



NOW IN **SEXY** 4:3 GLORY!



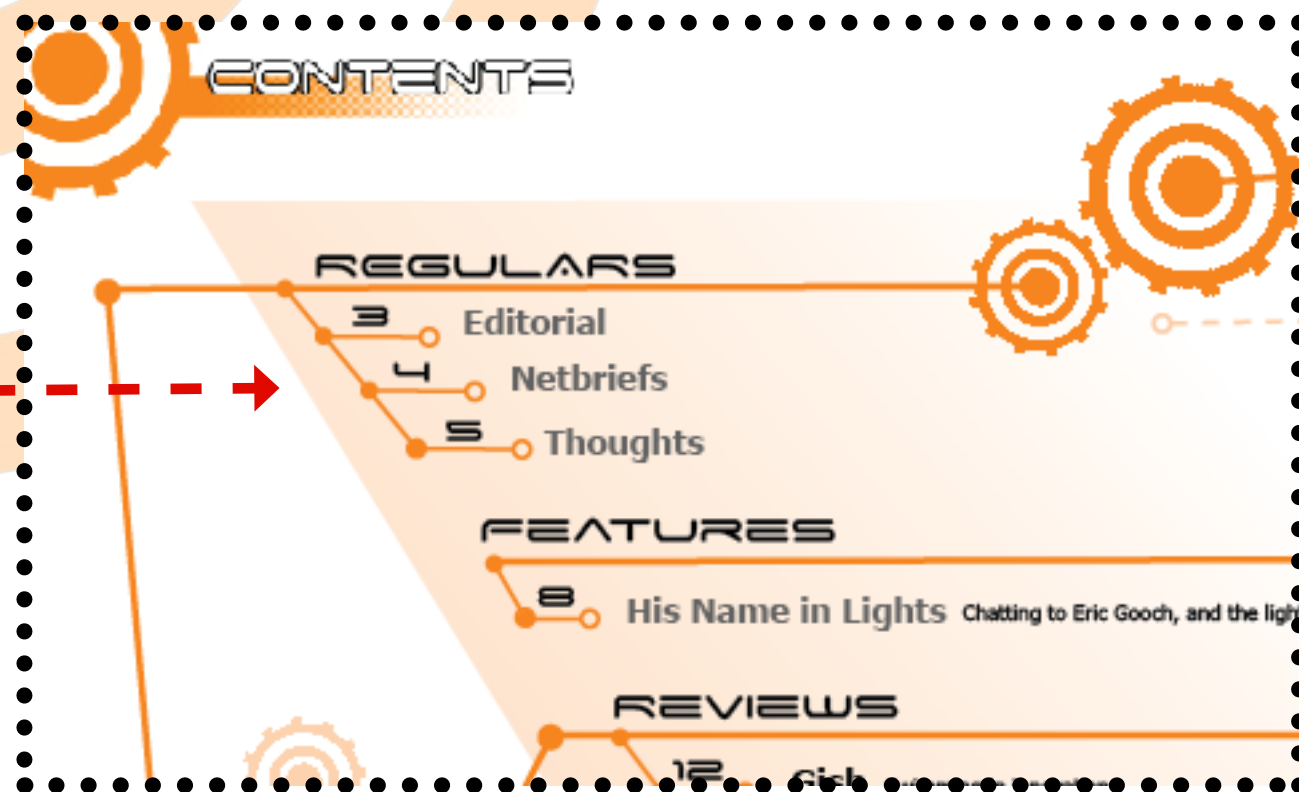
GISH

**being a blob of tar has never been so
much fun!**

INSIDE:

GISH • ERIC GOOCH • PHOTOSHOP PT 7 • BLENDER CLOTH SIMULATION •
gGUARDIAN POSTMORTEM • NEWS and OPINION + MORE!

WHAT'S NEW IN DEV.MAG



PRESSING THE RIGHT BUTTONS

Marvel at the **navigational brilliance**! Tired of scrolling through pages and pages of content? Looking for something specific? Well, **scroll no more**! Using the newly featured **nagivational buttons**, you can get to where you want to be, and back again, in no time at all.

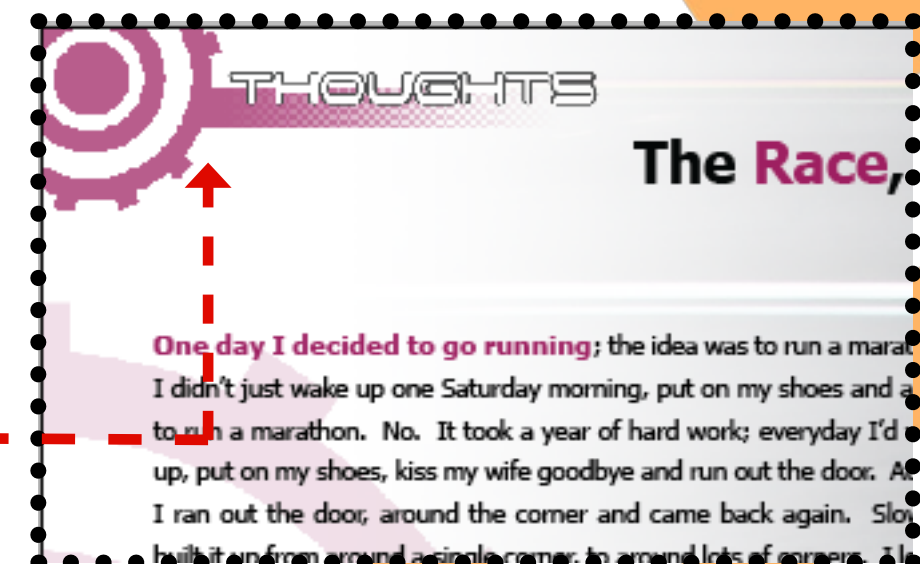
Click **any title on the contents page** to be **transported directly to your selected article**

To get back, simply click on **the title bar of any page** - it's that easy!

As you all may have already noticed, there have been a few (Mi-nor) changes to Dev.Mag's look this issue: Our layout team (or rather, person) has worked overtime to take Dev.Mag to the next level of interactivity and style!

Gone is the "For-Print" A4 page layout! Dev.Mag is now giving you the whole picture with complete 4:3 aspect ratio sexiness. That's right! No more edges! No more annoying zooming! Dev.Mag now takes up your *entire* screen!*

As you may have noticed when you opened up this PDF document, no longer are you forced to move your hand *all the way to the bottom of your monitor* to hit the 'full screen' button! **We've gone ahead and done it for you!** No need to thanks us.



*Does not apply to widescreen, obviously

CONTENTS

REGULARS

4

5

6

FEATURES

9

Chatting to Eric Gooch, and the lighter side of life

REVIEWS

13

Swimming in innovation

DESIGN

16

For a limited time only!

TUTORIAL

20

Bring on the coding!

27

Covering GUI and menu components

32

We take a look at cloth simulation

TAILPIECE

37

...programming ninja by night!

EDITOR

Claudio "Chippit" de Sa

DEPUTY EDITOR

James "Nighttimehornets" Etherington-Smith

COPY EDITOR

James "Nighttimehornets" Etherington-Smith

DESIGNER

Quinton "Q-Man" Bronkhorst

CONTRIBUTORS

Rodain "Nandrew" Joubert
 Simon "Tr00jg" de la Rouviere
 Ricky "Insomniac" Abell
 William "Cairnswm" Cairns
 Danny "Dislekcia" Day
 Andre "Fengol" Odendaal
 Luke "Coolhand" Lamothe
 Rishal "UntouchableOne" Hurbans
 Gareth "Gazza_N" Wilcock
 Sven "FuzzYspo0N" Bergstrom
 Kyle "SkinkLizzard" van Duffelen

WEBSITE ADMIN

Robbie "Squid" Fraser

WEBSITE

www.devmag.org.za

EMAIL

devmag@gmail.com

This magazine is a project of the South African Game.Dev community. Visit us at:

www.devmag.org.za

All images used in the mag are copyright and belong to their respective owners.

Laying all those damn gears, and I all I get is this small space to say something witty or clever. I sure as hell better not waste it on anything stup- OH SHI-!

So things have reorganized themselves a little around here. A new name has mysteriously appeared in the staff list. I've been informed that it belongs to our new dedicated designer, **Quinton "Q-Man" Bronkhorst**, who will be handling the mystical art of making all the words look pretty. **James "NightTimeHornets" Etherington-Smith** was also surprised to find his own name considerably higher on the ladder. He hasn't quite recovered from the shock yet, but I'm confident he'll convalesce in time for the next edition. There's the ice-cream too, but I'm not so sure that's new, or wise, for that matter, considering it's winter here now.

We've also made a few changes on the presentation front, most of which you've likely already noticed by the time you got here: the layout is specifically designed to be easier to read on a monitor now, with a different orientation and the introduction of hyperlinks. The other changes are highlighted in our special "what's new" page.

Lastly, but certainly the most important, as well as saddening, change: As you likely already know, our beloved editor has unfortunately had to step down, which leaves little ol' me to occupy his virtual office. I must say I'm impressed with the place, though. He had a mahogany desk he never

told me about. If I'd known before I'd likely have been very jealous for a very long time.

But enough of me blathering on about notional furniture. The magazine must live on, right? And indeed it will, and has. Strong content is about like always, and in this issue we have an enlightening feature interview with **Eric Gooch**, the artist responsible for much of the graphics in Westwood's Command & Conquer series, and known for playing the role of Seth, everyone's favourite dictator's right-hand man. We then close things off with a look at what our former ed really did in his spare time.

Finally, this is the time for me personally to wish **Rodain** the best of luck and to thank him for the dedication that brought us this far. The magazine wouldn't exist otherwise and it endures now because of your lasting influence.

Thank you, and farewell.

~ Claudio, Editor





GameBryo 2.5 released

<http://www.emergent.net/en/Products/Gamebryo/>

GameBryo, the popular cross platform engine for PC, Xbox360, Wii and PS3—known best for use in games such as Civilization 4 and Elder Scrolls IV: Oblivion—has just been updated. This engine features large numbers of great assets and internal gems but also harnesses each platform's power and ability without sacrificing on other platforms. This makes it fast on each platform and easy to manage, ensuring that development is about the game, not about the engine. Worth a look, for certain.

SCEA releases PlayStation-EDU

<http://blog.us.playstation.com/2008/06/06/playstation-edu/>

The PlayStation has always been a great console at its peak moments, but development for it has always been a little out of reach for students and interested parties. PlayStation announced recently that the PlayStation-EDU incentive, a new study program, is now on the prowl and is taking its fighting spirit into universities and colleges worldwide. Taking PS2 and PSP development kits into the learning place and giving students the exposure to draw them into the development is clearly a good move that is going to benefit SCEA in the long run.



Dream-Build-Play 2008 begins

<https://www.dreambuildplay.com/>

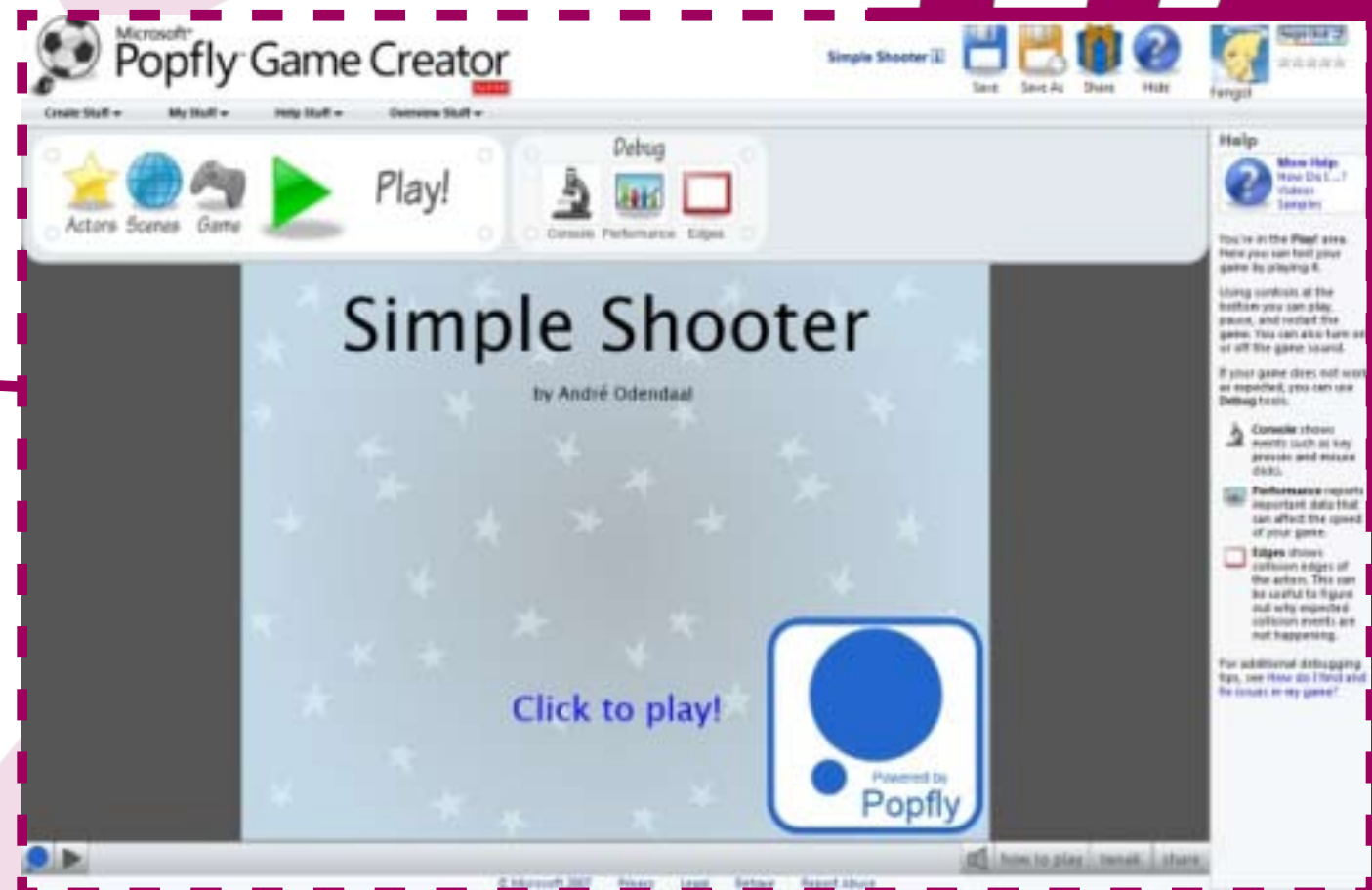
Microsoft's Dream-Build-Play competition has kicked off again this year, boasting a considerable \$75,000 in prizes and a possible LIVE publishing contract. This year the competition is limited to development on Xbox 360 only, using XNA again, of course; but this caveat is not as dire as it appears. Registering for the competition will grant teams a free 12-month trial membership to the XNA Creator's Club, which is everything you need to deploy your games to 360. We're entering. Are you?

On the Fly - Games with Popfly

I feel compelled to talk about Microsoft Popfly Game Creator; not only because it's a practical (read: something you would use on a daily basis) and feature-rich implementation of a web application using Silverlight that doesn't just involve viewing videos; but because when using the app and fiddling with the game dev API, it also looks like the developers have taken heed of what's out there in the hobbyist/indie game dev community. Launched on the 2nd May 2008, the Popfly team introduced Popfly Game Creator to their portal. Popfly is a web portal, providing visual tools for creating web pages, mixing content and sharing it online. Currently it's in a Public Beta phase with a 25mb storage limit and visitors are required to install the Microsoft Silverlight extension (about 1.4mb; think of it like the Flash extension you need to install to visit certain sites).


I'm not much of a content sharer but I was quite keen to check out the Game Creator and after a day of playing and making a simple game, I'm impressed. Anyone who's used Game Maker should quickly find themselves at home but the first notable feature was that when starting a new game, the site already offers a host of different game templates to start with which means you can have a working game type straight off the bat (and first impressions for the masses count). The second notable feature is the site's use of free resources like Danc's graphics from Lost Garden. These well drawn sprites give a sense of fulfillment when making your game as your creation looks already polished. Visitors to the portal also seem able to add new content to the free resources pile which means content creators like Danc can continually add new stuff for you to play with. Of course you can always upload your own graphics for use in your game.

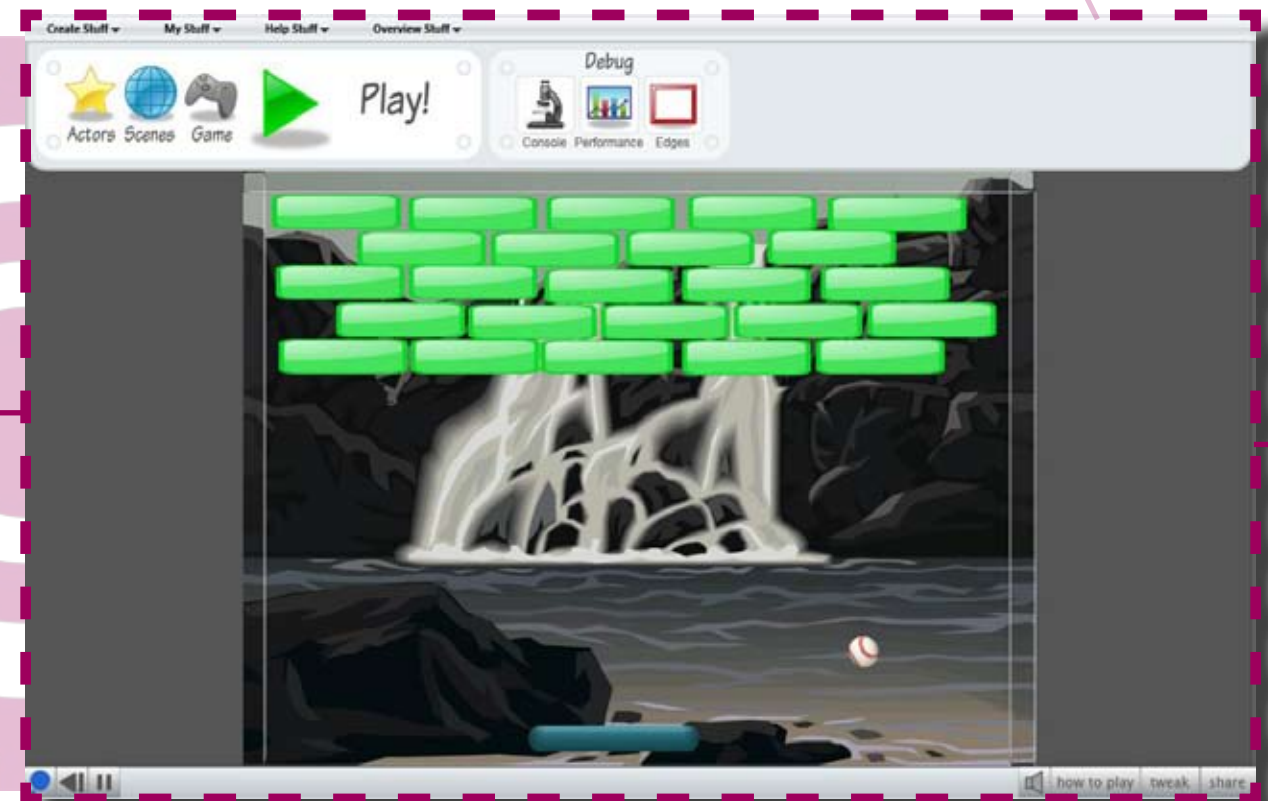
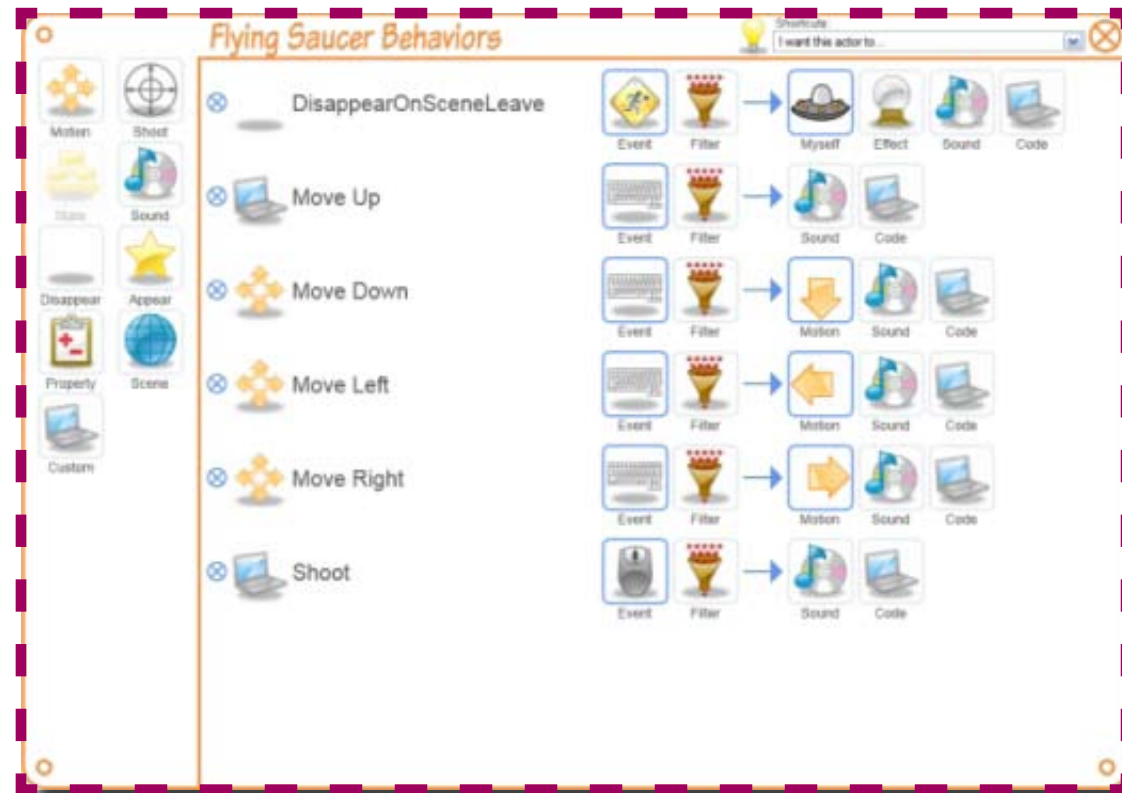
Manipulating actors (the objects in your game) using the visual tools is a little awkward initially but you get used to it; the third notable feature is the ability to use predefined behaviors like "run and jump by pressing keys" or "fly like a spaceship by pressing keys (top view)" which is available from a drop-down and makes for quick implementation of your game idea. Like a good dev tool, you can also bypass the designer and dive into the code which is written in Javascript. Coding was a little frustrating as most of the properties for an actor seem to be stored in a hash table and you have to use GetValue() and SetValue() a lot, but the Help and API links on the side of the screen are very clear and helpful.



Looking at the environment, it's easy to dismiss it saying no one can build a decent game with it and I'd agree with you for 90% of the people who try it out; but as Game Maker and Flash games have proven before, a dedicated developer can make something that'll knock your socks off. From a business developer perspective, Game Creator is a mind-blowing example of how to build a web application that isn't just viewing reports or shopping carts. It's fast, with near instant access to different parts of the application as well as server side resources like pictures, sounds and code. I'd say it's a humbling experience for anyone serious about delivering applications via web and worth playing with even if you're not a game developer. If this is the future of online applications; it's bright and sunny and full of more good cheer than sugar-induced laughter at a 4 year old's birthday party.

 **André "Fengol" Odendaal**

 "A DEDICATED DEVELOPER CAN MAKE SOMETHING THAT'LL KNOCK YOUR SOCKS OFF."



The Race, The Business, The Game

One day I decided to go running; the idea was to run a marathon. I didn't just wake up one Saturday morning, put on my shoes and arrive to run a marathon. No. It took a year of hard work; everyday I'd wake up, put on my shoes, kiss my wife goodbye and run out the door. At first I ran out the door, around the corner and came back again. Slowly I built it up from around a single corner, to around lots of corners. I learnt about pronation, energy supplements, power and hill training and even injury recovery processes.

The other day I bought a business; the idea was to make a fortune. I didn't just deliver quality projects and make money in the first month. No. It takes a lot of hard work; everyday I wake up go to work and learn. Every day I learn a bit more, and every bit I learn makes my company more successful. I learn a bit about Tax calculations, employee remuneration structures, the products I sell, the development process to produce the products I sell and even how to sell the products I sell.

Yesterday I woke up and decided to write a new Multiplayer, online powered Real Time Space Simulation with Role Playing Game Elements (a real MMORTS/RPG in space). So I sat down and wrote Pong. Today I might write Pac-Man and tomorrow it might be Tetris; hopefully next week I can write Diablo. Before then I need to learn a bit about reflective multi dimensional modular shading techniques, maybe a little about instantaneous multi-pointed network flow diagrams, and even a bit about space warps.

Everything in life is a journey. Enjoy your game making journey; there is little in my life that exceeds the pleasure I have had writing games.

 **William "cairnswm" Cairns**

"**TODAY** I MIGHT WRITE PAC-MAN AND **TOMORROW** IT MIGHT BE TETRIS; HOPEFULLY **NEXT WEEK** I CAN WRITE **DIABLO**."



His name in

Lights

Does anyone remember Seth from the original Command and Conquer? Does anyone remember the awesome graphics in games such as the Ratchet & Clank series or Emperor: Battle for Dune? Eric Gooch might not be a well known name at the local LAN but in the game development world he is a master who can claim respect at a moments notice. With a portfolio of games dating back as far as 1995, Eric has left a pronounced mark wherever he has worked. Dev.Mag managed to grab a hold of him and ask him some questions after his recent trip to Insomniac studios, his current employer.



Sven "FuzzYspo0N" Bergstrom

Tell us who you are.

My name is Eric Gooch; I'm an artist living in North-Eastern Nevada. I work for Insomniac Games, a game company based in Burbank, California that makes games for the PS3. I am fortunate in that I am able to live out in the country, and work remotely from my home over the internet doing lighting for games that I find fun to work on. I also do artwork of my own, mainly Science Fiction and Fantasy, which can be seen at my website www.cybergooch.com.



You are a game artist; tell us what you do.

I am a lighting artist. There are currently 3 of us at Insomniac Games. My job is to light the levels after the environment artists are done building them. Usually there is a certain look that the art director is after, so we work together with the art director and the concept artists to achieve believable lighting; whether the levels are huge outdoor areas or claustrophobic tunnels and indoor environments. The technologies we use are always evolving, so it's a real challenge to bring everything together. It's great to see it all take shape, as the lighting makes a huge difference to the final look of the levels.

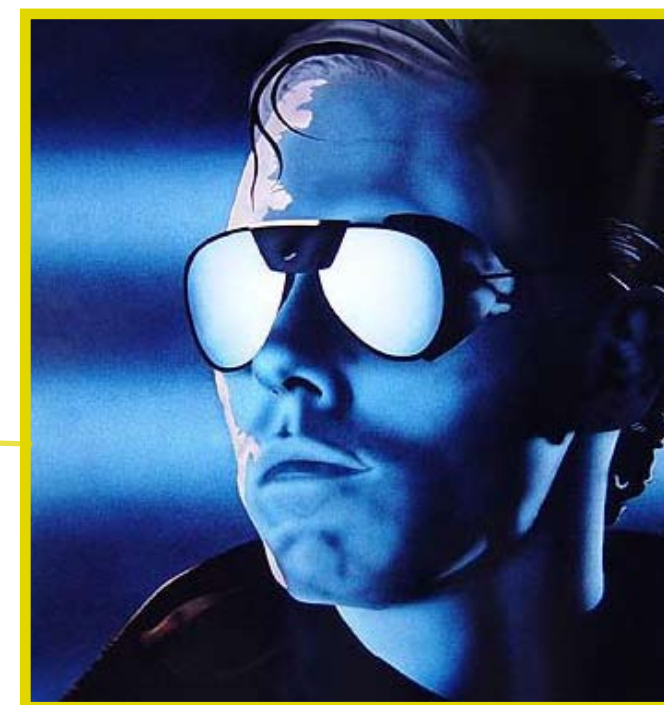
Every 8 weeks I fly to Burbank for a few days, for meetings and to get caught up on everything.

"MY JOB IS TO LIGHT THE LEVELS AFTER THE ENVIRONMENT ARTISTS ARE DONE BUILDING THEM."

IT'S GREAT TO SEE IT ALL TAKE SHAPE, AS THE LIGHTING MAKES A HUGE DIFFERENCE TO THE FINAL LOOK OF THE LEVELS

Where have you come from and where are you now?

I was born and raised in Ohio. After high school I moved with my family to Philadelphia, where I went to Photography school and got an Associates degree. Since then, I've lived in Tucson, San Francisco, Detroit, San Diego, Hollywood, Las Vegas, Los Angeles, and now I've settled in the tiny town of Spring Creek, Nevada. It took a lot of moving around to figure out what I wanted to do!



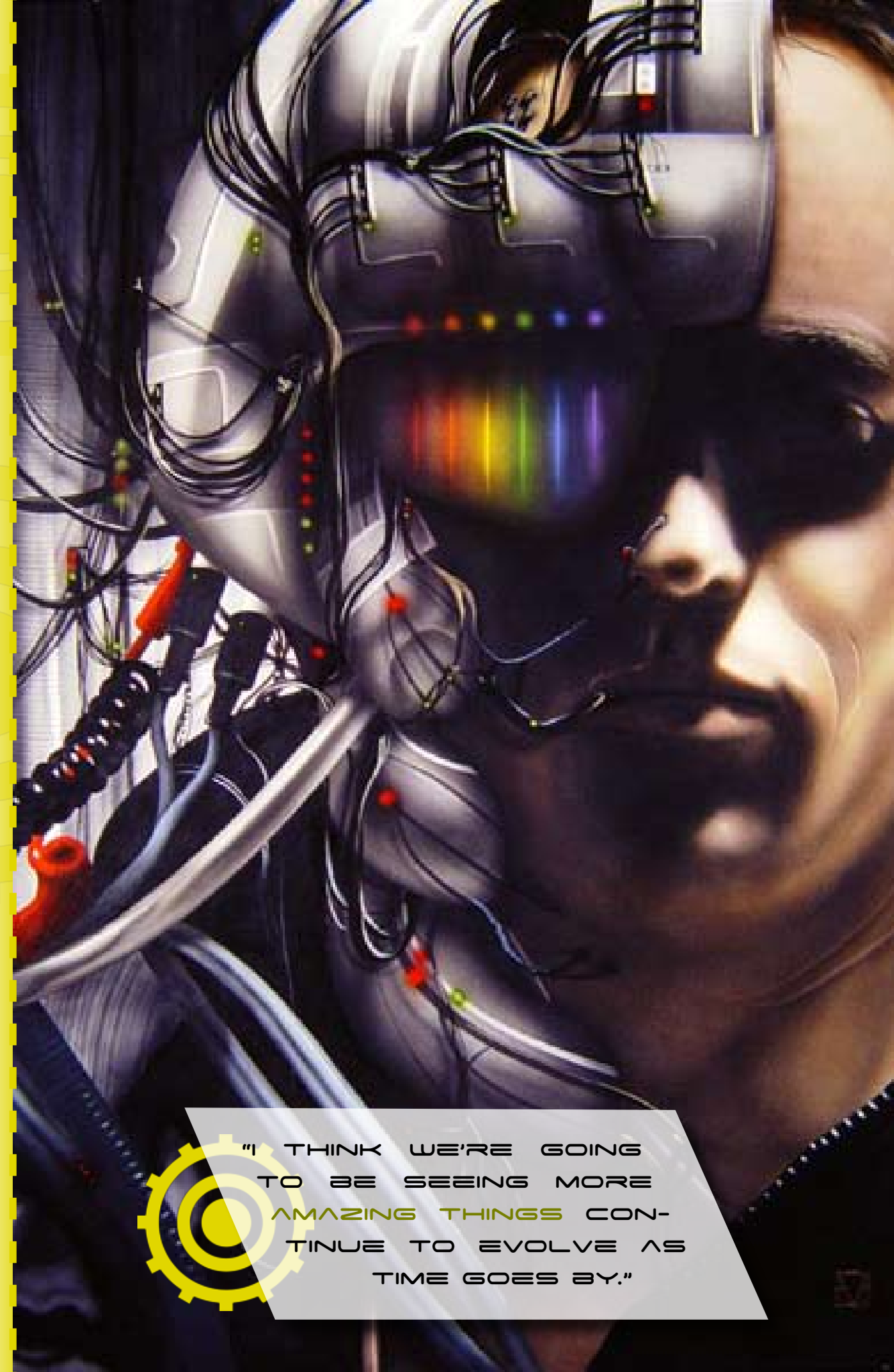
During your career in game development, what have you learned that is most valuable to you?

I'd like to address that in two ways: First, regardless of game development or anything else you may end up doing, my number one rule has always been; "Don't get good at something you don't like." I have known too many people that chased after something they thought they should be doing, only to realize it wasn't what they really wanted after all. I really believe you should go after what you truly love doing, even if you're afraid you might not make as much money doing so.

Second—specifically concerning game development—learn to be flexible. You can't see into the future and know for sure what tools and techniques are going to be used and/or needed. It's a good idea to experiment with different software and with different tools in general to know how things work throughout the areas you'll be involved with. You will still probably want to specialize in one area, but keep your options open.

What do you think the future of game art holds?

More and more realism. There are still limitations in what we're able to achieve, but those limitations are being chipped away with every new generation of hardware. I think we're going to be seeing more amazing things continue to evolve as time goes by, and for those games where realism is not necessarily the goal, we'll still be seeing advances in what we're able to convey visually. I think it's a pretty exciting time to be involved in computer graphics, as things are advancing so rapidly.



"I THINK WE'RE GOING TO BE SEEING MORE AMAZING THINGS CONTINUE TO EVOLVE AS TIME GOES BY."

If you could spread your knowledge to indie game developers, what pearl of wisdom can you offer the aspiring?

I haven't really been involved with the indie scene, but from what I've seen in general, I'd say the most important thing is organization. Having a master document that describes every detail about your project so that everyone is on the same page is key. Too much gets lost in assumptions and miscommunication. Along the way, there are invariably going to be times when things seem overwhelming or confusing, and a solid foundation document can make all the difference in the world.

"DON'T GET GOOD AT SOMETHING YOU DON'T LIKE."

QUICKIES

One liners:

*"Give a man a job he loves
and he will never work a
day in his life."
- Confucius*

Day or night?

Night

Music or silence?

Silence

**Windows / Linux
/ Mac / Other?**

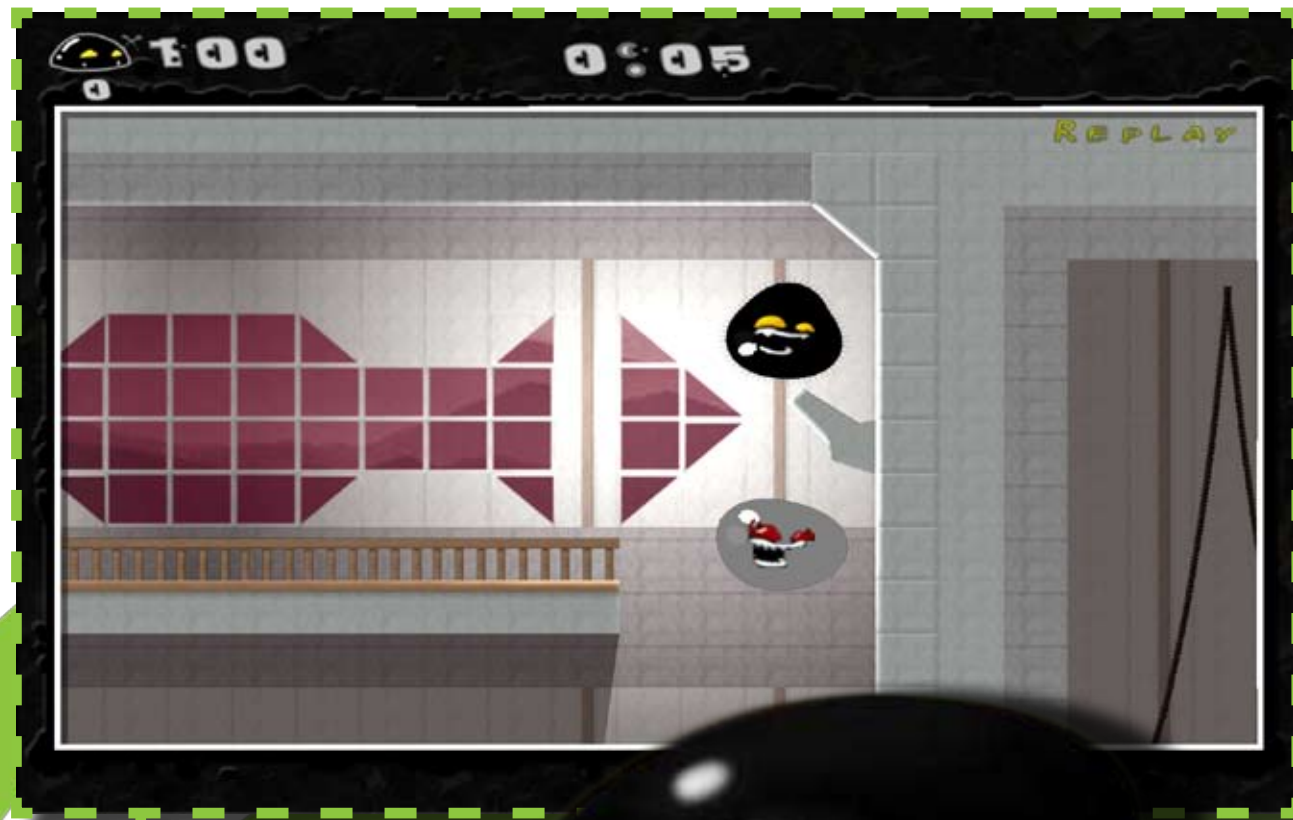
Windows



Getting to Grips with Gish

Gish is a strange game in all regards; the theme, the storyline, even the general idea; but innovation comes from strange places, and Gish swims in innovation. Whilst the presentation may be a bit rough around some edges, the game is otherwise very impressive; visually, with its neat lighting, but mostly in the way the physics handles the player's movements.

VERSUS OPTIONS CREDITS EXIT



The player avatar is represented by an amorphous blob of tar named Gish. The player controls Gish via basic omnidirectional movements, as well as allowing him to assume any combination of four properties: Gish can be made sticky, allowing him to cling to walls and other objects; he can increase his mass in order to break things; he can be made slick in order to slide into narrow gaps; and he can expand his volume, allowing him to jump. Different combinations of these properties can also be used to varying effect.

The entire game revolves around these properties and it constantly challenges the player to make creative use of them in order to pass the varied levels. The simplicity of the design makes the game very easy to pick up and understand, but it requires a significant amount of practice before one can perform the most daring and impressive feats. This is exacerbated by the very steep learning curve, which often throws extremely difficult challenges at the player without warning or previous experience of similar techniques.



"CHAOS IS FUN!"

However, your goal is rarely unclear—barring a few occasions with boss characters that require special actions beyond the basic ‘Gish smash!’ to kill—so, with sufficient persistence every level is definitely passable; even if you’re tempted to throw your controller at the wall during the process.

I do recommend playing the game with an analog controller of some sort if possible. It simply feels better that way. There’s something to be said about picking up a controller, handing a second to a friend, and hitting up one of the multiplayer modes. Football (my personal favourite) involves two players attempting to wrestle a ball from each other and off the edges of the screen or launching it over the goalposts. All the multiplayer modes become truly chaotic at times; the flexibility of Gish’s movements makes them far more interesting than the descriptions imply; and chaos is fun!



Claudio “Chippit” de Sa

REPLAY

“THE PLAYER AVATAR IS REPRESENTED BY AN AMORPHOUS BLOB OF TAR NAMED GISH.”

"GGUARDIAN TAUGHT ME A
LOT ABOUT PACING A GAME
PROPERLY."

Gazza_N Presents:

gGuardian

POSTMORTEM

THE GGOOD

THE GGRIEF

Gareth "Gazza_N" Wilcock





Your game can be anything—any genre, any theme, any mechanic you like—as long as it ends conclusively within ten minutes. Sounds challenging? Not? Well, then obviously you didn't take part in Game.Dev Competition 10, where the object was to produce a game that fit that criterion within a month.

I haven't really ever played a formal tower-defence game, but I know that I love sieges in games. I've always enjoyed the sequences in FPS or RTS games where you have a time limit to prep for an impending invasion with limited resources; slap together a patchwork defence; and then scramble to hold off enemies when the attack hits. Best of all, this mechanic lends itself perfectly to a time limit scenario. It was this thinking that gave birth to gGuardian.

The concept is simple: you control a robot drone from a top-down perspective, with your job being to defend a cryogenic storage bay full of star ship passengers from an overwhelming force of attackers, eliminating said attackers within the ten-minute round (this was justified by stating that the robot only had a ten-minute power supply.) The idea is to have the player spend the first five minutes setting up their defences, then the next five fending off the invaders. If the invaders reach the cryo-bay, you have a minute or so to kill them before they break in, or you lose the game. In addition, if there are still aliens left after the ten minutes has expired, you lose the game. Simple enough.



The gGood

Above all, I wanted to inject a little tension into the game: force the player to think quickly under pressure. I realized early on that having five whole minutes just to place turrets would result in a pretty dull first half of the game. One solution was to decrease the preparation phase time and have the combat phase last longer, but that made the combat feel stretched out. My first solution was to procedurally generate the levels. This had multiple benefits: it kept the player on edge by forcing them to explore a different map during the prep phase of every single round; it ensured replay value; and it meant that I wouldn't have to spend excessive time designing a whole bunch of maps myself. I also decided to scatter randomly across the map weapons needed by the player, to create even more tension and to add extra incentive for exploration. This turned out to be a good decision, as it did end up adding the desired tension to the game, and very few who played the game complained about its pacing.

I was also very lucky to receive a deluge of constructive criticism while building the game, which resulted in many elements of the game being changed or tweaked for the better. One forum member (Anihillist) offered me so much criticism that he ended up messaging me to ask whether he had offended me or not! Fortunately, all the critique I received from him and the other Game.Dev regulars was solid and valuable, and was essential to shaping gGuardian into its final form.

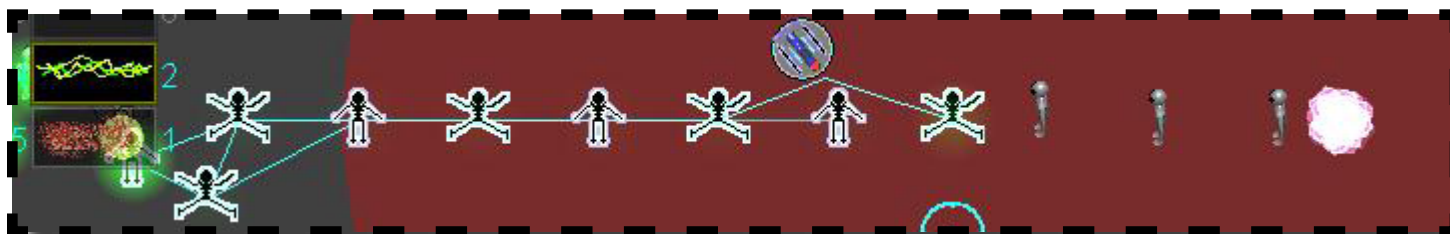
The gGrief

Ironically, the month-long competition time limit was my undoing when creating gGuardian. Given its importance to the design, I was forced to spend a lot of that time getting the level generator running properly. As a result, I had very little time left to polish the rest of the game. Graphics consisted almost entirely of rudimentary MS Paint lameness rather than the pre-rendered 3D sprites I had envisioned. Levels were graphically bland, with nothing but one or two random MS Paint doodads sprinkled here and there to give visual diversity. Weapon balance was good enough for the game to feel fair, but some weapons were underpowered and ended up being ignored in favour of their more effective counterparts. The path finding for the alien hordes tended to bug out at times, leaving lost aliens wandering the corridors aimlessly, forcing the player to actively hunt them down (a task made easier by the fortunate decision to include a mini-map). Worst of all, the environmental traps I had intended to include (i.e. pipes that could be breached to create walls of fire; toxic waste vats that could be spilled onto aliens;) had to be eliminated altogether in the interests of getting the primary game play elements working in time.

Conclusion

gGuardian ended up coming second in competition 10 based on its pacing and the sense of panic it evoked. However, many improvements were suggested that I hope to implement at a later stage. There's still plenty of room to grow gGuardian now that I know that the core game play works properly. The graphics certainly need improvement, and I'd like to finally put in the environmental traps to give the player more to do in the game than simply shuffling turrets around. An option for co-op play isn't out of the question.

Overall, gGuardian taught me a lot about pacing a game properly, and about pacing a game's development properly. It isn't perfect at the moment, but with a bit of work it could turn into something I can really be proud of.





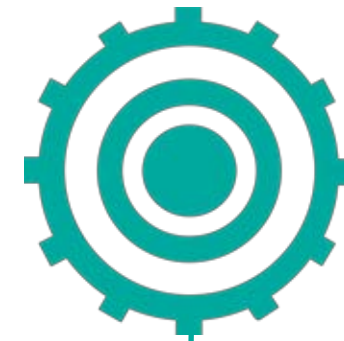
IN CASE OF EMERGENCY
BREAK THE MOULD



IRRLICHT

PART 3: BRING ON THE CODE

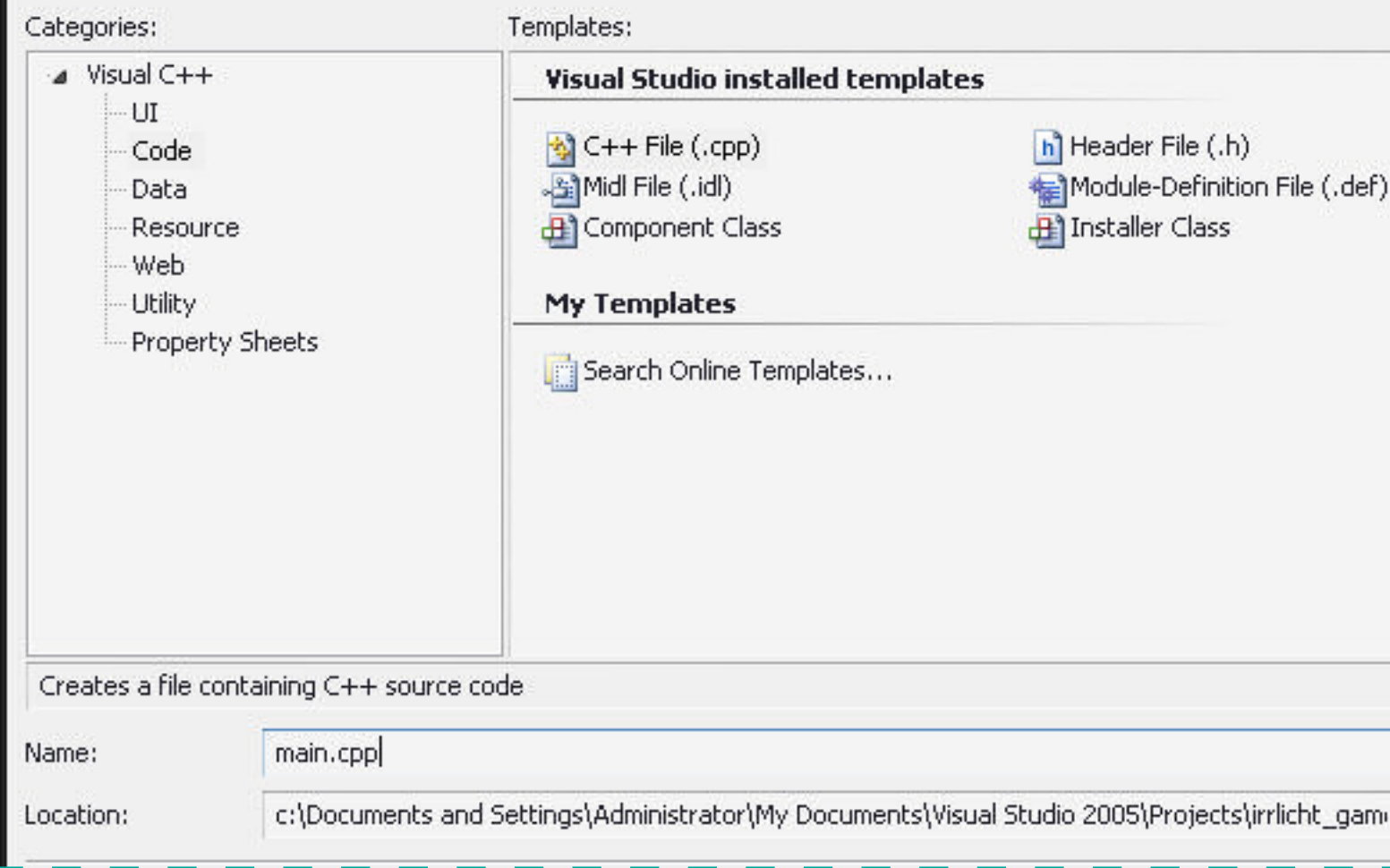
 Sven "FuzzYspo0N" Bergstrom



Unless otherwise stated
we will be using the **1.4
release** version of the
engine

"**Now comes the interesting part**," I hear you say. I dare not cover 'setting up the engine' again; there are tons of these all over and many can be found on <http://irrlicht.sourceforge.net> as well. Let's get straight into the programming and waste no further time without an example. This part in the series is going to showcase more than just a black screen or a hello world; it is going to encompass creating a black screen, adding some text, adding some GUI elements, and creating some images. This gives you enough of a handle to see and feel how Irrlicht likes information and how easy the engine is to use.

Add New Item - irrlicht_game.dev_part3



Step 1

I'm going to follow an Irrlicht favourite here and use the main.cpp file as an example. The main file consists of a couple of routine commands, and a nice introduction screen that will help you understand the goings on of the engine.

Let's go:

We need a couple of things for this example: we need the Irrlicht engine core; we need a scene manager; a device driver; and a GUI. We have already mentioned these but we are making some pointers to the following:

IrrlichtDevice*

video::IVideoDriver*

gui::IGUIEnvironment*

scene::ISceneManager*

The star is cpp for pointer, and the stuff before the :: just means we are keeping things together for now. If I had said *using namespace gui;* in the code at the top, I would not have needed to put the gui:: there in the first place. For now, I'm leaving it there to show you where things fit in. Let's take a look at the code, so far.

This code will do nothing, but will become the basis of our application. The downloadable source does contain comments, as well as descriptions of everything.

```
#include <irrlicht.h>

#pragma comment(lib, "Irrlicht.lib")

using namespace irr;

IrrlichtDevice*      irr_engine;
video::IVideoDriver* irr_driver;
gui::IGUIEnvironment* irr_gui;
scene::ISceneManager* irr_scene;

void main()
{
}
```

major two would be 16 or 32. 32 is obviously the current standard for 3D applications on PC and most consoles.

bool – fullscreen, stencilBuffer, VSync

Logical answer, true or false. Would you like full screen? How about a stencil buffer or vertical sync? Each of these are your choice.

IEventReceiver – receiver

This is what would receive events, keypresses, mouse movement and that kind of thing but we don't need it at the moment, so let's leave it at 0.

That covers all the information Irrlicht wants. We are going to call **createDevice** and we are going to return a device pointer; you store this device pointer in the **irr_engine** variable for using while we work with the engine. After we have this engine variable we are now going to use it! The other elements we need were listed above: the driver; the GUI; etc. The next step is to create these elements directly from the engine. The code is shown below.

```
void main()
{
    irr_engine = createDevice( video::EDT_OPENGL,
    core::dimension2d<s32>(1024, 768), 32, false, false, false, 0);

    irr_driver = irr_engine->getVideoDriver();
    irr_gui = irr_engine->getGUIEnvironment();
    irr_scene = irr_engine->getSceneManager();

    irr_engine->setWindowCaption(L"Hello Dev.Mag!");
}
```

As you can see we are using the standard 1024x768x32 resolution and we are not going to be using stencil buffers or vertical sync. This is enough information to setup our scene.

Step 2: The engine and its joys

Yes, we have reached the part (already) where we are going to be making a blank screen. For those of you still getting into C++ do not fear, it is one loop, and some function calls. We need the engine. There is one function that has two ways of being called: one is called **createDevice** and the other is called **createDeviceEx**. The first of these will be just fine for what we want to do and we hand it a couple of things it wants. The documentation tells us it wants the following:

video::E_DRIVER_TYPE – deviceType

This is the driver. Choose between DirectX or OpenGL or one of the software renderers. The **E_DRIVER_TYPE** is a hint as to what to type to 'find' it in intellisense. For example **EDT_OPENGL** is the OpenGL driver. The rest are listed when you type **EDT_**.

const core::dimension2d< s32 > & - windowSize

Sounds complex? It's not really complex when you see the answer to what it wants. It is a template which is handed directly to you by Irrlicht core:: namespace. This means core::something can usually give you what you need. For now, we need a dimension2d that is of the type s32; this is basically 2 points, width and height, and it's an integer value.

U32 – bits (bpp)

This is standard graphics talk and there are a few options; the

Step 3: The loop and the darkness

We are going to do a loop, based on some information irrlicht hands us so nicely. Seeing as graphics rendering might be intensive, we don't need to be drawing or updating the screen while it is not the active window. This pauses irrlicht in essence because nothing is being done while the window is, for example, minimised. The other condition we will look at is whether the window is still there, as in, the engine window is still alive and running. If someone had hit the close button or alt-f4 the application will close leaving our loop. This is pretty basic to grasp, let's move on.

Currently the application will run and then quit, with memory leak issues. Our next step involves telling the engine to draw everything, until the program is closed. With most graphics rendering loops there is a **beginScene** and an **endScene**. All the relevant drawing and passing of information is done within these calls. This means we are looking for irrlicht to draw our scene (from the scene manager) and our GUI (from the gui manager) and also, to begin and end a scene. Irrlicht takes some information for the beginScene call. It says "what colour should I make the scene?" It asks whether we want a depth buffer and a back buffer. These are generally a good thing so let's enable them and let's set the color to black.

```
while(irr_engine->run())
{
    if (irr_engine->isWindowActive())
    {
        irr_driver->beginScene(true,true,video::SColor(0,0,0,0));

        irr_scene->drawAll();
        irr_gui->drawAll();

        irr_driver->endScene();
    }
}
```

This is now a running irrlicht application. Though it doesn't do much of anything, the massive engine behind irrlicht is now powering your application from within. Make sure that you have copied your irrlicht.dll into the necessary places for it to run nicely, and you can test it out so long.

Step 4 : The text and the conversion

This path has lead us to the point where we need some information on screen, so we are going to use some default settings to write some text to the screen. I will cover how to make your own fonts using the font tools included with irrlicht. Inside the SDK folder we find the bin directory which holds all the program executables related to irrlicht. The font tool is relatively simple to operate, and generates some important files. The first are the bitmap files which are in the format you specified (i.e. png) and then an XML file that holds the font information. We load the font up into irrlicht by using the **IGUIEnvironment**, and we can use it from then onward. When we have our font at hand we can use the font to draw directly onto the screen (remembering that it can be covered by things, and cleared off screen at times) or we can create a GUI element to hold the text. We will do both so you understand which way you prefer. Now that we have a handle on the GUI interface, we can just create things as we wish. We will create a variable for our **IGUIStaticText**, which is going to hold the text, and we declare a font that we want to use, an **IGUIFont**. Both of these are straightforward pointers to an object, using the pointer operator (->) we can see what each offers. We will add it right after our irrlicht engine pieces. We are well on our way; in fact we are going to add an image as well, while we are here. The image will be a picture in the background of the text.

Lets see what I looks like in code :

```
scene::ISceneManager* irr_scene;

gui::IGUIFont* ourveryownfont;
gui::IGUIStaticText* ourstatictext;
gui::IGUIImage* ourdbpicture;
```

and then we can see step-by-step what is going on.

```
ourdbpicture = irr_gui->addImage(irr_driver->getTexture("./data/image/db.png"),core::position2di(300,80));

ourveryownfont = irr_gui->getFont("./data/font/devmag.xml");
ourstatictext = irr_gui->addStaticText(L"dev.mag",core::rect<s32>(250,300,1000,600));

ourstatictext->setOverrideFont(ourveryownfont);
ourstatictext->setOverrideColor(video::SColor(255,242,135,40));
```


As with most pointers, the intellisense will give you all the options, and for irrlicht we need to understand what gives us what. You can only grasp certain aspects from the experience of using the engine (such as where textures come from). We use the graphics driver to get textures into the engine, that way we add it through **irr_driver** variable using the engine. Because we want the picture behind the text, we load it first, this will place it in the buffer first, everything coming above it, is above it. The next step is using the GUI engine to load up the fonts we want to use, by calling **getFont**. As you can see, we load it using the XML file from the font tool. That's all it takes! Simply put, this is all self explanatory; we create a static text box using

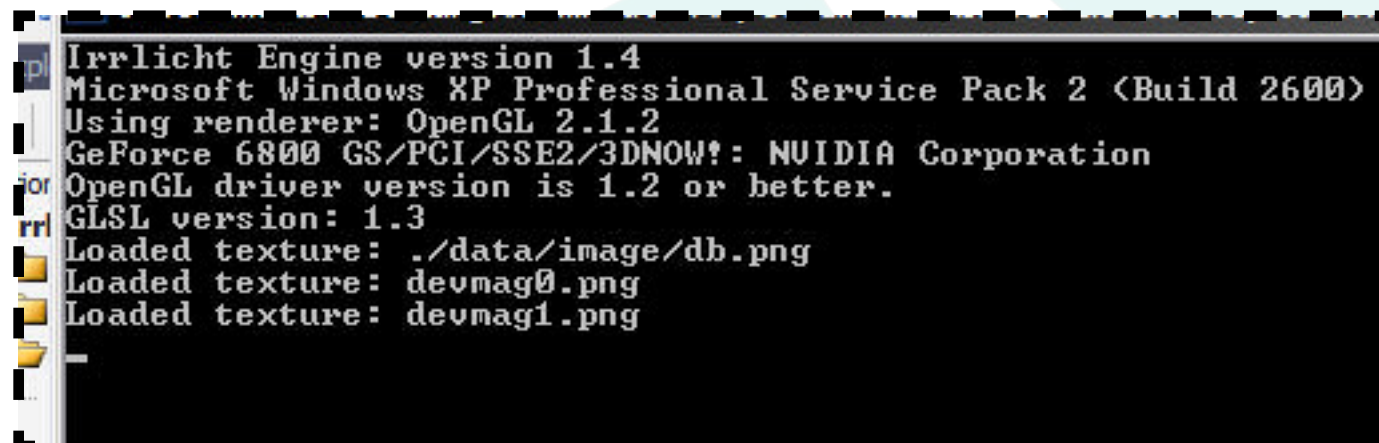
the GUI engine again. As you may see there is a `core::` reference. This reference has a ton of core functions that will help irrlicht understand c++ variables, and will create useful types that are completely optimised for irrlicht. Using the core templates takes a little understanding. Anything irrlicht wants, for example **rect<s32>**, takes s32 types in a rect structure. **S32** is an integer value, so it is a "single" 32 bit value. These are all from the documentation; the templates allow one declaration for many types. Watch out for these types in the `< >` templates. They are just looking for what it asks for. For example, `position2df` takes float values, `position2di` takes integer values, but **position2d<** takes any type that it allows. `Position2d<s32>` is the

same as **position2di**, if that makes sense.

Now we use the variable that holds the information of our static text box, to override the default font, and the default color of the text. This is using some things that will be important, the `SColor` type from the video side is used often in irrlicht, so remember to keep it in mind later on. Now we have all the code we need for a complete "scene" with images, and text, and we have a framework that can work in most situations in irrlicht. I added something in the main loop that uses the default font, by getting the information from the engine, and drawing text in the built-in irrlicht font. This is the final thing we add; we add it at the end of the main loop, just before the `endScene` call.

```
irr_gui->getBuiltInFont()->draw(L"GAME.DEV - ITS WHAT YOU WISH YOU WERE DOING",core::rect<s32>(450,390,800,400),video::SColor(255,255,255,255),true,true);

irr_driver->endScene();
```




```
Irrlicht Engine version 1.4
Microsoft Windows XP Professional Service Pack 2 (Build 2600)
Using renderer: OpenGL 2.1.2
GeForce 6800 GS/PCI/SSE2/3DNOW!: NVIDIA Corporation
OpenGL driver version is 1.2 or better.
GLSL version: 1.3
Loaded texture: ./data/image/db.png
Loaded texture: devmag0.png
Loaded texture: devmag1.png
```

In this window you can see all aspects are found by engine! That's great news.



The final output of our application

This is all it takes to have some stuff on screen! Check the final code snippet and download the files for this project from the site. The files contain project files and for VS2005, as well as the main.cpp file that can be compiled in most irrlicht-ready IDE's.

This series will continue with using the GUI elements, loading 3D scenes from the free tool irrEdit as well as some other great features. 

Final code snippet :

```
#include <irrlicht.h>

#pragma comment(lib, "Irrlicht.lib")

using namespace irr;

IrrlichtDevice* irr_engine;
video::IVideoDriver* irr_driver;
gui::IGUIEnvironment* irr_gui;
scene::ISceneManager* irr_scene;

gui::IGUIFont* ourveryownfont;
//To load the font, we use this
gui::IGUIStaticText* ourstatictext;
//To draw the font in a text box
gui::IGUIImage* ourdbpicture;
//To draw the picture of DB

void main()
{
    irr_engine = createDevice( video::EDT_OPENGL,
        core::dimension2d<s32>(1024, 768), 32, false, false,
        false, 0);

    irr_driver = irr_engine->getVideoDriver();
    irr_gui = irr_engine->getGUIEnvironment();
    irr_scene = irr_engine->getSceneManager();

    irr_engine->setWindowCaption(L"Hello Dev.Mag!");

    ourdbpicture = irr_gui->addImage(irr_driver->
        getTexture("./data/image/db.png"),core::position2di(300,
        80));
```

```
//continued

ourveryownfont = irr_gui->getFont("./data/font/devmag.
xml");
    ourstatictext = irr_gui->addStaticText(L"dev.mag",cor
e::rect<s32>(250,300,1000,600));

    ourstatictext->setOverrideFont(ourveryownfont);
    ourstatictext->setOverrideColor(video::SColor(255,242
,135,40));

    while(irr_engine->run())
    {
        if (irr_engine->isWindowActive())
        {

            irr_driver->beginScene(true,true,video::SCO
lor(0,0,0,0));

            irr_scene->drawAll();
            irr_gui->drawAll();

            irr_gui->getBuiltInFont()->draw(L"GAME.DEV
- ITS WHAT YOU WISH YOU WERE DOING",core::rect<s32>(450,3
90,800,400),video::SColor(255,255,255,255),true,true);

            irr_driver->endScene();

        }
    }
}
```


GAME GRAPHICS WITH ADOBE PHOTOSHOP

PART 7: GUI DEVELOPMENT AND MENU COMPONENTS



Rishal "TheUntouchableOne" Hurbans

This tutorial is building on the previous tutorial (Issue 22) that was based on creating a menu background. **We will be creating buttons and other components common to the menu screen and the user interface.** It is quite satisfying having a game with polished and professional looking graphics and a clean-cut, easy to use menu.

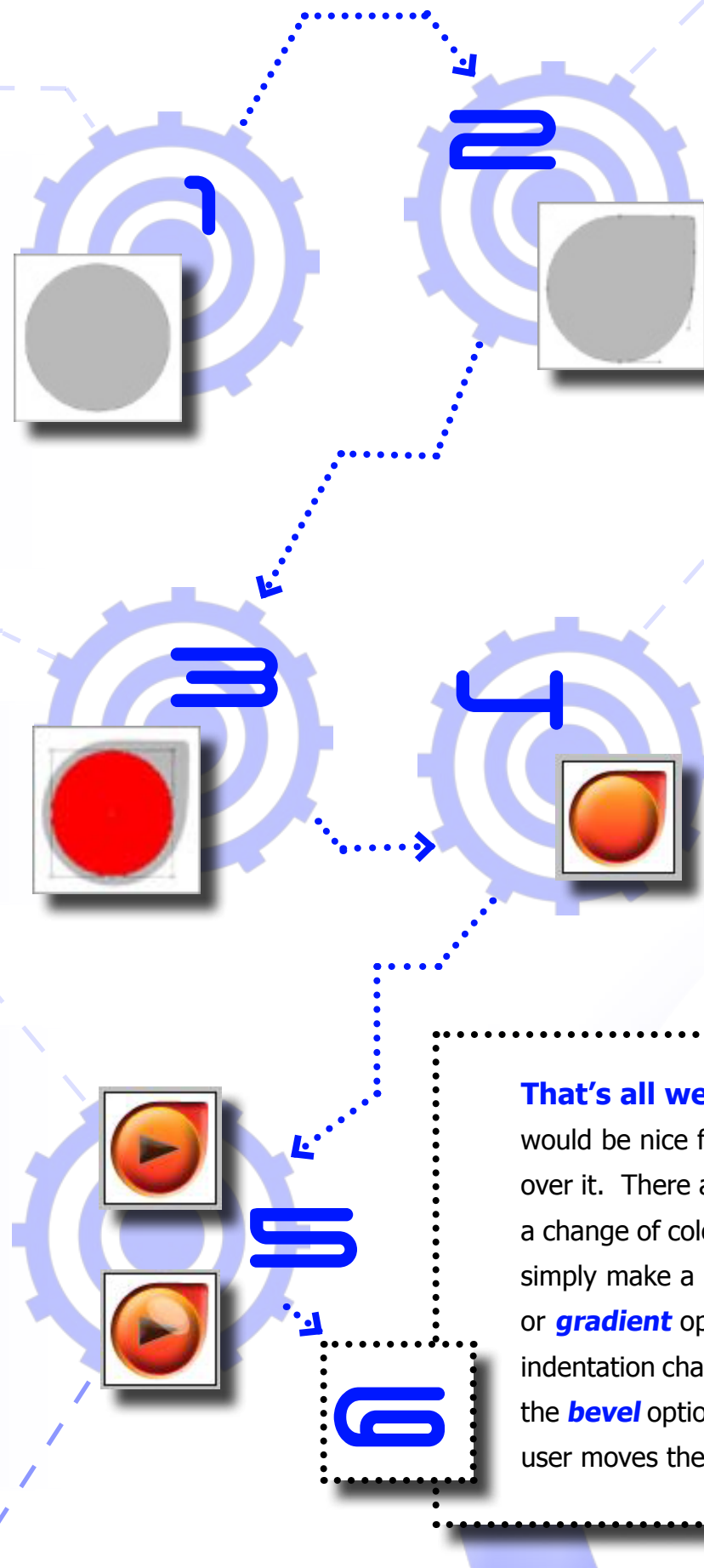
Other components that you require for a game's GUI can easily be created using the techniques of vector images and other methods used in this and previous tutorials.

Pictured is a very basic menu screen for a game that contains the components we will create in this tutorial

The first thing we need, as usual, is a clear canvas to work on. Create a new Photoshop project. The first component we will create is a button; we will create a small button that would usually be the “bullet” for the text that it represents. Make a new canvas of **64x64** pixels in size. We are going to start the button off as a vector image. Draw a circle that will fill most of the space as seen below. You may make the fill the circle with any colour you desire. I used gray.

Now draw another circle shape slightly smaller than the previous one using the **Ellipse** tool. Make sure it fits within the previously created shape. It should look similar to the following.

Now we need to indicate what the button does. We will make this specific button the “play” button for the game. Create another shape but this time it is going to be a custom shape that you draw. Select the **Pen** tool and create a triangular “play” symbol. Adjust the shape’s size using the **transform** options available in the Edit menu. You may apply **blending** options to this symbol to suit the rest of the button; the **opacity** of the symbol can also be adjusted to blend in better. After all the changes the button should resemble something similar to the following.



When you are happy with the circle you have just created, select the **Convert** point tool and add another three points to the top right part of the circle. Press the **CTRL** key and drag the points or use the point’s “handles” to adjust them to look similar to the image below. Note: you do not need to explicitly follow the shape used here; you are free to experiment and create some other shape that appeals to you.

Now we can use the **blending** options available to give the button all the effects and glamour that it needs. Make use of the **Bevel**, **Gradient** and **Stroke** blending options to give the button an awesome 3D look. Play around with the options to see what each of them is capable of; also change the lighting direction and shadows to give the button depth. After applying all the blending options, the button should look of similar quality to the following.

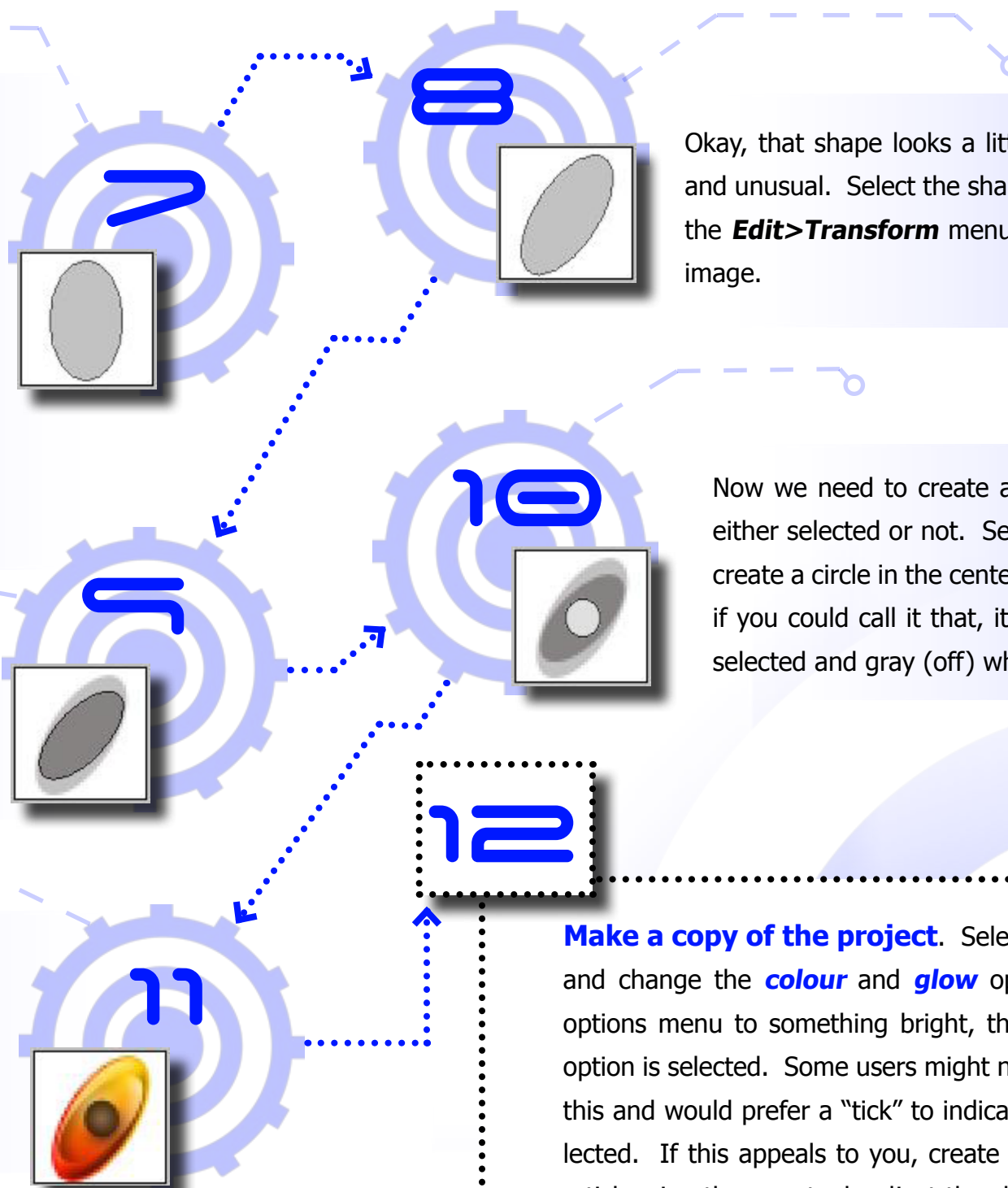
That’s all we need for a basic clean looking button. Now it would be nice for the button to react when the user moves the mouse over it. There are many different effects available for this. We will use a change of colour or an indent in the button. For the change of colour, simply make a copy of the project file, open it and change the **colour** or **gradient** option within the blending options. To make the buttons indentation change, make a copy of the project, open the file and adjust the **bevel** options to an appropriate level that will be effective when the user moves the mouse over the button.



The next component we will create is a check-box image. This will also consist of two images; one to indicate that the option/box is not selected and another to indicate that the option/box is selected. To start, create a blank canvas of **40x40** pixels in size. We don't want the box to be too large or too small so this size is suitable. Usually check-boxes are rectangular in shape, hence the name "check-BOX" but since our button theme is round and elliptical, let's make the check box a round one. Select the **Ellipse** tool and draw a ellipse similar to the following.

Duplicate the shape and scale down to a smaller size, then **Rotate** it slightly so it fits in the bigger ellipse.

The image looks very dull and boring at the moment. Apply the usual **brush** options to the shapes to give them some depth, colour and interest. I used a crimson colour scheme to correspond with the button created earlier



Okay, that shape looks a little boring; we going to make it funky and unusual. Select the shape and then select the **Skew** option in the **Edit>Transform** menu. Make the ellipse look similar to the image.

Now we need to create an indicator to show that the box is either selected or not. Select the **Ellipse** tool once again and create a circle in the center of the image. This will be a "light", if you could call it that, it will be green (on) when the box is selected and gray (off) when the box is not selected.

Make a copy of the project. Select the inner circle shape and change the **colour** and **glow** options in the **blending** options menu to something bright, this will indicate that the option is selected. Some users might not feel comfortable with this and would prefer a "tick" to indicate that the option is selected. If this appeals to you, create a shape that resembles a tick using the **pen** tool, adjust the shape using the **convert point** tool and apply the appropriate **blending** options to it to make it look attractive. The complete image in both cases can look similar to the following.

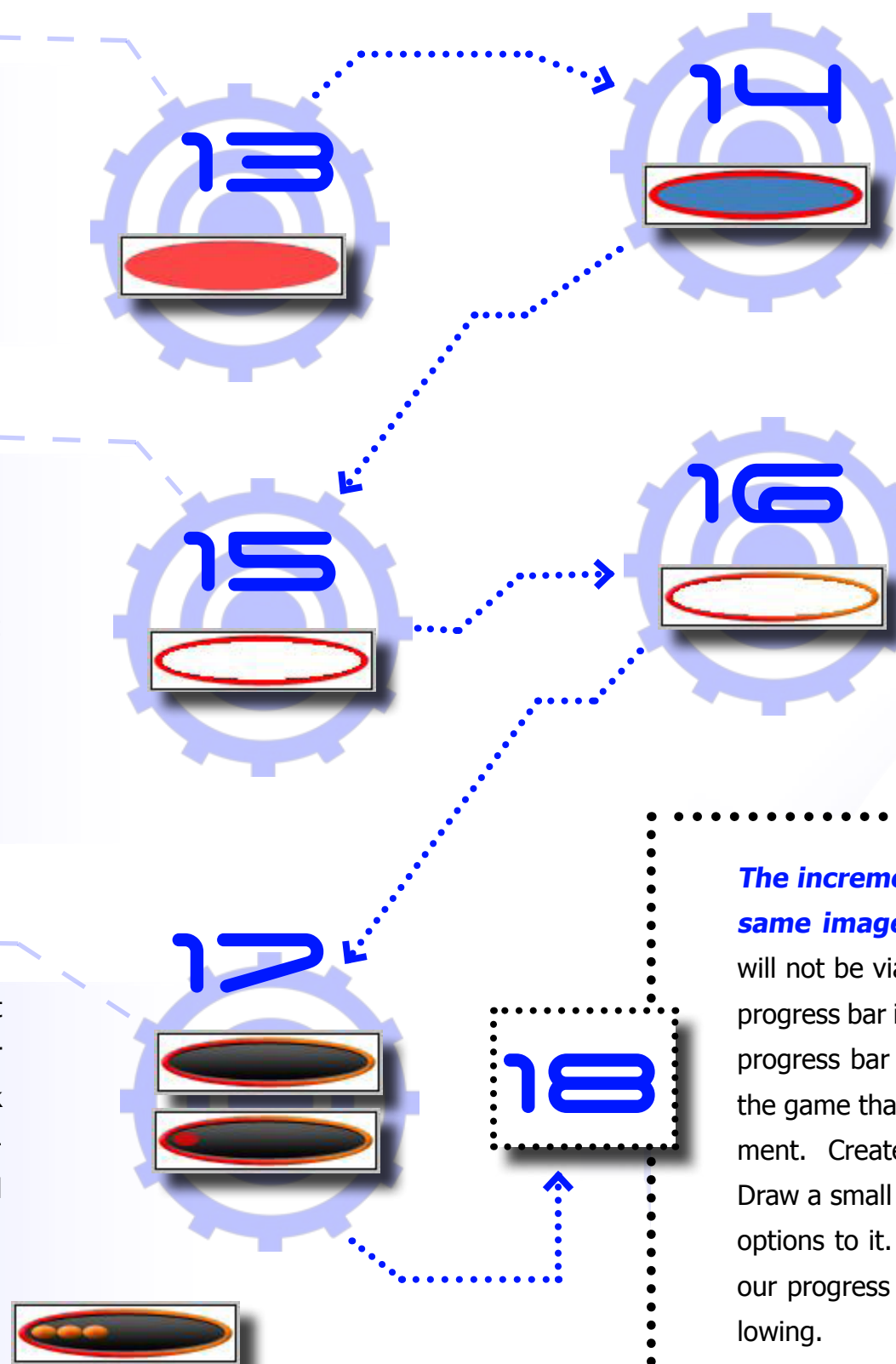


The last component that we will create is a progress bar. Create a new canvas of size **32x128** pixels in size. To stay with the same theme we have been using thus far, we will also make the progress bar an elliptical shape. Create an elliptical shape resembling the following.

The red space you see will be the frame of the progress bar. **Rasterize** the two shapes. Select the **Magic Wand** tool and select the inside portion of the smaller ellipses shape. Now select the larger ellipses shape in the **layer window**. Choose the **erase** tool and erase the **fill** of the larger ellipses. With the smaller ellipses hidden the image should look as follows.

Make the smaller ellipses visible again and adjust its **blending** options such as its **gradient** colour option and **stroke** option. The image should look similar to the image depicted. This is our completed basic progress bar. Draw a shape that will represent the increments of the bar.

The image (right) depicts what the progress bar could look like in a working situation.



We are creating a frame for the progress increments, so **duplicate** the shape we just drew and scale it to a smaller size as seen below.

Apply **blending** options to the shape, including the **bevel** option and **gradient** colour option for an attractive look.

The increment image cannot be drawn in the same image as the progress bar itself. This will not be viable when creating a game unless the progress bar is a fixed animated image that runs the progress bar without considering the processing in the game that would usually cause the bar to increment. Create a new canvas of size **16x16** pixels. Draw a small circle and apply some basic **blending** options to it. This will be the increment image for our progress bar and would look similar to the following.





Object Pascal has a lot to offer...

www.PascalGameDevelopment.com

Blender, in the right hands, can be more than simply a modeling tool. It has the ability to simulate many things that enhance your renders and, particularly, your animations. Among these are the soft- and rigid-body physics simulations and the liquid dynamics simulator. We'll take a look at each of these in turn, starting with what is possibly the most visually appealing and potentially useful one: The recently implemented offshoot of the soft-body simulator, cloth physics.

BLENDER

CLOTH SIMULATION

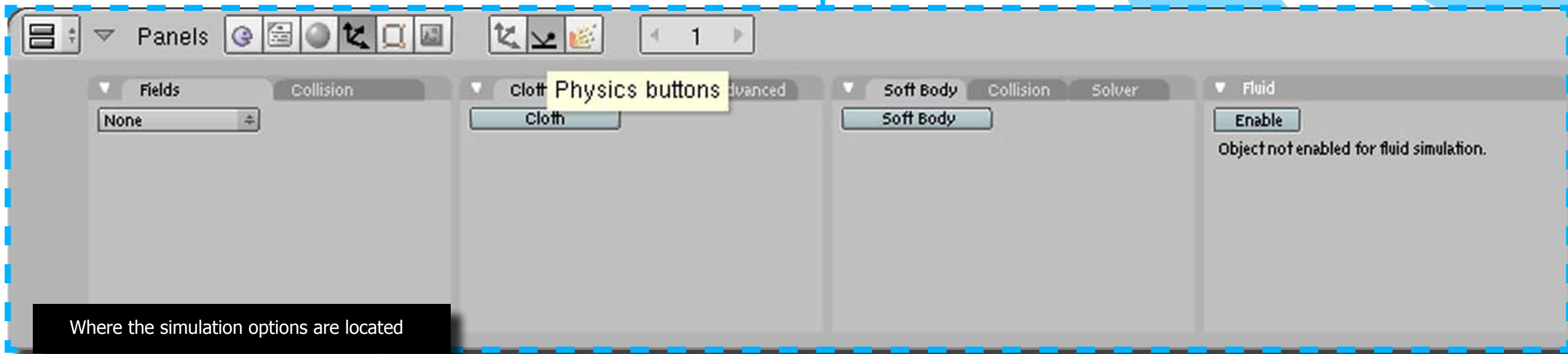
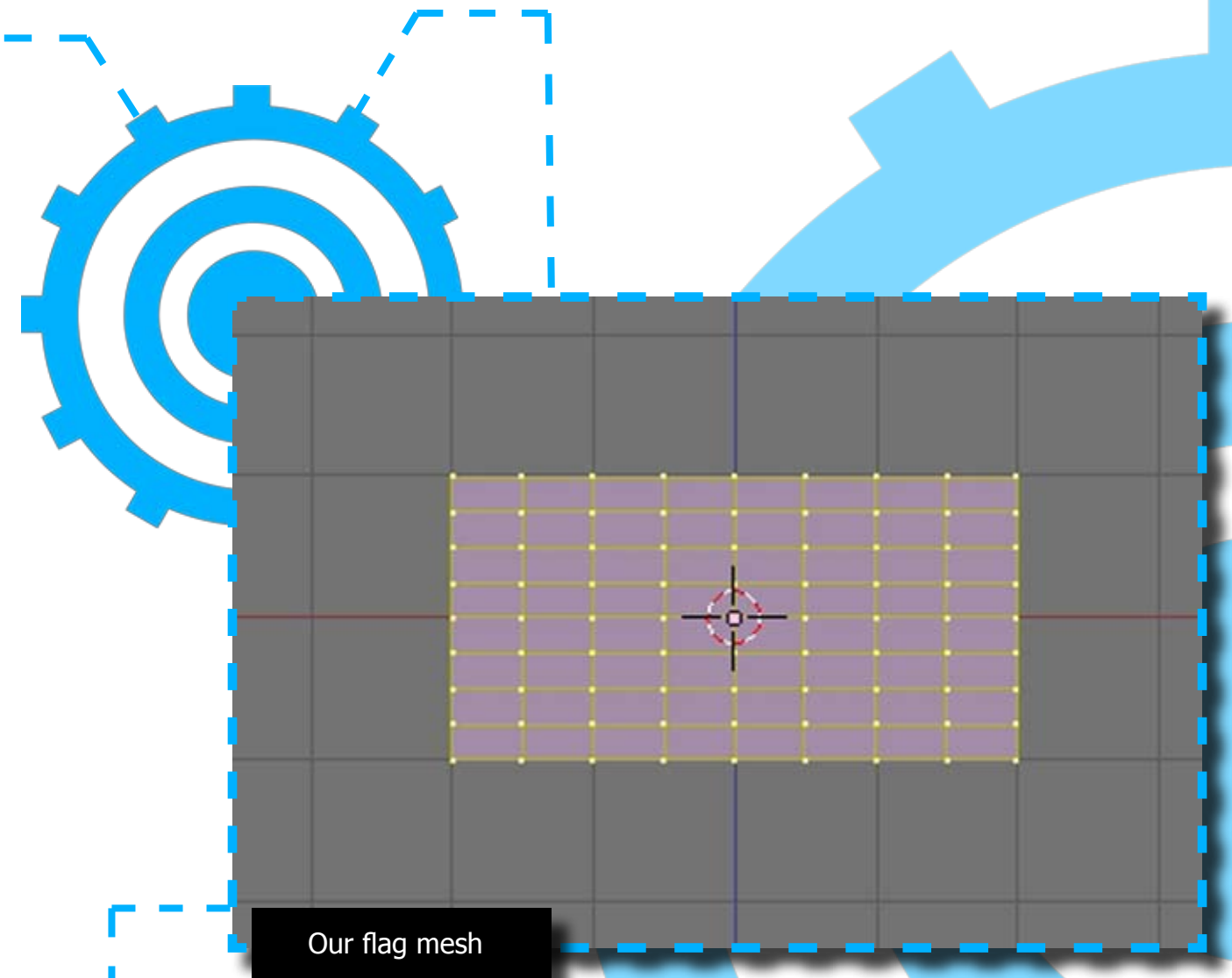
🌀 Claudio "Chippit" de Sa

 **DEV MAG**
CREATE · DEVELOP · EXPERIENCE

First off, you'll need to head over to Blender.org and download the very latest release of Blender (at the time of writing), version 2.46. As always, it includes numerous changes and improvements, most of which were implemented to facilitate the development of Blender's second open movie, *Big Buck Bunny*.

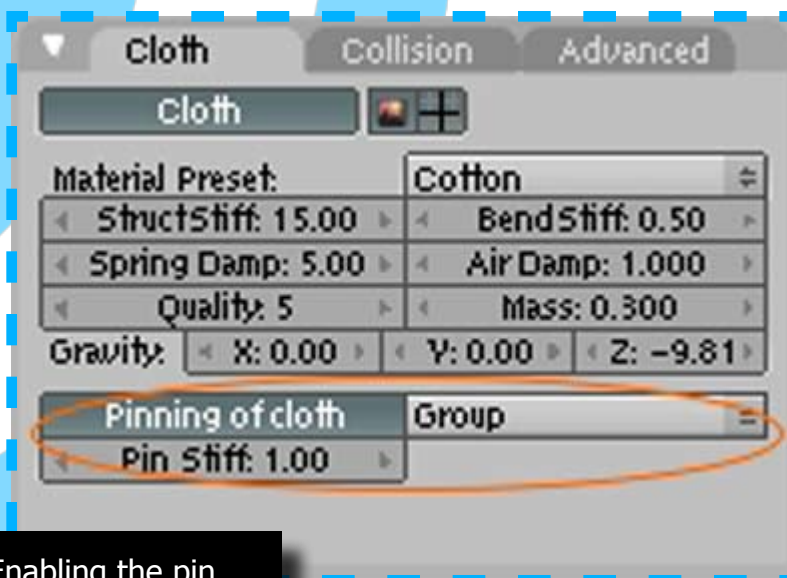
Once you've got version 2.46 up and running, start off by adding a plane in front view and stretching it slightly along the X axis. This will be the plane that represents the cloth we're going to be simulating. Blender simulations make use of the vertices of our meshes to perform calculations, so subdivide the mesh a few times to get a decent amount of vertices for the simulator to work with. More will increase the quality of the simulation, but will also substantially increase simulating times.

All of blender's physical simulation options are located under the 'Physics Buttons'. For now, all we're going to be dealing with are the options under the Cloth tab. Make sure that your new subdivided plane is still selected, and then click the Cloth button to enable Cloth simulation on that plane. That's all you need to do to get the simulator to run, and you can use Alt-A to preview what it would look like. At the moment, all it will do is fall under the force of gravity, but we'll work on that now.





Creating the vertex group



Enabling the pin

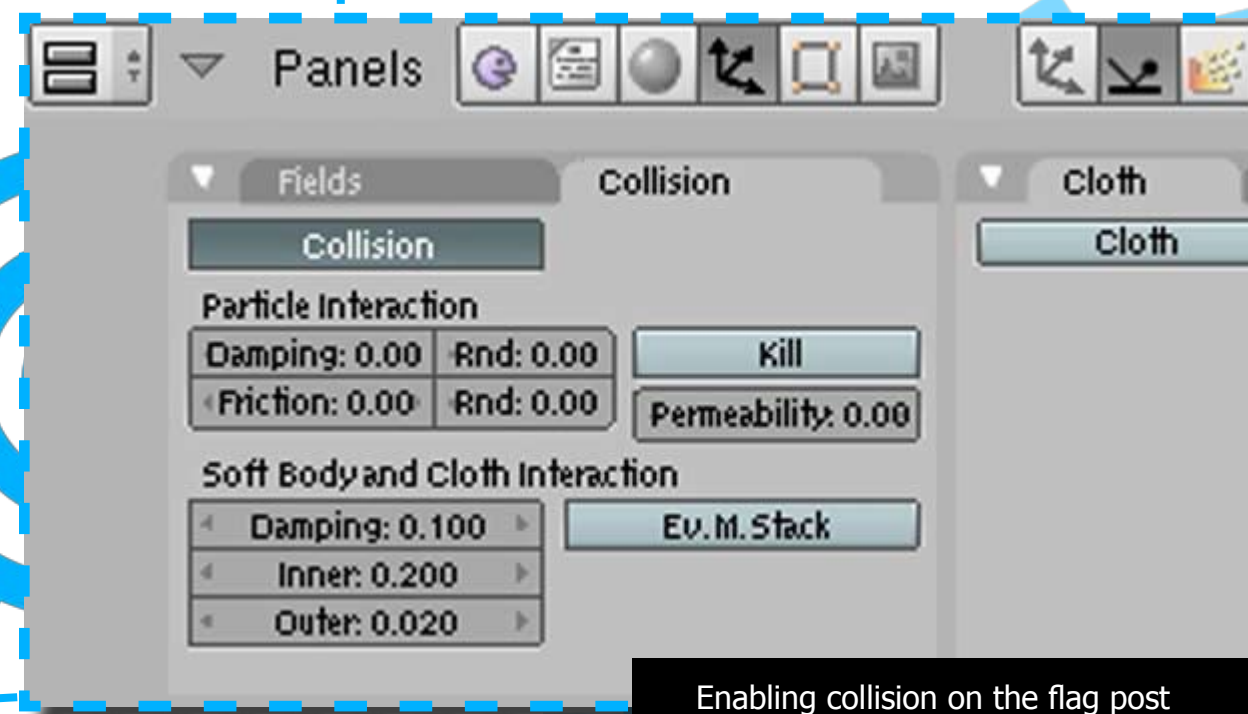
Firstly, we're going to 'pin' some of the vertices of this plane so that it hangs like a flag. This prevents these vertices from moving under the simulation and gives you a way to attach the cloth to something. To do this, we need to define a new vertex group. In edit mode, select the top-left and bottom-left vertices of the plane and, under the Links and Materials tab of the Editing menu, click 'New' to create a vertex group, and then 'Assign' to add the selected vertices to the group. You can name the group too, if you like, to make it easier to find it later.

Back in the cloth tab, click the 'Pinning of Cloth' button. You'll see all the vertex groups appear in a drop-down box to the right of the button. If your mesh only has one, it should be selected by default. If not, select it by the name you defined earlier. If you run the simulation now, you'll see that the flag still falls under gravity but it is now held in place by the two vertices we chose, as if it were attached to a flagpole.

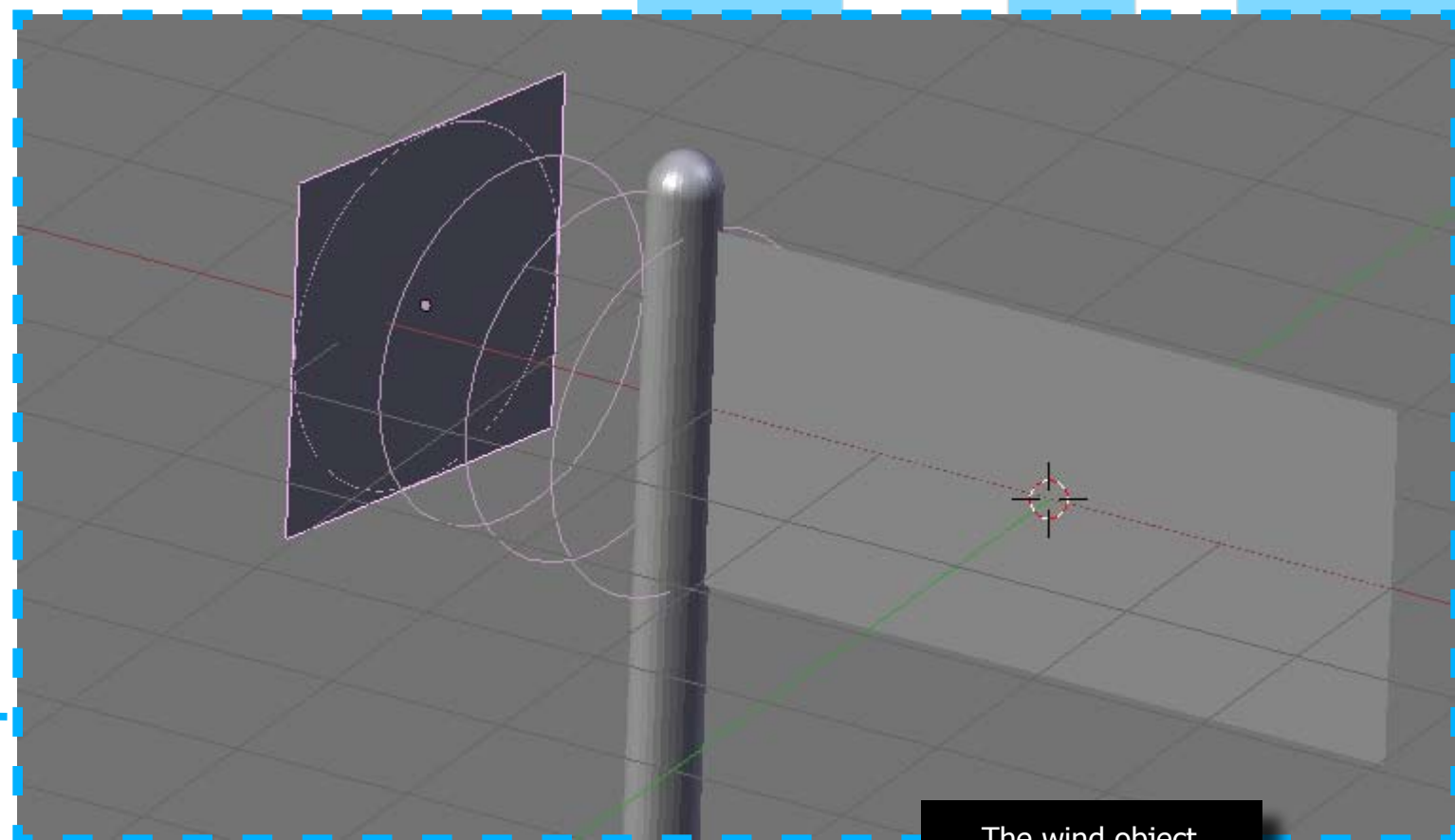
For effect, you can create an actual flagpole mesh just next to the plane so that it no longer appears as if the cloth is floating in mid-air. I simply used a tall, narrow cylinder capped with half a sphere for my post. When you do this you'll notice that, upon simulation, the cloth may pass through the flagpole. This is obviously an undesirable effect, so we'll have to enable collision on our flagpole. We do this from the same Physics Button menu as we used before, with the Collision tab grouped with the Fields tab. Simply click the Collision button there with the flagpole object selected and the flag will now collide with that object.

Now we'll twist things a little bit, and let this flag flap in the wind. In side view, create another plane, and move it so that it is situated behind the flagpole and facing towards it. I turned it slightly at an angle so that the wind glances off the flag. In the 'Fields' tab, select 'Wind' from the drop-down list, and adjust the strength of the wind. A value of around 12-15 should work well. You'll notice that, when you enable the wind and adjust its strength, circles will be projected outwards from the plane in the direction the wind will blow. This is the same as the direction of the plane's normal. Assert that is facing the right way.

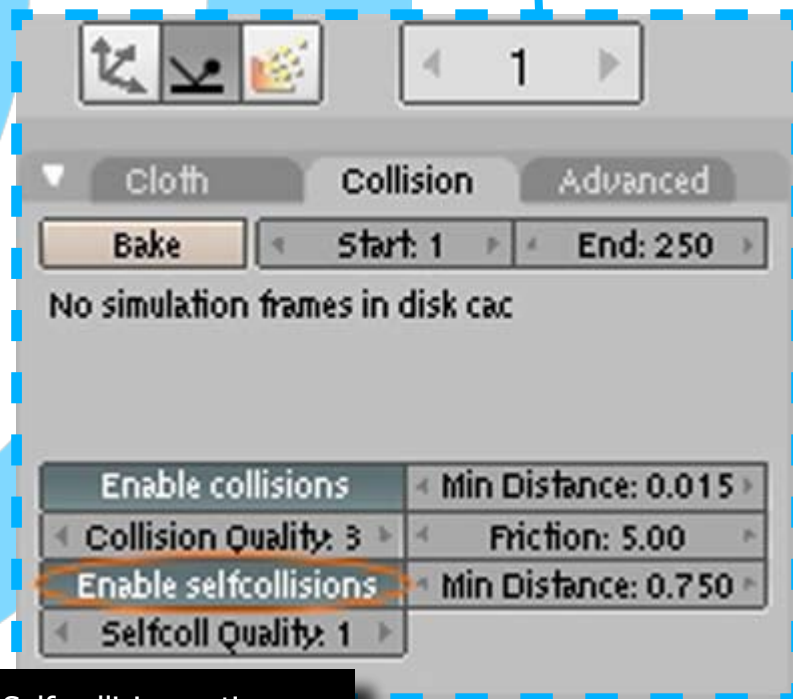
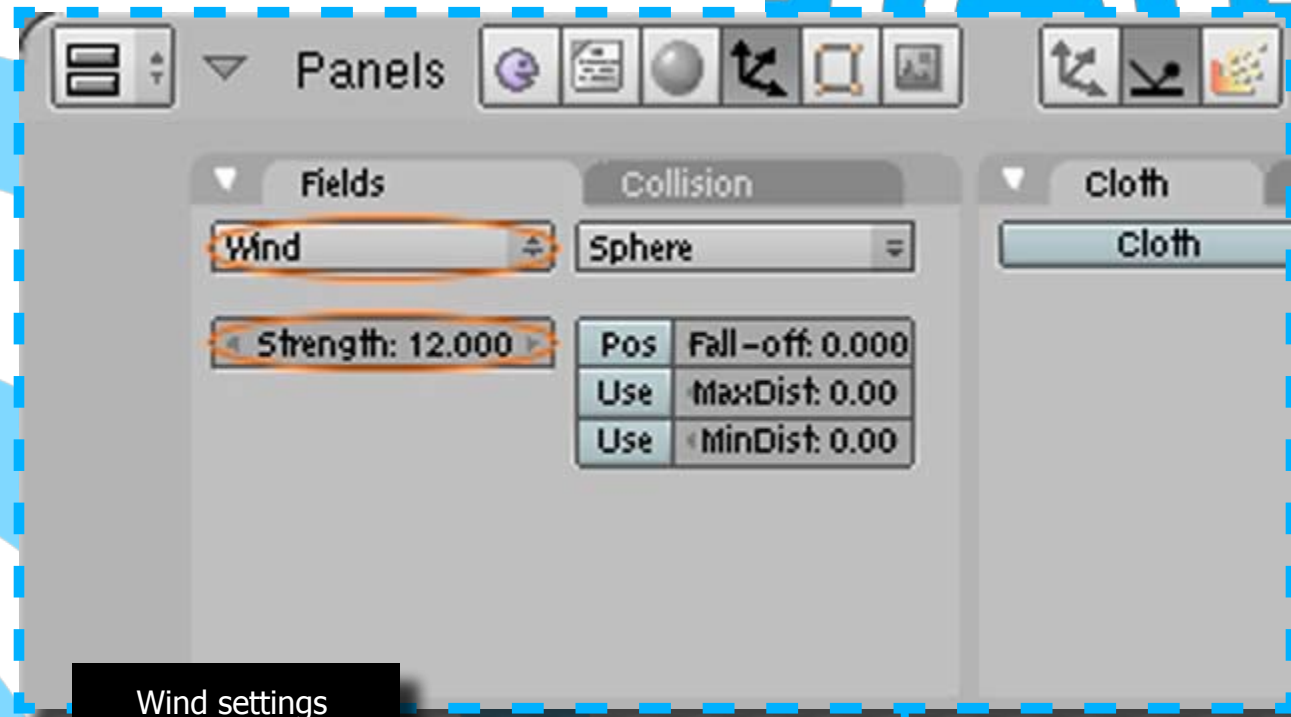
To illustrate that the wind effect works with realtime changes to the plane, I added some animation keyframes so that the plane would move in relation to the flagpole, effectively changing the direction from which the wind blows. I also enabled self-collision on the flag so that it would not pass through itself. This isn't always necessary in cloth simulations but it may happen in this situation, so it's best to enable this to prevent it from occurring. You'll find this option under the other Collision tab, next to the main Cloth tab. You may also want to enable smoothing on the flag so that the individual faces are not so blatantly visible in your render.



Enabling collision on the flag post



The wind object



Additionally, I texture mapped an image onto the flag. Note that, in this version of blender, face-select mode has been deprecated and all UV mapping is done from edit mode. Other than this change the process to map an image onto the flag remains the same as was handled in previous issues. When you're certain that your simulation works correctly, you can use the 'Bake' option under the collision tab where the self-collision option was located to pre-calculate all the animation frames. This means that Blender will not have to recalculate on every render or animation preview. You may want to set the material properties on the wind object so that it is not visible in renders, or move it to a different layer when rendering. Note that all particle based effects –such as cloth and soft bodies – are only affected by fields in the same layer in which they reside; if you move the wind object to a different layer you need to have baked the animation for it to render correctly. And to make changes to the animation, you'll need to move it back into the same layer.

That's all for this tutorial. Play around with the simulation settings to see how they affect the simulation output. As always, the .blend file as well as my animation render will be available from the website's content section. Good luck, and have fun. 🌀

ED BY DAY...



Claudio "Chippit" de Sa



Rodain Joubert. He isn't dead.

Nandrew, known to readers as Rodain Joubert, our former editor and the most prolific asset to Dev.Mag; but this is not all he was. In addition to the time he has invested to make this publication what it is, he also contributed heartily to Game.Dev itself, our community of game developers and designers. He created numerous quality games, above par in all instances and, more often than not, superior to other community offerings at the time. Some of his greatest creations, brought to fruition during the time he was working for Dev. Mag, are listed below:

LINE WARS

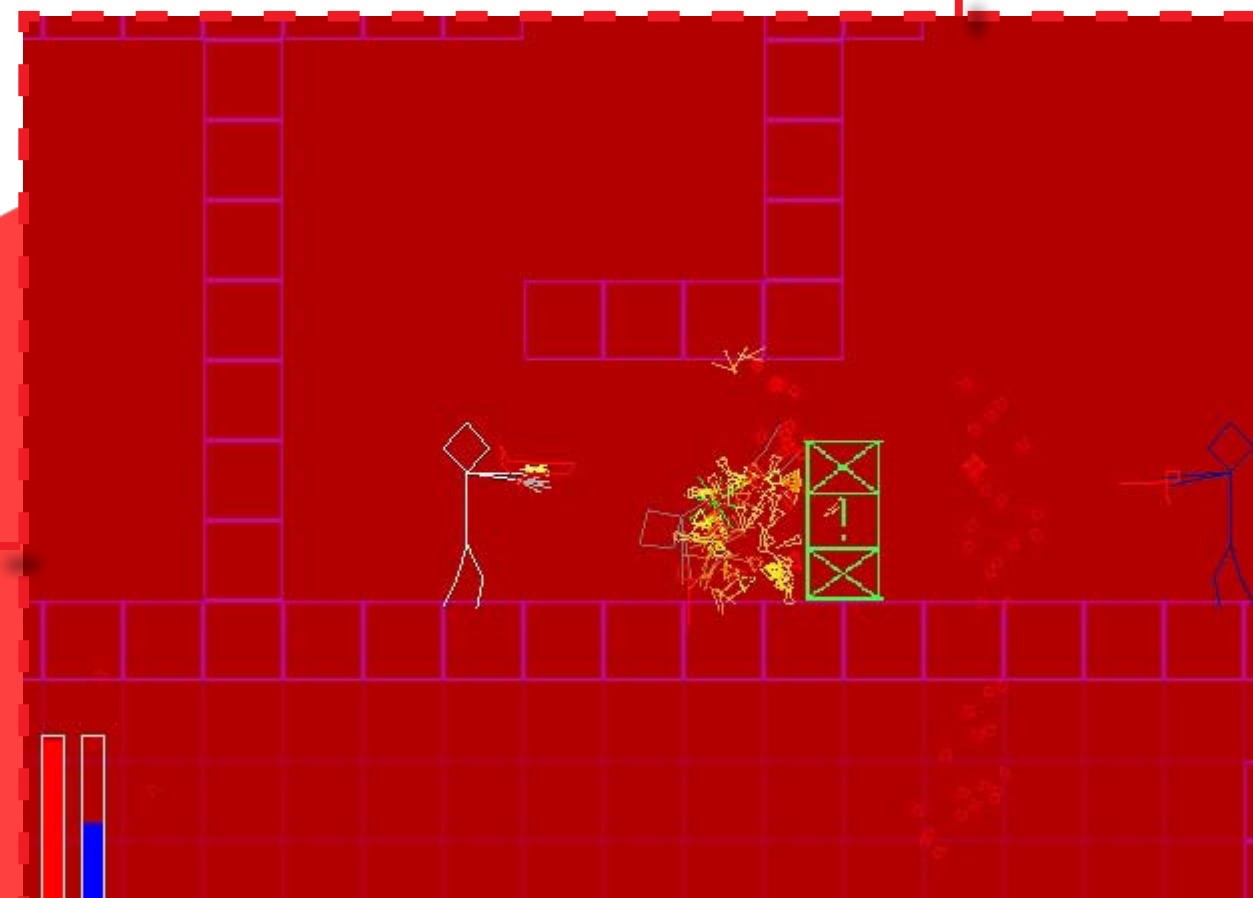
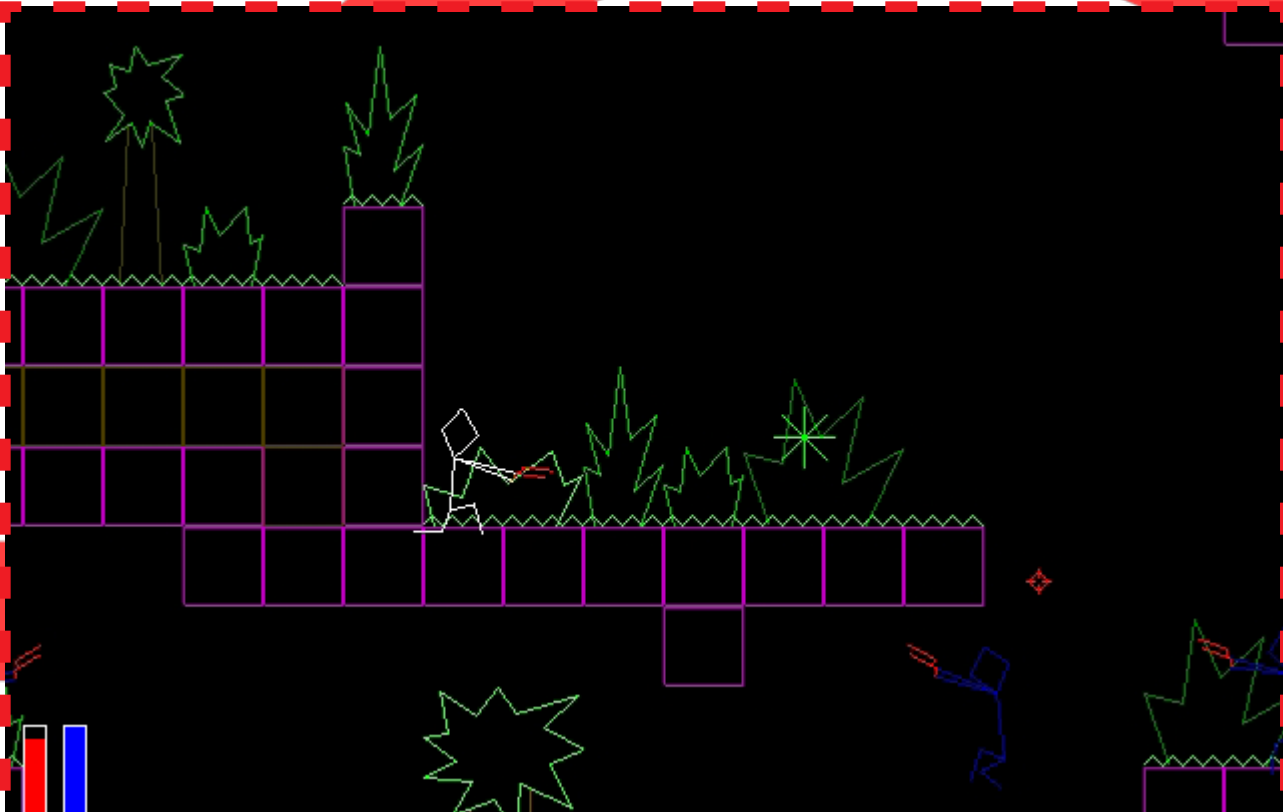
START GAME
LOAD GAME
QUIT GAME

LINE WARS

A stylized platform shooter, Line Wars was a side-scroller worth mention because, in an effort to ease the art requirements of the game, Rodain had created a limitation for himself: to only use lines in the game artwork. And, in doing so likely, he created his most visually appealing title.

Double jumping around while simultaneously firing 1-dimensional doom at enemies was incredibly satisfying, as was watching the enemies tumble away at the force of the blast. The time-dilation ability was also incredibly smoothly implemented and served to make the otherwise potentially difficult encounters challenging but still fun.

<http://www.devmag.org.za/uploads/LineWars.zip>



ZTICKY ZLIME

Zticky Zlime took everything that was fun about the Ninja Rope 'weapon' from the 2D Worms series and made a game about it. Using a very simple but intuitive mouse-based control scheme, the player guides a rather flexible green slime through the game, using nothing but his ability to roll and his ability to project a sticky mass at a wall or ceiling which he can then use to swing.

Later, the player will earn alternate forms that allow him to evade or even directly combat enemies. These grant the player extra abilities such as being immune to projectiles, or becoming heavy enough to smash through obstacles. The player also earns other upgrades that increase your effectiveness, but, frankly, the game is all about how fun it is to swing from walls.

<http://www.gamedev.za.net/filecloset/download.php?id=304>

ZTICKY ZLIME

(N)EW GAME
(L)OAD GAME

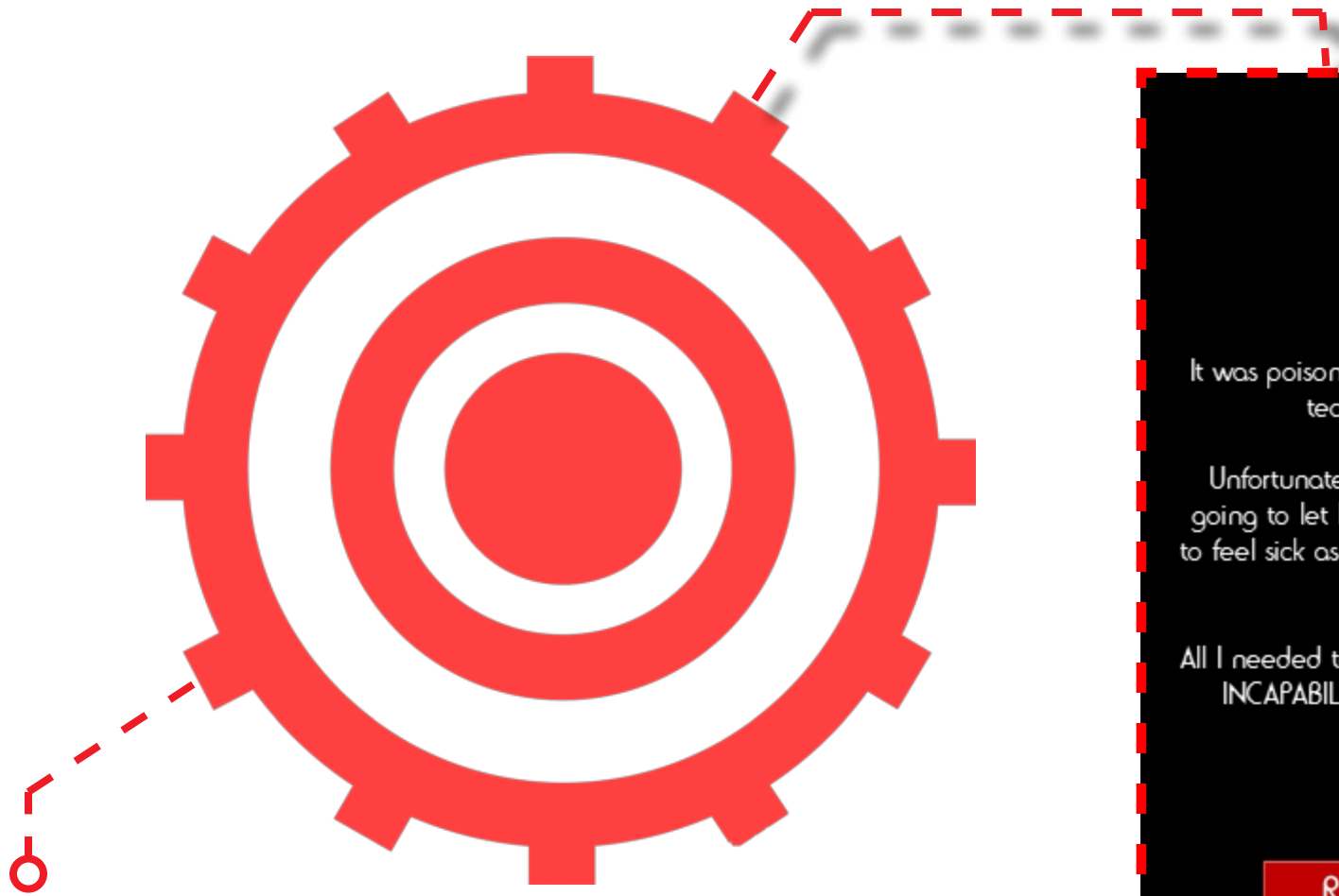
A DELICIOUS DEMO-SOMETHING BY RODAIN "NANDREW" JOUBERT

Discovered: LIMITATIONS

Fun fact #8273: there is 0% adhesion between Zeflon and any other known substance, making it an ideal coating for use in sanitised conditions like this! After all, janitors need to quickly and effectively clean up lab fluids, blood stains and dead Zlimes! Those Zeblogg scientists sure are smart, aren't they?

'Zeflon : tougher than grime, tougher than Zlime!'
Out of all the HoloTV slogans contained in my neuronet, that's my favourite! What's yours?

(IF A SURFACE IS COATED IN ZEFロン, YOUR ZLIME TRAIL WON'T STICK TO IT)



WHODUNNIT ...THINK QUICKLY

A study in immersion and pacing, Whodunnit is an unforgiving film noir-styled detective romp that offers up little assistance to the player in even less time. The player, nursing an unhealthy addiction to the calming effects of nicotine, is thrust into a generic detective setting: A hotel guest has been poisoned, the protagonist himself begins to feel the effects of the toxin corrupting his bloodstream and has but 10 minutes to interrogate the remaining surviving patrons to discover, by means of elimination, who is guilty and force that person to offer up the antidote to save the lives of the innocents – as well as his own.

Whodunnit succeeds remarkably because it uses both limited time as well as suspenseful music and sudden twists to create a sense of tension previously unheard of in Game.Dev creations, and it more than certainly deserved its competition victory.

<http://www.gamedev.za.net/filecloset/data/files/357/whodunnit.exe>



WHO IS NANDREW?

A tribute and invasion of privacy by **Quinton "Q-Man" Bronkhorst**

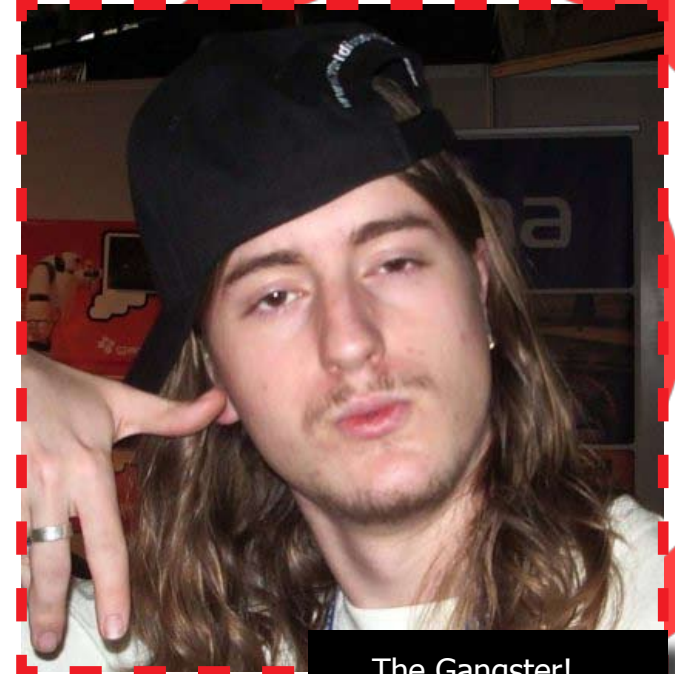
We all know Nandrew. He's that awesome guy with many talents; he was a dedicated editor, a creative and innovative programmer, and sexy as hell. But do we **really** know him? Do we know the **true** Nandrew? This Author went undercover, and stalk- er, closely followed Nandrew in his every day life* and found out that Nandrew isn't who we all think he is. A Man of many mysteries, and many talents, indeed...



The Viking!



The Dominatrix!



The Gangster!



The Power Ranger!



The Rock Star!

GEAR COUNT:

206



ONLINE

DEV MAG

CREATE • DEVELOP • EXPERIENCE

WWW.DEVMAG.ORG.ZA