ISSUE 9 2006

DEV.MAG

CREATE • DEVELOP • EXPERIENCE

DEVVING IN DURBAN

GAME MAKER COMMUNITIES

8 GOLDEN RULES OF GAME DEVELOPMENT

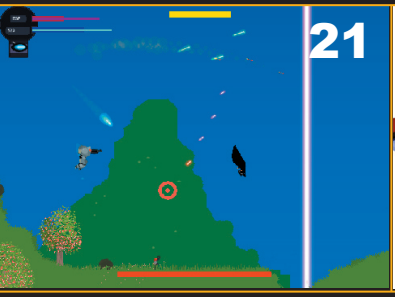
SOUTH AFRICA'S FIRST GAME DEVELOPMENT MAGAZINE

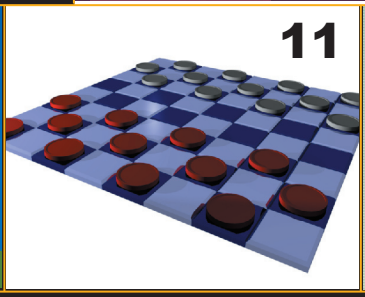
REVIEWS : THE CLEANER. FEATURE : DURBAN DEVLAN. TECH: DATA STRUCTURES 3

Cover: "The Cleaner" by Darthlupi

THE GAME DEV. POWERHOUSE

CONTENTS





ED'S NOTE

No lies -- this has been quite a month. There's a big bad boil-up near the end of every year, with this annum being no exception. During the production of this issue, work commitments took from us one of our dear designers, several of our writing staff and six packets of greatly cherished peanuts (which, the witty may suggest, are used to pay these staff in the first place).

Thankfully, the scantness of this issue is going to be cancelled out by the size of our two-month special, due for release in late January and promising over 50 pages of game development goodness. Why late January? Well, like everyone else, Dev. Mag enjoys a holiday now and again, and our brief sabbatical from the field will hopefully do good in refreshing us for another year of writing. So, be sure to stick around, and don't worry -- we won't be dropping off the face of the earth. Not for longer than a month, anyway.

From the looks of things, Durban is rapidly heating up as a secondary base for local Game.Dev activities, moving from its successful D# Hotlabs onto its very own DevLAN held in November. Two prestigious Game.Dev figures were there to give talks and host workshops over the course of two days, much to the delight of local devvers. Read more about this rewarding experience in this month's featured article.

Adios, enjoy the holiday, and we'll catch you early next year!

Deputy Editor
Rodain "Nandrew" Joubert

DEV.MAG
CREATE • DEVELOP • EXPERIENCE

THE STAFF

DRIVER'S SEAT
Stuart "GoNzO" Botma

PASSENGER'S SIDE
Rodain " Nandrew " Joubert

FURNISHERS
Brandon "CyberNinja" Rajkumar
Paul "Higushi" Myburgh

ENGINEERS
Simon "Tr00jg" de la Rouviere
Ricky "Insomniac" Abell
William "cairnswm" Cairns
Bernard "BurnAbis" Boshoff
Danny "dislekcia" Day
Andre "Fengol" Odendaal
Heinrich "Himmler" Rall
Matt "Flint" Benic
Luke "Coolhand" Lamothe
"Skinklizzard"

WEB ACCESS
Claudio "Ch1ppit" de Sa
Robbie "Squid" Fraser

WEBSITE
www.devmag.org.za

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

This magazine is a project of the South African Game.Dev community. Visit us at
www.gamedotdev.co.za

DIGITAL STOMPIES

NEWS

3D fun with Panda

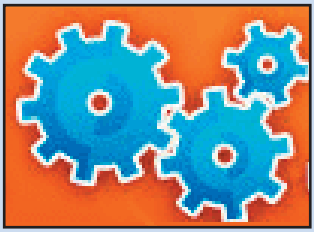
http://panda3d.org/

Panda3D version 1.3 has just been released, adding lots of goodies to the growing 3D engine. Panda3D is a C++ library with Python bindings, meaning that it is callable from either of the two languages. It focuses on rapid game development with a short (and assumedly painless) learning curve, and has already succeeded in making many 3D projects extremely quickly. Panda3D was originally used by Disney to create their MMO Toontown, and was first released as free software in 2002.

New dev-oriented community site

http://www.greatgamesexperiment.com/

The Great Games Experiment is a community aimed at both gamers and game developers, trying to get the two groups to network and gain exposure to one another. Given an online "space", game developers or groups can profile themselves, meet other people in the industry and generally have a new platform on which to promote themselves. This website was developed by Garage Games (www.garagegames.com) and is currently in an invitation-based beta state. Newcomers are, however, allowed to request a beta invite and sign up for the newsletter.



Podcasts on mobile technology

http://www.khronos.org/podcasts/

The Khronos Group has recently started on a series of Mobile Media Developer Podcasts. The group, which describe themselves as "the developers behind the industry standards for 3D, 2D, video and audio for mobile devices" focus on explaining the new technologies in their podcasts. Two podcasts have been released so far, dealing with OpenKODE and OpenSL ES.

New Torque release

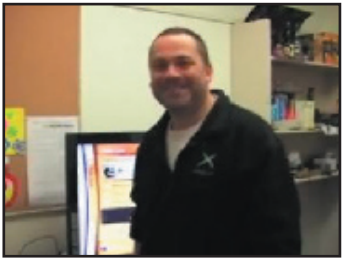
http://www.garagegames.com/blogs/33542/11746

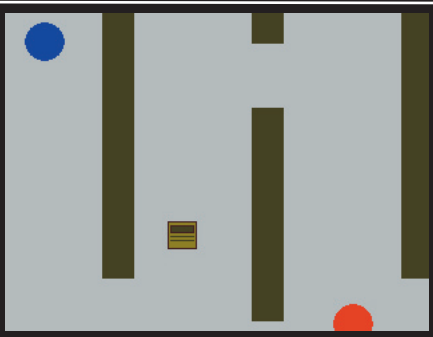
Take a look here for a detailed outline of the new Torque Game Builder release, version 1.1.3. The windows version of the release has already been made, and includes a new documentation framework, a new text object, alignment tools and, of course, a major list of bugfixes which are sure to make a lot more users happy. If you don't know about the Torque Game Builder already and would like to find out what it's about (or benefit from a 30-day free trial), take a look at http://www.garagegames.com/products/torque/tgb/ for more info.

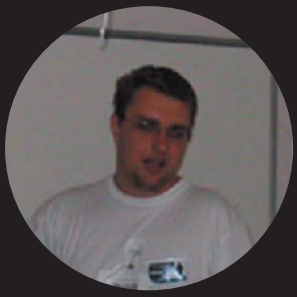
Some XNA eye candy ...

http://channel9.msdn.com/Showpost.aspx?postid=257928
http://channel9.msdn.com/Showpost.aspx?postid=261254

The Channel 9 forums recently blessed XNA enthusiasts with a two-part video series on XNA called "Looking at XNA". The first part of the series is an in-depth interview with Boyd Multerer about the tool, including the idea behind XNA and the Studio IDE which it uses. The second part gets XNA in action on a 1080p display, showing off its immense power, and also consists of a bit of chit-chat with Frank Savage, one of the people who worked on the old Wing Commander titles.



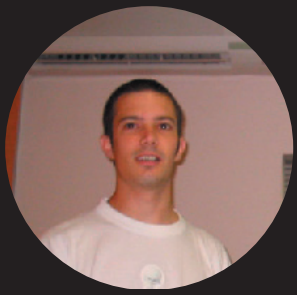




Dozens of game developers.

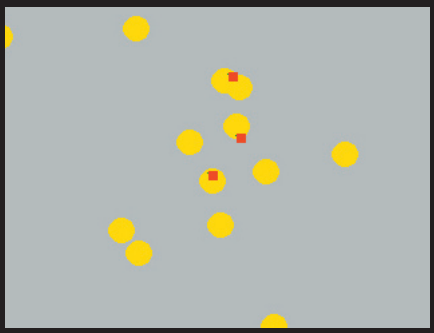
Two Game.Dev gurus.

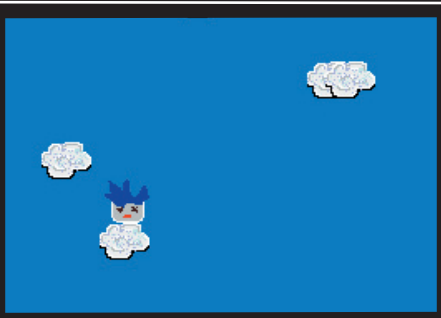
One Durban DevLAN.

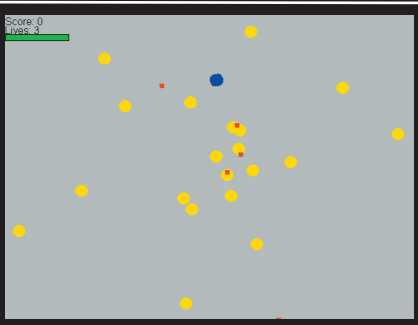


FUN AND GAMING IN *DURBAN*

FEATURE



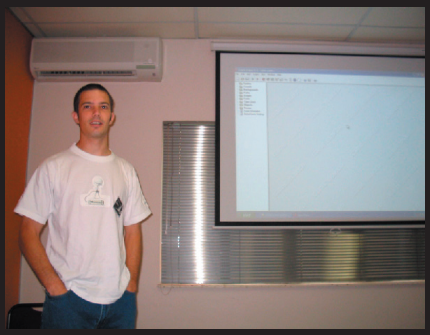


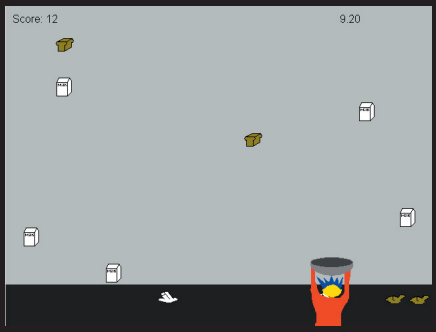


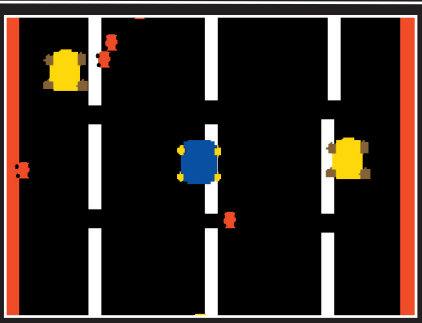
On the tenth and eleventh of November, Durban saw its first DevLAN/Game. Dev workshop. Held in the local branch of I.T. Intellect, it received a score of budding and enthusiastic developers who gathered to listen, watch and most of all absorb valuable and insightful ideas from two of South Africa's leading game development gurus, Danny Day and Miktar Dracon.

The weekend kicked off on Friday evening with an introductory presentation by Danny on game development in SA. The avid listeners were quickly brought up to speed on the local situation: to grow the industry in this country, games are needed – good quality games mostly, but games nonetheless. Not ideas, as the lack of local publishers means that local companies are hard-pressed to sell game ideas. This little speech put a purpose in these avid developers' hearts – they were going to save SA's game development industry by making the best, most ultra awesome game ever. But, rather than let them all run off, cause havoc and ultimately get nowhere, Danny moved on to the process of sound game design by giving the attendees a challenge. This challenge was to come up with an idea and design a board or card game so as to emphasize the importance of good design and planning. To help them all out with this strenuous challenge he even gave a small talk on game design to nudge them in the right direction.

Day two, Saturday, started bright and early at nine o'clock. This was when the real fun began. After a quick introduction to the basics of Game Maker 6.1, a very powerful little game creation tool, along with a demonstration of just how much could be done with it, the developers were once







again challenged, but this time by Miktar. The challenge was simple, or so it seemed: one had to write a game about a day in the life of fictional character John. How hard could that be? Well, it turned out to be tougher than the developers thought, not because of a lack of inspiring and awesome ideas but rather the lack of experience working with Game Maker. However, with Miktar taking the reins and setting up a projector to do a few tutorials in the one lab and Danny the same in the other, this problem was overcome.

Lunch break. Ahhh, that wonderful point when you stand up, rub your eyes to try and remove the purple spots of retina burn and take a break from strenuous coding to get food. The developers dispersed: some got themselves Steers burgers, others pies, chips and gravy, and in Miktar's case a coke and a chocolate bar that he couldn't finish. Being the kind soul that he is, he generously gave his half eaten chocolate to a ravenous (or so it seemed) growing boy who needed the energy, on one condition (yes, he puts conditions on giving away food) – the chocolate was not to be sold on eBay (it's going for free but shipping to anywhere will cost R250).

Having replenished their energy, it was back to work for the developers. Interestingly, Miktar's tutorial digressed from the life of John to a tutorial on how to make a classic top-down shooter: blue dot against a horde of yellow dots. In his defence though, not only was John allowed to dream, but the chain reaction caused by the power pickup was totally and completely awesome. Think power bullet hits enemy releasing 10 others all in random directions which hit other enemies and so on and on. Many diverse and interesting ideas were used for Johns life: for waking up, one group decided to have the player throw John off of his bed and try and hit an X on the floor with him, while another game had the player firing neurons into Johns synapses to wake him up. For work, John was anything from a bling thief to a policeman to the only thing between this world and the rabid invading teddies.

All in all, the weekend was a great success with fun being had by all, in the process of learning how to make a game… That's the power of the level 90 arch devvers Danny and Miktar, turning normal innocent humans into rampaging devlings, but hey – who doesn't want to be a devling with the chance of one day becoming a great arch devver?

SKINKLIZZARD

REVIEW

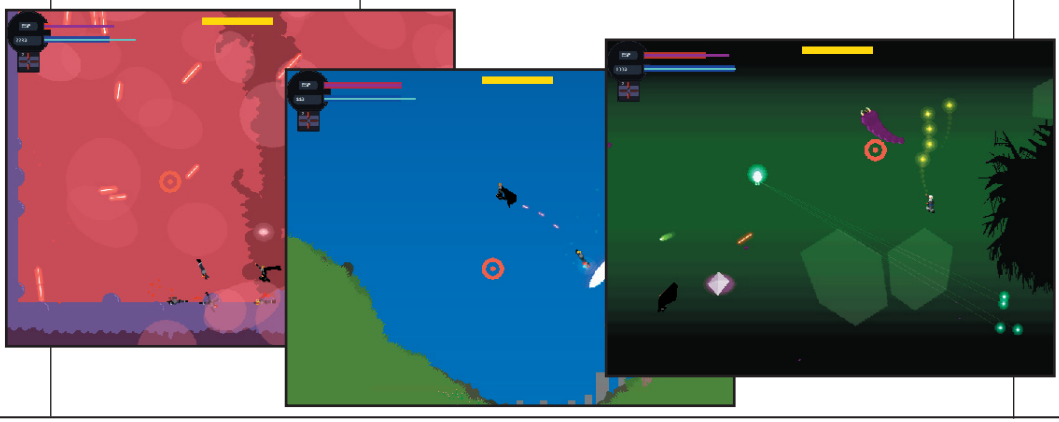
The Cleaner

Despite its somewhat bland name, The Cleaner is a prime example of what can be achieved by talented developers with the right tools. It oozes quality and has a length to rival, and perhaps even surpass, some commercial games. And the best part about it? You can get it for free!

The Cleaner sees the player adopting the role of a mage whose goal is to rid the 'multiverse' of an evil race that seeks to obliterate the friendly race of magic wielders. Players will travel throughout multiple unique worlds, all presented in an incredibly impressive and stylish manner. Every detail is there, right down to the way the player's cape flows behind him as he flies through the air.

The game's storyline advances by use of numerous cutscenes, which set the scene for the player. The back-story quickly fades into obscurity, though, and the focus is quickly established on the action. And action there is! The game constantly throws challenging situations at the player, and everything from the terrain to the boss fights becomes part of the challenge.

As the player progresses through the game, battling enemies and solving puzzles, he or she is rewarded with experience points which are in turn used to purchase new offensive and defensive spells or upgrades. These spells are varied enough so as to provide strategic uses for all



of them. In addition to your magical arsenal, the player also has the ability to fling objects at enemies using telekinetic abilities.

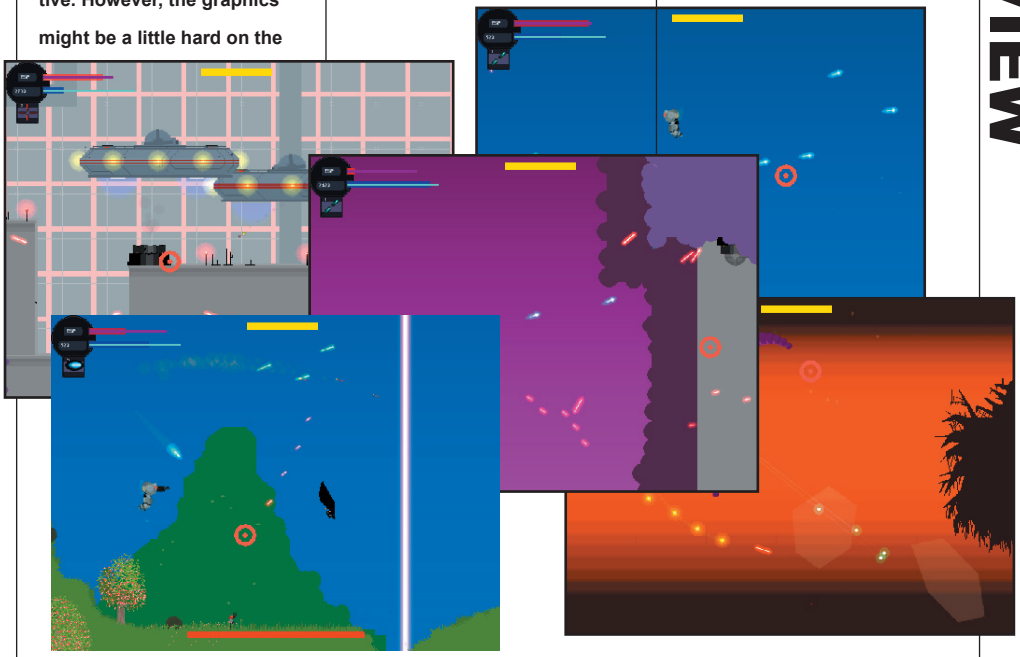
The game also includes totally unique artwork, musical score and sound effects. The music works well to set the often tense atmosphere, and doesn't become repetitive. However, the graphics might be a little hard on the eyes and the sounds effects can become harsh and annoying. Despite these minor shortcomings, it is the exquisite style in which the whole game is assembled that truly makes it shine.

It is a true boon to gamers that the game's creator, Darthlupi, decided to release this game for free, since it could be easily sold as a commercial title. The Cleaner is probably one of the best games ever to be created using Game Maker, and is a true showcase of the framework's potential.

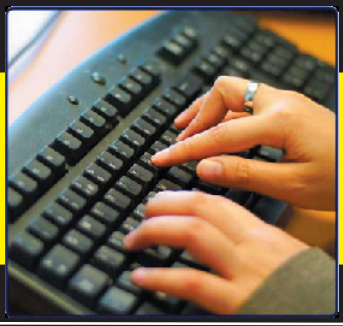
CH1PPIT

Developer: *Darthlupi*
Website: *http://darthlupi.com/*
Genre: *Action*
Year: *2006*

REVIEW



8 GOLDEN RULES



DESIGN

Applying Shneiderman's 8 Golden Rules to Game Design

In the world of IT, there is a whole branch dedicated to a topic called Human Computer Interaction (HCI). The term HCI is fairly self-explanatory: it's basically the study of how people interact with computers and how well the interface between the user and computer is designed. Another good way to phrase it is simply 'user-friendliness'.

In the field of HCI, a man by the name of Ben Shneiderman created 8 rules to be applied when designing the user interface. While they seem relatively simple to most of us when applied to creating business software, they can also be applied to game design with great effect. Let's take a look at the rules and how they can improve our games.

Strive for consistency

When a player plays a game they get used to and comfortable with the controls and gameplay mechanics as time progresses. While it's good to create a sense of advancement by adding things as the player progresses, don't go overboard. Sometimes, in the process of making a game, you test it so often that it becomes too easy for you and you are tempted to make things harder. As you design the learning curve and progression of difficulty, always bear the inexperienced player in mind.

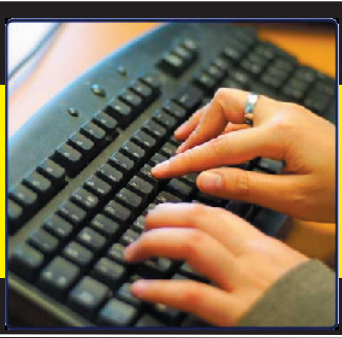
Enable frequent users to use shortcuts

As a player invests time in a game, they improve and seek ways to achieve things quickly with less effort. Strategy games often offer keyboard shortcuts that allow players to do things quickly and on-the-fly, all while giving them an advantage. Things like mouse gestures make surfing the Internet so much more of a joy and it can have the same positive effect on a game.

Offer informative feedback

Players need to know what's going on in the game; things like ammo and health are important to a player as it influences their decisions. Make sure your system provides all the statistics a player needs in a clear and concise manner. A common, useful method is to display things in the corner of the screen. Recently, some developers have decided to remove the on-screen display to improve realism. While it's great to be different, an alternate, effective solution which still depicts vital stats must be put in place.

8 GOLDEN RULES



Let the user know when they have completed a task

Ever wasted time in a game searching for something when you've already found it? It's not fun and can ruin the experience. Make sure things are clearly designed so as to let the player know if they have been successful or have failed.

Offer simple error handling

Offer a struggling player help when it seems he or she has moved astray. Sometimes a player attempts a mission, but doesn't see the first vital check-point and runs off in the wrong direction. Some subtle guidance can give them much needed help. For example, if the player has spent more than a certain amount of time running around aimlessly, give them an arrow leading them to the correct destination or introduce a 'side-kick' character who can point them in the right direction. It seems so simple but can prevent the player from becoming frustrated and giving up.

Permit easy reversal of actions

Everyone makes mistakes, and whether the player has accidentally pushed the wrong tile in a puzzle game or has missed the second-to-last golden star while jumping around in a time trial, offering something as simple a quick-restart option is a useful anti-frustration measure. This gives the player a break and allows them to try again.

Provide a sense of user control

Gaming is an interactive form of entertainment, so players like to feel like they are controlling the action. While cut scenes are necessary to advance the plot, try and keep them at a minimum. You could even be really innovative and give the player some form of control in a cut scene.

Reduce short-term memory load

Humans have a limited short-term memory, so don't over burden them with things to remember while playing. Not many people are going to remember some little detail like the security code they used 2 hours ago, not without prior warning or some kind of reminder. While you may have spent many hours coding and know the game inside out, the player doesn't have the same experience as you.

I hope that these points have given you some ideas and things to keep in mind the next time you design a game. Try to remember that, while designing, the player's experience is of utmost importance. These points can help improve the experience, and a good experience results in a happy gamer.

INSOMNIAC

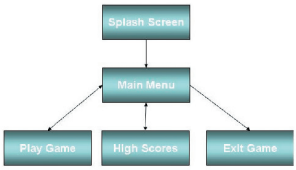
State Management

While a game is running, it will pass through a number of global states. These states dictate what the player is currently seeing and experiencing within the game. It is important to manage game state and understand the flows between the different game states to ensure that the player always has control over their in-game experience.

State Management refers to many different aspects within a game. The weapon selected by a player in an FPS is an example of state management while the screen the player is currently seeing is also an example of state management, often referred to as the Game State. This article will examine various methods of managing Game State within a Game's State Management framework.

Game State can be managed as part of the standard game loop. This is often used when a single input method is being managed or when the number of state is kept very small. To enable State Management within the Game Loop a global state variable can be set and evaluated each execution of the game loop. Based on the currently active state the relevant rendering and game update procedures can be called, or even on a very small game the relevant rendering and updates can be done within the Game Loop itself.

A separate Class can be created to manage the game state. A class can be created for each game state and the main game loop will only execute the relevant methods of the active state. With inheritance and polymorphism, the implementation of a class-based state system is very easy. A base class can be created that contains virtual methods for the standard functions such as Rendering, Update and Input. As new states get added to the game the base class is extended to manage the new requirements. When relevant the Game State is marked as active and is immediately used within the game. A class based state management system will typically ensure that each state is atomic and knows nothing about any other state in the game. This therefore sometimes requires other mechanisms to pass messages between states



Basic Game States

for things like highscores or to save games etc.

Game Maker uses rooms to represent game state. Each room is effectively a class that manages the player's current in-game experience. As the game state changes and the player moves into a new room, the old room no longer affects the player or the screen in any way and the new room now becomes the item controlling the game experience. Game Maker ensures that the Input, Update and Rendering functionality for each room is completely independent of each other room.

Whichever way State Management is implemented within a game, certain aspects must be included in the design. If these aspects are not included as part of the design they will impact the later game implementation quite heavily. If these aspects are included as part of the original State Management design, the effort required to extend them for each game state is minimal.

Each state needs to be able to render the required graphics to the screen. Each state therefore needs its own render method. If state management is managed within the game loop the rendering of the graphics may be a custom procedure called based on the current state. A class-based state management system will implement different rendering processes for each state within its class.

The management of input for each state can sometimes be more complex than the rendering as the program typically receives the input messages and needs to pass them to the relevant state management system. The class based state management can implement a custom method within the class to manage the input handling, while the game loop based state management system may define independent functions to manage the input or in simpler games may do all input management within a single method. Input management can also be implemented using events that the various state management systems subscribe to.

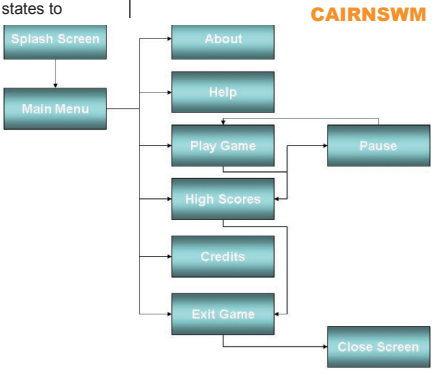
As the player interacts with the game the current game state needs to react to the player. Each state therefore needs to implement its own update method that allows the various entities on the screen to move, collide etc. Each state once again needs to implement the update process to only update the items relevant to the current state.

One state that is often forgotten or ignored is the Pause state. The pause state is a special state in that it either represents a static screen or it keeps the screen of the previous state and waits until the pause state ends. When the pause state ends the game should resume with the previous state active and as if no pause had happened.

Based on events and actions within the game states must change. Game Loop state transitions are easy as the current state parameter only needs to be updated with the new state to allow the game state to change. A class based state management system has more complex mechanisms needed to implement state transition. Either each pass through the game loop must check for the end of the current state and then initiate the new state, or an event must be raised by the state that is caught by the main game code which controls the transition to a new state.

In many cases state management can be extended to allow states to have initialization and finalization functions. These functions can include things such as game start-up processes, screen fade ins and outs and load game data etc. The design o f the state management system can be done in such a way that the development of new functionality in the game is easy and reusable. A Class based state management system allows the design of generic state structures such as a Movie play class, a Company intro screen, a Please register screen that, with very little rework, can be used over and over again in many different games. A game loop-based system is easier to code and allows the creation of detailed game templates but suffers from not having enough flexibility to adjust older games to newer ideas.

State Management is a key concept needed when creating games. With a good flexible state management system, the options available to the game developer expand and make new functionality easier to add. A well designed state management system can decrease the amount of work required for a new game significantly as the look and feel of the game is already predefined. By spending some time understanding the ins and outs of state management each and every game the game developer creates will be easier, quicker and more complex than the previous game.



Complex States

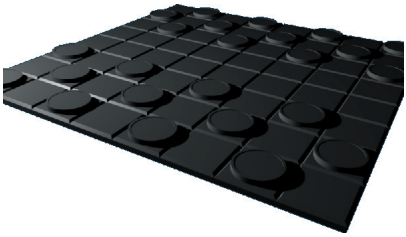
DESIGN

```
public class GameState
    {
        protected GraphicsDeviceManager graphics;
        protected ContentManager content;
        protected SpriteBatch spriteBatch;
        protected GameMouse mouse;
        protected KeyboardState keyboard;
        protected ImageList Images;

        /// Constructor to create a new Game State
        public GameState()
        {
            Cursor = null;
            CursorOn = false;
            mouse = new GameMouse();
        }

        /// Initialize all the required fields of the GameState.
        /// This must be called after the game state is created.
        public virtual void Initialise(GraphicsDeviceManager NewGraphics,
ContentManager NewContent, SpriteBatch NewSpriteBatch, ImageList NewImages)
        {
            graphics = NewGraphics;
            content = NewContent;
            spriteBatch = NewSpriteBatch;
            Images = NewImages;
            LoadResources();
        }

        private TImage _Cursor;

        public TImage Cursor
        {
            get { return _Cursor; }
            set { _Cursor = value; if (value != null) { CursorOn = true; } }
        }

        private NeheFont _Font;

        public NeheFont Font
        {
            get { return _Font; }
            set { _Font = value; }
        }

        private string _GameTitle = "C# XNA";

        public string GameTitle
        {
            get { return _GameTitle; }
            set { _GameTitle = value; }
        }

        private bool _CursorOn = false;

        /// When True the Cursor Image will be displayed.
        public bool CursorOn
        {
            get { return _CursorOn; }
            set { _CursorOn = value; }
        }

        public event StateEventHandler OnEndState;

        /// Check User Input - Call DoInput()
        public void Input()
        {
            mouse.GetState();
            keyboard = Keyboard.GetState();
            DoInput();
        }
```

```
    /// Update the objects etc
    public void Update()
    {
      DoUpdate();
    }

    public virtual void LoadResources()
    {
    }

    /// Call the Draw function
    /// Display the Cursor if neccessary.
    public void Draw()
    {
        DoDraw();

        if (CursorOn & (Cursor != null))
         { Cursor.Draw(mouse.X, mouse.Y); }
    }

    /// Update the game state - Must be overriden by child classes
    public virtual void DoUpdate()
    {
    }

    /// Called when a state is entered for the first time
    public virtual void DoStartState()
    {
        mouse.Clear();
    }

    /// Called when the state exists - Must be overriden by child classes
    public virtual void DoEndState()
    {
        mouse.Clear();
        if (OnEndState != null)
        {
            OnEndState(this);
        }
    }

    /// Override to respond to user Input- Must be overriden by child classes
    public virtual void DoInput()
    {
    }

    /// Draw the state onto the screen - Must be overriden by child classes
    public virtual void DoDraw()
    {
    }

    private bool _Active;

    public bool Active
    {
        get { return _Active; }
        set { bool OldActive = _Active;
              _Active = value;
              if (_Active & !OldActive) { DoStartState(); }
            }
    }

}
```

BLENDER TUTORIAL PART 5

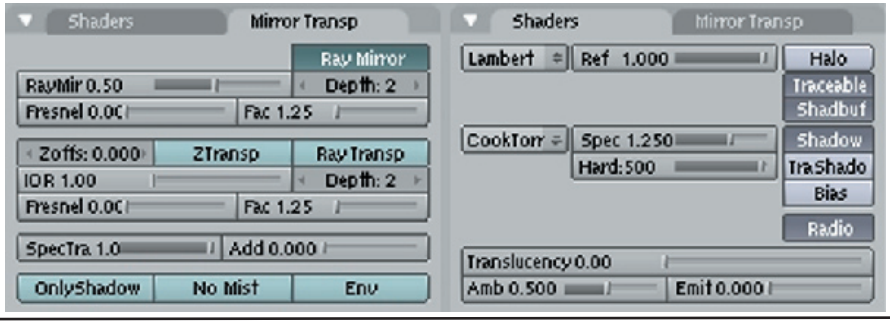
Welcome back. This month we'll be completing the checkerboard we modelled last time. If you followed the last tutorial, you should have the basic design of the checkerboard with pieces. If you don't, go back to last month's tutorial and create the scene or download last month's file from the Dev.Mag website's content section.



While we have a reasonably good and accurate-looking model, a render doesn't do it any justice at the moment. It needs colour, shading and some better lighting. If you used the duplication methods described in the previous instalment, then applying the materials to the correct objects will be easy.

Let's start with the board itself. Select one of the board pieces, go to the materials tab, and create a material. This new material will automatically be applied to any linked duplicates of the piece you selected. We'll set the material properties first, and then come back to colouring the pieces. Firstly, naming the material is often a good idea, especially in larger scenes where multiple materials are used. Change the box which reads 'MA: Material' to another, more descriptive, name. Then use the settings for the material shown in the figure at the bottom of the page.

Once that is done, select one of the board blocks that is not linked to the one you just applied your material to. In the material tab of this new block, select the material you just created from the drop down material box, and then select Add New. This will create a new, independent, material that is based on the settings of the previous one. Give this new material another descriptive name.



Now, for the sake of accuracy, bear in mind that -- in a traditional layout -- the light blocks always alternate from the lower-right corner of the board and the actual pieces are all placed on the dark blocks. Apply the colours with that in mind. I used the following blue colour scheme for the board:

Dark blocks:

Colour: *(R: 0.0, G: 0.0, B: 0.2)*

Specular: *(R: 0.0, G: 0.0, B: 0.4)*

Mirror: *(R: 0.0, G: 0.0, B: 0.6)*

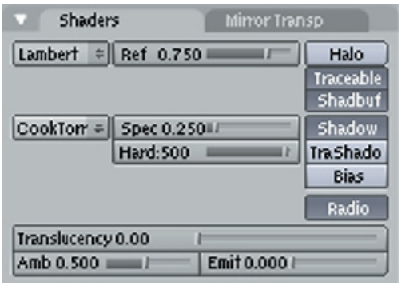
Light blocks:

Colour: *(R: 0.6, G: 0.6, B: 1.0)*

Specular: *(R: 0.7, G: 0.7, B: 1.0)*

Mirror: *(R: 0.8, G: 0.8, B: 1.0)*

We'll repeat the same process for the actual pieces. As before, create a material for the one set of pieces, apply the shading settings, create another material based on it, and then apply the colouring to the two separate materials. The shading settings I used are visible below. Our pieces won't be transparent or reflective, so you can leave those settings at their default values.



I picked the following red/white scheme for the pieces. The mirror value won't be used here, so you can ignore it.

Red Pieces:

Colour: *(R: 1.0, G: 0.2, B: 0.2)*

Specular: *(R: 1.0, G: 0.3, B: 0.3)*

White Pieces:

Colour: *(R: 1.0, G: 1.0, B: 1.0)*

Specular: *(R: 0.9, G: 0.9, B: 0.9)*

We'll also change the colour of the background from the old boring blue. Switch to the Shading Menu, if you aren't still there, and click the World Buttons icon to bring up some new options. All you have to worry about here is the left-most colour block. Click on it and change it to white. You could also type in RGB values like you do with materials.

Finally, we'll modify the lighting for better effect. Select the original point lamp, make 3 linked duplicates and arrange them around the four corners of the board. Make certain that they're pretty high over the board. Set their brightness to 0.4. Disable their shadow casting ability by clicking the Ray Shadow button.

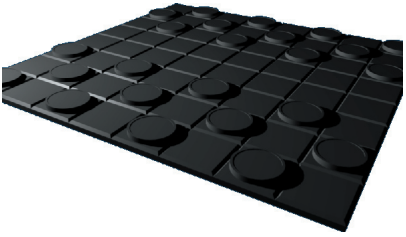
Now add a new spotlight, place it along one of the edges or corners of the board, and point it obliquely across the face of the board. You might need to adjust the Dist value to give the light a little more reach. Then, in the 'Shadow and Spot' tab, increase the 'SpotSi' value to increase the angle of the spotlight so that it encompasses a greater portion of the board. Also change it to cast shadows using

ray-tracing instead of the shadow buffer.

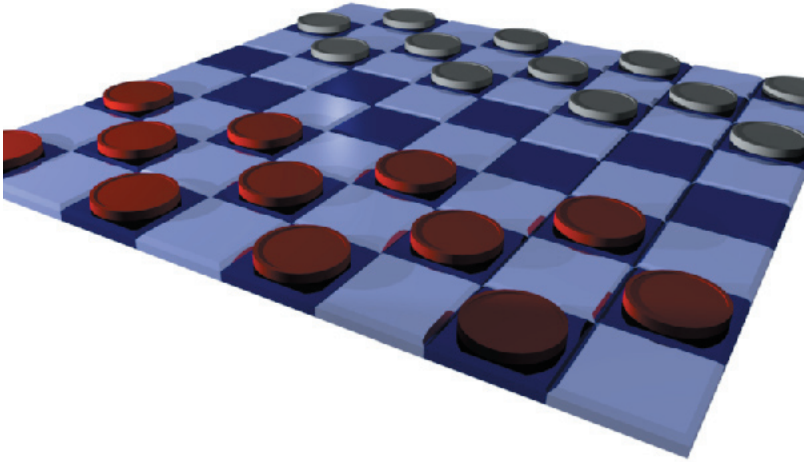
And that's it! Your scene is done, and a render should produce a scene similar to the picture below. Congratulations, you have created a complete scene in Blender. Feel free to make your own additions to the scene and see if you can improve on it. [end.png]

The completed scene can be downloaded from devmag.org.za.

CH1PPIT



What we started with.



Our final scene. Good enough to play with.

THE TECH WIZARD

Data structures part 3: Trees

Trees are a very popular form of data structure, mainly due to their ability to store data hierarchically in a sorted manner. This functionality allows for the data in a tree to be accessed very quickly. Most often, trees are used to store world data for rendering or for processing, but they can also be very useful for general data storage

Basic Tree Structure

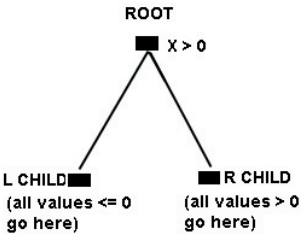
The root level of a tree is made up a single node. This node has some kind of equation(s) or condition(s) associated with it that is used to organize the incoming data, as well as for doing searches when data is requested. Along with these equation(s) are various branches, each corresponding to an outcome of the equation(s). Finally, at the end of each of these branches are other nodes, also with some equation(s)[1], as well as more branches.

This process continues recursively until nodes are reached that can no longer have an equation. At this point, instead of having more branches, any data that has filtered down here is merely stored[2], usually in the form of a linked list.

Binary Trees

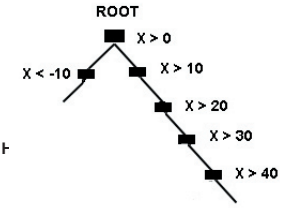
The simplest form of a tree is known as a Binary Tree. This means that each node of the tree has at most 2 child branches that lead to new nodes / data, and they are a very useful way to store data in a sorted method so that it can be retrieved efficiently at a later time.

A very simple binary tree would be one where the root node's equation was (x > 0). This means that all data sent into the tree that was > 0 would be placed into the right child branch of the tree, and all data that was <= 0 would be placed into the left child branch of the tree



However, this would be very inefficient usage of a tree, as we are only dividing all of our data in half. Therefore, if you added 1000 values to the tree and then wanted to find a specific one, you would still have to search linearly through many of them until the one that you wanted was reached.

To expand on this example, the equation used at each node down the tree could be modified so that the data is easily sorted into ranges of 10s. To do this, the node belonging to the right child branch of the root node could have the equation (x > 10), and the node belonging to the left child branch of the root node could have the equation (x < -10). This would continue recursively for as long as we wanted, with each new node adding an addition 10 to its equation.



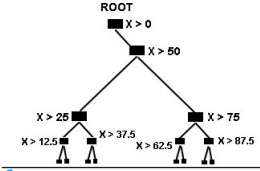
1 -- The equations at each node of the tree are usually derived from the root node equation, modified based on the current level of the tree that you are currently at.
2 -- Many types of tree usage also allow for data to be stored at nodes that do in fact have branches down to lower nodes.

TECH

However, this would lead to a very unbalanced tree. This is because the right child node of the right child node would have (x > 20) as its equation. Given this, we would only be able to add to its right child node and we wouldn't be able to do anything with its left child node as all numbers between 1 and 10 that get this far will remain at this node.

Unbalanced trees are not as efficient as balanced ones, as they are always much deeper than balanced trees, and each level of a tree that needs to be traversed takes up additional time. A more optimal solution would be to pick a maximum value that our number could ever be, and instead of merely increasing the value in our equation linearly, we adopt a divide and conquer method.

In other words if we choose 100 as the maximum value that our number can be, instead of setting the equation of the root's right child node to be (x > 10), we would set it to be (x > 50). Accordingly, its right child node would then have the equation (x > 75) and its left child node would have (x > 25). This would continue until we reached a level that would be deemed sufficiently small enough and not worth dividing again.



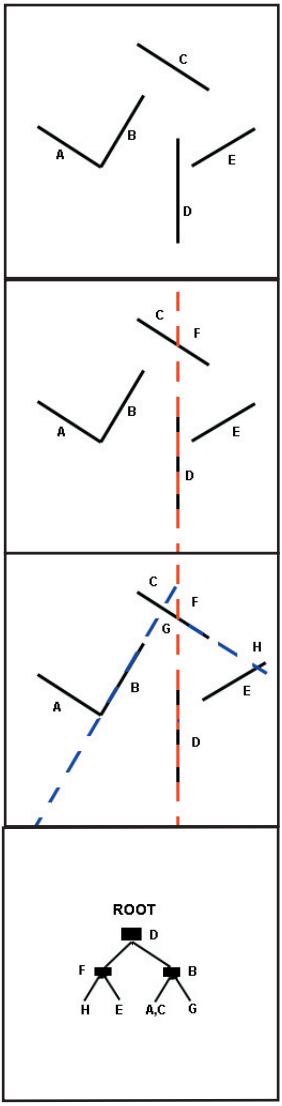
This keeps the tree more balanced and reduces the number of potential searches needed to locate data in the tree.

BSP Trees

Perhaps the most well known kind of tree is a BSP Tree, which stands for Binary Space Partitioning[1]. As the name implies, it is really only a specific implementation of the simple binary trees discussed above, however it was devised to specifically handle the rendering of static world geometry in 3D environments. They have actually been around for many years, but were only made popular by the first generation of 3D FPS games (Doom, Quake, etc.) as a way to efficiently render the visible world.

A BSP Tree works by dividing the geometry in a world so that only the visible portion of the world is rendered at any given time. What makes them different from standard Binary Trees is that the equations used at the individual nodes for creating the tree structure are actually the planes that form the geometry of the world itself. This is why they were very popular for the standard indoor environment FPS games, as most of these worlds were just made up of flat, perpendicular walls.

In order to create a BSP Tree, any plane can be chosen as the "equation" for each node in the tree. However, choosing different planes can provide more balanced and therefore optimal tree creation, so usually some kind of exhaustive testing algorithm is used to try various combinations of which planes to use at each level of the tree.



The reason behind breaking up the world in this manner is so that for any given plane in the world, it is easy to tell which other planes are

1 -- Binary = 2; Space = World/area that you are dealing with; Partitioning = Division

in front of it as well as behind it. This then allows for very straightforward rendering of the world, as given the position and direction of a camera you will easily be able to traverse through the built up tree and at each node check if the plane data contained there is in front of the camera[1].

If it is, you can render that data of the world geometry represented by that plane. The process is then repeated in a recursive fashion down each branch of the tree that the camera is in front of until a terminating node is hit, at which point you have finished rendering all visible world geometry.

KD Trees

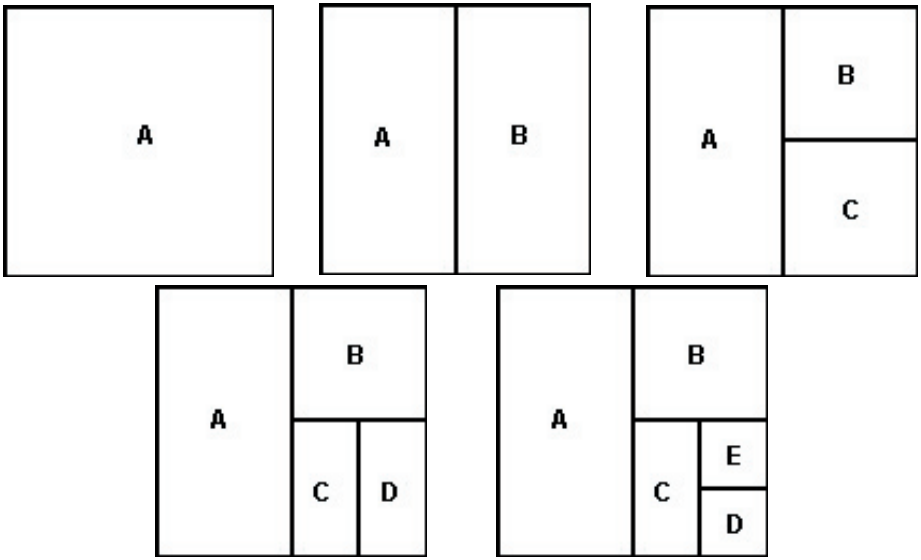
KD Trees are similar to BSP Trees except that instead of using actual world data (ie. the planes of the polygons in the world), they only use planes that are perpendicular to the world axes. Generally, KD Trees used in a 2 dimensional sense (ie. X and Z splitting planes), but can be used 3 dimensionally as well (ie. X, Y, and Z splitting planes)[2].

So, instead of choosing a splitting plane corresponding to some arbitrary world geometry, the world is initially split in the direction of a single axis. It doesn't matter which one, and it also doesn't matter at which point of the axis it is split (ie. (X = 0) and (X = 5) would both work).[3]

Once the world has been divided along that initial axis, each of the new halves is then divided along an alternate axis (ie. if the X axis was used for the first division, then the Y axis would be used for the next). This continues recursively until some kind of size limitation is met, at which point the tree is complete.

Generally, KD Trees are used for storing objects in a game world as it allows for easy searching of ones that are close to each other in the world for such things as rendering, as well as collision testing, etc. The main advantage that KD Trees have over BSP trees is that they do not require the typical indoor FPS style of "walled" worlds in order for them to produce effective and efficient trees.

COOLHAND



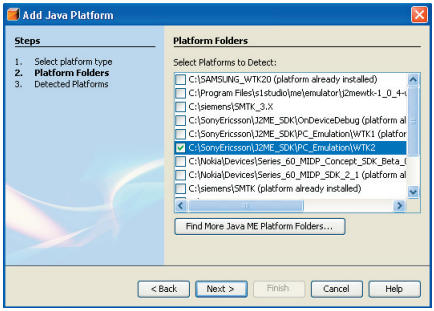
1 -- This is done by using simple algebra and plugging in the camera data into the Ax + By + Cz + D = 0 plane equation
2 -- However, like BSP Trees, KD Trees can be used in both 2D and 3D worlds
3 -- Generally, the axis used as well as its equation, like with BSP Trees, are decided upon by doing exhaustive searching at the tree generation time

Multi-Platformer

For the last couple of months, we have been working under a very utopian assumption that Java's 'write once, run anywhere' ideal is realistic. Even within the same profile and configuration standards, different devices have their own quirks. As a mobile developer, you need ways to deal with these quirks. The first weapons in your arsenal are emulators. We have been using the standard WTK emulator, but most of the major cellphone manufacturers provide their own versions which should more closely resemble their handsets. We will be looking at how to set up one of these emulators and how to have multiple configurations for your project in NetBeans so you can easily build different versions of your MIDlet.

The first thing you need to do is install a manufacturer SDK such as the Sony Ericsson one mentioned last time (http://developer.sonyericsson.com/site/global/docstools/java/p_java.jsp). Once installed, select Java Platform Manager from the NetBeans Tools menu. In the platform manager, click Add Platform. Change the radio selection to Java Micro Edition Platform Emulator and click next. NetBeans will now search for any compliant emulators and list them for you. Check the emulator(s) you want to install and click Next and then Finish. The platform will now appear in the list in the platform manager, you can close it.

Now in the Tutorial project's properties dialog (accessible by the right clicking on the project), notice the Project Configuration section at the top of the dialog. At the moment you should only have DefaultConfiguration and Add Configuration in the dialog. Select Add Configuration and in the dialog that pops up, give your new configuration a name appropriate to your newly added emulator (for example SonyEricsson) and click OK. Now in the list on the left, select the Platform group, uncheck Use Values from "DefaultConfiguration" and choose your newly added platform from the



Adding a new platform to NetBeans

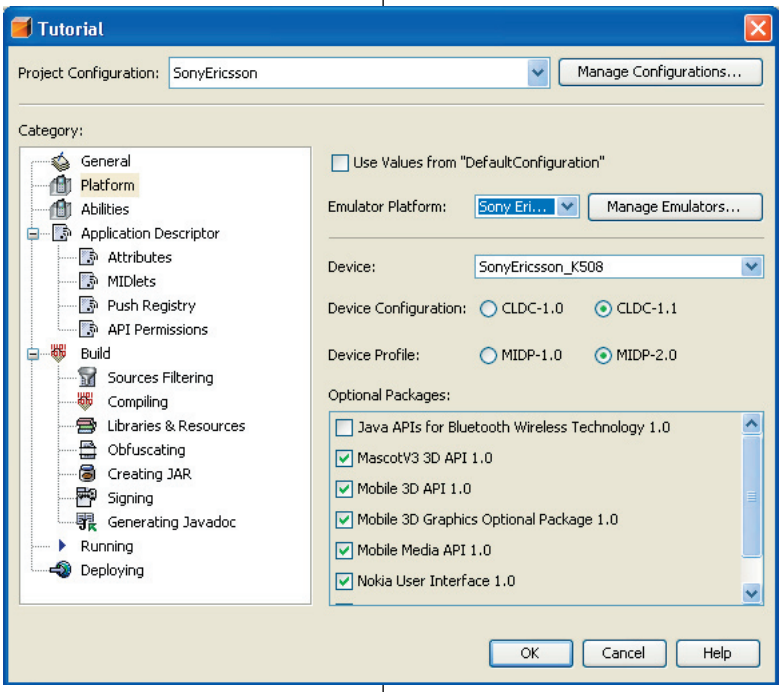
Emulator Platform drop down menu. Ensure that MIDP-2.0 is selected. Switch to the Abilities group and once again disable Use Values from "DefaultConfiguration". Click Add to add a new ability, and name it after your emulator (e.g. SONYERICSSON). Now close the properties dialog, and open TutorialCanvas.java so we can see what all this was about.

In your doPaint method, just before the drawString call that renders the available number of lives, add the following code:

```
//#if SONYERICSSON
g.drawString("SONYERICSSON", getWidth()/2,
getHeight()/2, Graphics.HCENTER|Graphics.TOP);
//#endif
```

Notice that the preprocessor directives (like #if) are 'commented out', this is because unlike in C++, preprocessing is not normally part of the build process (in fact some Java purists may be appalled by this necessary evil). NetBeans automatically preprocesses out code for us, saving us from some extra work. The SONYERICSSON tested for above is the 'Ability' we added in the project settings, so it may be different for you.

Now click in the margin to the left of the new drawString call and a pink block indicating a breakpoint will appear. Now run your project (F5) and see what happens. Two things should be different to what you are used to. First of all, rather than Sun's ugly (and unrealistically large) emulator, you will see that you now have



Property configurations are extremely customizable

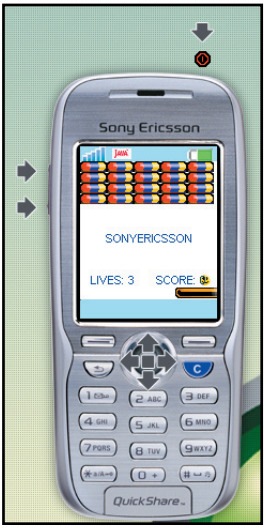
an emulator running in what looks much more like a real phone. Secondly, when you run your game, rather than being faced with the bouncing ball, paddle and blocks, you are kicked back into NetBeans right where you set your breakpoint. This is a very good thing. As recently as 18 months ago, this sort of thing was only possible through expensive IDEs like Jbuilder, and the typical hobbyist mobile developer would have to make do with scattering System.out.println calls everywhere to determine what was going on when something went wrong. Notice that, in the watches tab in the bottom right of NetBeans, you can actually see the values of all local variables currently in scope. Feel free to look around at the rest of the tabs to see what they offer as well. Click in the margin again to remove the breakpoint and press Ctrl+F5 to continue running the game. Notice the text in the middle of the screen? You expected that, right?

Now I'm sure you can guess what the preprocessor defines are for, but let's do a little proof of concept for interest sake. Close the emulator, right click on the Tutorial project and choose Set active project Configuration->DefaultConfiguration. Notice how NetBeans has commented out the code between the #if and #endif directives? Set a breakpoint in the same point and run the project again. Not only does the standard emulator launch, the breakpoint is never hit and our text is no longer drawn. So you see the combination of configurations, preprocessing and device specific emulators makes it much easier to tailor our code for each platform. In addition, if you chose to install the Sony Ericsson emulators, and you happen to have one of the newer Sony Ericsson phones, you can actually debug your game right on the device! Stepping through code on it is obviously a bit slower than just using the emulator, but this ability is amazingly useful considering it is almost impossible for manufacturers to make their emulators behave exactly like the actual phone.

That's it for this edition. We have really only scratched the surface of what is possible with emulators and configurations and the best way for you to find out more is of course to play around with these tools and see what you can discover. Also, take a look in the folder where your NetBeans project files are stored, and compare the java class files found in the preverified directory to the 'originals'.

Next week we will look at NetBeans' powerful WYSWYG mobile form editor and add some structure to our game.

FLINT



Swish! A shiny vendor emulator.

The Game Maker Communities

On the net there are several Game Maker communities. By this, I mean websites dedicated to the people who enjoy using the Game Maker development tool. In this article, I hope to give you an objective low-down on the most popular ones I have visited.

Game Maker Community

http://forums.gamemaker.nl

These are the official Game Maker forums. More often than not, they are filled with lots of controversies which no-one should play witness to. It is usually what some may call a "n00b-ridden" place, but it has its merits.

The Game Maker Community is a great place to get answers to problems, discuss Game Design and generally see what is happening in the world of Game Maker. It does its job in acting as the global center for all Game Maker communities.

64Digits

http://www.64digits.com

64Digits is an exceptional site which not only hosts your files, but also gives you access to your own Game Maker blog, download counters and badges. It is delightful to read all the like-minded developers' blogs, especially when Game Maker greats like GearGOD (1st lighting engine in GM) and FredFredrickson (King of Cagematch winner, ie Best GM game ever) blog there.

64Digits is a very friendly and an easy community to get into. One can easily craft one's own niche into the site. With several games developed specifically for 64Digits, like the LOLOMGWTFBBQ series and recent ColumsX, members can get badges from the games, which makes the community grow closer and closer.

Game Maker Showcase

http://www.gmshowcase.dk

Game Maker Showcase does indeed do what it says. It is an awesome place to showcase your games, since a lot of games get reviewed lengthily and quite frequently by the "reviewers". Its forum community seems close-knit – however, this does make it harder for newbies to get in.

Some of the threads (in the off-topic section) are legendary and a delight to read. It also seems that it isn't quite as geared toward game developing as other communities are.

A recent event caused the whole site to be remade, and it is still being reconstructed.

Game Maker Games

http://www.gamemakergames.com

This is home to greats like Darthlupi (The Cleaner) and Tapeworm (Seiklus). Game Maker Games is host to a vast archive of Game Maker titles! Most of the time, one's game does not get reviewed, but it is still a useful place to showcase your work.

It has a "top-sites" section, which usually falls victim to spammers, but it is still great place to check out the sites of other developers.

Its forum community is small and not as active as Game Maker Community.

GM Clans

http://www.gmclans.com

This is mostly a forum community built around the concept of "clans". While I have only recently started "lurking" there, it seems quite decent. Members get XP for their games, and level up according to what they did in a "clan".

Game Maker Info

http://gamemaker.info/nl

Although this isn't a Game Maker community, it's a great place to get all the news from all Game Maker communities, since it gets an RSS feed off all the main communities.

I hope I wasn't too subjective about all the communities, and I hope I catch you around on one of these forums sometime!

TR00JG