ISSUE 6 2006

DEV.MAG

CREATE • DEVELOP • EXPERIENCE

PROJECT MANAGEMENT
PART 2

BLENDER
TUTORIAL: PART 2

QUALITY TOUCH
THE USER EXPERIENCE

MOBILE GAMES:
HOW TO CREATE GAMES USING JAVA PART 4

SOUTH AFRICA'S FIRST GAME DEVELOPMENT MAGAZINE

REVIEWS : SULACO FEATURE : MAKING A GAME IN 48HRS OR LESS. SEPARATIONISM RANT

DEV.MAG ISSUE 6 2006

CONTENTS

ED'S NOTE

RAge is only a month away. This will mark a considerable change in the focus of this month's issue towards rAge and the game development talks being hosted there. With a large proportion of our staff being presenters at the event it'll definitely be worth your time. So make sure you read the write-up (Nandrew's article) of what to expect there, and don't miss out.

NAG's Game.Dev Comp 10 begun on the 1st August. The goal: Create a simple, yet fun, management game. With everyone attempting to get their grubby hands on the R5000 (1st place prize) it's got the makings of a great competition. If that's not enough to tickle your taste buds, then there's also the R1000 cash prize to the best new entrant. ('New' meaning anyone entering the Game.Dev competition for the first time)

Best of all it's free to enter, so make your way over to the Game.Dev section of the Nag forums! (www.nag.co.za)

Editor
Stuart "GoNz0" Botma

DEV.MAG
CREATE • DEVELOP • EXPERIENCE

THE TEAM

RANKING OFFICER
Stuart "GoNzO" Botma

SECOND IN COMMAND
Rodain " Nandrew " Joubert

DESIGN SQUAD
Brandon "CyberNinja" Rajkumar
Paul "Higushi" Myburgh

CEREBRAL SOLDIERS
Simon "Tr00jg" de la Rouviere
Ricky "Insomniac" Abell
William "cairnswm" Cairns
Bernard "BurnAbis" Boshoff
Danny "dislekcia" Day
Andre "Fengol" Odendaal
Yuri "knet" Oyoko
Heinrich "Himmler" Rall
Matt "Flint" Benic
Luke "Coolhand" Lamothe
Greg "Zphyr" Reveret
Geoff "GeometriX" Burrows

WEB WARRIOR
Claudio "Ch1ppit" de Sa

WEBSITE
devmag.googlepages.com

To join, make suggestions or just tell us we're great, contact:
devmag@gmail.com

This magazine is a project of the NAG Game.Dev forum.
Visit us at www.nag.co.za

DIGITAL STOMPIES

NEWS

http://www.agdinteractive.com/

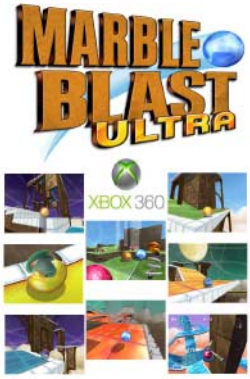
Anonymous Game Developers Interactive (AGDI) is a game development group working on remakes of the classic Sierra adventure games. If you haven't heard of them already, head over to their website, take a look around and be sure to download their very own versions of King's Quest 1 and 2. The team uses the highly popular Adventure Game Studio (http://www.adventuregamestudio.co.uk/) to create their games and are using it for their upcoming Quest for Glory 2 remake as well.

http://www.garagegames.com/

GarageGames has recently updated its Torque Game Builder software, bringing it up to version 1.1.1. Torque Builder is a 2D game framework which has the capability of running games on PC, OSX and even Xbox360. Although the $100 price may be a bit much for some casual developers, the software is of a professional standard, is easy to use and requires no royalties should you decide to distribute or sell one of your creations.

http://www.garagegames.com/mg/snapshot/view.php?qid=1123

Indie games can make it next-gen too! Marble Blast Ultra, a new Xbox360 title designed to take advantage of its multi-player capabilities, was developed for the console by GarageGames and has already received decent reviews from many online sources. Marble Blast Ultra in centred around trying to navigate your marbles through vast 3D levels, using the terrain and obstacles to collect gems and reach the exit as fast as possible. It also features competitive online play and leaderboards



http://www.gamasutra.com/php-bin/news_index.php?story=10450

Gamasutra brings us a postmortem from the person behind Stubbs the Zombie, Alex Seropian of Bungie (the name behind Halo). Seropian explains how he set out to create Wideload after leaving Bungie, what he wanted to achieve with the company and what sort of commandments were essential to professional and personal happiness. It also goes into great detail on the design process of Stubbs the Zombie, something which is definitely a worthwhile read.

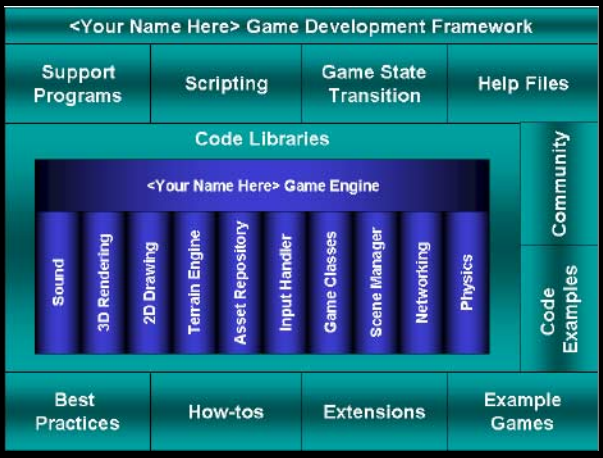
Making games in 48 hours

So suddenly today you've found out about the latest and greatest Game Development Competition (speaking of that, have you seen the NAG Game.Dev contest number 10 yet?) and now you need to get a game out as quickly as you can.

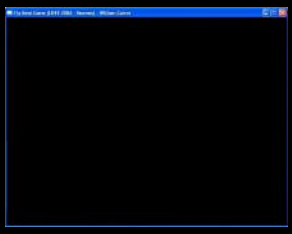
Well it's certainly possible to get a complete game out within the 48 hours you may have left. To make a very rapid game development project a success you need a plan, the important steps you will need in your plan are highlighted in this article.

As indicated before (Dev.Mag issue 3), a game framework is critically important to very rapid game development. There is no reason you should be spending time working out how to create a splash screen for your game when you only have 48 hours to do the complete game. Of course you need to know the structures and rules of the framework to be able to use it properly. Grab your favourite framework and start your project. In fact a splash screen is a really good place to start developing until you have an idea of what you actually want to do.

In many ways a game consists of two clearly different aspects. There is the core gameplay and then the peripheral extras like splash screen, credits etc (Refer to Quality Touch - Dev.Mag no 4 for a full list). If the theme of the contest doesn't immediately give you a Genre defining game idea, spend some time getting the peripheral screens complete. Create some generic theme images and add them as backgrounds while creating a quick menu system (quick because the basics are already in the framework). Start working on the Help screen, you can always populate the contents once the game works. Another thing to do while waiting for an idea is to start working on backgrounds and even theme relevant images that you might be able to use later, for example for



Your Framework is your friend.



Everyone starts from Nothing.

FEATURE

a dragon theme make a small dragon sprite, you might only use it for the games icon but it is an asset you'll already have.

Getting a great theme based idea is often the hardest part of the very short contests. Some themes may give you immediate ideas, while others may take some time to give you ideas. The most important thing about whatever idea you get is that it must be do-able. When you get your idea ask yourself, "How long will this take to do?" Just remember that you should at least double, triple or even quadruple your time estimates to get a better idea of how long its actually going to take! If the answer is more than the time allowed you may need to simplify it. Complex ideas can often be simplified, for example that cool RTS idea could be turned into a turn-based game instead.

A good strategy for getting the game idea into the game is to prototype it within the Game screen of the framework. If the prototype takes too long at least it's already in the framework and can be entered into the contest. The prototype should include the key design ideas of the game, this includes all the ideas that will make the game fun, and must be clearly linked to the contest theme. Unlike normal game prototyping where you only need graphics once the game play is ready you should be including the graphics as you develop the prototype. Once the prototype is done your game should be playable, it should clearly comply to the theme of the contest and have the key aspects of the game play. Typically at this point you could stop working on the game and still submit it to the contest as a complete game. Once the basic ideas of the game are working complete the peripheral screens. Finish the help screen, the splash screen and the default values for high scores. It is important to remember that the completeness of the game is heavily impacted by the quality of the other screens and not only of the game idea you are implementing and you should therefore complete these screens before polishing your actual game.

Based on the time left in the contest you need to decide if there is time to polish the game play, adding all the little extras that will earn you bonus brownie points, or if it's time to sleep. Typically by this point you'll be pretty sick of playing and replaying your game, you are getting tired from not enough sleep and your family is probably wondering where you are. If you submit your game at this point you'll probably be in the middle of the pack. Spending some time to add all the polish extras could lift you out of the pack and make your game something special.

If you decide you have enough time left to make improvements on the game, and you really can go the full 48 hours without sleep, it is time to add some polish. Spend your time wisely as you want to make as many very visible changes as you can, with the least amount of work. Good places to find these sort of changes are, improving the background images, better special effects, more sound effects and possibly adding a credits screen if you left it out earlier. Another good idea for polish is to spend some time making the game more intuitive. Make menu options more logical, make better transitions between the game states and even spend some time allowing the player to set more and more options. When it comes to polish you really do not want to optimise code, or find better ways of doing things that already work. In a contest the only polish that counts is the bits that the judges will be able to see and feel in the game.Bugs are a terrible reality in all computer software development and in game development they can be terribly elusive to find. Whenever you reach a point in your game development where you have a stable, working game you should make a backup. The number of people who have wanted to enter a contest and didn't because their last change broke something is very high, don't fall into this trap. Whenever you encounter a bug while play testing the game, you must try and sort it out as quickly as possible. If trying to fix the bug takes too long, rather roll back and recode the latest idea. As a great mind once said "If its complicated to code, its even more complicated to debug!"

Entering extremely rapid game development contests is great fun. You know you are competing against people with similar limitations as yourself, and the winners are typically the people that know their tools the best. By entering these contests you are pushing your limits and abilities. Success in these contests comes from knowing your tools and having a good understanding of the frameworks you have available.

CAIRNSWM

smallfry mobile

Smallfry Mobile is a South African-based cellphone game developer. They started in 2005 and have released two games, abYss and Demolish, that are sold in many countries around the world ranging from South Africa to America, Australia and more. Dev.Mag recently interviewed Matt Benic, one of the creators of Smallfry Mobile, to find out more

Matt Benic

Who is the Smallfry Mobile team?

Currently the core Smallfry Mobile team consists of myself and Chris Tsimagionnis. We share programming and design responsibilities, while I also handle the business side of things. In addition, Diorgo Jonkers creates our art assets and plays a large part in the design process.

What inspired you to start Smallfry Mobile?

About 3 years ago I got a new cellphone and was amazed at the quality of the games available for it. At that stage I started looking into what was involved in developing such games. When Chris and I left I-Imagine the next year to go into more 'traditional' jobs, we both knew we needed to keep doing some kind of game dev to stay sane. We decided to start doing mobile development as it seemed like the most practical way to develop something marketable with such a small team and with so little time available

Could you tell us a bit about the games you've made?

At this stage the only game we have created as a team and released is abYss, an old-fashioned side scrolling shooter. In the game you pilot a submarine through the depths of the ocean, facing deadly robotic sealife as you persue a mad scientist out to flood the world. Chris, Diorgo and I have all been gaming for ages, and enjoy 'old school' games, so we set out to create something old school with abYss. We settled on the underwater theme for abYss because, quite frankly, space and war shooters had been done to death. It also allowed us to make use of a unique and colourful look that really makes the game stand out from similar titles.In addition to abYss, we promote and sell Diorgo's game Demolish, which is an action platformer that casts you as a lone warrior out to retrieve a magical artifact stolen by an evil wizard. We will soon be releasing a colourful typing game called BubbleBee QuickType that should appeal to the SMS-mad masses. It is a relatively simple typing action game, but it looks great and is devilishly addictive. BBQT will also be our first game to feature online high scores, and we're hoping to keep that as standard functionality in upcoming titles.

When designing a game for a cellphone, what elements have to be approached differently that most people would never even think about?

For me, the biggest issues are the controls and screen size. There have been systems before in which developers had to find ways around resource size limits and limited hardware so the solutions to these particular challenges have basically been solved, but cellphones are not designed for gaming, and game designs really have to work around that by being economical with screen space and having well thought-out control schemes.Another big element is the wide range of phones out there, each with their own quirks and issues, not to mention different screen sizes. Games should ideally be designed to minimize the impact of such designs.

What was the most difficult part in getting started?

Getting started was the easy part, keeping going is what's tough. We both have 'day jobs' and wives, and finding the time to spend on game dev is not easy. In the end it takes dedication and passion to keep going and a bit of mutual encouragement doesn't hurt. You have to be realistic about what you want to achieve, and how much of your time you can allocate to achieve it.

What tools do you use?

In software terms, everything we use (other than the operating system) is free. We develop in Java, which is available from Sun and uses the NetBeans IDE. NetBeans is a fantastic piece of software, easily rivalling paid-for IDEs in terms of functionality and usability. The phone manufacturers also provide emulators of their devices which integrate into the IDE in most cases. Our art assets are all created in Gimp, which is an extremely functional art package that more than meets our needs.On the hardware side, we need to have a variety of phones to test on, and of course these are definitely not free. We get phones on upgrade when we can, and take advantage of everyone we meet that has some unusual model that we've never tested on. An online testing service called GetJar has also proved very valuable

What has been the best moment from getting the company off the ground to where you are today?

I always get a kick out of seeing people having fun playing our games, and that has to be the best part of it. I think another great moment, in a strange way, was finding a thread on a Russian site where a guy was looking for a warez copy of abYss. Something about the fact that people liked the game enough to try and pirate it was just cool. :)

How have the local and global markets responded to your games?

That's extremely difficult to gauge, since we don't interact directly with the customers but rather sell through third parties. What player reviews we have seen of abYss have typically been extremely positive. Sales have not been what we would have liked, but we feel that's largely due to the structure of the mobile games industry and the fact that we don't have the kind of time we need to really push the games.

What approach did you take towards marketing your games?

We don't sell to gamers, but rather to distributors. The big platforms for games (cellphone networks, big web portals) never deal directly with developers, but rather with publishers or distributors that have a large catalog of games.

Do you think the current state of the cellphone game market has any more room for growth and innovation?

There is always room for growth and innovation. In the case of the cellphone game market, I think we are still waiting for the 'killer app' that will really represent the format. The market is currently saturated with retro remakes, licence tie-ins and ports of varying quality. There is a desperate need for original game ideas that really take advantage if the format's strengths, such as mobility and permanent connectivity.

Are there any useful words of advice you'd like to share for aspiring indie developers who want to start their own company one day?

Be realistic. It doesn't matter if you are planning to make cellphone games or more traditional games, be honest with yourself about what you can achieve and do that to the best of

your ability. You don't need to build the next Quake or Half Life to be a good developer, in fact if that's your ambition you're wasting the freedom to innovate that being an indie [developer] affords you!

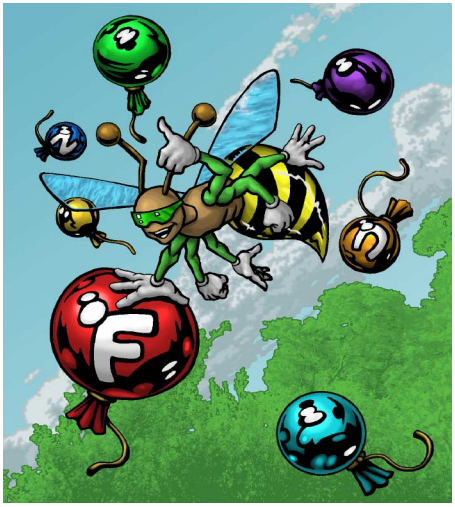
What are the future plans for Smallfry Mobile?

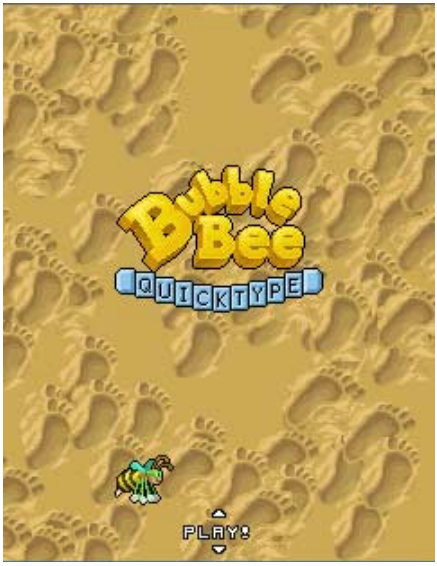
In the short term, we're hoping to release BBQT very soon. We also have another puzzle game in the works that should be available before year end. In the longer term, we would love to make Smallfry a full-time endeavour, but it's not feasible right now

Thanks for sharing your wisdom. Dev.Mag wishes you the best of luck in the future.

It's a pleasure, and thanks

To find out more about Smallfry Mobile, visit www.smallfrymobile.com or wap.smallfrymobile.com from your cellphone, where you can download a demo of abyss and a beta demo of BBQT





SPOT LIGHT

REVIEW

NeHe OpenGL

Although DirectX is the cool thing to be coding in nowadays, OpenGL still serves as a handy introduction to programming with 3D - short of all the painful base-level maths which early 3D gurus had to work in.

Yet, convenient as it is, OpenGL still needs to be learned. In this regard, there are few places out there on the Internet which are comparable to the NeHe OpenGL tutorials as far as catering to beginners is concerned. NeHe Productions deals with everything OpenGL-related, placing special emphasis on tutorials and other learning tools..

Although not strictly game development, any 3D devver wanting to get started with this would be crazy to skip out on such a treasure mine of tutorials - NeHe has almost 50 highly detailed and well-commented lessons which allow easy understanding of, and progress through, the mechanics of OpenGL. Even more impressive, however, are the 30+ programming languages that each tutorial is available in, meaning that people using anything from Python to Code Warrior can learn in an environment that they're comfortable with.

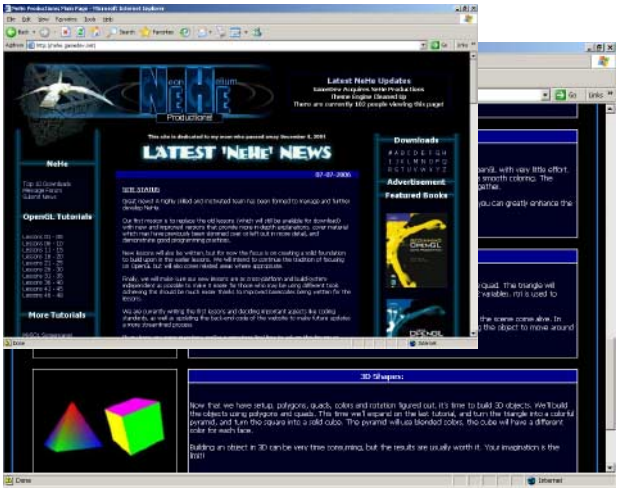
NeHe has been around for a few years already and still gets updates. An announcement in July revealed their plans to revamp the old lessons with easier, clearer ones, courtesy of a highly skilled team that's recently been assembled to manage the site. Additional advanced lessons are also in the works, with a focus on making them as cross-platform as possible to stick with their cosmopolitan approach to teaching. Of course, the tutorials aren't the only good thing about this site.

A series of about 20 articles on topics such as matrices, skeletal animation and particle systems are all available for easy viewing. Between these and the regular lessons, the lives of many newbie 3D programmers are bound to be made easier. Be sure to pop in at this website if you ever decide to discover the joys of OpenGL usage - you won't regret it.

NANDREW

Last known update: 07-07-2006

http://nehe.gamedev.net



Sulaco

Already mastered the basics of OpenGL? Fancy a bit of advanced development? Need a bit of local flavour? Head on over to Sulaco and see what sort of goodies are up for grabs!

Sulaco is a South African OpenGL site that's been running since 2001, containing a great deal of intermediate and advanced tutorials on a wide range of topics. The site is run by two individuals, but tutorials are submitted from a much broader community, meaning more content for the user! Learn how to use skeletal animation, project shadows, load up Quake 3 BSP files and even have a peek at the source code for Sulaco's Rage 3D engine.

All projects are created with Delphi and OpenGL, but programmers of other languages shouldn't have too much trouble adapting the code if they're already familiar with OpenGL itself. For beginners, there is a set of more basic tutorials available, although these are again restricted to Delphi and those who use other tools may have trouble learning. However, these lessons closely follow those of the famous OpenGL Programming Guide (more commonly known as the "Redbook"), making them relatively handy for beginners who do happen to work in Delphi/Pascal.

As mentioned already, this site is very much community-based. Advertised is Quake To Delphi, a joint effort with sourceforge.net to convert the original Quake source into Delphi code. Links to indie websites, OpenGL resources and forums also feature, making this a comprehensive place for most newbies to start.

At its heart, however, this site was built for power players, and power is what you'll get if you look around here. Some of the stuff featured is truly impressive, and it would be a shame if any OpenGL programmer was to skip over this handy local site.

NANDREW

Last known update: 07-07-2006

http://www.sulaco.co.za/

QUALITY TOUCH PART 2:
THE USER EXPERIENCE

DESIGN

The points raised in the first article of the Quality Touch can be considered as "Must Haves" for creating a complete game. In this article we'll look at "Should Haves" to allow user to control his own experience which will immerse the player further into the game. Games that are easy to play and give the player a number of options to make their experience better are more likely to be successful.

Controlling the user experience means allowing the user to manage the audio and visual outputs of the game, as well as the inputs which he uses to play the game. Not all players are created equal and neither are the environments in which they play, nor are the specs of the PCs on which your game will be played on. These selections are typically configurable within an options screen within the game.

Our first look is at audio which can be broken into direct sound (also know as sound effects) and ambient sound. Direct sound can be classified as sounds made by object interactions between the player or other elements of the game; they give audible confirmation of what is happening in the game. Ambient sound are sounds which give tone to the game and help to immerse the player in the world. Background music is ambient sound and ambient sounds play regardless of what the player is doing.

The player should be able to manage the volume of direct and ambient sound or turn either one of them off if they become a distraction or make the game unpleasant to play. Tastes vary between players (and often between game developers and players) and an unpleasant audio experience will find the game in the recycle bin. The opposite can also be true, if the player wants to immerse himself deeper into the world he might want to set ambient sound louder than the direct sound to hear what's going on.

The second look in this article is at the visual outputs of the game and since this is where generally most of the processing power of player's machine goes, the first thing the game should have is to allow the player to improve speed performance by switching off graphical features. This could be dynamic lighting, shading effects and

particle emission. In fact (and specifically with 3D games) at installation and configuration time the game should test for graphical support and turn off features which aren't supported on the player's graphics card. The number of ambient objects in the game can also affect how well the game performs (ambient objects are objects in the game which don't impact the game-play or objectives of the game) and the player should be able to control how many or how few he wants to have.

Another visual output to consider is the brightness or gamma control in the game. The real life environment around the player plays reflects light onto the screen and can darken what he sees. By allowing the player to adjust the brightness of the game, the player can counteract the lighting in his physical environment and see what's going on within the game. An easy option to give the player is the choice to play the game in a window or as a full screen application. Some players might prefer one over the other. Having the option in-game is also possible but not a necessity.

The last thing covered in this article is player input. Changing the key mappings, swapping devices and allowing the player to set device sensitivity ensure that the player can provide as much feedback into the game as possible. The game should allow player to re-assign keyboard controls.

Laptops especially have keys in different places which can make the game difficult to play. The player should also be able to manage mouse (or other devices) sensitivity to correctly relate physical movement into game space. The player should be able to swap an input device for another one to create a different feel for the game.

This could be a gamepad, joystick or game wheel and peddles.

While the last Quality Touch article listed a set of minimum requirements a game "must have" to be called complete, this months article lists a number of things that will start making a game seem polished and professional. While a game can certainly be called complete without these items, they will certainly improve the players experience and therefore encourage him to play again and to tell his friends about the great new game he discovered.

CAIRNSWM, FENGOL

SEPARATIONISM RANT

DESIGN

We have a problem and it's time we dealt with it.

For some reason, there's an insidious issue gnawing away at game development in South Africa. No, it's not Telkom, high software prices, terrible console distributors or even overseas publishers who aren't comfortable paying African companies … it's us.

We're the problem?

Yes, we are. It's something we do, instinctively as people brought up in SA, it's a part of our culture. We want independence.

It's true that in some cases being independent is a wonderfully good thing, it's just that when it comes to growing an industry, skill-base, investor-base, favourable public opinion and a market at the same time (as we game developers have to do here) that it's an issue.

So?

The reason it's a problem is simple: As soon as someone gets an idea for a cool game or a great mod, they go off and start their own forum/website/fanbase/circle and don't bother talking to anyone else who already knows what's going on. Those of us who have been watching all the attempts over the years have seen this pattern play itself out time and time again: some guy with a bright idea decides to set off on his own journey through game development, doing all the hard work along the way of setting up his own forum (with hundreds of sections, each devoted to a single aspect of the eventual game) and creating his own website to wow people into contributing art, time or money to his idea in the hopes of making it a reality.

Newsflash - that doesn't work. If it did, we'd have hundreds of games popping up out of the net-dreams of each geek with webspace. Heck, we wouldn't have to talk about things like growing a game development industry in SA because there would already be one, with tons of games!

Here's why it doesn't work: you're wasting effort maintaining and running a website, effort that isn't going into your game.

But wait, you're running your own site so that other people can help you, except that so few people are going to see your brand new site that you'll get so little help and thus won't get much that's useful. Finally, even though you think you're keeping the game "yours" and not giving it away to anyone, you're still relying on other people to help you make it.

Building a game is a lot of hard work. You don't want to be working on things that don't help you get your game finished, fun and playable. Yes, there's a place for websites devoted to completed games (especially if you're selling them), but having hundreds of armchair developers all screaming at people to "come to my forum, come to my forum!" isn't going to make any one of them successful…

But aren't independent developers a good thing?

The "independent" in indie, refers to independence from publisher money to make games. Just like indie films don't have big-time Hollywood producers funding them, or indie rock bands don't have deals with record labels that pay their bills.

SEPARATIONISM RANT

Being indie simple means that you don't owe your game to a publisher because they paid you to make it. Technically, all the game developers in SA that have made a game are indies, except maybe the big places like I-Imagine and Devon Systems (depending on if the guys in PE have a publisher or not).

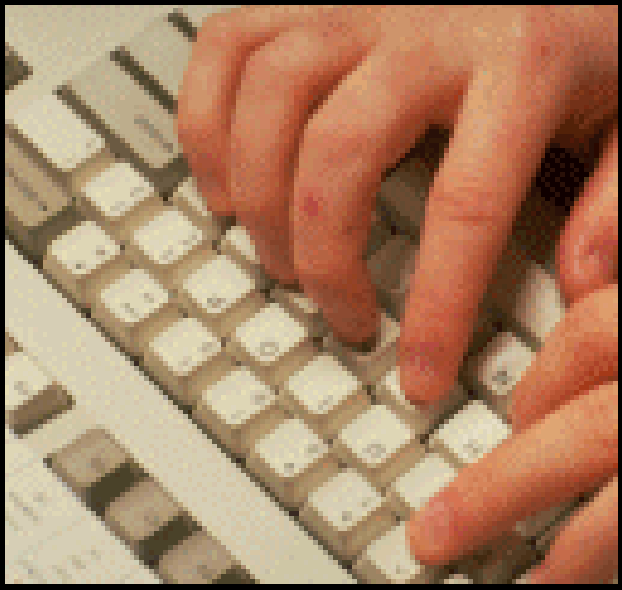
Being indie is not about being alone

There are communities out there that can help you. Places filled with people who have exactly the same want to run a game company and to make the coolest games ever, except they're not under the delusion that they're going to be "the first in SA" to do so.

We need to destroy the desire to be isolated so that we can remain in charge of our games and our ideas, 99.9% of isolated developers fail! Nobody is going to steal your idea if you talk about it in something like Game.Dev, instead they'll help you make it better. Nobody wants to "take over" your game, but if you go where the developers, artists and designers are instead of insisting that they come to you, they'll offer their assistance where they can…

In the end, we're a small portion of a small group of society in this country: gamers that want to make games. There's no reason that we should all insist on being kings of our own little patches of mud with pretty concept art. There's every reason to band together and share our experiences, knowledge and skills while we all make our dream games. The only way forward for our industry is to kill the separationist attitude once and for all - if isolation worked, you'd all have jobs making games.

DISLEKCIA



" We need to destroy the desire to be isolated so that we can remain in charge of our games and our ideas, 99.9% of isolated developers fail! "

Project Management: Part 2

This month in Project Management, we are going to be looking into the most important stage in the development of your game. Before I begin, I feel it is important to mention something that separates PM in game development from regular PM. In game design, it is often difficult to determine exactly how your game will play. It's easy enough to write out its gameplay dynamics, but one never can tell if these will truly work the way one intends them to.

I give you the art of iterative development. What this means is that you will be developing your game in waves. This process involves laying down the groundwork first, then developing that groundwork practically (actually creating it). Once you are happy that the foundations of your game are in place you can continue.

By following this system, you are ensuring that your game will, most importantly, have solid gameplay. It also means that future coding and bug fixing will be simplified, since you can easily detect at what level in your development the problems are occurring (if you encounter a bug in the third iteration, you know that the problem will most likely be in the second or third iteration).
There are many advantages to iterative development, but ultimately it is up to you how you decide to create your game. You could even use a combination of iterative and incremental (building one full piece at a time) techniques.

With that said, let us move on to the third part of project management.

Step 3 - Design

The design stage will determine exactly what happens in your game. It is during this stage that you will design your levels, enemies, NPCs, dialogue, items, quests, and anything else in your game. Bear in mind that this stage is still not practical and you should be doing all of this on paper.

Having said that, I feel it is important to address the issue of prototyping versus traditional development. In prototyping, the developer will rapidly create small sections of the game and will either continue to build on those sections, or will throw them away and begin again. Prototyping can be suitable for certain parts of development, especially aspects such as interface design, gameplay balancing, and timing-critical situations. Traditional development is more suited towards building the core of your game, as well as developing supporting mechanics.

Whether you choose prototyping or traditional development, it is important that you follow the stages presented to you in these articles. Prototyping may seem like an easier and more entertaining way to develop, but bear in mind that each phase of prototyping should be made up of its own 5 stages.

Ultimately, it is best to choose a method best suited to both your game's genre, and your own personality. You could, for example, design incrementally using prototyping if you are creating a simple, linear, action game, or you could design iteratively using traditional development if you are creating a complex, non-linear RPG.

The Hook

When designing your game, you should create certain goals that you would like your game to achieve. Examples of such goals could be to have the player completely enthralled by the sheer scope of the game (an example would be the Elder Scrolls series), or to have the player spend hours making decisions (the Civilization series), or to have the player scared witless after playing the game (the Doom series); it could even be something as simple as the rich, tangible feeling of games like Loco Roco.

If you keep these goals in mind during the design of your game, it will help you to create a particular flavour for your game, something that will remain with the player long after they turn off their machines and will keep them coming back for more. It is, however, important not to force the player into any such situations. Since each gamer is unique in their approach to any game, it is unwise to limit their options when playing just so they can get to "that scary part" of your game.

Linearity versus non-linearity

Certain games require linearity while others are nothing without the scope of choice they give their players. Linearity is a valuable tool, as it simplifies the development cycle, while creating an interactive "movie" for the players to indulge themselves in. The problem with linearity comes into play where the game's core mechanics are not interesting or dynamic enough to keep the player involved in the game. This can be caused by insufficient change or choice in the game, which leads the player to keep on a straight, often boring, line and will suffocate their game play experience. Choice can still be present in a linear game, but a fine balance must be achieved to give the player an impression of scope, while at the same time leading them to a predetermined goal. An excellent example of linearity that creates the impression of non-linearity is the Grand Theft Auto series.

When creating a non-linear game, it is vital to have the groundwork perfected before you begin designing the possible paths a player can take. It is also important to give the player sufficient goals or rewards depending on which path they choose. Many management or simulation games follow a non-linear design.

When looking at a game like The Sims, one can see that the fundamental mechanics driving the game, such as the family or gene systems, have been designed, created and tweaked until perfection.

Above all, when designing your game, irrespective of the approach that you choose, it is important to remember one thing: your game must be fun to play. It's senseless designing hundreds of resource gathering options if the only reason to gather resources is statistics. People play games for pleasure, sport, pure time-wasting or a combination thereof. Whatever your target market is, you must make your game enjoyable for it to be successful.

Next month, we will examine stages 4 and 5, which will finally see you creating content for your game. Enjoy designing your game!

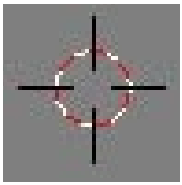
GEOMETRIX

Blender Tutorial - Making a simple scene

In this second installment of the Blender Tutorial, we will learn how to create new objects, learn about different kinds of lights and make a basic scene.

For this tutorial, we won't need that old cube, so let's start off by removing it. Select it with your right mouse button and, making sure you're in Object mode, click the object menu at the bottom of the 3D view and click delete (or press 'X') to remove it from the scene.

Now, let's have a look at the 3D cursor. The little cursor is shaped like a targeting crosshair and it dictates where a new object will appear in your scene. Clicking the left mouse button allows you to move it around. It's often easier to work using the grid, so let's snap the cursor to the gridlines to make our lives easier. Press Shift + S to bring up the Snap To menu, and select Cursor to Grid. This option is also available in the Object menu under Snap.This menu can also be used to snap objects and even vertices to the grid, so it can come in handy. Use it if your objects become misaligned.



The 3D cursor is where all new objects will be added.

The editing panel.

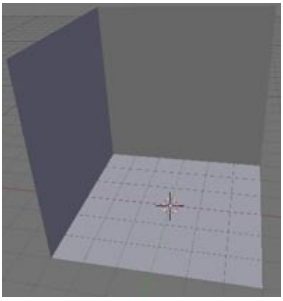
Adding things

Press the Space Bar to bring up the add menu. From here you can add items to the world and have access to various editing functions. Objects are also aligned based on your current viewpoint, so be careful of which angle your looking at your scene when you add items. Making sure you're in object mode and Top view, click Add, Mesh, Plane. This will add a flat plane to the world. Press S and move the mouse outwards to scale the plane. Watch the numbers in the bottom-left edge of the screen to see the degree of the transformation. You can type in numbers using your keyboard if you want to be precise, or hold in CTRL to constrain to fixed units. Scale the plane to 3 times size. Now that we have a floor, let's add a rear and side wall. Switch to front view (Numpad 1) and, in Object mode, add another plane. Scale this one to 3 times size as well. Move this new plane to the rear edge of the existing floor plane and align it neatly. You may need to move the view around using the middle mouse button, and switch between front, side and top views to make sure it is placed correctly. Remember to hold CTRL to constrain the movement to grid units. This will make moving things a lot easier.

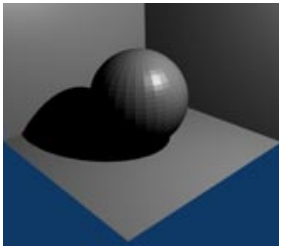
Make sure you're in object mode when adding meshes through the add menu.

Since our side wall will be the same proportions as the rear wall, we can simply duplicate the rear wall rather than creating it again. Select the rear wall in object mode, and click Object, Duplicate (Or press Shift+D) to create a double. Align this new wall to the edge of the floor and the existing wall.

You'll need to rotate the object with R to do this. Once you're done, your scene should look similar to this.



Let's add something slightly more interesting to the scene now. In top view and object mode, add a UVSphere mesh from the Spacebar menu. Accept the default values of 32 for segments and rings. Switch to the front view, and move the sphere so that it rests neatly on the floor. Also check in all views to make sure the sphere is near the middle of the floor. You may need to move the camera object so that the scene is clearly visible in the render. Remember that CTRL + ALT + NumPad 0 will move the camera object to match the view of the 3D window. Rendering the scene now should result in an image similar to this.



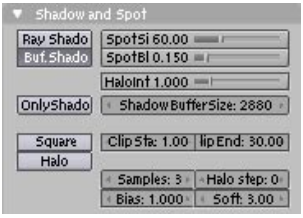
You'll notice that the sphere in the render doesn't really look round due to all the individual faces being distinctly visible. To remedy this we'll need to use the buttons window at the bottom of the screen. With the sphere selected

The shading panel.

in Object mode, click the Editing button or press F9 to change to the Editing Panel. Click the 'Set Smooth' button in the Links and Materials tab to smooth out the ball and make it look round.

Lighting

The lighting in our scene leaves something to be desired. To make things look a bit nicer, we'll change our simple lamp into a spotlight. Select the lamp, and change the shading panel by clicking the shading button or pressing F5. In the preview

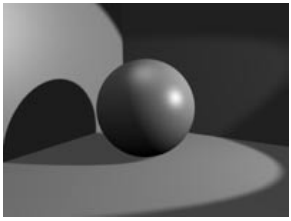


The shadow and spotlight tab. Change various lighting and shadow related settings here.

tab, click spot to change the lamp's type to spotlight. You'll notice the lamp now shows a cone shape when selected. This is the direction in which the light beam is focused. Check the top, side and front view to make sure the sphere lies in this cone. If it does not, rotate or move the spotlight until it does. You can also change the size of the spotlight beam by adjusting the SpotSi value in the Shadow and Spot tab. I set it to 45 degrees for a more focused beam. Experiment with different values and see which outcome you prefer.

Also, make sure the light is casting shadows using a shadow buffer instead of raytracing. This creates faster, softer shadows ideal for our low polygon model. Click the Buf Shadow button in the Shadow and Spot tab to change to buffered shadows.Now your scene will have a more focused light source, but some areas are totally black. Let's create another light to create a little bit of ambient brightness. In top view, add a Hemi light to the scene. This kind of light will create a 180 degree directional light source. It will not cast shadows, however, so it is ideal for adding a little bit of extra lighting. Drag it above you scene so that you can have some light shining from above. In the Shading panel again, look in the lamp tab for an energy value and change it to about 0.4. This will make hemisphere light dimmer than the spotlight.

After I had done a little bit of tweaking with the light angle and placement, repositioned the camera, and resized the walls and floor, my final scene looked like the following. You can save this file through the main Blender menu by clicking File, Save. We'll continue



from this scene in the next tutorial when we'll use materials give our items some texture. The file will also be available for download from the Dev.Mag website's content section.

CH1PPIT

MOBILE GAME DEVELOPMENT IN JAVA

PART 1: IN CONTROL

I'd like to start this instalment of the tutorial off with some clarification on Java versions and which are applicable to us, since some readers have apparently had a problem with this.

Because the latest stable release of the WTK is version 2.2 (2.5 is only available in Beta), we are using that. As this version of the WTK only supports the Java 1.4 JDK, that is what we use.Those of you that do other Java development and are concerned about downgrading your Java version, you need not worry because the two can happily coexist on a single machine. If you have any questions regarding this or any other issue related to these tutorials, please don't hesitate to contact me through the NAG forum. The link pages to the versions you require may be found at:

JDK 1.4: http://java.sun.com/j2se/1.4.2/download.html

WTK 2.2: http://java.sun.com/javame/downloads/index.jsp

And without further ado, let's add a player controlled sprite and some control elements to our game. First, use your art package to create a block image 50 pixels wide and 10 high (feel free to make it as fancy as you like) and save it in the same place as your ball image as bar.png. We use this image by creating another Sprite object in our Canvas class called barSprite, initializing it and placing it at the bottom of the screen in the same way as ballSprite.As before, we render this sprite in our doPaint() method. In the run method, we determine what keys are held down using the built in getKeyStates method.

This method must only be called once per game loop, so we cache the results in the local variable keyStates. We can check this result against convenient defines like LEFT_KEY to check what keys are active and move the bar with Sprite.move() if it is not against the edge of the screen. All of these changes can be seen in Listing 1.

Keep the little bugger on the screen or you'll lose a life!

Now that we have player input, we'll get the bar and ball interacting and give the player a couple of lives to look after. First of all, we add another variable to keep track of the lives (suitably named lives) and initialize it to 3 for the sake of tradition. We place our sprite movements inside an if construct that checks for remaining lives so that when the player loses all of their lives, we don't still have a ball bouncing around on screen.The ball movement is updated so that instead of bouncing off the bottom of the screen, the player loses a life and the ball is reset if the ball falls through the bottom of the screen. We also check for collision between the ball and bar using Sprite's very useful collidesWith method.

Listing 1. TutorialCanvas after the player controlled bar has been added.

```
class TutorialCanvas extends GameCanvas implements Runnable
{
        ...
        static Sprite barSprite;

        //Constructor
        public TutorialCanvas()
        {
                ...
                try
                {
                    ballSprite = new Sprite(Image.createImage("/ball.png"));
                    barSprite = new Sprite(Image.createImage("/bar.png"));
                } catch(Exception e)
                {
                        e.printStackTrace();
                }
                ballSprite.setPosition(1, 1);
                barSprite.setPosition((getWidth()+barSprite.getWidth())/2,
                                        getHeight()-barSprite.getHeight());
        }

        //run the game loop
        public void run()
        {
           while(!exit)
           {
                //update our bar position
                int keyStates = getKeyStates();
                if ((keyStates & LEFT_PRESSED) != 0 &&
                     barSprite.getX() > 0)
                {
                     barSprite.move(-1, 0);
                }
                else if ((keyStates & RIGHT_PRESSED) != 0 &&
                     barSprite.getX() + barSprite.getWidth() < getWidth())
                {
                     barSprite.move(1, 0);
                }
                ...
           }
        }

        //paint the canvas
        protected void doPaint(Graphics g)
        {
                ...
                barSprite.paint(g);
        }
}
```

In this case we specify false to tell the method to not check pixel level collision. Finally, in our doPaint method, we add text displaying the number of lives remaining, as well as text telling the player the game is over if they are out of lives. For all intents and purposes, we now have a game! It may be a very simple game, and probably won't hold anyone's attention for very long, but it is a game nonetheless. At this point you may already suspect exactly what kind of game we're gearing up to create, next lesson we'll be rounding off the basic gameplay and learning a bit about Java Vectors in the process.

FLINT

Listing 2. TutorialCanvas after player lives and sprite interaction has been added.

```
class TutorialCanvas extends GameCanvas implements Runnable
{
        ...
        static int lives = 3;

        //Constructor
        public TutorialCanvas()
        {
                ...
        }

        //run the game loop
        public void run()
        {
                while(!exit)
                {
                        if(lives > 0)
                        {
                            ...
                            int ballY = ballSprite.getY();
                            if(ballY < 0)
                            {
                              ballVY = -ballVY;
                            }
                            else if(ballY + ballSprite.getHeight() >
                                        getHeight())
                            {
                                  lives--;
                                  ballX = ballY = 1;
                            }
                            else if(ballSprite.collidesWith(barSprite, false))
                            {
                                  ballVY = -ballVY;
                            }
                            ...
                        }

                        ...
                }
        }

        //paint the canvas
        protected void doPaint(Graphics g)
        {
                ...
                g.setColor(0x0000ff);
                g.drawString("LIVES: "+lives, 10, 10,
                                Graphics.TOP|Graphics.LEFT);
                if(lives == 0)
                {
                    g.drawString("GAME OVER!", getWidth()/2, getHeight()/2,
                                Graphics.HCENTER|Graphics.TOP);
                }
        }
}
```

THE TECH WIZARD

Data Structures - Part 1

Data structures are the foundation for computer programming, and are especially important for any kind of meaningful game development. Their use is exactly what the name implies: structures or systems used to store and / or manage data.

They range from simple methods of merely storing data, to complex systems that can be used for such things as accessing specific data quickly or easy management of dynamically changing data.

TECH

Variables

The base method for storing information in computer memory is the variable, which is a representation of a single piece of data. While different programming languages have their own implementation of variables, the standard variable types that exist in most of them are bytes , integer numbers (ints) and floating-point numbers (floats).

As their name implies, bytes are used to store information that is up to 1 byte in size. Ints generally store information that is up to 4 bytes in size, however some languages and platforms support 2, 8, and even 16 byte integers. Floats are also generally used to store up to 4 bytes of data, but like ints, also have language and platform specific implementation that can provide storage for either 8 or 16 bytes of data.

C-Based pseudo-code of some simple variables:

```
char  character;   // stores a single alphanumeric
                   // ASCII  character

int   objectID1;  // stores a single object ID
int   objectID2;  // stores a single object ID
int   objectID3;  // stores a single object ID

float x;          // the x element of a vector
float y;          // the y element of a vector
```

Arrays

However, most games are complex enough that they need methods of storing many pieces of similar or related data in a way that is easy to manage.

Most languages accomplish this by simply evolving a variable into what is called an array, which can be best described as a list of variables of the same type, that is of a fixed length, and is stored linearly in memory.

(In many languages, bytes are referred to as characters or chars)

They are perfect for storing large amounts of similar data when you know the maximum number of items that you will have.

However, they should only really be used in cases where you will not need to discard items from inside of list, as it is usually quite inefficient to keep track of any empty list positions other than the current end element of the array.

Arrays are also great when it comes to storing data that needs to be iterated through, as you only need to increase your current index into the array to obtain the next value in the list.

Not only is iterating through an array quite easy, but it is also efficient as you are working in the same area of computer memory due to the linear nature of an array.

C-Based pseudo-code of some simple array usage:

```
char    filename[64];       // filename string of up to 63 ASCII
                            // characters
                            // (+1 for the terminator value)

int     objectID[100];      // 100 entries to store used
                            // object Ids

float vector[2];            // 2 floats used for the x, y
                            // elements of a vector

InitialiseFunction()
{
        int    i;



        //store the name "pic.bmp" into the filename array
        filename[0] = 'p';
        filename[1] = 'i';
        filename[2] = 'c';
        filename[3] = '.';
        filename[4] = 'b';
        filename[5] = 'm';
        filename[6] = 'p';
        filename[7] = NULL;

        //initialize array of object Ids
        i = 0;
        loop(100)
        {
                objectID[i] = i;
                i = i + 1;
        }

        //initialize the vector coordinates
        vector[0] = 10.0;
        vector[1] = -10.0;
}
```

(Each item of an array is generally referred to as an element. It is often quite inefficient in terms of memory cache usage to jump between different areas of computer memory)

Structures

While arrays are great for storing large groups of similar data together, there are many times when different types of data, requiring different types of variables, needs to be grouped together. Generally these types of data structures are called, quite simply, structures .Structures are probably the most commonly used type of data structure, as well as the most helpful. The idea behind a structure is that it can be used it to create custom variables that can be made up of both basic variable types (ie. bytes, ints, floats, etc.) as well as other user created structures. Once created, a structure can be used just like any other variable .

The versatility of structures makes them invaluable to game development in particular. Almost all areas of game programming require that various types of related data be grouped together, even if the types of data are different. Usually this is down to management purposes, and is technically not necessary as a game could be created by only using single variables. However, doing so would make working with those variables very difficult, not to mention incredibly messy. In practice, once structures are used properly, there is no substitute to them for easily managed data storage.

C-Based pseudo-code for defining a simple game object with structures

```
//2 Dimensional Vector Structure
structure
{
        float   x;      // X position of the vector
        float   y;      // Y position of the vector
}Vector2D;


//Rectangle Structure
structure
{
        int     top;        // top coordinate of the rectangle
        int     left        // left coordinate of the rectangle
        int     bottom;     // bottom coordinate of the rectangle
        int     right;      // right coordinate of the rectangle
}Rect;


//Graphic Image Structure

structure
{
        char        filename[64];   // name of the file that is
                                    // loaded for the image
        int         width;          // width of the image
                                    // (in pixels)
        int         height;         // height of the image
                                    // (in pixels)
        byte        *imageData;     // pointer to memory that has
                                    // been allocated to store the
                                    // pixel data for the image
}Image;


//2D Sprite Structure
structure
{
        Int         flags;          // bit flags for the sprite
        Rect        source;         // rectangle defining the area
                                    // within the parent image that
                                    // represents this sprite
Image           *image;         // pointer to the image that
                                // the sprite is referencing
}Sprite;


//Simple Game Object Structure
structure
{
        int         objectID;   // id of the object
        int         flags;      // bit flags for the object
        Vector2D    position;   // position of the object
                                // in the world
        float       rotation;   // rotation of the object
        Sprite      sprite      // sprite representing
                                // the object
}GameObject;
```

(The name does vary between programming languages, although the implementation remains relatively the same For instance, an array of any structure type can be created for use if required)

The proper use of data structures is important in not only creating efficient code, but easily managed and maintained code as well.

As is the case with many areas of computer programming, understanding what the various types of data structures are and how they can be used takes precedence over knowing how to implement them properly, as only once you understand how they work will you best be able to judge which ones are best suited for a given task.

These are but a few of the types of data structures available for computer programmers to use. In Part 2, we will move on to more complex systems that will allow for even easier code management and efficiency.

COOLHAND

It's rAge, baby!

Right. So you know about rAge? Of course you do. It's the single most important gaming event in the world, bringing inconceivable joy to billions upon billions of people and is, in fact, considered a key element to finally obtaining world peace. Or something like that.

Marketing hype aside, rAge really is the biggest thing to hit SA in terms of gaming, so it comes as no surprise that the art of game development rears its head there too. That's right. In a way that previous incarnations have never seen, this year's rAge is going to go all-out on its game development section, complete with stands, keynote speakers, free stuff, prizegiving events and, of course, your friendly local Dev.Mag. Want to learn more? Well, here's five things to remember when you go devving at rAge this year:

Dev.Mag!

The event of the year just wouldn't feel complete without the presence of the writers, designers and marketers responsible for your favourite monthly e-zine. Expect members of the mag to fly, drive, walk and crawl their way to rAge, just for their loyal readers!

That, and the free t-shirts they get so that you can identify the team on the exhibition floor! Most of us will be around for the whole rAge event, doing promo work, distributing copies of the mag, managing stalls or helping out with some sort of menial task or another. Send them a "Howzit!" if you run into any of them!

Free Stuff!

Right, so free copies of the mag aren't the only thing. We've archived the entire set of magazines onto promotional CDs filled with game development resources, projects, tutorials, links and other handy tools for any wannabe game maker to get his or her teeth into. Look out for flashy posters and people wearing the Game.Dev / Dev.Mag t-shirts to make sure you get in on the action! For those of you entering NAG's comp 10, we'll be handing out some sweet amounts of cash at the rAge prizegiving to those entrants who managed to get themselves in one of the top positions.

Education!

Throughout the expo, the gaming development crew will be running hour-long sessions consisting of either talks or workshops. Speeches consist of industry professionals (some of them from amongst the ranks of the Dev.Mag writers) delivering talks on a wide range of game development topics, such as frameworks, development communities, prototyping, evolution of game making and more. When there isn't a speaker at work, audiences will be sure to find an engaging workshop to get involved with.

Game Dev Idols!

Nope, this is not a national sing-off (or code-off, or design-off, or whatever takes your fancy). We're talking about industry professionals and big names coming to tell us what-for and how they see the game industry today. Pick up some courage, ask them questions, show them your ideas. It's bound to be rewarding, as long as you don't do something creepy like tattoo their names onto your body. They will, of course, be involved with the talks and workshops.

Your Place in the Limelight!

Yep, you read correctly - come show us what you've done and we'll give you a projector time to show everyone. No catch. These local dev spotlights will be running in between the workshop sessions and are a great way to not only advertise your projects, but get real face-to-face feedback and attract a few "oohs" and "aahs" at the same time. So start preparing something now to bring to the table when rAge arrives.

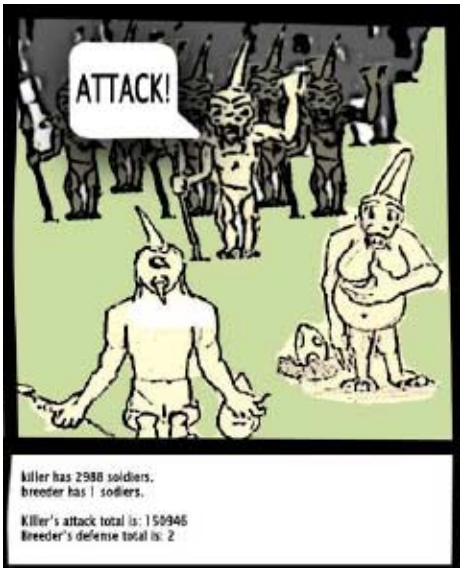
If you weren't going already, make sure you haul yourself to this mega-event at the end of September! We'll be waiting for you!

NANDREW

Keep an eye out for general rAge info on the following sites:

www.nag.co.za
www.gamedotdev.co.za
www.arena77.co.za

COMIC



"Abstract, isn't it? This comic has a deep, hidden meaning which can only be truly discovered if you've played Prehysteria (www.prehysteria.org). That's right -- if you don't play Prehysteria, your life will remain forever meaningless. We are indeed truly shameless."

www.prehysteria.org

game.dev
It's what you wish you were doing.

MAKE A GAME AND WIN

In a first for South African game development, NAG magazine is sponsoring R10 000 in prizes for a local game development competition. Run by Game.Dev, a local developer support community, the competition asks entrants to design and build a management game over the course of two months. This is the tenth competition Game.Dev has held in two years, but never before have such large amounts of money been up for grabs.

For more information on the competition and instructions on how to enter, take a look at the sites below:

NAG
(http://www.nag.co.za)

game.dev
It's what you wish you were doing.
(http://www.gamedotdev.co.za)

ONLINE

devmag.googlepages.com