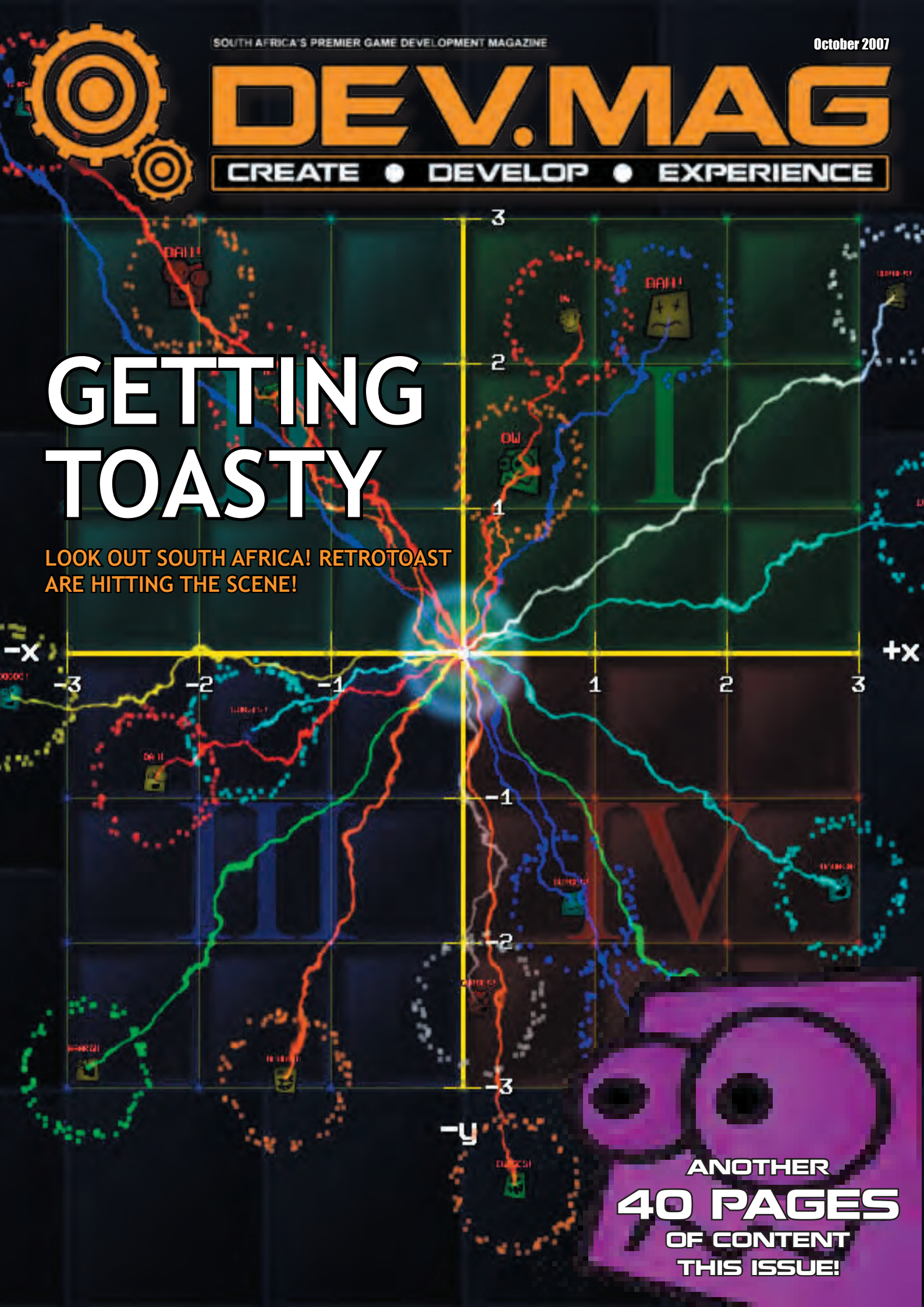
SOUTH AFRICA'S PREMIER GAME DEVELOPMENT MAGAZINE

October 2007

DEV.MAG

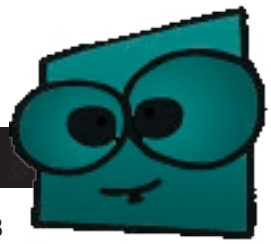
CREATE • DEVELOP • EXPERIENCE



GETTING TOASTY

LOOK OUT SOUTH AFRICA! RETROTOAST ARE HITTING THE SCENE!

ANOTHER 40 PAGES OF CONTENT THIS ISSUE!



REGULARS

FEATURE

OPINIONS

REVIEWS

TUTORIALS

DESIGN

TAILPIECE

DEAR READER ...

The rAge expo this year has come and gone, leaving most of us exhausted. Of course, we only get a brief breather period before hitting the workmill again. Interesting news has been pouring in from all sides this month, quickly filling our little bucket of content and forcing us to send people to hunt down some more containers.

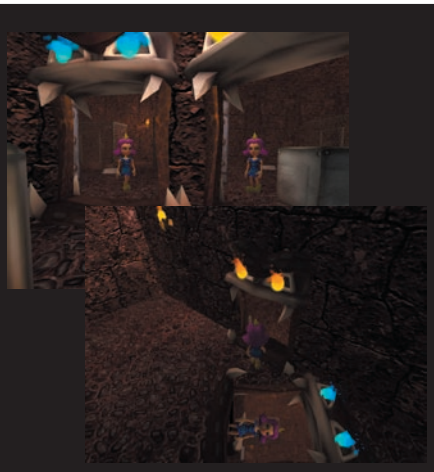
Revolting metaphors aside, we really do have quite a few interesting things to blab this month. First off, we're pleased to announce the recent success of Retrotoast Studios, a group of local developers who have just published their first game — after already winning R5 000 from Mindset in Game.Dev's Comp 15. Not only have they just set up their website and released the game to the buying market, but they kindly agreed to an interview while at rAge this year. Dev.Mag spoke with Cadyn Bridgman and Louis Pieterse for a few minutes about the company and their plans for the next couple of months, and the full report can be found in this month's feature.

Another juicy piece of content is the 6-page monster in this month's design section, graciously provided by the folks at Luma. Their strong presence at rAge this year is a result of their success with Mini#37, an arcade racer that has raised the bar for South African developers everywhere.

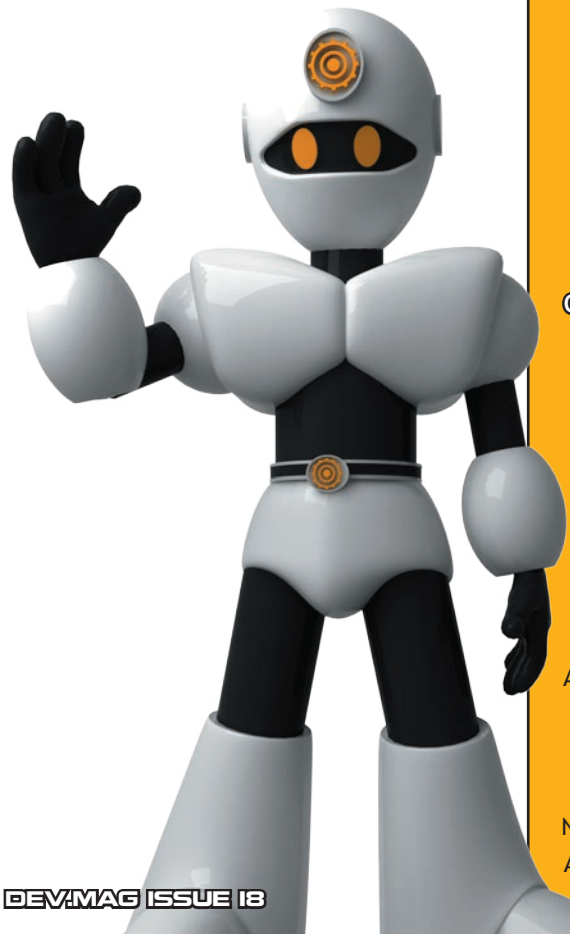
Finally, the mag has undergone some more experimentation with design and content in this issue. Most of it is rather subtle, and sometimes it's just individual articles which are played around with. But changes, expansions, cuts and random nosepicks have indeed been made, and we need feedback on it all. Be sure to comment on anything that strikes you as particularly good or revolting in this edition so that we can keep it or scrap it! Our e-mail address is, as always, advertised in the right column of this page.

That's all for now. Enjoy this edition, and don't forget to write to us!

RODAIN "NANDREW" JOUBERT
EDITOR



A little while back, we did a review of Narbacular Drop. This game was the spiritual predecessor of Portal, created by the same team while they were still students at Digipen. If you're interested in seeing the title that sparked off a gameplay revolution, check out the original website at http://www.nuclearmonkeysoftware.com/



DEV.MAG
CREATE • DEVELOP • EXPERIENCE

EDITOR
Rodain "Nandrew" Joubert

DEPUTY EDITOR
Claudio "Chippit" de Sa

SUB EDITOR
Tarryn "Azimuth" van der Byl

DESIGNER
Brandon "Cyberninja" Rajkumar

MARKETING
Bernard "Mushi Mushi" Boshoff
Andre "Fengol" Odendaal

WRITERS
Simon "Tr00jg" de la Rouviere
Ricky "Insomniac" Abell
William "Cairnswm" Cairns
Bernard "Mushi Mushi" Boshoff
Danny "Dislekcia" Day
Andre "Fengol" Odendaal
Luke "Coolhand" Lamothe
Rishal "UntouchableOne" Hurbans

WEBSITE ADMIN
Robbie "Squid" Fraser

WEBSITE
www.devmag.org.za

EMAIL
devmag@gmail.com

THIS MONTH'S GUEST WRITERS:

Quinton "Q-man" Bronkhorst

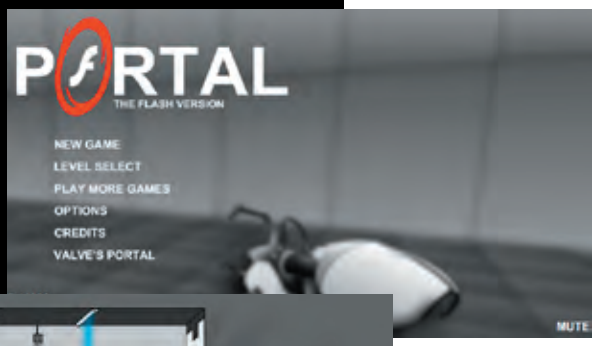
James "NightTimeHornets" Etherington-Smith

This magazine is a project of the South African Game.Dev community. Visit us at: **www.gamedotdev.co.za**

Now you're thinking in portals. And, erm, game development.

FLASH VERSION OF *PORTAL*

http://portal.wecreatestuff.com/



Looking for a bit of light entertainment between tasks? Try this nifty little 2D Flash version of the hit game Portal, lovingly crafted and professionally executed by its creators. It has 40 levels and carries over most of the material from the original game, give or take some items depending upon the particular demands and limitations of a two-dimensional environment. If you don't mind the inevitable loss of gameplay depth that accompanies the loss of the game world's depth, throwing portals around can prove to be an entertaining pastime even in the realm of Flash.

RETROTOAST UP AND RUNNING

http://www.retrotoast.com/

If you're keen on supporting local game development, then it wouldn't hurt to stop on over at the new Retrotoast site that's been set up to help promote their first published game, Cartesian Chaos. An educational game that has proven itself to actually be more fun than facts — something which is difficult to pull off properly nowadays — CC doesn't rest on hope and fancy. It also proved its worth by coming first place in a recent Game.Dev competition. More details in this issue's feature.

SHARING IS CARING IN GAME DESIGN

http://lostgarden.com/2005/08/why-you-should-share-your-game-designs.html

If you're looking for an interesting read, head on over to Lost Garden for an essay on game design that says you should - gasp!- share your ideas with other people. Most people's first instinct when it comes to design is that ideas should be guarded jealously - if you give the premise away too early, somebody else could rush in there and snatch your intellectual property. This article provides several convincing reasons as to why that is not necessarily the case, and why developers should be more open with their musings.

GAMEDEV.NET 4E COMPETITION STARTS

http://www.gamedev.net/community/contest/4e6/

Gamedev.net's prestigious annual competition, known as Four Elements (or simply 4E by aficionados) has reared its head this year for a sixth incarnation, this time with some very interesting ideas up its sleeve. This competition's distinguishing characteristic is the fact that all games created must deal significantly with four provided elements, either physically or thematically. This year's elements? Ponies, accountants, crystals and explosions. More details about the competition can be found on the site.

INDEPENDENT GAMES SUMMIT RELEASES VIDEO SERIES

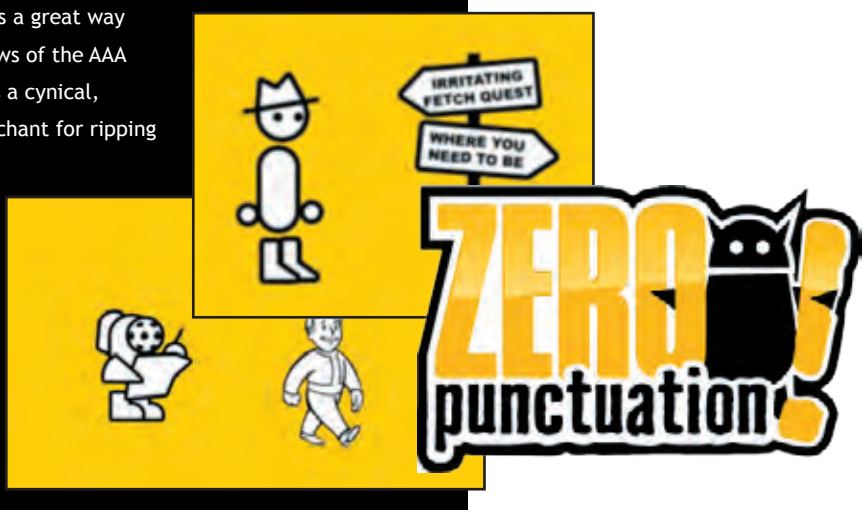
http://www.gamasutra.com/php-bin/news_index.php?story=16042

If you've got the bandwidth for them (and Quicktime Alternative), be sure to download all the Independent Games Summit (IGS) panels and speeches that are being released on video at the moment. They're just over 100megs each (some are a little bigger) but they're well worth the watch. One of these videos, provided in the link, focuses on Kim Swift of Valve talking about indie game development, Narbacular drop and getting hired by Valve to make portal. Videos can also be found at http://www.indiegamessummit.com/

ZERO PUNCTUATION, ZERO SHAME

http://www.escapistmagazine.com/articles/view/editorials/zeropunctuation

It's not strictly game development, but it's a great way to relax while poking fun at the design flaws of the AAA titles out there. Ben "Yahtzee" Croshaw is a cynical, wisecracking British-Australian with a penchant for ripping off games with the use of some hilarious videos. Interestingly enough, he's also the creator of some well-known AGS games such as 5 Days a Stranger, so once you've wiped a tear away after laughing hysterically at his videos, it's a good idea to go check his projects at http://www.fullyramblomatic.com/



YOYO GAMES LAUNCHES FIRST OFFICIAL COMPETITION

http://www.yoyogames.com/gamemaker/competition01

Five weeks, and you need to make a game themed around winter. Easy, right? Well, yes! Play your cards right, and you may also win prizes of up to US$1000. The Yoyo Winter Competition is Yoyo Games' first official comp, with a flexible premise and easy game creation using the Game Maker development tool. Moreover, entrants are allowed to submit two games apiece, increasing their chances of winning if they have multiple creative ideas. The competition ends at midnight on 23 December, so be sure to get cracking! More details on the site.

BLOGGING IT UP WITH MINI#37

http://www.garagegames.com/blogs/70158/13826

For those interested in keeping up to date with the progress on Luma's arcade racer, Mini#37, check out Dale Best's blog on Garage Games. The game has already been met with some very high praise from readers, and even the actual GG staff have allegedly given the game the thumbs-up. Mini#37 runs on the Torque Engine, created by Garage Games, and is an upstanding example of what can be done with the tool in the hands of professionals. For more details on Mini#37, turn to the 6-page postmortem in this issue's Design section.

STORYTELLING FOR THE DEVELOPER

http://www.escapistmagazine.com/news/view/70852-Next-Gen-Storytelling-Part-One-What-Makes-a-Story

Some developers out there are huge proponents of interactive storytelling (one of our writers is a vicious Deus Ex fanboy for that very reason). If you browse through The Escapist after watching Zero Punctuation, then you may come across an awesome series on storytelling in games written by the man himself, Warren Spector. If you haven't, then here's the link for you. Read it if you aspire to tell stories through their games somewhere down the line. Do it. Do it now.

RETROTOAST STUDIOS

South Africa's latest success story

by Rodain "Nandrew" Joubert

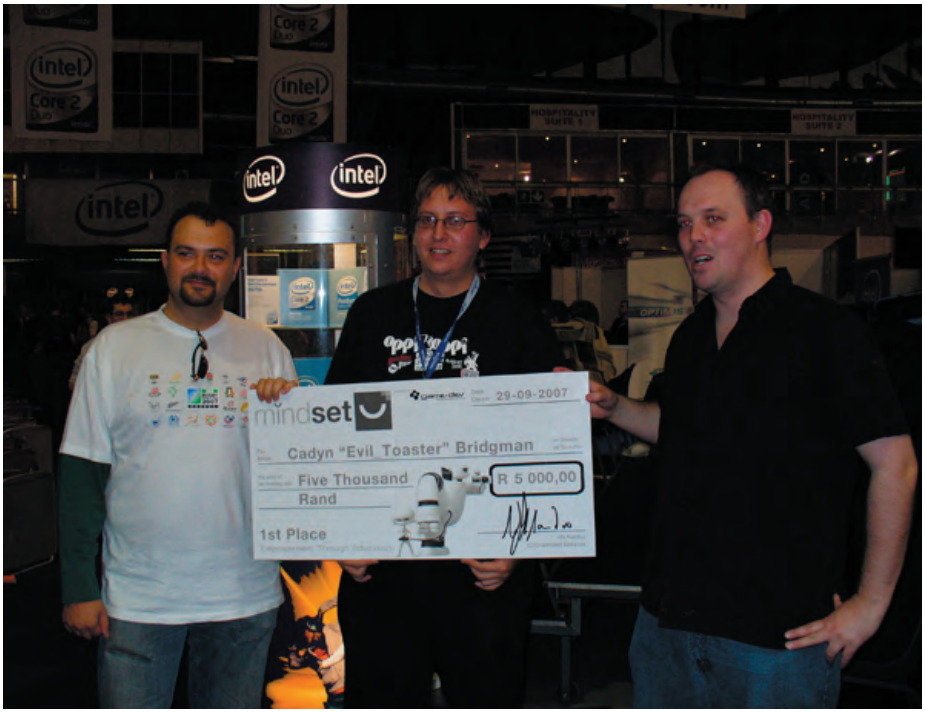
It's rAge 2007. We're sitting down to have a couple of drinks at Northgate's Wimpy, making the most of these few free minutes between presentations and prizegivings. There's not much time for an interview, so the little recording device is flicked on almost immediately, and the mike gets shoved under the nose of two men who may very easily become the next big thing in South African game development.

"Wait a minute, we need some beer first!"

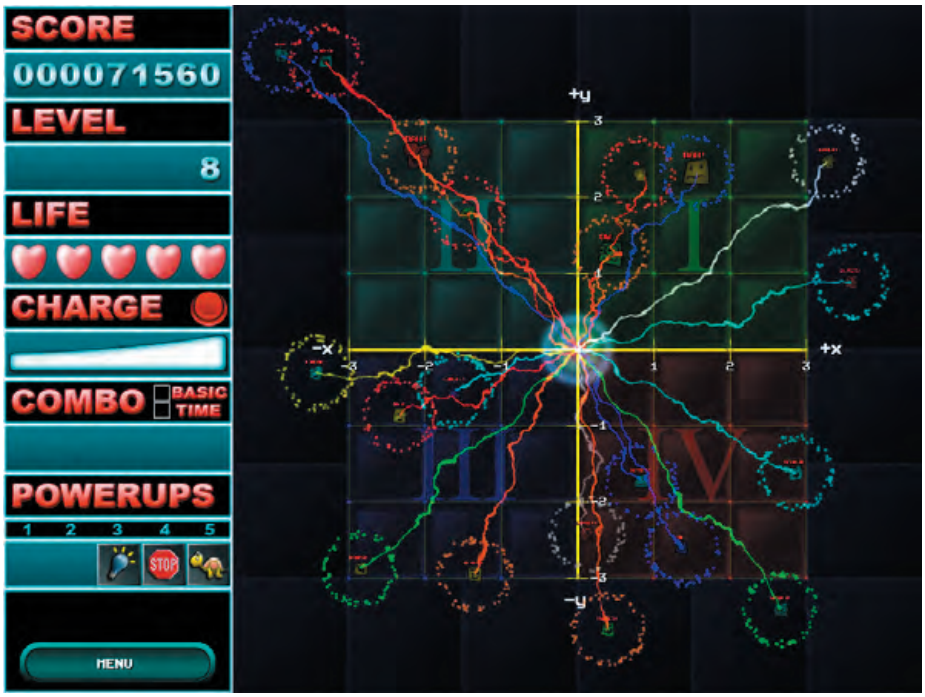
I'm sitting down with Cadyn Bridgman and Louis Pieterse from Retrotoast. They're part of a small team (which also includes 'visual angler' Daniel Petroff) responsible for bringing the world Cartesian Chaos, an educational game that earned them R5 000 — and later a publishing deal with Savage Software — when they claimed first place in a recent Game.Dev competition. The premise of Comp 15 was simple enough - through Game.Dev, Mindset sponsored a huge prize pool of R10 000 to make an educational game that would not only help people learn, but actually let them have stacks of fun in the process.

Claiming first place, the Retrotoast team were given an unprecedented opportunity to market their creation. Their publishing deal with Savage has had them working away furiously ever since it was officially confirmed - the goal being to tout their product to a mainstream audience.

As far as the toasters are concerned, this is all they need. "Sure, money's nice," Cadyn says. "We generally use what we're earning right now to pay for business costs such as software licenses and web hosting. But we



The jolly Retrotoast crew at rAge 2007. Note the sinister red eyes. And the giant cheque from Mindset. Present are (from left to right) Daniel Petroff (artist), Cadyn "Evil_Toaster" Bridgman (project manager / programmer), and Louis Pieterse (systems / management).



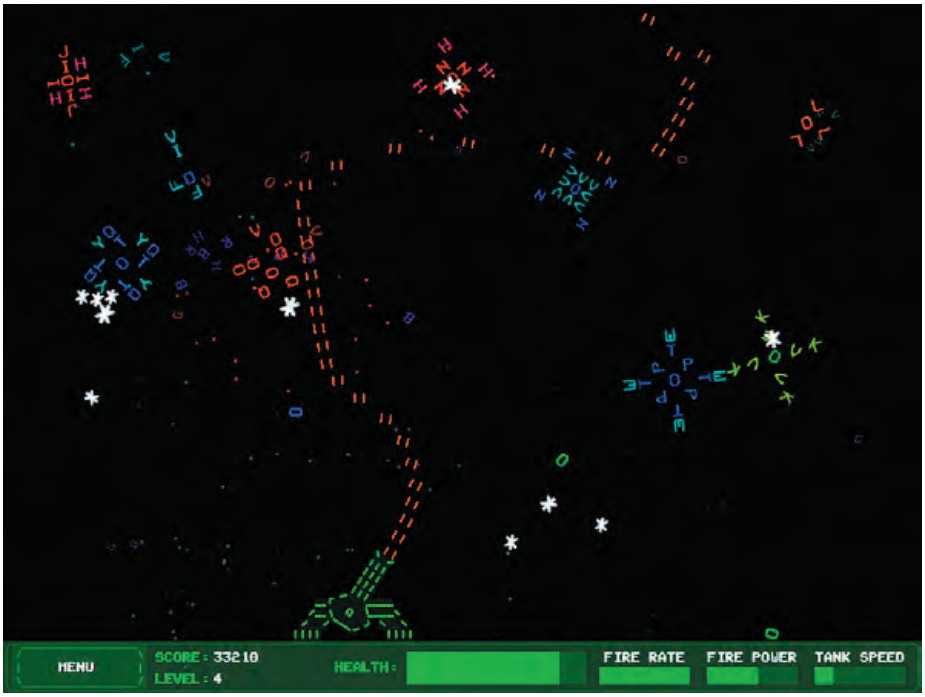
also enter competitions for inspiration." He sips his beer. "And discipline."

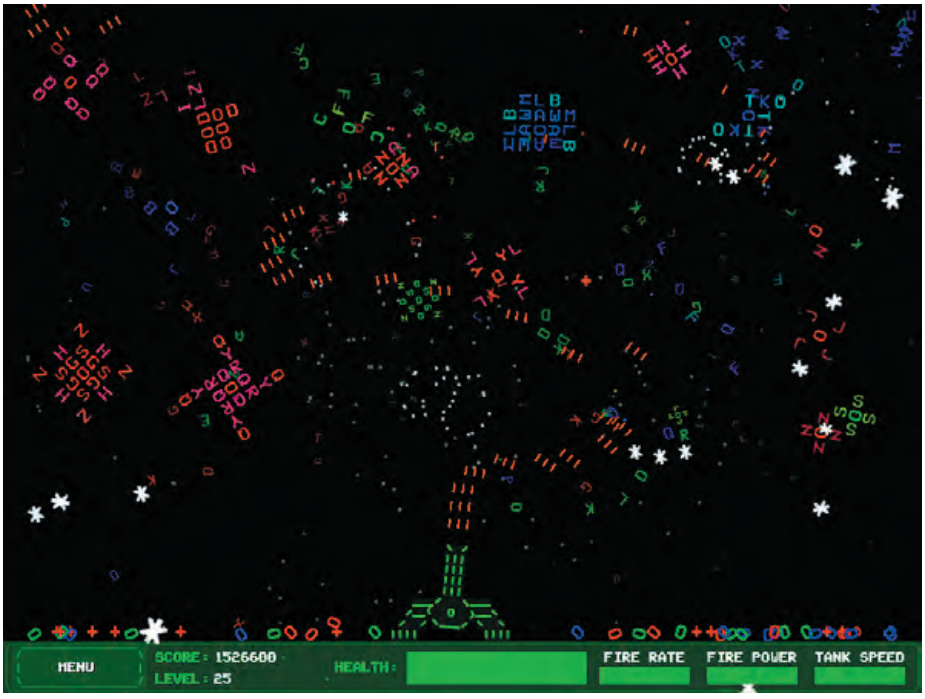
Cadyn has been involved with game development for a while, making his first considerable mark on the local scene with his management title, Fast Food in Space, which took first place in last year's major Game.Dev competition. It was shortly after this project that the idea of Retrotoast came into being, and he's been working on it with his team ever since.

The Retrotoast squad make extensive use of free and open source software for their creations. Moreover, all of their games run on a custom system known as the Toasted Engine, which constantly undergoes refinement — Louis volunteers an explanation of this. "When Fast Food in Space was made a while back, the engine was this crap." He makes a small space between his hands. "Now it's THIS crap!" he continues, throwing his arms wide open. "Crap means good, by the way."

A knack for creative endeavours combined with a keen work ethic and marvellous sense of humour easily put Retrotoast on a par with professional indie developers. In fact, not only do Retrotoast produce quality games, but they have the drive to back it all up. "We have a game plan," Cadyn explains. "It's not a case of 'let's make games and become rich and famous'. We've got a marketing scheme, we've got a list of goals and we're working towards them realistically."

The next six months in their game plan are simple - promote their newly published Car-

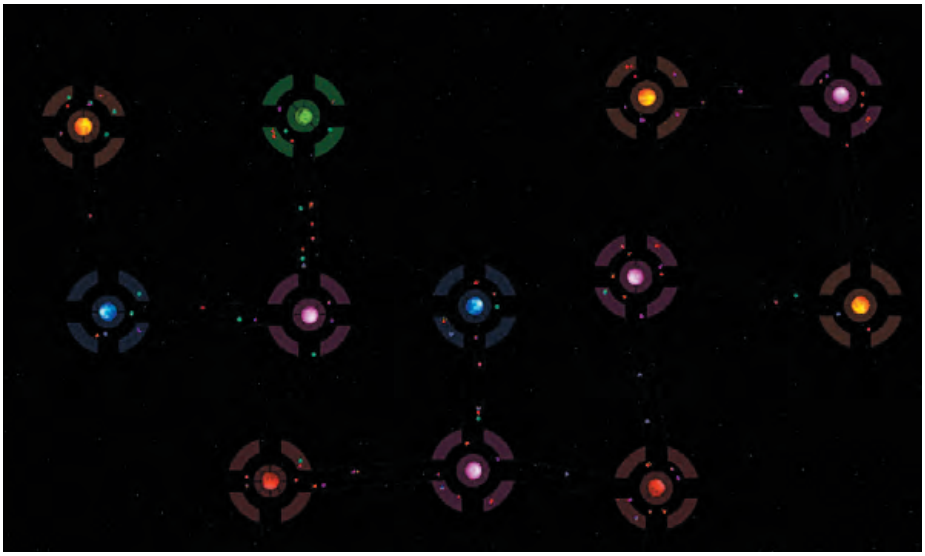




tesian Chaos, get their website fixed up, and minimise expenditure so that the focus can move towards churning out games. Another current project of theirs is RetroTank, which Cadyn hints at possibly being Retrotoast's next commercial venture.

Although the Retrotoast guys don't have an enormous amount of time to commit to community involvement, they do see the value of game development groups such as Game.Dev. "Sponsorship, awareness and community," are what they describe as the main perks, not to mention the raw cash that they've won by participating in local competitions.

With their game release just around the corner and a lot to look forward to, Retrotoast serve as a great example of what small-scale developers can actually achieve in this country. Their advice to other keen developers? Start small. Don't be lazy. Have a game plan. "And most importantly," Cadyn says, "always ask for a beer when somebody wants to interview you."



The crew have just recently set up their website, so if you're curious about anything concerning Retrotoast, or would like to order a copy of Cartesian Chaos, check up on http://www.retrotoast.com/. The publisher, Savage Software, can be found at http://savagesoftware.com.au/

“Thinking with portals”

by Quinton “Q-man” Bronkhorst

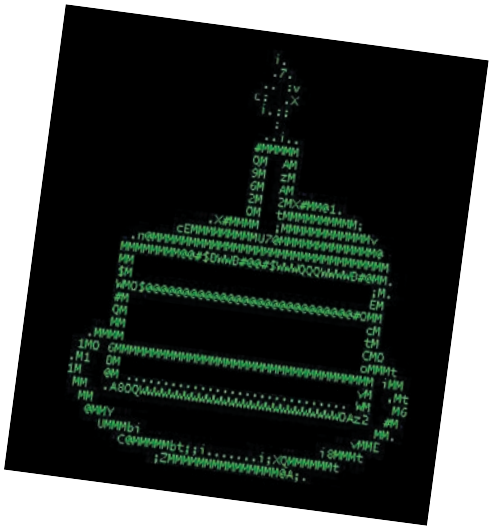
Thinking with Portals is a revolutionary gaming concept. Not only does it open entirely new avenues of gameplay for the average user, but it has sparked a completely new way of thinking for people to engage in. We’ve all opened up the Orange Box and reveled in the pure, untainted awesomeness and ingenuity that is Portal. We’ve been ‘flinging’ ourselves across test chambers, falling into pits of sludge, and incinerating our weighted companion cube friend after it tried to warn us that the cake was a lie. But we didn’t listen. We never listen.

Wait, what?

Moving swiftly along: Portal has opened up a entirely new way of thinking for many of us and, were the portal gun a reality, could make our lives simpler, far more enriching and downright entertaining - apart from the generic (read: filthy) uses of the portal gun, including spying on the ladies’ locker rooms or [insert another suitably risqué use in here].

Let’s try really hard here to get our heads out of the gutter for a few seconds, and imagine how portals could assist us in those every-day, realer than real, situations in life. Like when ninjas attack you in the middle of a lecture - using portals to escape the most agile of people and sneak up on the sneaky to greet them with a decent stab to the eye is far more awesome than the usual shlep of having to dodge shuriken and ninja blades. In a far more unrealistic situation, just think how handy it would be to use portals to get your keys which you so absent-mindedly locked in your car. Or space shuttle.

Endless slides, eternal roller-coasters, quick transport to and from various locations, or even an everlasting free-fall (which you can end with a sudden stop - and subsequently, death). Communication face-to-face over huge distances, interaction [naughty] to [oh dear] over huge distances! Whatever tickles your fancy, the concept of portals stretches your mind enough to realize that the world you’re currently living in sucks on so many levels!
And that it would be far cooler if you were just think



Windows Vista

Open Source Win64

GameBoy Advance

DirectX 10

.NET 2.0

XBox 360 via XNA

Linux

Mac OS X

Nintendo DS

OpenGL 2.1

SDL

Object Pascal has a lot to offer...

www.PascalGameDevelopment.com

Chrome

Free Pascal

Turbo Delphi

Delphi for Win32

Lazarus

THE WHITE CHAMBER

http://www.studiotrophis.com/

by Simon "Tr00jg" de la Rouviere

When I first heard of The White Chamber, I thought it would be some strange scientific action game. It turned out to be strange, and slightly scientific. It also turned out to be one awesome indie adventure game.

The White Chamber was developed by Studio Trophis. It takes place on a space station. It starts rather innocently, with the main character - a woman with huge purple hair - waking up in a coffin without any memory why she would be in there. Okay, it's not quite that innocent, but you will quickly see why.

After realising that you are in a deserted space station, you do what any intrepid adventure gamer would do: explore it fearlessly. You nonchalantly prance around the space station until the first "trippy" sequence. You do not expect it at all! By "trippy", we mean horrific and bloody images.

Then, as quickly as it comes, it vanishes.

So now, Studio Trophis created a perfect atmosphere. You are always scared when re-entering any room for fear of something trippy happening. From there on, the game catapults you on a mind-bending journey into discovering why you don't remember anything and why no-one is alive.

The story is brilliantly conveyed through strategic "video" discs scattered around the space station. You finally discover what it is all about, and it is a real shocker.

The White Chamber is a spectacular example of indies pouring their heart into a game. The graphics are splendidly done, and the story, which is a cross between Solaris and Silent Hill, is gripping.

The music combined with the art creates a haunting atmosphere.

Although the game is a bit short (1-3 hours) and easy, it is done so for a reason. There are only a few rooms in the space station to explore, but they utilise it to their fullest extent. Quality vs Quantity, I always say.

While the game is mainly horrific, there are classic comedic easter eggs in the game. A few of these are revealed at the various endings which you can encounter - eight in total.

The White Chamber clearly shows that it's possible to make a great title with sufficient polish and attention to detail. I recommend anyone, not just adventure gamers, to try this splendid indie title.

SPROUT

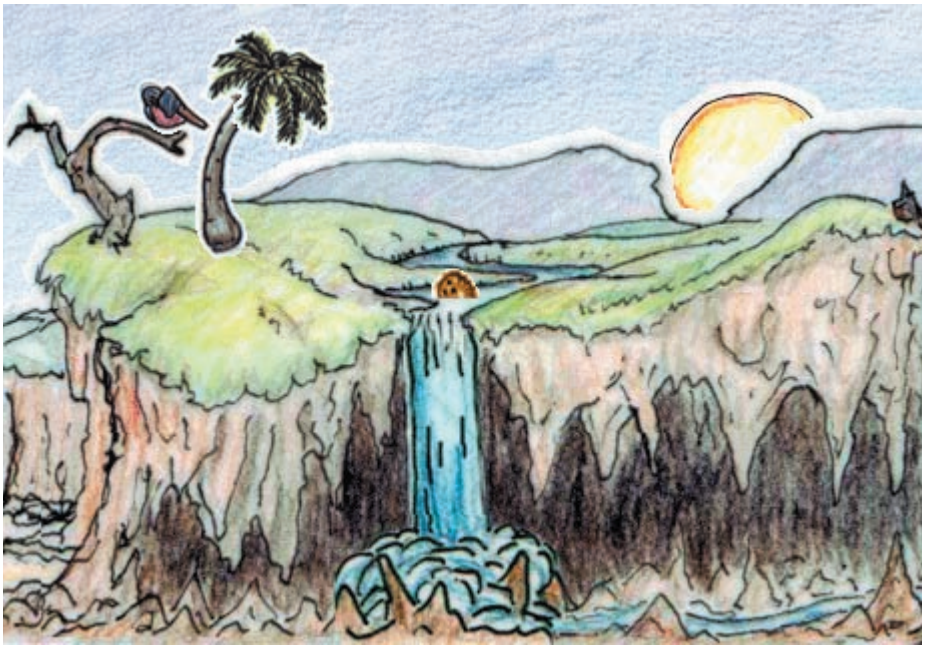
http://www.kongregate.com/games/customlogic/sprout

by Claudio "Chippit" de Sa

Sprout is a highly stylized flash adventure game boasting an interesting and unique premise of guiding an ambitious seedling from its island habitat to a lush oak forest. Sprout's simple gameplay mechanic means it literally has no learning curve at all and is simple to grasp. Yet the game still manages to have a surprising amount of depth despite the fact that the player is never offered more than 4 gameplay choices at any one time.

Sprout features highly stylized hand-drawn graphics which appear rather like a cut-and-paste comic. It also features a distinct variety in locations, each bearing its own challenge to traverse. Scorching deserts, steep ravines, fast-flowing rivers and vast oceans are some of the challenges that await the little seedling.

The play dynamic is truly where the appeal of this game lies. The seedling which the player controls has the ability to 'learn' how to grow into different species as it encounters them. Each variety of plant has a different attribute that allows the seedling to traverse different terrain features. A palm tree, for example, will form a coconut that can float across water or roll down hills, and a vine will allow the seedling to reach the top of a cliff or tree.



As the player progresses, the seedling will encounter 4 different species of flora and, by using each one's unique ability, the player is challenged to reach the lavish woods where the seedling will eventually become the grandest oak of the forest and produce its own seedlings with their own ambitions. Pineapples, anyone?

Although Sprout may not take you more than 20 minutes to complete, it's an incredibly unique experience throughout and requires quite a bit of thought, unlike other similar titles.

FRETS ON FIRE

http://fretsonfire.sourceforge.net/

by James "NightTimeHornets" Etherington-Smith

Frets On Fire, a nifty little open source title from indie developer Sami Kyöstilä, is a tribute to Guitar Heroes (if you haven't heard of this game, you should get out of the cave more, or at least log on to the internet more often). Kyöstilä wanted to bring the console-based Guitar Heroes experience to PC users. For anyone unfamiliar with the basic concept, the gamer is presented with a vertical guitar fret board which scrolls towards the viewer. This fret board will display five different 'notes' which correspond to a key command and are timed with the guitar-driven music. As the fret board scrolls, the gamer will have to hit the correct note combination. There is also the element of 'strumming' which must be done at the same moment a note is hit. The game is all about rhythm and timing, and although a simple concept it is extremely addictive. It becomes quite tricky at higher difficulty levels, requiring a bit of practice (and patience) to get just right.

Frets On Fire (FOF) pretty much emulates the gameplay of Guitar Heroes, albeit without the fancy 3D graphics in the background and none of the licensed musical content. The player can use a keyboard to control the game by holding the keyboard upside down and using the left hand to press keys F1 through F5 to control the five notes; Enter is used by the right hand to 'strum' the virtual guitar. This works quite well if you have a compact keyboard but can be troublesome for people with larger media keyboards, which usually have loads of extra function keys along the top. FOF is also capable of using a wired Guitar Heroes controller, which is picked up as a joystick and requires simple mapping of the keys. The download package includes 3 songs created by the developer.

"What's the use of having a game with only three songs?" you may ask aloud, causing a nearby loved one to become concerned that you have begun talking to yourself again. Well, this is where the extensive user community comes into the picture. Thanks to the multitudes of people working around and supporting FOF, you can download any number of modifications that enhance the interface and game environment, even adding 3D models and backgrounds. The most enticing element of the community is the contribution of user 'fretted' songs. The list of songs modified to work in FOF is already huge and is sure to keep on growing along with the community. This is one area in which FOF is sure to outpace Guitar Heroes. Couple this with the numerous guides and friendly community members that will help you convert your very own favourite tracks to Frets On Fire format, the replay value is obviously tremendous. If that's not enough, there is also a music converter which is able to import tracks from any Guitar Heroes game you already own.

This is a great example of open source development working well. FOF is a superb emulation of its mega successful inspiration. It is rather impressive to see the amount of work going into modifications and, as an ongoing project, FOF seems to be inching its way towards the quality level of Guitar Heroes itself. Whatever the future may hold, one can be sure that one's keyboard will never be the same again.

BLENDER TUTORIAL

Making 3D Buttons

By Stefan "?rman" van der Vyver

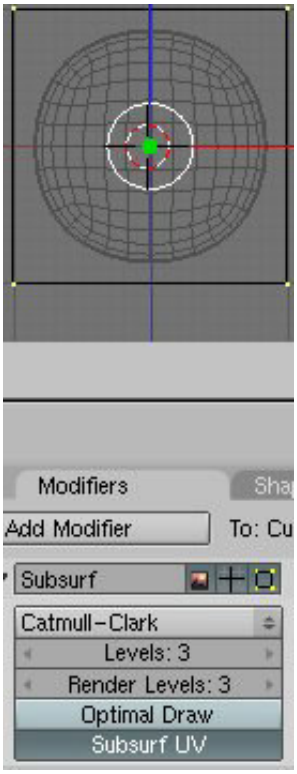
This article refers to resources available at the "Contents" section of the Dev.Mag website (www.devmag.org.za). It is recommended that you visit the site and download these resources.

There are many applications that create customized buttons. We can also use Blender to do this. This allows you to set the shape, size and colours of your buttons exactly. For this tutorial we will build a cube then use a subdivision modifier to divide it enough to make it round. Then we'll build a 'sleeve' for the button using a 'plane' object. Lighting techniques and shadows will help create the illusion of the button being in a pressed or non-pressed state.

Start Blender. Delete all the objects in the scene (AKEY, then XKEY). Ensure that your 3D cursor is in the centre of the screen by pressing SHIFTKEY+CKEY. In the front view (NUMPAD1KEY) press SPACEBARKEY and insert a Cube object. This cube will become the button.

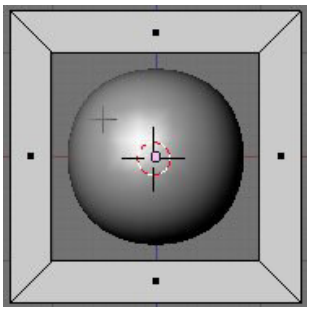
The next step is to go to the editing buttons, and add a subdivision modifier. This will smooth out the corners of the cube forcing it into a round shape. Set the Subdivision levels as shown in the picture below.

While in the editing buttons window, see if you can spot the Set Smooth button. Press the button to see how beautifully smooth the cube becomes.

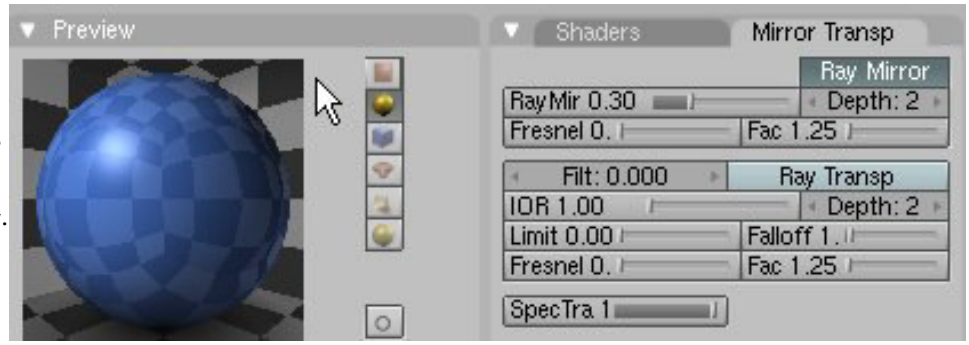


Now it's time to add the sleeve for the button. Exit edit mode with TABKEY, or use the menu at the bottom of the viewport to change into Object Mode. Ensure that your 3D cursor is in the correct spot. Blender uses the 3D cursor as the location for adding new objects. In this case, I want to add a plane right on top of the other cube so that they fit together snugly. Should the 3D cursor not be centred on the cube, select the cube and use SHIFTKEY+SKEY to move the cursor to the selection. With the cursor centred on the cube, use SPACEBARKEY to add a plane.

Change your viewport to wireframe mode, by pressing ZKEY. Don't deselect anything. When adding a new object, all the vertices are selected. Press EKEY to extrude the selected vertices. Left click immediately. It would seem that nothing happened. Don't worry, something did! Now hit SKEY, and scale the new vertices out to the width that you want the sleeve for the button to be. There is a problem with what we just did. We essentially created two planes on top of each other. The test for that is that there is no hole in the middle of our sleeve. Hit ZKEY again to see this. Change to Face mode by hitting CTRLKEY+TABKEY, and selecting Faces. Now select the faces in the middle of the object and delete them XKEY ---> Faces until you can see a hole in the middle of the object.

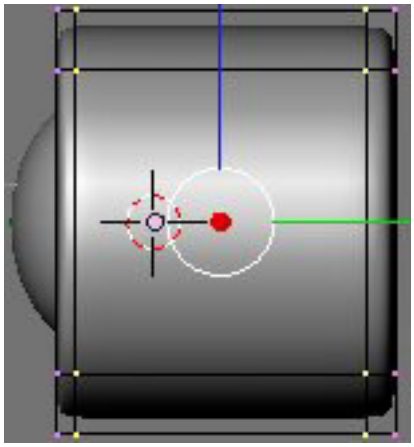


Mirror settings

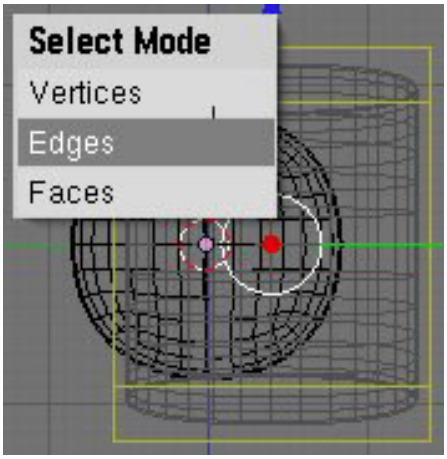


Now we need to change to side-view to position the sleeve and extrude it to give us the right amount of depth. Hit NUMPAD3KEY for side-view. Press AKEY to selects all the faces. Position the faces where you want the front end of the sleeve to be. I choose my location so that the button protrudes slightly. Then hit EKEY and extrude the sleeve along the Y-axis (hit YKEY after hitting EKEY). Add a Subsurf modifier again, using the same method as what we did previously. Set your render levels to 3, and hit Set Smooth.

Stay in side-view and change to edges mode (CTRLKEY+TABKEY). Deselect all the edges then use box select (BKEY) to select all the edges along the middle of the sleeve. Use WKEY to subdivide the edges. Then switch to Vertices mode and select the vertices we just added. Then move almost right to the front of the sleeve. Subdividing the edges and moving the new vertices towards the ends means that we are giving more definition to the ends of the sleeve object. Now we are almost ready to render our buttons!

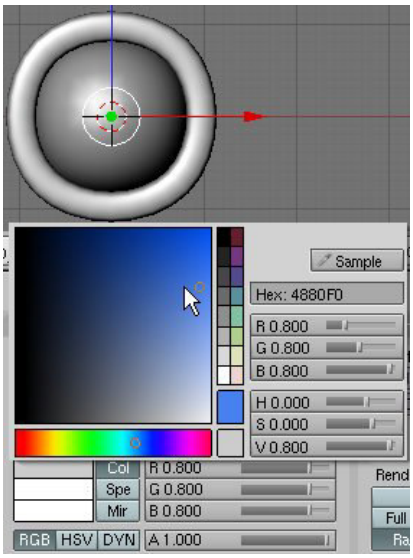


Result after subdivision



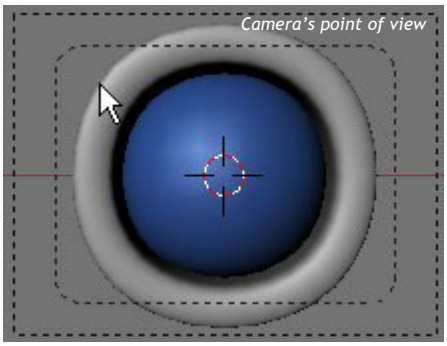
Edges Mode

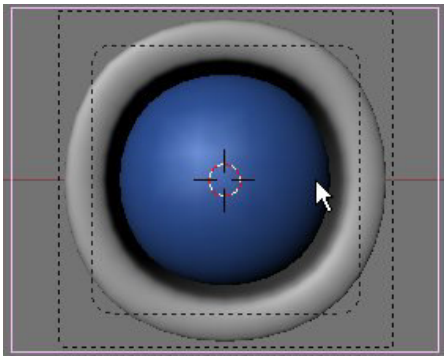
Add a material and colour of your choice to the button. Leave the sleeve gray if you want a metallic type of look. To add a material, select the object you want to add a material to, and click the Shading icon. Click the Add New button. Hit the gray colour bar (top one of the three) and select the colour for your button. I like adding a bit of reflection, so I always go to the Mirror Transparency tab in the Shading buttons panels. Activate the Ray Mirror button if you wish. I set my RayMir value to 0.30. I also added a new material to the sleeve and put a mirror value on that material as well.



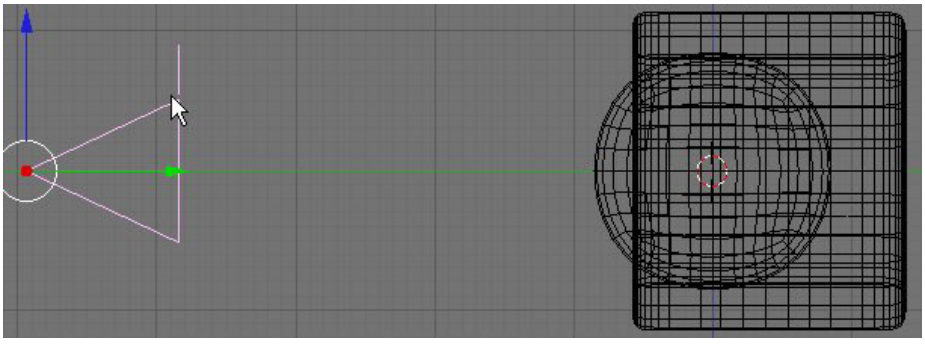
Material Colour

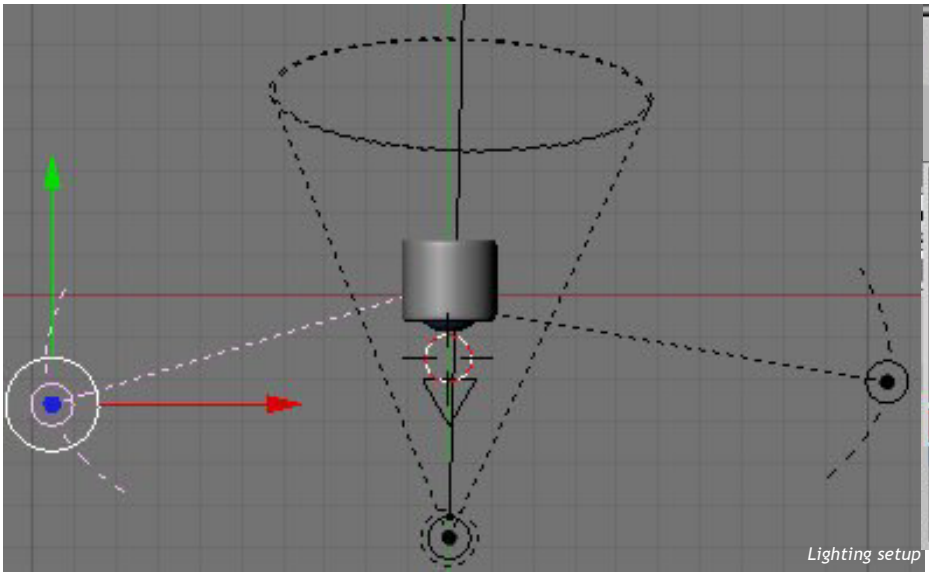
Now, add a camera to your scene (Remember that Blender adds new objects at the 3D cursor position). SPACEBARKEY brings up the menu to add a camera. I then positioned the camera in front of the button, as can be seen in the side view below. Change the viewport to camera view to see the alignment and viewpoint of your camera with the NUMPAD0KEY.



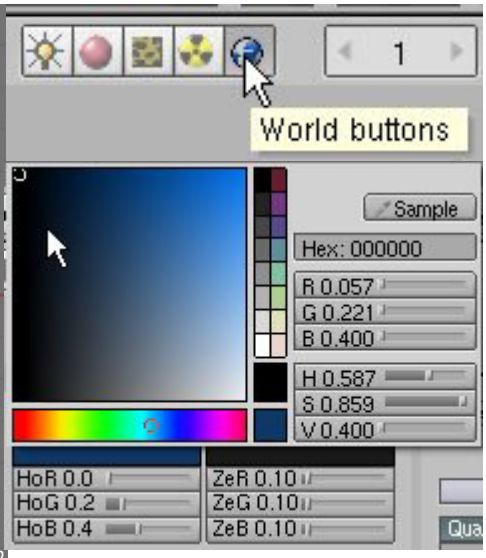


Side view of camera





Lighting setup



World background colour

Go to the Scene panel. We can set the camera to render the smallest possible area around the button, and also to render exactly the right sized image for instant use of the rendered image. The power of Blender lies in the accessibility of these options. In the "Format" tab" you can specify the pixel dimensions of your rendered image. You can also set the aspect ratio. The settings I entered appear above.

I moved the camera again so that the button fills up the whole of the available camera view. That will prevent the need for cropping in an external graphics editor at a later stage.

Now start adding lights (SPACEBARKEY) to your heart's content. The lighting is up to you, since it has to fit your requirements for the button. I used a number of hemisphere lamps to light the model from all sides, and then a single spotlight with shadows switched on to add a key light to the scene. One thing that I always do is change my background rendering colour. You can do this by going to the World buttons, and changing the blue colour on the left to a pure black.

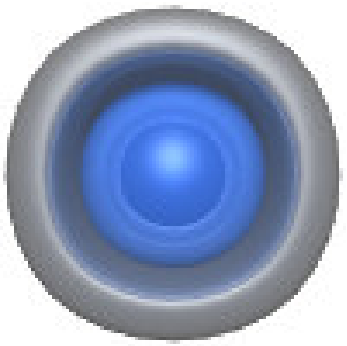
Firstly, render (F12KEY) an image with the button in the out position. Now, move the button backwards (y-axis) in the sleeve so that it looks like it has been pressed in. To get the button pressed effect, I go back to the Shading window (F5KEY). In the Shaders Tab, enter a value next to the Emit slider. This makes the colour of the button glow. I also add a blue light of the Lamp type in front of the button, to light the edges of the sleeve object. Now render the scene again.

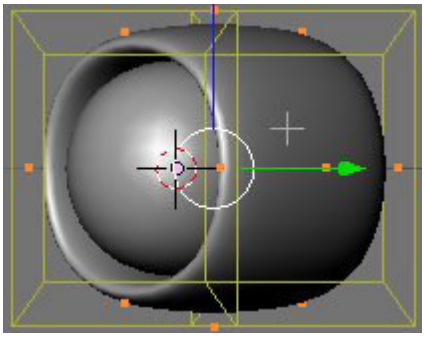
Setting Emit value

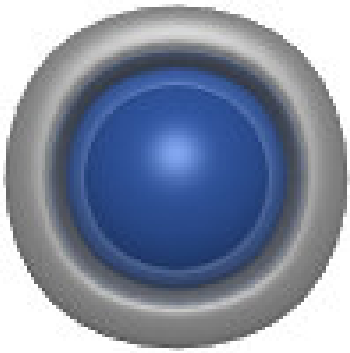
File format settings

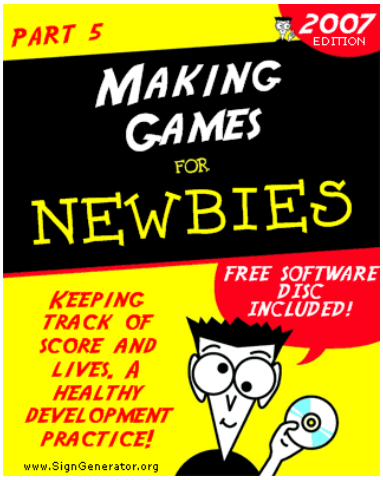
If you need transparency for the background, ensure that the RGBA button is on in the Scene panel (F10KEY). Also make sure that you have chosen either TGA or PNG as render format.

The blender file for this tutorial is included as well and should help illustrate what was presented in this tutorial. Good luck with your Blender designs. I sincerely hope that you will be able to use Blender effectively as a tool to increase the options you have for creating graphics for your games.









SCORES AND LIVES

Beginner's Guide to Making Games

By William " Cairnswm" Cairns

This article refers to resources available at the "Contents" section of the Dev.Mag website (www.devmag.org.za). It is recommended that you visit the site and download these resources.

Welcome to the next instalment in the Beginners Guide to Making Games. Each month a single important concept required for making games will be discussed in detail. This month we are looking at Score/Health and Lives. Most games need one or more of these concepts in them to make them complete games.

The main goal of the Beginners Guide series is to try and ensure a detailed understanding of the various concepts so that they can be applied to other new and exciting games. While most traditional tutorials will show what logic is needed, these guides will ensure that you walk away actually understanding each of these concepts.

This article is aimed at someone who has just started learning to make games. While it is expected that the reader of the article has completed the first Game Maker tutorial and can thus create sprites and objects it is quite possible to follow the article without having done so. The article is structured to introduce a new programmer to the concept but yet give some value to the intermediate level programmer as well.

Score - GM Tutorial

The vast majority of games require some sort of scoring system within the game. Many games also require a system to manage the number of lives the player has left, and some games have effects that eat away at the health of a player until the player dies. Game Maker has global functions to manage all these tasks.

This article will contain the following sections:

- A brief Game maker tutorial showing the basics of getting a scoring system working.
- This is followed by a discussion, in detail, of the various Game Maker actions available to you to manipulate the score, health and lives of the player.
- Lastly, a look at various Game Maker Language (GML) functions that can be used to affect the score, health and life of the player, as well as some options for making each object have its own health.

Game Maker Tutorial

To show how easy it is to create a game that controls the number of lives a player has we are going to make a little game where the player needs to continually bounce a ball against a wall. If the player misses the ball he will lose a life. Each time the player hits the ball we can allocate him a score.

This tutorial is not supposed to be a complete game, but is designed to show only the specific function of managing a players life and score in a game context. This game is almost the basis of a small Arkanoid game, and the fun part is that Game Maker actually comes with all the sprites needed to complete an Arkanoid game.

Step 1 - Create Sprites

For this tutorial we need sprites to represent a ball, a wall and a bat, nice sprites for these objects can be found in the Breakout images directly that comes with Game Maker. If you don't know how to create sprites in Game Maker yet I suggest you refer to the first article in the Beginners series where we learnt about how to move an object around the screen.

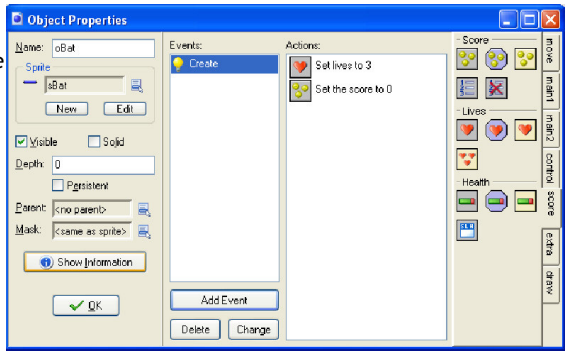
Step 2 - Making the Objects

Now create an object for each of the sprites we have created. Remember to make the wall and bat solid as we want the ball to bounce off them.

Create the following events:

- Within the On Create event of the ball make it move in a diagonal up direction.
- Make a Collision event on the ball, so that when it collides with the wall it bounces off the wall.
- Add movement controls to move the bat left and right with the arrow keys. In this sort of game I prefer to only move the bat when the key is actually held down.

Now for the meaningful events in this tutorial. On the Bat object we need to create a Create event where we can initialise the score and lives values. In the actions of the Create event place a "Set the Number of Lives" action and set the value to three. This means that in the game the player may only drop the ball three times. Also add a "Set the Score" action and give it an initial value of 0.



Object creation settings

In the ball object we need to create a Collision event with the bat. In this event we will bounce the ball and add some score.

Also in the ball object we want to make sure the player loses a life when the ball is missed. For this Game Maker supplies a nice event call “Outside Room” with is an option in the Others event type. When the ball leaves the room we need to subtract a life, and move the ball back to where it started. To ensure the game does not go on for ever we could also speed up the ball a little.

Of course we need to ensure that the game ends when the player has no more lives. Again Game Maker makes this easy with its “No more lives” event. In this event we can just put an “End the game” action to finish off the game.

Step 3 - Adding our objects to a room

Create a room and place walls all around the edge except along the bottom. Place the bat near the bottom of the room and the ball somewhere in the middle. When we run the game the ball bounces neatly around and bounces off the bat. You'll notice that the first time the ball hits the bat the title bar suddenly starts showing the score.

Step 4 - We want to see Lives/ Score

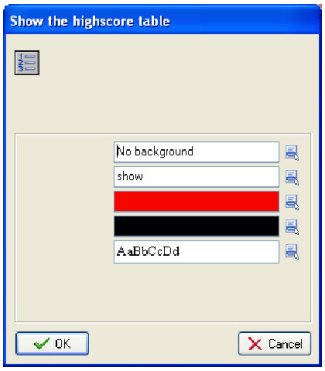
Obviously we want the player the be able to see their score and lives all the time. The easiest way of doing this is to define a “Controller” for the game that displays the Score and Lives in its draw function.

More Stuff with Actions

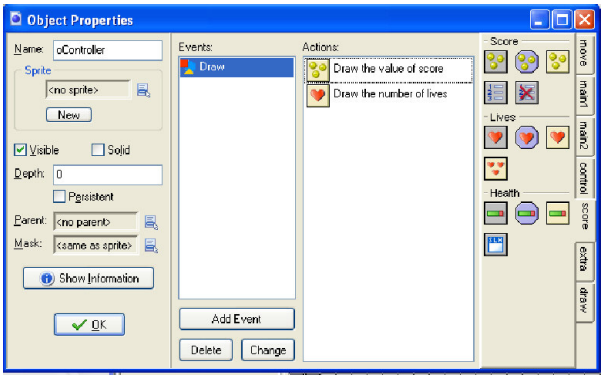
Things like managing a player’s score and lives is actually very easy, and Game Maker makes it even easier than that. Game Maker contains a number of actions that work with Score Lives and Health. Game Maker even supplies a fully working High Scores screen that can be linked into the game with no more effort than dropping an action into the “No more lives” event.

Health works slightly differently to Lives and Score as it has a predefined low and high values. Health always varies between 0 and 100. Game Maker supplies an event called “No more health” that triggers when health is set to 0. This can be used for things like people shooting at the player and different bullets cause different damage etc.

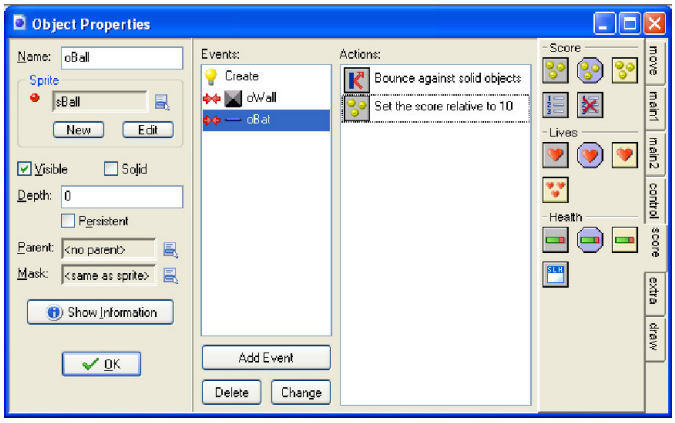
Because health ranges from 0 to 100 it becomes tricky to display in an easy manner. Game Maker supplies a “Draw the Health bar” action that draws a neat little coloured bar showing the current value of Health. The health bar is a nice green colour when it is full (near 100) and degrades to red to indicate how close the player is to having lost all their health.



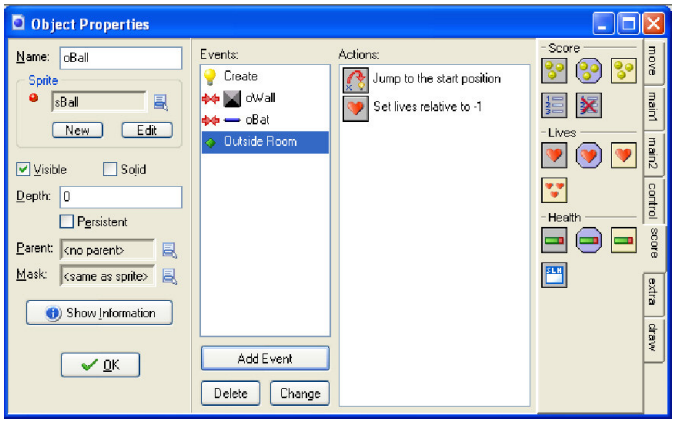
Controller object



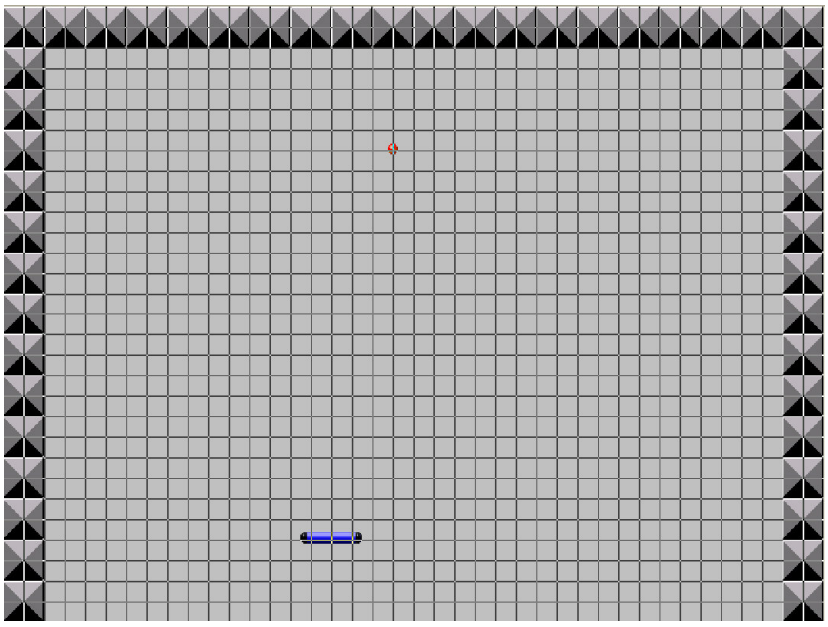
Collision events



Outside room event



Our room



To allow a player to capture their High Score into a table just drop the "Show the highscore table" as an action before ending the game. This will pop up the high score table and allow the player to enter their name into the table (if they have a high score), and only then exit the game. Most games would obviously return to a menu screen rather than just exiting the game.

Game maker includes an action to clear the highscores table of the existing values that can be very useful as part of a Menu or Options screen.

While text is usually good enough to display score most games prefer to display a players lives using little pictures instead. Game maker allows you to do this with the "Draw the lives as Images" action.

Lives and things with GML

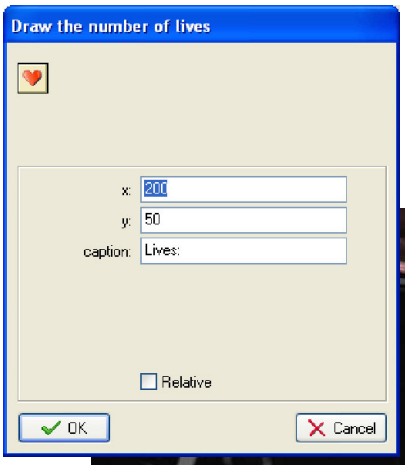
Always remember that each action you can use is mirrored as one or more GML statements. So if you prefer using GML for your game you can do each of the actions listed above directly in code. Lives, Health and Score are global variables that can be used directly in GML. GML also gives you the option of changing the caption in the title bar to your own values (Internationalisation anyone?)

Here are the basic global variables and settings you can use:

score The current score.
lives Number of lives.
health The current health (0-100).
show_score Whether to show the score in the window caption.
show_lives Whether to show the number of lives in the window caption.
show_health Whether to show the health in the window caption.
caption_score The caption used for the score.
caption_lives The caption used for the number of lives.
caption_health The caption used for the health.

Another cool thing in GML is the ability to display a health bar anywhere on the screen. So if you want enemies that had their own health you could create a local variable on a certain object called OwnHealth and then in the objects draw function, display the sprite as well as calling the draw_healthbar method to draw their own health, this is typically what we see in RTS games where each unit has their own miniature health bar.

draw_healthbar(x1,y1,x2,y2,amount,backcol,mincol,maxcol,direction,showback,showborder) With this function you can draw a health bar (or any other bar that indicates some value, like e.g. the damage). The arguments x1, y1, x2 and y2 indicate the total area for the bar. amount indicates the percentage of the bar that must be filled (must lie between 0 and 100). backcol is the colour of the background for the bar. mincol and maxcol indicate the colour when the amount is 0 and 100 respectively. For an amount in between the colour is interpolated. So you can easily make a bar that goes e.g. from green to red. The direction is the direction in which the bar is drawn. 0 indicates that the bar is anchored at the left, 1 at the right, 2 at the top and 3 at the bottom. Finally showback indicates whether a background box must be shown and showborder indicated whether the box and bar should have a black border line.



GAME GRAPHICS DESIGN

Part 2: Basic Vector Sprites

By Rishal "TheUntouchableOne" Hurbans

This article refers to resources available at the "Contents" section of the Dev.Mag website (www.devmag.org.za). It is recommended that you visit the site and download these resources.

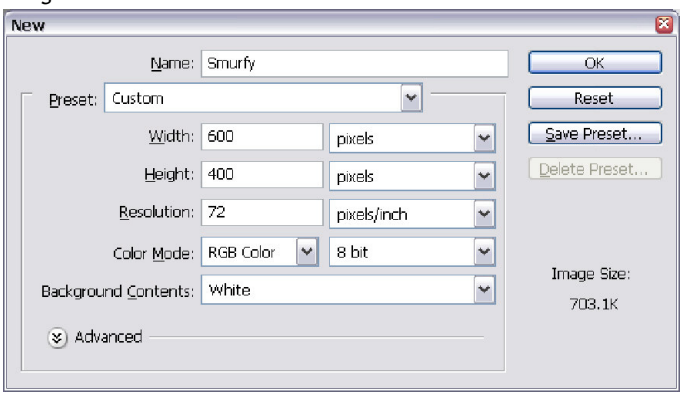
Welcome to the tutorial on basic vector sprites. In this tutorial you will learn how to create a very basic vector sprite that can be used in a game and even animated at a later stage. Some hobbyists use tile sets to create their game's graphics. A tile set is a number of images that contribute to the graphics of the game such as patches of grass, walls, enemies, etc. The sprites we create can be scaled and used in a tile set. We will look further into tile sets in the future.

What's a Vector Image? A vector image is an image created using math in the "background" (you don't need to apply any math so don't worry). Plotting points on the canvas creates the shapes; these points collectively create a path. A vector image is more useful for creating sprites as the image is sharper and cleaner cut. It doesn't store a colour in each pixel to create the image as bitmaps do. A vector image can be resized without any loss of image quality; this makes it ideal for game sprites as we may need to scale the image for different uses. When creating vector images we will make excessive use of the pen tool. The other tools will be used in the future.

Let's get started! The first thing we need is a canvas to work on so create a new image by pressing Ctrl+N and enter the details as seen in figure 1.

Spites are usually small images but we are using a 600x400 canvas to fit all the views of our character in the image and to make it more comfortable to draw the sprite without worrying about size restrictions. The different views can be copied to a smaller canvas and scaled to have several different sprites of the character that can be used in a game.

Figure 1



I was thinking of a fun and basic character to create for a first time user and thus "Smurfy" was born - he is part alien, part Smurf. We will start with a side view of the character, as we would see it in a platform game.

Since we are creating the character from scratch it would be rather difficult to create an outline of the entire character from head to toe, so we are going to create each body part separately. Firstly, our character needs a head. Select the pen tool (P). Now hold the "Alt" key while you click on the canvas, each click will create a point in the path. Create a shape that resembles the shape of a head, you don't need to worry about making it perfect the first time as we can adjust this. Your side view of the character's head should now look something like figure 2 below.

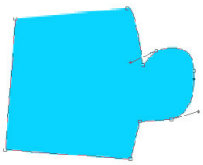


Figure 2

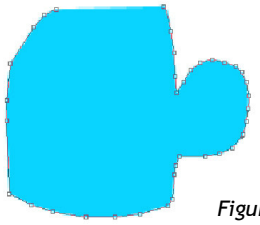
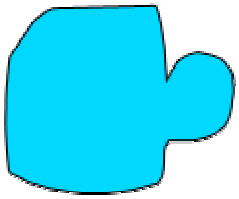


Figure 3

> ***Note:***
> If you have trouble creating shapes with the pen tool, just experiment with it by creating different shapes until you become comfortable with its use.

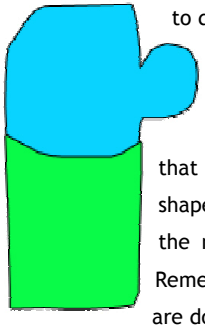
We now need to add some smoothness to the character's head; this can be achieved by adding, deleting and moving points of the path. For this you will need the "Convert point tool", this is a tool in the "Pen tool" drop down list. Look for areas on the path that needs more shape to it or areas that are too sharp and need smoothness. You can add more points as you see fit, though the fewer the better. Add a new point by right-clicking and selecting "Add Anchor Point". If there are unnecessary points, you can delete them by right clicking on the point and selecting "Delete Anchor Point". The best way to achieve the shape you want would be to move the points into the correct positions. Hold the "Ctlr" key, click on the point you want moved and drag it to the position you desire. You have full control over each point as you can change any aspect of it to suit your sprite. After playing around with the points, you should have a shape that looks similar to figure 3.

Once you are finished with a shape, it would be a good idea to create an outline for it. This makes the shape look a lot better and helps you distinguish between the different parts of the character's form. You can apply an outline by right-clicking on the shape's layer in the layers palette, selecting "Blending Options" and checking the "stroke" checkbox. Then click on the stroke label and enter the details as shown in figure 4. Your result should look like the image above.



There we go. We now have a head for our character. I think he would appreciate a body too. For this character, we are aiming at a wacky, cartooned style so you can go wild with creativity. We use the same method as we used for the head to create the torso. Plot the basic points and then alter them by adding, deleting and moving points. Before you start with the points, select a different colour in the foreground colour palette as it is better to distinguish between the different shapes and a colourful character is usually an attractive one. The image should now look similar to that in figure 5, it is a very basic shape and doesn't look like much at the moment, but it gets better. Remember to add an outline after you are done plotting the body. It would also be a good idea to start naming the different layers respectively in the layers palette, simply right-click on the layer's name and type the appropriate name.

Figure 5



The next part we will create is the character's arms. Plot a path for the right arm as shown below in figure 6 remember, you can always manipulate the points using the "Convert Point Tool". Use the same colour as the torso for the arm.

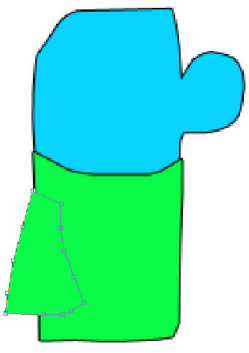


Figure 6

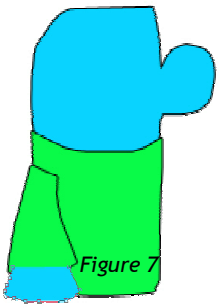


Figure 7

Now the character needs a hand. Since he is an alien and you probably would find it difficult to plot a real looking hand, we will give him an alien type of hand. Plot a shape that looks like a hand of sort as shown in the image to the right. Don't worry about plotting the hand over the arm layer, we will move the hand layer below the arm layer to fix this, just make sure you have a smooth flowing shape. Your image should now look similar to figure 7.

The hand layer can be moved behind the arm layer by moving it in the layers palette. Just click and drag the hand layer so that it is positioned before the arm layer. The outline can also be added now.

We can't have a character with one arm and we definitely can't waste time drawing another arm and hand shape so we will simply copy the current arm and hand shapes. Do this by, selecting the arm and hand shapes in the layer palette (hold Ctrl and click on the layers). Then right-click and select "Duplicate Layers", this will make a copy of the layers and they will be selected automatically. Use the move tool (V) to move the two layers to the right of the canvas; you will now see the duplicated layers. To put the character into perspective, we need to make the left arm and hand layers smaller than the right side as it is slightly further back. While the two layers are selected, click Edit->Transform->Scale. You can now scale the arm and hand slightly smaller as seen in figure 8 alongside.

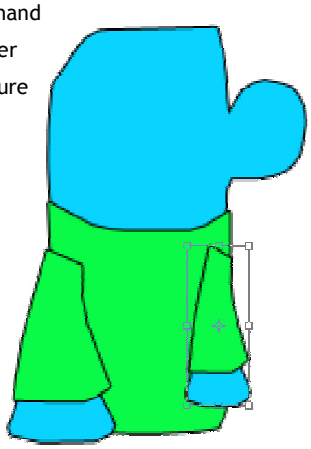


Figure 8

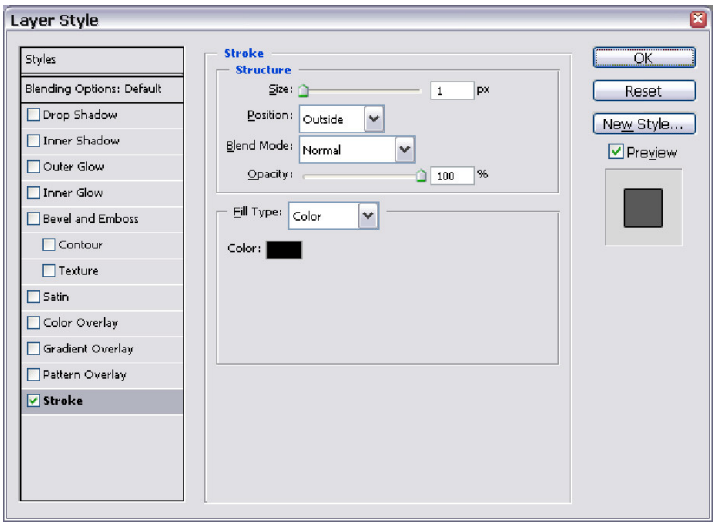
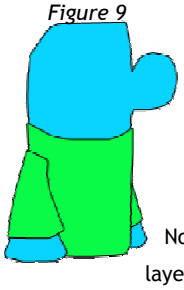


Figure 4

Figure 9



> ***Note:***
> In the transform menu there are a variety of tools that can be used to manipulate the image. It would be useful to browse through them after the tutorial as they can be useful in animating sprites.

Now simply move the two layers behind the body layer as we did earlier in the layers palette. It should look similar to figure 9

The character looks slightly better with his newly-acquired arms. We will now add some pants to the character. We are going to make the character rather short so defined legs are not needed. Select a new colour and create a basic squared type of shape for the pants, we will use some blending options later to add some detail. The plotted shape should look as shown in figure 10.

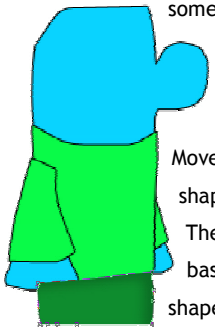
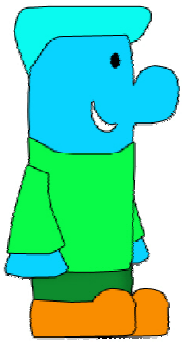


Figure 10

Move the pants behind the torso shape and add an outline to it. The character still looks very basic but it is taking some shape. The character now needs some shoes; we are giving him large clown shoes just to add to the wackiness. Choose a different colour and plot a shoe for the character using the pen tool. Duplicate the shoe as you were shown earlier and move the layer behind the right shoe's layer, then move it a little to the right to show just the front of the shoe.

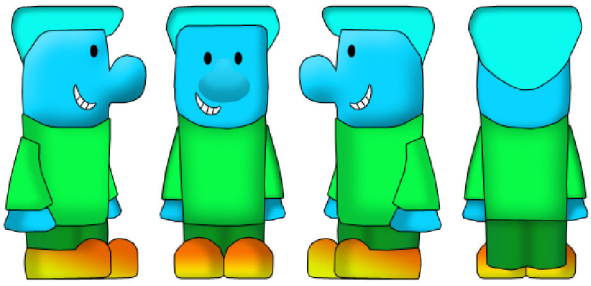
Our character is almost complete, but he seems to be complaining about the fact that he doesn't have any hair, eyes or mouth. Lets select a different colour and plot a shape that resembles hair (you can choose any hair style you want - this is a military-cut gone bad).

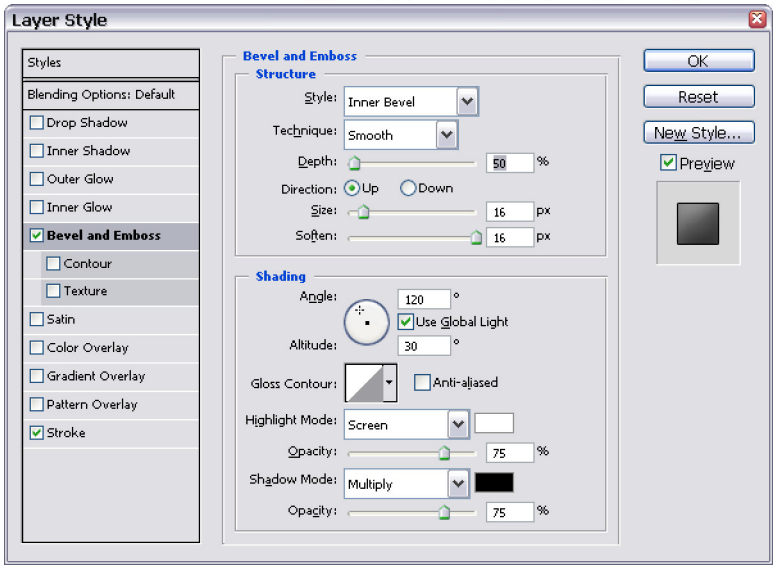
The character also needs sight! Select black as the foreground colour in the colour palette then select the ellipse shape tool and draw a small eye for our character. You can now plot a small mouth using white as the foreground colour in the colour palette.



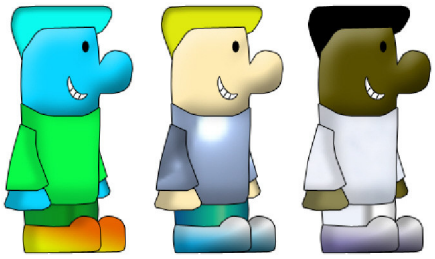
We now have our character. It is a very basic character with basic colours, so we need to embellish it. We want to keep it a vector image so the best way will be to use the various blending options for each body-part. We can add different textures, gradients and shadows to give the character depth. Select a layer in the layers palette and experiment with the different effects, a simple effect applied well can add a lot to the sprite's over-all appearance. Once the sprite is complete you can add a Fuchsia coloured background, chop off the unused blank space and save the file as any format that can be used in your games. The effect shown below was applied to the head layer.

I created a range of views of the character and applied all the shading shadow, textures etc. to make them look a little bit better. It is useful to create multiple views because they are needed when creating a game; you can't have a character walking right but facing left, so you change the sprite to the games needs. A gradient was also applied to the pants to give the effect of legs. It just goes to show how useful simple techniques can be.

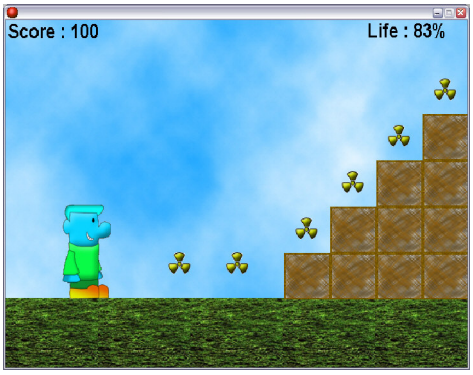




The blending options allow a variety of effects to be created and this allows you to unleash your creativity. Here are a couple different effects I used on the same sprite.



That's it for this tutorial, I challenge you to start creating your own unique, imaginative sprites or maybe just finish the rest of the views of this sprite, just to make sure you got the hang of it. If you need this example of "Smurfy" you can grab the photoshop project file at the link below. Good Luck!
(www.devmag.org.za/uploads/Smurfy.psd)



Above is an example of what this sprite could look like in a game made with Game Maker.

Racing Away with Mini#37

A game development postmortem

by local studio Luma

Local design studio and game developers Luma have already been featured in Dev.Mag for their endeavours. Although the company's gaming division, LumaArcade, has already moved on to other exciting products, their campaign to showcase Mini#37 is still going strong, and report the 2007 rAge expo as a huge success for the product.

Mini#37 is a single/multiplayer racing game which has players racing around real South African environs — including as the streets of Cape Town and the Golden Mile at Durban — armed with some mean Minis and an arcade game dynamic which reflects that of popular titles such as the Need for Speed series.

The development team graciously decided to enlighten us about the creation process and give Dev.Mag's readers a better idea of what goes into a game like this. If you don't already know about Luma and would like to know more about them and their creations, take a look at http://www.luma.co.za/.

Dale Best

What I did: >> Creative Director / 2D artist / GUI / texturing / Producer / Client liaison

When looking back at our design document — and seeing how Episode 01 and 02 have turned out — I have to say we pretty much hit all our objectives, and there really has been nothing extra that crept in along the way. Our goals up front were quite clear, and a few "nice to haves" on the wish-list ended up being executed. One area of game play, however, was not possible to do to its fullest extent, because of restrictions imposed on us by the dreaded DIF format, which Torque uses for large world objects like floor areas and buildings.

One of the areas I championed up front was a strong multiplayer aspect. We went down to the Zone in Rosebank, and played Daytona USA at the arcade. The experience we had at rAge this year proved that we had fulfilled this objective.

The game was played in excess of 2000 times over the 3 days. I wanted a group of friends to jump on, have fun, and then walk away with smiles on their faces. This happened.

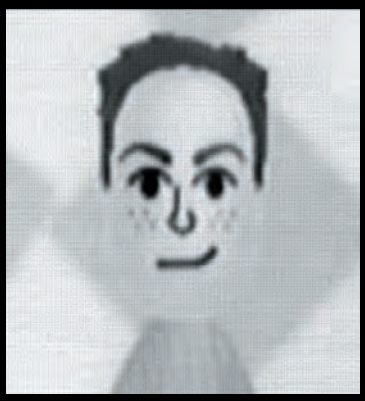
One thing we decided to do was sneak in 3 additional tracks (Durban) from Episode 02 which we had not finished completely into the demo version we had up at rAge. These tracks contained a new aspect of the game which we had just recently incorporated,

namely the loops. By watching people play theses new tracks, we identified a weakness in the design regarding the lack of alternative routes around these loops. You can't playtest enough, ever!

When we set out to make MINI#37, I wanted to make a game that ANYONE could play. At the same time, a more experienced game player still needed to be challenged and learn how to power slide for example, or simply drive well - finding the 'zone', which gives this game its distinct personality.

As a starting point, from a visual point of view, we took some cues from Need for Speed Underground. The great arcade checkpoint style of Outrun was an inspiration for gameplay. The relatively simple track layouts made from city blocks in the original PGR on XBox helped us lay out a methodology for the track design.

Additional game play elements like the world ranking leader board, which is to be published on the website, all ended up in the final version. Our initial objectives were pretty straight forward, our intention was never to reinvent the wheel for this project, we wanted a robust, fun to play, and simple racing game, that looked good, played well, and had MINI cars in South African cities.



Luke Lamothe

What I did: >> Technical Director

When work began on MINI#37, the first task that us programmers had to face was to decide on what technology we would use in order to bring our vision of the game to life. Seeing as we had such a short turnaround time available to us before we had to deliver the first public demo of the game (3 months!), we immediately took the stance that we would have to make use of a 3rd party engine instead of creating our own technology from scratch.

Given the kind of technology that was necessary for us to licence (specifically vehicle physics and networked multi-player support), combined with the budget that we had access to for our development, it quickly became clear that the best engine for the job would be the Torque Game Engine.

At first, TGE seemed to be a blessing from heaven as it contained support for everything that we needed in order to get this project done. It had fully featured world building and editing with both meshes and terrain, fairly comprehensive mesh support with LODs amongst other things, lightmapping, a GUI system, an audio manager, vehicle dynamics, and most importantly for us it was completely built around a networked architecture as multi-player was going to be a very important aspect of our game. Our initial work with the engine was very promising, but we soon began to run into both minor and major issues with it as soon as our learning curve began to straighten out.

Firstly, TGE contains a numerous amount of bugs in it. Most are not show-stoppers and only creep out in certain circumstances, but there are a few contained within the engine that caused us serious trouble. Most of these had to do with the art pipeline and the creation of world meshes for TGE, which use what are known as MAP files. MAP files define convex brush meshes which are used to generate a world in BSP format, and the

MINI#37 » CONCEPT ART AND FINAL EXECUTION.

ORIGINAL LOOK AND FEEL BOARD

FINAL LOOK AND FEEL

tool that comes with TGE that converts these files into the native format that the engine uses has an enormous amount of bugs in it. Secondly, we ran into serious issues with the vehicle dynamics, specifically with the collision detection and reaction systems.

As we eventually found out, these issues have been an ongoing problem in Torque since it was first released and have never really been solved in a satisfactory manner by the developers. We ended up making many changes to how the vehicle physics are processed, as well as completely re-writing the collision system to use a different method that wasn't based upon convex meshes (as all collision detection is in TGE). Also in respect to vehicles, we found that their networking support was less than stellar, and there were numerous changes that we had to make to the vehicle logic in order to reduce the warping effect of vehicles, as well as to fix how collisions worked in a networked environment.

In addition to the bugs that are present in TGE, there are also a lot of features missing from the engine as well as a lot of limitations. We had to add our own support for what should be basic effects such as cubemapping, alpha testing, and mip-map LOD bias settings. On the limitation front, TGE unfortunately still uses very old technology with respect to how its world meshes are rendered. Essentially, the world meshes are stored in a BSP format and are iterated through during each rendering frame, while the engine dynamically inserts the visible polygons into a vertex buffer for rendering. This causes a severe CPU overhead which effectually limits the number of polygons you can have on screen at a time. This meant that we ended up limiting ourselves to having no more than 20 000 polygons visible at once without any cars in view, which would approach 40 000 polygons when all 4 cars were on screen, which is quite a pittance when compared to the number of polygons that most modern games are able to push out!

Despite all of the negativity that we have developed towards Torque, looking back we definitely wouldn't have done anything differently. TGE was a very inexpensive option that allowed us to get a functional PC game up and running within a few months and fully completed in less than a year from start to finish.

David Baxter

What I did: >> Lead Artist / set up art asset pipeline / world building / models / textures

Torque Game Engine nearly ushered in my retirement from developing games after seven odd blissful years. If it weren't for the exceptionally talented and creative people I worked with, I would be a DTP monkey (no offence) or a waiter somewhere right now. They pulled me through. I will be forever grateful to them. After that statement and because I have been allotted only a couple of paragraphs to — quite obviously — vent, I won't go into too much detail and proceed immediately with the "Y" incision.

If you have zero budget, all the time in the world, like trolling around in forums and are an avid hobbyist, TGE (Torque Game Engine) is and should be your first choice, in my humble opinion. Good luck to you, welcome to developing games, collect your name tag at the door, the buffet awaits you.

However, if you have a vision, design ethic, timeline (consequently deadlines) and a client/publisher to report to, and are by definition a professional game developer, steer way clear of TGE. Do not believe anything an engine purports at being able to do, ever. The proof is definitely in the pudding. Take the time to get to grips with the likely candidates for your final game engine. The time spent on doing just this will expose you to their secretive inner workings, supplied documentation and the community that holds them all together. Once you are happy with all these factors, and have convinced your programmers that you have a winner (everyone must be on board), then full steam ahead!

Herman Tulleken

What I did: >> Programmer / GUI / AI

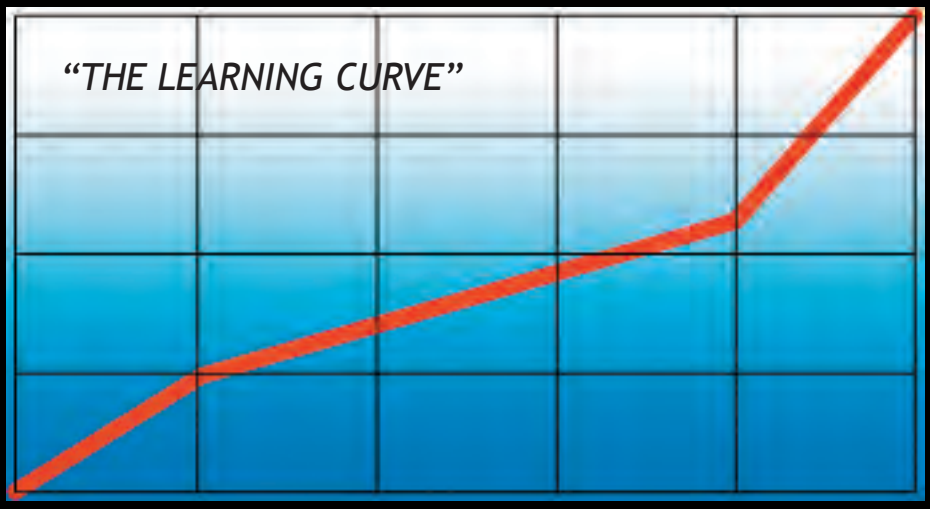
The GUI provided fertile soil for new features and bugs alike. The GUI was expanded considerably over the course of the project, not just to add more features, but also to give more information to the player. We also made many changes after play tests to ensure the information we wanted to convey was visible and understandable.

An important technical design goal for MINI#37 was to make the game very moddable - just drop some assets and setup files in a folder, and a new colour, car, or level would automatically appear in the menus and work in the game. This goal had a big impact on constructing the GUI. We could not use the built-in GUI editor (it is geared towards very static GUIs), and widgets had to be designed to work in very general conditions. Ironically, because of the complex dependencies on assets, "dropping a few files in a folder" turned out to be a tricky, error-prone task (they had to be named and placed very specifically), and it is questionable if we saved time by not hard-coding everything.

One of the biggest problems we faced was making the GUI robust for different resolutions. Its solution required many changes to engine, including changes to the font system, GUI animation speeds, and new layout modes. Because the changes are so scattered, resolution bugs kept appearing, and testing any change across all resolutions soon became standard procedure.

Developing the racing AI drivers was very rewarding, even though at times their (AI) stupidity drove me up the wall: Why was he just sitting there? Why is she (yes, some drivers were female) trying to drive through the wall? Why is he circling endlessly around a check node? A steering indicator, implemented before even starting with the AI code, allowed us to see where the AI driver thought it should be going, and where it detects collidable objects which proved invaluable in debugging the AI.

It became clear that the best algorithms in the world meant nothing if you couldn't get sufficient data regarding the path to your agent. The AI went through three major versions, each change motivated by a new tool that allowed us to get better path data into the game. The final AI employs a state machine, which makes it easy to add driving behaviors. To create paths for this version, we used Diorgo's 2D level editor TuDee, which he modified to support paths and custom node data. This allowed us to specify AI behavior very precisely, including optimal speed, when to use nitrous and handbrake, and whether a special driving behavior should be used, as is necessary for loops and narrow passages.



Chris Cunnington

What I did: >> 3D Artist / World building / modeling / texturing

In the early days we had high expectations for the look of the game. Curving roads, specular on the tar, cambered corners were all on the list as thing we wanted to have. It was at this point that we began producing assets, simple things like the trees and roads. Static meshes (DTS format) worked out perfectly, and it wasn't long before we had trees, lamps, road signs, etc exported and ready for the game.

It didn't take long before we hit a major snag in the DIF format. (For those who don't know Torque, the DIF format is basically designed for modeling interior objects and is based on BSP technology). The DIF technology began firstly by simply being a nightmare to export from 3D Studio Max and convert into the DIF format. Texture coordinates were unstable, polygons would sometime not render or simply not retain collision information. After months of research the pipeline became reasonably stable as long as we worked within extremely limiting boundaries. Those boundaries included being forced to use cubes to model and not standard mesh modeling methods, keeping faces on all polygons planar, no UVW Unwrapping was allowed, being unable to skew textures, etc. The list is quite extensive. We made do with what we had, and actually found workarounds for most of the major downfalls.

Each location in the game was scouted out on Google Earth and track design was based on real life locations. After choosing the routes (all around 2km's in length) we traveled to every location on a photo shoot. The photo shoots, gave us all the texture reference we needed for re-creating the worlds in the game, it also allowed us to shoot panoramic images that were then used in the production of the skyboxes and some photos ended up in the GUI designs.

As time went on, modding the engine and writing tools allowed us to take every new track to the next level. We started playing with curved surfaces which resulted in the loops in Episode#2. We also began using static meshes more often and using hidden DIF collision meshes together. This allowed a marked visual improvement for specific world items, such as the tunnel in the Barbican and shadows/skid marks in the day light levels.

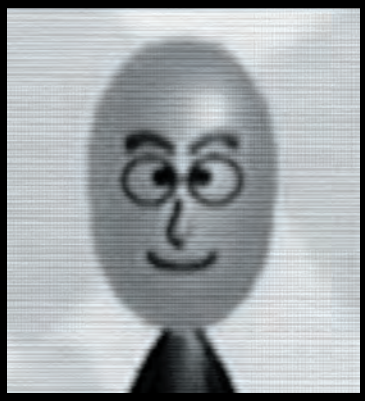
Not having specular or bump maps became something that we had to settle with. Texturing had to be stepped up a notch and had to be done quite carefully, nearly all lighting details, and simulated 'bumps' ended up being included in the textures.

MINI 37 » EPISODE 02: SNEAK PEAK (SOME TRACKS)

DURBAN: ESPLANADE

DURBAN: I.C.C.

SANDTON: MANDELA SQUARE



Diorgo Jonkers

What I did: >> Programmer / Testing / Tools

In July this year, I started work at Luma, primarily to make mobile games. So I only arrived at the final stages of the MINI#37 game development cycle for Episodes 01 and 02. I did play testing and worked on an editor to place the A.I. paths for the game. Play testing was a fairly easy process, since most of the bugs had been sorted out and features added by the time I started work at Luma. I played the game and made notes on any obvious graphical issues, unusual or suspicious A.I. behavior, and general suggestions to make the game more polished or the interface more user-friendly. Occasionally I took some screen shots and made notes on the screen shots. The most fun aspect was the multi-player tests we did. We had the game installed on four rigs (a monitor, steering wheel and comfy seat) and had good fun.

Herman (one of the programmers working on MINI#37) was talking about the process he and Chris (one of the artists working on MINI#37) were using to set up the paths for the A.I. vehicles. The splines were created in a graphics program (Inkscape) and no properties could be applied to the splines. I showed him a level editor that I was busy working on at home, called TuDee. Splines were one of the features I was planning to add to TuDee, so I decided to add the splines before doing anything else. Herman gave me a few features he wanted me to add to the splines. The most important of these was the ability to set properties for each path node. This will allow him to more accurately control the A.I., for example, he could specify at which node they must use nitrous or handbrake. I started adding the splines and kept providing Herman with updated versions of the editor. He made good suggestions which made the spline editing more user-friendly. What did I learn from this? Don't assume your program is user-friendly until someone has happily used it.

IN SUMMARY

We think we pulled off what we set out to achieve in a methodical, persistent manner. We understood many of our limitations up front, though on the technical side there were many curve balls which we did not expect. These issues were examined, and then prioritised. Adjustments to the Design Document were made to accommodate these unknowns, and we made peace with what we could and could not do within the restrictions of time and budget. Working within parameters and restrictions is always a rewarding process, as it forces you to think more clearly, and come up with creative solutions to problems by thinking outside of the box. What's also cool about 'solving' issues pertaining to flaws in your technology is that people always ask "How did you do that?!", and thats very rewarding. All being said and done, game development, to a large degree, is after all just smoke and mirrors!

On a more sober note, we overshot our deadline by around 60 days, and consequently went over budget as a result. We also had not budgeted in the minute-long CG Intro, but decided to do one anyway and cover that cost too. The result was that LumaArcade subsidised the total cost of development by around 25%. In our opinion this was not a problem since we got to build our first game, build a core team, and still got paid 75% of the cost to do it. Our business model for MINI#37 as a franchise will also cover these upfront costs over time as we continue to develop more Episodes. Given the opportunity to decide, the team at Luma would definitely do this all over again!

CODING ETIQUETTE

Consistent Styling

by Luke "Coolhand" Lamothe

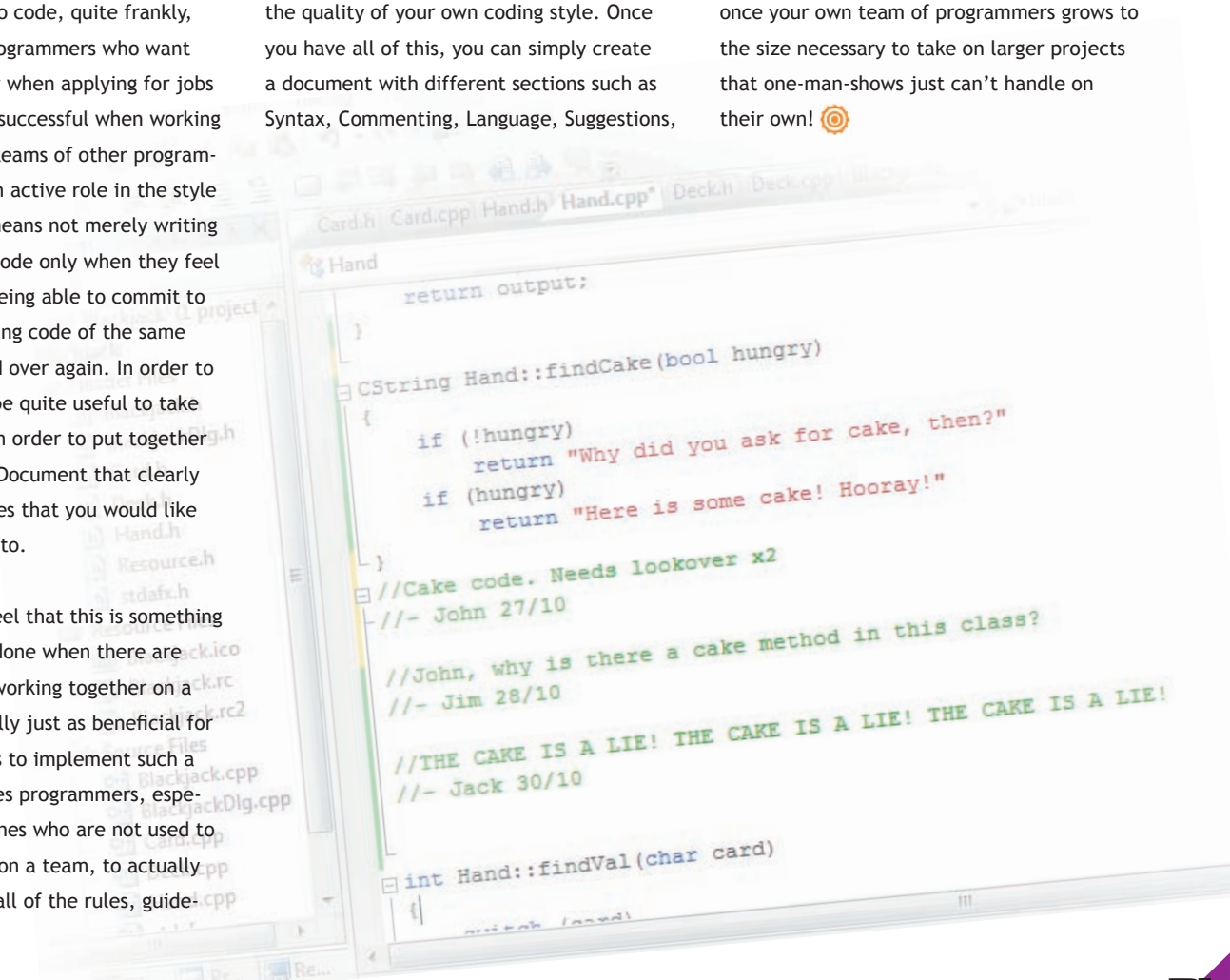
The past six articles in the Coding Etiquette series have been designed to make programmers think about how they code, not in terms of squeezing as many clock cycles as possible out of their algorithms, but instead in terms of how they can make their code more pleasing to not only their own eyes, but also to the eyes of anyone else who may have to view their code. In this, the final entry of the series, we will look at how all of what we have gone over up until now can come together to create what is known as a Coding Guidelines Document, which is designed to help lone-wolf programmers as well as team-oriented programmers maintain a consistent style to all of the code that they produce.

Being aware of the fact that there are both good and bad ways to code, quite frankly, just isn't enough. Programmers who want to be taken seriously when applying for jobs and who want to be successful when working together with large teams of other programmers need to take an active role in the style of their code. This means not merely writing neat and organised code only when they feel like it, but instead being able to commit to consistently generating code of the same high quality over and over again. In order to achieve this, it can be quite useful to take the time necessary in order to put together a Coding Guidelines Document that clearly defines all of the rules that you would like your code to adhere to.

Most programmers feel that this is something that should only be done when there are many programmers working together on a team, but it is actually just as beneficial for single manned teams to implement such a document. This forces programmers, especially independent ones who are not used to working with others on a team, to actually put down in writing all of the rules, guidelines, and coding styles that they feel make up the way that they code. The added benefit for independent programmers creating a document like this is that if and when additional programmers should join their team, they will already have a clearly defined set of rules for the these members to follow.

The actual creation of a Coding Guidelines Document is very simple, if a little time consuming. The easiest advice to give is to go through each of the previous articles in this series and make notes of any suggestions given that you feel would be helpful to the way that you code. Next, examine your own code as well as any other code that you may have come across in tutorials and the like, and look for certain styling that appeals to you or that you think would improve on the quality of your own coding style. Once you have all of this, you can simply create a document with different sections such as Syntax, Commenting, Language, Suggestions, Best Practices, etc., and list the guidelines or coding style samples for each topic that you feel are useful to follow.

At the end of the day, each and every programmer will have unique coding styles or rules which suit them, none of which are necessarily right or wrong. It is important however when writing a Coding Guidelines Document, that you make sure to include every possible item that could affect the way that you wish for your code to be written. Every aspect of coding that can be thought of (ie. line indentations, bracket placement, function usage, variable naming, commenting, error checking, syntax, etc.) should be written down so that there can be no misunderstandings when it comes to how you would like your code to be styled, especially once your own team of programmers grows to the size necessary to take on larger projects that one-man-shows just can't handle on their own!



THE HISTORY OF I-IMAGINE

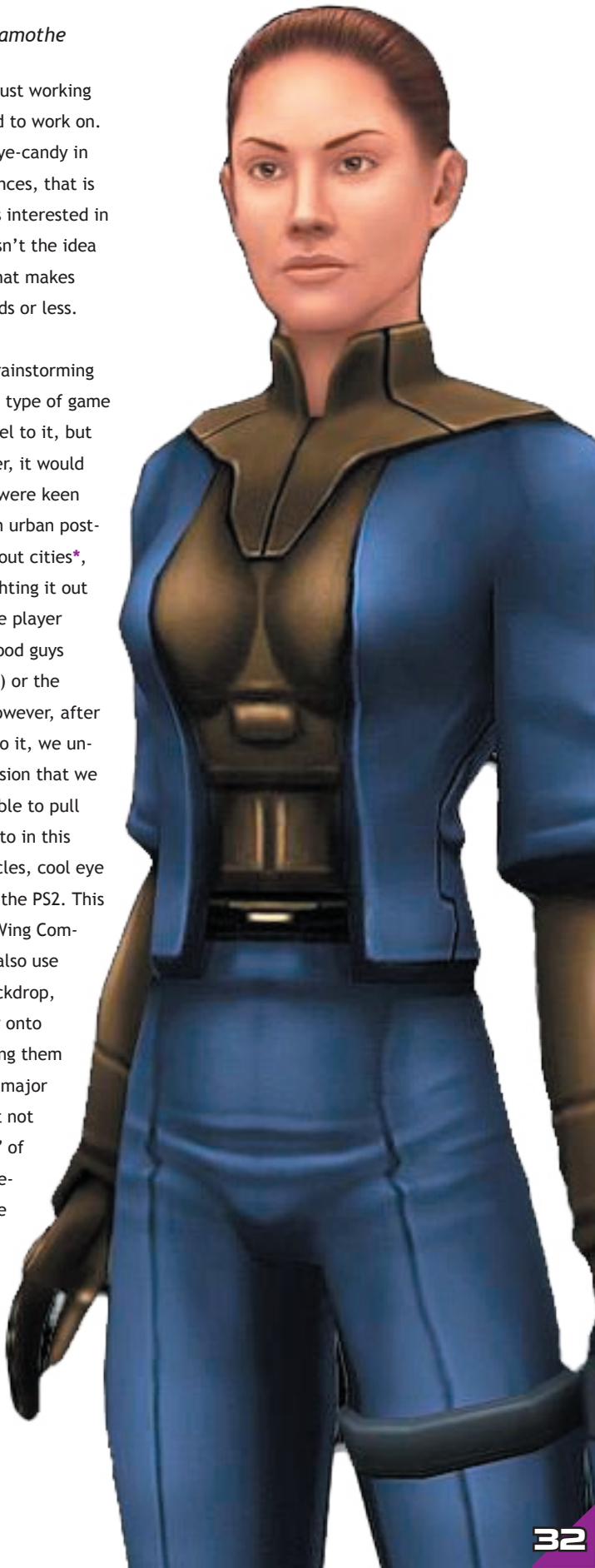
PART 8: " 'Final' Armada? "

by Luke "Coolhand" Lamothe

After spending the previous two years unsuccessfully trying to convince publishers to sign us up to a deal for either our own original IP or to develop an existing IP of theirs, I-Imagine as a development studio was beginning to get into trouble. There was increasing pressure coming from various investors amongst the venture capitalists who provided the initial funding for the company, with regards to when we would stop spending "their" money without having anything to show for it other than a single game for a single platform after nearly 5 years. Knowing of this, Dan and I sat down around the middle of 2004 with the purpose of talking about what would most likely be our final shot at developing a game that would be of interest to publishers, and that would allow us to finally become financially secure. Before actually discussing what kind of game we thought we should make however, we first agreed to a set of parameters which any game idea that we would decide upon had to adhere to.

The first parameter was that the game had to be designed first and foremost for the PS2, with Xbox and PC only as afterthoughts due to the fact that publishers only really wanted PS2 content as the new generation of machines was on the horizon and PS2 was clearly the market leader in the current generation. The second one was that the game had to be achievable with a minimum amount of changes to our existing technology, which invariably meant that it had to be a vehicle-based game as we didn't have the time or money to spend on developing brand new technology from scratch. Thirdly, it had to be a game that we would be excited about developing which meant that it had to be a game that we would want to play, as we knew the power of having people work on something that they themselves could get excited about rather than just working on something that they are told to work on. Lastly, it had to have a lot of eye-candy in it as from our previous experiences, that is ultimately what gets publishers interested in your game. In other words, it isn't the idea or the substance as much as what makes them think "Cool!" in 30 seconds or less.

Given these parameters, our brainstorming quickly began to form around a type of game that had a Wing Commander feel to it, but instead of being a space-shooter, it would use land vehicles. Initially, we were keen on building the game around an urban post-apocalyptic setting with burnt-out cities*, revolving around rival gangs fighting it out over control of these areas. The player could take on the role of the good guys (some kind of law enforcement) or the bad guys (one of the gangs). However, after some more thought was put into it, we unfortunately came to the conclusion that we weren't confident with being able to pull off everything that we wanted to in this game (large cities, lots of vehicles, cool eye candy) on our target platform, the PS2. This pushed us even more into the Wing Commander vein as we decided to also use space and the future as our backdrop, only we would place the player onto actual planets instead of keeping them in the void of space itself. The major benefits of doing this were that not only could we use the "excuse" of having the game set on underdeveloped planets that were more rural and therefore potentially more sparse in terms of the art resources necessary to make them look good, but this also allowed us to not have to keep in line with creating art that people



* *Think "Escape from New York".*

would compare the fidelity of to Earth-based settings*.

Once actual development on Final Armada was underway, our first goal was to have a prototype that demonstrated the player controlling a vehicle and having the ability to fight and destroy other vehicles and buildings. The idea behind this was that if we could prove that this would be fun to do, then we knew that we were on the right track and that we could end up making a game that other people would have fun playing as well. Around the end of October 2004, we were able to complete this demo which had a vehicle with machine guns and missiles fighting against AI vehicles against the backdrop of an alien planet, and lo and behold, it was actually really fun to play! Once the fun-potential of our game was proved to us, we set out to nail down the technology necessary to pull it off on the PS2**.

We ended up going through quite a few iterations of this technology, from our world mesh rendering (we didn't have a terrain renderer), to the particle system, to having the ability to convincingly draw large scale environments spanning more than 3km2. Unfortunately, we were always hindered by what the PS2 could do with these kinds of large scale environments which was mostly due to the machine's relatively low amount of memory***, but also due to the rendering power of the the PS2. After much trial and error, we reached a point where we happy with what we were able to achieve in terms of graphical output, with mesh-based LODs for our worlds (similar to what Shadow of the Colossus would eventually use), alpha fogging of our world geometry combined with permanently visible meshes for our backgrounds and large objects in order to give the illusion that the entire world was always being drawn, and a very robust effect system which allowed our artists a large amount of freedom and power to come up with unique and very pretty eye-candy for all things particle-based. Also, our own in-house game editing tool was written during this time in order to help us manage our worlds with things like lighting, scripting, and special effects. This tool ended up proving itself invaluable to us and we wouldn't have been able to do a lot of the things that we did in the game without having access to it.

The majority of publisher interaction during the development of Final Armada has actually escaped my memories, as the 2+ years that were spent working on the game actually seem like nothing more than a few months in my mind! I do remember that initial publisher interest in the title was very similar to our earlier efforts, and that while we did have some parties who seemed keen on the title, it was always the same old story of wanting

---

** Who knows what a plant or tree would look like on Planet X?*
*** Our initial demo was only running on the Xbox as that was quickest and easiest to develop for.*
**** The PS2 has 32MB of RAM (of which only 31 are available to the developer) where your game data is stored and 4MB or VRAM (video RAM) where your video buffers (frame, Z, offscreen, etc.) and textures are stored.*

us to come back to them later when we had more to show. The fact that it was primarily a PS2 game, however, did lead to a lot more interest amongst the smaller publishers as they just wanted to get any product out into the market. We even had a few deals offered to us mid-way through our development by these smaller publishers, but they were always very low in terms of upfront money, although decent in terms of the potential royalty returns. Each time one of these deals came up though, it was turned down by the directors of I-Imagine as they wanted to hold out and secure a deal that meant that we would be able to turn a profit from the sale of the game, and not have to rely on royalties in order to do so, which was something that I wholeheartedly agreed with.*

By the time that 2005 began to draw to an end, we were still having publisher bites but the I-Imagine investors were starting to grow weary of us spending money with nothing to show for it. At the AGM that year, it was decided that if a deal for the game wasn't signed before the end of the year, that I-Imagine would close its doors and that the investors would take their remaining money back from the company. As expected, the team was quite upset about this turn of events, and while nearly everyone began to start looking for jobs elsewhere in the industry (particularly in the UK) they still did their all in order to try and secure a deal for Final Armada before the end of the year. However, when December rolled around and we still had not secured a publishing deal, the final decision was made and nearly the entire staff of I-Imagine was retrenched. The only exceptions to this were Dan and myself who would remain employed for a limited amount of time (until the end of March) while still trying to finish off and sell the game that we had, as well as one of our artists Dave who was brought back on a contractual basis in order to complete the resources for various sections of the game. Luckily, most if not all of our retrenched employees were able to find work quickly, with the majority of them moving overseas to the UK or the US.

Ironically, not long after the retrenchments happened, we began to communicate with a publisher called Virgin Play. They were based in Spain and were looking at extending their PS2 and PSP ranges in a bid to start publishing into the rest of Europe. While we continued to work our way through January and February, talks with them seemed to progress and by the time the end of February came around, I went away on my honeymoon hoping to return to some good news. As it happened, by the end of March a deal was made with Virgin Play for PS2 and PSP versions of Final Armada that were due to be completed by the end of 2006. Upon hearing this news, we knew that we would need to bring on board at least one other programmer** and another artist*** in order to com-

---

** You also want to give reasons to a publisher to invest time, money, and effort in promoting your game, to ensure that it reaches a good number of sales; if they have invested a decent amount of money in it upfront, they are more likely to do so compared to if they have only invested a relatively low sum of money.*

*** The technology was done for the PS2 version, but we needed to convert it to PSP as well as have networked multiplayer gameplay working on that platform.*

**** Most of the art resources were completed but some still remained, and we needed to complete all of the game's pre-rendered cutscenes as well as re-factor various resources for the PSP (related to performance and screen-resolution).*

plete Final Armada in time. As a result, we brought back another of our previous artists, also named Dave, in order to complete the cutscenes as he was busy waiting for a work permit in order to enter the UK and take up a job there which he had already secured. As for programming help, we turned to Danny Day, otherwise known to readers of Dev.Mag as 'dislekcia', to handle all of the network code for the PSP, as well as to get our engine working in a networked manner as it had never been designed to do so.

Our first task upon signing the deal with Virgin Play was to convert what we had of Final Armada to PSP as we needed to pass through Sony's internal approval process as quickly as possible. We were luckily able to use the PSP version of RenderWare to make the initial stages of the conversion relatively painless, and once various art resources had been reworked for the hardware*, we had the game up and running with most of the same features as the PS2 version within a few months. Once we were comfortable with the PSP version actually being a possibility, we turned our focus towards finishing off the PS2 version of the game. We were able to do this by October 2006, with things such as localisation into various European languages, small bug fixes, and feature requests from the publisher being the only real hurdles that needed overcoming.

Once the PS2 version was done and off in SCEE certification-land, we moved on to getting the PSP version completed. There were still some specific resource changes and PSP code necessary, as well as certain programming techniques which needed to be rethought and reworked entirely for that hardware as it didn't support some things that the PS2 did. Most notably though was that we had to complete and implement an entire multiplayer mode which caused rippled changes to occur through the menu system, to the GUI, to additional levels and effects that were needed. Finally, towards the end of November 2006, the PSP version was ready for certification. Around this time, the PS2 version passed its certification process, and by early January 2007, the PSP version finally passed certification as well.

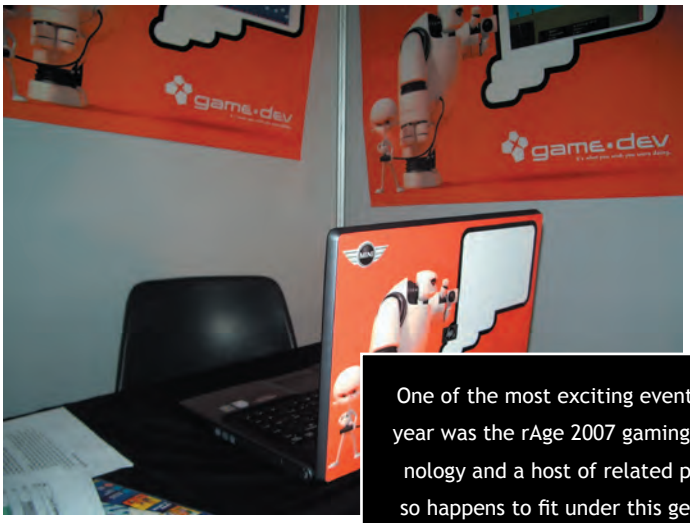
By the time that both versions went "gold", I-Imagine had already formally stopped its day-to-day operations. I had begun a new job at an animation studio in Joburg called Luma, where they were starting up their own game development division to complement their existing 2D and 3D design studios [Their first game development project, Mini #37, has already been featured in several Dev.Mag issues – check out the postmortem in this edition]. Dave-1 also joined me at Luma in order to head up their art team for their first project, which was to be a racing game featuring Mini Coopers set in various South African locations and commissioned for MINI South Africa. Dave-2 left for his job in the UK once his cutscenes were finished, which was quite a while before the game was actually completed, and Danny's contract ended once the PSP version of the game was certified and his network programming skills were no longer needed. Dan moved on to a new role with MI Digital as their Marketing Director, where he was responsible for bringing the Xbox360 into South Africa, making use of all of his previous years of built-up connections with the people at Xbox.

By the middle of 2007, all of the investors were finally paid out their investment capital and I-Imagine was officially reduced to nothing more than a bunch of computers, some software licenses, various in-house developed technologies, and all of the original IP that it had developed over the years. However, if rumours are to be believed, it appears as though I-Imagine may be on the verge of a resurrection – at least in spirit, if not in name. Apparently, Dan is busy working on a title for the Xbox360 at the moment through MI Digital and has re-hired some former employees of I-Imagine, but nothing has been officially announced yet. So while the title of I-Imagine's last game may have seemed prophetic at one time, it may turn out not to be the case in the foreseeable future ...

** The most notable changes were to make textures a lower resolution, reduce the polygon counts of various meshes, and redesign the GUI for the PSP's screen.*

UNLEASHING THE rAge

SA'S biggest gaming expo gets a game development kick



One of the most exciting events on the Game.Dev calendar this year was the rAge 2007 gaming expo, dedicated to games, technology and a host of related pursuits. Game development just so happens to fit under this generous umbrella, and the Game.Dev organisation has established a firm presence at the expo for the past two years. Over the next few pages, you'll be treated to a healthy dose of eye-candy and some visual depictions of what really goes on at rAge. Browse through and enjoy - we hope it encourages even more people to come next year!

THE SETUP

Big expos like this don't just pop into existence, you know. There's a hefty amount of advanced and not-so-advanced planning that goes into sprucing up the final product, including a fair share of grunt work and sessions of standing around while looking confused. It was worth it, though. The Game.Dev stand ended up sporting some very fancy features, including a rear-projected display for the presentations.



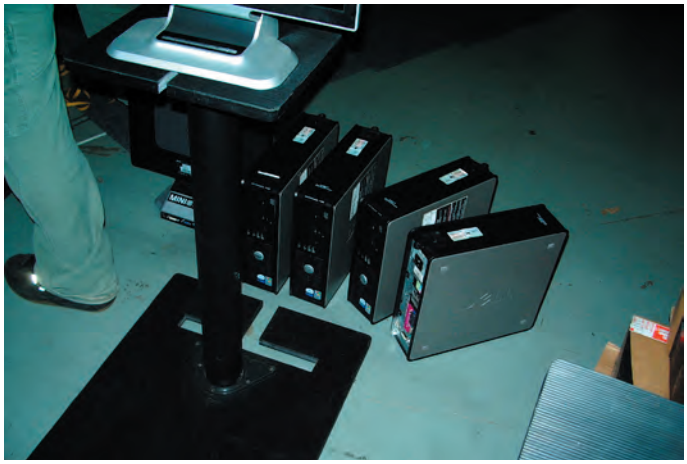

Danny "fast-like-ninja" Day sets up the stand in approximately 0.42 seconds, fetches an audience and finishes it all off with a sweet ninja pose.

Our ammunition. We pack about a hundred of these babies.

Luma's stand freshly set up and, as we soon discover, completely blocking our own.

Luma hands us some very pretty posters and demo computers to draw in those people who enjoy bright colours and loud noises.

The food court. Our new master.



Miniature computer cases — we all wanted one, really.

THE TALKS

Presentations and workshops were one of the major reasons for Game.Dev's presence at rAge. Several of these talks were scheduled on each expo day, presented by various professionals involved with game development in some way or another – including representatives of South Africa's current flagship studio, LumaArcade.

Another wary spectator wanders into a Game.Dev presentation, nervously looking out for any traps.

TV cameras too!? We're famous!

Retrotoast's Cadyn Bridgman gives audiences something to gawk at.

Dale best from Luma also gives presentation a go while Ninja Danny strikes more poses. A formidable team indeed.

THE COMPETITIONS

Prizes! The word is always a drawcard, especially amongst those who stand a good chance of winning. Sure, not all of Game.Dev's competitions at rAge had prizes attached to them, but compensation did come in the form of a single game-making competition which had a whopping R10 000 cash pool tied to it. The rAge expo was deemed an ideal spot to present the winners with their prizes.

Danny Day. Evangelist.

Mindset also have their say. They threw the money at us, after all.

The three winners of Comp 15. None of them really wanted to be on camera.

"Whaddaya mean, we don't get to keep the oversized cheques?!"

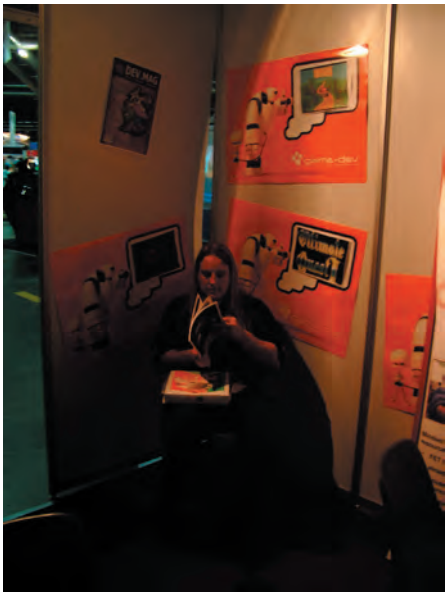
"ATTAAAAAAACK!"

"Okay! Okay! You can keep them! Just please don't stab me!"

THE PEOPLE

rAge just wouldn't be complete without all the people there to make it happen. Gatherings like this quickly foil the all-too-common perception that devvers are all exclusive loners. Events such as Game Dev Idols, a meet and greet and even an open discussion forum were all available for people to take advantage of, and less formal gatherings were frequent. Community involvement is the backbone of Game.Dev.

"And if I turn this way, you get a beautiful view of my left cheek!"



We found him in a corner while cleaning out, and decided to keep him as a pet.

Genuine happiness, or mind-altering substances? You decide!

He doesn't work for Luma, but we like to pretend he does.

Wait, what?

"A cellphone, you say? What a truly remarkable device!"

Hotdogs. They bring people together. Then they get eaten.

That's all, folks! We hope you all come along to rAge 2008! For further plans and events, remember to check out www.gamedotdev.co.za