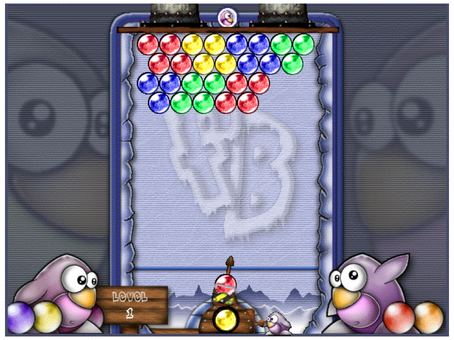
Blowing Bubbles

The new Frozen Bubble game, as reviewed by Damian Walker



A newcomer to the EPOC32 gaming scene, Frozen Bubble was released for the Series 7 and Netbook in October 2007. It's a puzzle-style arcade game, where the object is to burst all the coloured bubbles on each level with the aid of a directed cannon. The bubbles are stuck to the ceiling of each level, and the bubbles must be burst before the ceiling is lowered to the ground. The cannon fires coloured bubbles, and must be aimed so that they hit bubbles of the same colour, which then burst and fall to the ground.

What's most striking about this game is its addictiveness. Once started, it's difficult to put down. The graphics are excellent, much better than we're generally used to on this platform. Sound is also rather good. The downside of this is that the sound and graphics take up a lot of space: the whole package takes nearly 6mb of space, and a further 1.8mb of memory to run. There's little that a developer can do about this, however, as a large expanse of detailed graphics will always take up a lot of bulk.

There are a few downsides to the game. Compatibility across the EPOC32 range is almost non-existent. The game runs on the Series 7 and Netbook only, and given the reliance of colour and the vertical orientation of the level screens, it's unlikely that a Series 5 or Revo port will appear. The game also runs a little slowly, with noticeable delays when bubbles are about to burst.

While the interface is straightforward it doesn't even pay lip service to EPOC32 conventions, so your Menu key won't work, and Control-E won't exit from the game. The program does respond to some EPOC32 events, though, and it is possible to close it from the system screen.

All the basic features are present in this game: sound can be controlled, and the keys can be changed to your preference. There's also the opportunity to save a game if you have to leave it half way through.

All in all this is a very good game for the platform, and the authors should be encouraged to write more. It certainly has the “wow” factor, and can be used to show off your EPOC32 machine to your friends. If you're not short of memory or disk space, and enjoy a challenging puzzle game, then I can heartily recommend it—it's free, after all.

By	Patisoners
URL	www.vakoveverky.net
Licence	Freeware
Compatibility	Series 7 & netBook
Rating	☆☆☆☆

Issue 2: January 2008

Edited by Damian Walker

EPOC entertainer

Happy New Year, and welcome to the second issue of *EPOC Entertainer*. The last issue was well received, with a number of positive comments sent back to me by email and on the forums. Keep them coming!

I'd be particularly interested to hear what else you'd like to see in the magazine. Even better, volunteers for article writing would be welcome...

I've managed to squeeze in two reviews in this issue, as well as the second part of our popular programming tutorial. This month we'll actually get around to some OPL programming!

Keen-eyed readers will notice some stylistic changes to this issue. As *EPOC Entertainer* is quite new, I'll be experimenting with style over the next few months until, hopefully, I can settle on something both you and I are happy with.

entertainer@snigfarp.karoo.co.uk

Animating OPL

The second in our programming series by Damian Walker

Laying the Tiles

Now we can get down to some programming. Create a new Program file called Bouncer.opl. We'll start simply, by putting the tiled floor on the screen. This is done very simply: by loading the Floor.mbm image and placing copies of it across the whole screen. The following procedure does this for us:

```
PROC DrawFloor:
 LOCAL floor%,x%,y%
 floor%=gLOADBIT
                ⬀("\Bouncer\Floor.mbm")
 gUSE 1
 x%=0
 WHILE x%<gWIDTH
  y%=0
  WHILE y%<gHEIGHT
   gAT x%,y%
   gCOPY floor%,0,0,16,16,3
   y%=y%+16
  ENDWH
  x%=x%+16
 ENDWH
 gCLOSE floor%
ENDP
```

Lines starting with ⬀ should be added to the previous line. Here's how it works. Up to 64 bitmaps or windows, collectively called *drawables*, may be loaded into memory at a time. Each is given a unique ID number, by OPL, not by the programmer. This is what the floor% variable is for. The x% and y% variables are counters used to count our way across and down the screen.

The gLOADBIT function loads the Floor.mbm image, and returns the ID number for us to use to refer to it. It also has the side effect that any further drawing commands will manipulate the newly loaded bitmap, rather than the main screen. We don't want this, so the gUSE command tells OPL to turn the attention of subsequent drawing commands back to the screen, itself regarded as a drawable whose ID is 1. The OPL manual refers to the destination of its drawing commands as the *current drawable*.

The gAT and gCOPY statements within the two nested WHILE loops copy the floor tile to each successive location on the screen. The gAT statement tells OPL to turn its attention to a particular part of the current drawable, while gCOPY copies there all or part of another bitmap. In our case, that other bitmap is the floor tile bitmap floor%, and the following four parameters indicate that we're copying all of it (16×16 pixels aligned with its top left 0,0). The final 3 is a flag to indicate that the floor tile should completely overwrite what was there before.

At the end of the procedure, the gCLOSE statement tells OPL to forget about our floor tile bitmap, now that we've finished with it. Remember that OPL can keep track of a maximum of 64 drawables at a time. While that's more than we're going to need for this demonstration, it's always good practice to close a loaded bitmap, or any other drawable, when it's no longer needed.

Drawing the floor is only one of a series of tasks we'll be performing in our demonstration, so I've put it in its own procedure to keep it self-contained. In order to organise this and future procedures, you will need to add the following procedure to the *top* of the program:

```
PROC Bouncer:
 DrawFloor:
 DO UNTIL GET=27
ENDP
```

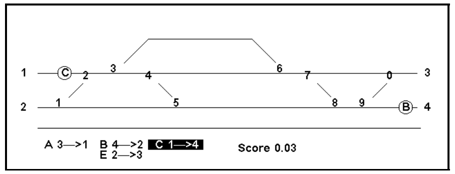
There's not much here at the moment, but we'll be adding to it as we develop new procedures. Currently it calls our DrawFloor procedure, and when that's done, it waits for the ESC key to be pressed before quitting the program. With the graphics drawn, and these two procedures entered, the program is ready to translate and run.

If you think I'm glossing over some of these new statements and functions, you're right. The OPL manual gives a more detailed explanation of these commands, and if you want to learn more about these and future statements and functions I'll introduce, then I recommend you keep the manual handy and look them up.

In the next issue I'll be introducing *sprites*, the animated moveable graphics that we'll be using for the ball itself.

Off the Rails

A look at AORailSim, as reviewed by Damian Walker



On desktop machines, there is a popular genre of games almost entirely missing from the EPOC platform: the railway simulator, On the PC, this is exemplified by Railroad Tycoon, Transport Tycoon, Transarctica and a host of others. The closest we come on the Psion is Adelino Oliveira's *AORailSim*,

AORailSim is a simple real-time puzzle game that concentrates on the routing of trains along an existing section of railway. The aim of the game is to route trains safely from one end of the track to the other, by ensuring that various sets of points are set correctly at the crucial moment before the train passes them. Failure means that two trains can suffer a head-on crash, losing the game.

Like all games from this author the graphics are very simple, In some instances simplicity is best, but the spartan graphics of AORailSim go a bit too far. Railways are represented by plain straight lines, trains by circles containing a letter. Points are numbers next to the appropriate junction.

Sound is non-existent in the game. While this is useful for a quick game in a crowded room, it would have been nice to have a few noises that could be switched on or off.

Of course, it could be that game play saves the game where presentation fails. In this case, I think that the lack of graphics and sound makes the game completely devoid of atmosphere, so that even railway fans will find it difficult to identify with the theme. The idea of the game is sound, though, and I find myself giving it a brief try every now and then.

On the technical side, the simplicity of this bare-bones game does give it a small memory footprint, and leaves little opportunity for bugs to creep in. It also respects standard EPOC conventions, like Ctrl+E, and system events.

On the other hand, there is no compatibility with screen sizes other than the Series 5. This presents no problems for users of the Series 7, netBook and GeoFox One, but Revo and Osaris owners are left out in the cold. The timing of the game appears locked to the machine's processor rather than its clock, meaning that speed settings are not consistent across machines, but this is unlikely to be a problem.

All in all one must judge this game for what it is: a simple puzzle game kindly released free of charge by its author. But fans of railway simulation and management ought not to pin their hopes on this game.

By	Adelino Oliveira
URL	www.tucows.com/preview/57363
Licence	Freeware
Compatibility	Series 5 & 5mx
Rating	☆☆