



SOUTH AFRICA'S PREMIER GAME DEVELOPMENT MAGAZINE

August 2007

DEV.MAG

CREATE • DEVELOP • EXPERIENCE



REGULARS

Ed's Note.....	P03
News	P04

SPOTLIGHT

Nicklas Nygren, creator of Knytt.....	P05
---------------------------------------	-----

OPINION

A Really Juicy Story.....	P08
---------------------------	-----

REVIEW

Newgrounds Flash Portal.....	P09
------------------------------	-----

DESIGN

Blender Intermediate Tutorial: Converting 2D to 3D.....	P10
A Beginner's Guide to Making Games: Part 3.....	P12
A Beginner's Guide to Making Games: Part 2 EXTRA.....	P14

PROJECTS

Roach Toaster 2: Big City.....	P15
--------------------------------	-----

TECH

Coding Etiquette: Doxygen.....	P16
Game Coding with Trigonometry: Part 3.....	P19
Go optimise!.....	P22

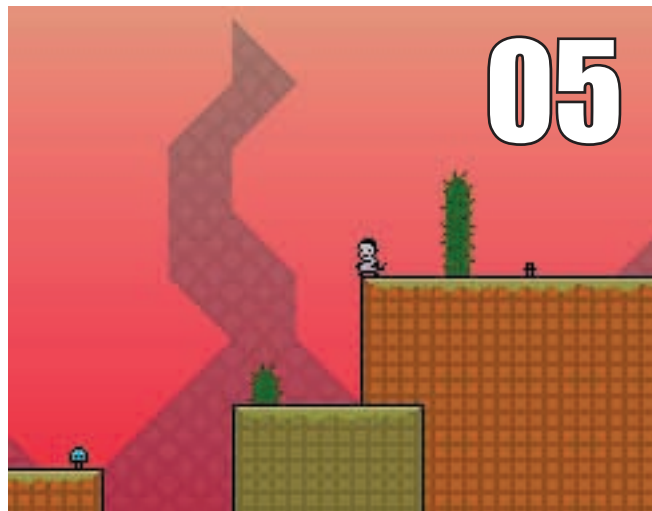
HISTORY

The History of I-Imagine Part 6: Enter the Publisher.....	P25
-----------------------------------------------------------	-----

TAIL PIECE

In Casual We Trust.....	P29
-------------------------	-----

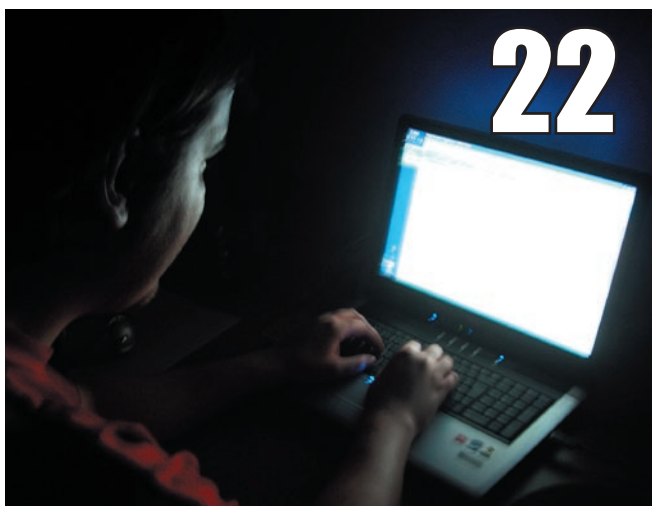
05



10



22



25



Wowee, the situation is getting intense. I will confess: things have been slow this month. With Issue 16, that is. But at the same time, stuff is going really, really fast and it's getting pretty tricky to keep up with everything. On the community front, the winners of Comp 15 have been announced and a nifty R 5000 goes to Evil_Toaster for his game Cartesian Chaos (lucky bugger). We've also got some pretty cool games from the runners-up and a full report will be present in the next Dev.Mag. In the meantime, we've also got Comp 16 starting up (text-based graphics being the theme) and people are scrabbling to get some entries in before the 28th of this month.

Of course, I still haven't explained why Issue 16 was so long in coming out: as it so happens, we've got rAge coming up at the end of this month. In honour of this annual SA technology expo (a biggie on our calendar), Dev.Mag Issue 17 will come out complete with a full visual makeover, a massive amount of content, and some other exciting odds and ends which are taking up a lot of our attention. As of now, while I'm sitting here furiously tapping out my little editor's spiel, there's another little window open on my computer showing the first draft of the new-look mag, accompanied by a tentative first article. Hopefully, the problems encountered with this edition will be forgiven once our bumper issue sees the light of day.

Until then, you have this offering. I don't quite know how the team manages it, but every month has at least one person toddling up to me and presenting something awesome for either a spotlight or a feature. This month, we've snagged a talk with Niffilas, the creator of the popular (and absolutely adorable) freeware game, Knytt. If you don't know what it is, turn to page 5 and follow the link. If you do know what it is, turn to page 5 and have an awesome time reading.

That's it for now – if I blab any longer, this mag's going to be delayed even further, and avid readers are going to be bringing the torches and pitchforks to my proverbial door to make sure I shut up. Stay tuned for Dev.Mag 17. It's going to be a blast, and you won't want to miss that.

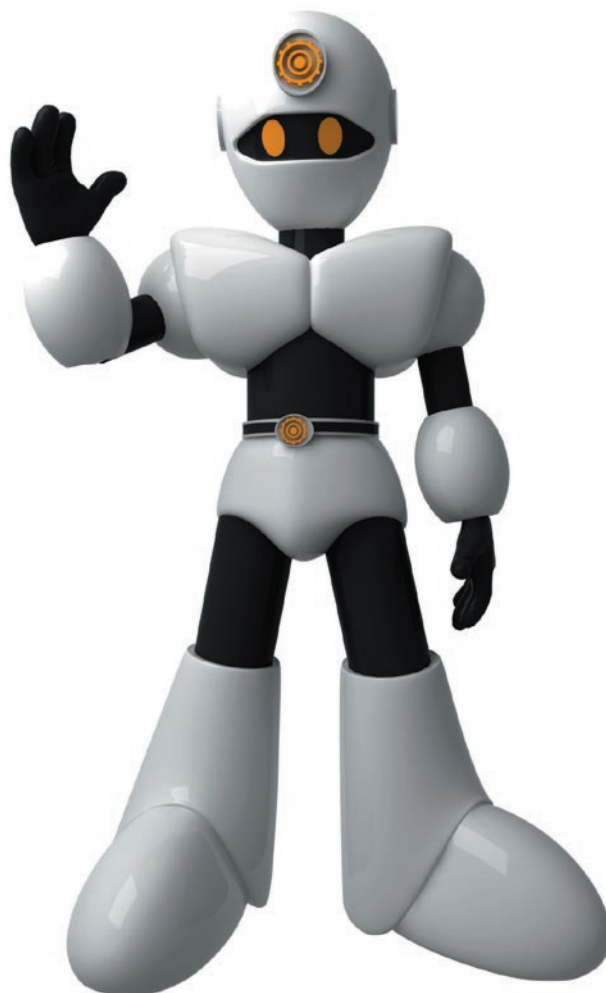
Editor

Rodain "Nandrew" Joubert

DID YOU KNOW?

Think that having a job as a tester is easy? Think again. A recent chat about the testing process of Crackdown, a free-roam urban game consisting of 495 city blocks, revealed some startling figures. One of these was the amount of environment bugs that were reported and squashed: a whopping 10000. Not a typo.

This month's opinion columnists:
Quinton "Q-man" Bronkhorst



DEV MAG
CREATE DEVELOP EXPERIENCE

EDITOR

Rodain "Nandrew" Joubert

DEPUTY EDITOR

Claudio "Chippit" de Sa

SUB EDITOR

Tarryn "Azimuth" van der Byl

DESIGNERS

Brandon "Cyberninja" Rajkumar
Geoff "GeometriX" Burrows

MARKETING

Bernard "Mushi Mushi" Boshoff
Andre "Fengol" Odendaal

WRITERS

Simon "Tr00jg" de la Rouviere
Ricky "Insomniac" Abell
William "Cairnswm" Cairns
Danny "Dislekcia" Day
Andre "Fengol" Odendaal
Heinrich "Himmmler" Rall
Matt "Flint" Benic
Luke "Coolhand" Lamothe
Stefan "rman" van der Vyver
Gareth "Gazza_N" Wilcock

WEBSITE DESIGNER

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.gamedotdev.co.za.

All images used in the mag are copyright and belong to their respective owners. If you try and claim otherwise, think again. In fact, think casual.

... no, wait, scratch that.
Think sentient strawberries.

Casual games study

http://www.gamasutra.com/php-bin/news_index.php?story=15145

Casual games seem to be the popular choice for Internet users, according to a recent study conducted by Parks Associates. Gamasutra reports that the company has found casual games to be even more popular than major players such as YouTube and Myspace. Not only is casual gaming the most popular pastime, but its growth in recent years has been impressive and this will probably make room for more avid developers. Time to clamber onto the bus.



XNA Gamefest videos released

<http://creators.xna.com/Headlines/presentations/default.aspx>

GameFest 2007 was an educational experience which also provided benefits for the armchair enthusiasts. Yes, even those who haven't attended the festival can still watch

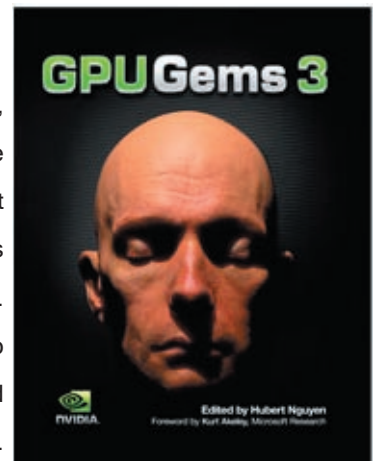


several comprehensive presentations on the versatile XNA game development tool, thanks to the XNA Creators Club website. There are 7 videos viewable online, most of them at least half an hour long and ranging from topics like "XNA Game Studio for Fun and Profit" to profiles of XNA works in progress. Worth a look if you have the time.

NVIDIA releases third GPU programming book

<http://developer.nvidia.com/object/gpu-gems-3.html>

NVIDIA has released its third GPU programming book, entitled "GPU Gems 3", which outlines some of the more advanced uses of the graphics processing unit in NVIDIA cards, even outlining uses such as physics simulations, financial analysis and even virus detection. Whoah! The book, which is roughly 1000 pages, also has the latest algorithms for a menagerie of advanced graphics effects. The price? \$69.99 from Amazon.



Torque MMO kit free under BSD-style licence

http://www.gamedev.net/community/forums/topic.asp?topic_id=459909

Want a good reason to buy the Torque Game Engine and ArcaneFX? Well, Gamedev.net reports that the crew from GarageGames have given fans precisely that with the offer to make the Torque MMO kit free for both commercial and non-commercial use to anybody who already owns TGE and ArcaneFX. According to the market blurb, The MMO kit represents four years of development and thousands of hours of labour, and the full Python and Torquescript source is now available.



TALKING TO NICKLAS NYGREN

Creator of Knytt

Niffas is an individual on the web who seems to enjoy creating the cutest and most intriguing little games. One of these is Knytt (<http://niffas.ni2.se/>), a freeware jump 'n climb platformer that's notable for being remarkably polished and eerily engaging. We decided to have a little talk with him about the whole game making deal.

Tell us about how you got into game development? What got you interested? When did you start?

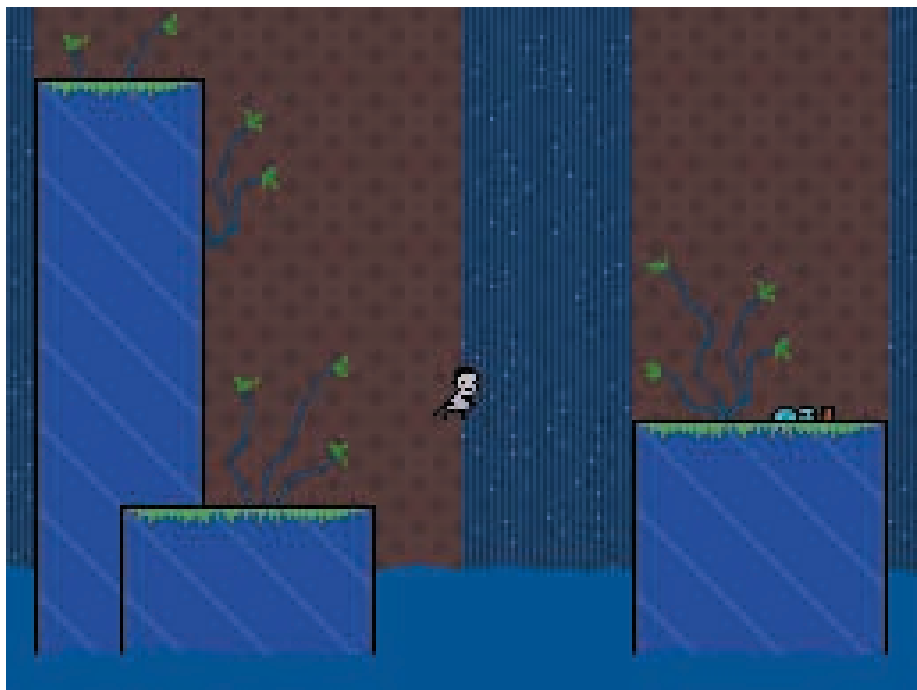
I started out around 10 years ago with Klik & Play. Since it was a long time ago, I can't quite remember what got me interested. I've simply always had an interest in creating things, no matter if it have been building things with LEGO, composing music, drawing things, or attempting to create computer games.

It took me quite a lot of years to figure out that I should keep my game projects at a reasonable size. If it took Square a large development team to create a game like Chrono Trigger, I should have known better than believing that it would be possible for me alone to create an RPG that you'd need to spend over 60 hours to beat.

A few years ago I realised this though, and gave game development another go, and created #modarchive story - and since then I've been releasing my games on a more or less regular basis. Since then my games have increased in size and more people have been helping me out, but I'm of course still never creating anything nearly as large as the stuff I planned in the past. But really, it's not the size of the game that express me anyway, it's the content.

What made you decide to create a game such as Knytt?

Well, I was working on Within a Deep Forest



2, but wasn't really inspired. I wanted WaDF2 to have some mini-games, so I planned Knytt to be a WaDF2 mini-game, but as soon as I started to create the first part of Knytt, I got a massive amount of inspiration to continue on that, while I spent less time on WaDF2. In the end, I abandoned WaDF2 because of the lack of inspiration, and released Knytt. I'm still really happy about that decision. I know a lot of people would have preferred me to finish WaDF2, but personally I care so much about creating things which really express myself - and Knytt really does that, perhaps more than any other of my games ever has.

Tell us more about your upcoming game, Knytt Stories.

I've been working on Knytt Stories since the day I released Knytt (December 2006 if I don't remember wrong). It's based on the Knytt platform engine, but features more challenge, power-ups, and a powerful level editor. This time I'm trying to encourage the players to create

their own levels with their own unique stories. It's with no doubt my largest creation, and it's release is just around the corner.

You created a niche on the net for your games. Do you think every developer can have this kind of niche?

I see no value in being original just for the sake of being original itself. I do however think that most games are too alike these days, as if the developers just see their games as entertainment created to meet the demands of the audience, rather than a form of artistic expression. Of course, there are plenty of exceptions.

The music in Within A Deep Forest and Knytt lends a great deal to the atmosphere of the game. Have you ever considered making music as a career?

I better mention first that I didn't create most of the music for Within a Deep Forest, I only created one of the songs on my own (the one in Utopiaca). I created around half of the music in

Knytt though. Earlier I was planning to making music as a career, but at the moment, things looks like game development is my best chance for a career...

You have a whole community of fans helping you on Knytt Stories. How do you find working with such a community??

It's great fun - I got an insane amount of help designing tilesets for the game, which is why Knytt Stories will come with a library of 256 tilesets that can be used in level design. Most games just come with the graphics that are used in the game itself (of course, Knytt Stories levels can feature their own custom graphics too).

Ico was a huge inspiration for Knytt. Do you think games like these are grossly underrated?

I can't say Ico is underrated, although Ico never became a huge commercial success (if I don't remember wrong, that is). The reviewers really loved that game, and gave it the feedback it deserved. But yeah, I'd love to see more of those kind of games.

Do you want to continue working on small games like these, or are you interested in getting into the major game industry?

I'm open for anything, as long as I don't have to sacrifice the personality of my games. In other words, I'd rather keep releasing small productions than joining a big company and help out designing games that I don't really like or have any control of. A small company with people who share my ideas would be the ideal thing.

What recent game that you have played made you sit up and say "Wow, this is genius"?

Shellblast by Vertigo Games is fantastic, although I never liked the "war on terror" phrase used in it (negative associations). It's portable too!

language? Do you think there is a place for games not made in English?

You're not likely to find Knytt in the average dictionary, but it's still a word sometimes used to describe small harmless creatures. Originally, the word was invented by "Tove Jansson" (the children's book author) known for the Moomin trolls. The particular book that uses the word is titled "Vem ska trösta knyttet?" (which is translated to "Who will Comfort Toffle?" in the English version). I can highly recommend this children's book.

About the other question: yeah, I think there's a place for games in any language, but as with all other things, you can get a wider audience if the game is available in a language that more people speak.

TROOJG

Quick Questions ... 3 ... 2 ... 1 ... Go!

Pizza or Pasta?

Pasta

PS3, Wii or Xbox-360?

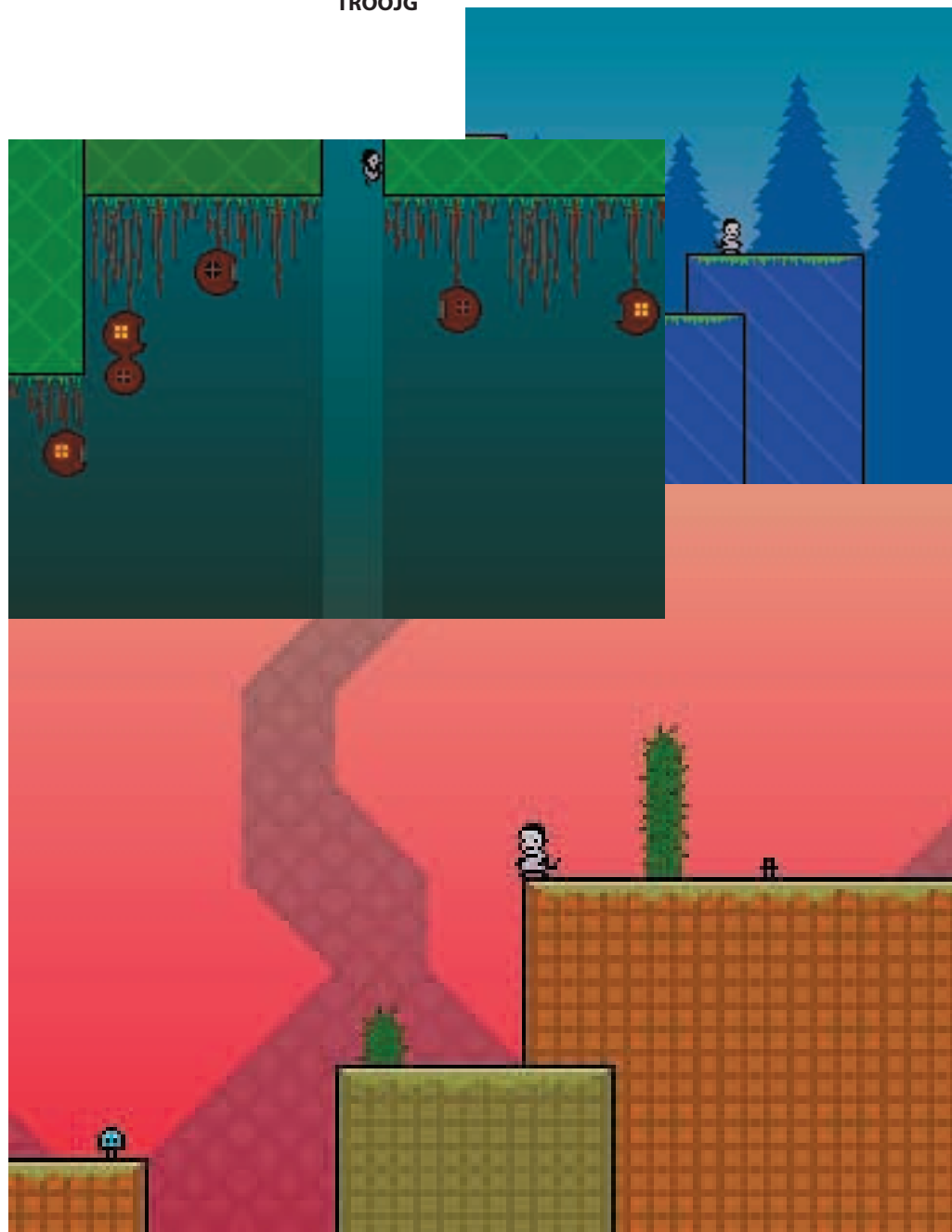
XBox have always had more good games, but Playstation have always had the most interesting and unique productions (although there are only a very few of them). I'm gonna have to go with PS3.

The Simpsons or Family Guy?

Simpsons

Guitar or Piano?

Piano



Windows Vista

Open Source Win64

GameBoy Advance

DirectX 10

.NET 2.0

XBox 360 via XNA

Linux

Mac OS X

Nintendo DS

OpenGL 2.1

SDL



Object Pascal has a lot to offer...

www.PascalGameDevelopment.com

Chrome

Free Pascal

Delphi for Win32

Turbo Delphi

Lazarus



“A really juicy story”

When I was approached to write a nice little opinion piece for Dev. Mag, I found myself sitting in front of my word processing application, staring rather blankly at the screen. Only having done low-level programming in high school (in the form of Turbo Pascal 7.0) I came to realize that I don't really know much about game development - and that which I do know, is nowhere near enough to fill a whole opinion article. After all, the most I've done in terms of game development, is a little text-based RPG of epic failure.

So while I started typing out space filler about how little I actually know about game development, and how what I did know would not be sufficient enough to fill a column - it hit me like a proverbial brick to the head; instead of wasting all this space, waffling on like a pretentious git, I should write about something I do know a lot about:

Boy oh boy, do I love to tell a good story..

It should be obvious to most people, that having good programming in a game, although extremely important, isn't always going to make your game the winner. Every game needs a story; from a plump plumber in red saving a pink princess from a dragon-thing with a spiked turtle shell on his back, to a journey into the horrors of one's own mind to face psychological demons brought out to attack you, by subconscious guilt. I would even hazard a guess that

even games such as Tetris were created with some form of world in mind.

So we're making a game right? Ok, so let's get started then. Before we can even think about programming anything, we need some sort of concept. Oranges. Oranges that are alive. Let's work with that. Where are these living oranges found? Let's just say they're living in the magical city of Citrus, where the buildings are all fruity and bright. We're going to now introduce our main character, Oscar, who is just an every day kind of orange, going about his everyday orangey stuff.

Oh dear, but that would be a boring concept (unless it were a fruit city simulator - covered in another article sometime perhaps) so we need to bring some sort of conflict in here. Strawberries! The evil strawberries have declared war on Citrus City and have besieged the town, taking all of the oranges captive - to be squeezed for their juices to make sugar-filled, impure cold-drink in special 'Concentration camps'. The only orange who can stop them is Oscar!

Now that we have our location, characters and conflict, we can start getting our heads around how Oscar will go about saving his people. I do believe this is the point where programmers' ears should prick up. As Oscar rolls through the city, he encounters evil strawberries which he must either 'power-charge' through, 'bomb-drop' on, or merely avoid, by bouncing over

them. There could be minor or major obstructions he must overcome, or even strawberry generals to destroy - there are no limits from this point on.

It may sound like a ridiculous story, but if you've read up until this point, it must have intrigued you in some way. Without a story for your game to work on, your game isn't going to make much sense to the player. Why am I shooting these people? Where are these chickens coming from? What is the point of collecting all this ham? These are the kinds of questions you need to answer.

If you can sell someone a compelling story of a brave young orange saving his people from an evil strawberry siege, and have it make sense in the context of your game; you just might have something solid to start working on.

Q-MAN

If you fancy saying something about game development and have enough faith in your writing ability to do so, feel free to submit content to our monthly opinions section.

Send your work, 400-500 words in length, with the subject title "Opinion Submission" to devmag@gmail.com, and you may just wind up in our pages. Please note that we reserve the right to decide what material is suitable to publish in the magazine.

The Dev.Mag staff does not expressly support or agree with the views of guest columnists in this section. In fact, the Dev.Mag staff does not expressly support or agree with the views of Dev.Mag staff writers in this section. It's a crazy world, isn't it?

Newgrounds

Website

<http://www.newgrounds.com/>

Updated

Continuously

Simple, quick and easy games are king in the world of casual gaming. In this vein, it is no surprise that flash games are immensely popular online, with numerous flash-game portals appearing all over the web. Nothing is quicker or easier than navigating your browser to your favourite portal and selecting a game.

Newgrounds is likely the largest flash portal around, boasting over a million registered users and 300 000 submissions, including flash videos and games. Since its foundation 12 years ago, Newgrounds has done for flash gaming what YouTube did for online video.

Part of its success can be attributed to a large and active community, spurred on by a ranking system that rewards users for participating in the site. Rating games scores

users "Grounds Gold" which places more weight behind future votes, as well as increasing the users rank, earning the user more respect in the community.

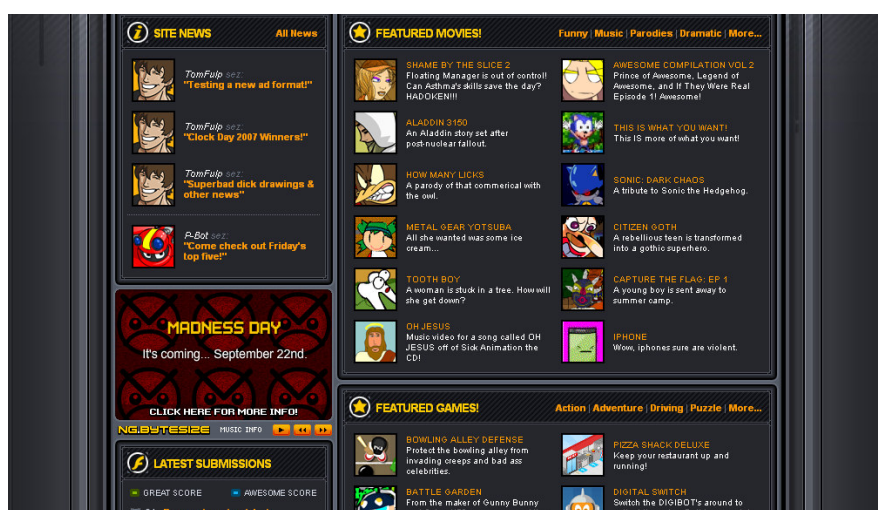
To keep the site fresh, submissions that have a low aggregate score are automatically removed from the website, or 'blammed'. To encourage fair voting, the website awards blame/protect points to users whose votes accurately depict the eventual fate of the game. These points also contribute towards a users rank.

Newgrounds has been home to many popular flash video and game series,

including Salad Fingers and Super Mario Brothers Z. It is also the original home of The Behemoth's surprise hit Alien Hominid, whose prototype was first displayed at Newgrounds. Alien Hominid went on to be a popular PS2 and Gamecube title, with a HD Xbox Live Arcade version recently released. The Behemoth is also currently working on another XBLA title, Castle Crashers, which also looks set to be a success.

For a quick fix of gaming fun, or for a little bit of a humour, Newgrounds is always a great place to look for a quality diversion.

CHIPPIT





BLENDER INTERMEDIATE TUTORIAL:

Converting 2D to 3D

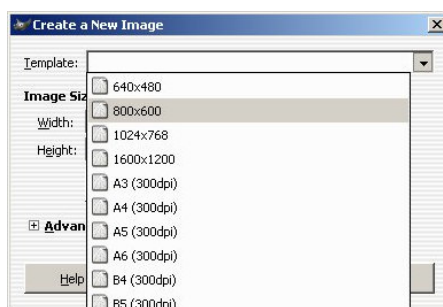
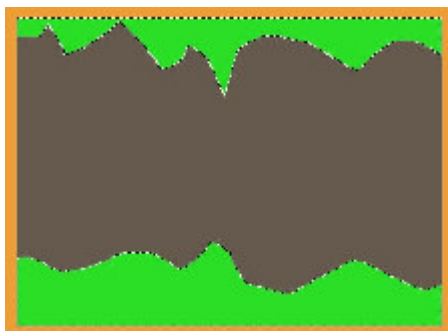
(If you're new to Blender, check out Dev.Mag Issue 5 for the beginning of our Blender tutorial series!)

Converting 2D drawings into 3D. Yep, that sounds pretty cool, right? In this article I will show you how to take a 2D design and transform it into 3D. Yes, there are limitations, but what we can do here might just come in very handy.

We will use the GIMP, considered the best Open Source alternative to Adobe Photoshop. In the GIMP we will draw something, select that drawing and use the GIMP to create an outline. The outline will be exported out of the GIMP, and imported into Blender. Blender will then add the 3D element. The GIMP can be downloaded from www.GIMP.org.

For this tutorial we will create a simple game interface screen using the following steps:

- Draw in GIMP
- Select the drawn bits
- Convert the selection to a path
- Export the path
- Import the path into Blender
- Add depth to the object



Let's get going. Fire up the GIMP. Assuming that my game screen size will be 800 X 600 pixels, I will create an image of that size in the GIMP. What actually happens is that you can scale the object to any size once you have it in Blender, but it is easier to work with the actual dimensions in the GIMP. The best suggestion I have for creating a design that will work well for conversion into 3D is:

- To do the design in one colour only.

This makes it easy for the selection tool to select only that colour.

- Your design pieces cannot have holes in them. If they do, you'll need to think carefully.

Now, use the "Select regions by colour" tool to select a colour on the image. You should see the single colour surrounded by a flashing dotted outline. Now that we have a selection, we convert that to a "path". A path is also called a vector, or curve. It is a way of

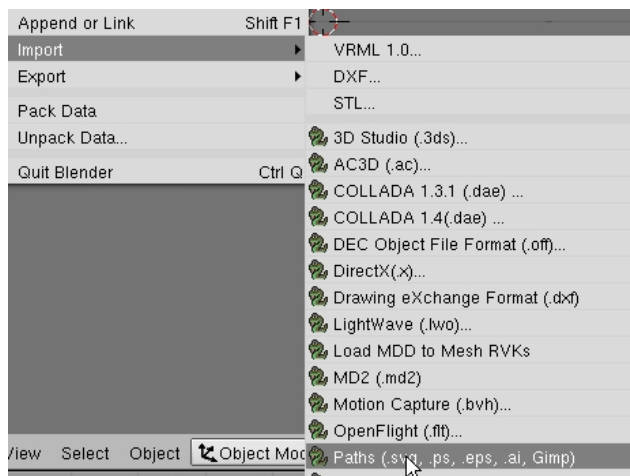


describing an outline in mathematically precise terms. You can then scale or move that line without losing any quality. Use Select --> To Path to convert our selection into a path/curve/ vector.



Now, open the Paths dialog. Select the path you just created, right click on it, and choose "Export Path". Type a name, and be sure to include the extension. Call the file "interface.svg". The .svg extension is for scalable vector graphics.

Now, finally, we get to Blender. Here we can now import the file as a GIMP 2.0 file, giving us the outline of what we created in 2D. Please delete everything you have in your scene. Easiest way is to AKEY (select all) and DELETEKEY. Now go to File --> Import --> Paths (svg, .ps ...). Choose GIMP 2.0 in the next menu. Browse to the location of your exported path .svg file, and select that file. Next, just click on "Import Path", and click "OK" in the import options menu.



Well done if you got this far without hassles! Most people don't even think of this option of combining 2D and 3D design.

You now have a "curve" object in Blender. You options are:

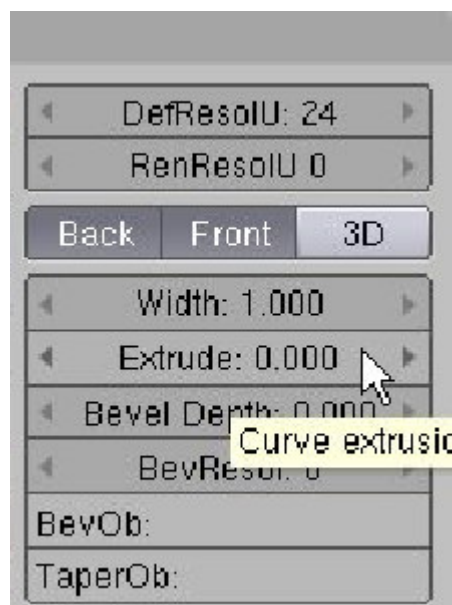
- Use the curve options menu to extrude the outline into 3D
- Convert the outline to a mesh and work from there.

I'm going to use the curve menu options to extrude the outline into a 3D shape. Go to the Edit button. This is where you'll find the curve options:

Extrude: Makes the object 3D

Bevel Depth: Adds bevelled edge

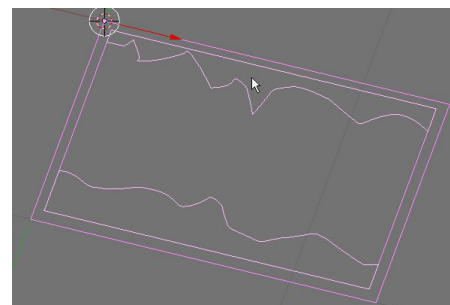
BevReso: Smooths the bevelled edge



I decided to bring the outside orange border in as a curve as well. In this case, Blender found it easy to understand the curve. It consists of a rectangle (4 points) on the inside and a rectangle on the outside (4 points). You may find that if your shape is more complex, GIMP will use different numbers of points to define the curves, which will lead to unreliable results in Blender. There are, however, ways to solve these problems, but they go beyond the scope of this specific tutorial.

After importing the orange outline as well, my Blender curve objects look like the image to the right.

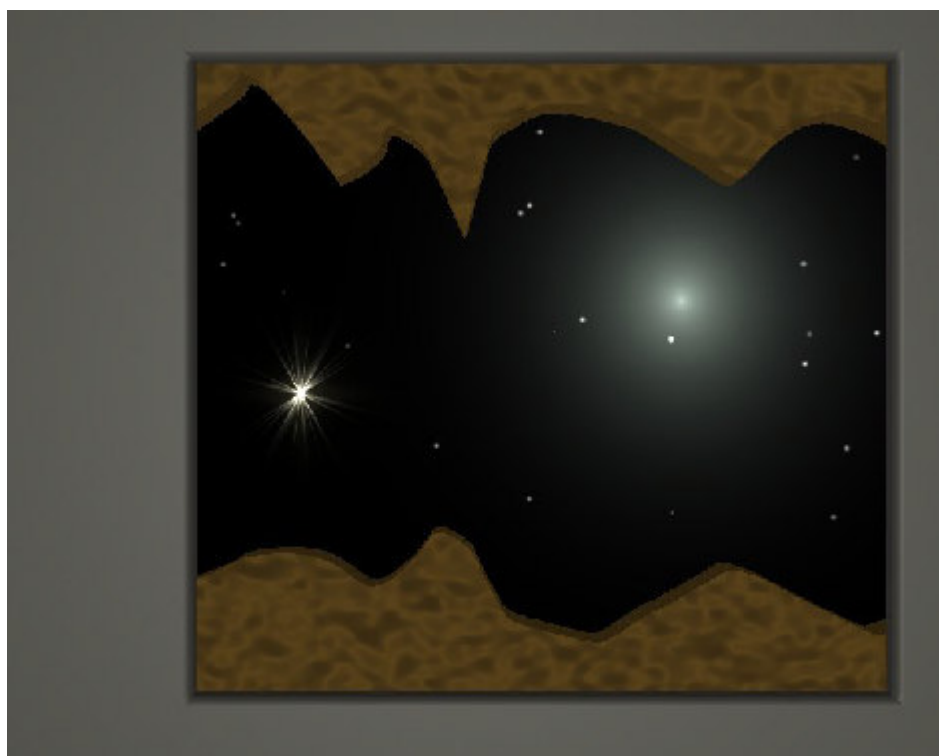
I had a side-scrolling space shooter in mind, so I will change the colours and presentation of the interface to reflect that. I also edited the curve on the outside so that I could have space on the left to add some interface buttons later on.



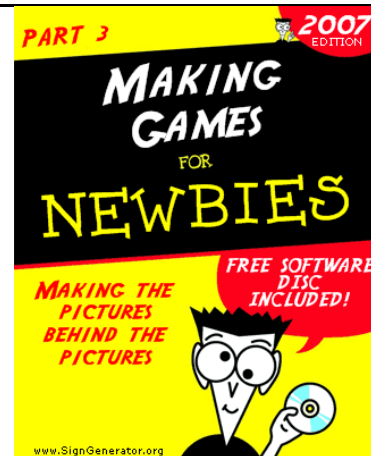
I won't be entering this design for any gaming competitions as candidate for best interface design, but I trust that your creative spirit may take something from this, and apply it to your own work.

Happy Blendin', folks!

?RMAN



Beginner's Guide to Making Games - Backgrounds



This is the third article in the Beginners Guide to Making Games. So far, we have looked at making objects move and checking whether they collide with each other. This month we will look at various options we have for setting backgrounds in our rooms.

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

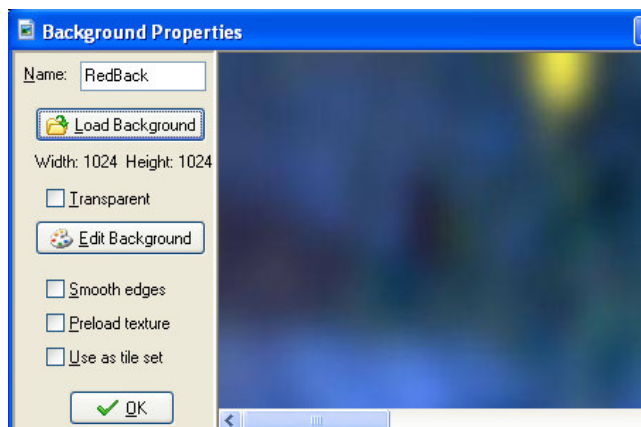
This article is aimed at someone who has just started learning to make games. While it is expected that the reader of the article has completed the first Game Maker tutorial and can thus create sprites and objects, it is quite possible to follow the article without having done so. The article is structured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

This month's article will contain the following sections:

1. A Game Maker tutorial looking at setting a background
2. A Game Maker tutorial looking at a tiled background
3. Fiddling with backgrounds in GML

Backgrounds - GM Tutorial

Backgrounds are critical in setting the theme and feel of a game. In Game Maker, it's easy to set up a background and use it in a room.



Step 1 - Create the Background

Add a new background (either under the backgrounds link on the left) or by selecting Add/Add a background from the menu. Choose the image that you want. For now, select an image that is big enough to cover the whole screen.

In most cases, backgrounds don't need to be transparent or smoothed as they should be designed to look right just as they are.

If your background image is not large enough to fill the whole screen it will need to be tiled. In this case, you need to make sure that the image can tile seamlessly from left to right and top to bottom.

Step 2 - Create a Room

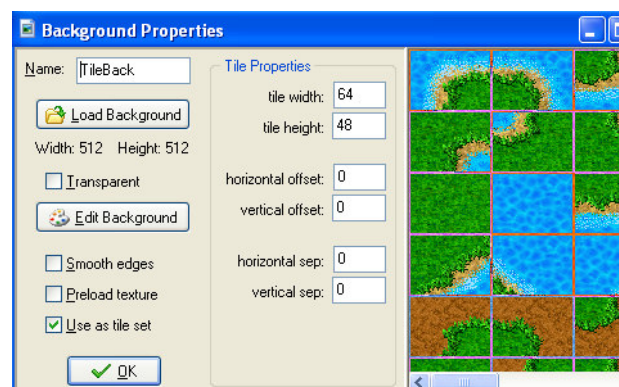
Add a new room and name it. Select the background tab and select the newly created image as the foreground image. Uncheck the 'Draw background colour' check box. It wastes processing power to paint the background and then to draw the selected background over the painted background.

If the image used for the background needs to be tiled set the Vertical and horizontal check boxes and set the number of images to tile over the horizontal and vertical space.

Another option for backgrounds is to stretch a smaller image across the full screen. Very small images will become very pixelated if stretched so only images that are close to the screen size should be used as stretched backgrounds.

Tiles - GM Tutorial

Tiles are very similar to backgrounds, but instead of using a single image to cover the whole room the image used to tile is split into small pieces and added to the screen bit by bit to form an image.

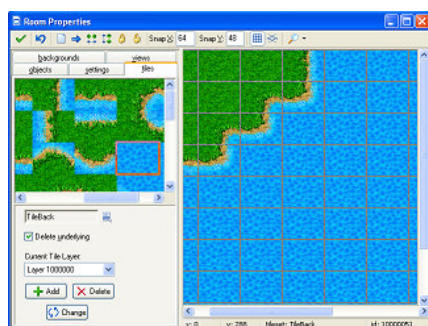


Step 1 - Create the background tiles

Create a background image in the same way as the previous tutorial. Check the 'Use as tile set' option. A Tile Properties panel will open up where the size of the tiles can be set. Note how the individual tiles are marked as the size of the edit boxes.

When creating an image to use as a tile set make sure that all the tiles are the same size and are evenly spaced across the whole image.

Step 2 - Create a room



Create a new room. Select the 'tiles' tab. Select the background that was created as the tile set. On the left side panel the tile set will be displayed. On the right, a grid will be displayed (defaulting to the whole tile set). One by one select the

tile in the left window and click on the grid on the right to set the tile.

By choosing a tile on the left and placing it on the right a full tiled background can be created. This allows complex and customisable rooms to be created to look like a world map or a maze etc.

Game Maker allows multiple different tile layers. Each layer can then be managed separately. This allows the use of different layers for different types of objects. This would for example allow the creation of a map in the background with buildings etc on the foreground.

Fiddling with backgrounds in GML

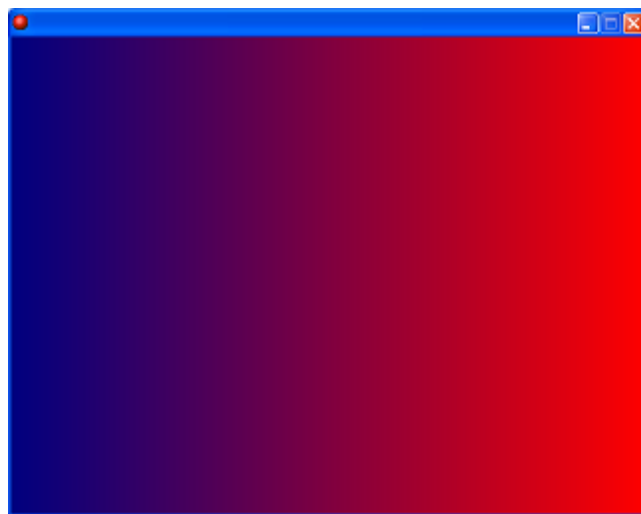
Backgrounds are not the most amazing things to fiddle around with. However, there are some fun things to do with backgrounds. Not all backgrounds need to be created up front. In fact, backgrounds can be created while the game is running and Game Maker has a number of methods available to make this happen.

Add the following code to the Room's 'Creation code' to create (at runtime) a red to blue gradient background.

```
bg =background_create_gradient(room_width, room_height,
    c_navy, c_red, 0, true);
background_index[0] = bg;
room_set_background(room1, 0, true, false, bg, 0, 0,
    false, true, 0, 0, 1);
```

```
background_create_gradient(w, h, col1, col2, kind,
    preload)
```

Creates a gradient filled background of the given size. col1 and col2 indicate the two colors. kind is a number between 0 and 5 indicating the kind of gradient: 0=horizontal 1=vertical, 2= rectangle, 3=ellipse, 4=double horizontal, 5=double vertical.



```
background_create_from_screen(x, y, w, h, transparent,
    smooth, preload)
```

Creates a background by copying the given area from the screen. This makes it possible to create any background you want. Draw the image on the screen using the drawing functions and next create a background from it.

CAIRNSWM

Beginner's Guide to Making Games - Part 2

EXTRA!



Last month, we had a look at different ways Game Maker manages collisions. This month's Extra will look at various methods of doing collision detection in 2D in other programming languages. The benefit of using Game Maker is of course that you don't actually need to understand all the intricacies of the process.

When looking for collisions between 2D objects there are two basic methods of doing so:

1. Bounding Box collision detection
2. Pixel perfect collision detection

A third alternate method is also possible through

3. Map-based collision detection

Many developers feel that Bounding Boxes do not give an accurate enough level of collision detection and insist on doing more fine-grained collision detection. However in most cases there is no real visible difference between bounding box and pixel perfect collision detection.

Bounding Box Collision Detection

Bounding box collision detection is done by defining simple geometric shapes around the sprites used for the objects and checking if the geometric shapes overlap. Typically, the easiest shapes to check for overlap are rectangles.

The overlap of two rectangles can also be defined as the intersection between the rectangle or even more simply as being true if any corner of the second rectangle is inside the first rectangle. For example if each rectangle is defined by x1,y1 as the top left corner and x2,y2 as the bottom right corner we can define a collision any time rectangle two's x1 or x2 is between rectangle one's x1 and x2 and the same for the y coordinates.

```
Function BoundingBoxCollision(rect1, rect2 : rectangle)
{
  If ((rect1.x1 < rect2.x1 and rect1.x2 > rect2.x1) or
    (rect1.x1 < rect2.x2 and rect1.x2 > rect2.x2)) and
    ((rect1.y1 < rect2.y1 and rect1.y2 > rect2.y1) or
    (rect1.y1 < rect2.y2 and rect1.y2 > rect2.y2)) then
    Result := true
  Else
    Result := false
  End if
}
```

Bounding boxes are usually smaller than the full size of the sprite being used as the picture of the sprite will often be surrounded by a transparent border. In addition, the bounding box could also be defined as a circle with a certain radius. If the distance between two circles is less than their combined radius, there is a collision.

If the shape of a sprite is not generally rectangular or circular it is also possible to define the collision area of a sprite as a collection of shapes and check for collision with any of the bounding box shapes.

Pixel Perfect Collision Detection

Pixel perfect collision detection is where each pixel of the two sprites is checked to see if they are in the same place as each other. Due to the number of tests needed to implement pixel perfect collision detection, it should only be used when bounding box collision detection is clearly not accurate enough. (such as when checking if a starfish and an octopus collide).

Of course, it is not necessary to check every single pixel of the two sprites, in fact in most cases it would be sufficient to use only the outline of the sprites or even in many cases just a few of the outline pixels.

Typically, pixel perfect collision detection is used along with bounding boxes. If a bounding box collision is detected, then more detailed tests are done to check that there are in fact pixels that collide with each other. By wrapping the pixel perfect collision detection within a less complex test the performance impact of the pixel testing is minimized.

Map-based Collision Detection

Map based collision detection works on the presumption that firstly you know where in the game world each item is and secondly that only one item can exist in each space of the world. By using an array to represent the world and storing a link to the item that exists in each space it's a simple check to find which object is in which space at any time. The use of an array to store where each and every item of the game is, makes it a lot quicker to check for a collision.

Whenever an object moves, it is first removed from the world array, its new location is calculated and the array is checked to see if another item is already in the required location. If no item is found in the destination location, the item is again added to the world array. If an item is already in the location the item wants to move to a collision between the objects is created.

Map-based collision detection often requires objects to check for collisions in more than one location in the world array due to things such as item size or just to make the collision checking more realistic. Buildings are typical examples of where an item needs to exist in more than one map location simultaneously.

Map-based collision detection is often very suited to isometric-style games.

Choice

The choice of collision detection method is often based on the game being developed. Each method of collision detection has its place. Most games will find that bounding boxes are sufficient for their needs.

CAIRNSWM

ROACH TOASTER 2: PICKING OUT THE BUGS

PART 7

(Wondering what we're talking about over here? check out Dev.Mag Issue 10 for the beginning of this Project series!)

Well well ... This series has come a long way now. It's almost a year since it started. After the previous article, I have not had time to work on Roach Toaster 2 due to holidays, work and school.

As I write this article, I have one week of school left. After that, it's all just exams until the end of the year. This is also my senior year in school, so my work will take priority.

In previous exams, I have worked on games throughout, but as I sit here, I am not quite sure if I will have enough time to really pay attention to Roach Toaster 2 during the exams.

RT2 has been through thick and thin. It's taking me much longer than expected. As I have mentioned previously, I thought I would be finished with beta by December 2006.

So, considering that I don't really know what will happen and RT2's tumultuous devving, I have decided to end this series for now. You will most likely see a post-mortem once it is done, or a beta-feedback article.

So, I thought I'd use this article to once again explain what Roach Toaster 2 and its development.

"What the bug is Roach Toaster 2?"

Roach Toaster 2 is the sequel to the award-winning Roach Toaster 1. By "award-winning", I mean it won an international and local competition. Roach Toaster 2 also won a place in a local competition as a WIP (Work In Progress).

RT2 places you in the command of teams of roach toasters as you defend Big City from being destroyed by a roach infestation.

It contains multi-level gameplay. You must manage your teams on a city-wide scale to obliterate the roaches. The original and addictive gameplay from RT1 is still here in a new guise. With updated gameplay and new enhancements, each level will be a unique challenge that can be tackled the way you see fit.

"What is the story behind RT2?"

After you defeated your neighbourhood's roaches in RT1, your team set off to Cancune to enjoy a well-deserved holiday. Where did they get the money? Don't know, but who cares?

Unbeknownst to them, the Broodmother incubated inside the Hoover. She then came back with a vengeance. She is livid, and wants to destroy Big City once and for all.

You must defeat her.

Where did you get all the money? Is there something greater at stake? World-domination? Find out when RT2 gets released.

Roach Toaster 2 has been an immense learning curve for me. It's been my puppy for more than a year now. Despite everything else that occupied me, I always had time to fit in a bout of RT2 devving.

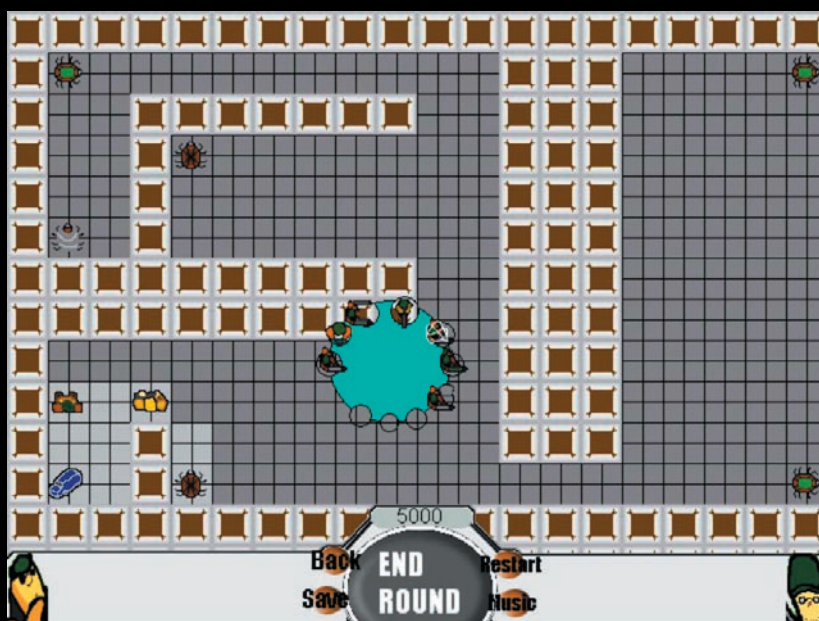
I can't wait to finally complete the finished, polished product.

Let me finish by saying that I already have Roach Toaster 3 planned out. How it will play, or what it will be, will have to wait.

Keep toasting!

For more info on Roach Toaster 2: Big City, head on to <http://www.shotbeakgames.za.net> or the new site <http://www.roachtoaster.com>

TROOJG





CODING ETIQUETTE: DOXYGEN

It is a fact of programming life that programmers will make mistakes in their code. These mistakes or errors are not usually down to “bad programming” or any lack of talent, but merely due to the sheer scope that most programs are comprised of. When any one programmer is churning out hundreds upon hundreds of lines of code a day, it’s only a matter of time before some of those lines will contain errors.

As discussed in the second article in this series, proper commenting of code is a very important part of game development. Building upon this idea, people have gone about developing systems that allow for developers to easily create fully featured documentation of their code by merely altering their comments to include markers which then allow the comments themselves to be converted into documentation. One such system that allows this, and that I fully advocate the use of, is Doxygen (www.doxygen.org).

Doxygen is incredibly useful because it is very simple and straightforward to use in its most basic form, and the results that you can obtain from it are terrific for anyone managing the code base of a project. It allows very robust documentation to be generated for all of the most important areas of your code such as files, functions, variables, and data structures with a

minimal amount of effort. It also provides much more complex documentation features such as call graphs, dependency graphs, inheritance diagrams, and collaboration diagrams for those developers who want to integrate these more complex features into their project documentation. Another great thing about Doxygen is that it has native support for most of the more popular languages such as C, C++, C#, Java, Python, and even PHP (sorry, no native Delphi support!).

The documentation that is generated by Doxygen can be output into HTML, PDF, and (my personal favourite) compressed HTML (CHM), amongst other formats. The benefits of having this documentation generated for you automatically should be fairly obvious as it provides the ability to have your entire code base documented in whatever format suits you (ie. PDF for reading or printing, or HTML/CHM for PC-based hyperlink navigation and searching), with very little overhead on the part of the person writing the comments.

This documentation can then become a very important tool for your team as it will allow programmers to easily search for and see information (in an easily accessible and readable format) on any code that they may need to deal with. This functionality becomes extra

helpful if they are dealing with sections of code written by other programmers, and especially so if the original writer of the code is no longer on the team or isn’t available to ask for help in understanding the code that they wrote.

The basic idea behind Doxygen is that it looks for and parses various token keywords that are placed inside of specially formatted commenting blocks that work inside of the language you are writing your code in. There are four kinds of commenting blocks supported by Doxygen, and any Doxygen keyword that is found inside of these comment blocks will be parsed and used for documentation generation, while any keywords found inside of the standard language commenting blocks will be ignored.

Supported Commenting Block Formats:

```
/**      add comments here...    */
/*!      add comments here...    */
///      add comments here...
//!      add comments here...
```

One thing that you must note is that by default, Doxygen assumes that the code being commented on is located after the comment block. In order to tell Doxygen that the code is actually located before the comment block, you must



add a extra '<' to the comment format. This style of commenting is most useful when you are creating a comment for a variable defined locally, globally, or inside of a data structure.

Trailing Commenting Block Formats:

```
/**<    add comments here...    */
/*!<    add comments here...    */
///  
///  
//!<    add comments here...
```

The basic code-items that Doxygen provides documentation support for are files, functions, and data structures. The keywords that denote these items are placed into comments that exist next to them, and are used to specifically define various types of code for the documentation generation.

Basic Doxygen Keywords:

\file -- Used to denote a file

\struct -- Used to denote a structure

\class -- Used to denote a class

\union -- Used to denote a union

\enum -- Used to denote an enumeration

\var -- Used to denote a variable

\def -- Used to denote a #define

\brief -- Used to provide a brief description about something (ie. function, structure, etc.)

\param -- Used to mark each parameter of a function

\return -- Used to mark the return value of a function

Doxygen also provides various miscellaneous keywords that can be used at any time in order to mark various points of interest in the code. These keywords are very helpful in creating more robust documentation, especially for the purpose of adding specific types of notes to complement the more technical nature of the documentation related to the code itself.

Misc Doxygen Keywords:

\ref -- Creates a hyperlink reference to a specific keyword

\note -- Creates a 'NOTE' in the source documentation

\warn -- Similar to \note, but it is more visible in the generated documentation

\todo -- Creates an entry in the 'TODO' section of the generated documentation

\fixme -- Creates an entry in the 'FIXME' section of the generated documentation

Remember that all of these keywords comprise only the core handful of keywords that Doxygen has available for use. There are dozens and dozens more keywords that exist, and it is up to each and every developer to find out which ones may or may not be helpful for their documentation needs.

COOLHAND

Main Page

Classes

Files

File List

File Members

vector.cpp File Reference

File containing vector math utilities. [More...](#)

Classes

struct	tVec2D	Structure that defines a 2-dimensional vector. More...
struct	tVec3D	Structure that defines a 3-dimensional vector. More...

Functions

float	Vec2DDot(tVec2D *psVecA, tVec2D *psVecB)	Utility function to calculate the dot product between two 2-dimensional vectors.
float	Vec3DDot(tVec3D *psVecA, tVec3D *psVecB)	Utility function to calculate the dot product between two 3-dimensional vectors.

Detailed Description

File containing vector math utilities.

Author:
Luke Lamothe

Date:
Saturday, 18 August, 2007

Version:

Simple Example of Doxygen Commenting

```

/**
    \file    vector.cpp
    \author Luke Lamothe
    \date    Saturday, 18 August, 2007
    \version 1.0
    \brief   File containing vector math utilities
*/

/**
    \struct tVec2D
    \brief   Structure that defines a 2-dimensional vector
    \note    There is a \ref tVec3D structure for use if you need to use 3-dimensional
vectors
*/
typedef struct
{
    float    x;        ///< The x component of the 2D vector
    float    y;        ///< The y component of the 2D vector
}tVec2D;

/**
    \struct tVec3D
    \brief   Structure that defines a 3-dimensional vector
    \note    There is a \ref tVec2D structure for use if you need to use 2-dimensional
vectors
*/
typedef struct
{
    float    x;        ///< The x component of the 3D vector
    float    y;        ///< The y component of the 3D vector
    float    z;        ///< The z component of the 3D vector
}tVec3D;

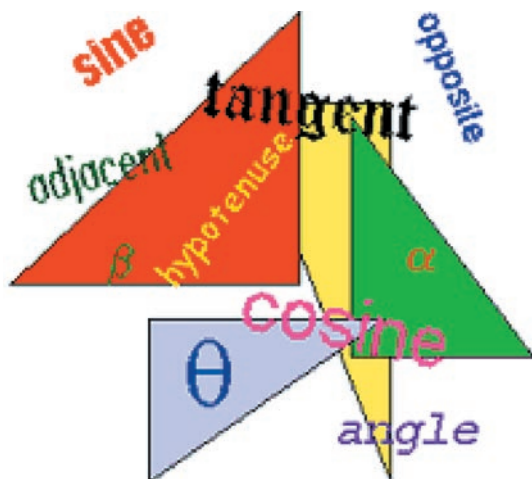
/**
    \param psVecA The first vector to use in the dot product calculation
    \param psVecB The second vector to use in the dot product calculation
    \return float The result of the dot product between the two vectors
    \brief Utility function to calculate the dot product between two 2-dimensional
vectors
*/
float Vec2DDot(tVec2D *psVecA, tVec2D *psVecB)
{
    float    fResult;

    //calculate the dot product of the two vectors and return it
    fResult = (psVecA->x * psVecB->x) + (psVecA->y * psVecB->y);
    return fResult;
}

/**
    \param psVecA The first vector to use in the dot product calculation
    \param psVecB The second vector to use in the dot product calculation
    \return float The result of the dot product between the two vectors
    \brief Utility function to calculate the dot product between two 3-dimensional
vectors
*/
float Vec3DDot(tVec3D *psVecA, tVec3D *psVecB)
{
    float    fResult;

    //calculate the dot product of the two vectors and return it
    fResult = (psVecA->x * psVecB->x) + (psVecA->y * psVecB->y) + (psVecA->z * ps-
VecB->z);
    return fResult;
}

```



GAME CODING WITH TRIGONOMETRY

PART 3

(What manner of trickery is this? To delve into the mysteries of trigonometry properly, maybe Dev.Mag Issue 14 would be a better starting point!)

Welcome back, fellow coders! Prepare yourselves for the final, bumper episode of the Trig Trilogy, where I cover a whole lot of awesome trig techniques for your game-making pleasure. We've got a lot to cover, so let's jump right in. It's time for my first trick! Now, as you can see, there is nothing up my sleeve ...

Relative Angling 101

No matter what kind of game you're working on, more often than not you'll need to calculate the angle between two objects for some reason. It's time to look at a chunk of code that can make your life very easy in this regard:

$$\text{Angle} = \arctan \left[\frac{(Y_2 - Y_1)}{(X_2 - X_1)} \right]$$

What this code does is first calculate the difference in x and y coordinates between the two objects, then divide as per the arctan ratio. X1 and Y1 represent the coordinates of the object you want to calculate the angle from. X2 and Y2 are, obviously, the coordinates of the object whose angle you want to measure relative to the first object.

We perform the subtraction because all trig calculations assume that our origin is at point (0,0) on the Cartesian plane. As a result, we need to use relative measures, essentially making point (X1,Y1) our origin. If this all seems really confusing to you, worry not. I've summed it up for you in these two diagrams. As you can see, there's a massive difference between the two depending on whether you subtract or not!

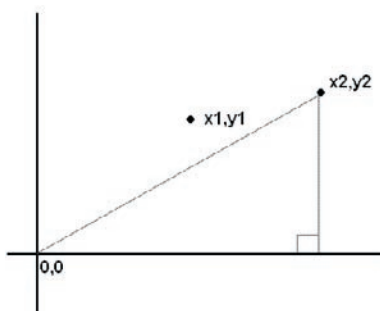


Figure 1a: Angle calculated without subtraction

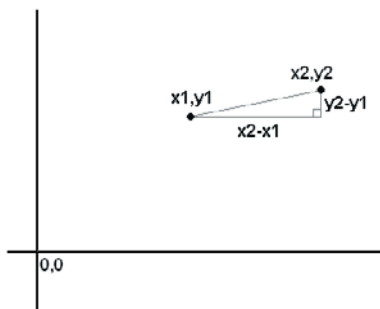


Figure 1b: Angle calculated with subtraction

One final thing to note is that this algorithm will give you the angle in radians, as was discussed last month. Remember to convert the answer to degrees if that's what your game requires!

And there you have it: all you need to know about calculating the relative angle between two objects! That wasn't so complicated, was it?

GAME MAKER TIP:

There are several functions you can use in GM to calculate your angles. You can use the same algorithm as above or use the slightly altered `arctan2(y,x)` function, meaning you don't have to divide first. By far the easiest, however, is to use the `point_direction(x1,y1,x2,y2)` function, which also converts the angle to degrees for you as a little added extra! The choice is yours.

Goin' Round in Circles

One of the most prevalent problems that first-time coders have is making objects follow a circular path. There could be any number of uses for this - having a targeting reticle orbit around your game character, or perhaps

a spinning-blade-on-a-chain trap for hapless players to wander into. Either way, if you've ever wanted to have one thing spinning around another, this is exactly what you need!

Fortunately, circular positioning is not at all difficult to do for us mighty trig-wielders. As always, let's look at exactly what we're trying to accomplish here in the form of a handy diagram.

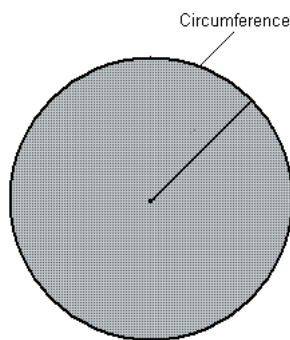


Figure 2: The humble circle

Figure 2 depicts what is commonly referred to as a "circle". This shape is notable because it has a constant radius. That means that regardless of where you are on the circumference, you'll be exactly the same distance away from the centre. This gives us an important clue.

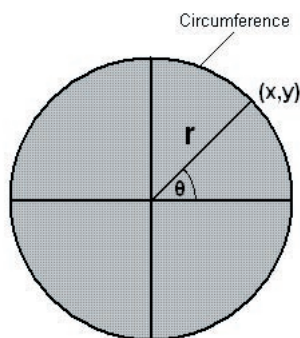


Figure 3: Shazam! Trig'd!

Figure 3 depicts our situation more clearly. What we really want to do in code terms is to calculate the x and y coordinates of an object while ensuring that, regardless of its angle to the centre, it always remains

within a set distance of that centre. We can do that using the following two algorithms:

$$X = \text{CentreX} + (\text{Radius} * \cos(\text{Angle}))$$

$$Y = \text{CentreY} + (\text{Radius} * \sin(\text{Angle}))$$

"centreX" and "centreY" refer to the coordinates of the position/object we want to rotate around. It's important that we have these, otherwise (as with our angle calculations) the object is going to rotate around the screen/level origin (0,0) instead of the centre we want. "Radius" is obviously the radius of our circle, and "Angle" is the relative angle between the centre and the orbiting object. Changing the Angle will rotate the object around the centre. Increasing or decreasing the Radius will move the object further from or closer to the centre. Changing the centreX and centreY coordinates will, of course, move the centre and the spinning object with it. That's all there is to it!

A final note. If you're implementing a spinning object where you keep increasing the angle to create rotation, it's wise to reset the Angle to 0 once it exceeds the measure for one full rotation (either 360 degrees or 2π radians). Although it shouldn't make much difference to your calculations, it certainly keeps things cleaner, and ensures that you know exactly what range of numbers your code is working with. This can make debugging a lot easier.

GAME MAKER TIP:

Game Maker makes things easy for you again by providing the `lengthdir_x(len,dir)` and `lengthdir_y(len,dir)` functions. These take your circle/ellipse radius (`len`) and direction (`dir`) and calculate the correct x and y values for you just like we did above. No mess, no fuss, and you can use them any time you need to resolve an angled line to x and y coordinates, not just with circles. Just remember to add the `centreX` and `centreY` coordinates to the final answers!

What's this? There's more?

Yes! Sometimes you may want an object to move in an elliptical fashion, especially if you're creating an isometric game where the perspective means that you need ellipses to represent circles and circular movement. Fear not, good people! By altering the radius values in the circular positioning algorithms, we can make objects follow an elliptical path.

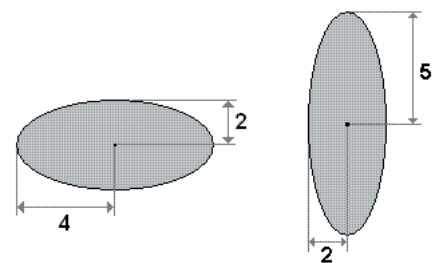


Figure 4: Ellipses gone wild

The trick is to realize that the radius in an ellipse is not constant - the radius for its vertical proportions is different from the radius for its horizontal proportions. For instance, the first ellipse in Figure 4 has a vertical radius of 2, which is half of its horizontal radius of 4. We need to alter the radius values in our algorithms to reflect this distorted relationship. For the first ellipse, our expressions would look like this:

$$X = \text{CentreX} + (4 * \cos(\text{Angle}))$$

$$Y = \text{CentreY} + (2 * \sin(\text{Angle}))$$

For the second ellipse, our expressions would look like this:

$$X = \text{CentreX} + (2 * \cos(\text{Angle}))$$

$$Y = \text{CentreY} + (5 * \sin(\text{Angle}))$$

I'm sure that the relationships between the radii that you specify and the ellipse's proportions are becoming crystal clear to you. Other than that, the code works in exactly the same way as with a regular circle.

Wave Motion - Cowabunga!

For our final trick, we're going to cover a very useful function of trig - creating wave motion (or oscillation), for game objects. Ever played a scrolling shooter where the enemy ships move in a wavy pattern across the screen, or stood amazed at the helical particle trail left by a weapon? Well, here's how it's done!

All the time that we've been doing trig, you'll have noticed that we've been using two algorithms - one to calculate the X (horizontal) co-ordinate, and another to calculate the Y (vertical). As you learned while we were covering circles and ellipses, we can use the two to create rotation around an object. Altering the radius values for either equation lets us warp the circle's proportions, creating elliptical motion.

Have you asked yourself what would happen if we left one of the calculations out? What if we neglected to calculate the X value? What if we left the Y value out in the cold? Well, we'd still get movement, but it would be exclusively on the axis for which we calculated it. That's the magic behind wave motion. To demonstrate, yet another diagram:

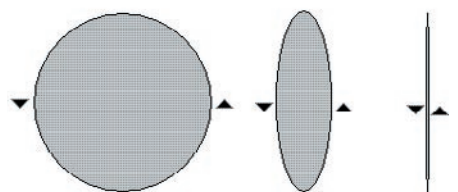


Figure 5: Thinner and thinner

The arrows in figure 5 illustrate the rotational motion. You should notice that as we reduce the horizontal component's radius to zero, the rotation becomes less "circular" and more "wavy", wobbling between the radius points and the centre. The point that I'm trying to convey is that, used alone, a trig function can be used to introduce oscillation to an object's movement.

Here are the algorithms you would use:

$Displacement = [CentreX \text{ or } CentreY] + (Radius * \sin(Angle))$
OR
 $Displacement = [CentreX \text{ or } CentreY] + (Radius * \cos(Angle))$

These algorithms are actually very similar to the ones we use for circular positioning. As before, Radius will determine the maximum distance from the centre our object can move, and Angle will determine at what point in the wave our object is. However, an essential thing to note is that we no longer have to use Sin for vertical displacement and Cos for horizontal displacement, because we're only displacing along one axis now.

Let's back up a bit. If we don't have to care about using Sin or Cos, what's with the big "OR" in there? Well, it depends on how you want your wave to behave. Oscillation using Sin or Cos results in slightly different wave patterns:

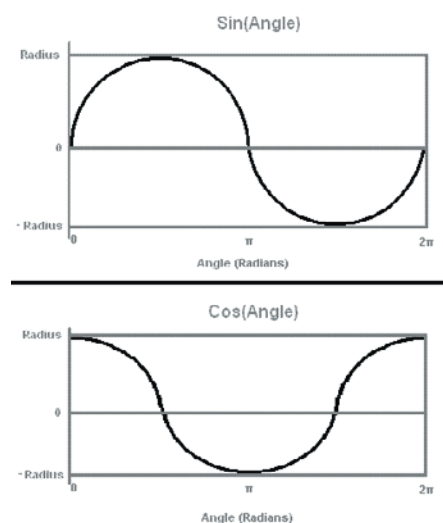


Figure 6: Sin and Cos graphs

The Sin graph begins its motion at a displacement of 0. This means that using Sin as your trig function will result in your object's motion starting at the centre point you specify. Likewise, the Cos graph begins its motion at the maximum point, so using Cos as your function for oscillation will result in your object starting at the maximum radius you've set.

So how does this translate to our code? Well, for example, will result in oscillation along the x axis starting at the centre, while will result in oscillation along the y axis starting at the radius. Clear?

One final thing. You may already have gathered this, but it is possible to alter the speed of oscillation by multiplying or dividing your Angle variable. You can use this to easily change how quickly the object moves along the wave. Multiplication will make your object oscillate faster, while division will slow it down. This can result in quite a few interesting special effects or challenges in your game.

It's Over!

This marks the end of this little introduction to game programming with trigonometry. As you've seen over the past three installments, trig has an incredibly wide range of effective uses when programming games. Hopefully you've not only grasped the techniques covered here, but you've also gained the ability to apply trig to whatever problems you may come across when coding in the future. Properly applied, trig is a highly flexible, incredibly powerful tool in your programming arsenal, especially if you plan on creating 3D games later on. Good luck, and happy coding!

GAZZA_N

FOR A SPEED SURPRISE, GO OPTIMISE!

Optimising data, optimising code ... it all sounds like such a waste. This article outlines some simple and effective strategies on spotting and fixing bad code concepts, which slow a program down in more ways than one. We will look at these strategies and cover some new ideas which are all relative to creating games, especially when stepping out into coding your own engine or even just coding. Regardless of the application or code usage, optimisation is important yet overlooked.

Writing cutting-edge games is on all of our to-do list, but are we really TRYING to get there, or are we in a frenzy to learn as much as we can as soon as we can, skipping vital steps which get us to a better rounded end product? I'm making this statement as it is one which resides in my mind whenever i step into any project. Rather take the time and make a masterpiece than skip a few steps and end up with another project to revise and fix.

Let's see what we have here...

Optimisation: Modifying a system to improve its efficiency - whether it's a single loop, a single program, or even the entire Internet.

Profiling: Setting up points in code that count the number of clock cycles and/or time it takes to execute certain blocks of code. This effectively singles out the slow code in your program/system and allows you to revise and improve the system.

Code optimisation may be something out of your reach, or so you thought. We spend time making games and i have run into cases where i end up with a non-viable solution which ends

up in the "bin" or in the revise folder, when a simple look at optimising DURING a project has saved me more than a dozen wasted projects.

Optimisation must be approached with caution. Tony Hoare first said, and Donald Knuth famously repeated, "Premature optimization is the root of all evil." It is important to have sound algorithms and a working prototype first.

System design and the internal architecture of the code is the largest weight on performance of a system. The choices in this stage affect efficiency more than any other area during design.

This was something i read recently which has quite the obvious pointed out to you - to look carefully at optimisation:

"Premature optimization' is a phrase used to describe a situation where a programmer lets performance considerations affect the design of a piece of code. This can result in a design that is not as clean as it could have been or code that is incorrect, because the code is complicated by the optimization and the programmer is distracted by optimizing."

Sometimes, early optimisation overthrows benefits and can destroy the project core with sloppy or dangerous code.

Optimisation is a subject many developers overlook, which is a fatal mistake in the industry, especially on the game development side. People think due to the fact that technology is daily becoming more and more powerful, code and other optimisation is not really something that is necessary in projects, but they forget the main point which they themselves would look

at when evaluating a product or solution: which one is faster? If the next creator of the same solution was marketing it as faster, whose solution would you pick up off the shelf?

Optimisation does not come without costs, as it is usually more difficult to maintain an optimised program than an unoptimised one, as many optimisations sacrifice maintainability for speed, or resources. When you optimise each piece of code, you need to be aware of a few things of importance. After I have "optimised" a piece of code, I must go back and check the correctness, where sometimes the code will cause incorrect output. Also, you should measure the performance and compare it with that before the optimisation. Often enough, it will be slower due to compiler optimisations which you cancelled out, or other less obvious changes in the compilation and execution stages. Once I have sorted it out and there is an increase, I may check the performance according to the goal set out, and stop optimising so I don't change anything unnecessarily.

There's a few simple tactics to optimise your code and not have to run through a hundred thousand lines of code searching for bottlenecks. Take these steps practically in your current/next project and notice how little time is spent on optimising small areas of code post-completion. Optimisation happens during coding as well as design. If you are inclined towards speed, it becomes a major part of the coding stage. Below, you will read the strategies I employ when working with a relatively "slow" acting program, and these apply to all code strategies and methods as well as most applications of optimisation - not only games. For example, I use these common techniques when I work on mobile ap-

plications due to the limited processor power, which greatly speed up time on the device, and on web development projects where effectively managing code and resources means less server-side overhead...less cost, less hassle.

Code efficiency

Creating code that is fast AND portable AND functional is a considerable task and takes practice to be able to catch the dodgy code in its tracks and fix the bottlenecks. Code profilers are your friend - run them and filter the results for anything that is slower than it seems it should be, then go to your source and optimise it. This is where the techniques and ideas behind optimisation can greatly

improve your code speed when employed accurately and in context with the project.

Loop hoisting

I found myself doing this a lot in my code, and after learning about the inefficiency, it became a habit to automatically change it if I saw it, or straight-up code it as I go. The idea behind these loops is illustrated easier through code, and with time and focus you could easily use the following to prevent useless redundancy. Below is an example in pseudo-code.

```
for counter = 1 to 10
    if (thisIsTrue) then
        doThis(counter)
```

```
else doThat
end if
```

Let's analyse this code ourselves. The loop runs ten times checking a certain value. You may notice that the code is inefficient because EVERY loop it has to break off into conditional statements, which is using up CPU power, as well as the CPU trying to guess logical output which if is wrong (+ 50% of the time) it needs to backtrack to carry on with the right instruction and reload it. This simple problem looks not so intense, until it is being used to poll or test every frame of a game, and aiming for higher framerates will result in even more work. Let's look at the new code, which uses loop hoisting as an effective means to release bottlenecks.



```

if (thisIsTrue) then
    for counter = 1 to 10
        doThis(counter)
    else doThat
end if

```

This is QUITE a large improvement - it now handles the if statement once and once only, not ten times every frame. When it comes to controlling larger structures, nested loops can be hoisted to avoid excessive use of the processor.

Data types and sizes

Back in the day, using smaller data types was better because of certain impacts on the system and its processor. These days, we see that using smaller variables is often not an issue for the size of the matter, but more for the speed. If you're using smaller sized integers, use ONLY that size in that piece of code or application. For example, using a small integer value and a long or larger integer value requires a conversion to different sizes EVERY time that code is used or accessed, wasting time away in the cycles of the processor. This has a large impact on the processor in more intense applications, such as those with image processing, video encoding, large array usage and many areas in game map management. These techniques are especially useful when it might be that extra "un-lag" strategy you need.

Array sizes

Remember that working inside your processor is many instructions and commands all usually which work a lot more effectively with powers of two. A basic understanding of what happens inside the processor will show that it has to make extra calculations for a 3 x 4 grid, because it needs to use multiplication and other means to find the index in the array, where as a simple 4x4 grid would use binary shifts which are a lot faster and more effective, reducing the

time spent on the array. These situations are common but often unnoticed and discarded because more array space may be needed, which again these days seems more or less irrelevant.

Profiling and disassembling

Running profiling programs on your code can give you a large amount of feedback. This will help you pinpoint various issues in code, and effectively eliminate the problem areas. Finding the area of problem is one thing, optimising it is another. Another strategy I use is to disassemble the compiled code, which gives you the assembler representation of your code. If you know or learn assembler, it helps you easily see wasted instructions and find better ways of removing redundant code.

Data management

Please, believe me when I say this: data is the most often overlooked bottleneck. It slows down almost every part of an application if implemented badly. Looking at revising a data module of sorts could be a hefty task, but weigh up how much it would be worth it if your data was well-placed, always ready to be used, and always efficiently not hogging up memory. The projects styles always have some restriction of sorts, and some concept of sorts, and almost always uses a data setup that is badly implemented due to the "it's data, it just needs to be there when i need it" attitude. This, above all else, can speed an application up by a MASSIVE amount. Load what's needed, and remove it when its not. There's no point in keeping the data in memory if the user decides to choose "Quit to menu". They're probably not going back into that section of data and memory right away, so unload it and load what's needed next, instead of having to unload it again while he is expecting to load a new level or different save game. Data management and effectively using the data will benefit you in any type of program.

Function sizes

If your function code is crossing boundaries - as in not doing its SPECIFIC function - it is being wastefully implemented. If you code your functions effectively, the profiling will be a hundred times easier. This is because it has to find the bottlenecks in the code, and if you have twenty functions in one, the profiler (usually working on a per-function profile) will tell you it's slower.

Parallel execution

Amdahl's Law states, "Parallel execution is the most common technique for speeding up large applications, but it alone is not sufficient for all cases." In particular, Amdahl's Law claims that the inherently sequential portion of a program is the limiting factor of speed improvements.

These are very basic and not in-depth, but will get you started before you hit your favourite search engine and find more information on taking control of code and effectively speeding up a system. Game development is an important optimisation concern as the faster a frame can be processed, the more that can be done in one frame. For example, this helps when speeding up the code for drawing the scene will allow enough of a speed increase to introduce another interesting effect or physics change in a game engine. This article is far from complete, but is sufficient in my views to take you to the level of understanding that one needs to watch out for some common pitfalls in code practices. Take this with a pinch of salt, and ignore what you don't need or like! Enjoy!

FUZZY SPOON

THE HISTORY OF I-IMAGINE

Part 7: Enter the publisher

(Who? What? Where? If you're lost concerning I-Imagine, check out Dev.Mag Issue 10 for the start of this article series!)

As the end of 2000 approached us, we were already well under way with our Xbox development of Chase. From what I remember, it took us less than a week to get almost all of the game as it was ported from PC to Xbox, which is a terrific testament to not only how comparable the basics of Xbox development were to PC development, but also how robust and easy to use the Xbox SDK was, as well as how good their support system was through not only their development Newsgroups, but also through their private technician support. Once we had finished getting the game up and running for Xbox, we began to perform various major overhauls on it in order to take advantage of the superior hardware that we now had at our disposal.

We started by completely throwing out all art assets that we had previously created and went about making all new worlds, props, and vehicles with polycounts that approached between 5 and 10 times what we had previously been using for the PC version. We also now had access to guaranteed shader technology (no more worrying about end-user specs!), so we were able to start incorporating some special effects such as bump mapping, light mapping, realtime cube-map reflections, realtime shadows for objects, and object self-shadowing. While this graphical overhaul was being done, additional changes to the game itself were being incorporated. Dave added motorcycle physics, complete with the ability to do "wheelies", to the game which now gave us four types of vehicles that we could use (standard four wheel vehicles, three wheeled vehicles, multi-body articulated vehicles, and motorcycles). Pedestrian and traffic AI were also overhauled in order to make them behave smarter as well as more realistically, and a

dynamic replay system was added which allowed levels to be recorded and played back.

Unfortunately, all of this exciting stuff was offset by more unsettling events within our team, the first of which being Matt's wife moving back to the US in early 2001. She was tired of not being able to work, due to the lack of a work permit, and she also was starting to miss her family back home. This obviously put a lot of strain on Matt as he really wanted to stay and finish the development of Chase, but in the end, he did the only thing that he could do and also returned to the US sometime around August of the same year. Dave and Staci had a baby boy early in the year, which I think made Staci miss her family even more, and as a result, she also moved back to the US when we went overseas for E3

in May. As expected, Dave also left us in order to be with his wife and child at the start of July, once he had come back to South African after E3 in order to tie up some loose ends. Luckily, we were able to find two very able replacement programmers for them in Chris and Matt who along with a third programmer, Andrew, helped us to complete the development of Chase. However, before Dave and Matt would leave us to be replaced by Chris, Matt, and Andrew, we had the small matter of E3 2001 to prepare for.

The lead up to E3 was pretty exciting for us as we were asked by Microsoft to submit some video footage of Chase for them to incorporate into their E3 promo video which would be running on the big screen in the middle of their stand. From what I can remem-



ber, I think that they ended up using three or four clips of the video footage, each one running for between three and five seconds or so.

It was definitely quite an awesome feeling to be standing in the Microsoft booth at E3 while watching footage of your game up on the big screen! But before we even got to Los Angeles for E3, I had to endure the craziest crunch of my life. For the most part, the deadline leading up to E3 was relatively normal. We had some pretty good stuff to bring with us already due to the pre-E3 deadline that we had imposed on ourselves in order to get the video footage to Microsoft, but there was one important feature that we needed to have working perfectly before going to E3, and that was our replay system.

The system itself was basically a list of inputs from the player, which when played back at their appropriate time, would recreate a previous gameplay experience perfectly. However, this means that the entire engine needs to behave in a deterministic manner, and if any code is added that goes against this philosophy, it will cause the replay system to come to a grinding halt. Unfortunately, this is what began to happen to us the day before we were due to fly to E3. I don't remember the specifics of what caused the system to break, but I think it was an addition to the AI code which was using the "random" function in ways that it wasn't supposed to. I do however remember staying awake at my desk for 21 hours while Dave worked his butt off trying to fix the problem. What made this crunch even more important (and therefore stressful...) is that Dan had left for E3 a day earlier than us without so much as a single working copy of the game, so it was up to us in order to get everything in order before we left to join him in LA. Luckily, Dave managed to sort out the issue at around 7am the next day and we boarded the plane to LA with a well put together demo of our game. Looking back, I can always manage to laugh about the situation, but at the time, it was insanely stressful and scary for all of us. Unfortunately, yet again all of the hard work that we put into the game before heading to E3 didn't pay off for us as we were still unable to attract a publisher for Chase,

With yet another trade show gone past us where we were unable to sign a deal for Chase, we decided that another overhaul was needed for the game in order to get everything to the level where we would be able to have publishers interested in signing us. We yet again threw out all of our worlds (I believe that there were only three at this time) and went about redesigning them to be even better looking than we had in done up to now. We also decided that we needed to make the game more than just A-to-B driving, and ended up coming up with a challenging yet rewarding system for the gameplay to be based around.

At the end of the day, our desire was to make the game fun to play while limiting the frustration factor as much as possible. We took to heart a lot of what we learned from Noah Falstein's time with us and decided to focus on rewarding the player while at the same time removing as many forms of punishment as possible. We achieved this by making each level in the game have a certain number of objectives to complete, where each objective was worth a certain number of points in the player's career. Once an objective was completed in a particular movie scene, it would stay completed and the player would re-

ceive the points for it. A player could then replay the level, doing another "take" for the movie scene and focus on completing objectives that he wasn't able to during the previous attempt. This system allowed us to make it that most levels actually required the player to do multiple takes if they wished to complete all of the objectives, as it just wasn't possible for them to complete each and every objective in a single take.

The points that the player received were necessary in order to "unlock" the next scenes in each movie, and ultimately the next movie in the player's career. This meant that the player could play a level and complete whatever objectives he was able to, then as long as they had earned enough points based on the completed objectives, they would be able to progress to the next scene in the movie. This enabled us to lower the frustration level for players who weren't necessarily "hardcore" gamers and possibly wouldn't be skillful enough to complete all objectives for each and every level. However, a player may only complete the bare minimum number objectives necessary to progress for a while and then reach a scene or movie that they wouldn't have enough points to unlock. In these cases, the player could



backtrack to a previous scene and attempt to complete any objectives that still remained in order to earn enough points to unlock the next scene or movie. In most cases, players would have increased their skill level with the game compared to what they had when attempting to complete earlier scenes, so objectives in these scenes that may have been too difficult earlier on, would now be within their ability to complete.

As mentioned, not only did this system reduce the amount of frustration in our game for novice to intermediate players, it provided players with a longer gaming experience without feeling punished for failure, as in each take they could focus on achieving an objective or objectives that they felt were within their ability. This also had the beneficial side effect of allowing our small development team to only have to build a relatively small number of levels for the game, and then reusing some of them to stage more than a single movie scene.

On top of creating better looking graphics and designing a more fun and interactive game-play system, we also felt that in order to make players feel as though they were involved in a Hollywood action movie that we needed to have a lot more action happening around them. In order to achieve this, we went about making the game environments as interactive as we possibly could. The first part of doing this involved creating a more advanced vehicle damage system that had per-vertex damage, which allowed us to blend both vertex and textures between normal and damaged versions of each vehicle in the game. We expanded this system to allow us to have parts of the vehicle detach and become interactive parts of the world once they had sustained enough damage. This meant that bonnets, doors, and even roofs would eventually break off of the vehicles once they had been sufficiently battered.

The second part of creating a more interactive world was for us to make as much of the environment as possible to be destroyable in some way or another. We were able to have pretty much every object in the game that could be hit except for buildings (ie. fire hydrants,



lamp posts, telephone booths, etc.) damage-able or destroyable in when interacted with.

This meant that all of these items either swapped to damaged versions of themselves, or broke up into multiple pieces, each one with their own physics (ie. a telephone booth would break into it's four sides plus the roof, a wooden fence section would break up into individual planks of wood, etc.). And on top of that, some objects would even have their own added effects (ie. a fire hydrant would leave behind a shooting jet of water that would apply a force to any objects that hit it, including the player's vehicle).

While we were busy implementing all of these changes, we were contacted pretty much out of the blue by a producer from a company called BAM!. They were relatively new to the world of game publishing and they were looking at adding titles to their Xbox lineup. The producer's name was Joe Booth and he liked what he saw in Chase from our website, so he wanted to come down to South African and visit us in order to see how our studio was set up. So some time in August 2001 I believe, Joe made his way to the I-Imagine offices where he was more than impressed at what he saw. Not only was he happy with the direction that the game was taking and how it looked in person, he was even

more impressed with the team that we had put together and the tools and technology that we had developed as the backbone of the company.

In the next week or so after Joe had returned to the UK, we were already in contact with them in regards to their interest in publishing Chase. We began to negotiate with them via phone and email, and then headed to ECTS in London at the start of September in order to work on the finer details of the publishing deal. As the deal wasn't 100% guaranteed yet, we also met with some other publishers while we were in London, but all of our focus and energy was really spent on getting the deal with BAM! done. By the time that we left ECTS, which was towards the end of the first week of September 2001, we were quite confident that the deal would be finalised in the next few weeks or so. However, only a few days after we got back to South Africa, the unthinkable happened. 9/11.

As everyone knows, the financial climate immediate after the events on September 11, 2001 was not very stable. There were big drops in the stocks of virtually all companies due to worry about the state of world affairs, and BAM! was not an exception. Due to the stock price of BAM! suffering as it did, they unfortunately had to pull out of negotiations with us for the time being as they just didn't have the capital in order to publish Chase available to them anymore. As you can imagine, we were quite upset about this turn of events, but luckily towards the end of the year, the stock markets stabilised and BAM! was once again in a position to publish Chase, so Joe flew back down to South Africa after Christmas in order to meet with us and our board of directors in order to finalise the deal.

Once the deal was completed, we were assigned a Producer by the name of Barry. At first, Barry worked with us remotely in order to get the production schedule finalised and to work on the final direction that the game would take. A few months into the schedule, he came down to our offices and stayed with us for a couple of weeks while he tested the game and we all worked together on ideas for finalising the rest of the game levels. Once this was finished however, we realised that we were behind on our initial time projections, and that we would not be able to make BAM!'s deadline for having the game to market. Unfortunately, this date could not change as they had already announced their financial projections for the quarter in which the game was due to be released, and they did not want to cause unrest with their shareholders. So, Barry came back down to South Africa a short time later and spent the next few months crunching with us in order to not only have the rest of the game built, but also gameplay balanced and bug tested so that we would be back on track again. This unfortunately meant that we had to drop 1 scene from each of our 4 movies, so we ended up with 16 different movie scenes instead of our intended 20.

Even with the change of schedule, we managed to complete the game around the start of August 2002, at which point it went into the TRC (Technical Requirement Certification) process at Microsoft. It failed at least once, possibly twice, but it was finally approved for gold status around the middle of September, and was available in stores in North American and Europe in the last week of September 2002. Once reviews for Chase: Hollywood Stunt Driver (as it was now known) started to come in, we were sur-

prised at how mixed they were. There seemed to be just about as many people scoring us at 80% and higher as there were scoring us at 50% and lower. Ultimately, I think that this was probably due to the pick-up-and-play design that we had for the game. A lot of people found the game very fun, compelling, and addictive to play, while a whole range of other people found the game to be too easy, bland, and repetitive.

After all of the reviews were finally done, we ended up averaging around 65% or so if you look at Metacritic (<http://www.metacritic.com/games/platforms/xbx/chasehollywoodstuntdriver?q=c>) and GameRankings (<http://www.gamerankings.com/htmlpages2/518048.asp>). While we weren't very happy with the lower scores, we were quite proud of being able to make a game that a lot of people found to be fun to play, which after all is primarily what playing games should be about. As for how well the game sold, the last sales figures that I was made aware of indicated that about 175 000 units had been sold worldwide. This isn't a tremendous amount at all, but we were pretty happy with it considering that it was our first title, that BAM! didn't spend very much money on marketing it, and that it was available on only a single platform.

Overall, we were pretty happy with how things turned out with Chase: Hollywood Stunt Driver. We made a lot of mistakes and learned a lot of good lessons. We lost a lot of good, experienced employees, but we were able to find many talented South Africans who were ready to jump in and take their place. In addition to the previous South Africans who were hired, during the final year or so of our development of Chase we managed to hire a whole slew of additional 3D artists with Steven, Louis, Corlen, Gary, and Lee. Around the start of 2001 we also began to put together a team to focus on developing titles for the Gameboy Advance. To achieve this we hired an additional programmer named Digo to work with Derek, and also hired Michael and Dominique as 2D artists to work alongside Jeanine in developing the art for the game, who's story will have to wait until another time.



COOLHAND

IN CASUAL WE TRUST

A look into the casual gaming market...

Put yourself in this situation: you're an enthusiastic gamer, you've played your fair share of the latest releases, you humbly proclaim yourself supreme overlord of Counter-Strike, and you regularly peruse your copy of the latest 'zines. For all intensive purposes, some may call you a hardcore gamer.

One day, you stumble into your friend's room and find him sitting in the corner with a Gameboy Advance, hunched over it like a guilty teenager caught reading Playboy - except the hunch isn't inspired by guilt. In fact, your friend hasn't even noticed you entering the room. He's completely absorbed with the little machine's electronic antics. You begin to wonder what the heck is going on. After all, this friend is not a gamer. Not by a long shot. He doesn't even know how to play Quake. In fact, you'd be surprised if he even knew that the game existed. Yet there he's sitting, enthralled by a Gameboy.

Surprising, but cute. The poor bugger's trying to dabble in a world he has no idea about! At this point, a strange paternal instinct kicks in (either that, or a subconscious urge to go laugh at him), so you go to see how he's doing with his foray into gaming. Then comes the next surprise.

"Oh, this old thing? I've had it for ages, play it every day."

Turns out he loves Gameboy. He also shows you his collection of bookmarked web games, puzzle titles and miscellaneous gaming doo-hickeys. All this time, he's been gaming even more than you had, and you never had a clue.



You've sat with your AAA titles and large-scale LAN events for all that time without a hint of what lay beyond your own small gaming world.

Yes. Small.

Ladies and gentlemen, the truth is that gaming as we know it is only the tip of the iceberg. Sure, the big titles out there receive the press attention, the rave reviews and the competitive events, but in the eyes of the industry, our world is but an afterthought - a sugar-coated entity which garners attention but is ultimately shallow. Under the radar, an estimated 60 million people are heavily involved in the world of casual gaming, an industry which is already worth about \$350 million.

These numbers are set to increase radically in the next few years as gaming becomes even more accessible and appealing to the mainstream. And game developers need to cotton on to this idea.

Why casual gaming?

More realistic scope

As much as most new or hobbyist developers are hoping to create "the next big thing", the casual gaming market is not only far more accessible, but a far more logical target. In bygone eras of game development, a one-man team could go ahead and code a masterpiece - nowadays, the industry is dominated by ... well, the industry, which leads most indie developers to believe that the noose is tightening around their necks. Yet a quick look on the Internet will provide an alternative - a ripe playground free of hundred-strong teams and gigantic budgets. Developers, your niche calls!

Focus

A common trait of casual games (particularly that quick "puzzle" brand of games that people can play in 5-minute sessions) is that they rarely fuss with such concepts as storyline, mid- to long-term user goals, variation or continuity.

Yes, there is the consideration of replayability value, but when your aim is to keep somebody occupied for five minutes instead of half an hour, your task becomes a lot easier in many respects. A good game designer with a solid and fun concept can go about putting it into action without being burdened with most of the worries that plague most "large" game developers.

Portfolio boost

Even if you have your heart set on building the bigger games, a common aspect of your work that potential employers ask about is your portfolio of functional prototypes and completed works. In this respect, the ability to use pixel shaders and create epic storylines is sometimes secondary to the merits of having an extensive showcase of original, bite-sized and fun-to-play games which have received exposure and commendation. Not only is casual gaming conducive to quick and easy prototyping, but it's far easier to finish a game with few resources demands beyond the central gameplay dynamic.

Distribution opportunities

Continuing on the previous idea, the big WWW is the perfect spot for showcasing casual games. However, not enough people realise that there are different manners in which this showcasing can take place. Flash portals are amongst the

culprits - places like Newgrounds (<http://www.newgrounds.com/>) are great for quickly uploading masterpieces that can be accessed by a lot of interested people. Taking it one step further, sites such as The Great Games Experiment (<http://www.greatgamesexperiment.com/>) aim themselves at linking gamers and game developers. From the moment you load up the front page, you have access to a wide variety of blogs, flash games, game downloads and many other handy resources that are ready and waiting for any interested parties.

Even the big guns are doing it

Biggest-publisher-ever-ever (known to most mere mortals as EA) recently announced that it was going to split itself into four development divisions, one of them committed solely to casual games. Ubisoft is also hopping onto the casual games bandwagon, putting its weight behind a game called "Word Coach". Konami is doing something similar with a beauty care game on the DS console. Even episodic gaming (pushed by notable developer Telltale Games, amongst others) is a reflection of this more-than-viable market concept.

Half-Life 2 Episodes and the new Sam and Max series of games are testament to the fact that the public have just as much fun with gaming in

more bite-sized chunks. The big boys are getting quite serious about being casual, which further questions the perspective of indie developers who stick their noses up at doing "2-bit titles".

Considerations for casual game design

It's a sprint

As a rule of thumb, casual game sessions tend to be far shorter than those of their triple-A counterparts. A far cry from the hours spent grinding in World of Warcraft, the casual game designer often has to cater for an individual who will sit in front of their computer and play a game in half an hour. Most good game design involves the designer catering for short-term, mid-term and long-term goals. For example: in some generic FPS game, the player may be given a long-term goal of blowing up the enemy space platform. Their mid-term goal is finding a lever to open the door to the main engine room. Their short-term goal is to survive an encounter with the monsters in the next room.

With shorter games, these goals are either compressed or in some cases eliminated, leaving only the short-term gratification to be catered for. Thus, a "long-term" goal may in fact be fulfilled within 30 seconds, or may even be considered as non-existent.

If your game hinges around the short-term, it's all the more necessary for every single one of the player's actions to be justified and interesting. If a player needs to shoot a helicopter, then immediately shoot another helicopter to further boost score, it goes without saying that the action of shooting should be a satisfying one, or at least satisfying enough to keep the player going at it for however long he or she needs to.

Goals should be easily attainable, and if it's technically possible to win a game, it should be easily done in a short amount of time (provided that players know what they're doing).

Some "casual" games can actually get very long and involved, but it may be a good idea to build up a bit development experience before attempting these sort of projects.



Replayability

There's a lot of casual games out there that you can only really play once. Perhaps they're detective stories or linear platformers. On the whole, however, casual games are notable for their replayability - consider Tetris, Bejeweled, or even Solitaire. The whole idea behind these games is that you play them for as long as possible (or win as quickly as possible) then hit them again with renewed gusto. This changes the developer's goals considerably - now, it's no longer a matter of sustaining player interest with feature creep or plot development. There's less concern of getting a player to explore your game fully - chances are, if they put it down after five minutes, they've still gone through it.

Instead, you need to be able to offer an experience that a player will want to repeat. Variety and true replayability are very important - at the most basic level, you're going to need an element of randomisation in your game so that no two experiences are the same. Additionally, it's important to consider the "easy to learn, impossible to master" trick - online leaderboards and score accolades are a great way to challenge a player even after they've made it through the game the first time.

Exploration

Casual gaming is a brilliant arena for checking out new avenues in gaming, seeing what works and what doesn't. Most players will try

something new at least once, provided it's easy to get into and brings rewards timeously. Concepts that would be too difficult or unsuitable to implement in full-length games are perfect for the casual environment. Want to create a game centred around the player's ability to randomly turn into a sandwich (with all the masticatory superpowers that being a sandwich would naturally grant you)? That's great! Instead of sitting down and puzzling out 30 stages around the concept, burning your brain out and eventually realising that such an idea is simply not going to work in the long term, you can put your hero in a single arena with a bunch of constantly spawning enemies and work on designing a single good level that simultaneously incorporates all the design goodies that you were trying to stretch out over the long term.

Conclusion

Casual gaming is quickly gaining ground, and not enough indie developers are latching on to the idea that it's actually a viable plan to base your game design career on a title that gives as little as five minutes of joy to any given player.

Too short? Not "big" enough? If these are really our complaints when it comes to the idea of games that are by now "beneath" our development skills, perhaps we need to shed some collective pride and realise that the player on the other end is going to thank us far more for a few minutes of awesomeness than any amount of wasted time blundering through a tech demo that reeks of incompleteness and an overextended developer whose good intentions were marred by some very real constraints. A lot of game developers advise newbies to start small. My own advice is to keep small.

NANDREW



CREATE. DEVELOP. EXPERIENCE.....**ONLINE**



DEV.MAG

CREATE • DEVELOP • EXPERIENCE



www.devmag.org.za