great a magnitude will cause movement from X0 to X1 to X2 to look jerky or perhaps unrecognizable. Zero vectors, while completely legal, are avoided for most UNATRON charaters. Once one has stopped moving it can be very hard to get it to go again!

Vectors may be generated randomly or may be directed. Three subroutines exist for these purposes. The first, RNDVEC, produces a vector with a maximum displacement of one pixel in the X and Y directions.

The second, NEWVEC, generates one towards the player's character or whoever holds the first position in the c-list. Note that taking the negative yields a vector away. The maximum displacement is two pixels in two directions.

The third, DWNVEC, gives a vector towards the location stored in the two byte word equated to AIM. Generally AIM is set to the point between the computer's guns.

The results for all three subroutines is placed in VOUT. Each character in the c-list has it's own vector stored along with it. The player's vector is of course generated from the joystick position.

3.2 Movement

In this section the generic process of moving a character is outlined. All types of characters have other features specific to themselves. Those are outlined in section 3.3.

Before moving any character the first thing to do is erase it. If we didn't, the best we could hope for is a smear as it made it's way across the screen. A call to SHPADR locates the instructions for the shape. The subroutine ANTISH takes the real location and bit set of the character we are looking at and uses it as the starting cursor location for the erasure. The shape has now been blacked off the screen.

A call to the subroutine NEWLOC takes the character's vector and screen location and adds them together. The result is a new screen location placed in PSCR. For the sake of the graphics subroutines a call to REALCO translates the screen location in PSCR into a real location and bit set which are placed in RLOC and RBIT respectively.

Now we are ready to try and make the move. The subroutine OKMOV is called to see if at the location where we wish to place the character there is not at least one pixel already lit. If there is, the move is flagged as not ok. A random vector is generated and the procedure tried again and then once more if neccessary. If the character still cannot be moved it is re-written where it was when we started and the move is given up for the time being.

If OKMOV returns ok, a call to WRTSHP draws the character on the screen at the new location. The vector, screen location, real location and bit set used are picked up and placed in the c-list along with the character to complete the process.

3.3 Collisions and special features

Atoms: Whenever an atom collides with something else it starts a period of wobbling. Say the atom has character number X. Every second cycle shape number X+2 is being written in place of shape X to create the wobble.

When the collision is detected byte 3 of the character's c-list entry is made non-zero. It is decremented every cycle from then until it reaches zero, the odd values causing the wobble. Byte 3 of the c-list entry is called the wobble byte.

Mines: After a collision a mine will enter a period of aimless wandering (provided it wasn't the player who was hit). The wobble byte for the mine is set to a non-zero value and decremented each round hence. When it reaches zero again the chase resumes.

Holemakers: Upon collision the holemaker is given a new random vector. At the point of impact a shape of completely black pixels, a hole, is drawn.

Shots and neutrons: Collision here causes the shape to be deleted from the c-list and an explosion to be drawn. The blast occurs at the first pixel found to already be lit by OKMOV when it checks. The first reactive atom or mine, if any, found in a crude rectangular area around the blast is also deleted and handled appropriately. Only one atom or mine may lose it's life in a single explosion.

Inert shots: When a mine is exploded the peices which fly away are temporarily inert computer shots. The truth is, they are actually different characters altogether. The inert shots are loaded into the c-list with a wobble byte of 5. When 5 cycles are over their shape numbers are replaced with those for the actual computer shots. This is done because in the small area of an explosion a fair number of the shots released would explode on one another if they were not temporarily inert.

4.0 Data

The personality of UNATRON is largely traceable to the data ingested by the program.

4.1 Overlaid data

Round by round the game changes. This is accomplished not by counters or flags but by changing the first level of

One byte equates:	
G1L	Length of shot originating from gun #1.
G2L	Length of shot originating from gun #2.
GCH	Relative chance of computer guns firing.
MAXH	Max # of holemakers to appear on the screen this round.
MAXM	Max # of mines that can appear on screen this round.
MBEFH	Number of mines that must appear before a hole appears.
MINCH	Relative chance of a mine appearing.
MINSPH	The longest number of cycles a mine will wait before chasing player.
MINSPL	The shortest number of cycles a mine will wait before chasing player.
NUMN	Number of neutrons released when a mine is exploded.
Two byte equates:	
AIM	Screen location where CURC is heading for, affinity based on ATRCT.
G1S	Screen loc of where gun #1 appears.
G1V	Vector for shot originating from gun #1.
G2S	Screen loc of where gun #2 appears.
G2V	Vector for shot originating from gun #2.
MANST	Screen loc of where player's character starts.
MSCR	Screen loc of where mines appear.
STBO	Addr of start of screen borders layout.
SETPTR	Addr of next set of overlaid data.
SHP1ST	Screen loc of where first atom starts

Figure 4.1 Overlaid data equates

data driving the program.

There are parameters with values specific to a round which must be consulted repeatedly by UNATRON. Examples are; the screen locations for the computer's guns, the maximum number of mines allowed on the screen, etc. Because the values have to be looked up often, any savings in time in doing that is significant. A very fast method it to store all values in the 256 byte page of memory pointed to by the direct page register. Access to the parameters is direct, as opposed to extended, and hence takes less space and time.

The direct page register is set to page $24 in UNATRON. The first $29 bytes of that page is what is referred to as overlaid data. At the start of each round those $29 bytes are picked up from the memory location indicated by an equate called SETPTR and transferred to $2400 through $2428. The overlaid data controls what shape table will be used this round and where to find the data words to draw the screen among other things. Figure 4.1 is a list of these items and their descriptions. See also appendix B. At present $1F of the $29 bytes are being used.

In addition to the overlaid data, all temporary storage takes place in page $24 also.

4.2 Data tables

1) The addresses of shapes are in 7 seperate tables at $800, $900, $AC0, $B00, $C00, $D00 and $E00. One of these shape address tables is accessed each round. For argument, say the round we are in uses the table at $B00. The address for shape #2 can be found at $B02, the address for shape #4 at $B04, etc.

2) The shape instructions start at $1000 and extend up to $1A2C. There is some free space after that.

3) Screen border layouts start at $2C00 and extend to around $2200. The layout for screen #9 is tucked in at $F00. Border layouts consist of consecutive two byte screen addresses in two groups. The addresses where shape #108 is to be drawn come first, delimited by a negative address and then followed by addresses for shape #110.

4) Overlaid data. The information here controls the difficulty of each round, what screen layout to use, what shape table to access and where to find the next set of overlaid data for the next round. There are nine sets of data, one for each round. Each is $29 bytes long. Sets start at $2200 and are overlaid byte for byte starting at $2400. Overlaid EQUates are marked by an asterisk in the source. (See figure 4.1)

5) Text strings start at $1800 and extend to $189D

5.0 Modifications

There are two ways to modify UNATRON. The first, shown by examples in section 5.1, involves changing the data that UNATRON eats. The second, section 5.2, involves replacing the main loop with a different piece of code.

5.1 Data modification examples

1) $221A is the location which controls the number of holemakers allowed in round 1. (See appendix B) It is normally set to 1. Changing the 1 byte value at $221A changes the number of holemakers which will appear in round 1.

2) The address of the screen layout for round one can be found at $2202. It will be found to be equal to $2000. Placing the following four 16 bit words at $2000 will cause the screen layout of round one to have only one horizontal and one vertical brick at the upper left at the screen.

```
   $CC00  ;screen address 0
   $FFFF  ;done with horizontals
   $0000  ;screen address 0
   $FFFF  ;done with verticals
```

3) The shape # for the hole the holemaker writes when it hits something is 104. In the first round, the shape table is located at $800. Adding the 104 + $800 gives $868, the address of the instructions for writing the hole. The instructions for writing an "e" are at $1A1D. If the two byte value at $868 is changed to $1A1D an "e" will be written whenever the holemaker bounces into something.

4) The address of the start of the text string "Computer Hits" can be found at $2463 (from listings). The address is $185E. Copying the following bytes into memory starting at $185E will cause the word "potato" to be printed instead of "Computer hits". Note the bytes are in decimal form this time.

```
   186   ;"p"
   184   ;"o"
   198   ;"t"
   160   ;"a"
   198   ;"t"
   184   ;"o"
     0   ;end of text
```

5.2 BIRD13X - Replacing the main loop.

Load UNATRON into your computer from tape as you normally do. Don't type EXEC this time (though it won't hurt if you do, just hit the reset button). Type CLOADM <cr> a second time. A short program called BIRD13X is loaded on top of and replacing the UNATRON main loop. You can type EXEC <cr> now.

BIRD13X uses the c-list, the same shapes, shape tables and subroutines as UNATRON. The mountains are from round 4, the birds are from round 9 (have you ever seen them before?) The object of the game is to jump off one of the mountains, onto a bird and fly all the way to the top of the screen. If you fall off and plunge too far you go splat and lose the game.

Like BIRD13X, the program in appendix C can be assembled and loaded ON TOP of UNATRON to make a completely different program. It loads 80 dots into the c-list and causes them to alternately cluster and disperse. The program watches the value in TMP3 to decide what the dots should do.

Appendix A - Shape numbers and descriptions

2	Player facing 9 o'clock
4	Player's character facing 3 o'clock
6	Player's character facing 12:00
8	Player facing 10:30
10	Player facing 1:30
12	Player facing 6 o'clock
14	Player facing 7:30
16	Player's character facing 4:30
18	Atom #1 normal state
20	Atom #1 wobble state
22	Atom #2 normal state
24	Atom #2 wobble state
26	Atom #3 normal state
28	Atom #3 wobble state
30	Atom #4 normal state
32	Atom #4 wobble state
34	Atom #5 normal state
36	Atom #5 wobble state
38	Atom #6 normal state
40	Atom #6 wobble state
42	Chain reaction neutron
44	Overlay shape for explosion
46	Explosion
48	Player's shot first shape.
50	Explosion overlay shape
52	Explosion
54	Player's shot 2nd shape
56	Explosion overlay shape
58	Explosion
60	Player's shot 3rd shape
62	Explosion overlay shape
64	Explosion
66	Player's shot 4th shape
68	Explosion overlay shape
70	Explosion
72	Player's shot 5th shape
74	Explosion overlay shape
76	Explosion
78	Player's shot 6th shape
80	Explosion overlay shape
82	Explosion
84	Player's shot 7th shape
86	Explosion overlay shape
88	Explosion
90	Player's shot 8th shape
92	Explosion overlay shape
94	Explosion
96	Computer's shot
98	Explosion overlay shape
100	Explosion
102	Holemaker
104	Hole (shape is black)
106	Mine

108	Screen layout shape 1
110	Screen layout shape 2
112	Inert computer shot (mine disintegration)
114	Unusec
116	When player hit, this piece flitters away.
118	Overlay for above
120	Flittering piece when player hit
122	Overlay for above
124	Flittering piece when player hit
126	Overlay for above
128	Flittering piece when player hit
130	Overlay for above
132	Flittering piece when player hit
134	Overlay for above
136	Computer gun #1
138	Computer gun #2
140	"0"
142	"1"
144	"2"
146	"3"
148	"4"
150	"5"
152	"6"
154	"7"
156	"8"
158	"9"
160	"a"
162	"C"
164	"c"
166	"d"
168	"e"
170	"g"
172	"H"
174	"h"
176	"i"
178	"l"
180	"m"
182	"n"
184	"o"
186	"p"
188	"R"
190	"r"
192	"S"
194	"s"
196	"T"
198	"t"
200	"u"
202	"v"
204	"Y"
206	":"
208	" "
210 - 254	unused

Appendix B - Overlaid data locations

Variable	Round1	Round2	Round3	
SETPTR	2200	2229	2252	...
STBO	2202	222B	2254	...
MANST	2204	222D	2256	
SHP1ST	2206	222F	2258	
G1S	2208	2231	225A	...
G1V	220A	2233	225C	...
G1L	220C	2235	225E	
G2S	220D	2236	225F	
G2V	220F	2238	2261	...
G2L	2211	223A	2263	...
GCH	2212	223B	2264	
MSCR	2213	223C	2265	
MAXM	2215	223E	2267	...
MINCH	2216	223F	2268	...
MINSPL	2217	2240	2269	
MINSPH	2218	2241	226A	
MBEFH	2219	2242	226B	...
MAXH	221A	2243	226C	...
SHTBL	221B	2244	226D	
NUMN	221D	2246	226F	
AIM	221E	2247	2270	

Note that each successive occurance of a data item is $29 bytes from the last occurance. There are presently 9 sets of overlaid data with room for a tenth in the space immediately following the ninth. The location SETPTR tells where the next $29 bytes will be found when this round is finished.

Appendix C - Sample program

```
        SETDP   $24
ADDCHC  EQU     $2FA2
ANTISH  EQU     $308B
NEWLOC  EQU     $3074
REALCO  EQU     $2FCB
WRTSHP  EQU     $30B8
NEWVEC  EQU     $2ED4
RNDVEC  EQU     $2F15
SHPADR  EQU     $2EB5
OKMOV   EQU     $2FE5
PSCR    EQU     $2446
RBIT    EQU     $2443
RLOC    EQU     $2444
VOUT    EQU     $243C
PSHP    EQU     $243B
TMP1    EQU     $2432
TMP3    EQU     $2430
SHTBL   EQU     $241B
        ORG     $2500
        LDA     #$24
        TFR     A,DP
        STA     65478
        STA     65481
        STA     65482
        STA     65485
        STA     65487
        STA     65488
        STA     65472
        STA     65474
        STA     65477
        LDA     #255
        STA     65314
        LDX     #12547
LO1     CMPX    #16383
        BGT     LO2
        CLR     ,X+
        BRA     LO1
LO2     LDD     #$0000
        STD     SHTBL
        LDD     #5898
        STD     PSCR
        JSR     REALCO
        LDA     #38
        STA     PSHP
        JSR     SHPADR
        CLR     TMP3
XO1     INC     TMP3
        LDA     TMP3
        CMPA    #80
        BGT     XO3
        JSR     ADDCHQ
        LDD     PSCR
        ADDD    #01
        STD     PSCR
        BRA     X01
X03     INC     TMP3
        LDX     #12538
X04     LEAX    9,X
        LDA     ,X
        BEQ     X03
        TST     TMP3
        BGE     X05
        JSR     NEWVEC
        LDD     VOUT
        STD     7,X
        BRA     X06
XC5     BGT     X06
        JSR     RNDVEC
        LDD     VOUT
        STD     7,X
XC6     JSR     ANTISH
        JSR     NEWLOC
        JSR     REALCO
        JSR     OKMOV
        BEQ     X09
        JSR     RNDVEC
        LDD     VOUT
        STD     7,X
        JSR     NEWLOC
        JSR     REALCO
        JSR     OKMOV
        BEQ     X09
        LDD     4,X
        STD     RLOC
        LDA     6,X
        STA     RBIT
        LDD     1,X
        STD     PSCR
X09     JSR     WRTSHP
        LDD     PSCR
        STD     1,X
        LDD     RLOC
        STD     4,X
        LDA     RBIT
        STA     6,X
        BRA     X04
        END
```