The Right Connections

Damian Walker continues this series with a look at Connect 4, by Ben Vaughan

Ben Vaughan's Connect 4 is the only shareware offering in this otherwise freeware line-up. Its graphics are designed only for the Series 5, though it runs quite happily in letterbox mode on the Series 7, and presumably on the Geofox too.

This game uses a slightly larger than usual board, at 8 slots by 8 rows, rather than the more standardised 7 by 6. This is not a disadvantage for most people, but those practising for tournament play might have preferred the standard board. I would have liked to set Connect 4 against some of the other programs in a tournament, but the lack of a standard board size prevented this. It is unfortunate that none of the games in this line up have the option to change the board size.

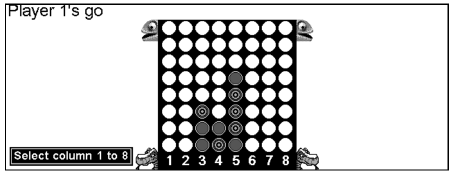
Its most interesting difference from the other games is that, rather than offering multiple levels of difficulty, it instead offers four computer players with different styles of play. This would have been even more interesting had the program allowed the computer to adopt one of the styles at random, as not knowing what kind of player you are up against would have added to the replay value.

A characteristic of all Ben Vaughan's games is the extensive use of sound. Connect 4 is no exception, with speech being used throughout the game. During the game itself, for instance, a voice announces the slot number into which you've dropped a piece. It wasn't long before I started to find the sound irritating, though, and I eventually felt more like I was travelling in a lift rather than playing a board game. Thankfully, the sound can be switched off at any time.

Despite its good presentation both in sound and graphics, Connect 4 is quite a simple program, only lightly loaded with features. You have the option to decide who goes first– strangely, it's only the simplest of the programs that offer this option. Help is offered in the form of a single dialog box. That's not a criticism: the game is straightforward enough, and widely known, so a single dialog might be sufficient. There are no in-depth features that need to be described.

On the technical side, Connect 4 doesn't score well. It doesn't respond to operating system events like requests to close, so it might be a nuisance if you've got it running at the same time as trying to make a backup. But the program is otherwise reliable, and is still supported at the time of writing, so it might be a good option for owners of the Series 5.

In next month's article, I'll be looking at the next game in the line up, Four In A Line, which was originally by Purple Software, and since then was taken over by ZingMagic and released for free.



EPOC ENTERTAINER
Issue 4: March 2008
Edited by Damian Walker
This month...
Games on the Osaris

Welcome to a new issue of *EPOC Entertainer*. This month there is a treat for Osaris owners, who often have reason to feel a bit left out. The two running series also continue. As always, I welcome feedback on the contents, so please do get in touch at the email address below.

entertainer@snigfarp.karoo.co.uk

Animating OPL

In this month's tutorial by Damian Walker, we'll load the ball sprite and place it on the screen.

Before we can put a ball on the screen, we need to load its bitmap images. The method of doing this is similar to loading static bitmaps. But instead of using gLOADBIT, we need to use a procedure from the OPX, BITMAPLOAD&, if those bitmaps are intended for use in a sprite. The following procedure, which should be added to the end of the program, loads all of our ball bitmaps and masks in one go:

```
PROC LoadBall:
    ball&(1)=BITMAPLOAD&:
          ↗("\Bouncer\Ball1.mbm",0)
    mask&(1)=BITMAPLOAD&:
          ↗("\Bouncer\Mask1.mbm",0)
    ball&(2)=BITMAPLOAD&:
          ↗("\Bouncer\Ball2.mbm",0)
    mask&(2)=BITMAPLOAD&:
          ↗("\Bouncer\Mask2.mbm",0)
    ball&(3)=BITMAPLOAD&:
          ↗("\Bouncer\Ball3.mbm",0)
    mask&(3)=BITMAPLOAD&:
          ↗("\Bouncer\Mask3.mbm",0)
ENDP
```

This procedure uses global array variables ball& and mask&, whose elements have a similar purpose to the floor% variable in LoadFloor: they identify the individual bitmaps we'll be using for the ball. We need to declare these arrays in the main Bouncer procedure, which you should modify to read as shown here:

```
PROC Bouncer:
    GLOBAL ball&(3),mask&(3)
    DrawFloor:
    LoadBall:
    DO UNTIL GET=27
ENDP
```

If you translate and run this, no effect will be apparent. The lack of any error message will tell you that the program has no errors and that the bitmaps were loaded successfully, but we still can't see the ball. To rectify this, we need to put these bitmaps together in a sprite, and display that sprite on the screen. That is the purpose of the following procedure, which you should add to the end of the program:

```
PROC PlaceBall:
    spritex%=gWIDTH/2-8
    spritey%=gHEIGHT/2-8
    sprite&=SPRITECREATE&:
           ⬀(1,spritex%,spritey%,0)
    SPRITEAPPEND:(375000,ball&(1),
                    ⬀mask&(1),1,0,0)
    SPRITEAPPEND:(250000,ball&(2),
                    ⬀mask&(2),1,1,1)
    SPRITEAPPEND:(125000,ball&(3),
                    ⬀mask&(3),1,2,2)
    SPRITEAPPEND:(250000,ball&(2),
                    ⬀mask&(2),1,1,1)
    SPRITEDRAW:
ENDP
```

The global variables spritex% and spritey% are used to find the centre of the screen. The offset of -8 is used to take into account the fact that the sprite's size is 16×16 pixels. The co-ordinates of sprites, like static bitmaps, are referred to by their top-left corner, so to centre a sprite around any particular co-ordinate, you need to subtract half its height and width from that position.

The SPRITECREATE& procedure brings our sprite into existence. The first parameter is the drawable onto which we want the sprite to appear; you'll remember that 1 is the ID of the background screen. The next two parameters specify where we want the sprite to appear; this will be in the centre of the screen as we've already calculated. The final parameter allows us to make the sprite flash; we don't want this, so the parameter is zero. The global sprite& variable is a unique ID for the sprite.

At this point, the sprite is still invisible, as we've not said what we want it to look like. That's what the following four lines are for. Each call to the SPRITEAPPEND procedure defines a single frame of animation. The first parameter is the number of microseconds for which the sprite will appear. The next two parameters are the ID of the bitmap and mask we want to use for this frame. The next parameter, 1, specifies that we want to use the mask to blank out pixels, as we did in our Temp procedure. The final parameters specify an offset for this frame. You'll remember that the second and third frames that we drew in Sketch are each smaller than the frame before; we therefore have to modify the offset in order that the ball stays centred as it bounces. If you find this explanation unclear, try changing these to 0,0 in every case.

The last line, a call to the SPRITEDRAW procedure, is what actually makes the sprite visible. Before any of this will work, you need to make the following modifications to the Bouncer procedure, to declare the new variables and to call this new procedure:

```
PROC Bouncer:
    GLOBAL ball&(3),mask&(3)
    GLOBAL sprite&,spritex%,
                             ⬀spritey%
    DrawFloor:
    LoadBall:
    PlaceBall:
    DO UNTIL GET=27
ENDP
```

When you run this program you'll see that the ball is automatically bouncing up and down. No further action is needed on the part of your program to make this happen. In fact, as you watch the ball bounce, all your program is doing is waiting for you to press the ESC key.

In the next issue we'll make the ball move from its central position and bounce around the screen.

The Oregon Trail

A closer look at gaming on the Oregon Scientific Osaris, by Damian Walker

When developers release games for EPOC32, one machine that is often forgotten is the Osaris. It's a fairly solid machine that compares well with the more successful Revo: it has the same 8MB RAM, plus a backlight and a CF slot. What is it that makes it undesirable as a target platform?

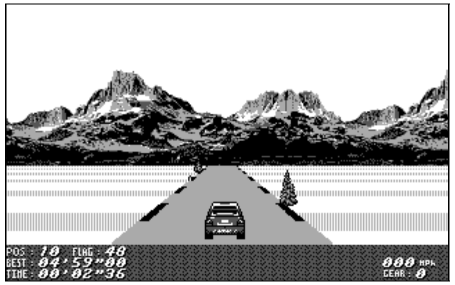
There are two probable reasons: its lack of sales compared to the Revo, and its 320×200 screen—the lowest resolution of all EPOC32 machines. There is no reason why this resolution *should* be an impediment, though. A lot of games on the PC worked admirably at this resolution, and games are now developed for smart phones with smaller resolutions still.

There are a number of EPOC32 game developers who have agreed on this point. About 50 games of the 300 I'm aware of support the Osaris screen, either by design or accidentally. Some programmers have made a conscious effort, tailoring their programs to adapt to the smaller screen. Examples of this include *Dice5*, *V-Rally* and my own *Senet on the move*. This approach can take some effort, though, with the need to design special on-screen graphics to fit the Osaris screen.

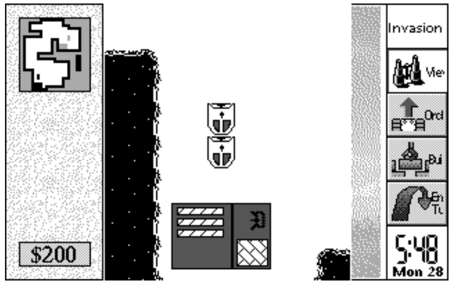
Other developers have gone down a more elegant route. Some games re-size themselves automatically to whatever size the screen happens to be. On occasion this doesn't quite work—sometimes there is simply too much on the screen to cram into a small resolution, and while developers may have kept the Revo in mind, the Osaris has been overlooked. But in many cases, like *Encore*, *No Man's Land*, and *Shut the Box*, the results are very good indeed.

Occasionally some minor screen glitches occur, like slightly misplaced scroll bars in *SimCity*, but these don't affect the ability to enjoy the game on the Osaris. A special mention must be made for the games of *PsionGames* and *Ten Dan*, who have managed to make their games scale flawlessly to fit everything from the Osaris to the Series 7, albeit in monochrome on all platforms.

The Osaris is therefore not so bad a machine for games as you might imagine. Its Compact Flash support, in contrast to the Revo, allows you to partake of these frivolous pursuits without taking up precious RAM, and the backlight lets you play in low light conditions. Future editions of *EPOC Entertainer* will therefore be keeping a special eye on the Osaris.



V-Rally, by TomTom.



Invasion, by Thomas Ashton.