

Creating Q*bert

It's a game that I have never played, in the arcade or on the Amstrad. I read about it growing up, and being a popular game, it has enjoyed numerous ports to different platforms. Q*bert already has a few conversions for the Amstrad CPC, but I thought I would see if I could develop a version for the Amstrad Plus machines. Using the advanced colour palette and hardware sprites, it should be possible to create a version that is close to the original arcade machine. I installed Mame and downloaded a copy of the arcade original.

I made a copy of my Oh Mummy Resurrected game environment and started modifying it. I will use this as a template to save me having to recreate a lot of functions for the game.

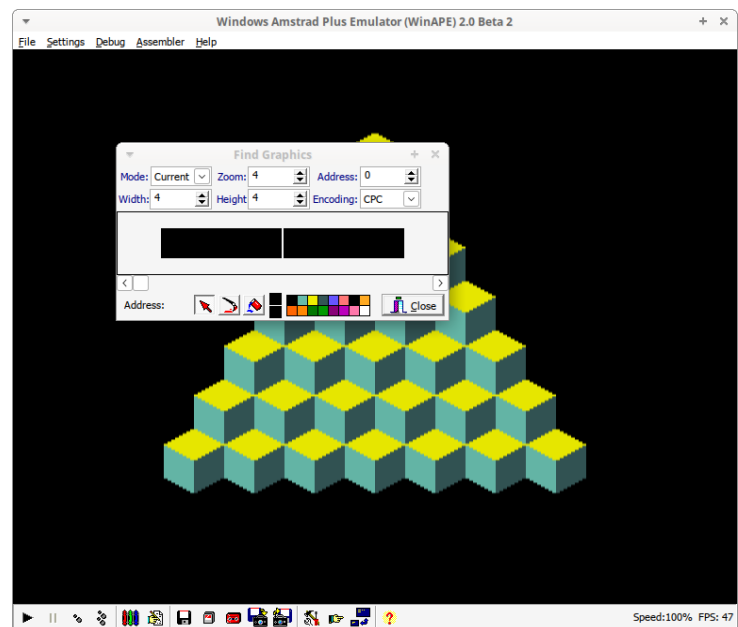
My first goal was to recreate the logo. I loaded up ConvImgCPC and traced out the logo comparing it to a screenshot of the original arcade version. Then saving it as linear assembly code, I could paste it into my game code. Then create a function to draw it to the screen.



I will need to modify my font to be more like the original arcade machine, but I will do this later on.

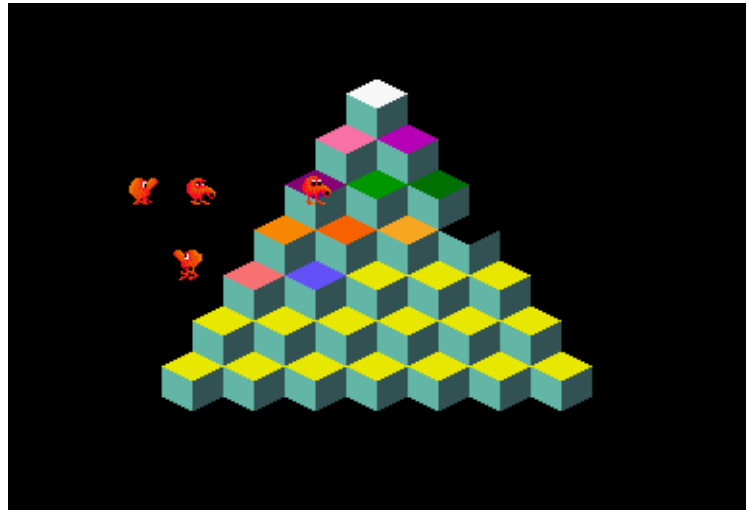
I used Mame to snoop on the original arcade palette. There are just sixteen pens, so it suits a MODE 0 screen resolution. I copied the corresponding values directly into the boot code of the game. I found that the isometric view of the blocks works very well with MODE 0 graphics. This also gives us plenty of colours to work with. So the goal will be to have the main blocks drawn in MODE 0, and the hardware sprites in MODE 1. This will correspond most closely with the original arcade graphics.

I drew an isometric block in MODE 0 resolution in ConvImgCPC and then saved it as assembly code into my game. There will be only one block sprite to save memory. The colours will be modified by a separate function as the sprite is drawn to the screen. This will enable me to change individual blocks to specific colours as required when Q*bert jumps on them.



Hardware sprites

The main sprite in Q*bert is 16x16 pixels, and as there are only 16 colours on screen in the arcade game, I was able to recreate the arcade image pixel for pixel. There are four main images to draw, with two facing the player and two facing away. One image depicts Q*bert standing, and the second one crouching. I am able to create flipped versions of these images when necessary as they are drawn to the screen.



On the hop

Getting Q*bert to hop is a little more complicated. Basically we need variables to store the pixel location of Q*bert, then variables for X and Y momentum, the current direction he is travelling, the current cube occupied as well as the destination cube he is heading for.

When the player pushes a direction, we set the momentum for the sprite in the appropriate direction. The X momentum stays at 1, the Y momentum is set at a large value and decreases each frame. When the Y momentum reaches 0, the direction variable is flipped and the momentum starts to increase again. These momentum variables are added to the pixel coordinates each frame, and gives us a nice arc when he jumps. When the downward momentum reaches a certain level, we check if the destination cube exists (it might be empty space!). If we have a safe landing spot, we reset all the movement and momentum variables and draw the landed sprite.

We should be able to use this function for all the enemy sprites, and just change the location of the block of variables to modify.

Split screen

I wanted to keep the text as MODE 1 as much as possible, so as not to create a clash between the high resolution sprites and low resolution text. I thought about using hardware sprites to create the logo at the top of the screen. But this used up 6 hardware sprites, and other text on the screen would be rendered in blocky MODE 0. So instead I opted for a split mode screen, with the top being MODE 1. I had to move the screen slightly down so the interrupt would be in the right place for the text and not corrupt the MODE 0 cubes.

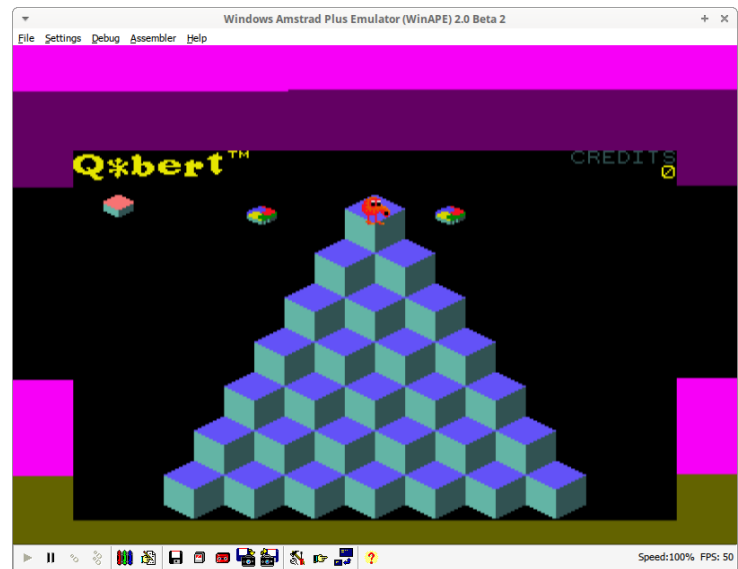
I created a function to draw a mini cube by drawing every other line and every other



byte horizontally. The two disks are hardware sprites, which will need their colours changed through an interrupt.

Disks

The spinning disks, which are platforms to help Q*bert escape from his enemies, are created using hardware sprites. The easiest way to animate them is to create an interrupt that cycles several colours of the sprite palette. This does mean that the disks take exclusive use of 4 colours of the sprite palette, but it saves having to create four separate sprites and copy each one in turn into the ASIC chip. There are only shades of green and purple used for Q*bert's enemies, so hopefully there should be enough colours left.



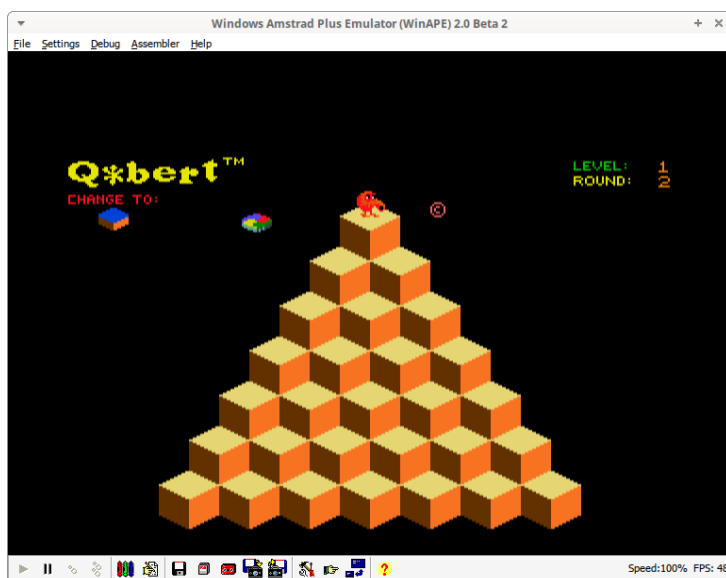
Font

I'm trying very hard to stay as close to the design of the arcade original as possible. So I needed to modify my existing font. Thankfully this was not too difficult as the Q*bert arcade font is comprised of 6x6 pixel letters in an 8x8 grid. So it was just a case of manually recalculating the bytes in each letter. Lower case letters are not used, so I will delete them once I have finished.



Different palettes for rounds

Examining a longplay video on YouTube of the arcade version of the game, I notice that the palette changes in each round. So I created a table of palettes. Each time the player advances a round, a variable is incremented which specifies which palette is to be used for that round. There are basically five colours that are changed on each level. The left and right sides of the cube, the unselected colour, an intermediate colour in the higher levels, and the required colour to complete the level. I had to set aside specific pens in the palette to allow this colour change, so it does not alter the palette in the MODE 1 section at the top of the screen.



Loading screen

I tried converting some of the decals from the arcade cabinet in ConvImgCPC to use as a loader for the game. Unfortunately the primary colours and artistic drawing did not look very professional when converted to the CPC screen resolution. So instead I used just the plain Q*Bert logo and converted it instead. It works very nicely in MODE 0 using the extended Plus palette. The extra shades of yellow and red blend in well and reduce the blocky appearance of the mode.



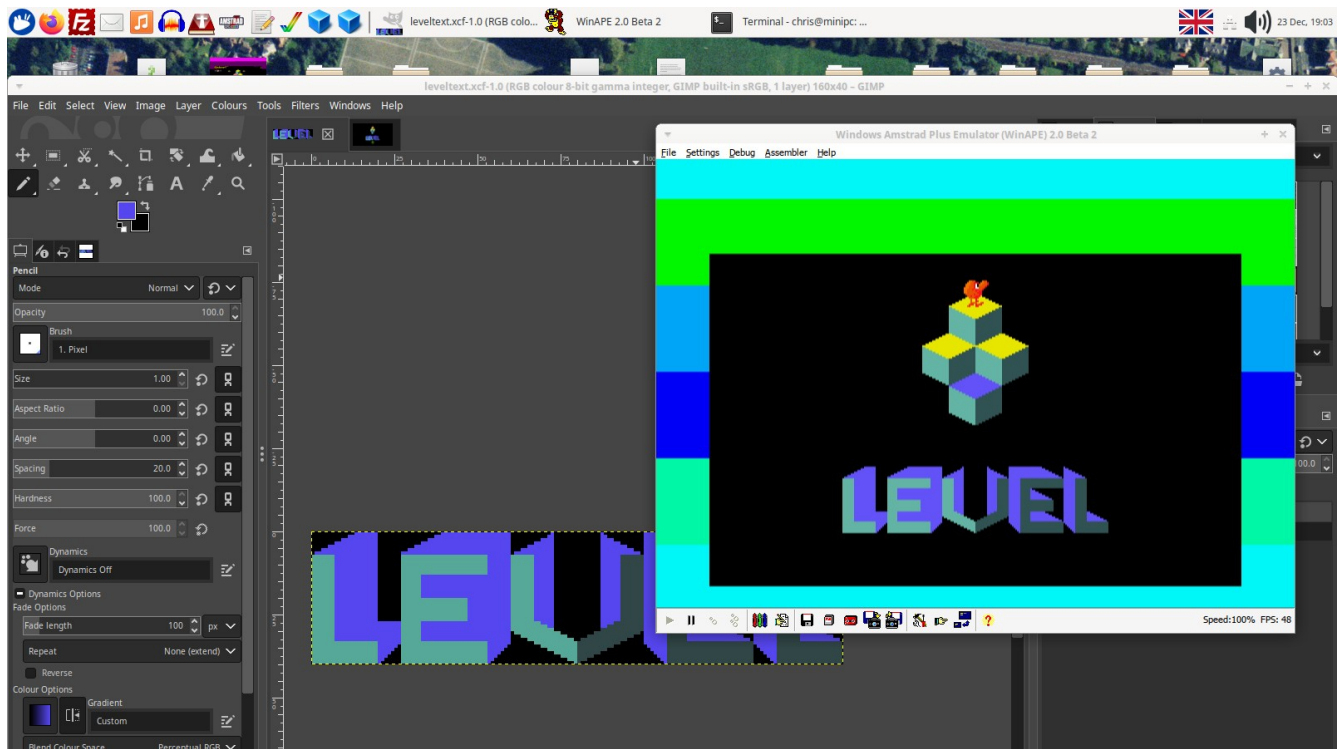
Disks

Working on disks. They are not too difficult to program. I basically created a function that modifies the object grid and inserts an ID where a disk should be located. This prevents Q*Bert from falling off the pyramid when he lands on empty space. Then we move a disk hardware sprite into the correct position on screen. There is a bit of a difference between the position Q*Bert lands on and the disk. I found out, when Q*Bert lands, the disk magically appears under his feet. Then the animation begins with him being transported to the top of the pyramid. It shouldn't be too difficult to replicate.



I have also started work on the new level screen, which contains a few cubes to demonstrate to the player how the level is completed.

New level screen



As part of the graphics of the level text logo uses smaller pixels, I was forced to use MODE 1 to define it. So I used a split screen and set the colours for MODE 1 so they would be the same as the cubes drawn in the next round. I should be able to get the flashing level number graphic in the interrupt below and set the appropriate pens for their specific colour.

Scores

I managed to get the score updating for every square that Q*Bert changes colour. I still have to design the player text above the score, but I am not yet sure whether to use hardware sprites for this, which will allow me to reproduce the correct colours of the arcade, or whether to just make it a single colour. I'm not sure whether I will have enough spare hardware sprites to do it.

Enemy movements

So Q*Bert's movements are working fairly well but the routines are all hardcoded to a single block of variables for the main sprite.

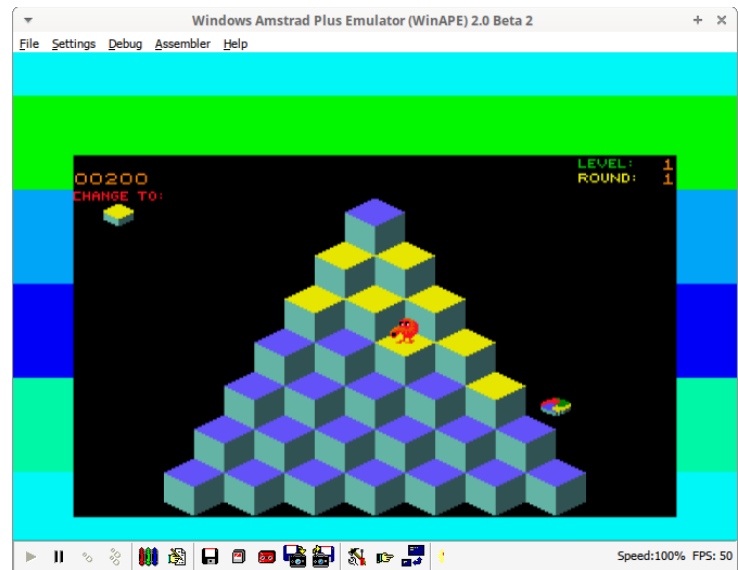
If we want to have multiple enemy sprites bouncing around the pyramid at the same time, we will need to adapt the functions so they can work on separate memory blocks as required. This seems to be fairly simple to do. To do this, we use the IX register, which works like an array in BASIC. For each hardcoded variable, we create a similarly named alias. I.E, a variable called `spriteid` gets a lookup offset called `ix_spriteid`. Each of these aliases are numbered according to their memory location in the data block, `ix_spriteid EQU 0`. In the main program loop, we simply point IX to the start of each sprite's data block. Then it's a case of changing each hardcoded variable name to an IX offset by changing `LD a,(spriteid)` to `LD a,(ix+ix_spriteid)` in the code. It seems to work okay. Although IX is a lot slower than using the direct lookup, but it saves space and us having to work to get HL pointed at the correct place in memory to read a variable. Any routines that are specific to the player sprite can be left as they are.

We will then need to create functions to spawn enemies at different intervals, and also functions to guide them as to how to navigate the pyramid.

Q*Bert jump tuning

I spent a bit of time tuning Q*Bert's jump. The problem is, his jump isn't exactly parabolic, so my X and Y variable function was creating too high an arc for him when jumping from one cube to another. This resulted in him disappearing off the top of the screen when leaping off the pyramid. Examining videos of the arcade version, I saw how his jump is much more shallow from one block to another. So I had to calculate out the pixel movements required to jump from one cube to the next. He needed to move 16 MODE 1 pixels horizontally and 24 vertically. As these numbers aren't evenly divisible, I had to create an offset table that makes him move either 1 or 2 pixels vertically for every 1 pixel moved horizontally. I also created a move counter variable. I set it to 16 when the player presses a direction. It takes exactly 16 moves to reach the next square. When the counter reaches zero, we reset all the movement variables. Once I had him sliding from one cube to another, I created another two offset tables to create the 'bounce'. One for moving up and another for moving down. This offset is added to the sprite's vertical position, but is not saved in the variable so it does not become cumulative. This turns the slide into more of a jump.

This change created a separate problem in that whenever Q*Bert leaps off the pyramid, we still need to make him move in a parabolic arc. So whenever the jump is initiated, we check whether there is a



destination cube. If there is, we call the normal jump function. If there is no destination, we call the parabolic movement function, which moves him in an arc until the bottom of the screen is reached.

Screen size

One of the problems I was having with the Amstrad conversion was the difference in screen size. The arcade screen is 256 pixels high by 240 wide. The unusual aspect, with the slightly taller screen size, allows Q*Bert to get right up to the top of the screen when he is being transported by the disks. I found that whenever my disk function was unable to reach the topmost block without going off the edge of the screen. The standard Amstrad screen size is 320 pixels by 240 pixels. Jason on facebook suggested using the CRTC chip to change the screen size, and thankfully I was able to find some example code that produced this aspect. I did need to change a lot of my lookup tables as everything was skewed, with the screen now being 64 bytes wide instead of 80 bytes. But soon everything was looking well, and we are hopefully nearing a more pixel-perfect arcade conversion.



The big font

The scoreboard and level numbers use a multicoloured big font. Thankfully I found an asset page on the internet that I could download it from, which saved a lot of work. Otherwise I would have had to redraw each letter in ConvImgCPC and import it manually. I imported the sprite graphic for all the numbers and letters in one block into ConvImgCPC and saved it as a MODE 1 assembly data file. As the letters were in three rows I had to create a function that draws the correct image on the screen by skipping so many bytes until we get to the right letter. It seems to work okay, though all the letters are left aligned in their frames. Also some letters are 16 pixels wide, and some are only 14 pixels, so the font is a little unusual. Still, I will try to keep it the way it is for the sake of authenticity.



Player text

I wasn't sure how I would accomplish the shaded player text. Initially I thought of doing it similar to my 'CHANGE TO:' text, using hardware sprites and cycling the colours. Unfortunately this would not work as it would use up four sprites, which we need for the enemy characters, and multiplexing isn't really possible in the interrupts. It takes too long to copy each sprite to the ASIC. Then I thought of doing it in MODE 1 and just keeping it all the one colour. Thankfully there happens to be an interrupt that occurs near the top of the screen, just before we change to MODE 1. So I was able to design the text so it closely matches the original arcade version. Making each line in the text a different pen, switching to MODE 0 and cycling the palette in this portion of the screen, we are able to reproduce an effect similar to the arcade version.



Spawning red balls

Finally after a lot of work, a red ball has appeared. I needed to check through all my variables to make sure they were referencable through each sprite block data array. I forgot to add the movement counter to the array. Once I had that corrected, I added an NPC function that basically picks a left or right direction at random and treats it as if it was input from the player. I also needed to update the sprite drawing function to grab my sprite ID number so it is updating the correct sprite in the ASIC chip. The result is a fairly fast red ball that descends the pyramid quite quickly until it falls off the edge of the screen. I will need to slow the movement down, and also create a timer so it will spawn a new red ball every couple of seconds. And do this alternately on two separate sprite data blocks, as we will need to have two red balls on screen at once!



Move delay

I tried slowing the balls down as they descend the screen, but the squashed bounce graphic was displaying so fast it was barely visible. It turns out there is a slight delay for each NPC before it executes the next move. So in order to implement a delay, I had to add a variable to the sprite data block. It gets set to 16 or something similar every time the sprite lands on a cube. When the function is called to calculate the movements of the sprite, this variable is decremented. Only when it reaches zero

do we allow the sprite to actually move. It turns out this is also true of the player sprite in the arcade game. This does make the movement a little more sticky.

Swearing

As we now have the red ball working, I was able to implement the comical feature in the instructions screen. A red ball drops from a height and hits Q*bert on the head. A speech bubble appears showing an expletive.

The ball drop was done by creating a modified version of the parabolic jump function. We set the horizontal movement to zero, but still allow vertical movement. When the vertical momentum reaches 18 pixels, we reset the momentum again and add horizontal momentum to make it bounce off poor Q*bert's head.



The speech bubble was created by copying the arcade version pixel by pixel. It took 6 hardware sprites to recreate, but I can use the same code for when he is hit by a ball in the game, as it will automatically appear in the correct position according to where Q*bert is on the screen. It will probably have to be used in place of any disk sprites, as we are limited to 16 hardware sprites on screen at once. I did have to move some of my logo graphics to the memory under the ASIC to make room.

Coily

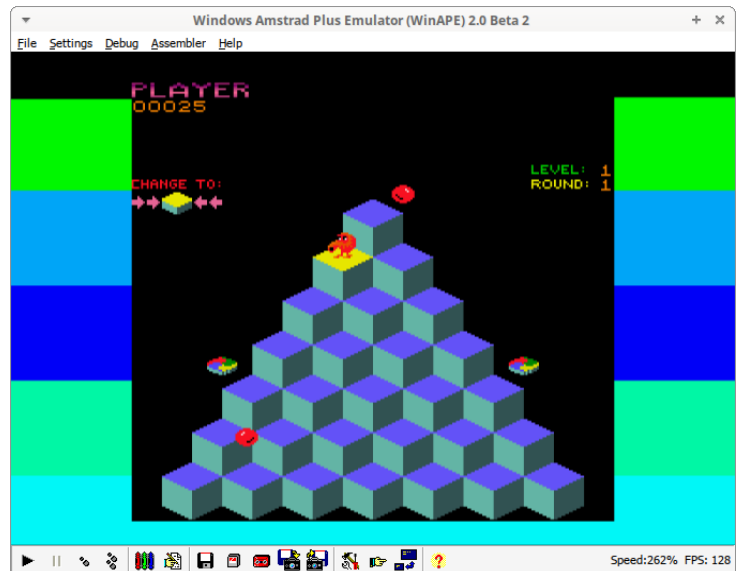
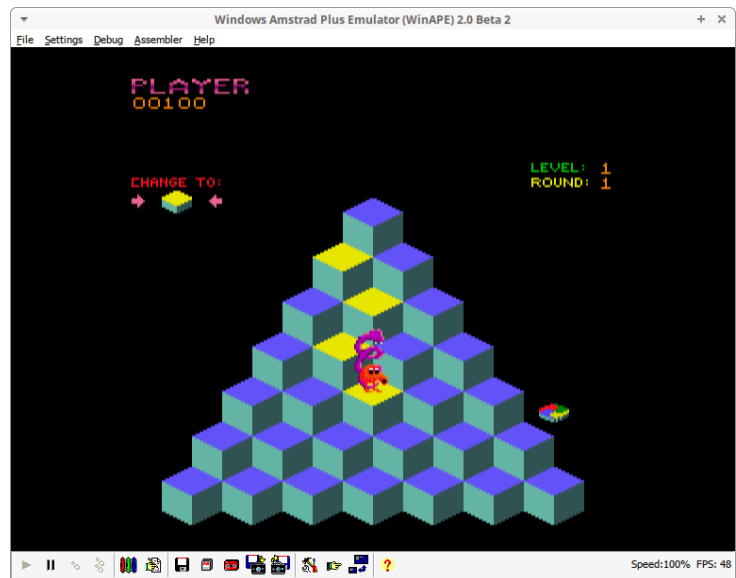
I also drew the Coily egg and set up a sprite ID and NPC for it. Using the NPC id, I am able to specify the particular movement code that is called when Coily moves. A counter stops him jumping off the bottom of the screen, and then a timer elapses before we change him into a proper snake. Changing the sprite and NPC id allows us to change the graphic and movement code used to locate the player.



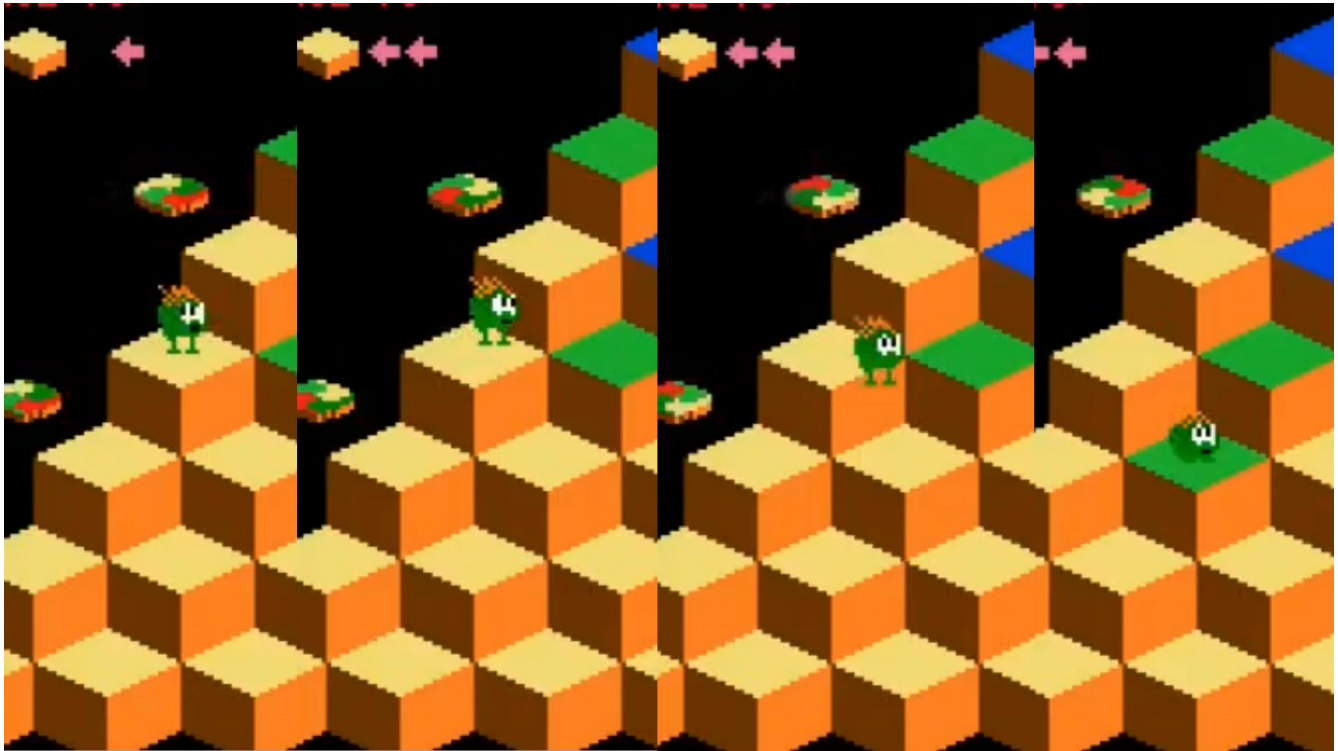
As Coily is 32 pixels high we need to use two sprites to draw him. I copied the sprite pixel by pixel from screenshots of the arcade game. I reserved the last hardware sprite specifically for Coily's head, and allocate the sprites from 14 down to spawn other enemies. I amended the sprite movement and drawing function to check if Coily is the NPC being moved. If so, we render the extra head sprite when he jumps and paste it 16 pixels above the normal enemy sprite. We also need to remember to hide it again when he lands. The parabolic movement function also needed adapted to display Coily's head when he leaps off the pyramid. Finally, I copied the direction finding code from Mimo's Quest. Plugging in the player coordinates and Coily's coordinates in the grid, we can find a direction we need to move Coily in to reach the player. As the grid is rotated at a 45 degree angle, we also need to rotate the results of the direction finding function. Up right on screen is really up in the grid, down right on screen is really right in the grid. Moving diagonally in the grid is not allowed, so we limit the results to just the four directions.

Drop start

I got the drop start working for the enemy sprites. I added an extra variable, which is set when a sprite is spawned. If the drop start variable is set, the sprite position is moved according to a separate 'parabolic' lookup table. The sprite still takes 16 frames to move, but it moves straight down instead of across, slowly building momentum. The same method is used for dropping Q*Bert from the disk. There is a time delay when Q*Bert is on the disk, before he gets dropped. Also, enemies are not permitted to spawn while Q*Bert is on the disk. This has the effect of clearing the screen of baddies.



Slick and Sam



So it turns out there are extra images for Slick and Sam, and Ugg and Wrong Way whenever they are jumping from one cube to another. Who knew? So it looks like we are going to have to check if the sprite we are moving is one of these enemies. If it is, we will need to add an extra line of code to copy the correct image to the hardware sprite based on a specific move counter value.

Slick

So I managed to get Slick drawn. Unfortunately the sprite palette only contains 15 colours, as one pen is used for transparency. As the arcade uses a 16 colour palette, and I had already set pens for the main characters, Q*bert and Coily, I would have to drop a pen somewhere. There is a medium green and dark green pen used for the green sprites, namely Slick, Sam and the green ball. As the dark green pen is the least used in the game, I was forced to draw slick using only medium green. Thankfully the player probably will not notice any difference. I also modified the function so that he unselects a cube when he lands on it, as opposed to Q*bert.

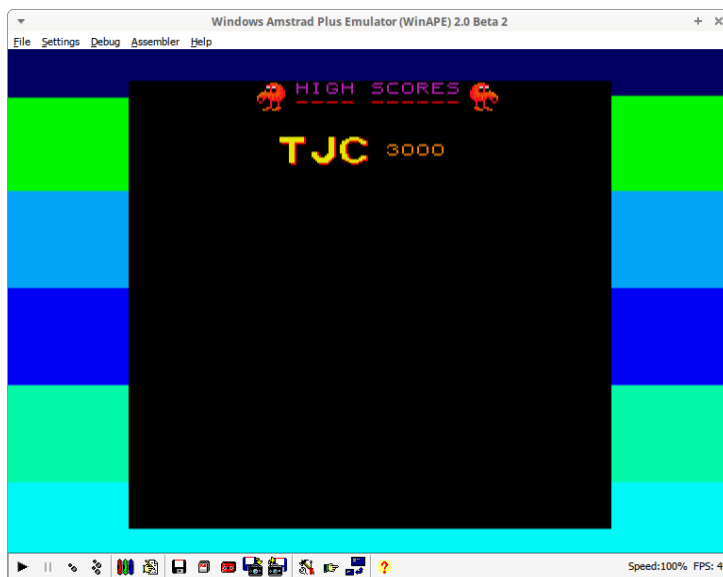


I have also modified the level screen menu movements. It turns out that there are different movements for each level up until level four, after which the same pattern is repeated for each level. So I made a level screen movement table containing the directions to move represented as numbers. Then a function reads the appropriate line based on what level the player has reached, and moves Q*bert accordingly.

Scoreboard

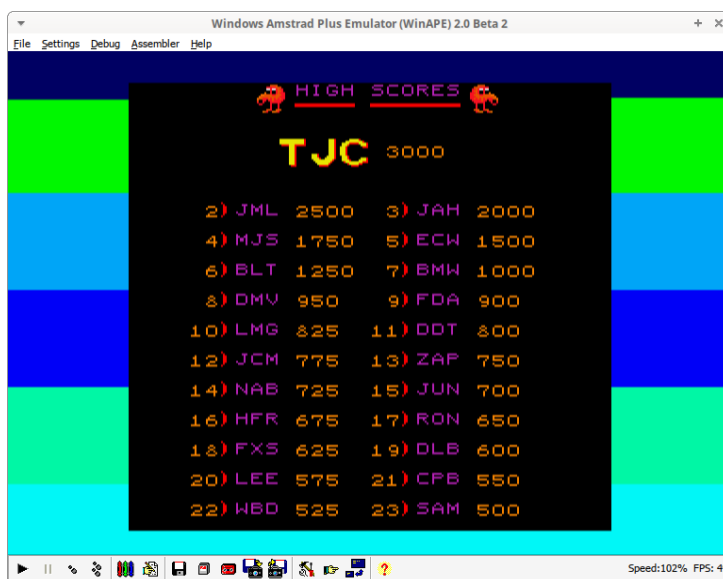
Working on the scoreboard screen. As there are a total of 5 colours on screen including the black background, we need to use an interrupt to draw the extra colours. Thankfully there is an interrupt at the top of the screen, and slightly further down, which enables us to print the high scores text in purple and the player initials in yellow, as in the original game.

I also added the scoreboard data table. There might be a problem with recording the score, as I see in the arcade longplay, it is possible to gain over 65000 points, which makes it difficult to keep track of on an 8bit machine. I'm not sure if the arcade machine allowed for large scores. I may need to develop a function that can increment a 32bit number.



Game attract

So now we have the high score table working properly, I was able to sequence everything together in the 'game attract' mode. The main menu is first displayed, then the instructions, the gameplay is shown and finally the high score table, and the sequence repeats. Getting Q*bert to move by himself wasn't too difficult. I reused the movement code for the level screen. I watched the arcade version to see what moves he makes in the demonstration game and added the directions to a movement table. There seem to be two different sequences, which I will copy across to my game.



Lives

In the arcade version, Q*bert's lives were depicted as mini Q*berts at the left side of the screen. Unfortunately as this area is MODE 0 to accommodate the drawing of the cubes, we can't draw mini Q*bert's without them looking extremely pixelated and spoiling the nice MODE 1 text and sprites. So as a compromise, I have made a 'lives' counter, similar to the level and round text at the right hand side. It should blend in better with our game.

Extra life bonus

I got the extra life bonus working. I created two defines with the first and second score levels for awarding extra lives. I update the instructions page with their values at the start of the game, in case we want to allow the player to change them later on. We store the current level in a variable, and each time we earn points, we deduct the same amount from the variable. When the variable reaches zero, we add an extra life and reset the counter.

Collision detection

I've got very basic collision detection working. Basically it only checks to see if the cube a sprite is moving into is occupied by the player. If so, the speech bubble is displayed, and the game reset. The player sprite needs to stay on the last cube it occupied when this happens, which took a bit of figuring out. If Q*bert is in the middle of a jump, the position needs to be reset, and having so many variables controlling the movement of the sprites, it was a little complicated.

I will need to improve the collision detection so it picks up if a player's location matches another sprite's location in the middle of a jump. I'm not too sure how to do that yet.

I also pasted some of the high score texts into a table from a Q*bert wiki page, so I will need to work on designing this screen for when the player is asked to enter their initials.

Background flash

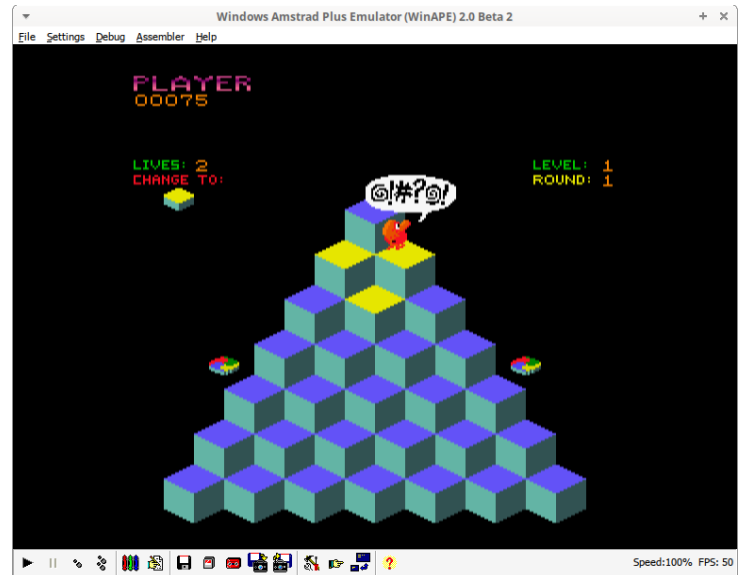
I got the background colour fade working when Q*bert hops on a disk. I had to make an extra function call location in the topmost interrupt. As we are already doing split palettes, the green colour needed to be set in the ASIC for the background, the border and the split palette table. It seems to work quite well.



Your basic collision detection

So I got your basic collision detection mostly working.

It took a bit of figuring out. I first check to see if it is the player sprite moving or an enemy. If it is the player sprite, we load the horizontal coordinates and compare them with all the other sprites on the screen. If it is an enemy sprite, we just compare their coordinates with the player sprite. If there is an exact match, we check to see if the squares they are occupying are the same. If so, we have a collision. I had to make a slight deviation for drop starts, by comparing the vertical coordinates instead.

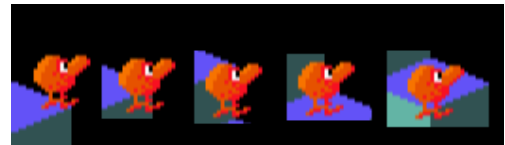


Unfortunately it is still possible to jump past an enemy if you are both in the air at the same time. I'm not sure why this is happening...

Jump mask

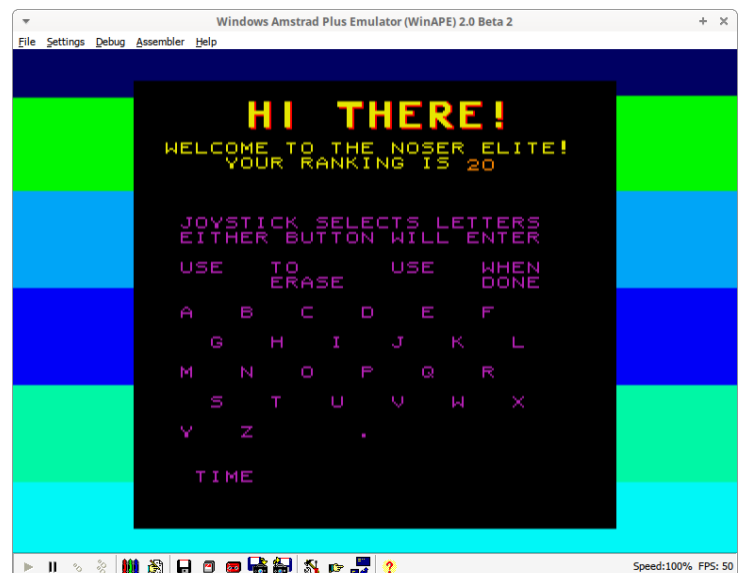
Getting Q*bert and Coily to disappear behind the pyramid as they leap off into oblivion might be difficult to achieve.

Although the process will be the same for each cube on the left and each cube on the right, there doesn't seem to be any easy way of going about erasing the appropriate pixels from the hardware sprite as he leaps off the cube. An alternative might be creating a couple of cubes as hardware sprites and hiding Q*bert behind them, but then that would involve messing about with different sprite hardware palettes for each round and it could get complicated trying to avoid clashing with the other sprites and colours that are in use. I may need to move to cartridge only, and see if I can create the appropriate Q*bert and Coily sprites pre-clipped and insert them into the ROM?



Scoreboard entry

Working on the scoreboard entry screen, I managed to get all the text in the correct places. Unfortunately the noser elite subtitle has to appear in yellow, as it is located in the interrupt along with the big text. But it doesn't look too bad. I will need to add sprites and some sort of function to allow the player to move the green ball around in order to enter their name. A little complicated, but I think it is better going for the authentic arcade look.



I managed to get the green ball bouncing between the letters. I had to create a copy of the cube properties grid, and replace it with letters of the alphabet. Then make sure that when the ball bounces around, it doesn't bounce off the screen...

Basing the name entry function on the cube location grid was super complicated. I had make checks to move the sprite location as well as modify the grid location to make sure we weren't trying to read a value outside of the table when we wanted to read a letter. In the end, I created a new table, with the letters arranged horizontally as shown on the screen. Then instead of using cube location, I use the pixel location of the sprite to decide which letter the pointer is located at. Once we know which letter we are pointed at, we can decide what to do if the player tries to move the pointer off screen. It's a lot simpler and works better. I added a test for the fire button so the player can enter their initials. I will also need to add the red underline sprite, which will appear when the ball lands on a letter.



I managed to add the red underline sprite to the high score entry. Unfortunately memory is now tight in the sprite bank, so we may need to move some code to the other bank if we still want to try to develop a disc and tape edition. I also did a lot of work on the high scores table, creating functions to calculate the player rank based on the score. This necessitated inserting the player score in the table and moving everything below down one line. And also checking in case the player score is the last entry in the list, in which case we do not move the list down. A little complicated, but it seems to be working well. I just need to make the name entry function for the high score table and then this section of the game will be complete.

Green ball powerup

So I managed to get the green ball powerup working. I adjusted the collision detection so that when the player collides with a green enemy, that no harm is done. Then I checked to see what effect colliding with the green ball has. It freezes enemies. It renders them harmless. It adds 100 points. It lasts for a few seconds. And it cycles the background colour. It was fairly simple to add a timer when the player collides with the ball, and then use that timer as a condition to disable collision detection and sprite movement for enemies. I reused the green flash code for the disks, and modified it so it cycles the colour guns instead of using a specified palette change.

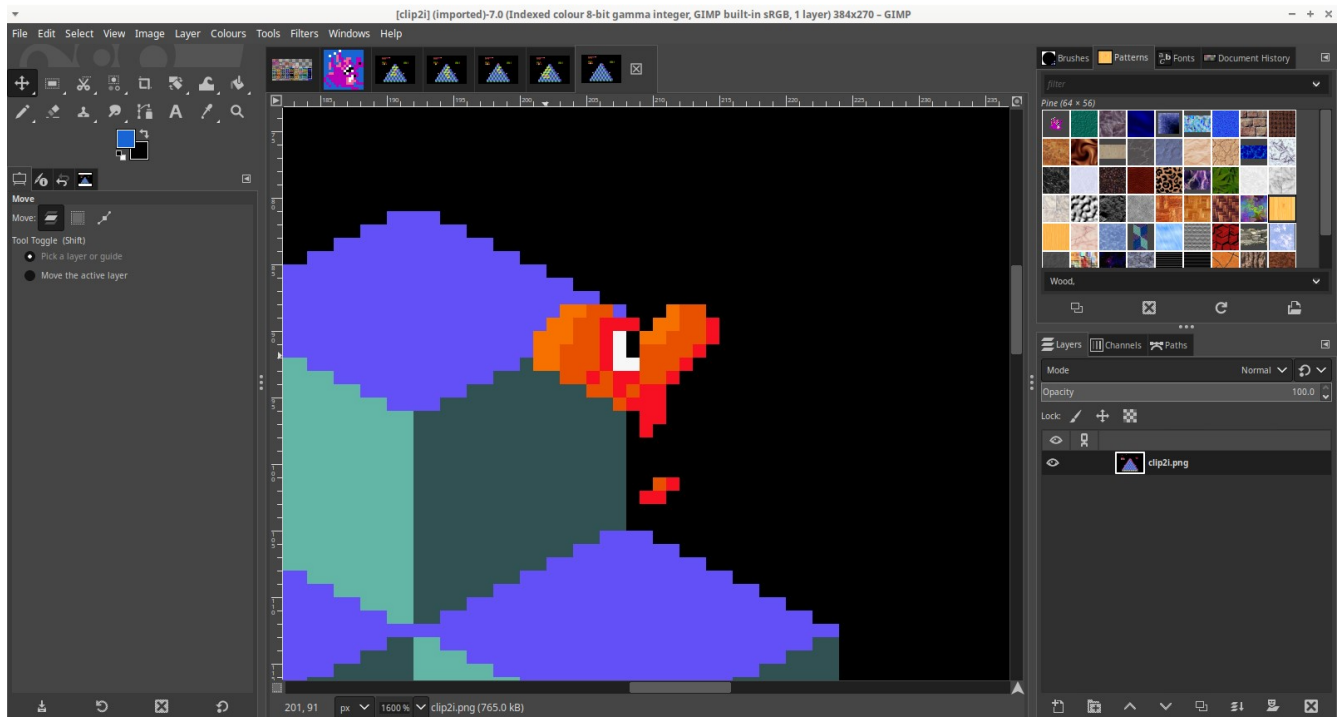


Room for sprites

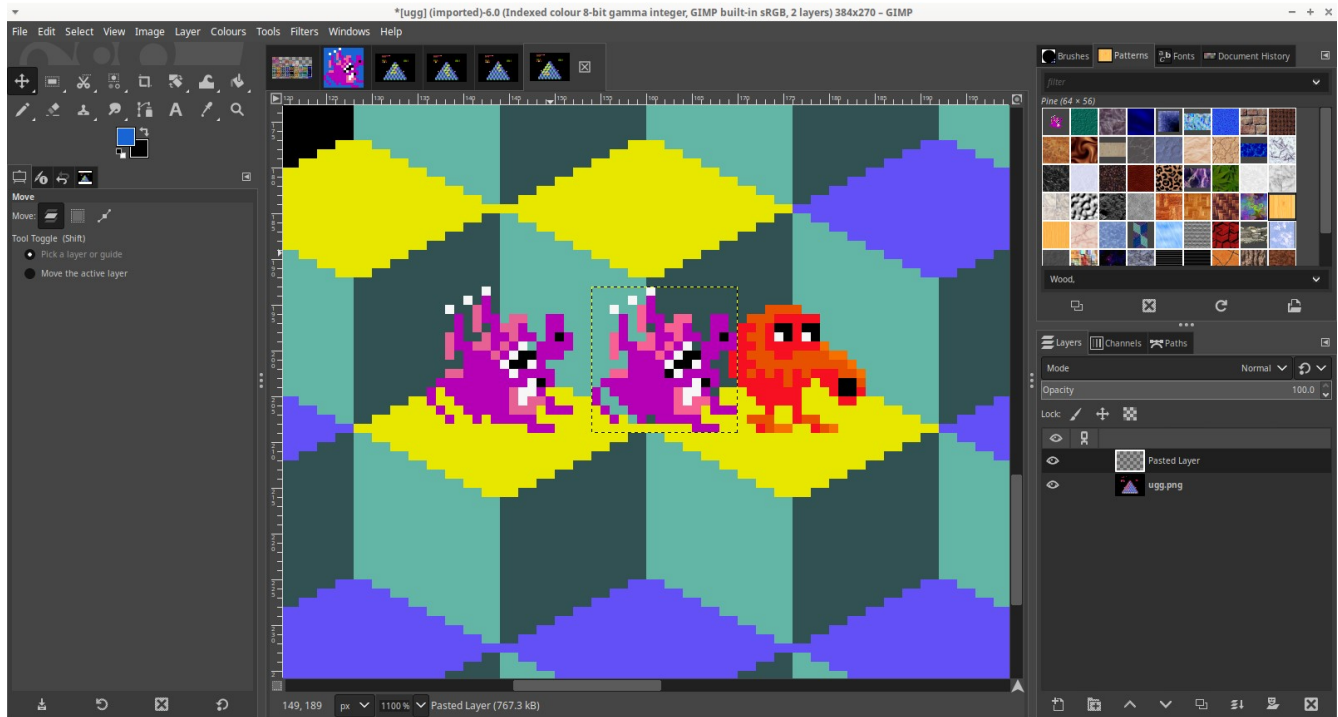
I still have to draw the Ugg and Wrong way enemy sprites. I will need 4 spaces of 256 bytes for each enemy and I am running low on memory. So I have moved the speech bubble sprites and menu sprites under the ASIC. If we want to use them we have to disable the ASIC copy them to a buffer, then enable the ASIC and copy them from the buffer to the ASIC. As timing is not critical this seems to work okay for this purpose. I may also need the extra space for speech synthesis later on.

Parabolic jump and sprite clipping

There was only one way to accomplish sprite clipping when Q*bert jumps off the pyramid. That is to manually paint in and out individual pixels so he appears to pass behind the pyramid. As Q*bert is drawn using a hardware sprite and these are separate from the main screen memory, there is no way of creating a mask to do it automatically. So 5 additional sprite images were needed for Q*bert, and I will need to do the same for Coily. I added a delay for the parabolic jump. It was happening much too fast compared to the arcade. This makes the clipping effect much more noticeable.



I have begun to draw Ugg and Wrong Way too. I will need to add in a movement function with the coordinates reversed so they will bounce across the screen rather than down.



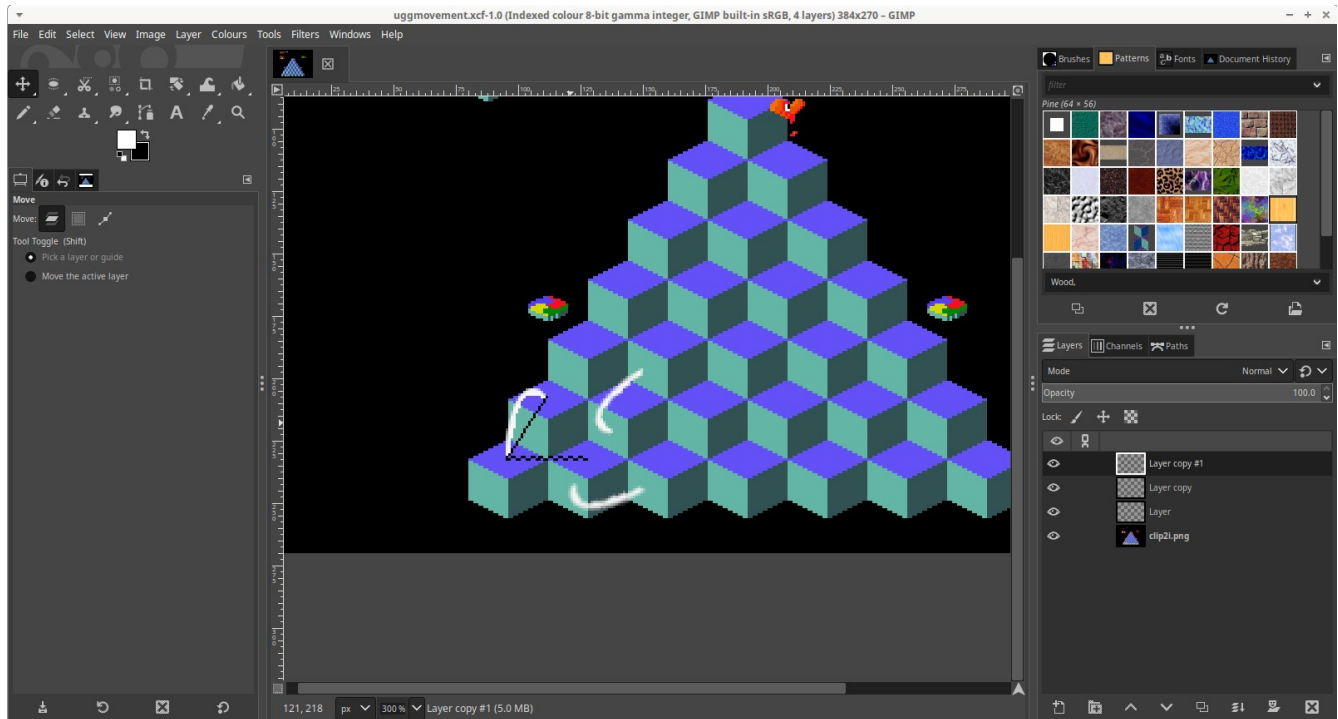
Sprite rotation

I wasn't sure if this was going to work or not. I noticed that as Ugg and WrongWay descend the pyramid from left to right and right to left, their sprites aren't actually flipped vertically as they jump from cube to cube. They are flipped and rotated at 90 degrees. This has to be done due to the perspective of the pyramid. As memory is tight, I wasn't sure if I could fit in four more images of pre-rotated sprites for Ugg and WrongWay. I tried building a function that copied the sprite to the ASIC and rotated it as it copied, by copying in columns instead of rows. Keeping all the sprites aligned to 256 boundaries meant all I have to do is SUB the low byte in L by 16 each time we copy, and when we finish a column, move back to the start and INC the low byte. Then this loop is repeated 15 times to copy the entire sprite. It seems to work well. Now I need to work on the bounce mechanics.



Bounce mechanics

Ugg and WrongWay's sprite movement is difficult, because it's not just that they move across the screen in a different way. Gravity for them is also rotated, so I can't just use the same movement functions as I do for the other sprites. This graphic demonstrates this fact.



The arch on the left is the normal sprite movement. The two arches on the right are WrongWay's movement. The coordinates for landing on the square are the same, but rotated by 120 degrees. I could spend a lot of time creating separate functions, but I hope there is a way I can create a table of movement coordinates and rotate them by 120 degrees so I can reuse the original movement functions without having to do too much additional work.

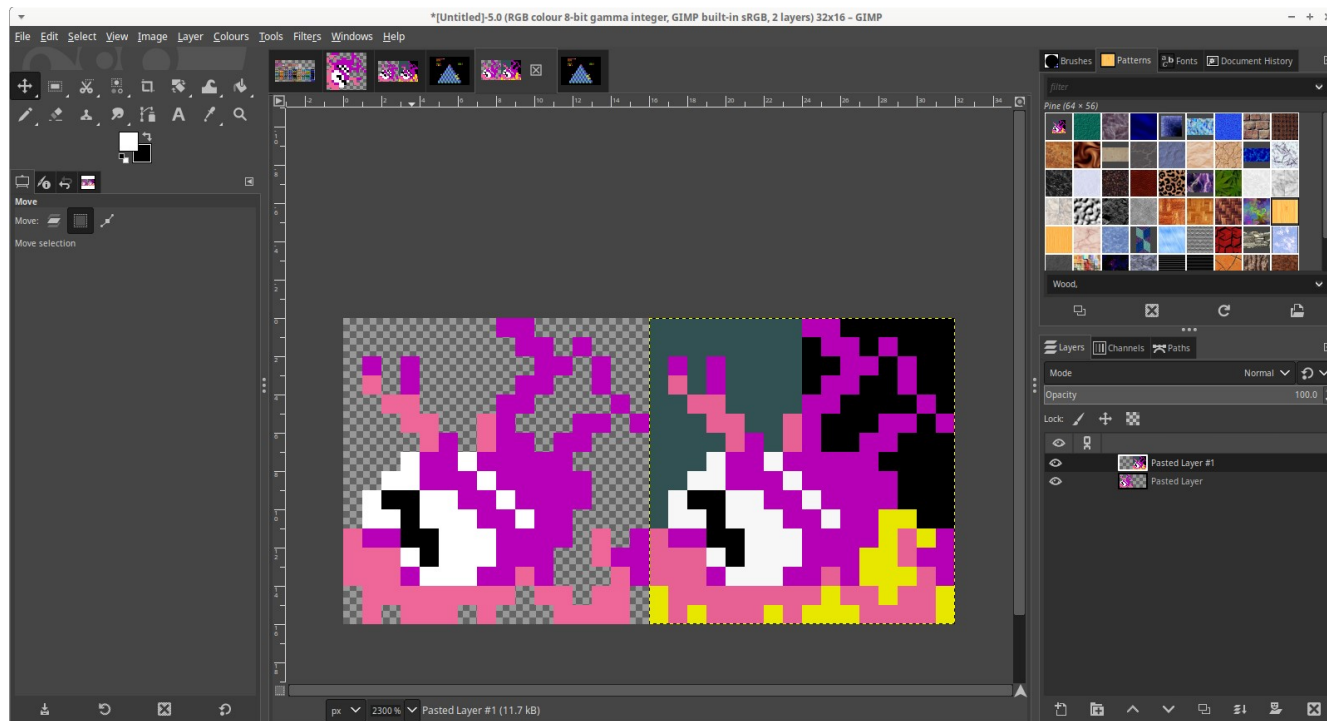
Ugg now working

Ugg is now bouncing across the pyramid correctly. It took a bit of work. As gravity affects Ugg and WrongWay differently than Q*bert and the other enemies, the arc that they jump in has to be slightly modified. I had to change the way all of the sprites are moved. I created four tables for each direction of movement for Q*bert, containing X and Y delta offsets for animation frames 0 to 16. As we are using a 16bit WORD to calculate horizontal sprite positions, I wasn't able to use negative numbers (i.e. single bytes) to adjust the horizontal sprite position. I can't just add



negative 2 to the sprite position when using a 16bit number, as negative 2 is really 253. So instead I set bit 7 to indicate if a value should be regarded as a negative number. The sprite movement function detects this and subtracts the value from the sprite coordinates instead of adding it. This enables me to use the same table to store positive and negative numbers, allowing me to create a bounce effect to the sprite movement. I then created two tables for Ugg's movements, with arcs moving to the left. I also had to add an extra function to allow Ugg to move in a sideways fashion as he jumps from one cube to another. It seems to work well, although I also need to amend the parabolic leap off the pyramid so he falls up left rather than straight down.

WrongWay images



Working on WrongWay images. The same as Ugg, only flipped vertically. Memory is getting tight so I may need to move some code into the RAM behind the ASIC to make room.

WrongWay finally bouncing across the screen

So I managed to get all of WrongWay's sprite images drawn. I had to make some space by compacting the CHANGE TO sprites, using some Run Length Encoding to remove a lot of the empty pixels. I also moved the high score congratulations texts to the memory behind the ASIC. Memory still is tight, and I may need to make more room to get Coily clipping correctly when he jumps off the pyramid. But at least I have all the enemies drawn. I had to duplicate the



movements for Ugg, and then reverse the gravity so he falls towards the top right side of the screen. I also had to amend the cube movements so he navigates the pyramid correctly. I had to create an extra function to rotate his sprite by 90 degrees in the opposite direction to Ugg's sprite rotation. It seems to be working well.

There is a problem with Q*bert's clipping, which was affected when I introduced the new gravity tables. I will need to work on that, and see if there is a way that we can clip both Q*bert and Coily without using too much memory.

All enemies completed

All of the enemies are now working correctly. There was a problem when Ugg and WrongWay were both dying at the same time. One of them would fall off the screen in the wrong direction. I eventually worked out I was setting the gravity on each jump rather than when they are spawned. So now it is possible to have up to four enemies on screen at the one time, bouncing around according to their characteristics.

I also fixed a bug in the scoreboard ordering.

I still need to fix the clipping for Q*bert and see if it is possible to fit Coily's clipping into the 64k of memory. I also need to develop the two player option, work on sound effects and music, and possibly speech. Although the speech may use either an external ROM or if I can disassemble some speech synthesis software, a Plus cartridge.



Score bugs

I added a keypress to allow the player to skip levels. 'R' skips to the next round, just like in Galactic Plague! I remember finding that cheat accidentally on Christmas day, 1987! This will allow me to check the colours, etc again quickly without having to recompile the game.

There was a problem with the bonuses. They weren't cutting off at 5000 after you passed level 5. So I managed to fix that bug. There was also a bug in the scores. As I was using a 16bit WORD to store the score, when this passed 65535 in the later levels, it wrapped back around to 0. This is a major problem. I couldn't figure out how to fix it. I tried extending the score variable with another WORD, and using my conversion function to print the two WORDS to the screen. But it still wasn't carrying when it wrapped. And the conversion function wouldn't have worked on both WORDS anyway, as I later discovered. It would have just displayed something like 0000065535. So I needed a different approach.

There are a couple of alternatives. One is to keep the score in ASCII string format and build a function to manually change the figures when we add to the score. This would have been very complicated. Another alternative is to divide the score by 5, since it goes up in increments of 5, and expand it every time we print it to the screen. This would have been very time consuming, and there is still the danger

of the score wrapping. I did some research and found that there is a method programmers used to keep large scores in memory. It is called Binary Coded Decimal.

Normally if we add two bytes together, &18 (24 decimal) and &64 (100 decimal), the result will be &7C (124 decimal). But if we use the Z80 command DAA immediately after we perform this function, the processor will treat the numbers as if they are decimals, changing the result of the sum to 18+64, which equals 82. The digits are stored in the left nybble and right nybble of the byte. Letters A to F aren't used. So adding to the score this way, and using the carry function, we can build a score with as many digits as we like.

I will need to modify the scoreboard routines to cope with this change.

Sound effects

So I managed to get the scoreboard and ranking all converted to Binary Coded Decimals. So it is working really well now, and there is no danger of the player getting too high a score. The scoreboard should cope, although the layout of the scoreboard in the arcade version was really only designed for small scores. So if the player does get a couple of large scores the text may get corrupted. But I think I will try to keep the design close to the arcade version.

I happened to notice that there are different sound effects for the bounce for each enemy, depending on their size. The next goal will be to analyse the pitch of each sound effect in Audacity and see if I can replicate it using my system. I will also need to add a function for a sound effect to play through any free channel so we can have more than one sound effect playing at once.

Bonus lives working again

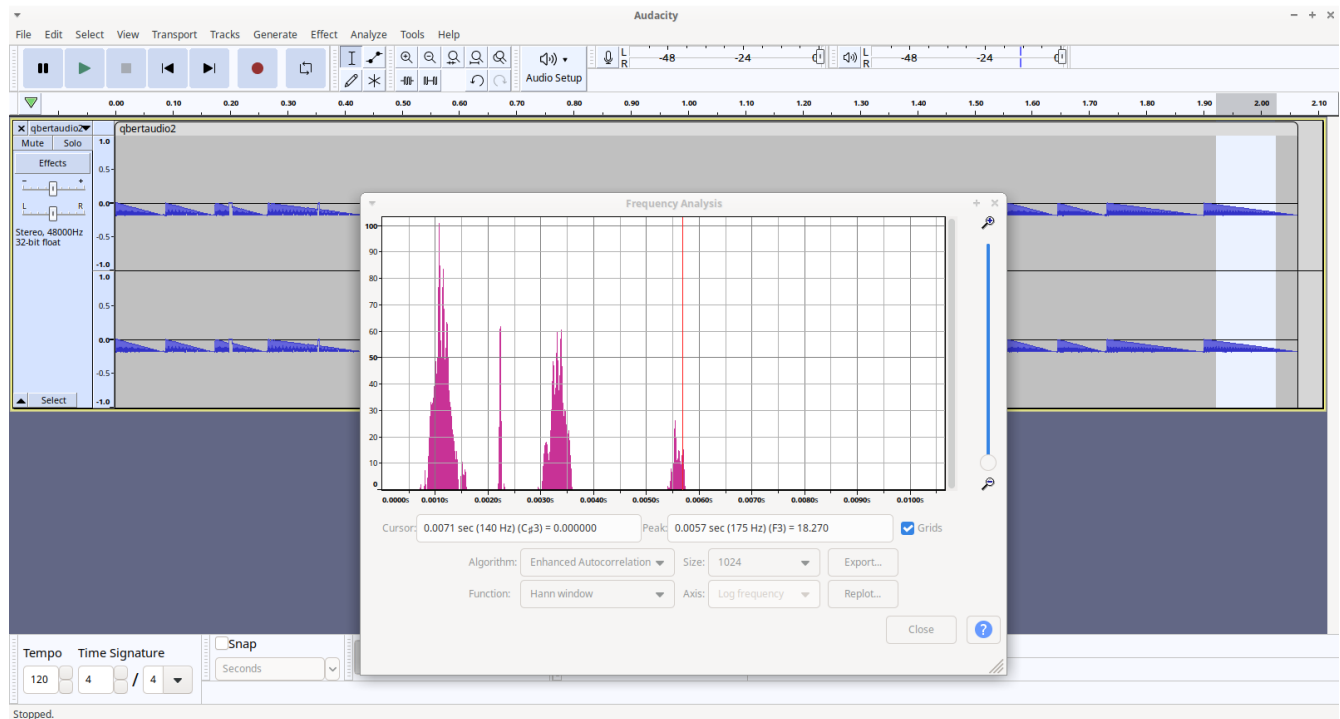
I forgot to update the bonus lives score checking function to work with the Binary Coded Decimal score. It is too complicated and time consuming to check a BCD score against another BCD number every time we jump on a cube or receive bonus points. So what we do instead is we keep the bonus milestone figure in ordinary decimal format in a WORD variable. Each time we get points we add the BCD equivalent to our score, but we also deduct the same decimal amount from the milestone variable. We have to do this every time we add points. Then once we reach zero, we award an extra life and reset the milestone to the higher amount.

Game over message

I added the game over message and enabled the flashing routine for it by utilizing the same function code as the flashing scores on the scoreboard. Seems to work well. Next goal will be to analyse the new level music and see if I can recreate the notes in my sound effect system.

Sound effects and jingles

I worked out the notes of the level complete tune using Audacity's frequency analysis tool. Then adding them to a score and enabling the music interrupt function when we complete a level, I am able to play the jingle while the cubes flash. I amended my music playing function so the main program is able to know when the music finishes.



I was able to use the same technique to add sound effects to the disk movement. A simple scale is crafted using my music playing function, and a suitable instrument is used that has a volume fade. This is played when Q*bert leaps on the disk and stops when he reaches the top. It sounds similar to the arcade version so I am happy with this.

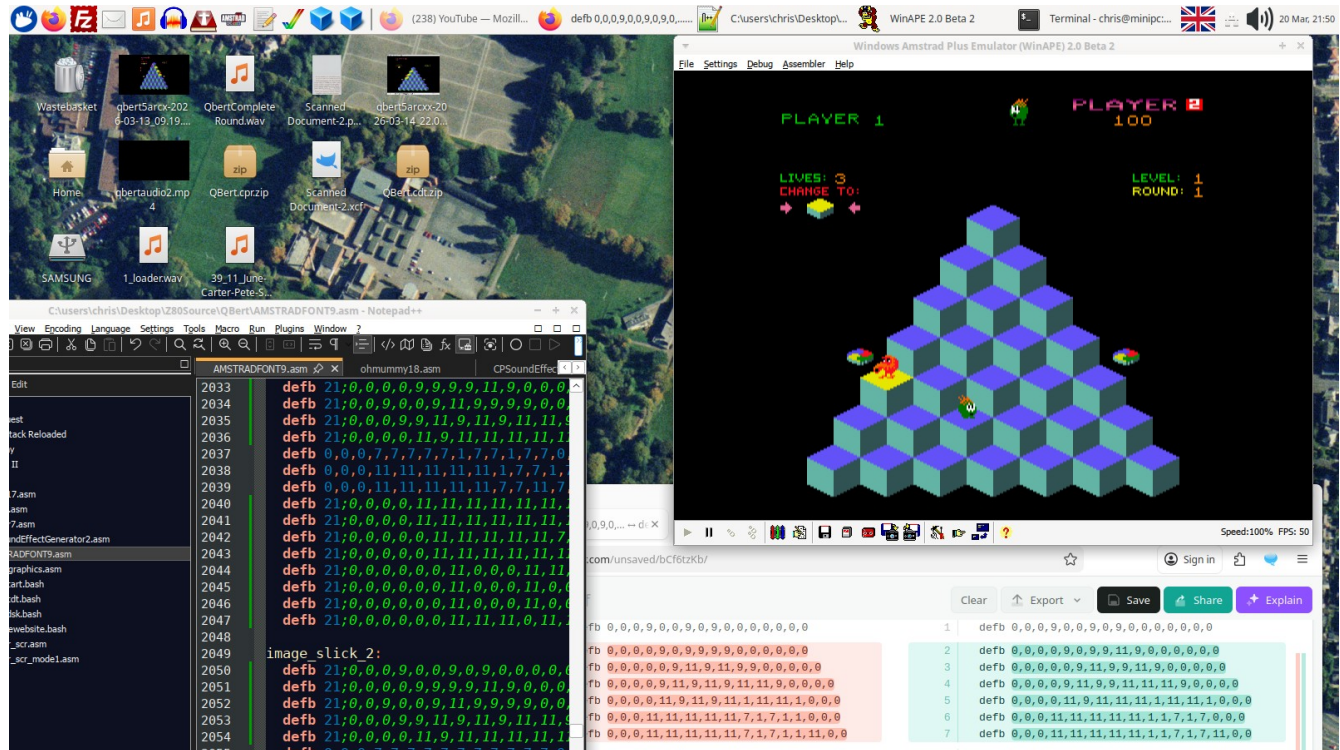
Colours corrected

I had used the screenshot facility on a longplay video of Q*bert to try to grab the colours for each level. This produced slightly inaccurate results. So I happened to find the DIP switches in the arcade ROM that activates the cheat mode, so I was able to skip through the levels and snoop on the palette to record the exact colours of the screen. So hopefully all the round colours will be identical to the arcade version now!

I also happened to notice there are sound effects when the player moves the cursor around the scoreboard name entry. So I will need to work on that.

Sprite compression

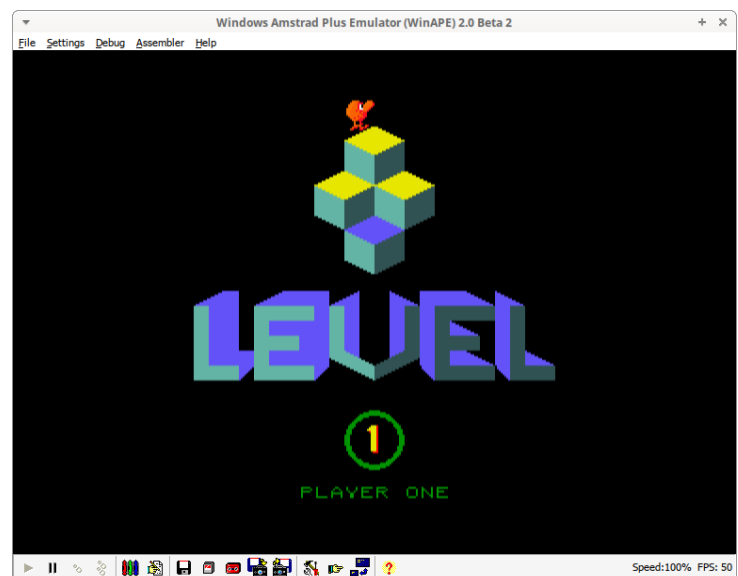
I've been working on implementing the two player option, unfortunately memory is getting tight. I thought about the possibility of expanding the cartridge version, but I wanted to keep to the 64k if possible. So I had to implement some sprite compression to save space.



As some of the animated sprites of Slick and Sam are similar, I decided to create a sprite function that only copies across the changes to the ASIC. As each byte of the pixel in the sprite definition goes up to 15, we are able to use higher numbers as control bytes to tell our sprite function what to do. I added the number 20 as a control byte. This tells our function to insert a row of 16 empty pixels. The number 21 instructs the sprite function to leave a pixel row unmodified in the ASIC. This allows us to replace an entire row of 16 bytes with one byte. The sprite function examines each row in turn. If the first byte is a number below 20, then it treats it as a normal row of pixels. Using a website to spot the differences between two files, we are able to see which rows can be safely commented out using these control bytes.

Two player mode

I've been working on the two player mode. This has been quite tedious to debug. I changed the player text and score positions at the top of the screen depending on which player is playing, as well as adding a flashing text at the start of the round just so there is no confusion.



We also need to keep copies of the level data for each player. When a player dies, a record is kept of their score, bonuses, the cubes that have been selected, the disk positions, the level and round they have reached, etc. Memory was tight so I had to pack it into the start of memory. Gameplay is then switched across to the other player, and if they have already had a turn, their level stats are restored. If they have no lives left, we need to move to the scoreboard and enter their name if they have reached a high score. A bit of memory is taken up just doing checks who is playing. I guess there are probably better ways to do this, but this seemed the quickest as I had not designed the game with two player in mind.

I will also need to add in code to flash the two scores on the scoreboard when the last player dies.

Coily clipping

I managed to compress some of the sprites down using my control bytes, so there may be enough space to clip Coily, though I am not sure.

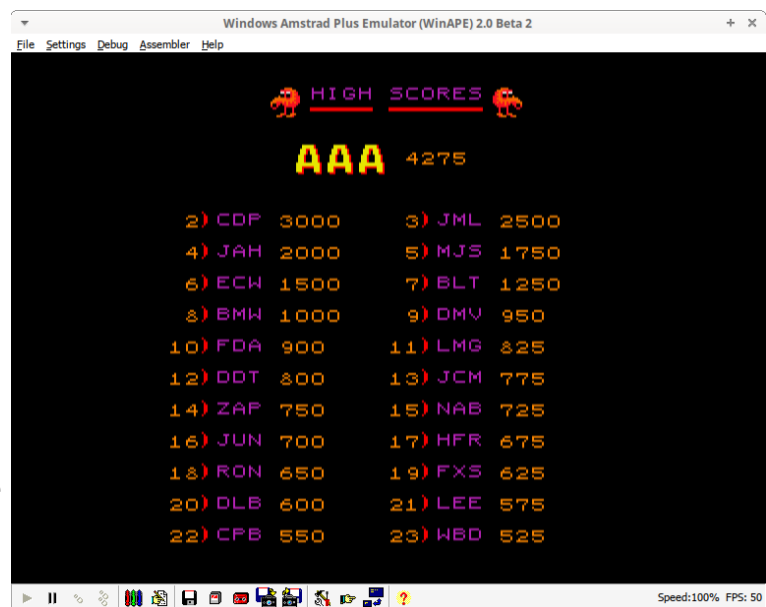
Memory saving

Good news. I managed to save about half a kilobyte of memory. My routine for flashing the text on the screen was over the top. Because I was using interrupts to display them, I was having to write and erase the messages as fast as I could. This was necessary to stop the interrupts from sliding into one another, causing corruption to the screen mode and palette splitting. This also used a lot of memory, as the only way to print the message to the screen fast enough for the interrupts to handle was to grab the text into a buffer and write it byte by byte to the screen.

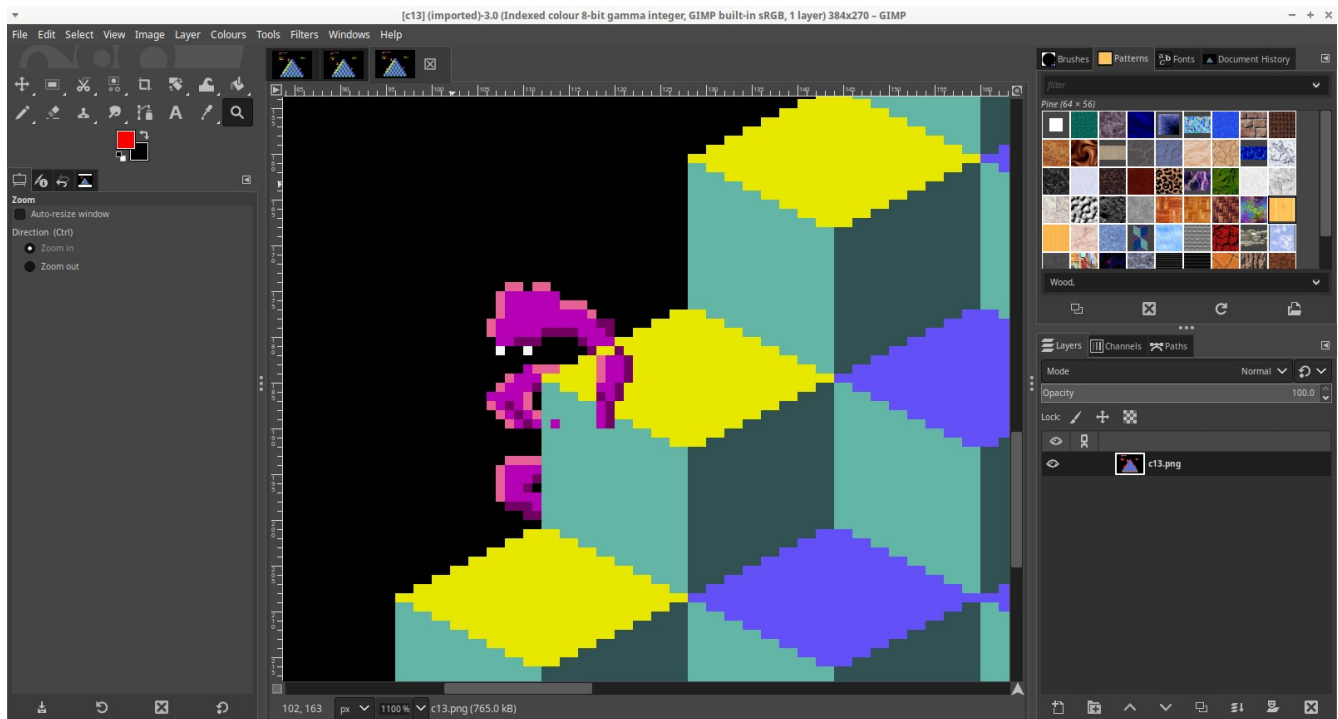
I have now removed the unnecessary memory buffers and placed the message flashing function in the main program loops instead of the interrupts. This means I can write the message and clear it again using my existing text printing functions. Timing is not critical for flashing the messages on screen. So I added a couple of decrementing counters in the loop to slow it down, as well as a boolean variable for toggling the message on or off. And voila! More memory for Coily sprites and sound!

Two player completed

The two player mode has been completed, and the high score entry checked to make sure it can flash both scores at the same time. Rather than creating a whole new string printing routine to paste the score in the right place, I just reused the existing score printing function. I created a table of score locations based on rank. We simply look up the screen position and alternately print the score to the screen or else a blank string. Slight adjustments to the position are used for the top score. Next task is to work on the Coily clipping!



Coily clipped



Coily's sprite has finally been clipped properly for jumping off the pyramid. Thanks to the sprite compression techniques mentioned above, I was able to fit the extra sprite images into 64k with no problems. There were 6 frames that needed updating, and some included both the head and tail sprites. But it seems to be working well.