



## Building a Platformer – Tutorial 9

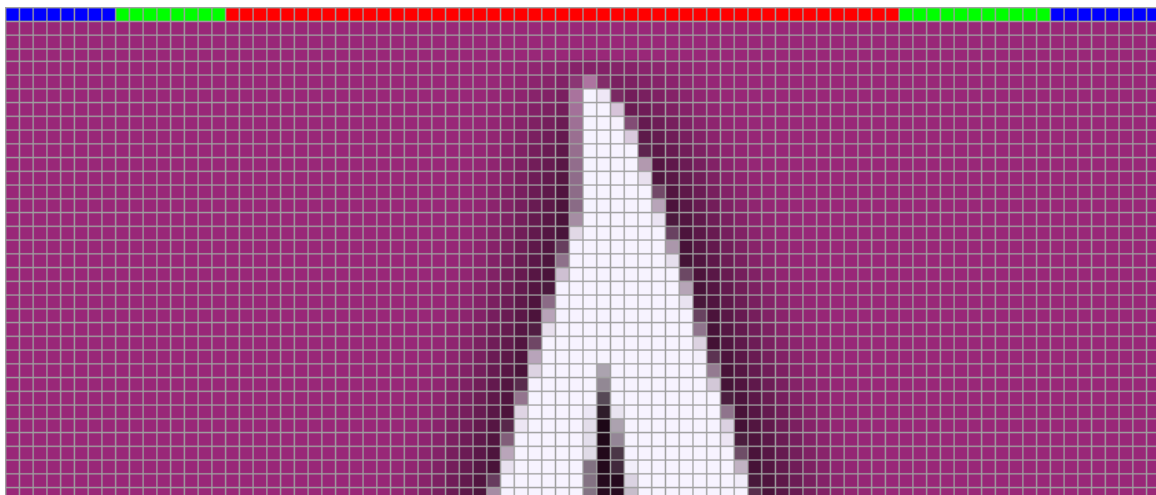
### BITMAP FONTS

There are several techniques you can use to get pretty text into your game, but there is usually a downside to each. Text commands are quick and easy but you are limited to single colours, not the most pleasing to the eye. Pasted images rely on screen resolution, as they can't be scaled easily, and take up a lot of images.

I wish I had thought of the method I've adopted, years ago. Really the idea to use sprites and adjusted UV coordinates was part of a demo supplied with GDK, I only learned about it after watching the Grant Clark convention video. To me it's simply genius, all the commands are there in DBPro to get, what I consider the ideal text system. By using sprites you can change text colour, rotate, fade out, scale. My favourite benefit though is by setting the UV coordinates of the sprite the system only needs 1 image, the text width data is read from the image then a sprite for each letter is created.

ASCII character sets are really quite nicely laid out already, so it makes sense to stick to that standard, character 32 in ASCII is a space, so I like to start from there and work forwards. This way a text string can be converted to individual ASCII values, and you don't need to cross reference the character set.

The letter A looks like this:





The backdrop is transparent, but shown in purple here for clarity. There is a scanline above each row of characters, which is read via memblocks and the kerning of each letter is set. The blue area is a space between the characters, the red area is the actual width of the character and the green in-between the blue and red is the width of the actual character with drop shadow included.

This allows for neater text as each character is only restricted in it's height; it can be any width, so troublesome characters like W don't need special conditions.

The font setup is called from the setup.ini file in the media folder, the font image is loaded then the command `'makefont(0,600,32,128)'` will scan the image for each characters widths and store them in a typed array called `bitfont()`. If there is a downside to this technique it has to be the extra housekeeping needed to support the variable widths and kerning. Once the hard work is done though, putting text onto the screen is nice and easy and affords us a lot of freedom.

The array `bitfont` has the following elements:

```
uxa as float
uya as float
uxb as float
uyb as float
width as float
height as float
lkern as float
rkern as float
```

Because sprites are used, they have to be cleared out when the text is being re-drawn. This is done via the `bitfont_clear()` function, the text sprites are set to start from 100 to avoid any other sprites that might be in use by the editor. A global integer called `bitfont_cur` is used to keep track of the current sprite, then only the sprites that have been assigned need be hidden, and if `bitfont_cur` has not increased the function is cancelled.

```
function bitfont_clear()
  if bitfont_cur=100 then exitfunction
  for p=bitfont_cur to 100 step -1
    if sprite exist(p)=1 then hide sprite p
  next p
  bitfont_cur=100
endfunction
```

The main function for actually placing text is called `bitfont_left(img,fontnum,x,y,t$,size,col)`, by passing the image, font number, position, text, and size and colour to the function it will place the sprites as needed. The size is tied into screen resolution using a multiplier global called `bitfont_scale#`. This is set depending on resolution so that the same text code can be used without worrying about screen resolutions. There is a similar format function called `bitfont_length(img,fontnum,t$,size)`, which returns the length in pixels of the specified text. This allows for easy text formatting, to center text for instance all that is needed is to calculate the length, divide by 2, then subtract that from the screens middle and pass everything back to the `bitfont_left()` function.

So when adding things like lives and score displays, it's just a case of deciding where on the screen to put them.

## MENU SYSTEM

With the basic bitmap font system in place, it made sense to add a menu screen. I find that adding polish sometimes even when the game is not near completion can be a good way to replenish interest and get back in tune with a project. I decided I wanted something kinda artsy on the main menu, so our 3 stooges have been cloned and painted red, and they will battle it out forever on the menu screen for my amusement (if nobody else's). A simple level screen was made at the bottom right of the map, and the menu screen was added as a block, then placed on the map. The main menu demo needs Bruce to fight for himself, the standard player update function was duplicated then renamed as `demo_player()`. This version is stripped out a lot as the characters don't need to do anything except run back and forth looking to fight Bruce, and Bruce fights back now, it can get quite bloody down there.

Using the cursor keys up and down and space, you can set the player controls, exit, start the game etc, although not complete yet, it gives a good indication of how it will look when done. Once the game is started the player control screen will appear and is also accessible with the space bar during play. As a standard now when Bruce starts it will be like a 1 player game, this is due to the maturing of the game logic, as now to properly edit the levels I have to collect lanterns from the previous screens, otherwise it's tricky to keep track of portals and barriers. The aim is to smash every lantern and finally kill the big boss at the end. In the original game the boss was very easy, just run along and collect the last lantern while avoiding fireballs – I think I can be far more cruel than that!

## LEVEL LOGIC

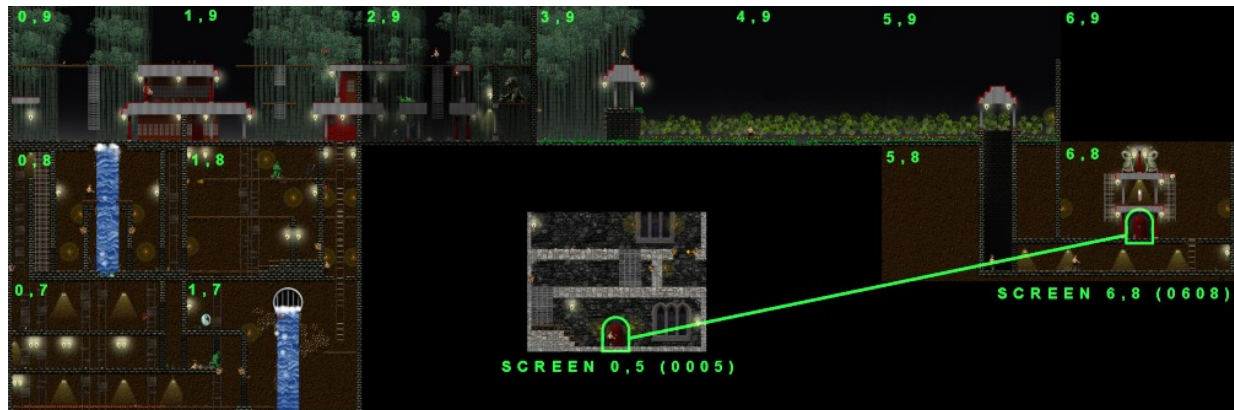
When it comes to handling portals to other areas, or barriers blocking areas, I like to keep it as simple as possible. It's best to consider exactly what you will need to happen and how you can do that on a frame rate budget. The logic needed for Bruce is very straightforward, areas have to be blocked, and although not a big factor in the original game, I'm using doors to get to other areas. So I decided that for each map block location I could set a command and variable that I would specify by pressing the 'C' key and entering the values. It's as cheap as that because integers and bytes will not cost a great deal to parse, as opposed to text strings for example.

2 new variables have been added to then map type:

```
com as byte
com_val as integer
```

The number of lanterns collected is now displayed at the left of the screen, so this is used primarily to unblock exits. A 'com' value of 1 denotes a barrier, and then the 'com\_val' is used to specify the number of lanterns needed before it's unblocked. If the area is blocked, a brick wall replaces whatever was in the map space before it, then after collecting a lantern the screen is checked and changed to suit any barrier changes. A particle effect of the bricks falling down has been added as well which helps to highlight the route ahead.

Doors are also a new addition this time; these are simply blocks that wait on Bruce pressing up in front of them. The exit is set by using a 'com' value of 2 and using 4 digits in 'com\_val' to represent the screen X and Y position of the exit. The door exit reminds me a lot of the original Prince of Persia game, Bruce turns and runs through the door, the engine goes into a custom loop then so the screen will freeze for Mr.Green and Ninja. It would be possible to have them run around at the same time, but I will have to make sure they can't hurt Bruce when he's entering a portal so I will look into it for next month. It might be fun to taunt Mr.Green and Ninja by entering the door just as they're about to hit you.



The door in screen 6,8 leads to the door in screen 0,6 – each door has a command block pointing to the opposite door. Screen 0,5 has been moved for illustration purposes. When setting the command block, it's just a case of making a 4 digit number from the X and Y, so screen 6,8 means 06 x and 08 y – 0608 – this way I only need to check a byte value and an integer value to know which screen the portal leads to.

### BLOCK SPECIFIC LOGIC

As level design progresses the need for special block handling becomes a necessity. So the function `update_blocks()` has been added which scans the whole current screen for special blocks. Right now it's only used for waterfalls, but it will play a bigger part as the game takes shape. If it finds a block with 'SCROLLTEXWATER' it will scroll the texture, and check a random bone on the characters to see if they are in range, then it will create a splash particle and force the character down. On waterfalls the player has to keep climbing or they get washed away, except for the waterfalls that flow up, Bruce Lee was quite a surreal game in places!

A new collision setting has been added, specifically for disappearing platforms, these are simple animated blocks with 100 frames, and if the frame number is above 50 then the block collision is disabled. The enemy characters do a fairly good job of getting over these if they have to, but it is fun to punch them just as they are about to make it to safety.

### PARTICLES

Another hot topic in the convention videos was using stacks to help keep recursive checks at bay. I'm sure there's more that can be done, but the particle system is a lot more robust now, just by applying a very simple stack technique. While the particles are being updated, a list of dead particles is made, and then this is used when a new particle is needed. This gives much more control over particles lives, as instead of skipping particles that are alive when looking for a new particle ID, it's simply taken from a pre-created list. Every particle is still checked to see if it's alive or dead, but now particles will never be used before they die so it's much more dependable and there will be no disappearing particles.

## EDITOR CHANGES

The most useful change I've done is the automatic block positioning and queuing. When placing blocks that are part of a set, say 3 stone bricks that go together, it can be a pain to have to place each one in sequence, so 2 new integers have been added to the block type. Chain is used to specify the next block that should be placed horizontally, there's no real need to support vertical automation as well. Next is used to specify the next block to set as the edit block, this means that if there are several versions of a block that you want to cycle through, it's less work. These variables are set in the setup.ini file, the blocks that would be set automatically do not get an icon now, so are not selectable. So they don't take up space on the block selection screen and it should save a lot of time, and make it easier to get more random spreading of tile variants.

## NEXT MONTH

One thing that I will cover is adding some shading to the level, I want Bruce and co to get darker when they are in dimly lit areas, it's quite a slow process but I quite like the results, often some tweaking is needed but I feel that the extra work is worth it, those underground areas are a lot more spooky now. The level layout will take shape more and more as well, there's quite a good play area already but lots of space left to fill. Finishing the level layout will be the main thing for the next month, adding new media here and there, and looking for inspiration for that final boss character. Now would be a good time to source a title track as well, so if you have some oriental music and a desire to see your name in my pretty bitmap font, please email me at the address provided in the newsletter.

## Editor Controls

- **Left** and **Right** cursor keys to run left and right.
- **Down** cursor key to lie down.
- **Up** cursor key to jump.
- **Control** key to attack.
- **Shift+Arrow keys** to select a screen.
- **L** to re-load the map.
- **S** to save the map.
- **U** to undo changes made on current screen.
- **B** to bring up the block selection screen.
- **M** to bring up the map screen.
- **C** to specify a COM and COM\_VAL at the current position.
- **Return** to render a 2D lightmap – more on this next month...