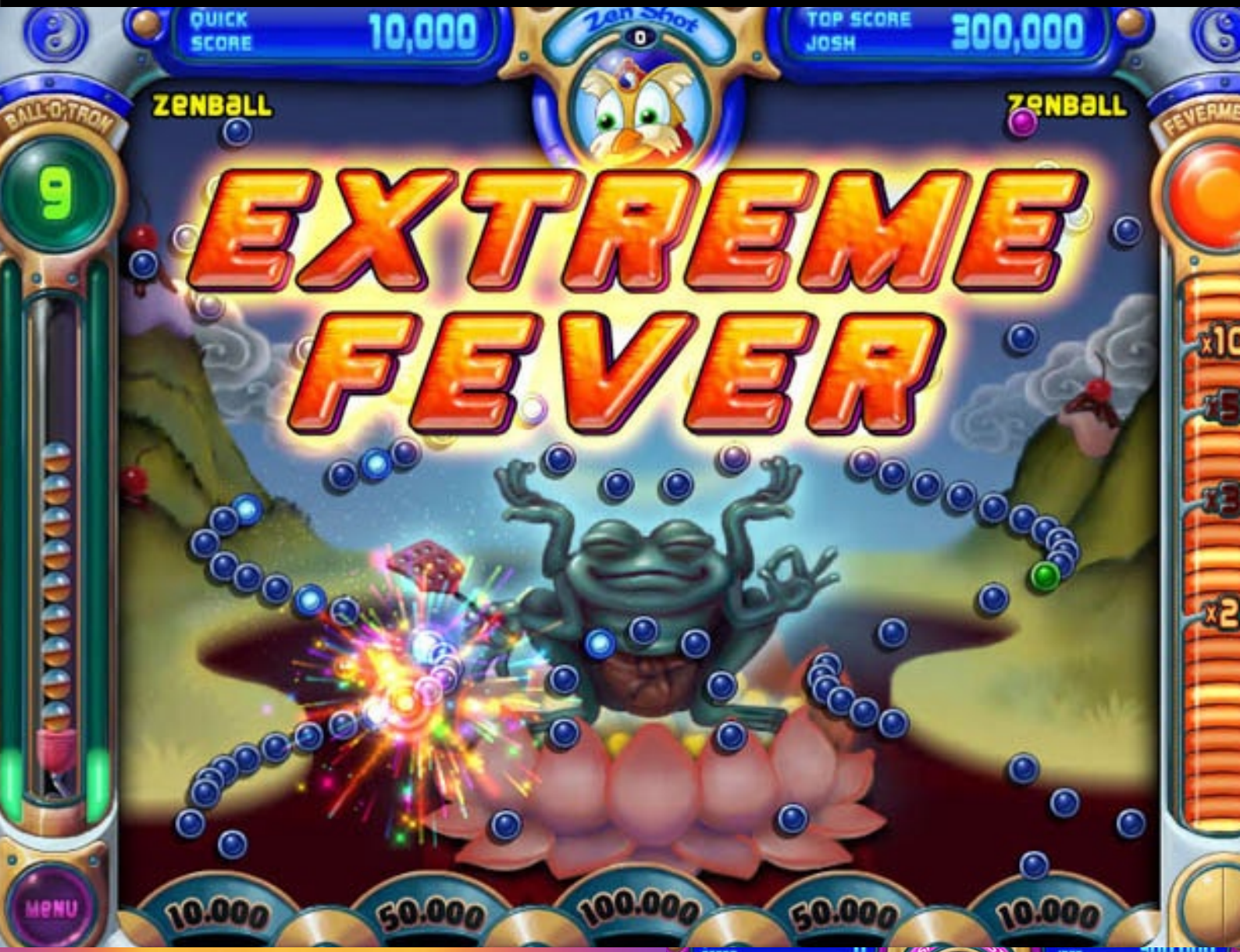
SOUTH AFRICA'S PREMIER GAME DEVELOPMENT MAGAZINE

FEBRUARY 2007

DEV.MAG

CREATE • DEVELOP • EXPERIENCE



REGULARS

FEATURE

REVIEW

DESIGN

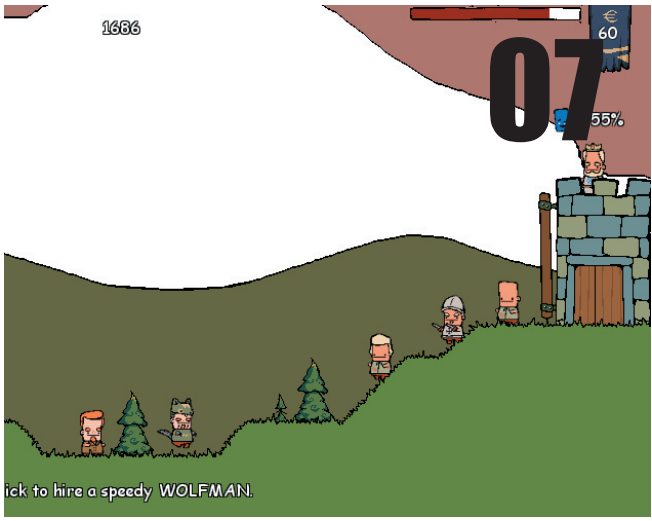
PROJECTS

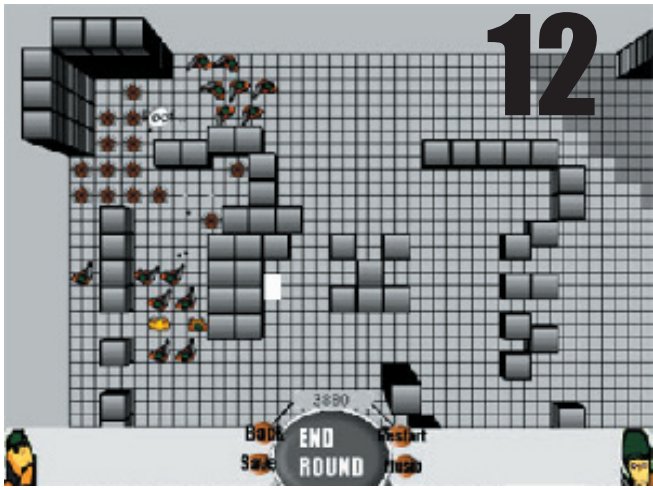
TECH

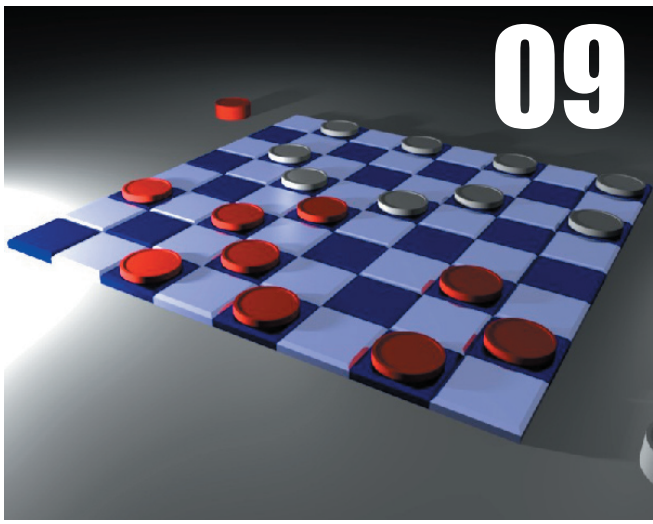
HISTORY

MOBILE

TAIL PIECE







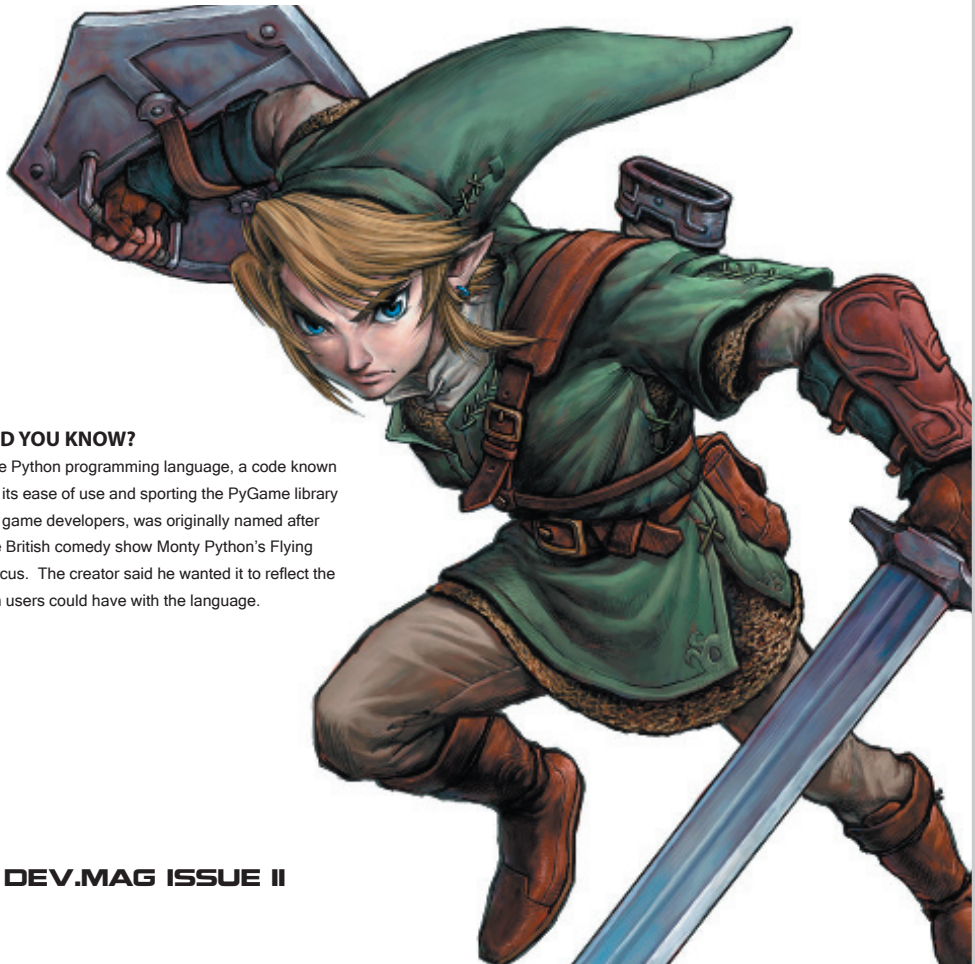
Stop the press, we're back! It's not been an entirely lovely experience, trying to get things sorted out this month, but we promise that we'll do our best to finally catch up on our schedule come next issue. That, and we'll give you candy. Lots of it. In fact, several key members of our staff have a truck of it hidden behind your house, so before you get angry at us, just remember that little truck. Is everything good now? Great, glad to hear it!

But seriously, real world commitments are a hassle right now, and it always seems that at the moments when we speed up the most, the world just gets the slowed-down afterthought. The Game.Dev organisation has attended a bunch of meetings (which we're not allowed to tell you about), discussed a bunch of business plans (which we're not allowed to reveal to you) and have been making plans for a whole bunch of activities (which ... well, we'll tell you all about those in good time). So the bitter, bitter irony is that there's really not that much we can report on, especially where the juicier stuff is concerned. Hopefully, we get the green light soon and tell you all about it.

Otherwise ... March. People have established schedules for the year, have their engines fully wound up and are looking forward to upcoming public holidays (one or two of which should be poking their heads out in the near future). As ever, that makes the staff at Dev.Mag get caught up in real-world commitments, some of which can be a real pain, but others which can turn out very, very rewarding in a game development sense.

Whoops, was I about to tell you what I meant just there? Ahhh, sorry, but that's another story that'll just have to hit next month's pages. Think about candy in the meantime, and look at all the other cool goodies we've got for you this time around!

Editor
Rodain " Nandrew " Joubert



DID YOU KNOW?
The Python programming language, a code known for its ease of use and sporting the PyGame library for game developers, was originally named after the British comedy show Monty Python's Flying Circus. The creator said he wanted it to reflect the fun users could have with the language.

DEVMAG
CREATE DEVELOP EXPERIENCE

EDITOR
Rodain " Nandrew " Joubert

DEPUTY EDITOR
Claudio " Chippit " de Sa

SUB EDITOR
Tarryn "Azimuth" van der Byl

DESIGNER
Brandon " Cyberninja " Rajkumar

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

CARTOONIST
Paul " Higushi " Myburgh

WRITERS
Simon " Tr00jg " de la Rouviere
Ricky " Insomniac " Abell
William " Cairnswm" Cairns
Bernard " Mushi Mushi " Boshoff
Danny " Dislekcia " Day
Andre " Fengol " Odendaal
Heinrich " Himmler " Rall
Matt " Flint " Benic
Luke " Coolhand " Lamothe

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: **forums.tidemedia.co.za**.

Games and the arts

http://www.gamasutra.com/php-bin/news_index.php?story=13171

The age-old question ... are games art? This Gamasutra feature brings the voices of names such as Peter Molyneux and Tim Schafer to the limelight, looking at their opinions of the 'art' status in games, how important that status is to the creators and how important it is compared to other standard gaming goals. It brings up an important question -- after all, does a painter ultimately decide to make art or paint his picture?

Free texture pack released

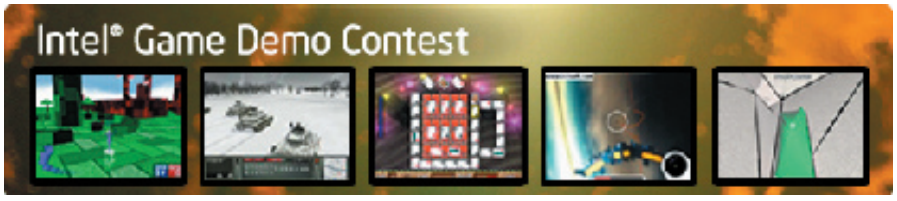
http://www.spiralgraphics.biz/packs/

Free stuff is always great! That's why the release of the third Spiral Graphics texture pack is being met with such enthusiasm. Bringing the collection to 450 seamless textures, these little gems are freely downloadable and devoid of royalty costs. Moreover, they're fully editable in the texture editing software Genetica, which is also featured on the site. Have a look, download the textures and try a free demo of the associated software.

Intel hosts "multi-core" game development competition

http://www.gamedev.net/community/forums/topic.asp?topic_id=439588

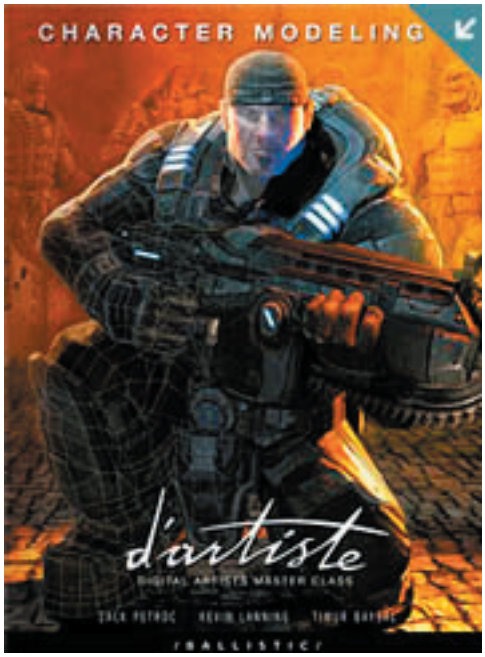
Gamedev.net reports a six-month contest being held by Intel, designed to test the power of multi-core processors and laptop form factors through -- you guessed it -- the medium of game demos. Prizes for the competition, stated by Intel, include goodies such as one-year IGDA memberships, gaming machines worth $3000, copies of the Torque Advanced engine, bundles of Intel products and lots and lots of money. The competition is divided into two categories: games that show off multi-core capabilities, and games that show off the advancements in laptop gaming. Tempted parties can check at the above site for a link to the competition page.

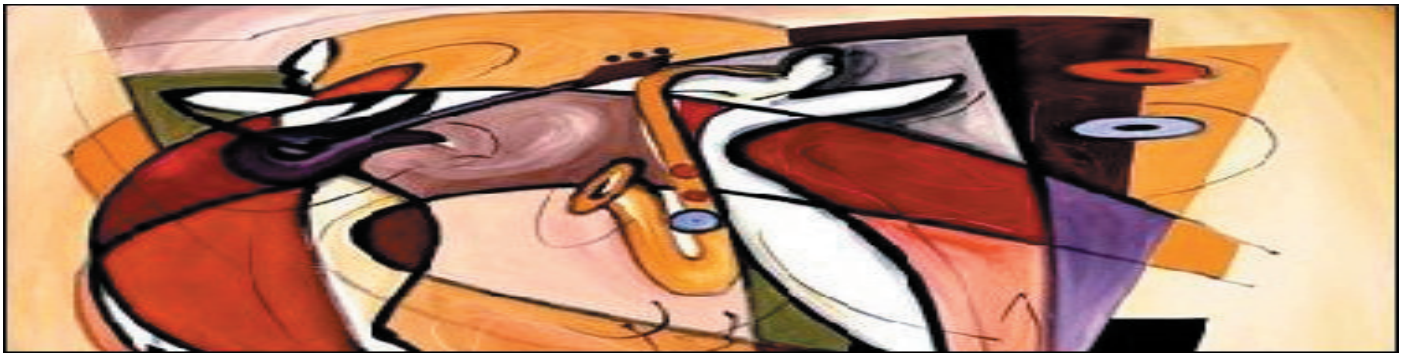


Book on character modeling released

http://www.gamasutra.com/php-bin/news_index.php?story=13172

A new d'artiste game design book, "Character Modeling 2", is now available for those interested in learning the techniques involved in creating and animating characters and creatures for both films and games. The book places particular emphasis on the creation of character models for Epic Games' "Gears of War", and also features techniques from artists such as Timur "Taron" Baysal and Zack Petroc. This is the fifth book in an already strong series, and comes highly recommended by Gamasutra.





Using Abstraction to Develop Better Games

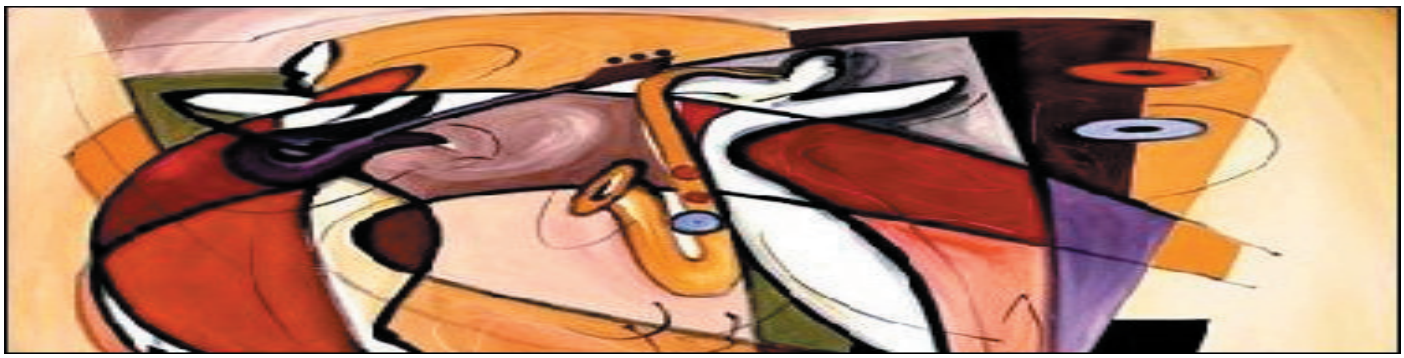
Abstraction is a development method used to hide how something is done while making the functionality available to other developers. Every function, method or class in your code makes use of some level of abstraction. However, with careful design abstraction can be used to build better, more modifiable and easy to configure games. The creation and usage of components is a perfect example of abstraction.

Components are effectively little 'Black boxes' within code that hide everything they do away from the developer. Black boxes are those portions of the code that deliver specific functionality that is needed but where the implementation method is completely unknown. When the functionality in the Black Box is reliable and consistent, such as in the case of commercial components, it is seldom necessary to know how the functionality was developed as it does not need to be maintained.

When Abstraction is implemented correctly, it delivers easy to use components that can be included into the current code base. Very good implementations of abstraction also promote the ability to switch out current components for new components without seriously impacting the code base. For example, when moving between different types of databases, good database components will enable the switching out of one component set for another very easily.

Good components need good design. When a section of code is identified as a possible component it should be extracted from the current code base, turned into a component with clearly understandable interfaces and then reintroduced to the original code base. By completely extracting the relevant code form the code base the new component can be completely independent from the original code. To ensure total independence from the original code small example programs that show only the new components functionality should be developed. These also act as miniature tutorials for other developers that use the new components.

Using components is sometimes a difficult task as the intricacies of the implementation are so abstracted away from the component user. Good documentation that gives a prospective user a detailed description of the functionality contained within the component is critically important to allow users to evaluate the suitability of the component to their requirements. Conponents that have been selected for use are typically expected to work with little or no configuration effort. When you select a component set to use, it's also often a good idea to develop smaller test applications to validate the functionality of the components



against your original need for a component.

Components for game development come in many different forms. These range from simple components for a specific task (e.g. collision detection) to complete game engines that just need content added (e.g. FPS creator). The level of component you use is defined by many different aspects of the development process. These include such things as language, graphics libraries, level of expertise and even the ability of the artist on the project. Needless to say, whatever your level of experience, game components can add significantly to the quality of the end product produced.

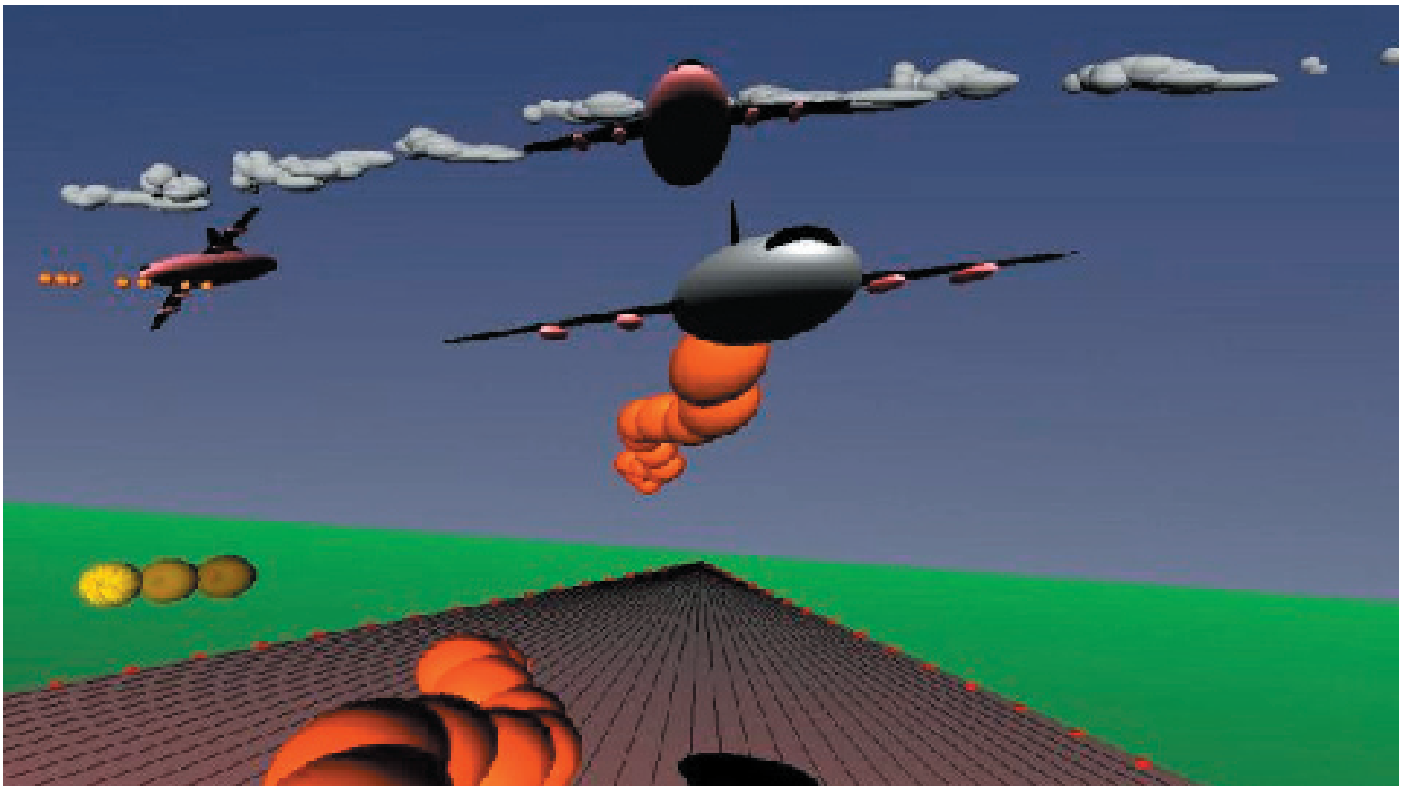
The most used set of game components are wrappers for the standard graphic adapter libraries. The components typically make it easier to interact with the graphics card without an indepth knowledge of the graphics adapter libraries, making it easier for the game developer to work with DirectX or OpenGL, or even abstracting the graphics layer to the point where the developer need not know whether they are working with DirectX or OpenGL but just allows the game to be developed.

Another commonly used set of components allows developers to quickly and easily add sound into their games. Currently there are numerous different sound component libraries available that abstract the complexities of the multitude of different sound standards from the developer.

Many developers make use of the various graphics and sound components within their games without much thought. A much smaller number of game developers have started making use of more specialised libraries for areas

such as AI scripting, physics, collision detection and even character animation. By investigating and evaluating the various component sets available and making use of the better ones in your games, development time can be decreased and the quality of the games increased.

CAIRNSWM



THE BOOMLANDS

Developer:

Shane Heres

Year of creation:

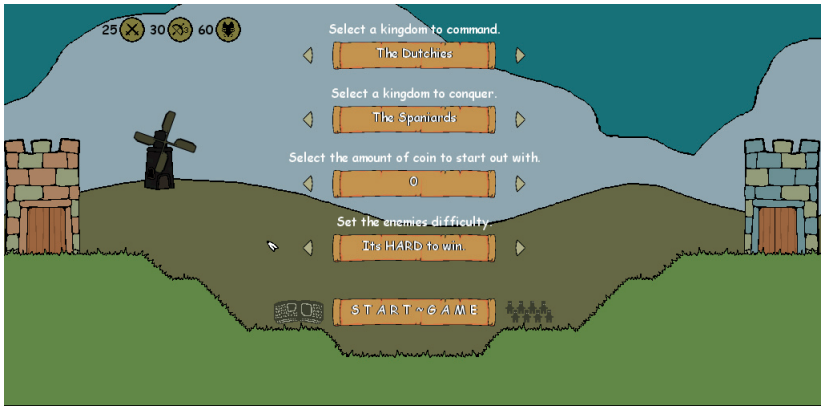
2006

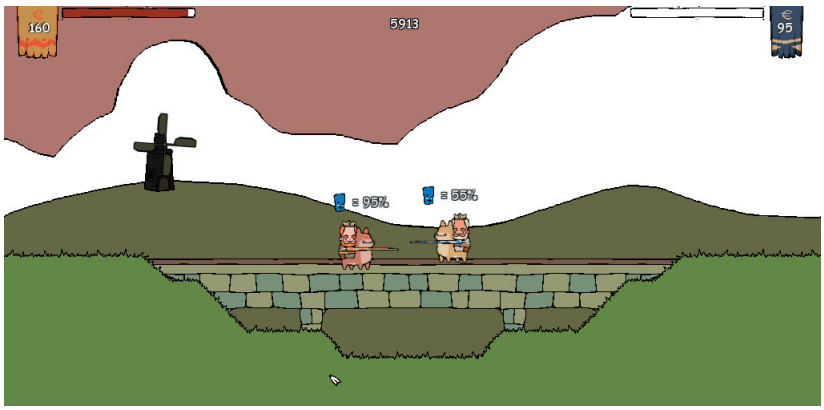
Website:

http://shaneaheres.googlepages.com/

Genre:

Strategy





A great king has just died and leaves his kingdom divided equally into two countries. So naturally one country gets jealous and decides it wants the other half too. How do they settle the dispute? The same way you settle everything in medieval times – with a big battle!

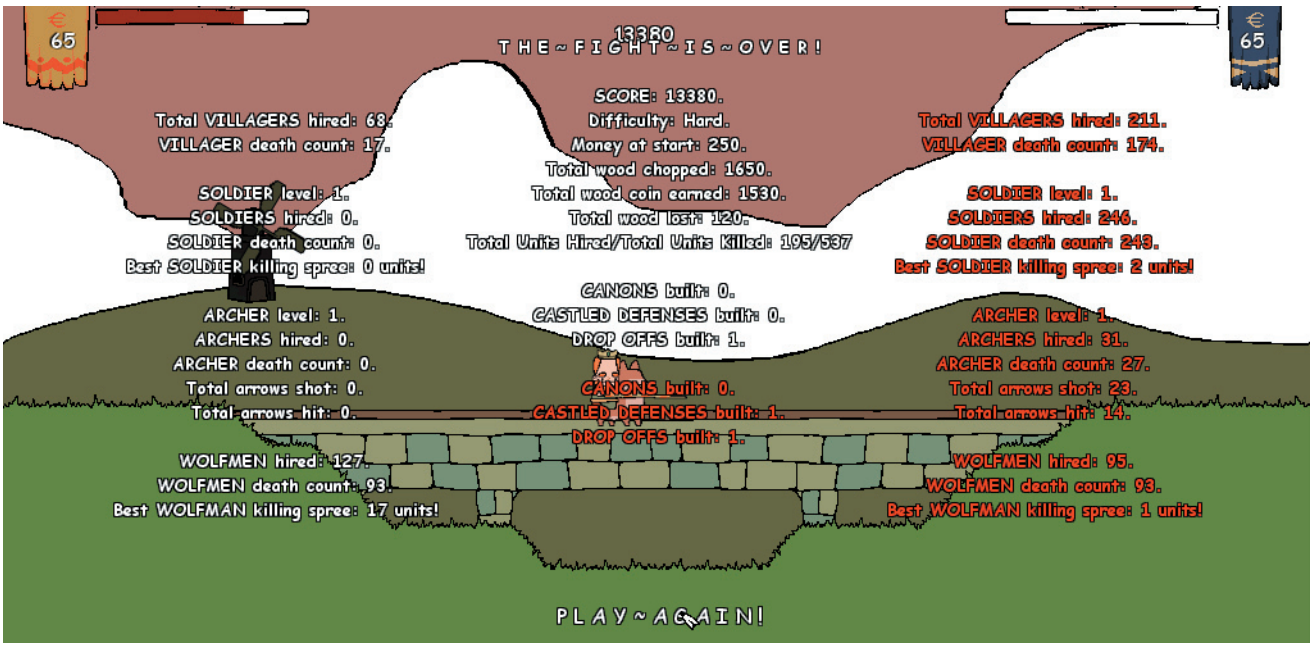
A rather large fight for control of "The Boomlands" ensues. The first thing you notice about this game is its impressive graphics. All the sprites and backgrounds are masterfully done, and the game is very pretty. Nothing suits eye candy better than a unique style. This game has a very cartoon-like feel. The characters bounce around with huge heads and tiny limbs. While everything in the game has some great ambient animation, most people will never even notice. The attention to the small details is amazing and the "earth" colours suit the game nicely.

What's more, "The Boomlands" has possibly the best gameplay I have seen in an indie game. When you start you can choose your country and who you will face. Your choices are either the Dutch, The Spaniards or the Scots. You can also choose your difficulty and the amount of starting gold for the sides. The battle takes place between two castles where the kings stand and survey the scene while making snide remarks. In the middle of the battlefield, trees grow in random spots. Workers can be trained to go and harvest the trees for gold. Gold is the only resource and vital to success. Luckily, the player earns five gold per second aside from lumber harvesting, making life easier.

The gameplay is remarkably simple as there are only seven buttons you can press that affect the game. Soldiers and archers can be trained to defend your workers and attack the enemy. Once you build a unit, it slowly starts moving towards the centre and if it survives will continue on to the enemy castle. Archers stop in the centre and fire arrows, while workers will stop at the nearest tree and carry it back home. Wolfmen are special units that move very quickly and can only be stopped by other Wolfmen or level 2 soldiers -- they do come at a high price though.

The key element here is timing. You have to time your creation of units so that they can protect your workers or kill enemy units before the baddies get there. When an unit dies, it seems to fall off the screen while making some quirky comment, an effect which is very cool. In order to win, your king must win the joust that appears after one of the castles is destroyed. Kings gain and lose stamina according to how his side is doing in the battle. The king with the most stamina at the time of the joust will win. Simple game play leads to some intense battles where diverse strategies and the ability to recognise opportunities will get you far.

The game is perfectly balanced and use of all the units is required for a victory. The AI is also rather pleasantly brilliant: your opponent will use clever tactics, ingenious timing and general craftiness to bring you down. The AI will also



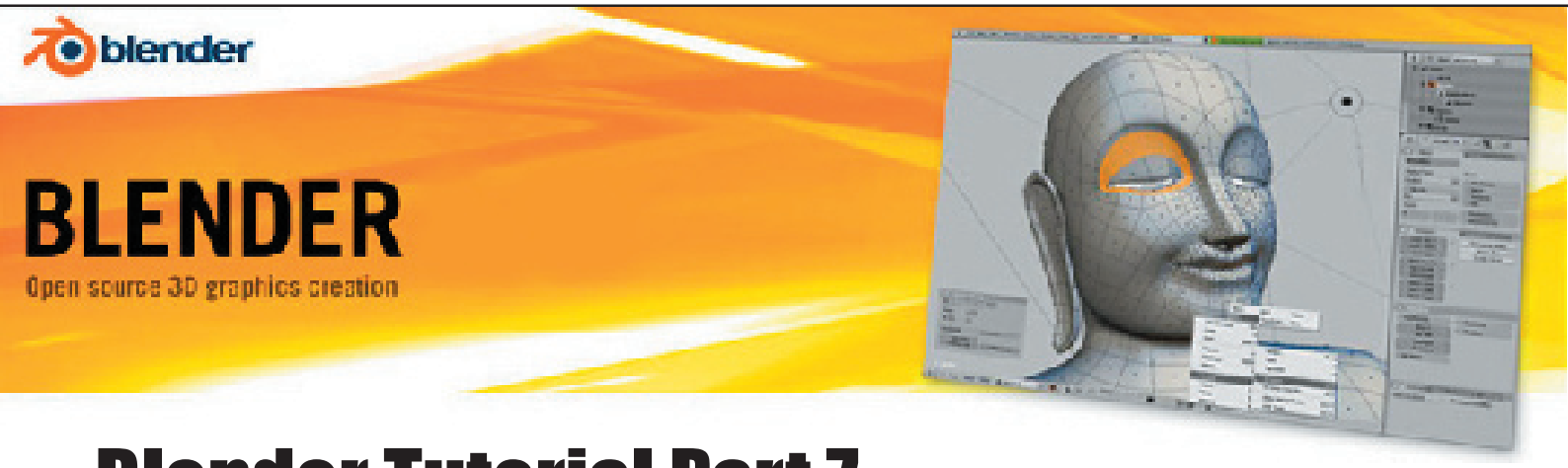
use smarter tactics as you progress through the difficulty levels. A welcome change, as most games just have imbalanced damage ratios.

The different strengths and weaknesses of the units also make for interesting gameplay. For example Wolfmen are fast and deadly but expensive and easily disposed of by Level two soldiers. Yep, soldiers and archers can be upgraded by collecting a large sum of money, something which is risky but can give you a major advantage. Wood depots, shields and cannons can be built. Depots are rather expensive during the beginning stages of the game but it means your workers can deliver the wood quicker. Cannons fire cannon balls at the opposing castle while shields protect your castle from cannonballs and arrows. The gameplay is addictive and very fun. The sound is rather average with no music but relevant sound effects that add to the general feel of the game.

One thing I would like to have seen is multiplayer. This game is perfect for the two people on one pc type multiplayer.

The most impressive thing about the Boomlands is the large amount of polish and attention to detail, the smallest things have animations and special effects. The end of a battle sports a large and informative statistics screen and the game is well documented and technically flawless. I have yet to come across a bug or poorly designed bit of the game. The gameplay is fun and addictive, but be warned this game will suck away the hours as you try to conquer the elusive challenge of winning on hard mode while starting with no gold.

SQUID

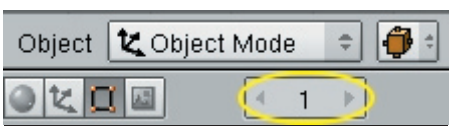


Blender Tutorial Part 7 : Animation in Blender

A great way to make any Blender scene look even more impressive is to make it move. Blender features an array of powerful tools that make anything from simple movement of objects to complex facial animation possible.

For this scene, we'll be working with that checkerboard we made a few issues back. It's got a lot of objects that we can move around and experiment with, and, with a little patience, we could even play out an entire game of draughts. If you don't have it, fear not. The relatively small file is available in the Dev.Mag website's content section. Look for 'Blender Part 5'.

Firstly, to make objects move in Blender, you'll need to understand how Blender handles animation. As with all videos, animations in Blender are composed by numerous still images (frames) played back in rapid succession. You can change the frame you're currently seeing in the 3D view by typing a frame number into the frame selection box, clicking on the arrows alongside it, or using the arrow keys on your keyboard to move in 1 or 10 frame intervals.



The frame selection box.

Doing that now isn't going to have any effect, though. To make an object move, they'll need a starting location, and a destination. You provide these in the form of keyframes. Without keyframes to direct their movement, the checkerboard and the pieces will just stay in the position where they were placed.

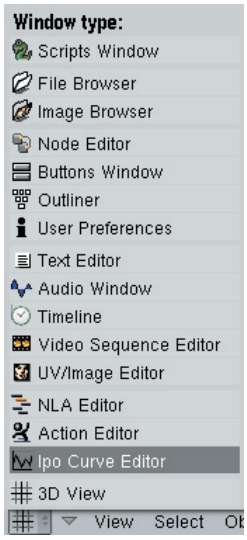
Make sure you're currently on frame 1 before continuing. Select a piece that you'd like to move and hit the I key on your keyboard, or select Insert Keyframe from the Object menu. You'll be presented with a menu to choose with properties of the object you'd like to store in the keyframe. In this tutorial, we'll only be working with the position of the object, so select Loc. If you need to manipulate and animate other properties of an object, such as rotation or scale, look for the corresponding option in this menu.

Choosing which values to store

Advance to frame 20 by typing in the correct value in the frame selection box or by using the arrow keys. Move the object to a new location and create a new Loc keyframe. Now, if you look at the frames between 1 and 20, you'll notice that the piece will now gradually and smoothly move between the two points. Advance to frame 40, move the object to another location and store another keyframe.

You'll notice that Blender automatically smoothes the movement of the object so that it makes no abrupt turns. While this is visually pleasing, it is not always what is required in certain situations. The IPO curve editor will solve all these issues and more. Movement of all objects in a Blender animation is governed by the IPO curve editor. There you can see and edit a curve that will dictate the path your objects will follow in an animation. You can use it to make fine adjustments to the animation.

To see the IPO curve for the selected object, click the grid-like icon at the lower left corner of the 3D view (default layout) and select IPO Curve Editor. You will be presented with graph-like display that governs how your objects actually move. You can navigate this view using the exact same controls you use in the 3D view. In fact, every single window in Blender obeys the same control rules, even the buttons and render windows.



IPO curve editor

Each property you've stored as a keyframe has a corresponding line on the IPO curve. All the properties are listed on the right edge of the IPO window, and you use the list to filter the lines that will be displayed. You can select individual lines and press TAB to edit them. You'll notice that each node is actually comprised of three linked nodes. This is because IPO curves are Bezier curves by default. This is what causes the smoothing of the animation. You can make the curve linear by choosing the appropriate interpolation mode from the Curve menu, or by pressing T.

You can also make a specific point on Bezier curve sharp by 'breaking' a node. With an entire node (or part thereof) selected in edit mode, press H, or select Free from the Handle Type submenu in the Point menu. This will allow you to edit the individual pieces of the node without affecting the rest of it, and it can be used to create sharp turns or abrupt movements.

Pressing K in the IPO window, or selecting Show Keys from the View menu, will display vertical lines that signify all your keyframes as a whole. You can manipulate the keyframe lines as well as the individual nodes to edit an IPO curve.

At the moment, your piece will simply slide laterally to its next location. We'll add an extra little touch by making the piece hop. Vertical movement is movement on the Z-axis, and is therefore controlled by the LocZ line. Select it, and enter edit mode. Select the Z nodes, one at a time, break them - to make the movement sharp - and move the corresponding section of the node upwards.

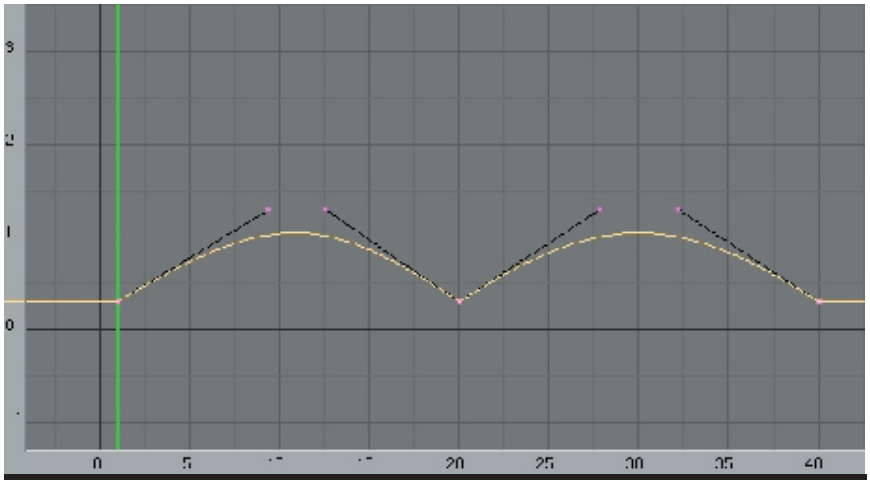
If you made the piece do two consecutive movements, your curve could look like the image below when you're done. If you only made one movement, then you will only need one 'bump' on the line.

Finally, note that all animation previews or renders will be bound within a start and end frame. The default is 1-250, but you can change them if your animation is shorter or longer. All animation related options are in the Scene buttons window (F10), and are highlighted in the image below. Be warned that rendering animations takes a long time, so if you'd like to render the entire thing, perhaps it would be wise to disable or lower your anti-aliasing (OSA) settings, reduce render size by a percentage value or by simply entering different x and y sizes. Once your animation is rendered, you can view by clicking the play button.

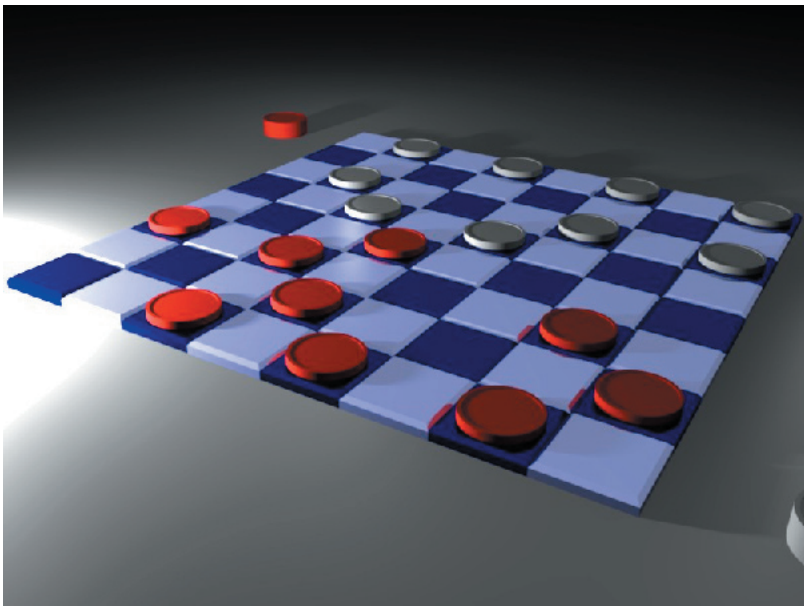
You can also preview your animation in the 3D view or IPO window by pressing ALT+A. The animation will play in real-time, and is also bound by the start and end frames.

I've made an example file, available on the Dev.Mag site as usual, that displays all the techniques learnt in the article, and used in practice to play out a little draughts game.

CHIPPIT



Our LocZ line, to make the piece hop.



```
<LINK REL=stylesheet TYPE="text/css
<META HTTP-EQUIV="Content-Type" CO
<META NAME="GENERATOR" CONTENT="Mo
<META NAME="DESCRIPTION" CONTENT="
<META NAME="KEYWORDS" CONTENT="jav
<SCRIPT language="JavaScript">
```

HTML FOR NOOBS - PART 1: Tags and formatting

While not strictly game development related, HTML is a good skill for a game developer to have, and its remarkably easy! For those of you that don't know what I'm talking about, HTML is the language used to make web pages. You can right click on any website and click "view page source" to see the HTML code behind it.

The first concept of HTML is that of tags. Everything in HTML is composed of tags. Tags always have an open tag and a close tag with things in between. An open tag looks like this <> and a close tag looks like this </>. Different tags to different things.

The first tag we are going to learn is the HTML tag. Open notepad. Type <HTML> and </HTML> on the next line. All our code will go between the two. An HTML page has two main sections. The Head section contains various things describing the page and the Body section contains all our actual code. For now we are just going to use the Body. So in your HTML tags add <HEAD>, </HEAD>, <BODY> and </BODY>. Now in your Body tags add the text "Hello World". Save your text file with the extension ".html". Now open it in your web browser and take a look!

Now we are going to add tag parameters, in your Body tag add BGCOLOR="#000099" after the word "BODY" but before the '>'. Now extend the body tag to contain the parameter TEXT="#FFFFFF". You can now change the colours of your page! Note that all the colours are RGB hex values. I think its time to add some basic formatting.

To start a paragraph with formatting add the paragraph tags and . Everything inside those tags will be affected by their parameters. So move your text into them and add the paragraph parameter ALIGN="CENTER" The same parameter will work for ALIGN="LEFT" and ALIGN="RIGHT". If at any time you need to check how your page looks just save the file and refresh your browser.

Now lets do some font formatting with the <FONT> tag. So add one around your text with the following parameters FACE="Comic Sans MS" and SIZE="5". This will change your font size and style, you can also change the colour individual bits of text with COLOR= in you paragraph tag. The tags <B>, <I> and <U> can be used for bold, italic and underlined text.

All sites need a bit of text but if we add any more it will appear on the same line as our heading. The answer? A linebreak tag (
) this is one of the few tags that does not have a close tag. Adding one will mean your text will appear on the next line. Adding two will mean you will skip a line. So now outside of your paragraph tag put two
 tags and then the text "This is my first web page!"

Now you have a basic web page with text all over it! Look forward to the next issues where we shall cover images, links and tables.

SQUID

What your code should look like:

```
<HTML>
   <HEAD>
   </HEAD>
   <BODY BGCOLOR="#000099"
    TEXT="#FFFFFF">

        <p ALIGN=CENTER>
        <FONT FACE='Comic Sans MS'
         SIZE='5'>
        Hello World
        </p>

        <BR>
        <BR>

        This is my first web page!

   </BODY>
</HTML>
```

ROACH TOASTER 2: PICKING OUT THE BUGS PART 2

"Dear diary,

After the legendary Hoover sucked up the brood mother in the fast food joint, we thought it was all over. By jove, my dear diary, we were totally wrong.

The brood mother began to breed inside the Hoover. Of course, us men never clean our home, so we stored the Hoover in an old closet.

And now... now that we are back from our well deserved holiday in Cancune, we found it missing! It wasn't long before we received the terrible news! They did not only take over the fast food corp, but by the powers that be, they took over the whole "Big City"!"
-Captain's log 1337

There you have it, the utterly cheesy intro to Roach Toaster 2: Big City. I have surprising things in store for the Roach Toaster storyline.

One of the main dislikes people had about RT1 was the graphics... While I am a semi-drawing wiz, I am not the expert at spriting or "drawing" with my mouse.

With this in mind, I mused on how to make "like-able" graphics without a sprite artiste. I first tried my hand at making my first 3D graphics. After about a week of meddling with 3D functions in Game Maker, I finally created a nice looking 3D world for Roach Toaster 2. It looked stylish enough...

After some playtesting I realized that it just did not work. Apart from the "horrible" gradients that burned people's eyes, gameplay wise it meant you had to scroll around the level and your view was sometimes blocked.

So, I inevitably scrapped the "3D"-look. I returned to Roach Toaster 1's roots. Although it was still ugly, it was kinder on the player's eyes.

I did not want to, but I seriously had to get some "expert" sprites to really make Roach Toaster 2 shine. Getting a sprite artist would also save me countless hours of time which in return I could spend on development.

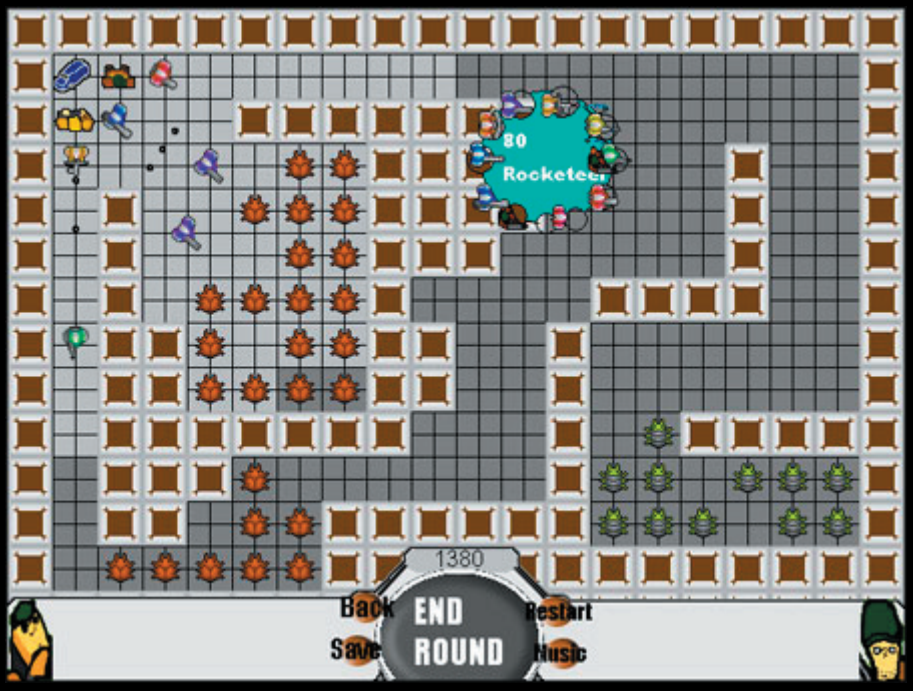
So, I managed to bribe a fellow Game.Dev member to be the sprite artist with promises of pink fluffy bunnies, a bottle opener and an all expenses paid trip to Zimbabwe.

The latest picture is below. Note that some sprites are still the old ones.

As you can see, the whole interative process of Roach Toaster 2's graphics is quite interesting!

For more info on Roach Toaster 2: Big City, head on to http://www.shotbeakgames.za.net

TROOJG



COPY PROTECTION FROM THE INDIE PERSPECTIVE

There's always one burning question in the mind of every aspiring indie developer that pops up once whatever game they're working on approaches some form of completion. A question that sums up the indie lifestyle in one neat little package. A question that everyone has to deal with at some point... "Hang on. If I'm going to charge people for this, how am I going to stop bastards from simply copying it?"

The Lofty Goal

An ideal copy protection system has a simple aim: Wrangle it so that every copy of your game has been suitably paid for. This ensures that you, as the developer, can afford to pay for inconsequential things like rent and food whilst slaving away at your next game, until eventually you're paying the rent on a three-storey mansion and swimming in caviar for breakfast every morning.

The problem comes in when other people try to play your game, your hard work, for free. The supposed reasons behind this want of theirs is beyond the scope of this article, but just accept that they do. Now, imagine that you release a game that's copy protected and it becomes popular. Suddenly you have hundreds (if not thousands) of people trying to get your game for free. If a reasonable percentage of those people are actively cracking the game's protection, there's absolutely no way that you are going to be able to keep up with security as a single developer.

So It Doesn't Work?

There are many, many forms of copy protection. The key to understanding them all is that they're all capable of being hacked in some way. As soon as you're trying to ensure security on someone else's machine, you're asking for trouble. CD-key unlock systems, encryption/obfuscation systems, hardware-lock systems and even dongle systems all break down on one key point: They're running on a computer you don't have control over. Anyone can do absolutely anything to your code (and some will) to get it working without having to pay.

So that leaves only one alternative: Online verification systems. Except there are problems here as well... Firstly, there's nothing stopping a good hacker from setting up a fake proxy that just sends back the correctly formatted messages (and then making this hack available to other people). Secondly, your typical paying customer is going to be freaked out if their shiny new single player game makes their firewall pop up every time they play it.

The Wrong People

That's exactly the problem of copy protection: The wrong people end up being the ones that suffer. Be it paying users having to deal with draconian anti-piracy measures, or developers going completely insane after spending months building a 100% secure system that only lasted 3 days in the wild. Pirates don't get inconvenienced nearly as much, plus if something's a little too much hassle, they just go and play another game they didn't pay for...

So, what do we, as developers, do about this? The consensus seems to be not to worry too much about copy protection, as it's a waste of time above a certain level of sophistication. By all means, have a CD-key system in place to keep casual copiers honest (the kinds of people who'll simply copy a CD to give to their friends), but don't waste time trying to defeat a bunch of faceless hackers that can think in memory adresses.

For that matter, don't waste time building your own copy protection system anyway. Just buy one for your game and use the time you've saved to start on your next game earlier. There are numerous cheap systems available out there, but make sure they're nothing like Starforce. You don't want to chase people away from your games!

In the end, the only really secure way to do anything is via an online system, but even then you can't simply tack one into a game that doesn't need the internet for single-player mode. If you can find a way to provide value over the net, like releasing new levels only to registered paid email addresses, or hosting high-scores for valid players, that's good. But be aware that you're not going to be able to protect your game 100%. Roll with the punches and appreciate the people who enjoy your work enough to pay you for it, you'll be much happier and more successful that way.

DISLEKCIA

CODING ETIQUETTE: COMMENTING

There has probably been a fair bit of debate in the world of computer science over what singular concept or component of computer programming is the most important or powerful. Perennial favourites are most likely to be Encapsulation, Pointers and Addressing, Recursion, and Object Orientation. However, there is one feature of computer programming that should be an integral part of this discussion, but more often than not, isn't. This is likely due to the fact that it's just not as "sexy" as those other ones, even though it serves a critical functionality and plays a role in the day-to-day lives of programmers that is at least as important as every other aspect of computer programming.

This feature is **commenting**.

The proper use of adding comments to your code is probably the single most important skill that a programmer can have, especially when working with other programmers on large projects. It is also the easiest of all computer programming skills to learn. However, far too many programmers ignore it as merely a waste of time. They usually convince themselves that, since they know what they are doing in their code, there is no need for them to waste 10 seconds here and 30 seconds there writing small messages and descriptions about what is actually happening or what their intent is.

Unfortunately, this laziness becomes a serious detriment to whatever team of programmers they are working with, as countless man-hours are often lost by other programmers who have to work through and understand uncommented code – especially if the original programmer is now longer on the team. As well, an ironic situation will usually come around from time to time when the original programmer will have to revisit code that he wrote months or ever years prior, and due to the lack (or complete absence) of proper commenting, will have to waste valuable time just trying to re-understand what he was so confident of as not needing commenting in the past.

All of these potential headaches and wastes of time can be avoided by merely spending a few moments here and there adding comments to your code. These comments do not usually need to be long-winded explanations of what is happening (unless the code in question is very complex), but can be simple point-form notations of what the alphabet soup below them means. However, you also need to be careful not to add too many comments to your code as that can make the reading of the actual code more difficult.

COOLHAND

Some simple rules to follow for efficient and helpful code commenting include (but are not limited to):

- Label each and every data structure (classes, structures, enumerations, etc.) with a simple one line explanation of what they are for.
- Label functions with not only a simple explanation of what they are for, but also note what each parameter represents as well as what the (possible) return value represents.
- Label conditional statements and loops with a simple explanation or overview of what they are doing.
- Label any line or block of code that is responsible for creating new data (ie. mathematical equations, calling of another function which returns data, etc.).
- Label anything in your code that you had to think about for more than 5 seconds before actually writing it.

THE HISTORY OF I-IMAGINE

Part 2: South ****ing Africa!

It was a typical February day in Vancouver - that is to say it was about 10 degrees Celsius, the sky was a single colour of gray from horizon to horizon, rain was falling in a persistently heavy mist, and the dampness ate at the unprepared individual like a fat man at a Las Vegas casino hotel buffet - when I got a most unexpected email from Dan Wagner.

I had been keeping in contact with him from time to time ever since our meeting the previous September. Since that time, he had moved back to South Africa and had been spending his time working on some personal projects while trying to drum up interest for his I-Imagine project. We had never really talked about anything related to that during these past few months, and while I knew that he was still quite hopeful that he would be able to get it off of the ground, I had pretty much resigned myself to the fact that my future lay elsewhere. Which is why this February morning email was so unexpected...

From what I remember of the email (seven-plus years of on-again / off-again crunch modes in a game development studio can cause pretty bad memory loss...), it was pretty short and to the point. Dan had some exciting news that he wanted to talk to me about, and asked when he could call. Needless to say, I was very curious about this news so I responded to the email with a time later that evening (a ten hour time difference can play quite a bit of havoc with one's schedule), and tried to go about my business for the rest of the day, which was nerve-wracking to say the least!

Eventually, at around 11pm, the phone rang and as expected, Dan was on the other end of the line. The excitement in his voice was evident as he proceeded to tell me that he had managed to gather up a quite a few South African venture capitalists who were interested in investing in a game development studio. However, before they would take discussions any further, Dan said that I, along with some of the other guys in the States that he had been talking to, needed to fly out to South Africa to meet with them. The reasoning behind this was that while they were sold on the concept of the company, they first wanted to meet the team that Dan was putting together as they knew that at the end of the day they would be investing in individuals and skills, not merely in an idea on paper.

I was quite shocked and excited at this turn of developments, as it was the last thing that I was expecting. Without a question in my mind I knew that I had to go. However, I had to organise a week off of work in order to do so. There were ample excuses for me to use, but instead I decided to go and talk to Meighan, the lady who was head of operations of DigiPen and basically ran the school. I told her everything, and she told me that although she didn't want me to go (as she didn't like the idea of me doing anything that may cause me to leave their employ), she knew that it was an opportunity that I couldn't let pass me by. With Meighan's help, I concocted a fake case of a death in my family (my grandmother I believe...) with which to tell the other people at work so they wouldn't know where I was really going. I thanked her for her understanding and left to go and organise the details of my trip.

A week or so later I was sitting quite uncomfortably in a South African Airways plane, awaiting our descent into the airport formally known as Johannesburg International. It was the final hour of my 28 hour odyssey from Vancouver, and although I had no idea what to expect once we had landed, I was quite looking forward to having a shower. Those feelings of longing for water streaming down upon me increased exponentially when I stepped out of the plane and could feel the intensity of the South African heat in the air. I met up with Dan after collecting my luggage and headed off to the Don Suite Hotel in Sandton, which would be my non-air conditioned residence for

most of my time here. After having one of the most satisfying showers in my life, I went out to dinner with Dan as the other guys from the States would only be arriving in the morning.

The next day, once Felix and Jim arrived and had settled in, we all got together with Dan and his friend Nir to discuss the current situation. Felix, along with being an artist, was the former owner of a studio called FlexTech (Felix's nickname was "Flex") where Dan went to work in Las Vegas after graduating from DigiPen. He was there not only representing himself, but also his brother Kenny who was a designer. Jim was a classmate of ours at DigiPen and was currently working at Paradigm in Austin. He was also representing another former classmate of ours, David who was working with him at Paradigm.

After getting up to speed with where they were with the venture capitalists along with what the initial plans for the company were, we went off to a meeting with a couple of the VC guys who were the most interested in what Dan was trying to do. After they finished quizzing us all, they seemed quite happy and keen to get the other VC guys on board. That next round of meetings was to take place a a game farm called Finfoot which was out towards to Sun City.

I would have to say that the most interesting part of the trip to the game farm and back was surviving the South African roads. The memories of driving there at around 200 km/h in a Merc still make my palm sweat to this day. Not to mention that the only animals that we saw there were some tree spiders and a shongololo. Throughout all of this though I was having the time of my life. Here I was, 21 years old and more than halfway across the the world from my home. It was steaming-hot summer time and I had just come from a much cooler and damper climate, so sitting around for a week with Jim and Felix in the sun drinking Amstel all day long while in complete disbelief of where we actually were was quite a blast!

With regards to the real reason of our trip to the game farm, it went off without a hitch. The rest of the venture capitalists were impressed with the guys that Dan had assembled to start up the company, and Dan and Nir spent an entire night getting the budget for the first 2 years in line with what the investors wanted to spend. We left the farm and (a few days later) the country with a warm, fuzzy feeling caused by the knowledge that even after all of the waiting and the doubt, we would be starting up our very own game development studio in the not so distant future.

COOLHAND

Mobile Game Development in Java Part 8: Wrapping up

For the final tutorial in this series, we will look at what probably seems to be the part of a game that is least game-like, it's presentation as an application. Luckily, we will also learn that NetBeans makes this part of the process quick and painful by giving us a visual application editor, so we can focus as much as possible on the game itself.

To use the visual editor, we need to create a new 'Visual MIDlet'. Thankfully, we moved most of the nuts and bolts of our game out of the MIDlet already, so this won't be too tough. To do this, right click on your project and choose New->Visual Midlet, accept the defaults in the wizard that pops up and click Finish. The new editor that appears in NetBeans is called the Flow Designer, and to start with all it contains is an image of a phone with start and exit connection points. Notice the palette to the side, this is what we will use to add all our new screens. We will be adding a splash screen, a list so that the player can choose to play or exit, and of course our game canvas. NetBeans does not know that our canvas can be used in the editor, so we will have to add it to the palette first. To do this, right click on the palette window and choose Palette Manager as illustrated in Figure 1. In the manager, click Add From Project, and in the wizard that pops up, choose the Tutorial project, click Next and then make sure TutorialCanvas is selected in the list and click Finish. Finally, close the Palette Manager.

We can now add all of our elements. We do this by dragging them from the palette onto the Flow Designer. Drag one instance each of the SplashScreen, List (both under Screens) and TutorialCanvas (under Custom Components) classes. Now to keep things clear, rename the TutorialCanvas you added from displayable1 to tutorialDisplayable by right clicking on it and choosing Rename. Figure 2 shows the flow designer with all the screens added. Don't worry that yours

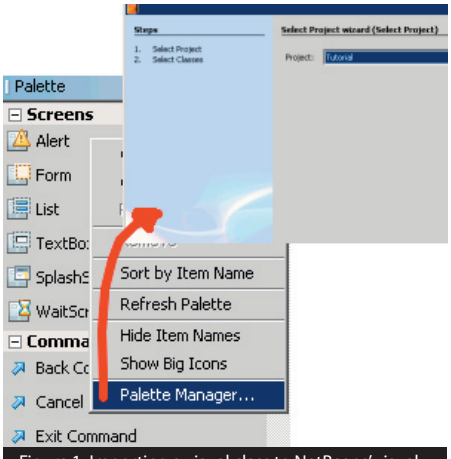


Figure 1: Importing a visual class to NetBeans' visual editor.

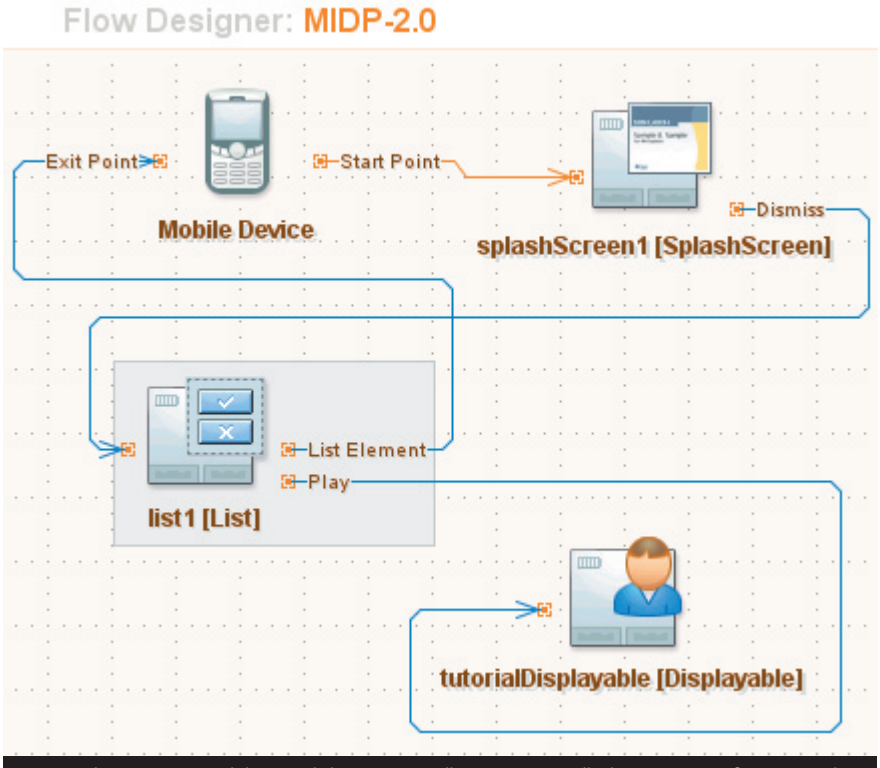


Figure 2: The NeatBeans Mobility visual Flow Designer allows you to visually design program flow in a MIDlet.

does not look quite the same, it's just missing the connections, which will be added next.

To make the splash screen run first, click on the Start Point next to the image of the mobile device, and drag a connection to splashScreen1. Now click on Dismiss next to splashScreen1 and drag a connection to list1. We can't just connect the list to our canvas, because we want multiple options, so double-click on list1 to open the ScreenDesigner. This works just like the Flow Designer and is illustrated in Figure 3. We want to add two options, so from the palette drag two List Elements onto the editing area. Click on the first element and change its String property to "Play" in the property editor pane below the palette. Click on the "..." button next to action to open a dialog that determines what choosing this option will do. Click the "Switch to Screen" radio button, and choose "tutorialDisplayable from the list. Close the dialog and change the second list element's string to "Exit", and change it's action to "Exit Application" in much the same way as above. Click the "Flow Design" tab above the editor to return to the flow designer. As you can see, our screens are all connected and ready to go. One thing is missing though – the list will set our canvas as the main displayable when we choose "Play", but what about the game thread?

To fix this, we will need to get into the code. To do this, right click on the "Play" connection on list1 and choose "Go to Source". See how most of the background to the code is blue? This is code generated by the visual editor, and we can't change it. Thankfully we have been left "hooks" where our code will be safe from the editor. We add our TutorialCanvas.gameThread.run() code just after the list selection sets our displayable as current, and set TutorialCanvas.exit to true just before exitMIDlet gets called. The final result is displayed in Figure 4. Now return to the flow designer by clicking the appropriate tab and we'll apply the finishing touches.

There's not much point in a blank splash screen, so we will add an image to that. First use your art application to create a new image called "splash.png" that will be displayed when the app starts and save it in your res folder with the rest of your images. A 100x100 image with the game's name should be fine, to start with-I called mine Happy Bounce. Back in NetBeans, double click on splashScreen1 to open the screen designer. In a similar way to adding list elements, drag an Image resource onto the splash screen. In the Inspector window, you should see that a new element has appeared under "Resources" called image1. Click on this item and it's properties will appear in the property window as in Figure 5. Change the Resource Path property to "splash.png", as soon as you accept the change, you should see your image appear in the screen designer.

Finally, compile and run your game. You'll notice you now have to choose between two MIDlets when you start, since both are in the jar. You can prevent this by deleting the old one from the project if you like. Also, notice that there is no way to get from the game back to the menu? See if you can figure out a way to get back when the game ends. Once you've done that, your game really is done except for gameplay tweaks. You now have the basic tools to create a great mobile game with a professional interface. You could extend what you have learned to adding high scores, or perhaps an options screen, it really is up to you. Good luck!

FLINT

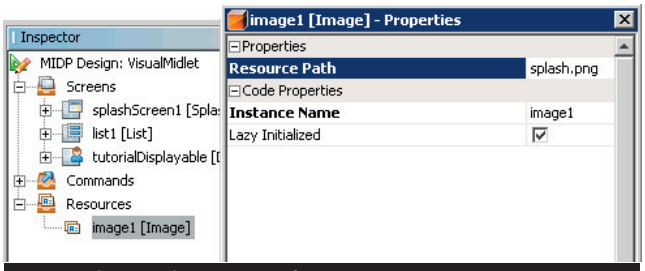


Figure 5: Changing the properties of a resource image.

Figure 3: The Screen Designer is a drag and drop WYSIWYG screen layout tool.

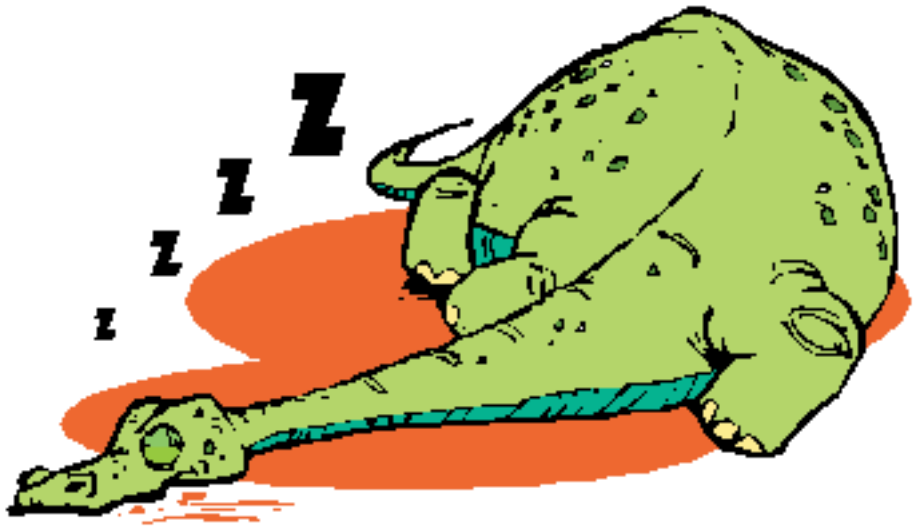
```
// Insert global pre-action code here
    if (displayable == list1)
    {
        if (command == list1.SELECT_COMMAND)
        {
            switch (get_list1().getSelectedIndex())
            {
                case 0:
                    // Insert pre-action code here
                    getDisplay().setCurrent(get_tutorialDisplayable());
                    // Insert post-action code here
                    TutorialCanvas.gameThread.run();
                    break;
                case 1:
                    // Insert pre-action code here
                    TutorialCanvas.exit = true;
                    exitMIDlet();
                    // Insert post-action code here
                    break;
            }
        }
    }
// Insert global post-action code here
```

Figure 4: Boilerplate code (in blue) and the 'hooks' left for custom code where we have made our changes.

10 THINGS LEARNT RUNNING PREHYSTERIA

Dion Scher is one of the wizards behind the design and running of Prehysteria (http://www.prehysteria.org), the game where you play dinosaurs fighting and trying to out-evolve each other to be the top of the bone pile.

A few issues ago, Dev.Mag featured Prehysteria as an up-and-coming online game. Now, after several "ages" of gameplay, Prehysteria has grown considerably and Dion shares with us what he found most important to remember during the creation and maintenance of such a project.



1. Have a plan

When my partner and I first sat down to write Prehysteria, we wrote out an action plan. Basically we covered high-concept topics like:

- What should the game be like?
- What are our goals and outcomes?
- What challenges lie ahead and how can we overcome them?
- How do we plan on attracting players?
- What is the central theme and how do we reinforce this in the game?
- How does the game run? Is it turn-based or real time and how does this affect play?

It was a brainstorming session and our thoughts, in bullet form, formed this document. In the initial design sessions, we would actually bring this plan to the meetings and sit with it in front of us. Each major decision would either be required to fit into our plan or the plan would have to accommodate the change in direction. Nevertheless, having a written record kept us firmly focused on our theme and goals.

2. Meet early and often

Meet at least once a week, at least until you're over the initial development hump. This is a non-negotiable. The entire team must be there. Momentum is paramount in getting a game going, and organising regular meetings makes it more concrete and forces the involved parties to act.

Physically meeting is putting aside dedicated time to analyse and explore the game concept. It reinforces the evolution of the game, ideas are generated, progress is monitored and perhaps most importantly, firm decisions are made.

It's easy to put down the phone after a telephonic meeting and continue playing your Xbox 360. But when you have a physical meeting it's a different story. Even if only slightly, it can make all the difference.

3. Theme is everything

Your theme is essentially what makes your game special. In a world of 1001 medieval, space colonization and pirate clones, Prehysteria was to be something out of the ordinary. There was nothing similar to Prehysteria on the market. And playing prehistoric dinosaurs with attitude is cool. Certainly there were other browser-based games on the Internet, but that is merely a technology platform. The theme is what made Prehysteria unique.

My partner and I discussed our theme and how to develop it. Thereafter, every decision had to fit into theme. Fortunately Prehysteria is a quirky game so we had loose boundaries. (I.e. Who says dinosaurs can't build walks and tree houses?)

I cannot stress theme more. If you break your theme and become something generic, then you've lost the magic.

4. You can't please everyone

No matter what you do and no matter how hard you try, someone will always get upset with you for some reason.

Sometimes this is due to a misunderstanding. To alleviate this we established a forum, in addition to online help, and Frequently Asked Questions (FAQ).

Involve your player base and local development community in helping to build the game. These people are gamers or game designers and are a rich (and often educated) source of feedback. Have an "open door" policy. Listen to feedback. Reply promptly and patiently answer questions.

However, some people are just moaners and there's no pleasing them. For example, Prehysteria offers the four basic races for free. The four advanced races are on a par, but additionally have a special ability. We took care not to allow this special ability to give the advanced races an overwhelming advantage.

Despite this a player complained that $2 a month was an unacceptable fee to play an ad-

vanced race. I explained that there is no material advantage and even pointed out that the top ranked players were playing basic races. This player was still not happy. Bottom line, he wanted the game, the entire game, for free.

Which brings me to my next point...

5. It is not a popularity contest

You can't please everyone all the time and you will piss some people off. Running a game is not a popularity contest. You will need to make some difficult decisions. Make your decisions based on the improvement of the game as a whole. If you make your decisions to keep a player happy, but the logic of your decision is flawed, your game will be worse off for it.

Never forget, you are a wizard. And therefore you must act like one. Enforce where necessary, like cracking down harshly on cheating. At the end of the day be reasonable. Players want to have fun and they will as long as the rules are enforced equitably.

6. Marketing is a bitch

The Internet has opened up an incredible channel to market your game, but there is so much "interference" out there getting knowledge of your game's existence to your target market is an art form in itself.

Take part in forums. Find game sites that will publish your link. Advertise where you can. If you have a marketing budget, use it wisely. There are many shifty sites that offer marketing services that mean squat.

Ultimately word of mouth is your best method of advertising. This relies largely on the hype of your game and the enthusiasm of your community.

7. Technology versus Story/Theme

What's more important, technology or story/theme? The short answer is "yes". Technology enables or imposes limits on your game, but story/theme defines it. Such is the nature of game design. You can spend all your time perfecting the technology. Creating a supersonic graphics engine and a mind-blowing user interface, but at the end of the day all you have is the "plumbing" of the game. It's the equivalent of installing Windows XP on a computer but failing to install any applications. It may look great and have all the potential, but without any software applications no work can ultimately get done.

Technology is vital for game design. It is your foundation. It enables your game, controls the mechanics and defines your limitations. However, that's all it is. You still need to create the actual game. A game engine is just that. You still need to build the world, make it interesting and give it flavour.

8. Innovate

Keep the game fresh and exciting. Listen to player feedback as to whether you're on track or not. An enthusiastic player is your best ally. In fact, some of our best ideas have come from players. An enthusiastic player wants to see your game succeed as much as you do.

That being said, examine all suggestions carefully. Weigh up the pros and cons, development time and priorities. Not all advice is good advice.

For Prehysteria we had a Priorities Document. Each meeting we'd take all the feedback we'd received that week, make a decision if the suggestion should live or die. And then prioritise.

9. You are your target market

Never forget this. You are your target market. There is a reason you went into game design - because you love games. Similarly there is a reason you chose to design a particular genre of game.

When it comes to decisions or new features, try and put yourself in the shoes of a player. Imagine what you would enjoy in a game like the one you're designing. What features would you require? What characteristics would enthuse you and what would cause you to turn away?

Remember that you're creating this game because this is the type of game you love. And if a new idea inspires you, there's a good chance your players will get excited too.

10. Stop farting in the wind

You've got a game idea, so *just do it* (sorry Nike). Too many game designers talk their games into oblivion. Until you have a product that people can play, you have squat. Nada. Nothing. Okay, reckon I've made my point.

You can't get everything right first time around. There will be bug fixes. There will be tweaks. There will be decisions made and routes taken that you end up having to backtrack on. It's all a part of the process. You won't get it all right first time, but if your players are enthusiastic and having fun, then you've achieved success. After all, a game is made to be played.

Hope to see you all online... biting and nipping... so come visit us at http://www.prehysteria.org!

Prehysteria - the coolest game since the ice age thawed.