



DEV.MAG

CREATE • DEVELOP • EXPERIENCE

ON THE BACK OF A NAPKIN:

**PART 4: TRANSPARENCY,
TRANSLUCENCY AND
BLENDING**

GAMEPLAY:

INNOVATION VS FRILL

MOBILE GAME DEVELOPMENT IN JAVA:

**PART 3 SAY IT WITH
SPRITES**



SOUTH AFRICA'S FIRST GAME DEVELOPMENT MAGAZINE



DEV.MAG ISSUE 5

**REVIEWS : PREHYSTERIA, BALL FALL
FEATURE : I.T INTELLECT LAN
FUNDAMENTALS OF 3D/BLENDER**

PREHYSTERIA BY DION SCHER AND EVAN HURWITZ

CONTENTS

REGULARS

03 - ED'S NOTE

04 - DIGITAL STOMPIES

FEATURE

05 - I.T INTELLECT LAN

SPOTLIGHT

07 - DION SCHER AND EVAN HURWITZ

REVIEW

08 - PREHYSTERIA

09 - BALL FALL

DESIGN

10 - GAMEPLAY

11 - BACK OF A NAPKIN PART 4: TRANSPARENCY,
TRANSLUCENCY AND BLENDING

13 - POSITIVE AND NEGATIVE FEEDBACK

14 - FUNDAMENTALS OF 3D AND BLENDER

16 - 2D ASSETS WITH 3D SOFTWARE: PART 2

17 - PROJECT MANAGEMENT PART 1

MOBILE

19 - GAME DEVELOPMENT IN JAVA
PART 3: SAY IT WITH SPRITES

TECH

21 - A LITTLE BIT ABOUT COMPUTER MEMORY
SYSTEMS

TAILPIECE

23 - POEM



ED'S NOTE

I would first like to take this opportunity to apologize for the lengthy delay that was experienced with the release of our last issue. Due to certain errors and problems in the design process, the issue was held back.

Now for good news, everyone! From this issue onwards, you can expect a new issue of Dev.Mag to be released on the last Saturday of every month. Additionally, and in keeping with tradition, we once again have something new to offer: This month sees GeometriX joining the team with the beginning of his episodic series on project management in game development.

rAge is on the horizon and it has the full attention of the Dev.Mag team who are preparing for the trip to South Africa's largest electronic and gaming expo. It's going to be a fun-filled trip where we'll be pushing Dev.Mag around and discussing its future. I personally can't wait and I hope to see you there.

Editor
Stuart 'GoNz0' Botma



THE TEAM

ENIGMATIC ED

Stuart "GoNz0" Botma

DASTARDLY DEPUTY

Rodain "Nandrew" Joubert

DILIGENT DESIGNERS

Brandon "CyberNinja" Rajkumar
Paul "Higushi" Myburgh

JUBILANT JOURNALISTS

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

WIZARDLY WEBSTER

Claudio "Ch1ppit" de Sa

WEBSITE

devmag.googlepages.com

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

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



GAME DEV ADVICE FROM PENNY ARCADE

<http://www.penny-arcade.com/2006/07/05>

Penny Arcade, a popular webcomic and a great source for gaming news as well, recently released an article written by Chris Avellone of Obsidian Entertainment (a man who's worked on big titles such as *Neverwinter Nights* and *Knights of the Old Republic*). Avellone gives advice regarding the gaming world and how aspiring game developers can get into it -- he even makes mention of Obsidian currently needing to hire some more designers. Any takers?



QUAKE WARS DEV DIARY

<http://pc.gamespy.com/pc/enemy-territory-quake-wars/717741p1.html>

Gamespy is hosting a series of articles on the development of the highly anticipated *Enemy Territory: Quake Wars*, written by the game's designers themselves. Part one focuses around the dynamic of balancing an FPS containing asymmetrical teams, each with its own unique units and different game objectives. Definitely a significant issue within the game, especially if it wants to get into the competitive arena.



2006 PGD ANNUAL 'THE BIG BOSS' COMPETITION

<http://www.pascalgamedevelopment.com/competitions.php?p=details&c=1>

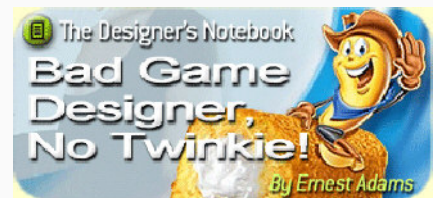
This Game Development competition, centred around development in Pascal and Delphi, recently announced the winners of this year's event, themed 'The Big Boss'. Entrants were required to design a game featuring bosses to challenge at the end of each individual level, with progress being monitored on a regular basis and intermediate deadlines being set to encourage development "road-mapping" and working on a regular design schedule. The winning games as well as prize-clinchers in various other categories can be downloaded and played from the competition's main page.

'BAD GAME DESIGNER, NO TWINKIE!'

http://www.gamasutra.com/features/20060710/adams_01.shtml

Now in its seventh installment, this popular series highlights common issues with games and why the people behind such titles can be considered bad designers. Flaws such as inadequate feature spreading or a lack

of subtitles are critically analysed, sometimes even using highly popular titles to show examples of bad design behaviour. An interesting feature of this series is that all 'twinkie denial' cases are submitted by readers, so if something grates you about games today, be sure to write to notwinkie@designersnotebook.com



THE CORPORATE ABUSE -- I'M LOVIN' IT

<http://www.mcvideogame.com/downloads-eng.html>

Yep, it's the McDonalds game by MollIndustria, designed as a satire against the major fast-food company and easily downloadable for your enjoyment. Flash games don't often cause such a stir -- but this one manages to do so, not only due to its controversial nature but because its fiendish difficulty is only matched by the equally fiendish addiction to play it. A prime example to Flash developers out there who want to make a game that people will enjoy.





I.T INTELLECT LAN

This is the ITI Dev.Mag LAN story. The date: 8 July 2006. The time: 14h00.

Empty... Unless you count me, Nandrew, and the dude who saw the LAN advertised on www.langames.co.za. Slight panic began to sneak in. There were moments where I dreaded the worst case scenario which would see the 3 of us locked in a battle of awkward silence, much like a scene from *Napoleon Dynamite*.

Fortunately, this was not to be. The cavalry started to arrive, bringing with them their weapons of war. Armed to the nines with high end graphics cards, gaming laptops and laser mice!. It then dawned on me that we had the makings for a small LAN party. I then did what any mediocre LAN organizer in my position would do: push play and kick-start the afternoon's proceedings with a thumping base line.

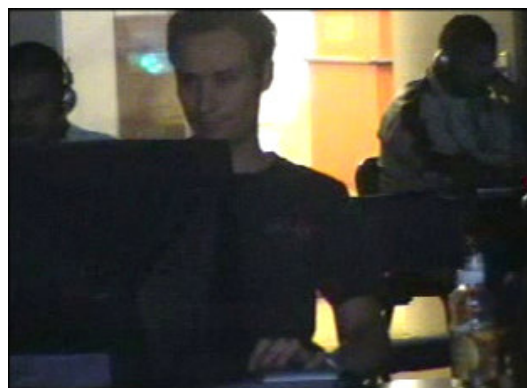
All over the room we had Dev.Mag posters plastered and placed, as, at the core of this massacre, there were 2 simple goals: To generate awareness and promote the Mag to the public. Dev.Mag: the brand, the image, the coolest online-focused game development magazine around; a magazine for the people, by the people! VIVA! The second goal was to generate funds so that we can start marketing our fantastic Mag to the people. That was my mission for the afternoon, and if I could have some fun whilst punting



Dev.Mag, then why the hell not? I also learnt a lot from the gamers present, including that *Counter-Strike* can become very repetitive if you are one of the unfortunate n00bs who continuously gets head-capped first in your team. *Quake* is an awesome pick-me-up from an hour-and-a-half long *DotA* mission.

Free coffee is always a welcomed sleep suppressor, and you can often get lost in a conversation over *Protoss* and *Zerg* strategies, or about which hero was the worst in the last round of *DotA*. Yeah! All and all the evening turned out to be a fun, fanatical, frantic frag fest. [Alliteration FTW! –

Eds] I was also privy to a bit of underground “indie” game-development when our very own Nandrew showed me his work on *Line Wars*, a mixture between your pre-school art lessons and *The Matrix* which makes for addictive shooter action.



FEATURE



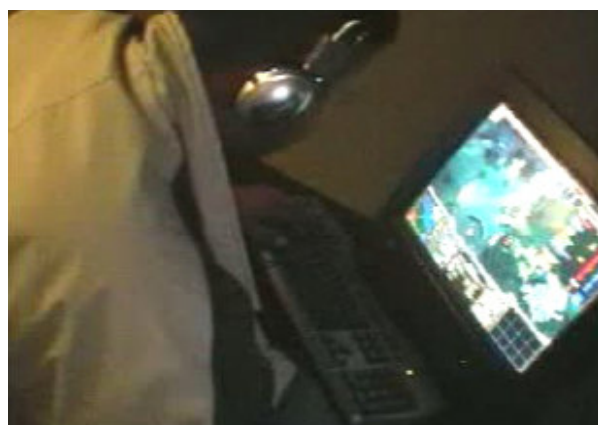
As the LAN carried on into the night we had attracted the interests of some guys that passed by. They must have heard the thumping *Infected Mushrooms* soundtrack playing and decided to investigate this small phenomenon. Needless to say that they were blown away by what they saw when they entered the room: An episode of *Bleach* projected on the wall, music pressure-cooking the atmosphere to boiling point, and a room full of eager dudes, kicking a little bit of ass! Laughter and cries for mercy filled the air. It was *electric*, for lack of a better word.

The venue was awesome, and it really was a pity that, due to circumstances beyond our control, we had a smaller turnout than expected. However, a lot of good also sprung up from this experience. The players saw the potential and value of ITI as a venue for gaming and will definitely return for future LANs.

This is wonderful news for the Dev.Mag marketing team, as we will be able to kick-start some form of tangible “merch” for the team and supporters alike. A new domain is also on the cards and definitely something to look out for in the months to come. I feel that this is the beginning of something big for Dev.Mag and I look forward to watching our valued readership grow until we eventually catch the eyes of some new sponsors in the not-too-distant future. “A journey of a thousand miles begins with one step.”

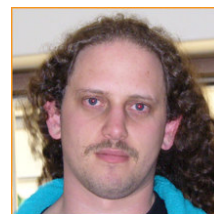
Be on the lookout for the Dev.Mag LAN promotional video on the website!

BURNABIS



PREHYSTERIA WIZARDS

Prehysteria is a new online strategy game in which players fight, evolve and eat their way to supremacy. **FENGOL** has the lucky honour of calling Dion Scher (left, 32) and Evan Hurwitz (right, 27) good friends, and when they released *Prehysteria* as Alpha, they invited him to play-test it. Now, with the release of the Beta version, he decided to go interview them. This is what they had to say.



How long have you been working on the project?

[Dion] We actually had the idea 2 years ago and have been talking about it on and off for a while. Last year in December I decided to start working on it properly and I asked Evan to help with the documentation and calculations.

What research have you done for *Prehysteria*?

[Evan] We used our own personal experience from playing games like *Kings of Chaos*, *Hell Wars* and *Darkthrone*.

In terms of game-play, what has been your biggest challenge? How did you overcome it?

[Evan] Dion has spent too much time playing as a Game Master in roleplaying games and tends to like hurting players more than necessary. Game balance and fine tuning is our biggest challenge.

[Dion] We've got Evan debugging 100% and writing the help file to help players in difficult areas. We also believe team-play is very important in our development, and we meet once a week in official meetings and spend up to 3 hours on the phone each day discussing ideas and plans.

Do you belong to any game development communities?

[Evan] No, but we're both active gamers and belong to local LAN and gaming groups.

Do you think there's any value in a game development community?

[Dion] Yes and no. While development communities provide valuable resources, opinions and good reviews, I worry about the politics that generally surrounds these groups. I'm also worried because of the heavy focus of designing into oblivion and losing momentum. When we go Beta we will be jumping on communities like Game.Dev to ask what people think and getting people to play-test the game.

When you go live what will you use for subscription payments?

[Evan] Paypal! It's been available in South Africa for awhile.

What technologies are you using in *Prehysteria*?

[Dion] ASP.NET 1.1, Javascript, SQL Server 2000

Are you going to be looking at any Web 2.0 technologies?

[Dion] Not at this moment, it's not important to the game.

You've built an MMO but the inter-player communication is rather weak (no trading resources, no collaboration except for premium species and no communication with lower ranked players), are there plans to improve this?

[Dion] We've tried to build an office game that only takes a couple of minutes a day to play. Collaboration would require more of the player's time plus collaboration would make it difficult for starting players. It also doesn't fit into the theme of the game; species don't mingle well.

What is your message to other game developers in South Africa.

[Dion] It's a good idea to let ideas mature between coding periods. Breaks are important to let ideas grow. You must also have an Outcomes Document so that you know where you going and can tick off what you've finished. It's very important that you finish your product.

[Evan] Like Nike, 'Just Do It!' Set real deadlines and don't be afraid to make mistakes.

For more info about the game, check out our *Prehysteria* review in this month's issue.



SPOTLIGHT



PREHYSTERIA



Prehysteria is an online, browser-based strategy game which pits you against other players in a turn-based eat-or-get-eaten "Prehistoric" world. After creating your own unique species, you have to hunt, forage and evolve your way to the top of the bone pile while making sure that you don't end up as somebody else's supper!

The premise to this whole game is startlingly simple. Eat whatever you can get your hands on (kinda like the French), and use this to increase your armies, buy more workers, create more structures or acquire all-important evolutions that will help you defend yourself against all the nasty sharp-toothed critters out there.

Competing against other players quickly makes things a lot more complex though, and soon you have to make critical decisions with your limited supplies – do you purchase additional defences to protect you against that annoying, aggressive mammal who keeps attacking you, or do you enhance your breeding capability so that your armies can grow quickly enough to smack him about later? What about getting a spy network to set up snares

and traps for the next time someone decides to stomp over and take a bite out of you? Add

to this the additional 'resource' of hatchlings per day for the colony and suddenly you have a game where quick adaptation to ever-changing situations is the key to survival.

Prehysteria is still in beta, meaning that the game world is currently going through a lot of interesting changes to experiment with new ideas or appeal more to players. Of course, this has both its advantages and its drawbacks – some balancing issues are still being addressed by the developers, and some chosen lifestyles seem to be more difficult to play than others. However, this *does* mean that designers are very much open to feedback on their title, allowing you to shape further development of the game yourself and even score a few free

hatchlings if the developers like your idea!

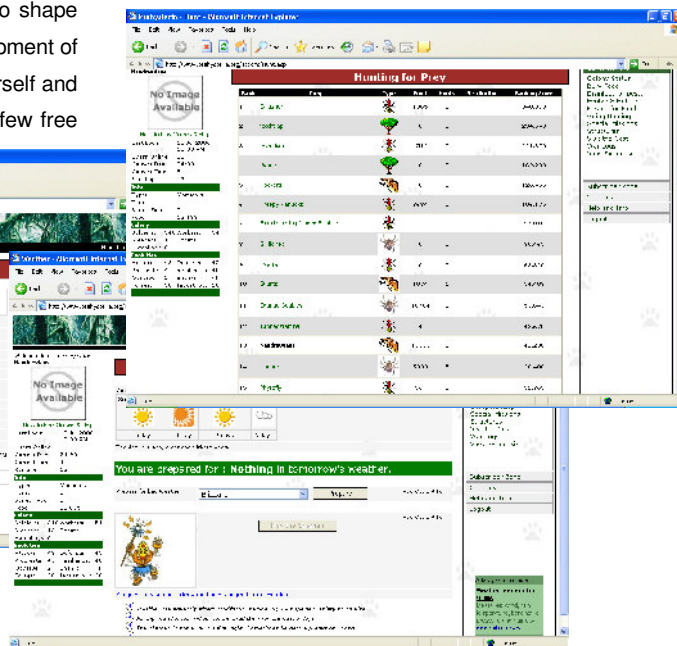
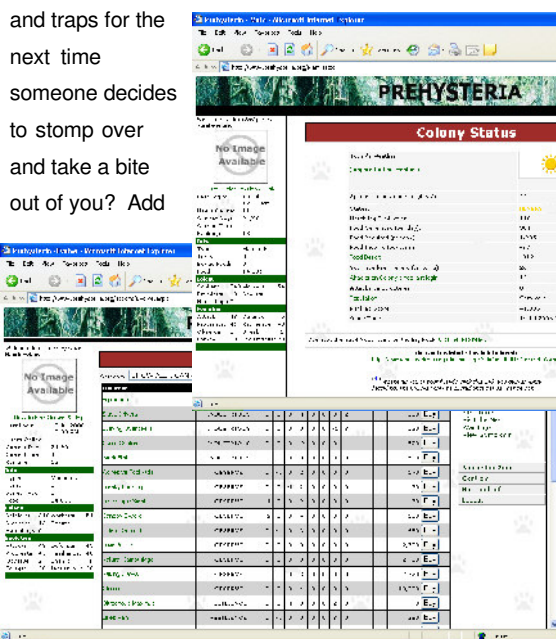
Prehysteria currently boasts at least fifty active players, all gathered during its first couple of weeks in existence, and it's growing fast. It's fun, takes up only a few minutes of your day and is easy on any system or connection. So go ahead! Signing up is free, and so is eating your friends! Play *Prehysteria* now!

NANDREW

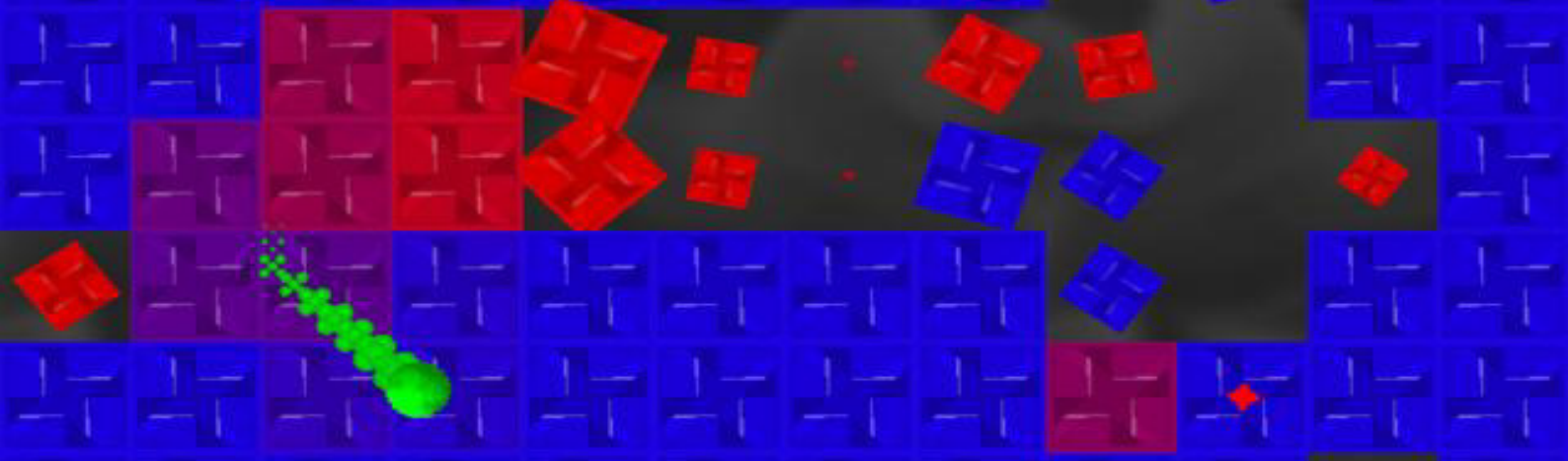
FIND IT!

<http://www.prehysteria.org>

REVIEW



Prehysteria takes place through a simplistic, 56K-friendly series of management screens, similar to other games of its genre.



BALL FALL

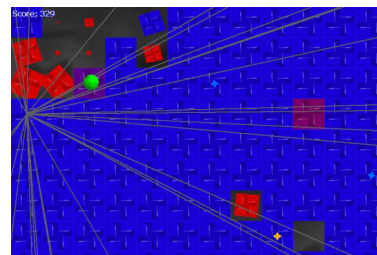
Ball Fall takes a relatively simple concept and turns it into a really addictive and fun game. The goal is to keep the ball on the board for as long as possible, with every tile on the board falling down when you pass over it and returning after a few seconds. The longer you stay on the board, the higher your score is. An array of powerups are thrown into the mix to keep things interesting

The game seems a little daunting and confusing the first time you play it but in no time you'll understand what's going on. Soon, you'll be hooked and will keep trying to break that elusive 10000-point barrier. The powerups add a lot of depth to the gameplay and are activated by passing over colourful, flower-shaped objects. Different colours do different things – for example, red will stop all blocks from falling for a brief period, while blue will create a spiral flame which keeps growing bigger until you touch the centre of the flame – the bigger the flame is when you do this, the higher the score bonus you receive.



Overall, it's a polished and great experience. The menu is simple and effective, and you get a clear explanation of the object of the game and what the different colour powerups do. There are also options to turn sound effects, music and fullscreen mode on or off which is a welcome touch. The background music is a midi version of the song Track 2 by Blur and doesn't get too annoying in the background – but if it does, then you can easily turn it off.

Unfortunately, while the game is really addictive and fun at first, after a while it becomes a bit repetitive. It would be great to see more levels which incorporate puzzles or a new challenge to add depth and longevity to the game. Aside from this particular shortcoming,



it's a great game which takes a simple concept and executes it well. Everyone should give it a try.

INSOMNIAC

Ball Fall was created by Brian Elting and can be downloaded from <http://www.gamemakergames.com/?a=view&id=184>

REVIEW



GAMEPLAY

Most triple-A games these days really don't live up to their expectations. They try to be too ambitious (even at that level) and fail horribly. What I have discovered is that developers want to improve their games by adding more. More eye-candy, more gameplay elements, more Chuck Norris.

This isn't working like it should. Usually, more is better best described by the term "the more the merrier" – but let me explain what I am trying to say.

Lets take a basic retro game as an example: Tetris.

Apart from the Russian music backtrack, Tetris was insanely addictive. It lived around the core gameplay of fitting blocks neatly into place. That is all. No other frills, just good plain addictive russian block-fitting!

Now, let's take a modern game ... I am going to suggest Black and White 2.

It is almost exactly like the first. The things that made it great return in the sequel, but what did the developers do to improve it? They added more stuff. Increased villager interaction, more spells, etc. What they dont realise is that they are just adding things on the side without any worth to the core gameplay. As you all know, Black and White 2 didnt score as well as the first. It wasn't as 'special' as the original. Why? The core gameplay remained exactly the same. The other added stuff did not add to the core gameplay.

Now, let's get back to Tetris. Let's say Tetris' addictive design is 'rediscovered' now. This is how it would turn out.

"Tetris! From the critically acclaimed AAA comes a new title so addictive you will want to go to Russia! It has an amazing unique backtrack done by the impeccable Yun Kamagi and totally awesome graphics."Right, nothing wrong with the above, but here comes the twist.

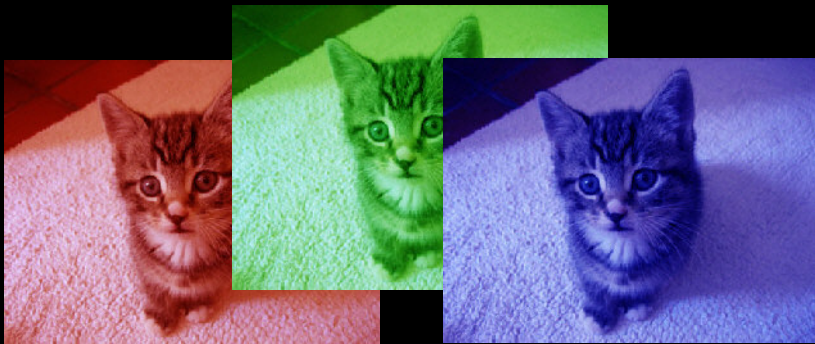
"Tetris not only has block fitting galore, but there is a deep system behind the block fitting. Small people known as

Tetris-ians command the blocks to the bottom! If you do not keep them happy, you will lose money. The management behind the Tetris-ians requires impeccable skill to contain them and keep them happy. Tetris is awesome!" Well, I tried to explain, but I hope by now you understand. Developers add things that do not add to the core gameplay.

The way to go? If you design games, more isn't always better. If you add something new, think about it. Does it add to gameplay or is it just frill?

TR00JG





RGB values: never look at household pets in the same way again!



ON THE BACK OF A NAPKIN:

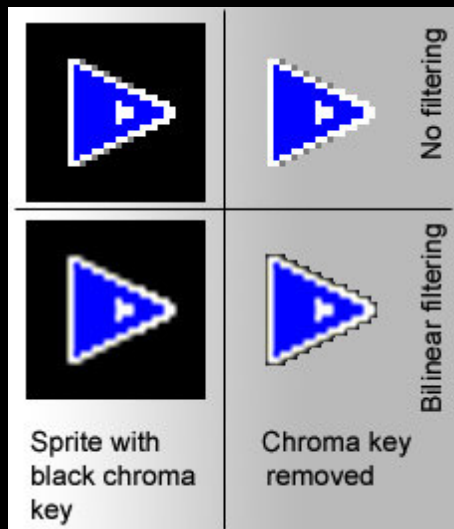
PART 4: TRANSPARENCY, TRANSLUCENCY AND BLENDING

Transparency effects in games are very important, not only because they make games look good, but also because they provide a unique set of problems and opportunities for smart designers to exploit. This is because transparency is implemented using a very versatile blending system, which can be put to use in tons of different ways ...

Transparency, the first method

Back in the day when people manually blitted their sprites to screen memory (if you're too young to know what blitting means, Google can help with that curiosity), they used to pick a specific colour in the sprite and simply not draw anything to the screen when they encountered that colour. Thus a technique called *chroma keying* was born!

Nowadays graphics APIs support chroma keying without the whole manual pixel colour check thing, which to be perfectly honest was a bit annoying to do. However, chroma keying does have a downside in modern applications: Things like bilinear filtering (remember last month's texture filtering article?) will "blur" individual pixel colours together, so you'll end up with the edge pixels in your texture having some of the chroma colour bled into them, giving you a nasty "halo" around your transparent object.



Translucency and the alpha channel of doom!

Hopefully many of you will have heard of the idea of "alpha". When we talk about textures and transparency, alpha refers to an extra channel per pixel. We already have red, green and blue channels that define the colour of a specific pixel in a texture or sprite, if we add another channel to that we can really start doing some funky things.

The simplest implementation of alpha is single-bit alpha. That means that each pixel has an alpha value of either 1 or 0 which means it's either on (visible) or off (not visible). This may sound similar to chroma keying, but the major difference is in how the alpha information is used by graphics APIs.

Before we tell whichever API we're using (GL, DirectX, or any engine that uses either of those) to use a specific texture while it's rendering, we can specify a *blending operation* for it to perform while it's using that texture. Essentially blending is all about controlling the final colour value of a pixel on screen by multiplying the source colour information (the colour that the system is trying to make the pixel, gotten from a sprite or texture) and the destination colour information (the colour that the screen pixel is already) in all sorts of different ways. The usual blending operation is to get the final colour on screen by multiplying the source colour by its alpha value, and adding the destination colour multiplied by 1 - the source's alpha. This gives us something similar to what you get by looking through a piece of coloured glass: The colour behind the glass is "let through" by how transparent the glass is, while also being tinted by the how un-transparent (opaque) the glass is. This means that an alpha value of 0 doesn't change the onscreen colour at all (ie: fully transparent, perfect glass) while

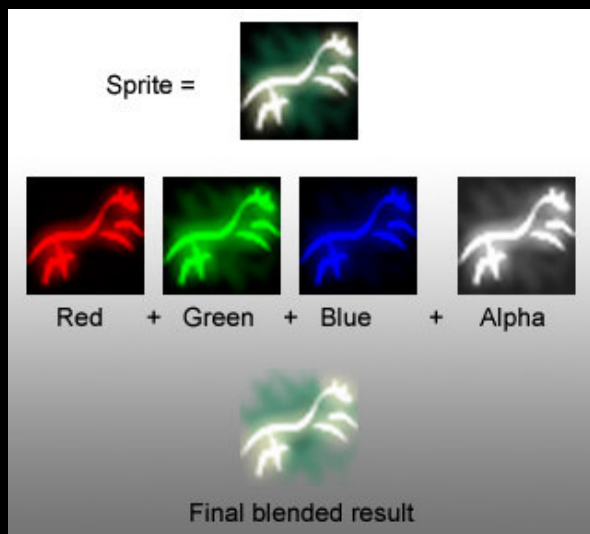
an alpha of 1 doesn't let through any original onscreen colour at all and we only see the texture colour (ie: totally opaque, like a wall).

Now, if we give our alpha channel more bits, we make it possible for the alpha to have more values than just 0 and 1. This gives us wonderful, smooth, graduated translucency, just like our coloured glass example. Imagine looking through a red pane of glass that's half-transparent at a green field: Our source (the glass) has a colour value of $S\{\text{red} = 1, \text{blue} = 0, \text{green} = 0, \text{alpha} = 0.5\}$ – this is called RGBA notation (I'll let you guess why). The grass behind it is represented by $D\{0, 1, 0, 1\}$ because it's completely green and also completely opaque! So, using the blending operation we described above, we get our final colour by adding S multiplied by its alpha to D multiplied by $1 - S$'s alpha (this is called giving the inverse of a value, subtracting it from its possible maximum value).

So:

$$\begin{aligned}
 F &= (S * S_a) + (D * (1 - S_a)) \\
 &= (S * 0.5) + (D * (1 - 0.5)) && \text{: Substituting } S_a = 0.5 \\
 &= (S * 0.5) + (D * 0.5) && \text{: Calculating } 1 - 0.5 \\
 &= ((1, 0, 0) * 0.5) + ((0, 1, 0) * 0.5) && \text{: Substituting } S \text{ and } D\text{'s colours} \\
 &= \{0.5, 0, 0\} + \{0, 0.5, 0\} && \text{: Multiplying out} \\
 &= \{0.5, 0.5, 0\} && \text{: Adding the colours together}
 \end{aligned}$$

So our final value for F is $\{0.5, 0.5, 0\}$, giving us a half-red, half-green, mucky brown colour. Exactly what you would see if you looked through a red glass window at a green field, yay! Thankfully you don't have to do this type of calculation all the time, your graphics card is doing it millions of times per second for every pixel on screen, more yay!



Hopefully some of you will have realised that you need to have drawn the grass before drawing the red glass, otherwise all you'll see is the grass (if you can't figure out why, redo the calculation above and switch S and D around, to see what you get). This is called transparency ordering and is one of the nastier problems of 3D graphics, we'll get to it later after we've discussed other nifty things like Z buffering...

Other uses of blending

As I mentioned earlier, blending is done by your graphics card and defined by the blending operation you choose. The blending operation we've gone over so far is only one of many different possibilities. You define a blending operation by telling your API what values to multiply the source and destination colours by, so the operation we looked at above is *blend(source alpha, inverse source alpha)*, this is the setting used to get "normal" transparency. Additive transparency is given by *blend(source alpha, one)* because we want the screen colour to be unchanged, we just want to add our source colour "on top" of it.

There are many different blending values that you can play around with: Destination alpha, inverse destination alpha, source/destination colour, inverse source/destination colour, maximum, one, zero, etc. Experiment and see what kinds of effects you can create. Blending is extremely versatile, some people even use blending to achieve advanced stencil and lighting effects, all it takes is a little thinking outside the box.

Key points:

- Blitting = oldschool.
- Chroma keying = colour that isn't drawn to screen, can have issues with filtering.
- Alpha channel = extra channel in textures/sprites along with red, green and blue.
- Blending = operation that does things to source and destination colour to get final.
- Blending values = many and varied, can achieve lots of effects.

DISLEKCIA



OMG ur dmg is
so imba

LOL im lvl 99
n00b!!!1!



POSITIVE AND NEGATIVE FEEDBACK

Everyone who plays games wants to see their character, country or team getting better as they play. The advancing of character skills, countries' power and team influence is the key to why people play the game. When a game designer creates a new game, the development throughout the game is core to the design.

Typically in games, a small advantage can often be turned into a big advantage. Such as finding a rare Diablo magical item, or having the upper hand in resource gathering in an RTS. An advantage like this can often help the player get a bigger and bigger advantage in the game. The RTS player can then build more armies from the extra resources, which results in said player being able to get a territorial advantage, and then more resources, etc.

The extra opportunities gained from a slight positional advantage in a game is called positive feedback. Basically positive feedback is the process in which a small advantage can be extended to a bigger and bigger advantage. Positive feedback is both a good and a bad design principle to build into a game. In single player games positive feedback can be used to allow the player to quickly build a bigger and better position. In multiplayer games its more important to limit positive feedback to give everyone a fair chance in the game.

The opposite of positive feedback is negative feedback. Negative feedback is used to slow down the speed in which players can gain an advantage. For example the cost of building new units could be based on the number of units

the player already has. Negative feedback can also be used to adjust the level of AI to match the players skills. Negative feedback can sometimes spoil a game for a player as they may find it difficult to grow their position in the game.

New players that play the game for the first time will often find the use of positive feedback an encouragement to playing the game. Their positions in the game will quickly improve, thus encouraging them to keep playing. Players that already know how to play the game will use and positive feedback systems to attempt to get an advantage over the other players, while negative feedback will make the game easier to balance between good and bad players by not allowing the better player to build such a big advantage over the other players.

Good players will find ways of using both positive and negative feedback systems to their own advantage. Both positive and negative feedback systems can be used to balance a game. However, it must be remembered that a positive feedback system would typically not allow new players to join in a multiplayer game as the first few players would have an insurmountable advantage purely from having started playing first. The use of a negative

feedback system could allow the players that joined later to catch up to the first few players more easily.

A game can implement a handicap structure by adjusting the effects of the positive and negative loops on the various players. Weaker players could be given a larger positive multiplier to allow them to compete more easily with better players, or they could be penalised less for any advantage that they achieve. For example, if each soldier a player builds costs twice the number of existing soldiers in gold (negative feedback) a weaker player could have each soldier costing only 1.8 times the number of soldiers they already have.

By keeping in mind the effect of the resource and skill systems on the advantages that players can achieve in a game, the system can be adjusted to ensure balance for all players of the game. Some games require quick sudden advantages in a player's game experience, while other games need to limit the speed in which the teams build their strength. Time spent identifying the effects of the positive and negative feedback loops in a game will make that game easier to balance and therefore more fun for its players.

CAIRNSWM

DESIGN

FUNDAMENTALS OF 3D AND BLENDER

A skilled 3D modeller is capable of creating life-like replicas of anything he can imagine, ranging from people to machines, to entire landscapes. Almost every object you see in modern computer games is a 3-dimensional model.

While there are many tools that can be used in modelling, most are extremely expensive. Blender is a free, open-source alternative that is capable of creating nearly anything that professional software can. This tutorial will teach you how to create 3-dimensional models in Blender.

To understand modelling, one must first understand the basic concepts and elements that comprise 3D models. The most basic unit is a vertex, which is simply a point in space. Two vertices form a line, or edge, and three or more vertices can form a flat plane known as a face. Multiple faces form solid, 3-dimensional objects. *[More detail on 3D concepts can be found in our "Napkin" series - Eds]*

Now, go ahead and open up Blender. You should be presented with a relatively complex screen full of buttons and gadgets. Most of these you don't need to use yet, so you can safely ignore them. The majority of your screen area is dedicated to the 3D view, which shows you your scene. The

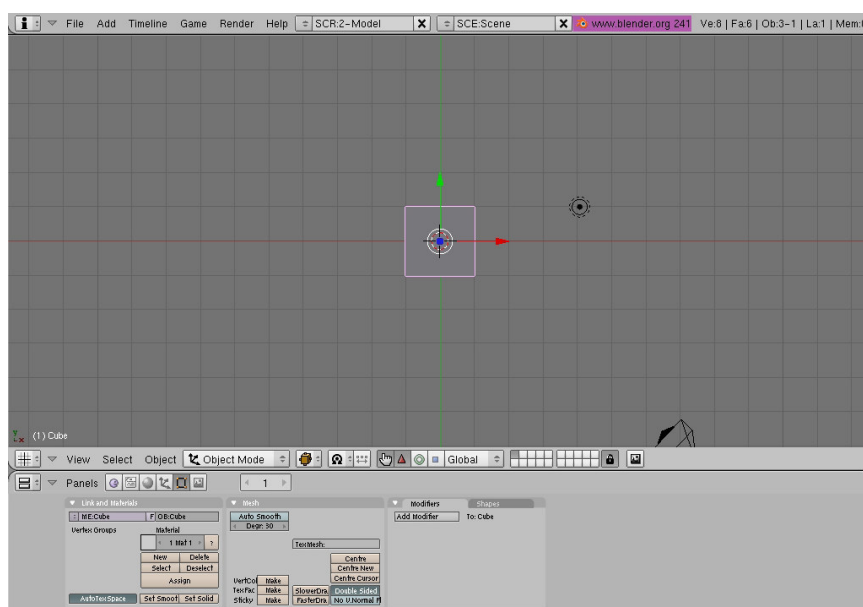
upper portion of your screen is the menu bar which contains more options related to Blender and its functions. The lower portion of the screen is the *Buttons Window* which contains all the functions you'll ever need to manipulate your model. At this moment, you shouldn't be overly concerned with their uses.

Looking around

The first step to creating a model is to know how to move your view around. The middle mouse button is used extensively in this process, but if you don't have one you needn't worry. The left mouse button, together with the ALT key, can serve as a substitute. Hold

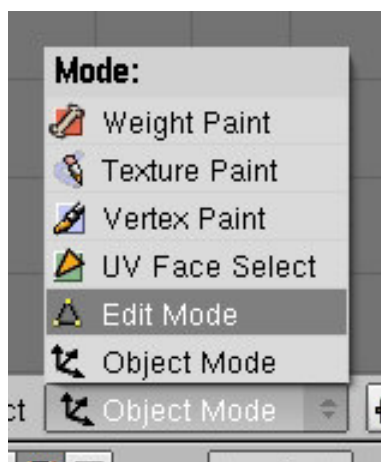
down the middle mouse button (Or ALT + Left mouse button) anywhere in the 3d view, and move the mouse. This will rotate the view. If you hold down CTRL while clicking, the view will zoom in and out, and if you hold SHIFT your view will pan. Try these now to become acquainted with the basic controls.

You can also use the numerical keypad (NOT the numbers above the QWERTY keys) to change to preset views. Press 7 to change to top view (which is the default view), 1 to change to front-view, and 3 to change to side view. 5 will toggle between perspective and orthogonal view. All of these view functions can also be accessed from the view menu under the 3d window.



The Blender interface

You will notice that, in addition to a cube, the 3d view is populated by two other objects: One is a circular item that represents a light source. Without lights, all items in a scene will simply be black and unlit. The other object, a pyramid shape, represents the camera. It is from this point that the scene will be seen when rendered. You can see the scene from the camera's point of view by pressing Numpad 0. You can render the scene now by clicking the Render option on the menu, then Render Current Frame, or simply by pressing F12. This will bring up a window that will show the scene from the camera's viewpoint. It's not much, but it's a start.



Changing to Edit Mode

If you are using a recent version of Blender, your view may also contain coloured arrows. These represent Blender's '3d transform manipulator'. This is just alternate way to manipulate objects and you can safely disable it to get it out of the way. We won't be using it. Press CTRL+SPACE to remove the arrows, or click to hand icon under the 3d view.

Moving and changing things

Press 7 to change to top view. The pink outline of the cube tells you that it is the currently selected object. If it is not outlined in pink, you can right-click on it to select it. If you hold down SHIFT

while right-clicking you can select multiple objects. Pressing A will toggle between selecting and deselecting all objects. Pressing B will allow you to select multiple objects by drawing a box around them. Try these various methods to select the three items in your scene. You may need to move your view around to see all of the objects. Also note that if you hold CTRL while performing any of these actions, the changes will be constrained to grid units.

Now, select the cube object with the right mouse button and press G to activate grab mode. If you move the mouse, you'll notice that the cube will move also. Press the left mouse button to place the object in its new position, or right-click to cancel. Pressing S will activate scaling mode, which will allow you to change the object's size, and R will activate rotation mode, which you can use to spin the object. Rotation can also be performed on the camera object and on certain lights, but will have no effect on the one in your scene. Try rotating, moving and scaling some objects now, and then press F12 to see what effect they had on the render.

In addition to these controls, the camera object can be moved in another, simpler way. By pressing CTRL + ALT + Numpad 0, the camera will move to the location of the 3d view. While manipulating objects as a whole is useful, to create detailed models one must be able to edit the finer details of an object. While making sure the cube is selected, click the drop down box under the 3d view that reads Object Mode, and change it to Edit Mode. Pressing TAB will also toggle Edit Mode. You will notice that the individual vertices of the cube will become visible as pink dots. Whilst in Edit Mode, you can manipulate each of these vertices just as you did with the cube as a

whole. Select some vertices and try moving them around to see how it affects the cube and the render.



Changing to Wireframe Mode

Changing the view to wireframe mode will allow you to see the vertices and the entire structure of the model more easily. To change to wireframe view, click the drop down box shaped like a cube under the 3d view, and change it to wireframe mode, or press Z. This is useful when in edit mode. There were a lot of things to cover in this section, but now you should be aware of all the basics. You should try using all these features to become acquainted with their use. You'll be using them often. And that is all for this Blender tutorial. Next time, we'll delve into more detail, and learn about lights and materials.

CH1PPIT

Where to get it?

Blender is available for free from www.blender.org. This tutorial should be correct for most recent versions of Blender. The latest available version is 2.42, and this tutorial was created with version 2.41

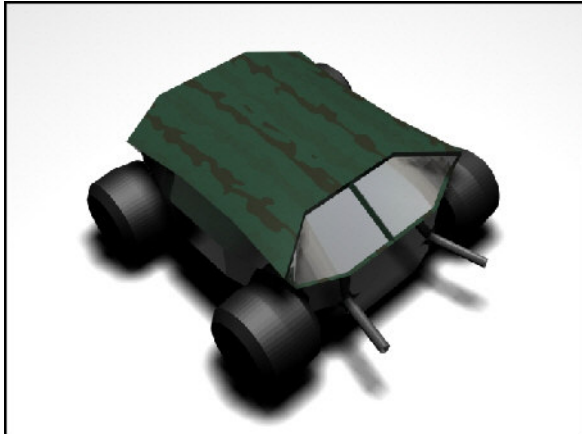
RECAP - CONTROLS

Middle Mouse Button: Move around. Pan with SHIFT, zoom with CTRL.
Right Mouse Button: Select objects.
F12: Render scene.
Numpad 1, 3, 7: Change to preset views (CTRL changes to opposite side)
Numpad 0: Change to camera view
B: Enable box select
A: Select/deselect all objects
G: Grab
S: Scale (Resize)
R: Rotate
TAB: Toggle Edit Mode
Z: Toggle Wireframe Mode

MAKING 2D ASSETS WITH 3D SOFTWARE PART 2

Last month we ended with a rendered teapot. Not very useful, is it? This month, I would like to put a model you've created, or downloaded, into Game maker itself.

To start off, you need a 3D model that you want to use in your game environment. I made myself this little buggy/



tank for my upcoming title, Mech's Destiny.

I imported the model into 3D max and set up my scene like we did last month. I used a white background once again, but this time it's because the tires on my buggy are black, and my environment in Mech's destiny is white. After I have set up all the lighting I desire, I make sure my top view fully extends the model. I then do my first render.



Now that I have my sprite rendered, I will have to reformat it a bit to work correctly with game maker. The first thing I normally do is open the new file in my painting program, and resave it as a .gif file. I use .gif format because it's 256 colours and is much smaller than .jpeg. One does lose some quality, but it makes up for it in the size. You could always keep the higher quality file, for a bigger but "higher quality" download version of your game. After resaving it as a .gif file, I close all images and reopen the .gif file. This loads the palette the newly saved file uses.

Most painting suites have the "magic selector" option. This automatically selects regions, much in the same way that Game Maker would decide what is opaque. I would then select the outside. After doing this, I would increase the colour depth to 24bit, and paint the outside of my image with a new colour that is not used in the sprite (I used red for my buggy).

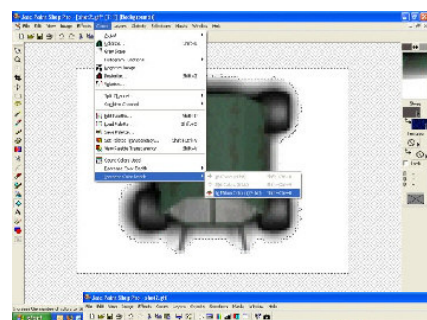
Now decrease the colour depth back to 256 colours, and resave the .gif file. Now comes the hard work of cleaning up

the shadows on the image. Take a pencil tool, and clean up the areas that need clarity *[the magic selector can also work in a pinch, but it's a little more fiddly -- Eds]*. When you are done, it should look something like the image below.

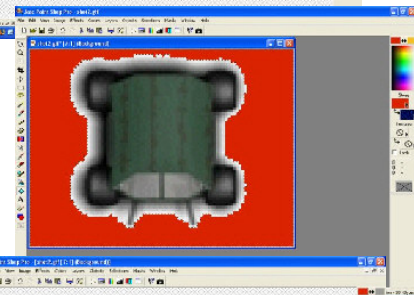
Now we finally get to use our asset in Game Maker. Open game maker and create your object. Add your new sprite and assign it to the object. How does it look? Need to redo some post-editing? Maybe you'd like to render it with a little more light, or with a different texture? Next month, I'll show you how to make a fake shadow to go along with your sprite, which is all done within game maker. And then, in the following month, we'll start with animated sprites.

HIMMLER

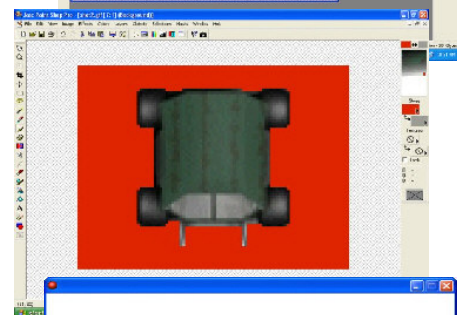
1



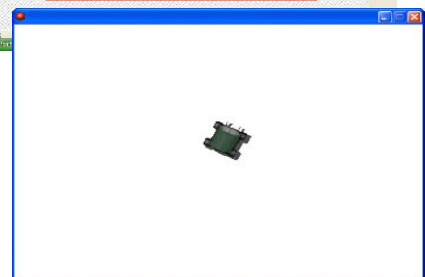
2



3



4



PROJECT MANAGEMENT

"Projects happen in two ways: a) Planned and then executed or b) Executed, stopped, planned and then executed."

-Project Management Proverb

Project Management is an oft-overlooked, yet invaluable part of any undertaking, be it building a paper airplane, or an entire operating system. Without properly applied project management, you are setting your project up for failure, or, at the very least – an exhausted schedule.

What does all this have to do with making computer games? Everything. The entire process of creating a game can (and should) be managed – from the interface, to the story, to the voice acting. So, how exactly *do* you manage a project?

There are hundreds of books written on the subject – each one giving a slightly different opinion, set of rules, or steps to follow. My goal behind these articles is to give you sufficient insight into PM as a whole, and give you the tools necessary to provide a basic level of management for any game you create.

There are 5 broad stages to successfully implementing a project. Each is essential, and their use will allow for a smooth, relatively snag-free project. These stages are:

- 1) **Initiate**
- 2) **Plan**
- 3) **Design**
- 4) **Execute**
- 5) **Maintain**

It can be argued that there are a lot of similarities between some of the stages. Planning and designing are often thought of as a single process, but this is far from the truth.

Throughout these articles, I will be detailing each step, and with each step – I ask you to use the tools I provide to either create a new game, or re-create an old one. By using these tools together with your game-building abilities, you will find

that your games will become easier to create, and present fewer problems towards the end.

Step 1 – Initiate

It is important for you to keep in mind that the initiation phase will not be technical. The primary purpose of this stage is to decide *what* your game will be, not *how* you will build or play it. Decide the genre of the game, as well as the main feeling you want the players to experience.

You will need to choose which platform you will be developing on, and which language or engine you will be using. The purpose of this phase is to simply make decisions. Once a decision is made – stick to it. If you choose to use Game Maker for building your game – don't change your mind halfway through.

Choose which programs you will be using. Include applications for programming, compiling, designing graphics and sound, as well as any resources you plan on using for your game (websites, image libraries etc).

Commit yourself to a deadline. This can be difficult if you are inexperienced, or are attempting something completely different to your usual games.

The important thing to remember is to *be realistic* – if you have studies, work, or other time-impacting duties, then you should factor those in. If you are building your game with or for someone else, it is vital to have their input for your deadline.

Once you have an outline of your project, and feel confident that you can achieve your deadline with the tools you have chosen, you can begin the next phase.

DESIGN



Step 2 – Plan

You've (hopefully) done everything from step 1, and are now ready to start fleshing out your project. There are many theories behind the importance of planning, but it is generally accepted that one should spend 70% of their time planning and designing, 20% executing (content creation), and 10% refining, polishing and maintaining. This may sound like a huge waste of time, but it is during this phase that the heart of your project will take place.

Let's begin then. Determine exactly what will happen in your game. Who is the main character? Why is he or she in the game, and what is their purpose? Provide overviews for the game's storyline (provided it has one). Include dialogue descriptions, inventory items, weapons, armour, NPCs and locations. You will also need to outline any puzzles that your game may have.

Document *everything*. You should be drawing mock-ups of each level and any graphics. It is important not to get bogged down with details at this point, however.

You should also be pre-coding the larger or more complicated aspects of your game. There is no need to go into the nitty-gritty just yet – but you should be at least outlining the basics of your code – such as where to use OO, and where to use procedures. You should also list any particular problems you foresee.

That brings us to the end of part one of project management. I strongly advise you to complete both stage 1 and stage 2, and to not begin *any* coding or graphic design just yet. In the next issue, we will go into the details of the design stage, in which you will delve a little deeper into the creation of your game, and will be essentially creating the whole game on paper.

GEOMETRIX



MOBILE GAME DEVELOPMENT IN JAVA



PART 3: SAY IT WITH SPRITES

Welcome back. As promised last time, this lesson will focus on the Game API found in MIDP 2.0 and later. This has the benefits of providing us with a couple of 'free' abilities like collision, a prebuilt sprite system and simplified input handling.

The first thing we need to do is actually import the Game package, we do this by importing `javax.microedition.lcdui.game.*` at the top of our source file.

Now we must make our `TutorialCanvas` class a `GameCanvas` subclass. To do this we first replace `extends Canvas` in our class declaration with `extends GameCanvas`, and then add `super(true)` as the first line of our constructor. We do this because `GameCanvas` has a boolean parameter in its constructor controlling how input is handled. We also slightly change the way we handle drawing by renaming our paint method to `doPaint`, and replacing the calls to `repaint` and `serviceRepaints` with calls to `doPaint` and the `GameCanvas` method `flushGraphics`. As you may have guessed, this automatically makes use of double buffering if the device supports it. All this changes can be seen (bolded) in Listing 1.

Now we're going to replace our simple graphics sphere with a bouncing sprite. We need an image of a ball. I created my 10x10 pixel smiley-ball using The Gimp (free from www.gimp.org), so that I could have a transparent background. You can use Windows Paint if you like, but your background will be solid, just be sure to save it as a .png file called `ball.png`. Copy that file to the `res` directory in your project directory (for example `C:\WTK22\apps\Tutorial\res`).

In the `TutorialCanvas` class, we add a `Sprite` member `ballSprite` and in the constructor we assign it a new sprite object initialized with an `Image` created from `ball.png` (notice the leading '/' in the filename) and set its initial position.

We can also move the `ballX` and `ballY` declarations into the game loop, since the sprite itself can store those values. In the game loop we modify our bounce code to take the sprite's size into account and update the sprite's position with the `setPosition` method. Finally we replace the arc drawing code from the `doPaint` method with a call to `ballSprite`'s paint method. You can see all of these changes in Listing 2.

Listing 1

```
Listing 1. TutorialCanvas updated to extend GameCanvas.
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;
import javax.microedition.lcdui.game.*;

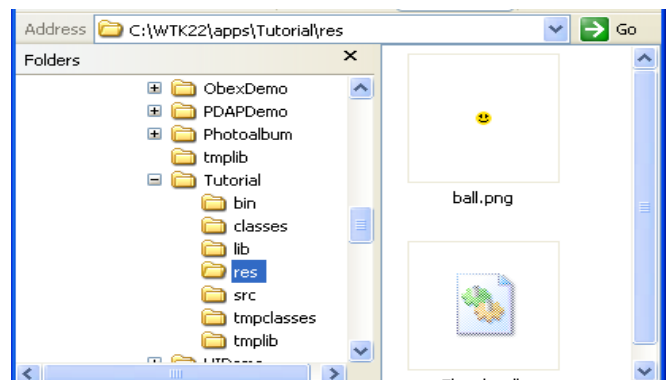
//Main MIDlet class
public class TutorialMIDlet extends MIDlet
{
    ...
}

//Canvas class
class TutorialCanvas extends GameCanvas implements Runnable
{
    ...

    //Constructor
    public TutorialCanvas()
    {
        super(true);
    }

    //run the game loop
    public void run()
    {
        while(!exit)
        {
            ...
            //paint everything
            doPaint(getGraphics());
            flushGraphics();
            ...
        }
    }

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



Img1ResDirectory.bmp: Copy ball.png to your res directory.

Listing 2

Listing 2. TutorialCanvas updated to extend GameCanvas.

class TutorialCanvas extends GameCanvas implements Runnable

```
{
    ...
    static int ballVX, ballVY;
    static Sprite ballSprite;

    //Constructor
    public TutorialCanvas()
    {
        ...
        try
        {
            ballSprite = new Sprite(Image.createImage
("/ball.png"));
        } catch(Exception e)
        {
            e.printStackTrace();
        }
        ballSprite.setPosition(1, 1);
    }

    //run the game loop
    public void run()
    {
        while(!exit)
        {
            //update our ball position
            int ballX = ballSprite.getX();
            if(ballX < 0 ||
               ballX + ballSprite.getWidth() > getWidth()
            )
            {
                ballVX = -ballVX;
            }
            ballX += ballVX;
            int ballY = ballSprite.getY();
            if(ballY < 0 ||
               ballY + ballSprite.getHeight() > getHeight()
            )
            {
                ballVY = -ballVY;
            }
            ballY += ballVY;
            ballSprite.setPosition(ballX, ballY);

            //paint everything
            ...
        }
    }

    //paint the canvas
    ...
}
```

And that's it, instead of a plain ball we have a bouncing sprite. You could replace the ball image with anything you like, maybe you'd like your girlfriend's face bouncing around on your phone?

By now you must be getting fed up with bouncing objects thought, so next time we'll add some player input and add lives.

Until then, there is loads of room to clean up and improve this code, and you're just the game coder to do it!

FLINT



THE TECH WIZARD



A LITTLE BIT ABOUT ...

COMPUTER MEMORY SYSTEMS

In order to become comfortable with programming complicated applications or games, a certain amount of familiarity with the architecture of computer memory systems is required.

MEMORY TERMS

The smallest value for memory on a computer is a bit, which can only contain a single value of 0 (off) or 1 (on). However, the preferred method for working with data on a computer is to use the byte, which is a value that is 8 bits long. Other memory amount terms based off of the byte are the kilobyte (KB), which is 1024* bytes long, and the megabyte (MB), which is 1024 kilobytes long.

All data on a computer is stored in its memory. This memory is separated into two different types: **random access memory** (RAM) and **hard disk memory**. RAM is used by currently running programs to store their active data into. Hard disk memory is used to store more long-term data, like the programs themselves as well as the data resources that they need to run with.

RAM

When a program first runs, it reserves a portion of the computer system's RAM for its own use. This RAM is then further divided into two separate sections of usable memory, which each have different uses: the **local stack** and the **global heap**.

LOCAL STACK

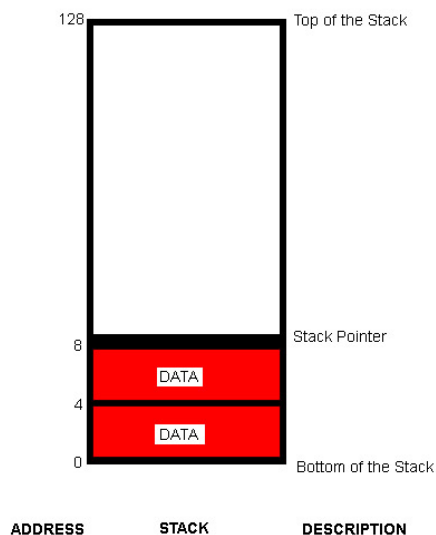
The local stack is used to store data that is created inside of functions. Its size can be any value, providing that it meets the demands of your program. Usually it ranges between 64 KB and 2 MB, which makes it relatively small compared to the size of the global heap.

Data that is created inside of a function (excluding global memory allocation), is considered to be **local** to that function alone. That is, it only remains valid while you are inside of that function, or go into sub-function (a function called from within a function).

Since all functions in your program share this same local stack memory, some behind the scenes trickery is necessary in order for there to be enough space in the stack for your program to work with.

This is achieved through the use of something called the **stack pointer**, which is used to keep track of the current location of the top of the stack. Each time data is stored on the stack, the stack pointer is moved to be after this new data.

Diagram 1:



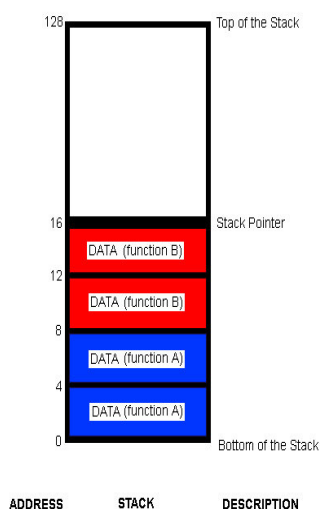
Whenever you enter a sub-function, the current location of the stack pointer is saved. Now, any variables created inside of the function are stored on the stack after any that were created inside of the calling function, and the stack pointer is moved along accordingly.

When the sub-function is exited, the stack memory that was used becomes available for use by other functions simply by

* Although the prefix "kilo" is usually associated with the value 1000, in computer terms it is associated with 1024 as it is considered to be a power of 2 value (ie. 2^{10}).

moving the stack pointer back to the where it was before the sub-function was entered.

Diagram 2:



If you travel too far down into sub-functions (like in the case of recursion), you may run into a stack overflow crash, which means that your stack has run out of space. In most cases this can be simply solved by increasing the size of the stack for your program. Be careful, however that you haven't run into a case of **infinite recursion**, which is caused by continuously going lower and lower into sub-functions without ever returning to higher levels.

GLOBAL HEAP

The global heap is used for a few different things. Firstly, any variables that are declared outside of the scope of a function are placed into this memory. These are known as **global** variables. Secondly, any dynamic memory allocation requests that are made by the program get their memory from here.

The size available for the global heap on most systems is regarded to as the amount of available RAM that the computer has, minus the amount of memory needed for the local stack, plus any additional **virtual memory*** that the operating system memory manager may have.

This makes it quite a bit larger than the local stack, and in most cases the amount of global heap memory used by an application to store its data will be anywhere from 2 MB to 512 MB.

MEMORY ACCESSING

In order to store data, you need to create a **variable**. Variables are the simplest form of data structures for use in computer programming. They generally come in three main types: single **bytes**, multi-byte **integers**, and multi-byte **floating point values**.

When you create a variable, you are telling the compiler reserve a piece of memory for your data, either on the local stack or the global heap. In order to gain access to your data, you need to know where in memory it is stored. In computing terms, the location of data in memory is known as its **address**. These data addresses are represented by numbers, which are simply offsets (in bytes) into the computer's memory for where your data is located.

In most cases, you don't care about the address at which your data is stored, as the compiler handles that for you behind the scenes. To accomplish this, the compiler uses your created variable as an alias to where in memory your data is actually located. In other words, the compiler uses your variable like an address, to know where to go to in memory in order to retrieve your data.

There are times, however, when you do need to have direct access to your data in memory. A good example of this would be if you wanted to allow a function to modify data that was created locally inside of another function. If you were to send the data that you wished to be modified into the function, you would only be able to read what the data is, and would not be able to modify it. However, if you instead sent the address of the data to the function, you will now have access to the actual location of the data in memory, so you can read or write to the data as you wish.

In order to do this, you need to find and store the address of your data into a variable. This new variable, generally referred to as a **pointer****, does not contain your actual data, but instead just tells you where your data can be found in memory. In essence, it works in a similar fashion to the stack pointer from above (note the use of the word "pointer").

The actual method of using the address stored in your pointer variable will differ depending on what language you are programming in. Generally, you will apply an operator to the variable that will extract or **dereference** the data from the pointer, and allow you use it.

CONCLUSION

If you can understand how data is stored in memory and how that memory can be accessed through use of addresses and pointers, then you've got past one of the most difficult concepts in the entirety of computer programming. Now you have the power to be able to go and start creating much more complex and advanced game systems and algorithms!

However, keep in mind that using addresses and pointers is not without its own dangers. It can be very useful to have a value that can be used by anything and at any point of your program, to either view or modify specific data.

The downside is that you can easily destroy or overwrite that data, so be sure to use pointers and addressing carefully and with responsibility!

COOLHAND

* Virtual memory is "simulated" RAM where the operating system memory manager uses hard disk space to store program memory data. It is usually quite slow to use, as the memory manager has to read and write to the physical hard disk.

** The term came about from the fact that the variable "points" to where in memory your data is located.

//Someone asked me why I like Game Development. This is what I told them.

```
Import mind.start.*;

It all begins with the mind,
{
    The passion();
}

If(you're not of those kind)
{
    then it's
    not for the cash-in;
}

for(love = 0; love<infinity; love++)
{
    the pleasures of creating
    gaming
}

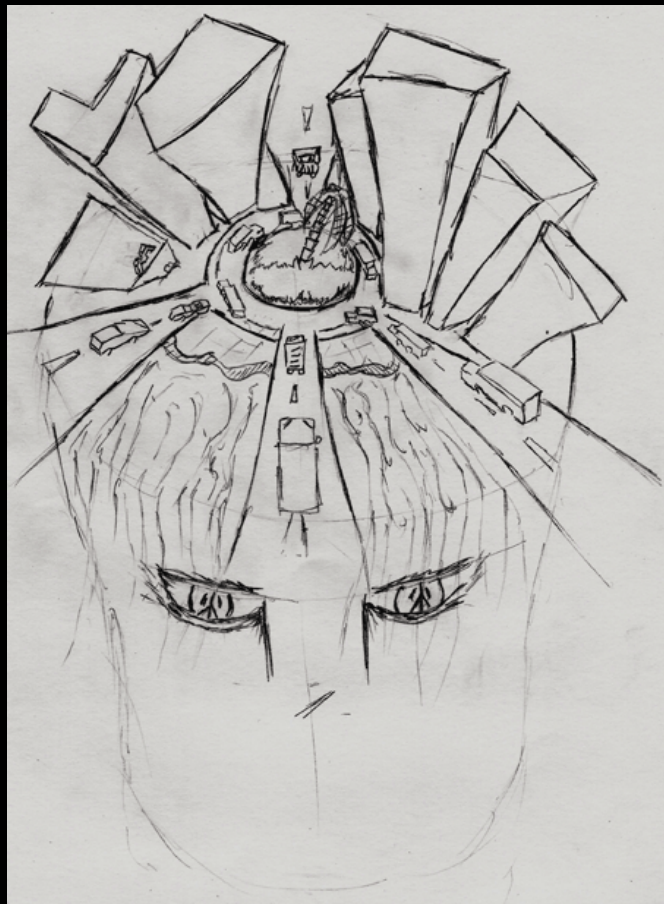
is > than anything else();

while(above is true)
{
    I'll continue
    With the
    Click click click
    Of creating my dreams;

    Ideal = true;
}

/*Build Build Build
Draw it
Make it
Design it
My own
Mine
My ideal!*/

If(complete == true)
{
    Silently close
    The completed
    Rejoice in personal
    reward
    To start again
    With something more();    //!!!
}
```



TAILPIECE

TR00JG

www.itintellect.com
www.itintellect.com

Durban
Ph 031 277 2000
info.dbn@itintellect.com

Bryanston
Ph 086 1 484 484
info.gp@itintellect.com

Cape Town
Ph 021 421 8555
info.ct@itintellect.com

Richards Bay
Ph 035 789 3115
info.rb@itintellect.com

Pietermaritzburg
Ph 033 386 6057
info.pmb@itintellect.com

Bloemfontein
Ph 051 447 3635
info.bfm@itintellect.com

DON'T PLAY GAMES...

B A: DEGREE IN GAME DESIGN

B S C: DEGREE IN GAME PROGRAMMING

N C I T: NATIONAL CERTIFICATE IN INFORMATION TECHNOLOGY

i.t. intellect
COMPUTER **GAMING** SOLUTIONS



O N L I N E

devmag.googlepages.com