SEPTEMBER 2009 £FREE

BlitzMAX coder

0.2

Interview with a 'Grey Alien'

Trouble with thrust or asteroids?

We disassemble an SLR

Bowled over by 3D?

TimelineFX

BlitzMax coder - Issue 02

BlitzMAX coder 0.2

Hi and Welcome to the second issue of the BlitzMax Coder Magazine!
In this issue we will introduce you to 3D, FX, Aliens and SLR's enjoy!
blitzmaxcoder.com

News

BlitzMax coder delayed due to crunch time for new game...

Well Arowx Games have launched a beta of our latest game mmjChallenge.

http://www.mmjChallenge.com

It's a Mah-Jongg game with challenge codes that allow you to compete with your friends and family!

Got a News item?
news@blitzmaxcoder.com

Want to provide Feedback?
feedback@blitzmaxcoder.com.

Want to twitter?
http://twitter.com/arowxGames

About BlitzMax

BlitzMax is a Basic programming language with Object-Oriented features and easy to use commands for graphics and sound.

Combined with an active and helpful community forum with additional modules for more advanced features.

BlitzMax is cross platform, allowing you to develop games and applications for Mac, Windows and Linux based systems.

BlitzMax can be found here:
www.blitzbasic.com

Try the free demo version to see it in action.

Next version of BlitzMax Faster...

There have been some reservations regarding the multi-threaded speed of BlitzMax's automated garbage collection system, without the developer having to intervene in the task of taking out the trash!

Direct from Mark Sibly the next version of BlitzMax is targeted to resolve this issue allowing you to develop super-fast multi-threaded applications without having to worry about taking out the trash!

Version 1.34 is due out shortly so by the time you read this it may already be out!

So what are you waiting for go on get multi-threading...

Check out http://www.blitzbasic.com for details...

Sections

Interview with a 'Grey Alien'

Today we have an interview with Jake Birkett director of greyaliengames.com and games developer for BigFishGames.com.

Allan: Hi Jake could you tell us a little bit about yourself please?

Jake: I'm a 34-year old programmer/designer working for Big Fish Games in Vancouver, although I'm originally from the UK. I've got a partner, two sons and a cat. I've been programming since the age of 8 when I got a Spectrum 48K - in those days computers came with BASIC programming manuals. I have a 2nd dan in Aikido, and I'm into self-improvement and positive thinking. My hobbies include playing computer games, playing the guitar, going for walks, and reading.

Allan: How long have you been using BlitzMax now and how did you come around to developing with it?

Jake: I've been using BlitzMax since Spring 2006, so over 3 years. Years ago I used Blitz Basic 2 for the Amiga and I loved it, then in November 2004 I was browsing the Internet and I found out about BlitzPlus, which I promptly bought. I used BlitzPlus for about a year an a half before I finally realised that I really needed BlitzMax for its OOP lushness and 3D card effects such as alpha blending, rotation and scaling.

Allan: You appear to have made the transition from indie games developer to games developer for Big Fish Games, in a very short time, could you tell us more about this transition and what you think were key factors in this progression?

Jake: Well I'm not sure it was a very short time, I'd say it took about 4 years!

After I bought BlitzPlus I was making a kung fu platform game in my spare time called Iron Fist, but after about 6 months I realised that it was going to take years to finish and that it probably wouldn't sell many copies either.

I'd recently played Bejewelled and I liked it and thought that I could program something like it.

So I switched to making "Casual" games AND I gave up my day job developing business software and doing IT consultancy.

That was a key time period - I took a big risk and survived on very little money for quite a long time.

At Christmas 2005 I released Xmas Bonus, my first Commercial game, and this was followed by Easter Bonus in the Spring. Then I switched to BlitzMax and programmed The Wonderful Wizard of Oz for a producer in the US.

That game got spotted by Emmanuel Marty (of Azada fame) from Big Fish Games who said that he thought it was well made and then asked me if I'd be interested in programming a game for them as a 3rd party developer. I said "Yes", although I did think about it quite a lot because I was keen to start producing my own games with small teams, but I thought that I could learn a lot from working with Big Fish Games. Before I started on their game I quickly released Holiday Bonus, which I'm really pleased with.

Then I made Fairway Solitaire for Big Fish Games, which was a smash hit card game. It was actually a improved version of an existing online title that they had on their site - I didn't design it, I just programmed it.

Due to the success of that game they offered me a contract to make more games for them and I started on Unwell Mel, my 6th commercial game.

Midway through that game the VP of BFG Studios casually mentioned that they were opening a new studio in Vancouver and asked me if I would like to be the first programmer/designer there. I discussed this with my partner because it was a pretty big decision and then I said yes and we moved over here in November 2008.

Now I work full-time for Big Fish Games in a cool studio in Vancouver with a really great team of programmers and artists.

So, uh, that's quite a long story, but I think that key factors

Interview with a 'Grey Alien'

were: the fact that I finished several games to a high quality, I kept thinking positive, and that I said "yes" to opportunities that presented themselves to me.

Allan: How would you compare the life and work of the solo indie game developer with the life of developing games in a large company?

Jake: As a solo developer I was able to start and finish work when I felt like it. I used to lounge around in my garden on sunny days and even grew my own vegetables. I was able to take time off whenever I wanted although I didn't get paid whilst I was on holiday.

It was pretty cool being an Indie developer, but working with remote teams was a pain in the ass due to having to write tons of emails to people who were often in different time zones.

Working in an office with other programmer/designers to bounce ideas off and with artists who I can talk to in person is really great. I just took a vacation to the UK and I got paid whilst I was there, which is novel to me.

Also I have a lot of freedom with the game I'm making right now which is great. Furthermore Big Fish Games employs many very talented people, and I like to talk to them and learn from them.

Vancouver is a fantastic place to live and I enjoy networking and meeting interesting people. I've even become an advisory board member for the Vancouver Film School's Game Design course.

Allan: What advice or tips would you give to aspiring games developers?

Jake: Practice! You can read and dream about game development all you want, but actually making games is the best thing you can do.

Start with small mini-games so that you can learn the basic skills and then gradually build up to bigger projects. Many developers start with an epic idea (or a game engine) that is way too big and thus they never finish it.

Build your game engine as you are making games and improve it with each game.

Don't just play games, study them!

Look at all the details, figure out how they work and what makes them good or bad.

It's best to learn these skills whilst you have a "normal" day job, although you will need to sacrifice other things like watching TV, getting drunk etc to spend as much time as possible programming.

Don't sacrifice sleep or healthy eating and exercise though.

Years ago I read an article by Steve Pavlina called "Cultivating Burning Desire" (Google it).

I knew that I REALLY wanted to make games for a living so I gunned for it. If you want to succeed then you have to want it badly and you have to think positively and work hard. Also make friends in the game industry as you never know what opportunities they may offer you in the future.

Allan: You have quite a few games under your belt and even a games development framework, what has been your favourite project so far?

Jake: Wow, this is a tricky question because I've enjoyed working on all my games.

My first game was exciting, simply because it was my first, and all the others were cool too. I'd say that my favourite project is actually my current game because I'm working with such a great team and I'm having a great time designing the game and seeing the amazing artwork that they are generating.

Every game has fun parts and boring parts. Prototyping is fun but the gameplay details can become a grind later on as can making the multitude of extra screens that a game needs - and localisation is a real slog. The best parts are

Interview with a 'Grey Alien'

coming up with the initial design and also seeing the game sell really well when it's done ;-)

Allan: Looking back over the games you have developed and what you have learnt are there any key techniques or technologies you would recommend to other developers?

Jake: Well obviously I'd recommend BlitzMax. I'm still using DirectX 7, and I don't have any fancy plug in modules.

You should definitely build a game engine over time, or try to get hold of some of the other ones out there to learn from.

Also it's worth compiling your games on Mac as well as PC because Mac games can sell pretty well.

To sell your game on the big portals you need to make sure that your game is Vista compatible, which is a pain.

Also most casual games follow a whole bunch of standards which you can work out by playing the best of them, or by asking on a forum like Indiegamer.

These days to make a top selling casual game you need to have great art and music, which normally would mean spending a lot of money unless you can find a team who will work for royalties.

Particle effects and polish are also vital for the best casual games but know when to stop adding polish and launch the game (avoid feature creep!).

If you are making a casual game, make sure it has an easy learning curve and don't punish the player - so many games get this wrong.

Get other people to test your game who are in your target audience - just sit there and watch them and make notes.

Some tech tips:

> Use Bitmap Fonts for speed and flexibility (like adding textures and outlines);
>
> use OpenAL or DirectSound in Vista, not FreeAudio which has a lag problem;
>
> use DirectX not OpenGL on PC because more people have it installed.
>
> Finally, believe in yourself!

But don't necessarily expect your first game to be a smash hit - it takes lots of practice!

So don't give in if your first game bombs, keep going and by the 5th or 6th game, you should be on a roll.

Allan: How have you found developing with BlitzMax?

I love it. It's flexible, powerful, fast and easy, and it has a great community forum. It's probably the most fun I've had in any language, and I've used a lot over the years. I like the fact that I can see my ideas turn into reality quicker than in most other languages.

Allan: There is a budding community developing BlitzMax it's modules and a framework, are there any features, modules or technologies that you would put on your wishlist for the future of BlitzMax?

Jake: I always wanted to be able to do really basic 3D in BlitzMax, so like get a texture and rotate it on the Y or X axis with perspective. I could use this for some neat effects. I'd want to do that at the same time as the normal 2D drawing if possible. Pretty much that's the only big thing I want because it would be cool for my future games.

Allan: The BlitzMax Coder magazine is just starting out what would you like to see covered in it?

Jake: Tips 'n' tricks (and tutorials) would be good. Some simple yet effective tips on how to use certain Types or how to do things faster or better, or how to avoid common mistakes/pitfalls.

I'd also like to see interviews with other developers, and perhaps some game reviews and tool reviews (like write ups about the various 3rd party IDEs that are available).

Maybe also articles giving game design advice and advice

Interview with a 'Grey Alien'

for other industries too.

Allan: You have quite an extensive blog covering a range of topics, could you give us an overview of the topics covered and why you cover them as well as links to your best articles, please?

Jake: Well originally I wrote about game development; specifically about running my own company, staying motivated and marketing - stuff like that. I also post news about my games and general Grey Alien Games news.

Over time I've branched out more and wrote about the various personal development things that I've been getting into and books I've been reading etc. So there's a whole range of topics there, something for everyone (I have an RSS feed, so please sign up. I'm also on Twitter at https://twitter.com/greyalien).

I actually have a huge backlog of possible articles to write but just not enough time to do them!

Anyway, I recommend that people check out:

http://greyaliengames.com/blog/my-best-articles-of-2008/

which links to loads of great articles from 2008.

There were tons of good articles in 2007 too, some of which were very popular, so go back and check them out.

Plus there are more great articles in 2009 which I'll summarise at the end of the year. I really should add my best articles to the sidebar or something...

Allan: I believe you recently presented at a game seminar in Canada and you covered "10 Secrets to Designing Instantly Enjoyable and Addictive Games" would you be willing to provide us with an overview of this presentation?

Jake: Yes it was at the Game Design Expo hosted by the Vancouver Film School.

You can read more about it here:

http://greyaliengames.com/blog/vancouver-game-design-expo-presentation/

It's hard to provide an overview with going into massive detail but basically the points were things like:

1. Make an impact at the start
2. Get the player playing quickly
3. Do a great interactive tutorial
4. Reward them early and continuously
5. Don't let them fail (die/loose)
6. Polish well and give great feedback
7. Use common UI elements
8. Make sure the basic mechanic is fun and build on it to add variation
9. Provide power-ups and bonuses
10. Pace the game well and break it into segments
11. Add a meta-game
12. Provide memorable moments.

There was more but that's the gist of it.

The advice sounds obvious but many people fail to do it. We were talking from the perspective of casual games mainly but we felt that the tips actually apply to almost all game types.

Allan: Are there any trends, technologies changes in the games market place that you think will set the stage over the next couple of years?

Jake: Ha, this old chestnut.

Well the industry moves very fast and this year's buzz-phrases are old hat next year.

Last year in-game advertising was "the schiz" but then the recession hit and it's clearly not as good a model as people would have liked.

This year the hype is around social gaming and micro-transactions, and of course iPhone.

I believe social gaming and casual gaming will continue to grow to massive proportions.

We should watch Google Android keenly and keep an eye out for more and more casual games on platforms like

Interview with a 'Grey Alien'

Steam, XBLA, Wii, DS and PS3, oh and phones of course.

More and more casual games seem to be turning into point and click adventure games, which is great because I love them, and the new ones have great production values.

Also a new casual genre may emerge. I'm certainly working on something pretty new (for the casual space) that I hope will start a big trend.

Casual-Core crossover games are also becoming more popular with companies like Popcap spearheading this effort with Peggle and Plants vs Zombies.

Allan: Your most recent game Unwell Mel appears to have done very well, could you tell us more about it and its development?

Jake: Yes Unwell Mel did do very well. It got to no.2 on the BFG top 10 and stayed in the top 10 for a month. I used an existing match-3 engine I had in BlitzMax and heavily modified it to add new features.

The game is actually a conversion of an existing online title but with lots of improvements. Customers seemed to like the smooth gameplay and the comedy of the title. Your aim is to cure "Unwell Mel" by going into his body to remove Gunk, Bugs and Crud with an assortment of funny powerups.

I developed most of the game as a contractor for Big Fish Games whilst in the UK, but I finished it off when I moved to Vancouver.

It took a long time for various reasons but I'm pleased with the end result - it's my best match-3 game. Try it out.

Allan: How have you found the transition to Canada, and have you noticed a general migration of games developers due to financial/tax incentives?

Jake: At first the transition was hard. It was the winter and I was working crazy hours finishing off Unwell Mel and my kids were not adjusting that well. Also I missed England.

However, that passed and as the Spring kicked in we've grown to love living here. The summer has been awesome and I have a great house in a lovely area.

Since Grubby Games were acquired by Big Fish Games in Vancouver, it's been such great fun working in the studio.

My favourite part is probably sitting on the bean bags and brainstorming. Plus going out to lunch in Yaletown is pretty cool.

I can't say I've actually seen any other developers migrate. I know Canada has a tax advantage for developers, but it's still a lot of work to import people, and a company is only allowed to have something like 1 in 10 non-Canadians in it!

Allan: So are you working on your next project and if so when can we expect to see it?

Jake: My next project is well underway. We've got a crazy big design doc that I'm slimming down to be more sensible and we've established the art style (it looks amazing).

I've also made a prototype and we've got several finished polished UI screens and an animated intro (gotta keep the artists busy).

So it's going well but it won't be out until early 2010 unless I do fantastically well and get it done before Christmas.

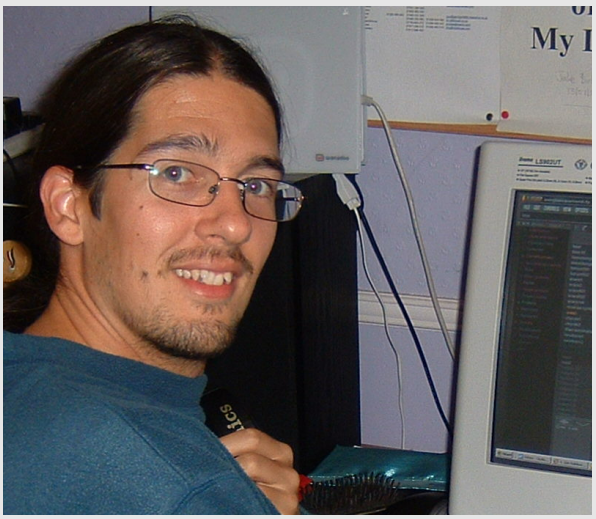
I believe it's going to be a really great game, different and an experience - so please keep an eye out for it.

I'll post it on the Blitz Forums and my blog as per usual.

Projects: Unwell Mel, The Wondeful Wizard of OZ, Holiday Bonus, BlitzMax Game Framework

Developer: Jake Birkett

Website: greyaliengames.com



Postmortem of Super Lazer Racers

Our first Postmortem is by Simon Read and covers his vector based racing game SLR...

Bio: Simon Read

I'm 33 years old and remember those heady days of swapping cassette tapes in the playground with my mates.

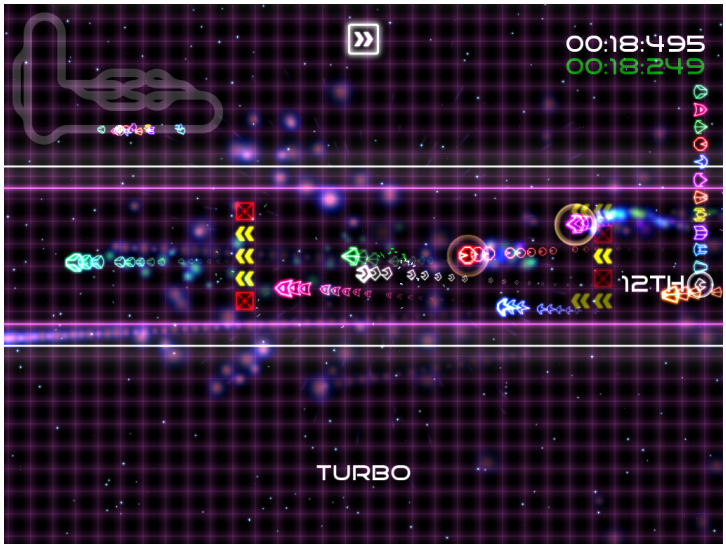
I was passionate about football and used to spend hours as a kid writing my own manager games and dreaming of fat royalty cheques arriving in the post, but I never got around to finishing anything.

I pretty much gave up on programming during my mid-teens and it wasn't until after university that I got back into it.

I started reading C++ manuals and wrote World Cup Manager learning as I went along.

I set up my own website and made a few hundred bucks but it was New Star Soccer that people started following with interest. The first games were all text based but by NSS3 I picked up BlitzPlus and added a full 2D match engine.

The success of that game allowed me to become a full time indie.



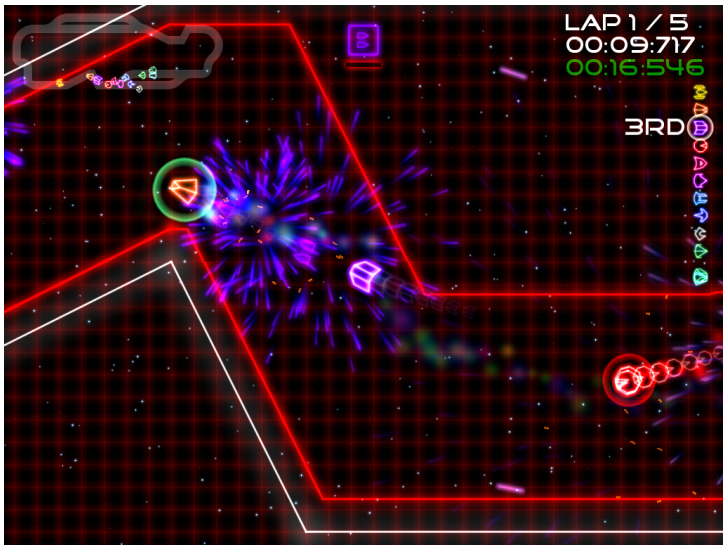
I have since moved on to BlitzMax and have used it to write New Star Soccer 4, New Star Grand Prix and Super Laser Racer.

Overview

After completing my previous game New Star GP I was considering replacing the racing cars with space ships and putting weapons in the game. It seemed like a fairly simple way to reuse the code that I'd spent the previous 3 months working on and quickly create an entirely new game.

Meanwhile I had been playing PuppyGames' Gravitron which I'd purchased through Steam and was mulling over the prospect of making a similar retro style game. It suddenly hit me that I could create the combat racing game with a neon vector look. Throw in some cool explosions and a techno sound-track and hey-presto, I'd have Geometry Wars meets Wipeout.

Like most of my projects, SLR was already perfectly formed in my head the moment I conceived it and required little in the way of design documents or planning.



This is perhaps my biggest fault as a designer as occasionally I have to rethink a certain aspect of the game and waste time re-coding it. However, this doesn't happen too often, if at all with SLR, and it never deters me from ploughing on with the creation of the game with nary a thought for planning. (Oh the joy of being an indie!)

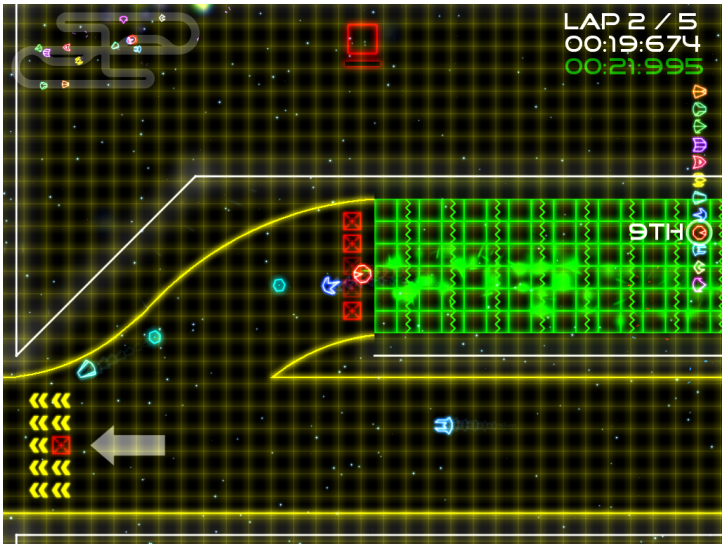
I don't deny that preparation is key for some projects but I find that the sheer joy of programming is what keeps me going each day and if I make some mistakes along the way, so be it.

I find the journey much more enjoyable than actually reaching the destination anyway.

Unlike my existing sports games I decided to stay away from a detailed career mode in Super Laser Racer and keep it strictly arcade.

Postmortem of Super Lazer Racers

One reason for this was to keep the development time down, the other being that contracts, sponsorships, shops and casinos don't really suit an abstract vector game. I also wanted to strip the game right down to the basics and concentrate on the gameplay, trying to achieve that elusive fun factor without the distraction of statistics or management.



Tools

SLR was created in BlitzMax. I love Blitz because it encapsulates everything that I love about programming. Put simply, it gets results fast. I don't mean the speed of the actual code, I mean the time it takes to get stuff on the screen and to get it flying about. When it comes to programming code I don't really care how it works, so long as it does what I want it to do.

If it's inefficient but runs ok, that will do for me and it's on to the next milestone. With that in mind it won't surprise you to learn that I'm all for using 3rd party modules if they help me get to the finish line quicker and this project was no different.

I used several BlitzMax modules, the main ones being Liam McGuigan's FryGUI (which simplifies the task of creating the interface) and a bunch of Bruce Henderson's fine mods for backend stuff like localisation and net connections.

The graphics were all done by myself using my ever trusty Paint Shop Pro X. The great thing about this project was that I didn't have to outsource any artwork. The simple neon style was all done with a few coloured lines and a bit of Gaussian blur.

The sound effects were mostly from Freesound.org and the music came from a Shockwave-Sound.com collection which at $99 was an absolute bargain.

What went right

In terms of development, everything went perfectly. Being built on existing game code meant that there were very few bugs to track down and I could do most of the play testing myself. It also kept the development time down and in the end it only took around 4-5 weeks.

The fact that I created the graphics and used royalty free sounds and music mean that the cost of developing the game was purely in man-hours.

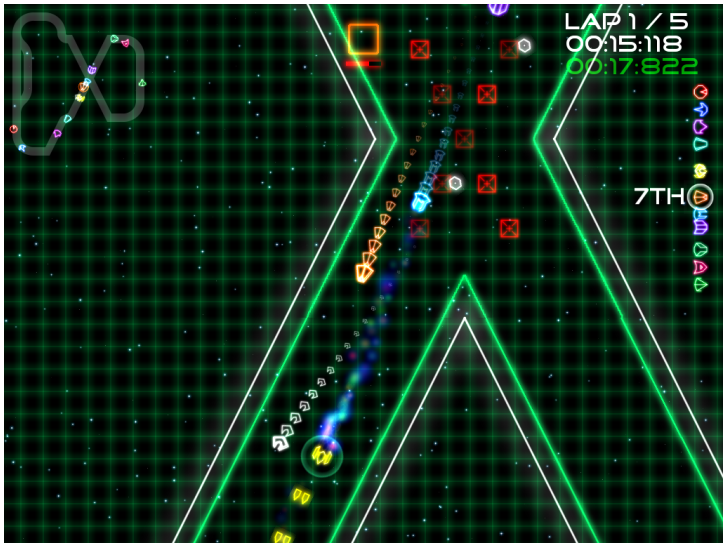
What went wrong

The main missing feature from the game is an online multi-player mode. Network coding is not a strong point for me so I felt that it might be time wasted if the game doesn't prove popular. On the other hand, the game might be more popular if it included a multi-player mode! Its absence though does mean that I have something to include in a future update or sequel.

So far sales have been a little slow but at the moment it is only available through www.newstargames.com which very much draws in soccer and racing game fans.

My mailing list is largely built on the success of New Star Soccer and as such they aren't too interested in a retro style combat racing game set in outer space!

For a project with such a short development time though it is definitely paying its way.



Conclusion

This was always an experimental project for me, breaking away from traditional sports games and reaching out to a different audience.

In that sense it is doing well as there is interest from some of the larger gaming portals which hasn't happened with my sports games in the past.

I hope to release new content in the form of new tracks and tournaments and the inbuilt editor is allowing players to share their creations via my forum which is great to see.

The initial reviews have been very favourable at around the 70-80% mark so I obviously got something right. All in all it

Postmortem of Super Lazer Racers

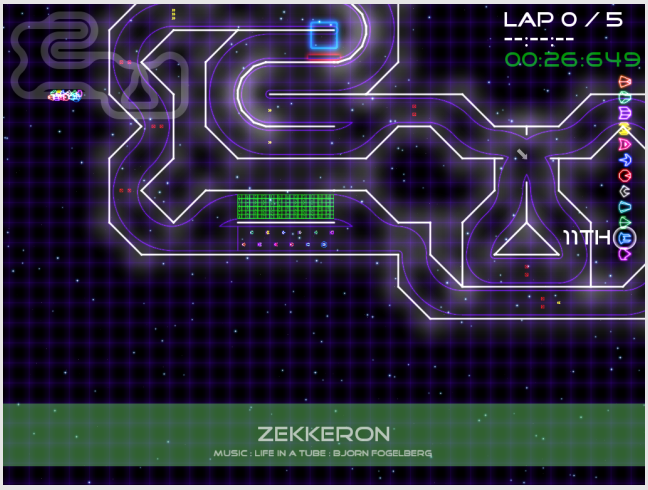
was a fun project with a quick turnaround that reminded me of the importance of creating clean re-usable code.

It was also great to get back to basics and concentrate on the fun factor.

Projects: Super Lazer Racers
New Star Grand Prix
New Star Soccer 4

Developer: Simon Read

Website: newstargames.com



Tech Overview

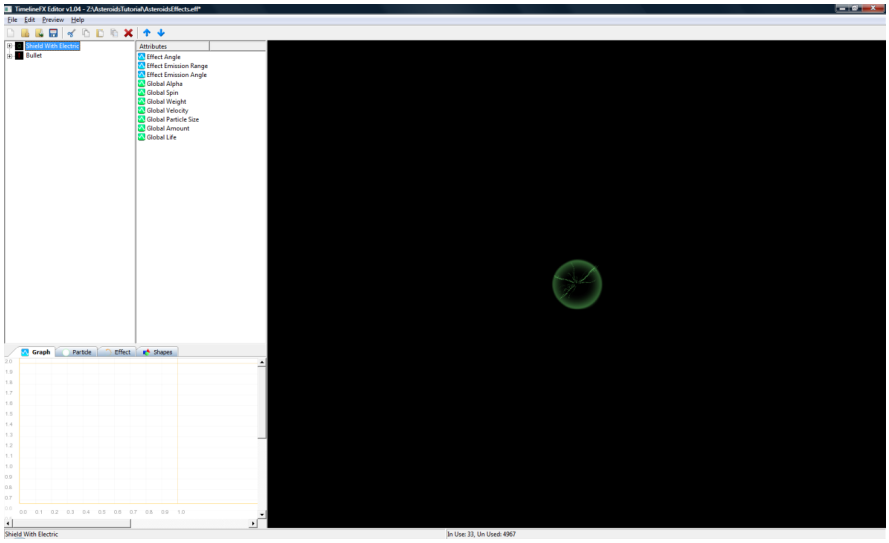
Timing:	Fixed rate logic with tweening [HardNutz Archive Code]
collision:	Circle to Circle Collision System
Physics:	2D vectors and Circle Collision response
Graphics:	Max2D OpenGL
Sound:	OpenALAudio
Input:	FreeJoy
Gui:	FryGui
Data:	Sqlite Database

Thrust or Asteroid problems try TimelineFX...

The first two amazing parts of a three part tutorial brought to you by TimelineFX creator Peter Rigby.

Part 1: Making a Thruster Effect with the TimelineFX Editor

This is part 1 of a tutorial aimed at beginners who have some grasp of Object Orientated programming, and how you can use TimelineFX to implement some effects into your games. We will make an asteroids clone and fill it with special effects using an effects library I have already made especially for shoot em ups, but I'll also take you through creating an effect in the TimelineFX Editor.



I've prepared a zip file with resources you'll need for this tutorial which you can find here:

http://www.rigzsoft.co.uk/files/AsteroidsTutorial.zip

Here's an outline of what we want to achieve:

1. Install the TimelineFX module into the Blitzmax module directory

2. Make use of the tlEntity type – a generic entity type in TimelineFX that handles a lot of the mundane things for us such as movement, animation etc. We'll make use of Blitzmax's Object Orientated features to extend this type and create a type for our player ship, and a type for our asteroids.

3. Use fixed rate timing to make our asteroids clone run smoothly and the same speed on all PCs that run it. For this we can make use of tweener.mod – A module that comes with TimelineFX

4. Use the Particle manager in TimelineFX to manage all of our particles.

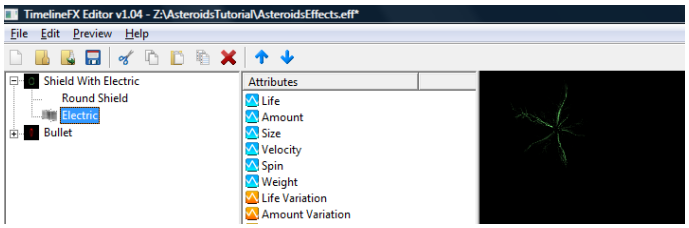
Before we do all that let's go over some features of the TimelineFX editor and create a thruster effect with it. If you don't own the TimelineFX Editor then you can download a trial version of it here:

http://www.rigzsoft.co.uk/files/TimelineFXEditorSetup.exe

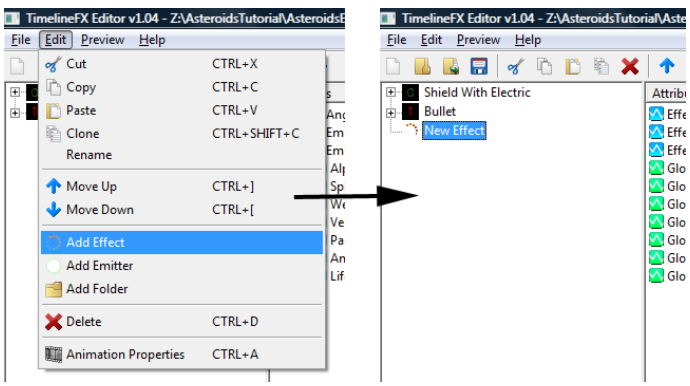
I'm sorry to say that presently the editor is Windows only, so if you're on a Mac you can just skip to the coding tutorial below.

So let's jump straight in and create an effect. Once you've installed the editor, load it up and load the AsteroidsEffects.eff which you'll find in the AsteroidsTutorial.zip. The effects file has 2 effects already created, click on the effect called “Shield with Electric”. You should now see the effect in the preview pane of the editor.

Each effect consists of emitters that can emit different particles. Double click on the “Shield with Electric” to expand the treeview and you'll see 2 emitters that make up the effect, try clicking on each one and you'll notice that that individual emitter will be the only one visible in the preview pane.



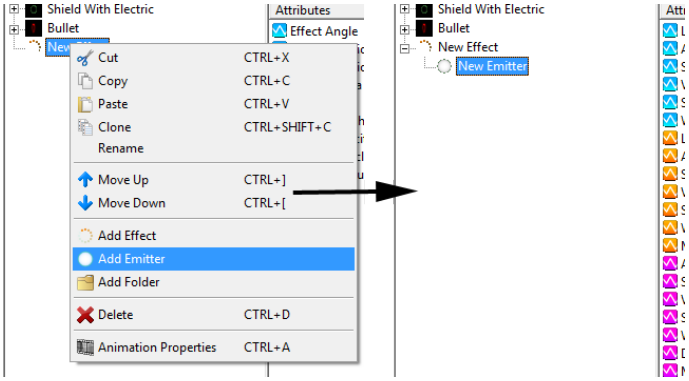
Let's dive straight in an create a new effect. I want to create a thruster effect that will be emitted from the player ship when it moves. Make sure that an effect is highlighted and then go to the **Edit** menu and select **Add Effect -** a new effect will appear in the list of effects. It's important that an effect is highlighted when you do this, otherwise an effect will be created as a sub effect if an emitter is highlighted.



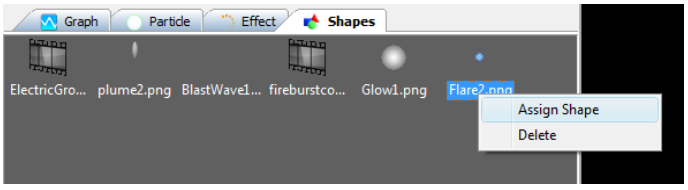
Next we need to create a new emitter within the effect, so right click on the New Effect we just created and select

Thrust or Asteroid problems try TimelineFX...

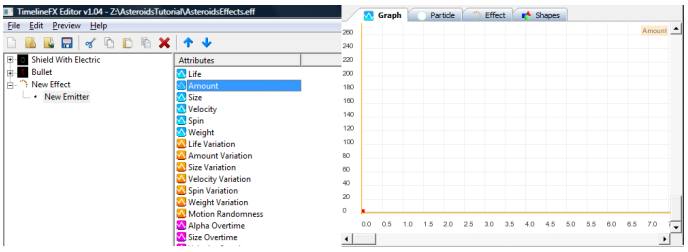
Add Emitter from the pop up menu. A new emitter will be created within the Effect.



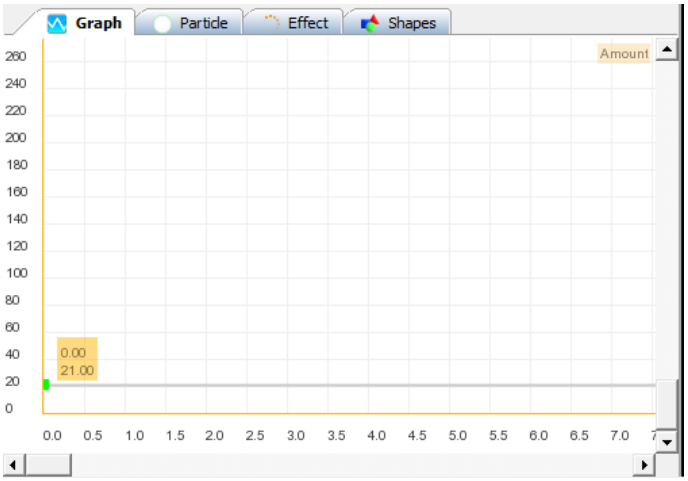
In the preview window you'll see the image particle shape, with No Image written on it, because as yet we haven't assigned a shape to the emitter, so let's do that now. In the bottom left of the editor you'll see some tabs with various different options on each one of them. Click on the **Shapes** tab and you'll see some particle shapes that are currently loaded into the effects library. To assign one of those shapes, make sure the New Emitter is selected and then right click on the shape called Flare2.png and select **Assign Shape** from the pop up menu.



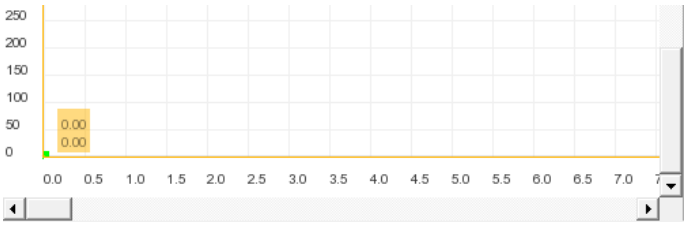
The No Image shape will now be replaced with the flare effect we just assigned. This will be the basis of a light effect produced by the thruster. At the moment the emitter is only spawning 1 particle per second, we need to increase this a bit, so lets introduce you the TimelineFX graphs. The graphs are what you use to describe how particles behave over time. Next to the list of effects and emitters is the list of attributes. Each attribute has a graph assigned to it, click on the attribute called Amount - the graph tab will then be selected for you.



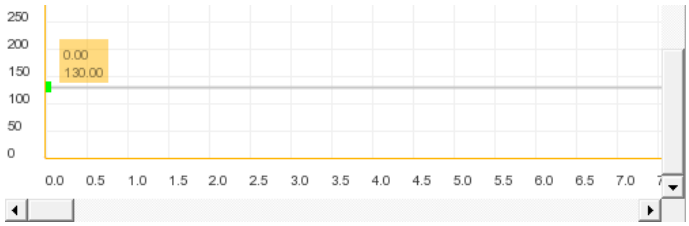
The x-axis of the graph represents the time in seconds that the effect has been running for, and the y-axis is the value represented by the attribute – in this case the it's the amount of particles spawned per second. On the graph you'll see a node represented by a red box which shows that at the moment, the attribute is set to spawn one particle every second. Lets change that, click and drag the node up to about 20 on the graph. Because it is the only node on the graph, it will remain clamped to the left hand side of the graph.



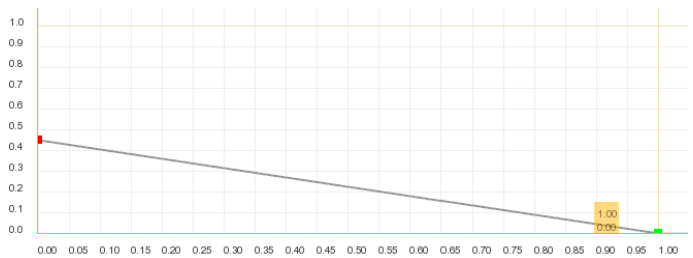
You should see straight away that the effect preview has reflected the change. What we want to do is create a flickering effect, so next, select the size attribute and drag the first node on the size graph down to 0.



The effect in the preview window will disappear as it no longer has any size. This is obviously no good, so now, select the **Size Variation** attribute and drag the node on the graph up to about 130.



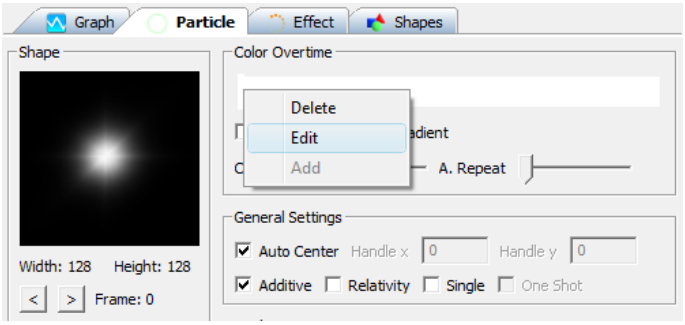
It's looking a little better but still rough around the edges! Next, select the **Alpha Overtime** attribute. Overtime attributes represent what happens over the life-time of the particle, so the x-axis represents the percent of life of the particle. We want the particle to fade out over the life of the particle, we also don't want it to be as bright as it is, so first drag the node that is on the graph down to about 0.45 and then create a new node by clicking and dragging anywhere on the graph, and drag the new node to 0,1 on the graph so it should look something like this:



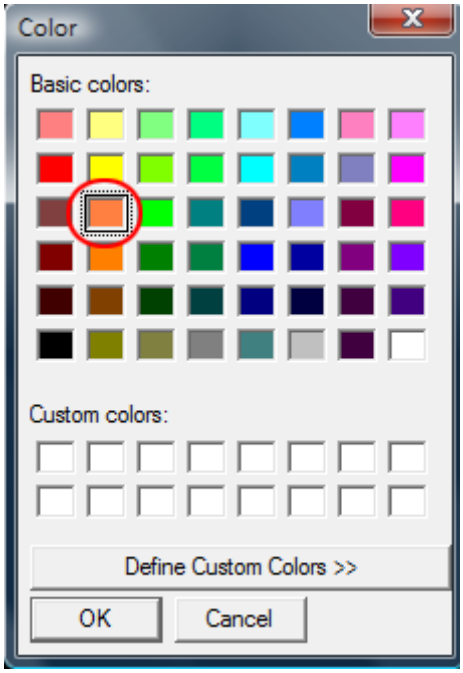
Now let's give it a bit of colour. Select the **Particle** tab where you'll see a number of options for adjusting the look of the particle. We want to change the colour, and at the top of the tab you'll see a colour bar which is currently all white. This bar represents how the colour of the particle

Thrust or Asteroid problems try TimelineFX...

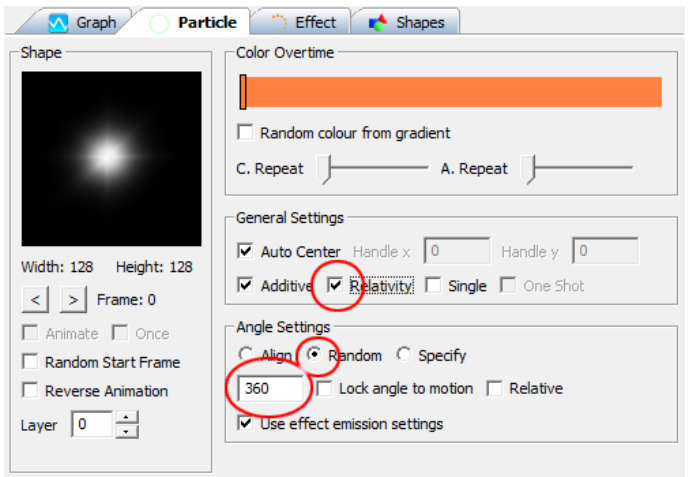
changes over time. At the beginning of the bar is a colour node, right click on that and select **Edit** from the pop up menu.



This will show a colour dialogue, select an orange colour like the one below and click OK.

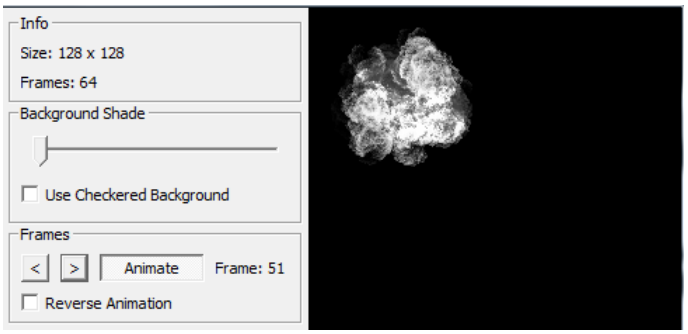


So our effect should now be an orange flickering light. To finish this emitter off, lets select a few additional options on the P**article** tab. We want to make the emitter remain relative to the effect, so to do this, check the R**elativity** check box. To see the difference, try clicking and dragging the effect about the preview window with the Relativity check box unchecked – it leaves a trail which we don't want. We also want the angle of the particles to be randomised each time they spawn, so select the **Random** radio button and in the field below it, type in 360. This means that every time a particle is spawned by the emitter, a random angle will be selected in the range 0-360.

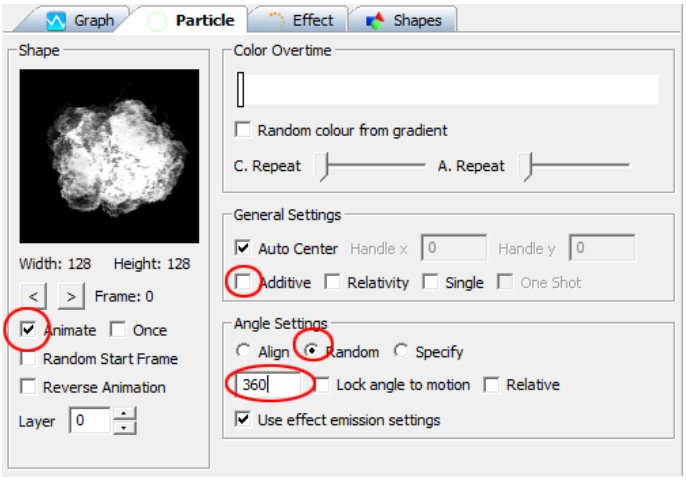


And that's the first part of this effect done! Let's quickly rename the effect and emitter we just created by right clicking on each one in the list and select R**ename** from the pop up menu. Call the effect “Thrusters” and the emitter “Flare”.

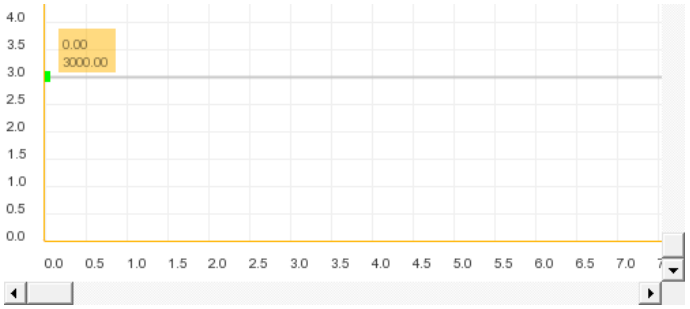
Next we want to add a fiery effect that comes out of the back of the space ship. Right click on the Newly named “Thruster” Effect and select **Add Emitter** from the pop up menu. Select our New Emitter and click on the shapes tab. We want to assign it the shape called FireBurstContinuous2.tpa which is an animated particle. The icon is a film strip showing that it's an animation, if you want to see a preview of the particle, you can double click on the shape icon, that will bring up the shape properties dialogue where you can see the particle shape animating by clicking on the animate button.



So, right click the FireBurstContinuous2.tpa shape and select assign shape. We'll start by setting some properties on the **Particle** Tab. Because it's a particle with more than 1 frame of animation we can animate it, so we'll check the **Animate** check box. Next uncheck the **Additive** check box as we want it to be more smoky then fiery, and finally check the **Random** radio button and type 360 into the the field below so the particle's angle will be randomised. Your particles tab should now look like this:

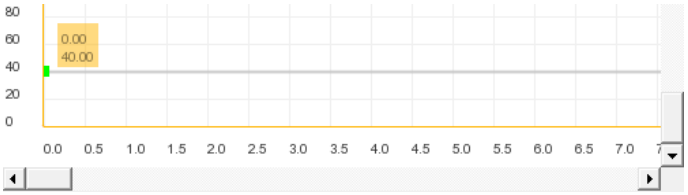


At the moment it doesn't look too good, to start with we want the particles to last a little longer, so click on the **Life** attribute and drag the node that is on the graph up to about 3, telling the emitter to spawn particles that live for 3 seconds.

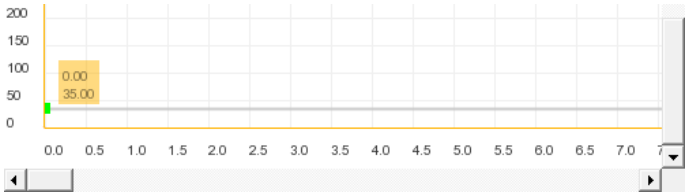


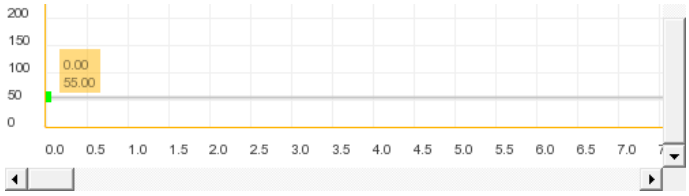
Thrust or Asteroid problems try TimelineFX...

Next select the **Amount** attribute and drag the node up to about 40

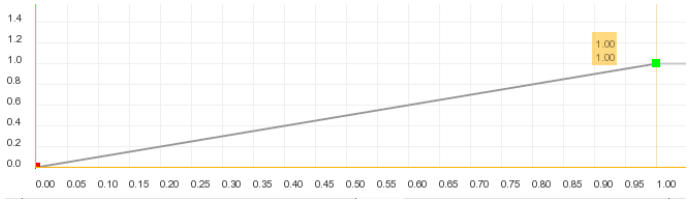


Let's sort out the size of the particle now, we want some variation but not too much, so drag the node on the **Size** attribute to about 35, and drag the node on the Size Variation attribute to about 55.

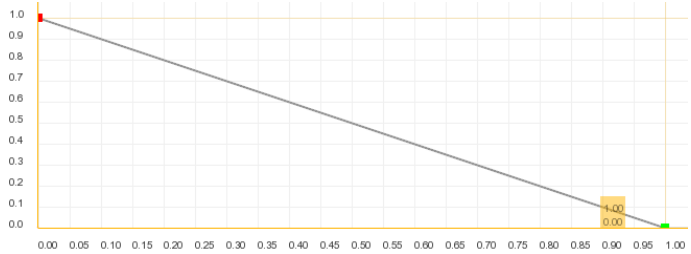




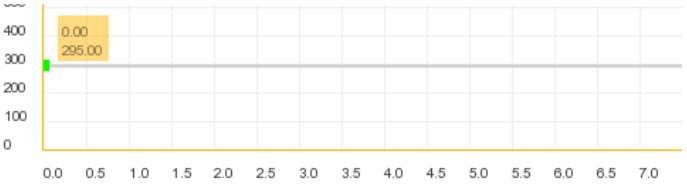
To finish the size off, we want to make it grow in size over the particle's life, so select the **Size Overtime** attribute and drag the first node down to zero, then click and drag elsewhere on the graph to create a new node and position it so that it's right over on the right hand side of the graph set at 1 on the y-axis. This means that at the end of the particle's life it would have grown to 100% of it's base size as dictated by the **Size** and **Size Variation** attributes. It's base base is calculated by taking the **Size** attribute and adding on a random value between 0 and whatever the **Size Variation** attribute is. This is what the **Size Overtime** graph should look like now:



Let's make the particle's fade out, select the **Alpha Overtime** attribute and create a new node on the graph and drag to 0,1 on the graph so it looks like this:

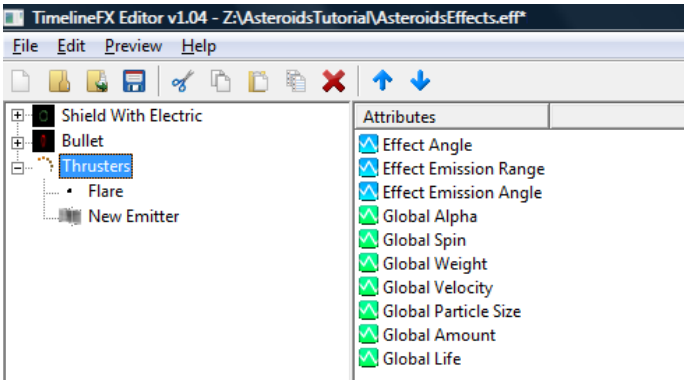


It's starting to take shape! As they're thrusters though, we need to add some velocity to the particles so it looks like they're flying out the exhaust of a ship. Select the **Velocity** attribute and set the node on the graph to about 300 (which equates to 300 pixels per second):

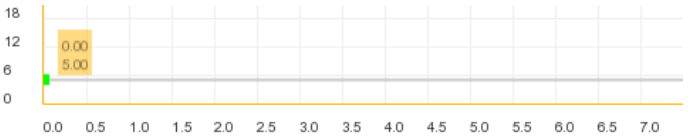


You'll notice in the preview pane that the particles haven't started move yet, this is because we need to set the **Velocity Overtime** attribute to something above 0 (its default). All overtime attributes scale their base values, so for example, if the **Velocity** attribute is set to 300, and the **Velocity Overtime** attribute is set to 0.5, then the particles will travel at 150 pixels per second. We want the velocity to start off fast but decelerate over the particles lifetime, so first select the **Velocity Overtime** attribute and drag the first node up to 1 and then create a new node and drag it over to the right to 0,1, then create a third node and place it at about 0.5,0.08 on the graph so it looks like this:

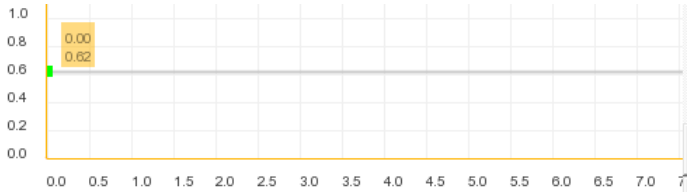
OK, we're getting there! Let's add a bit of scattering to the emission. Select the Effect of the emitter, the attribute list will change to show the global attributes of the effect:



Select the **Emission Range** attribute and drag the node on the graph up to about 5. This means that there will be a 5 degree variation to the direction that particles travel in when they spawn:

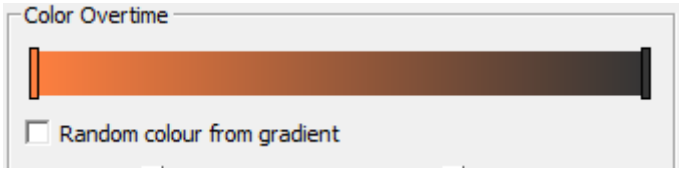


Starting to look a bit better, but I think there's a little too much 'boost' so select the Global Life attribute and drag the node there down to about 0.6. This will scale the life down of both emitters in the effect:

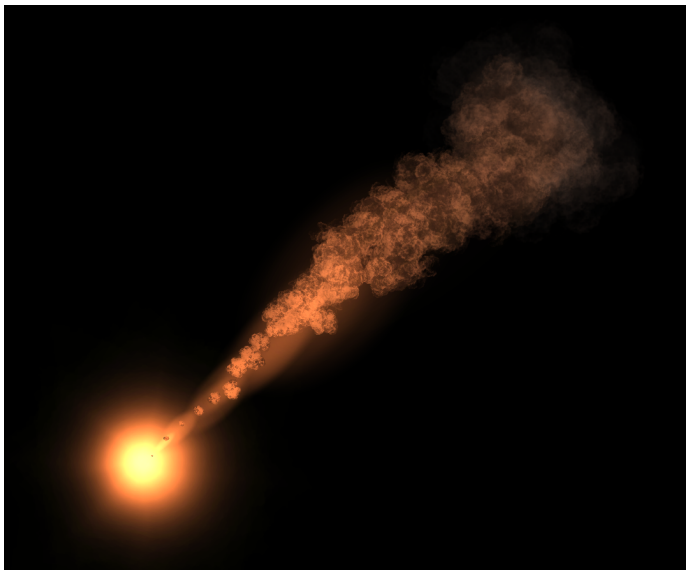


Thrust or Asteroid problems try TimelineFX...

And finally! We just need to change the colour, I want to make it an orange that fades to a dark grey. Click on the Particle tab and right click on the colour node that's there, and then select **Edit** from the pop up menu and select the same orange that we selected for the flare emitter. Click OK on the colour dialogue then right click elsewhere on the colour and select **Add,** then select a dark grey colour. Click OK again and drag the node to the end of the colour bar, so it should now look like this:



So that's our thruster effect, you can finish up by renaming the new emitter “Smoke”, if you open up AsteroidsEffectsPart2.eff then my version can be found there, I added a third emitter just to flesh it out a bit more so check it out!



Later I'll show you how to integrate that into a game and change the emission angle so that it can align with any ship that uses it. Which leads us onto the next part of this tutorial where we'll use TimelineFX to make a small Asteroids clone.

Part 2: Coding an Asteroids Clone using TimelineFX

Firstly of course, if you haven't already, you'll have to install TimelineFX, and you already have it installed then you might want to check that you have the latest version. If you've never installed a module before, don't worry it's a fairly painless procedure. Firstly you need to download the module, a direct link to that can be found here:

http://www.rigzsoft.co.uk/files/rigz.mod.zip

Once you have that you need to unzip it into your Blitzmax mod directory, this is found directly under the Blitzmax directory, e.g., Blitzmax/mod.
Lastly, TimelineFX has dependencies on 3 other modules:

bah.libxml by Brucey - This module is for all your XML needs in Blitz. XML is the format of Effects files.
gman.zipengine by Gman - A superb mod for handling zip files in Blitzmax. Effects libraries are stored in zip files.
dbs.d3d9max2d by Dstastny - The excellent mod that gives you the option to use DirectX 9 for rendering.

If you don't have these mods already then you'll need to install them too, and I've got them all zipped up and ready to download from:

http://www.rigzsoft.co.uk/files/extramods.zip

to make life easier. Once again, simply unzip into your Blitzmax/mod folder. Once you're done your mod folder folder should look like this:

```
Blitzmax/
    mod/
        bah.mod
        brl.mod
        dbs.mod
        gman.mod
        pub.mod
        rigz.mod
```

If you're on windows, then you're all ready to go, if you're on a Mac or Linux then you will have to compile the modules for your OS. To do this you can use makemods - at a command line in the Blitzmax/bin folder type:

bmk makemods -a rigz.mod
This will compile the module for your OS. Do the same for the other mods:

```
bmk makemods -a rigz
bmk makemods -a bah
bmk makemods -a dbs
bmk makemods -a gman
```

And with that you should be all set. Lets get started with our Asteroids clone!

Before we do any coding, we need some graphics for our player spaceship. Because I'm nice, I made one so you don't have to bother, because I'm rubbish at art, it's not particularly good! But feel free to draw your own. You'll find it in the AsteroidsTutorial.zip zip file at the start of this turorial.

So, start by creating a new file in your favourite Blitzmax IDE and then lets tell Blitzmax which modules we want to import:

```
SuperStrict
Framework rigz.timelinefx
Import rigz.tweener
```

If you're unfamiliar with framework, it simply a way of telling Blitzmax that you want to use that module as the main framework for our game. If you don't use framework then

Thrust or Asteroid problems try TimelineFX...

Blitzmax will import all modules by default leading to larger executable files. For this tutorial, that's all we need to import for now, just TimelineFX and tweener.
Now lets quickly create our first type:

```
Type tAsteroidsGame
	'Our particle manager
	Field ParticleManager:tlParticleManager
	'And a tweener for the game timing
	Field Tweener:tTweener
End Type
```

This is the type that will store some of the main game variables and the main game loop. 2 such variables that we will need is one to store our particle manager that will manage all the particles, and **Tweener**, which we will use to set up our timing code – more on that later.

Next we're going to jump straight into some of the object orientated features of Blitzmax. When I create types I like to normalise the fields as much as possible, similar to how you would normalise a database. What this means is that you remove duplicate fields from different types and put them into a main type that you can extend. Take this for an example:

```
Type tPlayer
	Field x:Float
	Field y:Float
End Type
Type tAsteroid
	Field x:Float
	Field y:Float
End Type
```

Both tPlayer and tAsteroid need to have fields that contain their location: x and y. But there's really no need to duplicate the field declarations like this. Instead you can create a base type and extend it like so:

```
Type tEntity
	Field x:Float
	Field y:Float
End Type
Type tPlayer Extends tEntity
End Type
Type tAsteroid Extends tEntity
End Type
```

In this case, tPlayer and tAsteroid inherit the fields of tEntity. In 'OO' terms this is called inheritance. So what's the point of me telling you this? Well, I want to create a type that inherits all the fields and methods of tlEntity, the main entity type of TimelineFX. tlEntity contains all sorts of fields for storing location, rotation, size, colour and more – all the general things you need for maintaining game objects. So, our first type for our Asteroids clone will be tAsteroidsEntity:

```
Type tAsteroidsEntity Extends tlEntity
	Field game:tAsteroidsGame
End Type
```

This is our basic game entity. Whilst it looks basic, remember that it contains all the goodies that come with tlEntity. Currently the only field it has is game, which will store a link to our tAsteroidsGame type giving each entity access to important game variables. Next we will create a type to store our player data:

```
Type tPlayer Extends tAsteroidsEntity
	'the current amount of shields the player has
	'measured in milliseconds
	Field shieldlevel:Float = 5000
	'A flag to store whether the shield is active
	field shieldactive:int
	'The rate of fire, starting off at 5 per second
	Field rate_of_fire:Float = 5
	'so we can keep track of the fire rate:
	Field fire_counter:Float
	'store the rate of turn of the player in degrees
per second
	Field rateofturn:Float = 150
	'rate of acceleration in pixels per second
	Field rateofaccelleration:Float = 10
	'rate of deceleration
	Field rateofdeccelleration:Float = 1.05
	'store the key controls
	Field turnleftkey:Int = KEY_LEFT
	Field turnrightkey:Int = KEY_RIGHT
	Field shootkey:Int = KEY_Z
	Field shieldkey:Int = KEY_X
	Field movekey:Int = KEY_UP
	'Store the speed in each direction so we can use
inertia when the ship moves
	Field xspeed:Float
	Field yspeed:Float
End Type
```

This will extend the tAsteroidsEntity so we get the field **game**, on top of all the **tlEntity** stuff. So why not just put the field **game** into this type and just do away with **tAsteroidsEntity**? The 2 main reasons are that I also want to create a **tAsteroid** type for looking after the asteroids that will be floating about, this type will also need a link to the game type so we will avoid duplicating the field in 2 different types which I think is a messy way of doing things. The other reason, is so that if we want to create fields that need to be used by tPlayer and tAsteroid in the future, we only need to add them to tAsteroidsEntity for them to have them also. The same goes for any methods too.

Now, let's make a create function for tPlayer, pop this in the tPlayer type:

```
Function Create:tPlayer(x:Float, y:Float,
game:tAsteroidsGame)
	Local player:tPlayer = New tPlayer
	player.game = game
	player.x = x
	player.y = y
	player.SetUpdateSpeed(false)
	Return player
End Function
```

The other thing we need to create before this type is of any use is an update method. To do this we will make use of

Thrust or Asteroid problems try TimelineFX...

another OO feature called polymorphism. It sounds a lot more fancy then it is! t**lEntity** has an Update method which updates the entities position, speed, animation etc. basically we will override the update method in **tlEntity** by creating one in tPlayer. However, we still want to run the update method in tlEntity to cover the mundane stuff of updating the entity position and such, so to do that we will use super.update. Super refers to the main type that we're extending from. The other thing to note is the line "**player.SetUpdateSpeed(false)**", this tells the entity not not update the speed calculations when we call **super.update** because we want the player ship to calculate it in a different way, so we can apply acceleration and inertia for our player ship when it moves. To help with readability I'm going to use "Self" a lot. Self basically refers to the object itself, so it will show more clearly where we are accessing the fields and methods contained within **tPlayer** and the types that we are extending – **tAsteroidsEntity** and **tlEntity**. Here is the update method:

```
Method Update()
	'Capture the player coordinates for tweening
	Self.capture()
	If KeyDown(Self.turnleftkey)
		'Turn left when the left key is pressed
		Self.Rotate(-Self.RateOfTurn /
Self.game.tweener.UpdateFrequency)
	End If
	If KeyDown(Self.turnrightkey)
		'Turn right when the right key is pressed
		Self.Rotate(Self.RateOfTurn /
Self.game.tweener.UpdateFrequency)
	End If
	If KeyDown(Self.movekey)
		'accelerate the player ship if the key is
pressed
		Self.setspeed(Self.speed +
Self.rateofaccelleration /
Self.game.Tweener.UpdateFrequency)
		'set the direction to the current angle of
the ship
		Self.SetEntityDirection(Self.GetAngle())
		'calculate the current pixelspersecond
		'pixelspersecond is a field of tlEntity to
temporarily store the speed of the entity
		'measured in pixels per second
		Self.pixelspersecond = Self.speed /
Self.game.tweener.UpdateFrequency
		'Calculate the x and y speeds
		Self.xspeed:+Sin(Self.direction) *
Self.pixelspersecond
		Self.yspeed:-Cos(Self.direction) *
Self.pixelspersecond
	Else
		Self.setspeed(0)
	End If
	'move the ship
	Self.move(Self.xspeed, Self.yspeed)
	Super.Update()
End Method
```

OK, so that's the basics of our update type. We are basically telling the player ship what to do when you press the movement keys. You can see that we're making use of a few of the methods that come with tlEntity such as **Capture**, **Rotate, SetSpeed** and **GetSpeed, t**hese basically do what they say on the tin, so-to-speak, but feel free to hit F1 on each keyword to find out more about what each one does. **Capture** is important for the timing code. What it does is capture the current coordinates of the ship, so that when they change, the timing code can interpolate between the old and new positions to smooth things out. You'll also notice that we're already making use of the **game** field to access the tweener so we can find out the **UpdateFrequency** the game is running at. We need to do this to calculate how much the player should rotate by and accelerate by. For example, if the the **rateofturn** is 30, then every second that the left key is held down the player should rotate 30 degrees, so we divide **rateofturn** by **UpdateFrequency** and rotate by that.

At the end of the method we call **Super.Update()** as mentioned earlier so that tlEntity can finish of the rest of the mundane updates for us.

OK, that's the basic set-up for our player type. Now lets go back to the tAsteroidsGame type and add a create method and a method to handle our main game loop. There's a few things we need to do with the create method:

1. Load in the effects library containing all the effects we want to use for our asteroids game. We can use AsteroidsEffectsPart2.eff found in the AsteroidsTutorial.zip.

2. Load the player sprite.

3. Create a particle manager, which will do all the hard work of maintaining all active particle effects.

4. Create a tweener for the game timing code

5. Create a new tPlayer

Once we have a create method we'll also need a method to run the game. When the game is running we need a loop to do the following tasks:

1. Include the timing code to run the game at a fixed rate and so we can use render tweening to smooth out all of the sprite and particle movements.

2. Update the player

3. Update the particle manager.

4. Draw the particles.

5. Clear and flip the screen.

With those things in mind here is the code:

```
Type tAsteroidsGame
	'Our particle manager
	Field ParticleManager:tlParticleManager
	'And a tweener for the game timing
	Field Tweener:tTweener
	'Our effects library containing all the effects:
```

Thrust or Asteriod problems try TimelineFX...

```
	Field EffectsLibrary:tlEffectsLibrary
	'Field to store the player entity
	Field Player:tPlayer
	Function Create:tAsteroidsGame()
		'Set the graphics
		'I use openGL here but dx7 and 9 work too
(assuming you're on windows).
		SetGraphicsDriver GLMax2DDriver()
		Graphics 800, 600
		Local game:tAsteroidsGame = New
tAsteroidsGame
		'Create a particle manager
		game.ParticleManager =
CreateParticleManager()
		'Orient the particles so that screen coords
are what we use to position stuff
	game.ParticleManager.SetScreenSize(GraphicsWidth()
, GraphicsHeight())
	game.ParticleManager.SetOrigin(GraphicsWidth() /
2, GraphicsHeight() / 2)
		'create a tweener for the fixed rate timing
		'We pass the value 30 to make the game logic
update
		'30 times per second
		game.tweener = New tTweener.Create(30)
		'We also need to tell TimelineFX how many
times per second
		'to update using SetUpdateFrequency,
although the default
		'is 30 we'll set it anyway:
		SetUpdateFrequency(30)
		'load in the effects library
		game.EffectsLibrary =
LoadEffects(                          )
		'Create the player
		game.Player = tPlayer.Create(GraphicsWidth()
/ 2, GraphicsHeight() / 2, game)
		'load the player sprite
	game.Player.SetSprite(LoadSprite(
)
		'scale it down a bit so it's a bit smaller
		game.Player.SetEntityScale(0.25, 0.25)
		'Capture the player coordinates and other
data so that
		'any initial tweening is correct. Otherwise,
when the player
		'is created it will zoom across the screen
as the tweening
		'code interpolates it between 0,0 and
wherever it spawns.
		game.Player.Capture()
		Return game
	End Function
	Method update()
		While Not KeyDown(KEY_ESCAPE)
			Cls
			'here is the timing code, update the
tweener to get
			'the number of ticks for this loop
			Self.Tweener.Update()
			For Local Ticks:Int = 1 To
Self.Tweener.FrameTicks
				'Update the execution time for
the tweener
	Self.Tweener.UpdateExecutionTime()
				'Update the player
				Self.player.Update()
				'Update the Particle Manager
				Self.particlemanager.Update()
			Next
			'draw the player
		Self.player.Render(Self.tweener.Tween)
			'Draw the particles
		Self.particlemanager.DrawParticles(Self.tweener.
Tween)
			Flip 0
		WEnd
	End Method
	Method Run()
		Self.Update()
		End
	End Method
End Type
```

So this is the main game type. This type is designed to contained all of the variables that we'd like our game objects, such as the player ship and asteroids, access to.

The other alternative is to use global variables and functions outside of any type structure, but for me, I find this way a little neater and easier to manage. I've created a **Run** method which at the moment doesn't do much except call the update method, but in the future if we add a title screen, it makes thing a bit easier. For example, you might make an UpdateTitleScreen method, and then in the **Run** method have a loop, so that when the game is finished, it loops back round and opens the title screen again.

You can test how things stand by adding the following to create a game and run it:

```
Local Asteroids:tAsteroidsGame =
tAsteroidsGame.Create()
Asteroids.Run()
```

We now have a ship that you can fly about the screen!

Before I add the ability to shoot bullets and start making use of TimelineFX let's make it so the ship wraps around if you fly off the screen, pop this into the **tPlayer update** method just before the **super.update** line:

```
Local needcapture:Int
'Wrap the ship around if it goes off the edge of the
screen
If Self.GetX() < Self.Collision_xmin
	Self.SetX(GraphicsWidth() + Self.Collision_xmax)
	needcapture = True
ElseIf Self.GetX() > GraphicsWidth() +
Self.Collision_xmax
	Self.SetX(Self.Collision_xmin)
	needcapture = True
End If
If Self.GetY() < Self.Collision_ymin
	Self.SetY (GraphicsHeight() +
Self.Collision_ymax)
	needcapture = True
ElseIf Self.GetY() > GraphicsHeight() +
Self.Collision_ymax
	Self.SetY(Self.Collision_ymin)
	needcapture = True
End If
```

Thrust or Asteriod problems try TimelineFX...

tlEntity automatically works out the size of the bounding box of the entity and stores the information in Collision_xmin, Collision_ymin for the points in the upper left hand corner of the entity, and collision_xmax, collision_ymax for the lower right corner, so we can use those to work out exactly when the player leaves the screen. You'll also notice that I created a local variable called **needcapture**, which we set to true if the ship has wrapped around the screen. This is important so that when the ship does wrap, it will not be tweened from one side of the screen to the other. So we also need to add the following piece of code *after* **super.update:**

```
'capture the ship coordinates if it has wrapped around
the screen:
If needcapture
      Self.CaptureAll()
End If
```

This time we don't use **Capture,** we use **CaptureAll** instead, the difference being that **CaptureAll** will capture any children of the player ship as well, this will be useful later on when we add effects as children of the ship such as a shield effect – more on that in a moment.

Try running it without CaptureAll() and you'll see what I mean, the ship will flicker from one side to the other.

OK, so I think it's about time we implemented some particle effects! First we need a bullet type to handle our bullets. Again we will extend tAsteroidEntity and make a create function and an update method similar to what we did for the player. Something else we will do this time however, is make use of the parent/child structure of tlEntity. Basically every tlEntity we create can have its own set of children, and those children can be drawn relative to their parent. So what we will do is add a particle effect as a child of our bullet type so that it will remain relative to the bullets – all the positioning of the effect will be handled for us automatically. Here is the type:

```
Type tBullet Extends tAsteroidsEntity
      'The amount of damage the bullet does
      Field damage:Int
      'the speed of the bullet in pixels per second
      Field bullet_speed:Float = 1000
      Function Create:tBullet(x:Float, y:Float,
game:tAsteroidsGame, e:tlEffect, Life:Int)
            Local bullet:tBullet = New tBullet
            bullet.SetX(x)
            bullet.SetY(y)
            bullet.game = game
            'set the length of time the bullet should
last for
            bullet.Setlifetime(life)
            'add the effect as a child of this bullet
type
            bullet.AddChild(CopyEffect(e,
game.ParticleManager))
            Return bullet
      End Function
      Method update()
            Self.Capture()
            'set the current speed of the bullet
            Self.setspeed(bullet_speed)
            'increase the age of the bullet, by the
current update time
            'update time is the updatefrequecy/1000
      Self.Decay(Self.game.Tweener.GetUpdateTime())
            'Destroy the bullet if it's passed its
lifetime
            If Self.GetAge() > Self.Getlifetime()
                  Self.Destroy()
                  Return
            End If
            Local needcapture:Int
            'Wrap the ship around if it goes off the
edge of the screen
            If Self.GetX() < Self.AABB_xmin
                  Self.SetX(GraphicsWidth() +
Self.AABB_xmax)
                  needcapture = True
            ElseIf Self.GetX() > GraphicsWidth() +
Self.AABB_xmax
                  Self.SetX(Self.AABB_xmin)
                  needcapture = True
            End If
            If Self.GetY() < Self.AABB_ymin
                  Self.SetY (GraphicsHeight() +
Self.AABB_ymax)
                  needcapture = True
            ElseIf Self.GetY() > GraphicsHeight() +
Self.AABB_ymax
                  Self.SetY(Self.AABB_xmin)
                  needcapture = True
            End If
            Super.update()
            'capture the bullet coordinates if it has
wrapped around the screen:
            If needcapture
                  Self.CaptureAll()
            End If
      End Method
  End Type
```

So, the create function is similar to the one in **tPlayer**, except this time we also pass a **tlEffect**. **TlEffect** is a type in TimelineFX that basically stores our effect. You can see that in the function we add it as a child using the **AddChild** command, it's also very import that when we do so we use **CopyEffect,** so that what's added is it's own unique copy. You also need to pass the Particle Manager that the effect is managed by, so we pass the one in our tAsteroidsGame type. We also set the lifetime of the bullet, this is how long the bullet will last for before it expires, we'll keep track of that in the update method.

For the **update** method, we set the speed of the bullet to the bullet_speed field we have created. As in the **tPlayer** type we call **super.update.** This is even more important here because that will also update all the children of the entity i.e., our bullet effect. We also use another one of **tlEntity's** methods called **Decay** which will age the bullet by a given amount, in this case we want to age it by the current update time of the tweener so that it will age in milliseconds. Then we check to see if its age is older then the lifetime and if so, destroy the bullet.

Thrust or Asteriod problems try TimelineFX...

Similar to the **tPlayer** update method we wrap the bullet around the screen if it goes off the edge, however, this time we make use of a different set of variables to work out when the bullet has left the screen. This time we use the AABB variables which stands for Axis Aligned Bounding Box, these variables take into account any children the entity has – in this case the bullet effect. These variables will always represent the bounding box that encompasses the entity and all of its children.
Next lets add a **tList** to the **tAsteroidsGame** type, to contain all the bullets fired from the player ship. Add this to the list of field decelerations:

```
'A list to store all of the player bullets
Field playerbullets:TList = CreateList()
```

And also we need a new method to update the bullets each logic update:

```
'method to update the player bullets
Method updateplayerbullets()
      For Local b:tBullet = EachIn playerbullets
            'if the bullet has not been destroyed then
update it
            'Otherwise remove it from the playerbullet
list
            If Not b.destroyed
                  b.update
            Else
                  b.Destroy()
                  playerbullets.Remove(b)
            End If
      Next
End Method
```

There is a variable in tlEntity called **Destroyed**, which is set to true if the method **Destroy()** has been called, such as when the bullet expires. This lets you check whether it should be removed from any list you have, as we do above. Finally, we need to call this method in the **update** method of **tAsteroidsGame.** So the logic update **for..next** loop should look like this:

```
For Local Ticks:Int = 1 To Self.Tweener.FrameTicks
      'Update the execution time for the tweener
      Self.Tweener.UpdateExecutionTime()
      'Update the player
      Self.player.Update()
      'Update the player bullets
      Self.updateplayerbullets()
      'Update the Particle Manager
      Self.particlemanager.Update()
Next
```

Of course, this isn't going to do much until we've actually done something in the **tPlayer** type to make it check to see if the player is shooting. So first, let's go to **tPlayer** and add a new field to store the actual bullet effect:

```
'The bullet effect for the player
Field bulleteffect:tlEffect
```

Then lets quickly go back to the **tAsteroidsGame** type and assign the bullet effect just after we create the player:

```
'Assign the bullet effect we want to the player
bulleteffect field
game.Player.bulleteffect =
game.EffectsLibrary.GetEffect(          )
```

Here we grab the bullet effect from the library using **GetEffect** and passing the name of the bullet effect in the library.

Now that's done, we can add the code to actually create the bullet when the player presses the shoot key. In **tPlayer** type add after the other key controls:

```
If KeyDown(Self.shootkey)
      'Rate of fire is measured in shots per second so
divide it by the update frequency
      Self.fire_counter:+rate_of_fire /
Self.game.tweener.UpdateFrequency
      'As soon as the fire counter is 1 or above a
shot is due
      If Self.fire_counter >= 1
            'we want to leave any remainder on the
fire_counter so just -1 off the variable
            Self.fire_counter:-1
            'create the bullet, and pass it the
bulleteffect we're using
            Local b:tBullet = New
tBullet.Create(Self.GetX(), Self.GetY(), Self.game,
Self.bulleteffect, 1000)
            'Set the angle and direction of the bullet
to the angle of the player ship
            b.SetEntityDirection(Self.GetAngle())
            b.SetAngle(Self.GetAngle())
            'and add it to the list of bullets in the
game object
            game.playerbullets.AddLast b
      End If
End If
```

So all that's happening here is we're counting up the fire_counter until it reaches 1 and then firing a bullet. Once the bullet is created we add it to the tAsteroidsGame bullet list after setting the direction and angle to that of the player to make sure it's aligned properly.

So if you run that then you should now be able to hold down the “Z” key to fire bullets.

Next, lets create the energy shield for the player, so that when he shield key is held down, the player is protected from asteroids. Let's add a new field to the **tPlayer** type to store the shield effect, and another one to store a copy of the effect. We'll need this as a link to the effect so that when the shields are inactive we can switch off the shield effect:

```
'The shield effect for the player
Field shieldeffect:tlEffect
'temporary field to store a copy of the shield effect
Field shieldeffectcopy:tlEffect
```

Then we can assign the shield effect to the **tPlayer** type when the player is created in the **tAsteroidsGame create** function:

Thrust or Asteriod problems try TimelineFX...

```
'Assign the shield effect to the player shieldeffect
field
game.Player.shieldeffect =
game.EffectsLibrary.GetEffect(                    )
```

Now we can add the condition for when the player presses the shieldkey in the tPlayer update method:

```
If KeyDown(Self.shieldkey) And Self.shieldlevel > 0
      'lower the shieldlevel all the time the shield key
is press down
      Self.shieldlevel:-
Self.game.Tweener.GetUpdateTime()
      'copy the shield effect and add it as a child of
the player
      'but only if the last shieldactive state was
false, we don't want to keep
      'adding it over and over!
      If Not Self.shieldactive
            'copy the shield effect to the
shieldeffectcopy field
            Self.shieldeffectcopy =
CopyEffect(Self.shieldeffect, Self.game.ParticleManager)
            'and change the size of the effect by
changing the global particle size of the effect
            'This scales to size of all the particles in
the effect
      Self.shieldeffectcopy.SetEffectParticleSize(0.75,
0.75)
            'add it as a child of the player so it stays
relative
            Self.AddChild(Self.shieldeffectcopy)
      End If
      Self.shieldactive = True
Else
      'kill the shield effect if it exists
      If Self.shieldeffectcopy
            'using softkill will stop the effect from
spawning any new particles and let
            'the current one expire naturally. You can
use hardkill to kill an effect instantly
            Self.shieldeffectcopy.softkill()
            Self.shieldeffectcopy = Null
      End If
      Self.shieldactive = False
End If
```

So basically, when the player presses the “X” key a shield effect will appear around the player. We're actually storing a copy of the effect in a field this time, so that when the key is no longer being pressed we can destroy the effect without having to look through the children of player to do the same thing. Once the effect has finished and no longer has any particles it will automatically tidy itself up and remove itself from the players list of children.

OK, so now I want to add one more effect for the player to finish things off – some thrusters! So another 2 fields are needed for the tPlayer type, just like the shield effect, and also add a thrusteractive field:

```
'The thruster effect for the player
Field thrustereffect:tlEffect
'temporary field to store a copy of the shield effect
Field thrusters:TList
'And a field to store whether the thrusters are
currently active
Field thrusteractive:Int
```

And get the effect when the game is created, so in the **tAsteroidsGame create** method add:

```
'Assign the Thrusters effect the player thrustereffect
field
game.Player.thrustereffect =
game.EffectsLibrary.GetEffect(                    )
```

Now we have to update the **update** method of tPlayer so that the thruster effect is added as a child of the space ship just as we did with the shield effect, so replace the current keydown condition for the movekey with:

```
If KeyDown(Self.movekey)
      'accelerate the player ship if the key is
pressed
      Self.setspeed(Self.speed +
Self.rateofaccelleration /
Self.game.Tweener.UpdateFrequency)
      'set the direction to the current angle of the
ship
      Self.SetEntityDirection(Self.GetAngle())
      'calculate the current pixelspersecond
      'pixelspersecond is a field of tlEntity to
temporarily store the speed of the entity
      'measured in pixels per second
      Self.pixelspersecond = Self.speed /
Self.game.tweener.UpdateFrequency
      'Calculate the x and y speeds
      Self.xspeed:+Sin(Self.direction) *
Self.pixelspersecond
      Self.yspeed:-Cos(Self.direction) *
Self.pixelspersecond
      If Not Self.thrusteractive
            'copy the shield effect to the
shieldeffectcopy field
            Self.thrustereffectcopy =
CopyEffect(Self.thrustereffect,
Self.game.ParticleManager)
            'offset the location of the thruster so
that it emits from the lower part of the player ship
            Self.thrustereffectcopy.SetY(10)
            'add it as a child of the player so it
stays relative
            Self.AddChild(Self.thrustereffectcopy)
      End If
      'set the emission angle of the thrusters to that
of the player ship
      Self.thrustereffectcopy.SetEmissionAngle(Self.an
gle - 180)
      Self.thrusteractive = True
Else
      'kill the thruster effect if it exists
      If Self.thrustereffectcopy
            'using softkill will stop the effect from
spawning any new particles and let
            'the current one expire naturally. You can
use hardkill to kill an effect instantly
            Self.thrustereffectcopy.softkill()
            Self.thrustereffectcopy = Null
      End If
      Self.thrusteractive = False
      Self.setspeed(0)
End If
```

Very similar to the shield effect earlier, except this time we are not adjusting the particle size, we are adjusting the emission angle of the thrusters to align it with the ship with the command **`SetEmissionAngle`**`(Self.angle - 180)`. We're also using **`SetY`**`(10)` to position the effect a bit lower on the player ship, otherwise it would emit from the centre of the ship which wouldn't look very good.

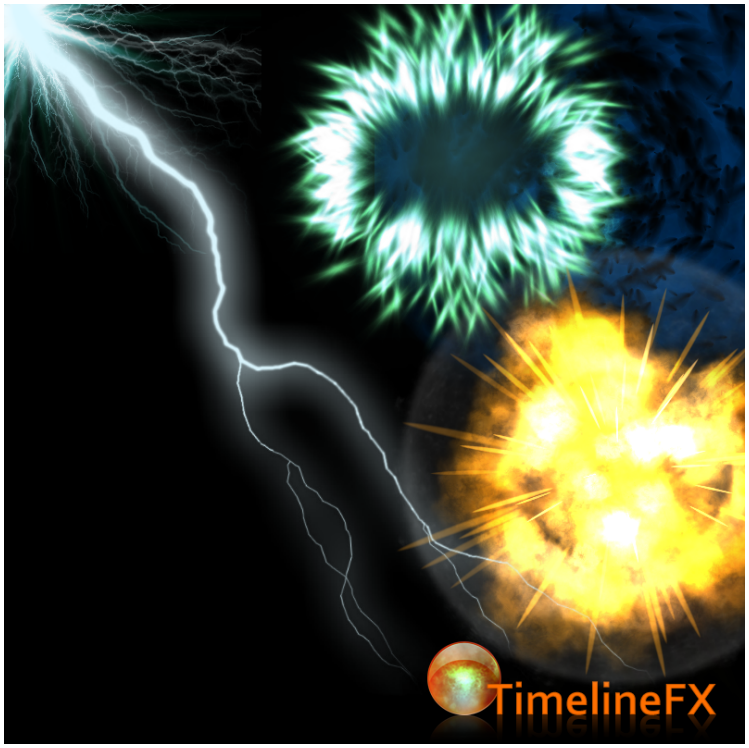
So run that and we should have a ship with thrusters, a

Thrust or Asteriod problems try TimelineFX...

shield and some bullets it can shoot!

That concludes this part of the tutorial, I hope you enjoyed it. Come back next month where we'll add some asteroids to shoot at and implement some cool looking explosion effects in part 2.

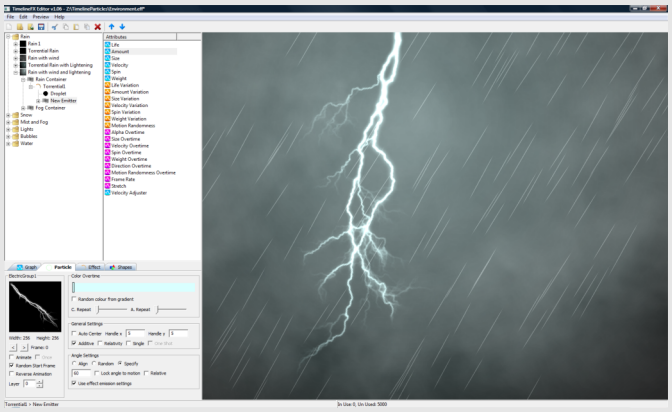
All the source code for the tutorial can be found in AsteroidsTutorial.zip.



Projects: TimelineFX

Developer: Pete Rigz

Website: www.rigzsoft.co.uk



Submit Article...

Project: Your Project
Level: Any
Author: You

Calling out to all BlitzMax Coders...

Can you fill this space with a stimulating Tutorial or 'How To' on a programming, development or design topic?

Kind Regards

Allan (aka Arowx)

submissions@blitzmaxcoder.com

Bowled over by 3D well help is at hand...

Introduction

In this tutorial I'm going to show you how to setup and use a 3d module with BlitzMax. We are going to be using Xors3d and [Phy]sics for displaying 3d graphics and adding collisions and player interactivity to create the foundation of a ten-pin bowling game.

I've picked Xors3d as it has an almost indentical command set to Blitz3d so it's very easy to use for those of you who moved onto Blitzmax from Blitz3d (like me). Even if you have no previous experience with 3d, this tutorial will guide you through all of the necessary steps to creating a simple 3d game.

Download and Install Xors3d and [Phi]sics

Before we get started, you will need to download and install Xors3d and [Phi]sics from:

http://www.xors3d.com

For Xors3d, click on the Depository icon, then click Core, and download the latest Setup executable. For Physics, once again go to the Depository section and this time click Add-ons, and then [Phi]sics. From there, download the latest Setup executable.

Now that you have both Xors3d and [Phi]sics downloaded to your computer, run them both to install the necessary files onto your computer. When the install is complete, you'll need to copy the Xors3d and [Phi]sics mod folders to your BlitzMax mod folder. By default, these are located in the following locations:

C:\Program Files\Xors3d Engine\port\blitzmax\rubux.mod
C:\Program Files\Xors3d Engine\addons\[Phi]sics\port\blitzmax\rubux.mod\phisics.mod

Please note that the physics.mod needs to be placed inside the rubux.mod in your BlitzMax mod directory. When you've done that, build your newly installed modules (for more information on building modules please visit the BlitzMax forums).

Create a new folder for our project and copy over the .dll files from the following locations:

C:\Program Files\Xors3d Engine\dlls
C:\Program Files\Xors3d Engine\addons\[Phi]sics\dlls

Now create a new folder and rename it as 'Media'. Finally, open up BlitzMax and create a new file, saving it in your project folder.

Creating a Simple 3d Scene

We'll be using Xors3d as our Framework (meaning that no other modules will be included when our code is compiled, making compile times quicker and leaving us with smaller .exe files). We then need to create a camera for 3d rendering, a light, and a 3d cube so we have something to look at. Here's the code:

```
'Use Xors3d as a Framework
Framework rubux.xors3d
'Setup Graphics
xGraphics3D 800,600,0,0,True
xSetBuffer xBackBuffer()
'Create a Camera
Global Camera = xCreateCamera()
'Create a Light
Global Light = xCreateLight()
xRotateEntity Light,45,45,0
'Create a Cube so we have something to look at
Global Cube = xCreateCube()
xPositionEntity Cube,0,0,10
Repeat
      'Spin the cube around it's Y axis
      xTurnEntity Cube,0,1,0
      'Update and Render the screen
      xUpdateWorld
      xRenderWorld
      'Flip buffers
      xFlip
Until xKeyDown (1) Or xWinMessage ("WM_CLOSE")
```

Modelling and Texturing some simple 3d Objects

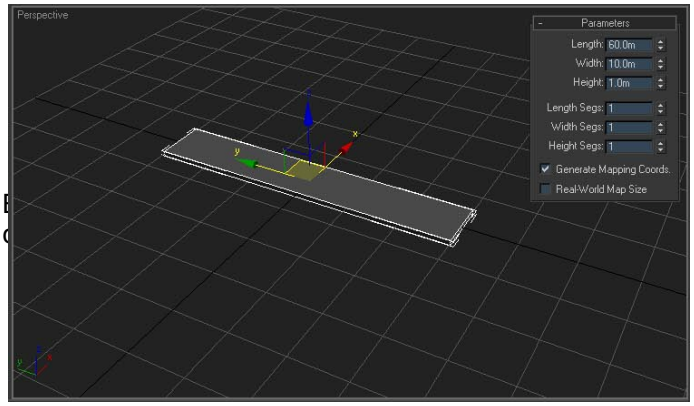
I'll be using 3ds max for making the models but the same procedure should be very similar in all 3d modelling applications. For an alternative (and free) 3d modeller, have a look at:

http://www.blender.org/

For the textures, I'm using Photoshop. Again, the techniques should be somewhat similar in other image processing applications. For a free Photoshop alternative, have a look at:

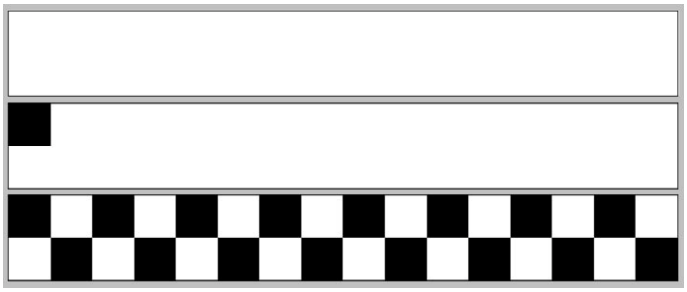
http://www.gimp.org/

Let's get started with our first model, a basic 3d box that we'll use as a bowling lane. Open your 3d modeller and create a new box with dimensions 60.0 height, 10.0 width, and 1.0 depth. Make sure that the model's pivot is centered and that the model is positioned at 0,0,0. This will become important later.

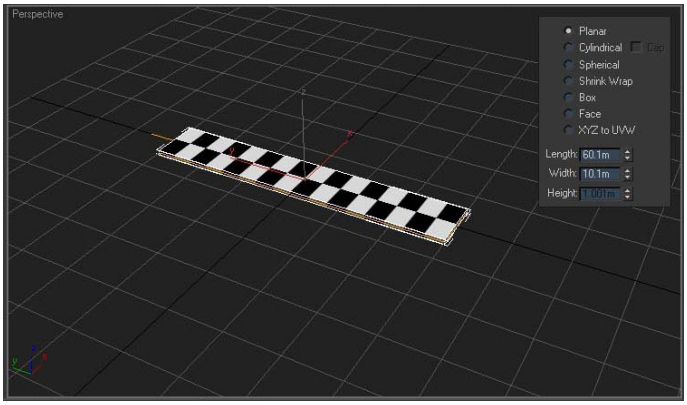


Bowled over by 3D well help is at hand...

whatever image processor you're using) and create a new file with dimensions 512 height and 64 width. Fill the canvas with a white colour and create a black 32 x 32 pixel square in one corner. Copy this square and paste it on alternate sides of the texture as shown.



Save your texture as Lane.png and go back to your model. Add a UVW modifier to the model with a planer setting and apply the Lane.png texture to it. You should see the checkered texture showing on the model and all of the squares should remain square, not stretched rectangles. If we were to use a square texture (for example 512 x 512) then the squares would become long and thin, warping and squashing the texture to fit the models shape.

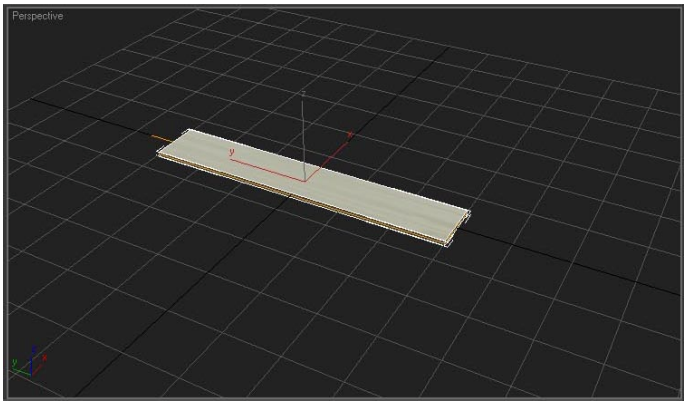


Go back to edit Lane.png and fill the canvas with a white colour. Using Photoshop, follow these steps:

Filter > Noise > Add Noise > About 30% for the Amount
Filter > Blur > Motion Blur > Enter 999 for Distance, -90 for the Angle
Filter > Blur > Gaussian Blur > Enter a Radius of 2.0
Filter > Sketch > Chrome > Play with the two values until it looks good
Image > Adjustmenst > Hue/Satuaration > Pick a nice pale wood colour

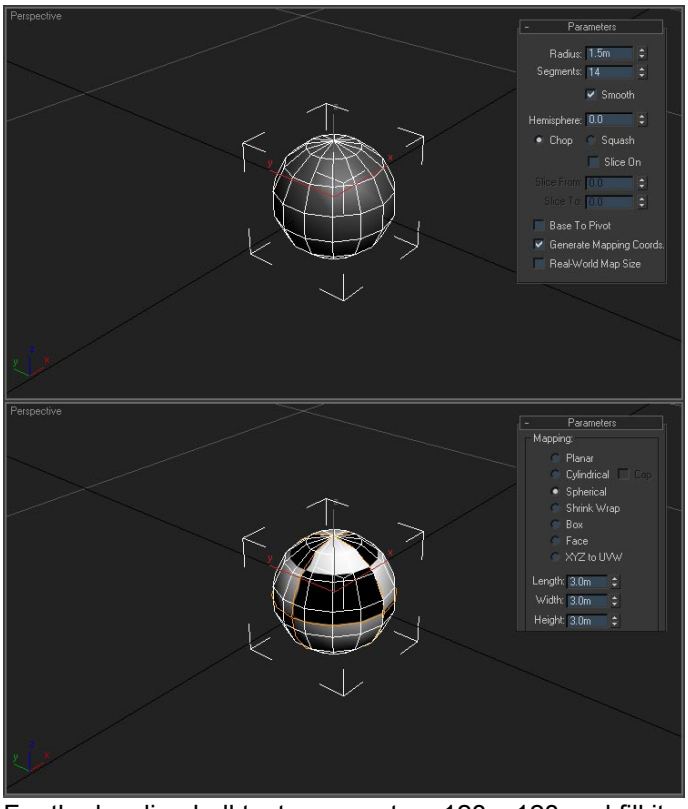
You could of course take this further with arrow and line details, and maybe a logo in the middle. For now though, I'll leave this here and move on.

Next up is our bowling ball model. Go back to your 3d



editor and make a new file. This time we want to create a Sphere with 1.5 radius and 14 segments (you can add more for a rounder look). Again, make sure that the pivot is at the center of the model and that it's positioned at 0,0,0. This time, apply a UVW modifier with Spherical mapping.

[BallModel01.jpg]

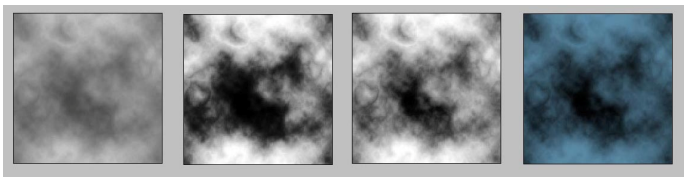


For the bowling ball texture, create a 128 x 128 and fill it with white and set your colours to the defaul black and white. Using Photoshop, follow these steps:

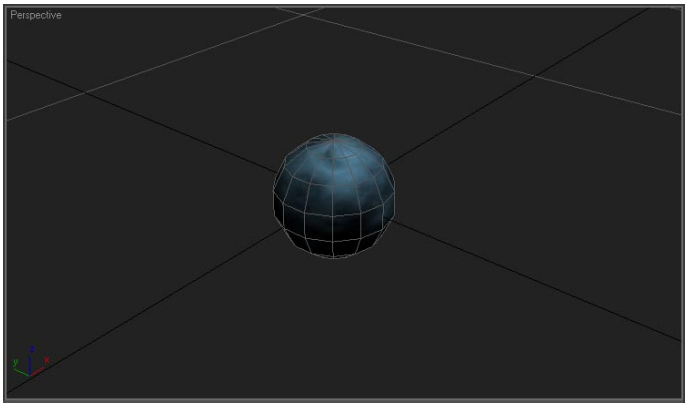
Filter > Render > Clouds
Filter > Render > Difference Clouds
Image > Adjustments > Shadows/Highlights > 100% Shadows 0% Highlights
Image > Adjustmenst > Hue/Satuaration > Pick a colour for your Bowling Ball

Bowled over by 3D well help is at hand...

Save the finished texture as Ball.png.



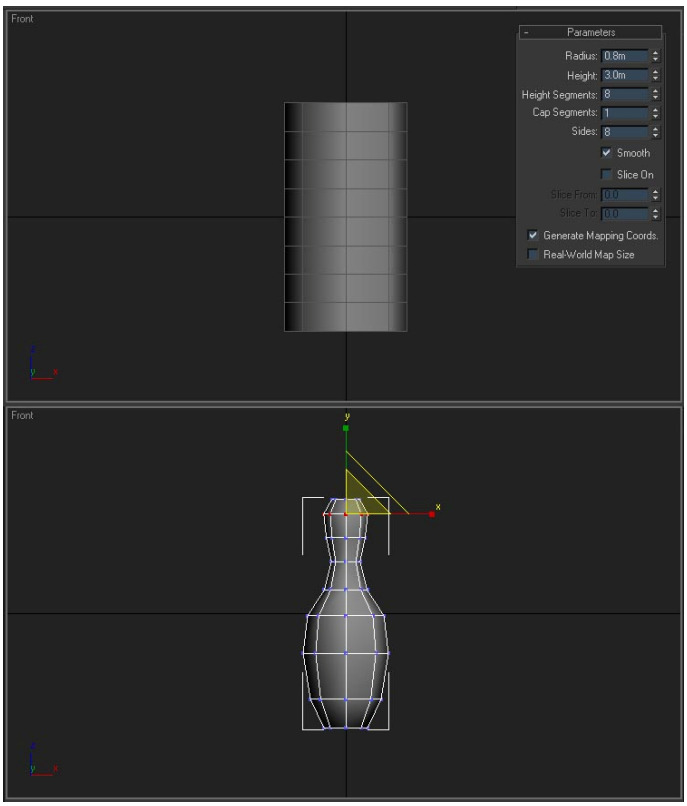
[BallTexture.jpg]



[BallModel02.jpg]

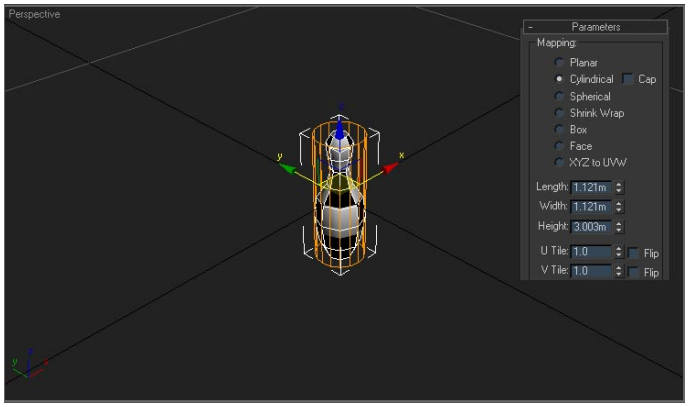
Finally, we'll make a model for our bowling pins. Create a new file in your 3d editor. This time create a Cylinder with a radius of 0.8, a height of 3.0, 8 height segments, and 8 sides. Select lines of vertices and scale them larger or smaller in groups until you have a basic bowling pin shape. When you're done, center the models pivot and position it to 0,0,0.

[PinModel01.jpg]



Apply a UVW modifier and select Cylindrical wrapping.

[PinModel02.jpg]

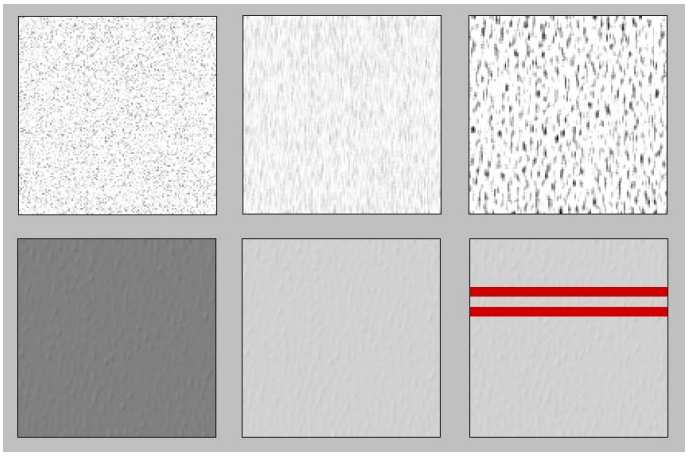


For our final texture, create a 128 x 128 texture. Fill the canvas with white and select the default black and white colours.

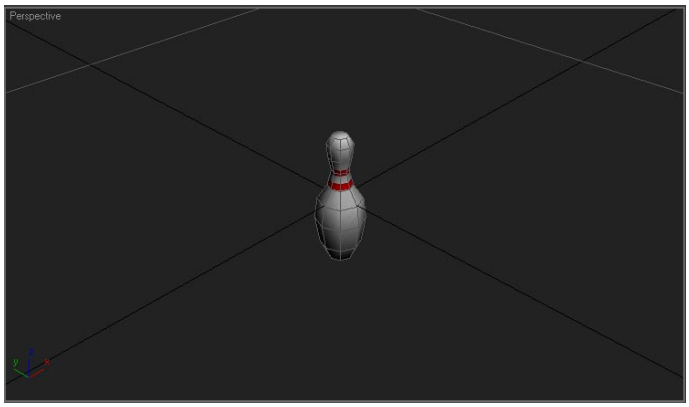
Filter > Noise > Add Noise > About 15% for the Amount
Filter > Blur > Motion Blur > Enter 5 for Distance, -90 for the Angle
Filter > Artistic > Poster Edges > Play with the three values until it looks good
Filter > Stylize > Emboss > Enter 1 for Height and 15% for the Amount
Image > Adjustments > Brightness/Contrast > Increase the Brightness by +100

Create a new layer, and draw two solid red lines on the top half of the texture. Set this layers blend mode to Multiply and flatten the image before saving as Pin.png.

[PinTexture01.jpg]



[PinMode03.jpg]



Bowled over by 3D well help is at hand...

Back in the 3d editor, export each of the finished models to our Media folder that we created earlier. Export the models as .b3d format, naming the models Lane.b3d, Ball.b3d, and Pin.b3d. Make sure that the texture for each model is also placed inside the Media folder.

Loading our 3d Objects into BlitzMax

As you can see in the code below, I've replaced the spinning cube with the Pin.b3d and Ball.b3d that we have created. These models are loaded into BlitzMax with the xLoadMesh command.

```
'Use Xors3d as a Framework
Framework rubux.xors3d
'Setup Graphics
xGraphics3D 800,600,0,0,True
xSetBuffer xBackBuffer()
'Create a Camera
Global Camera = xCreateCamera()
'Create a Light
Global Light = xCreateLight()
xRotateEntity Light,45,45,0
'Load and Position the Pin
Global Pin = xLoadMesh ("Media\Pin.b3d")
xPositionEntity Pin,-2,0,10
'Load and Position the Ball
Global Ball = xLoadMesh ("Media\Ball.b3d")
xPositionEntity Ball,2,0,10
'Load and Position the Lane
Global Lane = xLoadMesh ("Media\Lane.b3d")
xPositionEntity Lane,0,-5,40
Repeat
	'Turn the Pin and Ball so we can see how they look
	xTurnEntity Pin,0,1,0
	xTurnEntity Ball,0,1,0
	'Update and Render the screen
	xUpdateWorld
	xRenderWorld
	'Flip buffers
	xFlip
Until xKeyDown (1) Or xWinMessage ("WM_CLOSE")
```

Positioning the Pins and Creating a Second Camera

Adding the pins to our code as a Type allows us to create lots of them and control them all using the same code. Loading a single Pin.b3d model and copying it much more efficient than loading Pin.b3d each time we create a new one. We'll hide the first one that we load so it won't be rendered to the screen.

Creating a second Camera lets us view all the pins without having to move our main camera around. Here's the code:

```
'Use Xors3d as a Framework
Framework rubux xors3d
'We need to Import brl.linklist because we want to store
our objects in a TList
Import brl linkedlist
'Setup Graphics
xGraphics3D 800,600,0,0,True
xSetBuffer xBackBuffer()
'Create a Camera
Global Camera = xCreateCamera()
'Create a new Camera to look down at the Pins
Global NewCamera = xCreateCamera()
xPositionEntity NewCamera,0,10,57
xRotateEntity NewCamera,90,0,0
xCameraViewport NewCamera,50,50,200,200
'Create a Light
Global Light = xCreateLight()
xRotateEntity Light,45,45,0
'Load and Hide the Pin
Global PinOriginal = xLoadMesh ("Media\Pin.b3d")
xHideEntity PinOriginal
'Load and Position the Ball
Global Ball = xLoadMesh ("Media\Ball.b3d")
xPositionEntity Ball,0,-3,6
'Load and Position the Lane
Global Lane = xLoadMesh ("Media\Lane.b3d")
xPositionEntity Lane,0,-5,40
'Create a TList to store the Pins
Global PinList:TList = New TList
'Pin Type for handling the Pins
Type Pin
	Field Mesh
	Function Setup()
		Local X:Float
		'4 Pins for the 4th row
		For X = -3 To 3 Step 2
			CreatePin (X,60)
		Next
		'3 Pins for the 3rd row
		For X = -2 To 2 Step 2
			CreatePin (X,58)
		Next
		'2 Pins for the 2nd row
		For X = -1 To 1 Step 2
			CreatePin (X,56)
		Next
		'1 Pin for the 1st row
		CreatePin (0,54)
	End Function
	Function CreatePin (X:Float,Z:Float)
		Local NewPin:Pin = New Pin
		'Copy the 1st Pin that we loaded earlier
and Position it
		NewPin.Mesh = xCopyEntity (PinOriginal)
		xPositionEntity NewPin.Mesh,X,-2,Z
		'Add our new Pin to the PinList
		PinList.AddLast NewPin
	End Function
End Type
'Call our function the setup the Pins
Pin.Setup()
Repeat
```

Bowled over by 3D well help is at hand...

```
	'Update and Render the screen
	xUpdateWorld
	xRenderWorld
	'Flip buffers
	xFlip
Until xKeyDown (1) Or xWinMessage ("WM_CLOSE")
```

Adding Physics

~~Now we're going~~ to add some physics bodies to our models so that they can collide with each other. To begin using physics, we need to add this line before creating any physics bodies:

```
False "key"
```

Replacing 'False' with 'True' will create an infinite invisible plane at 0,0,0, that's not what we want. If you have purchsed the [Phy]sics add on from the Xors3d website then replace "key" with your [Phy]sics license key.

For the bowling lane, we're going to use a static physics body. Setting the mass of a physics body to '0' will make sure it's not affected by gravity or moved by other physics bodies. For the bowling lane, we will make a cube physics body:

If you remember earlier in the tutorial, we centered all of our models before exporting. When we create physics bodies, they are created from the center so having centered models is necessary for the 3d model to line up correctly with the physics body.

For the pins, we'll also use a cube physics body. It's much quicker and simpler to use a cube to represent our pins than to use any other method. There are other physics shapes that can be created such as a convex hull and a trimesh, however these are a little more complicated to setup and use.

```
'Use Xors3d as a Framework
Framework rubux.xors3d
'Import the rubux.phisics module
Import rubux.physics
'We need to Import brl.linklist because we want to store
our objects in a TList
Import brl.linkedlist
'Setup Graphics
xGraphics3D 800,600,0,0,True
xSetBuffer xBackBuffer()
'Setup Physics
pxCreateWorld (False,"key")
pxSetGravity 0,-10,0
'Create a Camera
Global Camera = xCreateCamera()
'Create a new Camera to look down at the Pins
Global NewCamera = xCreateCamera()
xPositionEntity NewCamera,0,10,57
xRotateEntity NewCamera,90,0,0
xCameraViewport NewCamera,50,50,200,200
xCameraProjMode NewCamera,2
xCameraZoom NewCamera,25
```

```
'Create a Light
Global Light = xCreateLight()
xRotateEntity Light,45,45,0
'Load and Hide the Pin
Global PinOriginal = xLoadMesh ("Media\Pin.b3d")
xHideEntity PinOriginal
'Load and Position the Ball
Global Ball = xLoadMesh ("Media\Ball.b3d")
xPositionEntity Ball,0,-3,6
'Load and position the Lane
Global Lane = xLoadMesh ("Media\Lane.b3d")
xPositionEntity Lane,0,-5,40
'Create a TList to store the Pins
Global PinList:TList = New TList
'Create a TList to store the Physics Bodies
Global BodyList:TList = New TList
'Pin Type for handling the Pins
Type Pin
	Field Mesh
	Function Setup()
		Local X:Float
		'4 Pins for the 4th row
		For X = -3 To 3 Step 2
			CreatePin (X,60)
		Next
		'3 Pins for the 3rd row
		For X = -2 To 2 Step 2
			CreatePin (X,58)
		Next
		'2 Pins for the 2nd row
		For X = -1 To 1 Step 2
			CreatePin (X,56)
		Next
		'1 Pin for the 1st row
		CreatePin (0,54)
	End Function
	Function CreatePin (X:Float,Z:Float)
		Local NewPin:Pin = New Pin
		'Copy the 1st Pin that we loaded earlier
and Position it
		NewPin.Mesh = xCopyEntity (PinOriginal)
		xPositionEntity NewPin.Mesh,X,0,Z
		'Add our new Pin to the PinList
		PinList.AddLast NewPin
		Local NewBody:Body = New Body
		'Create a Physics Cube as the collision
shape for the Pin
		NewBody.Body = pxBodyCreateCube
(0.4,1.5,0.5,2)'scale is half of editor size? varies
with each physics engine
		NewBody.Mesh = NewPin.Mesh
		'Position the Physics Body at the same
position as the Pin
		pxBodySetPosition
NewBody.Body,xEntityX(NewPin.Mesh),xEntityY(NewPin.Mes
h),xEntityZ(NewPin.Mesh)
		pxBodySetRotation
NewBody.Body,xEntityPitch(NewPin.Mesh),xEntityYaw(NewP
in.Mesh),xEntityRoll(NewPin.Mesh)
		'Add our new Physics Body to the BodyList
		BodyList.AddLast NewBody
```

Bowled over by 3D well help is at hand...

```
	End Function
End Type
'Body Type for handling the Physics Bodies
Type Body
	Field Body,Mesh
End Type
'Call our function the setup the Pins
Pin.Setup()
'Create a new Physics Body for the Lane
Local NewBody:Body = New Body
NewBody.Body = pxBodyCreateCube (5,0.5,30,0)'scale x y z
+ mass
NewBody.Mesh = Lane
'Position the Physics Body at the same position as the
Lane
pxBodySetPosition
NewBody.Body,xEntityX(Lane),xEntityY(Lane),xEntityZ(Lane
)
pxBodySetRotation
NewBody.Body,xEntityPitch(Lane),xEntityYaw(Lane),xEntity
Roll(Lane)
'Add our new Physics Body to the BodyList
BodyList.AddLast NewBody
Repeat
	'Align each Physics Mesh and Body
	For Local ThisBody:Body = EachIn BodyList
		xPositionEntity
ThisBody.Mesh,pxBodyGetPositionX(ThisBody.Body),pxBodyGe
tPositionY(ThisBody.Body),pxBodyGetPositionZ(ThisBody.Bo
dy)
		xRotateEntity
ThisBody.Mesh,pxBodyGetRotationPitch(ThisBody.Body),pxBo
dyGetRotationYaw(ThisBody.Body),pxBodyGetRotationRoll(Th
isBody.Body)
	Next
	'Update Physics
	pxRenderPhysic 30,False
	'Update and Render the screen
	xUpdateWorld
	xRenderWorld
	'Flip buffers
	xFlip
Until xKeyDown (1) Or xWinMessage ("WM_CLOSE")
```

Adding Player Controls

Next we're going add some player controls so you can roll the bowling ball down the lane and towards the pins. Moving a physics body is done by apply force. The following line is responsible for adding a forward force to the bowling ball:

In the code below, I've added controls to move the bowling ball left and right using the arrow keys, space bar to roll the ball, and 'r' to reset the pins:

```
'Use Xors3d as a Framework
Framework rubux xors3d
'Import the rubux.phisics module
Import rubux.phisics
'We need to Import brl.linklist because we want to store
our objects in a TList
Import brl.linkedlist
'Setup Graphics
xGraphics3D 800,600,0,0,True
xSetBuffer xBackBuffer()
'Setup Physics
pxCreateWorld (False,"key")
pxSetGravity 0,-10,0
'Create a Camera
Global Camera = xCreateCamera()
'Create a new Camera to look down at the Pins
Global NewCamera = xCreateCamera()
xPositionEntity NewCamera,0,10,57
xRotateEntity NewCamera,90,0,0
xCameraViewport NewCamera,50,50,200,200
xCameraProjMode NewCamera,2
xCameraZoom NewCamera,25
'Create a Light
Global Light = xCreateLight()
xRotateEntity Light,45,45,0
'Load and Hide the Pin
Global PinOriginal = xLoadMesh ("Media\Pin.b3d")
xHideEntity PinOriginal
'Load and Position the Ball
Global Ball = xLoadMesh ("Media\Ball.b3d")
xPositionEntity Ball,0,-3,6
'Load and position the Lane
Global Lane = xLoadMesh ("Media\Lane.b3d")
xPositionEntity Lane,0,-5,40
'Create a TList to store the Pins
Global PinList:TList = New TList
'Create a TList to store the Physics Bodies
Global BodyList:TList = New TList
'Pin Type for handling the Pins
Type Pin
	Field Mesh
	Function Setup()
		Local X:Float
		'4 Pins for the 4th row
		For X = -3 To 3 Step 2
			CreatePin (X,60)
		Next
		'3 Pins for the 3rd row
		For X = -2 To 2 Step 2
			CreatePin (X,58)
		Next
		'2 Pins for the 2nd row
		For X = -1 To 1 Step 2
			CreatePin (X,56)
		Next
		'1 Pin for the 1st row
		CreatePin (0,54)
	End Function
	Function CreatePin (X:Float,Z:Float)
		Local NewPin:Pin = New Pin
		'Copy the 1st Pin that we loaded earlier
and Position it
		NewPin.Mesh = xCopyEntity (PinOriginal)
		xPositionEntity NewPin.Mesh,X,0,Z
		'Add our new Pin to the PinList
		PinList.AddLast NewPin
		Local NewBody:Body = New Body
```

Bowled over by 3D well help is at hand...

```
            'Create a Physics Cube as the collision
shape for the Pin
            NewBody.Body = pxBodyCreateCube
(0.4,1.5,0.5,2)'scale is half of editor size? varies
with each physics engine
            NewBody.Mesh = NewPin.Mesh
            'Position the Physics Body at the same
position as the Pin
            pxBodySetPosition
NewBody.Body,xEntityX(NewPin.Mesh),xEntityY(NewPin.Mesh)
,xEntityZ(NewPin.Mesh)
            pxBodySetRotation
NewBody.Body,xEntityPitch(NewPin.Mesh),xEntityYaw(NewPin
.Mesh),xEntityRoll(NewPin.Mesh)
            'Add our new Physics Body to the BodyList
            BodyList.AddLast NewBody
      End Function
      Function DeleteAll()
            'Free the Physics Mesh, Physics Body, and
            For ThisPin:Pin = EachIn PinList
                  For ThisBody:Body = EachIn BodyList
                        If ThisBody.Mesh = ThisPin.Mesh
                              xFreeEntity ThisBody.Mesh
                              pxDeleteBody ThisBody.Body
                              'Remove the Pin and Body
from their respective lists
                              BodyList.Remove ThisBody
                              PinList.Remove ThisPin
                        EndIf
                  Next
            Next
      End Function
End Type
'Body Type for handling the Physics Bodies
Type Body
      Field Body,Mesh
End Type
'Call our function the setup the Pins
Pin.Setup()
'Create a new Physics Body for the Lane
Local NewBody:Body = New Body
NewBody.Body = pxBodyCreateCube (5,0.5,30,0)'scale x y z
+ mass
NewBody.Mesh = Lane
'Position the Physics Body at the same position as the
Lane
pxBodySetPosition
NewBody.Body,xEntityX(Lane),xEntityY(Lane),xEntityZ(Lane
)
pxBodySetRotation
NewBody.Body,xEntityPitch(Lane),xEntityYaw(Lane),xEntity
Roll(Lane)
'Add our new Physics Body to the BodyList
BodyList.AddLast NewBody
'A Global variable to let us know if the Ball has
already been rolled
Global BallRolled = False
Repeat
      'If we're still holding the Ball
      If BallRolled = False
            'Move the Ball Left and Right with the Arrow
Keys
            If xEntityX (Ball) > -2.5
                  If xKeyDown (203) Then xMoveEntity
Ball,-0.1,0,0
            EndIf
            If xEntityX (Ball) < 2.5
                  If xKeyDown (205) Then xMoveEntity
Ball,0.1,0,0
            EndIf
            'Roll the Ball with the Spacebar
            If xKeyHit (57)
                  'Hide the original Ball
                  xHideEntity Ball
                  'Create a new Physics Body for the
rolling Ball
                  Local BallBody:Body = New Body
                  BallBody.Body = pxBodyCreateSphere
(1.5,20)'scale + mass
                  BallBody.Mesh = xCopyEntity (Ball)
                  'Position the Physics Body at the
same position as the new Ball
                  pxBodySetPosition
BallBody.Body,xEntityX(Ball),xEntityY(Ball),xEntityZ(B
all)
                  pxBodySetRotation
BallBody.Body,xEntityPitch(Ball),xEntityYaw(Ball),xEnt
ityRoll(Ball)
                  'Add our new Physics Body to the
BodyList
                  BodyList.AddLast BallBody
                  'Roll the new Ball forwards towards
the Pins
                  pxBodyAddForce
BallBody.Body,0,0,500,1
                  'Set our Global variable to True so
we know that we're not holding the Ball anymore
                  BallRolled = True
            EndIf
      EndIf
      'Reset Pins and Ball with the 'R' Key
      If xKeyHit (19)
            'Check if there is a Ball
            If BallBody <> Null
                  'If there is a Ball, Free it
                  xFreeEntity BallBody.Mesh
                  pxDeleteBody BallBody.Body
                  'Remove the Body from the Body List
                  BodyList.Remove BallBody
            EndIf
            Pin.DeleteAll()
            Pin.Setup()
            'Show the original Ball
            xShowEntity Ball
            'Set our Global variable to False so we
know that we're holding the Ball again
            BallRolled = False
      EndIf
      'Align each Physics Mesh and Body
      For Local ThisBody:Body = EachIn BodyList
            xPositionEntity
ThisBody.Mesh,pxBodyGetPositionX(ThisBody.Body),pxBody
GetPositionY(ThisBody.Body),pxBodyGetPositionZ(ThisBod
y.Body)
            xRotateEntity
ThisBody.Mesh,pxBodyGetRotationPitch(ThisBody.Body),px
BodyGetRotationYaw(ThisBody.Body),pxBodyGetRotationRol
l(ThisBody.Body)
      Next
      'Update Physics
      pxRenderPhysic 30,False
      'Update and Render the screen
      xUpdateWorld
      xRenderWorld
      'Flip buffers
      xFlip
Until xKeyDown (1) Or xWinMessage ("WM_CLOSE")
```

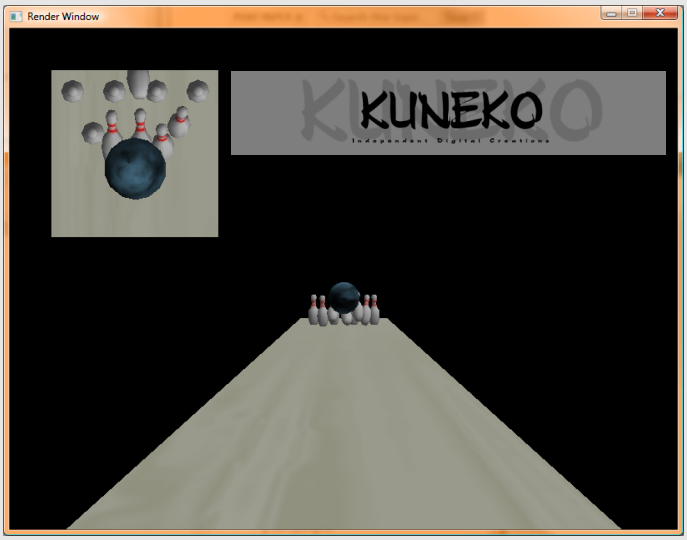
Hopefully, this tutorial will help get you started with creating your own 3d games.

Bowled over by 3D well help is at hand...

Projects: Bongo, The Pyramid, Chalk Physics...

Developer: John Adams (JA2)

Website: www.kuneko.com



Feedback

Well that's it for this month I know there are a couple of people working on articles for the next issue that just didn't make it into this one.

As you can see I have responded to feedback from the first issue and you now have a printer friendly white background.

And a slightly larger font size and two column layout.

I've kept the highlighted section headers as I like the gel button style, but I am open to futher advice and feedback.

I would like to thank everyone who has contributed to the first and second issue of BlitzMAX coder and hope that you will keep helping out with new articles...

Yet again let me know via e-mail at

feedback@blitzmaxcoder.com

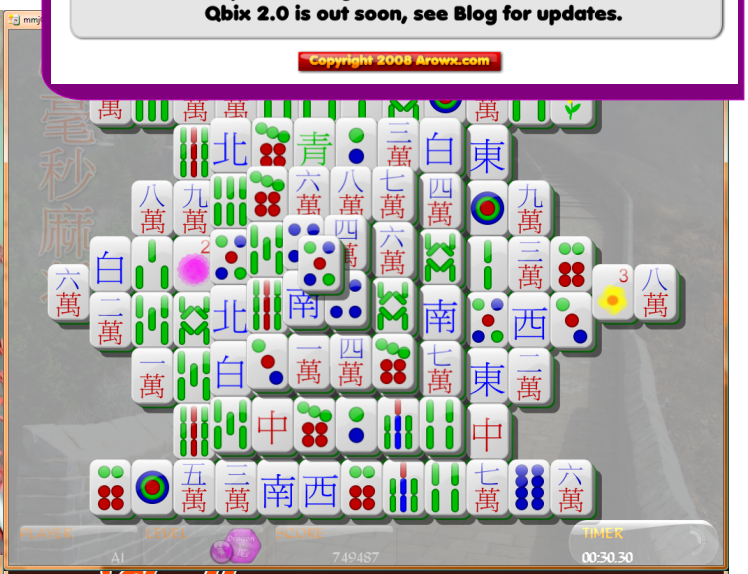
or on the blitzmax forums.

In case you missed my earlier advert for the beta version of mmjChallenge, yets it's Mah-Jongg but with challenge codes here it is again...

Projects: Candy Cascade, Qbix, mmjChallenge

Developer: Allan Rowntree

Websites: arowx.com
blitzmaxcoder.com
mmjchallenge.com



http://www.mmjChallenge.com