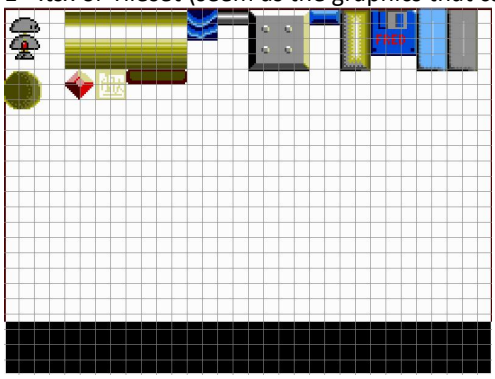
Turning a Tiled .tmx and .tsx file into a binary room map

Tiled uses a few different files to define a map:

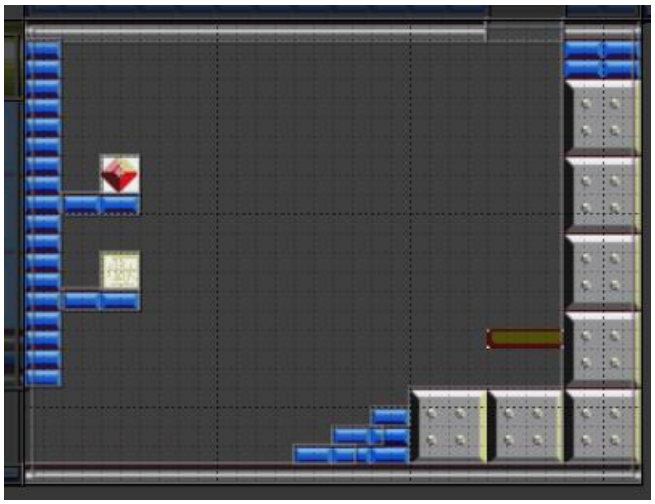
1 - .tsx or Tileset (seem as the graphics that can be used) I.e.usually shown in the bottom right.



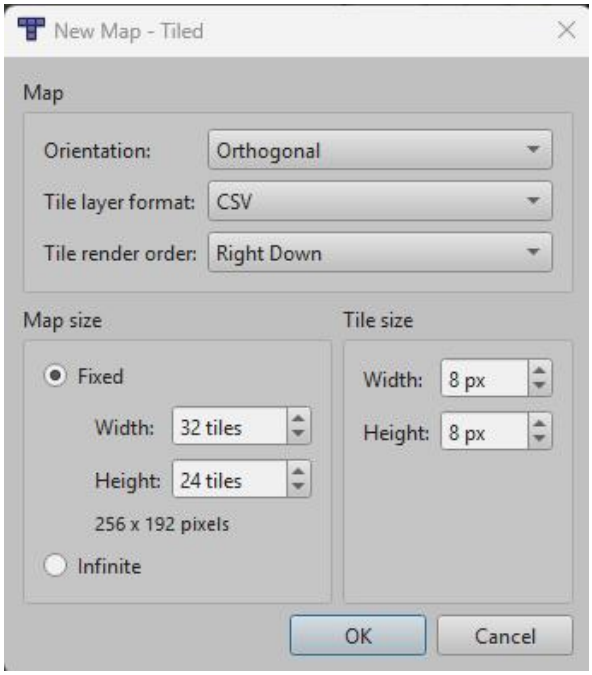
This was made by opening a .png file with the graphics to be used .
As I wanted to make rooms with a resolution of 256 x 192, I set the graphics to align to a grid of 8x8 pixels.
There also isn't a need to keep this file to 256x192, of can be a lot larger to fit all the graphics needed onto it if wanted.

In Tiled you select the squares you want from the Tileset, and then just 'stamp' them into the locations you want them to be in a Mapfile.

2 - .tmx or Mapfile
This is the file that stores the locations of the Tiles from the Tileset above.



To align the room with the tiles on the Tileset, I defined each room as the following when setting up:



This set the room up as a 8x8 grid, with overall dimensions of 256 x 192.
In my setup each 'room' is a single screen. I see others make one big mapfile of all rooms in one big map instead. I just felt it was easier to see exactly where the screen boundries are this way.
To join a number of separate rooms together to view on the same map, they are added together in a World. (.world file which basically lists the .tmx files and their locations).

The file format of Tiled files

The Tiled files are a type of .xml format.

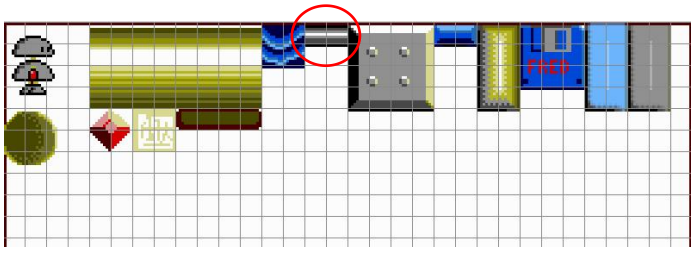
E.g. the mapfile looks lik this when setup:

```
<?xml version="1.0" encoding="UTF-8"?>
<map version="1.10" tiledversion="1.11.2" orientation="orthogonal" renderorder="right-down" width="32" height="24" tilewidth="8" tileheight="8" infinite="0" nextlayerid="3" nextobjectid="21">
 <tileset firstgid="1" source="sprites.tsx"/>
 <tileset firstgid="769" name="sprites" tilewidth="32" tileheight="32" tilecount="0" columns="0"/>
 <layer id="1" name="Tile Layer 1" width="32" height="24" offsetx="0" offsety="-0.29809">
  <data encoding="csv">
23,24,21,22,21,22,21,22,21,22,21,22,21,22,21,22,21,22,21,22,21,22,21,22,21,22,21,22,17,18,
19,20,
55,56,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,49,50,51,52,
87,88,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,81,82,83,84,
119,120,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,113,114,115,116,
23,24,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,25,26,27,0,17,18,19,20,
55,56,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,57,58,59,0,49,50,51,52,
87,88,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,89,90,91,0,81,82,83,84,
119,120,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,15,16,15,16,113,114,115,116,
23,24,0,0,133,134,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
55,56,0,0,165,166,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
87,88,15,16,15,16,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
```

119,120,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
23,24,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
55,56,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
87,88,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
119,120,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,21,22,21,22,21,22,21,22,21,22,
23,24,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,21,22,21,22,13,14,13,14,17,18,19,20,
55,56,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,21,22,21,22,21,22,45,46,45,46,49,50,51,52,
87,88,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,21,22,21,22,21,22,0,13,14,13,14,81,82,83,84,
119,120,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,21,22,21,22,21,22,21,22,45,46,45,46,113,114,115,116,
5,6,7,8,9,10,11,12,5,6,7,8,9,10,11,12,5,6,7,8,9,10,11,12,5,6,7,8,9,10,11,12,
37,38,39,40,41,42,43,44,37,38,39,40,41,42,43,44,37,38,39,40,41,42,43,44,37,38,39,40,41,42,
43,44,
69,70,71,72,73,74,75,76,69,70,71,72,73,74,75,76,69,70,71,72,73,74,75,76,69,70,71,72,73,74,
75,76,
101,102,103,104,105,106,107,108,101,102,103,104,105,106,107,108,101,102,103,104,105,1
06,107,108,101,102,103,104,105,106,107,108
</data>
</layer>

Basically a few setup definitions like grid size,width and height, which tileset file to use, followed by the room data in .csv format.

The numbers represent the grid reference of the Tileset selected, e.g. 23,24 is where the circled 8x8 blocks are:



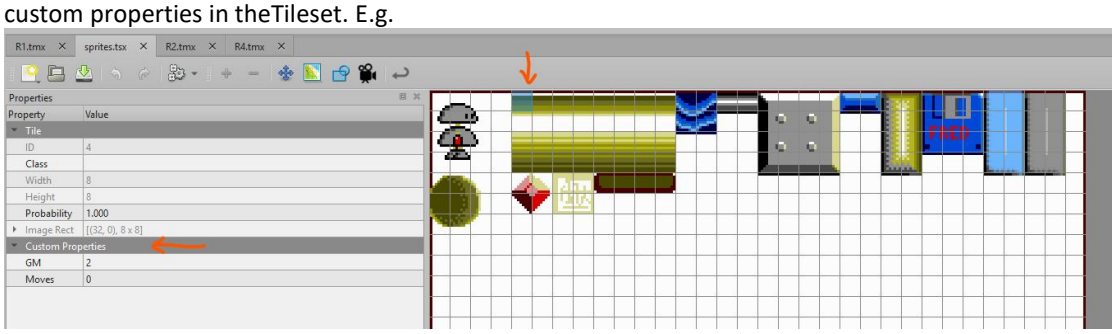
When I saw this format, I realised it should be possible to have some sort of lookup table to replace these grid numbers with a Gamesmaster Sprite number instead, and then if looping through the 32 columns and 24 rows, generate a whole room for use within Gamesmaster.

Defining the Gamesmaster Sprite definitions into the Tilemap.

Transferrring all the actual graphics seemed a much bigger task, so I was just concentrating on generating a mapfile that referenced Sprites already setup in Gamesmaster.

I already had the following setup in Gamesmaster:

I also realised that I could setup the references in the Tileset to the correct sprites numbers by using custom properties in theTileset. E.g.



Selecting the top left of each graphic, I added a Custon Property called 'GM' and one called 'Moves'. The number in 'GM' links to the actual Sprite number in Gamesmaster (2=pipe), and 'Moves' relates to how to place sprite when using Gamesmaster. If set to 1, the Sprite will be placed as a moving sprite (PLACE), but if set to 0, it will be placed as a background object that doesn't move (BACK). (only the top left is needed as the actual graphics are setup in Gamesmaster, so we just want to indicate the position to put the sprite).

Now when looking in the Tileset (.tsx) file, it shows all the GM sprites linked to the tileset grid number like this:

```
<?xml version="1.0" encoding="UTF-8"?>
<tileset version="1.10" tiledversion="1.11.2" name="sprites" tilewidth="8" tileheight="8"
tilecount="768" columns="32">
 <image source="../../sprites.png" width="256" height="192"/>
 <tile id="0">
  <properties>
   <property name="GM" type="int" value="1"/>
   <property name="Moves" type="int" value="1"/>
  </properties>
 </tile>
 <tile id="4">
  <properties>
   <property name="GM" type="int" value="2"/>
   <property name="Moves" type="int" value="0"/>
  </properties>
 </tile>
```

```
<tile id="12">
 <properties>
  <property name="GM" type="int" value="3"/>
  <property name="Moves" type="int" value="0"/>
 </properties>
</tile>
```

So now I'd just have to iterate through the Tilemap file .csv data, and substitute these values with the GM value.
As Gamesmaster only allows for upto 96 sprites, whenever the 'Moves' property is set to 1, I added 128 to the GM sprite number to allow a compact way of defining in Gamesmaster if a PLACE or BACK command is to be used.

Converting the .tmx and .tsx files into a compact binary file for use on the Sam.

All that I really wanted for each room was a file with 32 x 24 sequential numbers which corresponded to the sprite numbers that are to be placed.
For example,

6	0	6	0	6	0	6	0	6	0	6	0	6	0	6	0	6	0	6	0	6	0	6	0	6	0	6	0	6	0	6	0
16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	17	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	17	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	17	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	17	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	17	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	137	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	7	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	7	0
6	0	6	0	6	0	6	0	6	0	6	0	6	0	6	0	6	0	6	0	6	0	6	0	6	0	6	0	6	0	6	0

Fortunately there are already python libraries setup to pares the Tiled files, which helped greatly.
The one I used was 'pytiles_parser (https://pypi.org/project/pytiled-parser/).

My python script then basically reads in the tileset file and generates a look up table to replace the grid references in the tilemap file with Gamesmaster sprites numbers.

E.g.

```
ts_file = Path("sprites.tsx")
my_ts = pytiled_parser.parse_tileset(ts_file)
no_tiles=my_ts.tile_count
LUT=dict()
LUTline=dict()

for n in range (0,no_tiles):
  if n in my_ts.tiles:
    gm = dict.get(my_ts.tiles[n].properties, 'GM')
```

```
    moves = dict.get(my_ts.tiles[n].properties, 'Moves')
    LUTline=({'GM':gm,'Moves':moves})
    LUT[n]=LUTline
```

The script then uses this Look Up Table with the Tilemap file, and generates an array of the required Sprite numbers in sequence.

```
my_map = pytiled_parser.parse_map(map_file)
    reader=(my_map.layers[0].data)
    for row in reader:
      accm=0
      o_row=[]
      no_map= {'GM':0,'Moves':0}

      for tl in range(0,len(row)):
        tiled=row[tl]-1 #tiled adds 1 to reference shown in Tileset and csv output in TMX files
        vals=LUT.get((tiled),no_map)
        gm=vals['GM']
        mo=vals['Moves']
        if mo==1:gm=int(gm)+128
        o_row.append(int(gm))
```

In order to compile all the room files, the script will look for every .tmx file in the same directory as the script, open them in order, and append the data for each room.

I added a batch file to call the python script as a custom command in Tiled. This way, its easy to just press a button to rerun the script everytime updates to the rooms are made.

Current limitations of the script :

- **it doesn't read whch Tileset file to load as defined in the tilemap file, this is currently hardcoded to open 'SPRITES.TSX', - this is on the list to add this functionality**
- **It doesn't currently allow for more than one .tsx file, which can be added when implementing the above update too.**
- **It doesn't use the .world file at all. It should be possible to link filesnames and locations, but currently it just assumes rooms are sequentially numbered. Code has to be use to correctly grab the right room data, for example when exiting up/down compared to left/right from a room.**

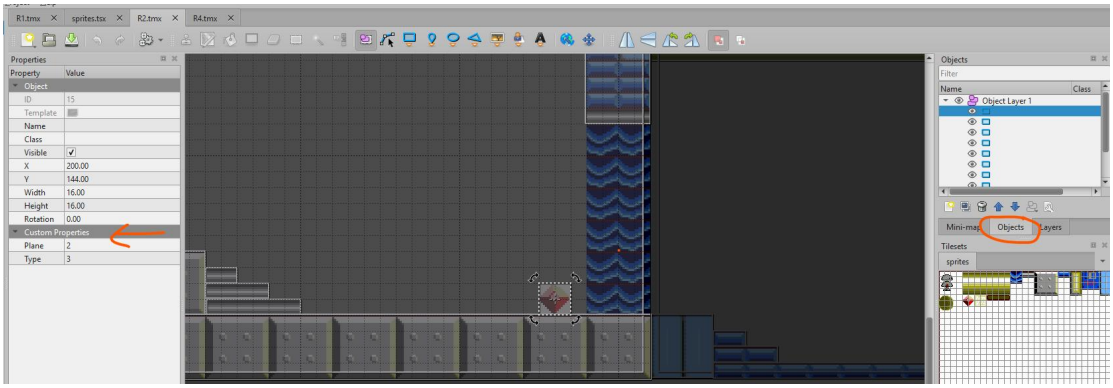
I also realised that in some rooms, a lot of the data is just 0's, so in order to speed up the process of generating the room later the code will place a high value instead of the first 0 when it finds consecutive 0's. This is on the basis that sprite numbers can only be 0-95, and if non-moving, then upto 95+128=223.
Any number above 223 can therefore be uses to signify line-skip or jump aheads.
For example if the code finds 16 consecutive 0's, it will place the value of (255-16)=239 at the start of the consecutive 0's. If an entire line is 0's, the first 0 will be replaced with a 255.
The code within Gamesmaster can then be set up to skip ahead in reading data when it finds these high values.

I also added some simple header values to the output file. It starts with 'R', then the number of rooms, no. of rows and no. of columns in the grid data. This is to allow some level of basic data checking. (e.g. my Gamesmaster code will report an error if it is asked to get room data and doesn't find the 'R' in the right memory location).

This resultant array was then written in a binary format into a single compact file, and transferred to the Sam. Corresponding code on the Sam was needed to then correctly use this file (see later sections).

Using Tiled objects to define Gamesmaster BLOCK data.

Gamesmaster uses blocks (drawn squares) to define areas in the game for collision detection, defining platforms etc.
It is a lot quicker to draw sprites that don't move(e.g. solid platforms) as background objects with a block definition around them than defining everything as a moving sprite with edge detection.
While Gamesmaster also allows you to fill blocks with sprite graphics, I found doing this can lead to flickering on screens where you try put a block over other backgrounds, so I have avoiding the use of BLOCKFILL.
In order to be able to define rooms for Gamesmaster, a way to set up blocks in conjunction with the sprite placement was needed in Tiled.
Thankfully Tiled already allows the use of Object layers, and allows you to draw rectangles.



Again in order to make work well with Gamesmaster I added a couple of custom properties.
'Plane' and 'Type' line up with the BLOCK definitions, for example I set Type 1 blocks to be platforms/walls, Type 2 blocks to be screen edge detection, and Type 3 blocks to be items that can be removed.
Although not fully documented, for Plane the number can be upto 63 (6 planes), +64 to make the block 'enclosing', **and +128 to make it 'supporting'.**
I havn't set up a custom property for belt speed, currently this can be done by using a BTYPE command within Gamesmaster on one of the Types.
Objects are added into the Tilemap file in the following format:

```
 </layer>
 <objectgroup id="2" name="Object Layer 1" offsetx="-1" offsety="-0.666667">
  <object id="2" x="16.4239" y="0.33333" width="208.136" height="8.33333"/>
  <object id="3" x="0.848485" y="160.182" width="254.818" height="30.8182"/>
  <object id="4" x="192.227" y="56.2125" width="32" height="5.3333"/>
  <object id="5" x="176.083" y="119.917" width="80.8258" height="8.93182">
   <properties>
    <property name="Plane" type="int" value="1"/>
    <property name="Type" type="int" value="1"/>
   </properties>
  </object>
  <object id="6" x="16.8333" y="80.1667" width="31.3636" height="7.90909"/>
  <object id="7" x="160.182" y="128.287" width="16.1818" height="7.59091"/>
  <object id="8" x="144.182" y="136.151" width="16.0909" height="8.27273"/>
  <object id="9" x="136.303" y="144.303" width="7.81818" height="7.63636"/>
```

```
 <object id="10" x="128.151" y="152.333" width="9.87879" height="7.54545"/>
 <object id="11" x="224.75" y="0.5" width="30.75" height="63"/>
 <object id="12" x="89.1667" y="24" width="43.3333" height="12">
```

Parsing of Object layers are also also supported in pytiled_parser so again, the required data was extracted and appended to the binary room data: (block number, x,y,width,height, plane,type)

```
blocklayer=len(my_map.layers)
    if(blocklayer>1):
      noblocks=0
      for n in range (0,len(my_map.layers[1].tiled_objects)):
        if not(hasattr(my_map.layers[1].tiled_objects[n],'text')): #for future implementation of text
maybe
          noblocks=noblocks+1
          p=1
          t=1
          pt = dict.get(my_map.layers[1].tiled_objects[n].properties, 'Plane')
          if (pt != None):#check for custom property 'Plane'- leave as 1 if not found
            p=pt
          tt = dict.get(my_map.layers[1].tiled_objects[n].properties, 'Type')
          if (tt != None):#check for custom property 'Plane' - leave as 1 if not found
            t=tt
          x=(my_map.layers[1].tiled_objects[n].coordinates.x)/2
          y=(my_map.layers[1].tiled_objects[n].coordinates.y)
          w=(my_map.layers[1].tiled_objects[n].size.width)/2
          h=(my_map.layers[1].tiled_objects[n].size.height)
          if(x+w)>255:
            w=255-x
          if(y+h)>191:
            h=191-y
          bf_outt.append([n,x,y,w,h,p,t])
      print("Blocks needed in room:", noblocks)
    print(roomf," complete")
    bf_out.append(bf_outt)
```

One thing that made this BLOCK data a bit more complex was I allowed there to be variable number of Blocks in each room, meaning the output file wasn't a nice regular rectangle of data like the room file, so needed a table up front of the room number verses how many numbers to skip ahead to get the Block data.

```
# LUT of two values - which is cumulative no of blocks (v1+v2) to allow upto 510 blocks to be defined
e.g.
#   4,0
#   8,0
#   ..
#   250,0
#   255,4
#   255,50   - means data for this room can be found 255+50=305 x 7 bytes later
```

To support this within Gamesmaster came with one issue - within the Gamesmaster code there isn't a way to define new Blocks, only to change existing already defined blocks. I.e. each room would need to have pre-defined number of blocks set up. And this has to be done manually in the user interface. I added some basic code to the Gamesmaster editor to make this a lot easier (see later), but there is still the disadvantage that there can only be 32 blocksets, meaning after 32 rooms, we'll run out of predefined blocksets. I may in future adjust the code so that it always uses Blockset 1, which will have a high number of blocks already defined, and then disable the additional blocks not needed each time.

Reading the binary file into memory and using in Gamesmaster.

Reading in the binary file has to be done in Basic, so knowing it could get fairly large with a number of rooms, I have currently set it to load into Page24.

```
200 DEF PROC loadrooms
210 LET pg=24: LET pgaddr=(pg+1)*16384
220 LOAD "roomGM.bin"CODE pgaddr
230 END PROC
```

As we can only PEEK in the Gamesmaster code within normal page boundaries (can't Page in different memory banks) the Gamesmaster code is set to drop back to Basic whenever the room is changed, get the relevant single room data, and move it to memory location 16384. *This can cause issues when dropping back to basic at times, but so far seems to be fine when running the Gamesmaster game code.*

```
18000 get_room: RETURN : REM called by GM basic for each new room

240 DEF PROC get_room
250 REM called by GM to to get room data into memory
260 getr: REM get value of r from GM code
280 LET pg=24: LET pgaddr=(pg+1)*16384
290 LET checkr=PEEK (pgaddr)
300 IF checkr<>82: PRINT "Error: room data not found in memory": STOP : END IF
310 LET totrooms=PEEK (pgaddr+1)
320 LET roomrows=PEEK (pgaddr+2)
330 LET roomcols=PEEK (pgaddr+3)
340 IF r>totrooms: PRINT "ERROR: room request greater than rooms in memory": STOP : END IF
350 LET pgaddr=pgaddr+4+((r-1)*roomcols*roomrows)
360 LET r$=MEM$(pgaddr TO pgaddr+(roomrows*roomcols))
370 POKE 16384,r$: REM peek now used by GMcode for this address
380 LET baddr=((pg+1)*16384)+(totrooms*roomcols*roomrows)+4
390 LET bcheck=PEEK (baddr)
400 IF bcheck<>66: PRINT "ERROR: no room BLOCK data loaded": STOP : END IF
410 LET noblocks1p=PEEK (baddr+2+(r-2)+(r-2)): LET noblocks2p=PEEK (baddr+3+(r-2)+(r-2))
420 LET noblocks1=PEEK (baddr+2+(r-1)+(r-1)): LET noblocks2=PEEK (baddr+3+(r-1)+(r-1))
430 IF r=1: LET noblocks1p=0: LET noblocks2p=0: END IF
440 LET bllen=((noblocks1+noblocks2)-(noblocks1p+noblocks2p))*7
450 LET baddrs=baddr+2+(totrooms*2)+((noblocks1p+noblocks2p)*7)
460 LET b$=MEM$(baddrs TO (baddrs+bllen))
470 POKE ((16384+LEN (r$))+1),b$
480 POKE ((16384+LEN (r$))),(bllen/7)
490 END PROC
```

R is setup to be the room number in the Gamesmaster code

```
60 DEF PROC getr
70 LET r=DPEEK (DPEEK (va+34)+bs+(CODE "r" BAND &df)*6-390)
80 END PROC
```

Gamesmaster GMCL code

You still need to use Gamesmaster to setup all the sprites, animations, controls, sounds etc, this code just shows how to implement using the room maps.

When need for new room data is triggered, drop to Basic and transfer the data to a PEEkable address

```
*** MODULE 6 ***
LET R=R+1
IF Y>15:LET X=X-105
IF Y<16:LET X=X-10
IF Y<16:LET Y=Y+120
SCLEAR
CALLBAS 18000      (BASIC line address that gets new room data)
CLS
CALLMOD 11     (Setup new backgrounds / Sprites)
CALLMOD 10      (Setup new Blocks)
PLACE 1,X,192-Y,1      (Player sprite)
```

Now with room and block data for specific room in right location, PEEK required values

```
*** MODULE 11 ***
LET U=16384         (address where single room data  data moved to in Basic)
FOR Q=0,23,1         (assumes 24 rows)
LET Z=0
LABEL B
LET S=PEEK(U)
CALLMOD 13
REM view needed to synch screens and stop flashing
IF Q=8:VIEW
IF Q=16:VIEW
IF Z>31:NEXT Q                  (assumes 32 columns)
IF Q>23:RETURN
GOTO B

*** MODULE 13 ***
IF S=255:GOTO A       (255 significies no data, so skip entire row)
IF S=0:GOTO B
IF S>223:GOTO C         (>223 means consequetive 0s, so can skip ahrad)
CALLMOD 12
GOTO B
LABEL A
LET U=U+32
LET Z=Z+32
GOTO D
LABEL B
LET U=U+1
LET Z=Z+1
GOTO D
LABEL C         (Skip ahead number of locations)
LET S=254-S
LET U=U+S
LET Z=Z+S
LABEL D
```

```
*** MODULE 12 ***
LET M=Z*4
LET N=Q*8
IF S<128:BACK S,M,192-N,1          (<128 means background, assumes Frame 1 to be used)
IF S>128:PLACE S-128,M,192-N,1   (>128 means movable Sprite, assumes Plane 1 for sprites)
```

Block Data

```
*** MODULE 10 ***
BLOCKSET R          ( Blockset with same number as room)
LET S=16384+769   (Block data is currently 769 bytes after the room sprite data)
LET N=PEEK(S)
FOR M=1,N,1
CALLMOD 15
BLOCK M,U,Z,Q,P,T,W
NEXT M

*** MODULE 15 *** (Get data for each Blcok definition)
LET S=S+2
LET U=PEEK(S)
LET S=S+1
LET Z=191-PEEK(S)
LET S=S+1
LET Q=PEEK(S)
LET S=S+1
LET P=PEEK(S)
LET S=S+1
LET T=PEEK(S)
LET S=S+1
LET W=PEEK(S)
```

Current limitations of code:

- GM code setup on assumption of 32x24 grid for room data. Should be possible to retrieve columns/rows from data in memory if really needed.
- No custom properties set up in Tiled to allow different Frame Number for background Sprites, or to place moving Sprites on other Planes.
- Used up a few of the limited number of variables, would be nice to look at someway to save these values before/after these modules so that they are still available for other modules to use.

Other considerations

Because the Gamesmaster game now needs custom Basic and other code files (room binary file) it was not possible to test the game within the Gamesmaster editor anymore. When running as a standalone game, the error messages are not visible, making it more difficult to debug the game.
I have made a custom version of he Gamesmaster editor which now includes:

- Ability to load ".bin" files from the Load menu.
- Has the required additonal Basic lines to move roomdata to the correct memory location when called.
- Has an additional function in the BLOCK definition menu item to automatically add custom number of BLOCKs to each Blockset, so you don't have to manually add each one for every room.

I have found this version to be a bit buggy though, I expect because of where the room data is getting placed (16384) and its impact on the much larger Basic program and variables in memory. This is still very much a work in progress!