

### 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 characters. 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 necessary. 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 CKMCV 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.