SOUTH AFRICA'S PREMIER GAME DEVELOPMENT MAGAZINE

April 2008

DEV.MAG

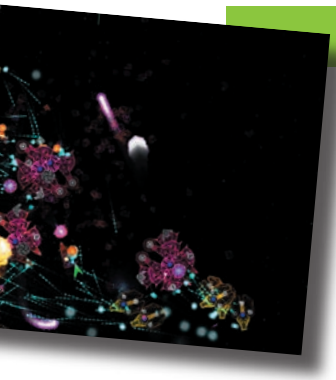
CREATE • DEVELOP • EXPERIENCE

ADVENTURE GAME STUDIO

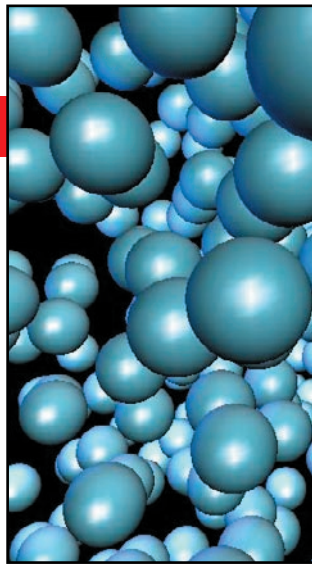
AN AWESOME TOOL FOR AWESOME DEVELOPERS!

REGULARS

FEATURE



REVIEWS



TUTORIALS

TAILPIECE

GameDevelopers Conference 08

DEAR READER ...

First off, apologies for the missing mag last month. Things were going a little bit crazy (it involved aliens and a bottle of bleach at one point) and we just couldn't get our material, people and inspiration all in one place. Don't worry, we all still love our work – it just gets rather difficult to keep the pace at times.

With the grovelling now out of the way, it's time for a brief discussion about what we've got lined up for you this month. The keyword: AGS. If you've played an awesome indie point-and-click adventure game recently, chances are it was made with AGS. The brainchild of one Chris Jones, AGS is an adventure game creator which has already charmed several members of the Dev.Mag staff with its powerful interface and pretty results. In this issue, we sport several AGS-related articles, including an interview with the creator himself as well as a few AGS game reviews and a Blue Pill primer article. Chris Jones has recently released version 3.0, so this is the perfect time to clamber onto the bandwagon and try making a few games yourself. If you're into adventure titles, you should try this one out – you may just be pleasantly surprised.

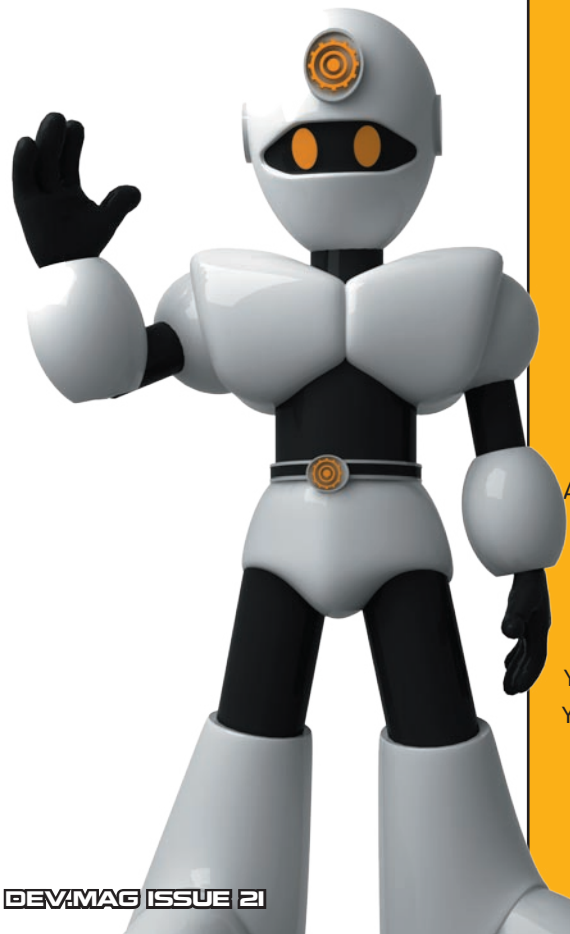
The 2008 Game Developers Conference was also held at the end of February, bringing with it the IGF and the indie games award ceremony which gets us all worked up every year. If you don't know about the GDC, then have a look at our tailpiece for a report on GDC 08 and several links that can help you get acquainted with what the conference is all about.

On a final note, I've been receiving a lot of e-mails recently from enthusiasts who are looking for information about the game development industry. Thanks for the interest, people! Not only is it flattering to receive a bunch of e-mails, but it gives us a lot of ideas about what articles to write in coming editions to help you on your way. So keep 'em coming – you'll help us help you.

That's all for now. Happy devving!

RODAIN "NANDREW" JOUBERT
EDITOR

We made mention of Trilby: The Art of Theft in a news brief last month, and we feel it's worth another mention now. This is a fine example of just how diverse you can get with AGS - and it's really, really fun to play. Several of us are still trying to get through it with the "Trilby" rating. See what we mean by hunting it down at http://www.escapistmagazine.com/



DEV.MAG
CREATE • DEVELOP • EXPERIENCE

EDITOR
Rodain "Nandrew" Joubert

DEPUTY EDITOR
Claudio "Chippit" de Sa

SUB EDITOR
Tarryn "Azimuth" van der Byl

DESIGNER
Brandon "Cyberninja" Rajkumar

WRITERS
Simon "Tr00jg" de la Rouviere
Ricky "Insomniac" Abell
William "Cairnswm" Cairns
Bernard "Mushi Mushi" Boshoff
Danny "Dislekcia" Day
Andre "Fengol" Odendaal
Luke "Coolhand" Lamothe
Rishal "UntouchableOne" Hurbans
James "NightTimeHornets" Etherington-Smith
Gareth "Gazza_N" Wilcock
Sven "FuzzYspo0N" Bergstrom
Kyle "SkinkLizzard" van Duffelen

WEBSITE ADMIN
Robbie "Squid" Fraser

WEBSITE
www.devmag.org.za

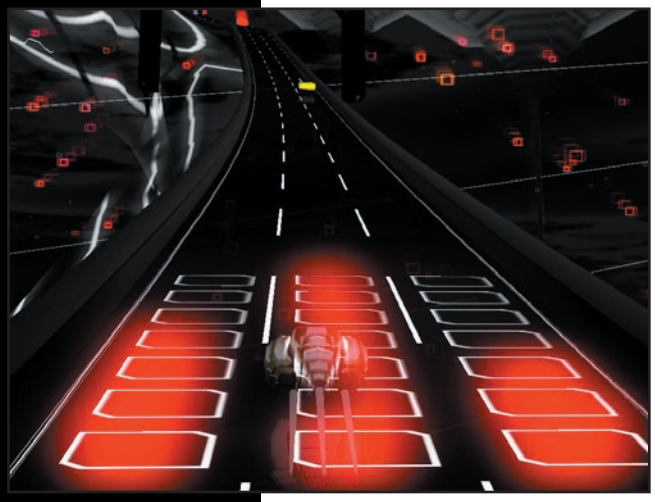
EMAIL
devmag@gmail.com

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

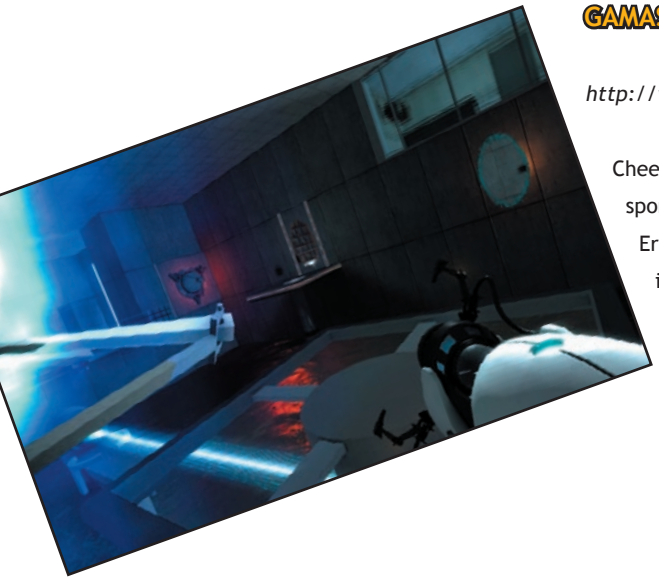
Your current score is 142 of 245. You have unlocked 21 editions of Dev.Mag. Your current rating: Journeyman.

AUDIOSURF RIDES ON IN ...

http://www.audio-surf.com/



The much-anticipated Audio Surf was finally released on the 15th of February. The game is available via www.steamgames.com for the extremely reasonable price of $9.95. The problems experienced during the beta tests have all been stamped out and many improvements introduced, such as customisable track colours and rendering effects, modifications to the track generation engine, an improved menu interface and Ironmode for the tough guys... The game is an approximate 380 megabyte download and comes packed with bonus bang for your bucks. The Orange Box soundtrack is bundled with the purchase and includes all the tracks from Half-Life; Half Life 2 and Episodes 1 & 2; Team Fortress 2 and Portal. It just makes one want to sing "This was a triumph..."



GAMASUTRA: PORTAL FEATURE

http://www.gamasutra.com/view/feature/3585/still_alive_kim_swift_and_erik_.php

Cheekily introduced as a feature about "a game you may have heard of", Gamasutra sports a few good pages on one of 2007's success stories. Interviewed are Kim Swift and Erik Wolpaw, who are the project's lead designer and lead writer respectively. The interview goes over the decisions made with Portal's more subtle design elements and how the narrative helped in shaping the player's emotions. Considerable discussion also surrounds their work with Narbacular Drop and the experience gained from their time at DigiPen.

STORMWINDS 1.5

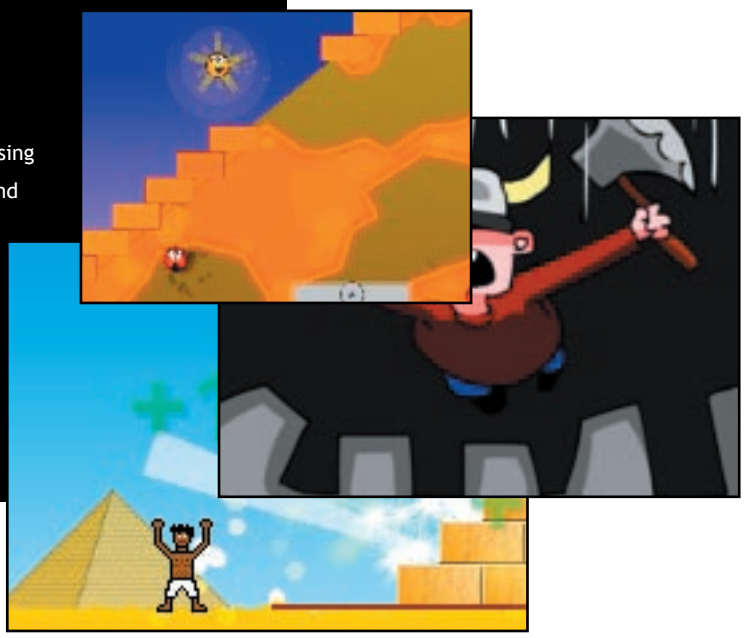
http://www.herointeractive.com/stormwinds_1-5/

StormWinds is a cute little web-based game by Hero Interactive which has you defending a castle against an onslaught of enemies by making use of turrets, upgraded turrets and even more powerful turrets. The game is reasonably fast-paced and filled with cool bits of action that should keep most avid gamers occupied for a healthy amount of time. Coupled with some great artwork and a considerable amount of content, StormWinds is definitely one of the more appealing games that you can find on the Internet right now.

ANCIENT CIVILISATIONS WITH YOYO GAMES

http://www.yoyogames.com/gamemaker/competition02

Yoyo Games are now holding their second competition, this time focusing around ancient civilisations. The high standards, flexible conditions and hefty rewards of the previous competition are set to be matched by this second incarnation. Not only is the topic delightfully broad, but the duration of the competition itself has been extended and the snowball started by the Winter competition also seems to have grown. If you're interested in producing some work with Game Maker and want to have a shot at the US$1000 first prize, try it out. At the very least, you'll get a bit of experience and exposure.



LOST GARDEN: HOW TO BOOTSTRAP YOUR INDIE ART NEEDS

http://lostgarden.com/2007/12/how-to-bootstrap-your-indie-art-needs.html

So, have you got the perfect game cooking in your head but just don't know how to make it look good? Afraid that your masterpiece is going to flop because players won't be able to make it past the hideous title screen? Worry no more. It's better to read this little article from Lost Garden and become enlightened. Contained within this link are a few valuable points of wisdom that any game developer can learn from if they're stuck for good game graphics. Even free tilesets aren't as bad as you may think.

BRAID ARTIST BLOGGING

http://www.davidhellman.net/blog/

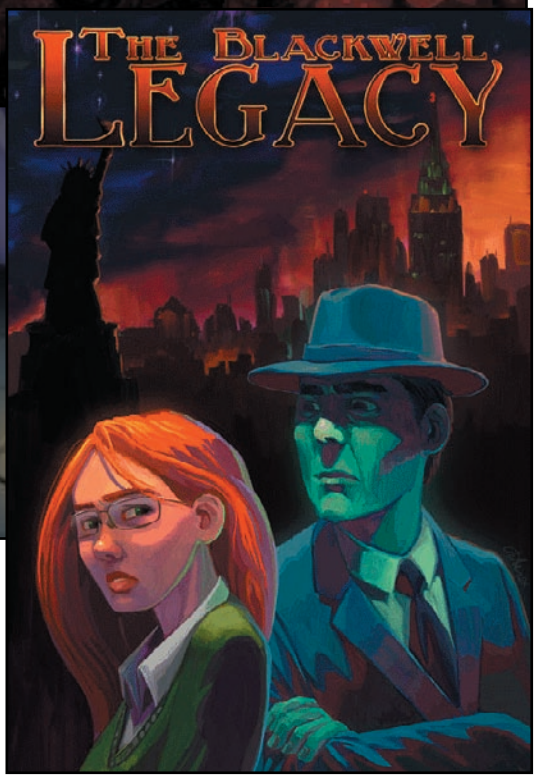
David Hellman, the artistic genius responsible for the graphics of indie game Braid, has his own blog lurking around on the Internet. As it so happens, he also has some really interesting things to mention about the evolution of Braid's artwork and visual style. Have a look at his fledgling Art of Braid series, or follow the links to the Braid blog itself if you don't yet know what the game is all about.

BEING ADVENTUROUS

WE POINT AND CLICK ON CHRIS JONES, CREATOR OF AGS

by Sven "FuzzYspoON" Bergstrom

Chris Jones is probably a name familiar to anyone who has made use of the popular game development tool AGS, also known as the Adventure Game Studio. With the recent release of version 3.0, there is a swell of excitement amongst fans and the software is commanding a fair amount of attention from the devving community. We caught up with Chris and harassed him for a few quick words on his work with AGS.



Tell us a little bit about yourself.

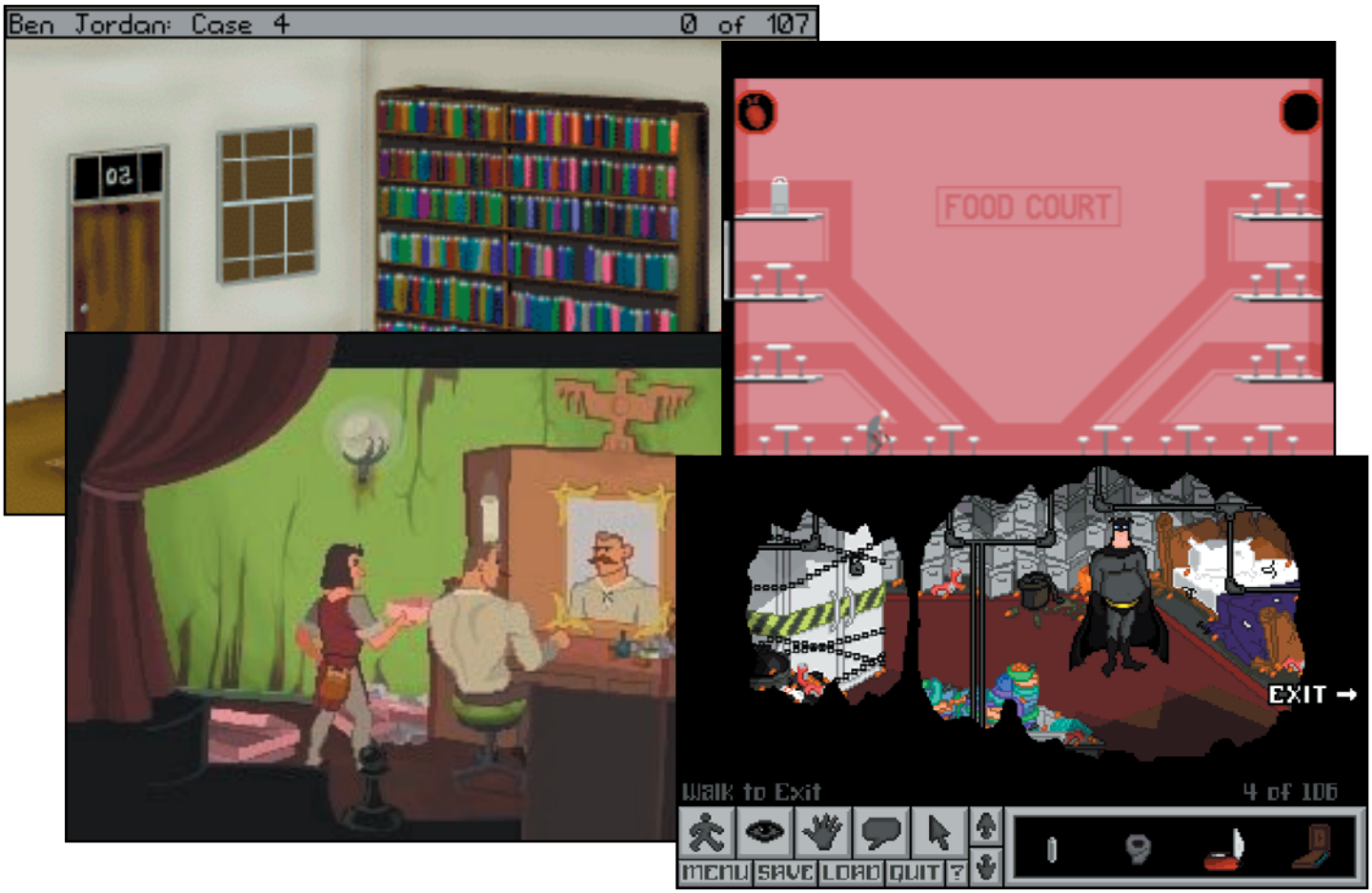
Well, I'm not the Chris Jones from Tex Murphy, I'm not Chris Jones the singer-songwriter, and I'm not Chris Jones the weightlifter. Nor am I Chris Jones the Corporate Vice President of Microsoft Windows.

How did game development draw you into its deep spiral?

Very much by accident. I wrote the first version of AGS (or Adventure Creator as it was called back then) just as an experiment, as something I was going to use to make a few games myself. This all came about after playing Space Quest IV, and realising that I wanted to make something like that myself.

What inspired you to create AGS and other tools for the masses?

It was never intended that way. As I say, I created the first version just for myself, and then as the internet started to take off I got a website and stuck Adventure Creator up there just for fun. I never really expected anyone to download it, much less use it for anything serious.



AGS has been going for years, how do you feel having recently released the latest version?

It's always a relief to finally get a new version released -- especially one that's been under development for as long as AGS 3.0 was. And then it's always interesting to read people's feedback, and decide whether it was worth the hassle or not.

What plans do you have for AGS in the future?

Hard to say. The two major things I wanted to do - rewrite the editor, and add hardware acceleration support to the engine - have now been done, but there's always room for improvement! People on the AGS Forums are never short of ideas for ways that it could be improved, so I'm sure I'll be kept busy for a good while yet!

Would you ever release the AGS source code?

Yes, one day. I would probably release most of the editor source code if someone had a good reason for wanting it. As for the engine, I prefer to keep that closed source while I'm still developing it - but if one day I decide to give up on AGS, at that point I'm sure I'd release the source code, or at least give it to someone trustworthy! ;)
It's interesting to note that SLUDGE was recently open-sourced, so we'll have to see if anything useful comes of that to decide whether there's actually any benefit to doing something like this.

In terms of projects, what do you like to get involved in?

Sadly I don't really have the time to get involved in any game projects these days. I've done some scripting work for a few games, but nothing significant.

How do you feel knowing some incredible games are being spawned from your platform?

Well, seeing the games that people are making with AGS is my inspiration for continuing development of it. After all, there's no point making a tool that nobody uses. There's a lot of talent out there, and if AGS helps people to nurture and improve theirs then that's a big bonus.

If you could recreate any retro game you wanted to, given the time, which game would you re-make?

That's a tricky one. If I had the time, I think I'd probably remake all the Lucasarts games to use the Sierra-style interface!

DB SAYS ...

We've thrust a fair amount of AGS coverage into this issue for interested parties. Check out the tool's potential, then visit http://www.adventuregamestudio.co.uk/ to try it out yourself!

Windows Vista

Open Source Win64

GameBoy Advance

DirectX 10

.NET 2.0

XBox 360 via XNA

Linux

Mac OS X

Nintendo DS

OpenGL 2.1

SDL

Object Pascal has a lot to offer...

www.PascalGameDevelopment.com

Chrome

Free Pascal

Turbo Delphi

Delphi for Win32

Lazarus

TRILBY'S NOTES

By Ricky "Insomniac" Abell

Back with the third installment in the Chzo Mythos series by Ben "Yahtzee" Croshaw we have Trilby's Notes which takes place 4 years after the events in the DeFoe manor from 5 Days a Stranger. The Notes part of the title refers to handwritten notes from Trilby which serve as a narrative to the story. A lot as happened to Trilby in the past few years as he has left his "gentlemen" burglar ways behind and is now an agent for the STP(Special Talents Project) which is basically some kind of secret government agency. A new career is not the only changes Trilby has gone through as he is still haunted by what happened in the DeFoe manor and not a day goes by without him thinking about it.

The game starts off with Trilby reflecting on the past few years and it seems his nightmares from the past might be coming back. Without spoiling anything his investigation leads him to a hotel to try and solve this mystery once and for all. This is were things really start to go crazy as one minute a room in the hotel can look perfectly normal and then all of a sudden, as if Trilby has just moved to an alternate realm, the very same room is covered in corpses and the walls are splattered with messages written in blood. It's now up to you to get your grey matter working and solve this mystery.

Gameplay has also gone through some major changes, you can throw your mouse out of the window for this adventure game as it makes use of text parser. For those of you too young to have played a game with a text parser, you type in various instructions such as "open door" to control the gameplay. It may seem quite different and cumbersome at first, especially if you have never played a game with a text parser before, but it is polished and works fairly well once you have wrapped your way of thinking around the concept in a fashion the parser will understand. Conversations are much more interesting as you get to type out carefully thought questions rather than just clicking an option. Nevertheless it does have some flaws, for example when you forget someone's name or you can see an object which looks like a book but the parser won't understand "book" as it's actually an envelope. However this can be solved by tweaking your approach for example typing "look at table" to help identify what's on it. Like the previous games there are still points where you can die and get taken back to your last save so you make sure you save regularly!

Yahtzee has once again raised the bar in terms of style and creepy atmosphere that will have you on the edge of your seat until the very end. From the mysterious characters you meet to sudden changes in your surroundings, this game will draw you in and have you wondering what twist lies around the corner. There are also a lot of little touches that really add to the creepy experience such as when you're in the alternate realm you can her eerie whispering in the background. There was also a time when I went through a door and got taken to a room from the last game in the series, 7 Days a Skeptic, which was a really unexpected shock.

The storyline is well paced and has you guessing what will happen next without leaving you feeling confused. Along the way Trilby has what you could call visions to events in the past which not only flesh out the story but are also playable which is a great immersive touch. Many gaps are filled in with regards to events in the past games which makes the story even better, and what an exciting ending the game wraps up with!

As the game can only run at a maximum resolution of 640x400 you can't really expect much but the drawings and animations are fairly impressive and get the job done, plus what it doesn't have in pixels it makes up for in style. Sound is also fairly simple but a lot of effort has been put into it making it quite effective at setting a scene's mood.

Overall Yahtzee has produced another great game in this series and manages to keep things new and interesting partly thanks to the text parser but also by filling in lots of gaps in the story. While not perfect, it's a great game and definitely worth your time.

6 DAYS A SACRIFICE

www.fullyramblomatic.com/

By Gareth "Gazza_N" Wilcock

One hundred and ninety six years ago, cat burglar Trilby escaped the horrors of DeFoe Manor, only to discover an even greater danger. One hundred and ninety six years from now, an unwitting starship crew will stumble upon a remnant of his attempts to avert that danger. And now, at the exact midpoint between those two events, a hero will rise. Yea, for Theodore DeCabe, Municipal Inspector extraordinaire, is going to tell the leaders of that fad religion exactly what his employers think of their constructing building extensions without a permit!

Yes, it's just a regular building inspection, at least until one of those crazy Optimologists decides to shove him down an elevator shaft...

So begins 6 Days a Sacrifice, the fourth and final game of the Chzo Mythos series. Don't let the fact that it takes place before 7 Days a Skeptic fool you, it really is the grand finale, and be forewarned: don't even try playing it before having completed the rest of the series. This isn't so much an issue of spoilers as it is of comprehension, because 6 Days takes all the loose plot threads of the entire series and marries them into a single epic climax, with the full assumption that you already know what's going on. That established, 6 Days is doubtless the most cinematic of the series, handling multiple plot threads through excellent exposition and pacing. As with its predecessors, events start out relatively normally, but quickly spiral into something much more sinister. One really has to play the game to appreciate how well the story has been constructed and told this time around, and it serves as a fitting and satisfying conclusion to the saga.

Of course, the quality of the plot means nothing if the game is frustrating to play. Rather than reusing the text parser from Trilby's notes, 6 Days a Sacrifice returns to a fully mouse-driven interface. 7 Days' right-click verb coin/inventory menu makes a welcome return in this instalment, which is no disaster given its effectiveness. But it isn't alone. To bolster his Arsenal of Interaction, Theodore has at his disposal a journal and a cell phone, both of which can be accessed at any time. The journal has no real practical application in the game, but is excellent in that it allows you to reread at your leisure all those little bits of exposition that you need to understand the story. The real stroke of genius, however, is the phone. Asides from its use in one or two puzzles, the cell phone allows you to communicate with the primary NPCs from any location, which saves you a lot of tedious walking around if all you need is a single clue or a reminder of what to do.

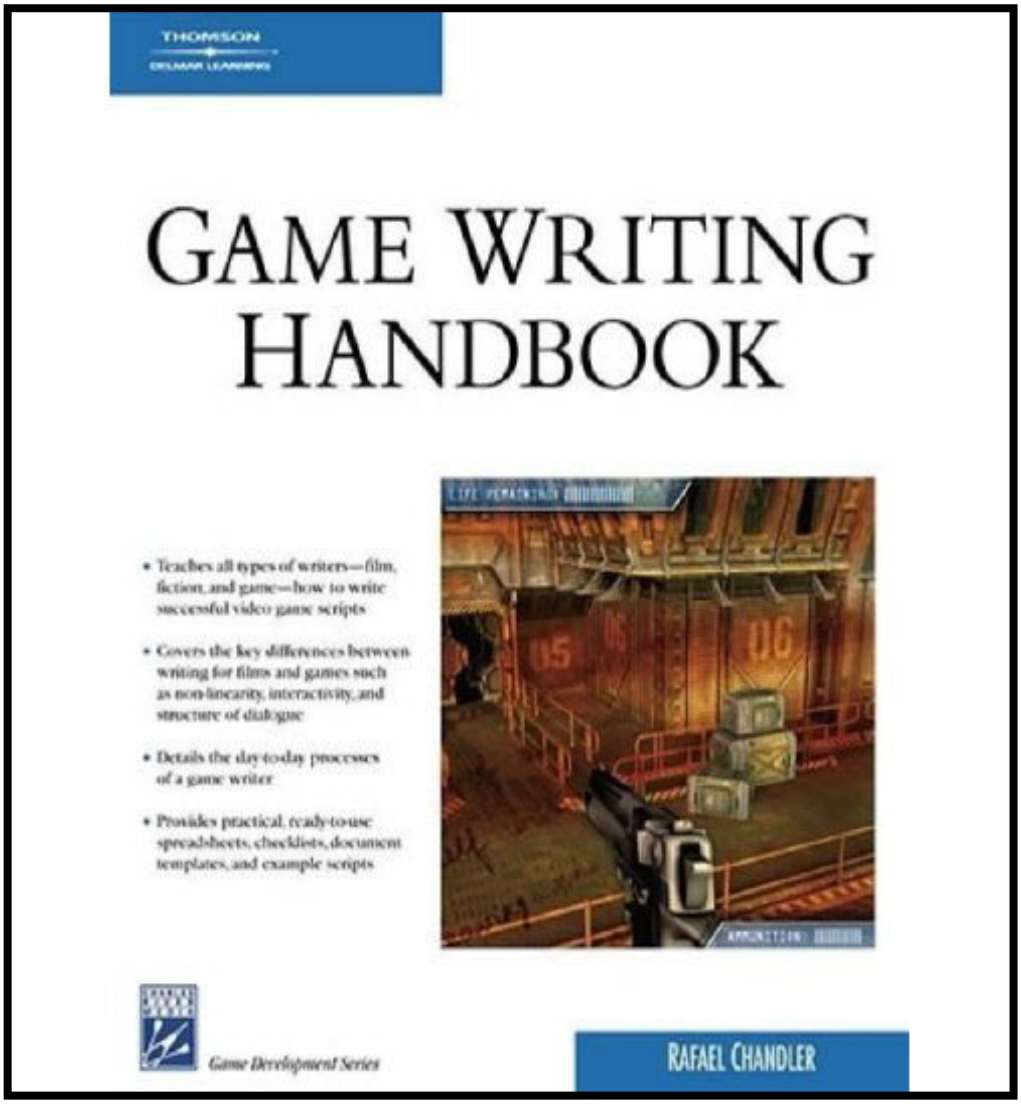
The puzzles in the game are a definite improvement over the sometimes nebulous ones in the previous Chzo games. Not to say that they are easy though - the later puzzles in particular require slightly more abstract thinking to solve. However, they remain logical throughout, with the solutions easily arrived at after a little skull sweat and observation. It also wouldn't be a Chzo game without some perilous life-and-death situations, but fortunately Yahtzee seems to have overcome the compulsion to kill the player out of the blue, which weeds out a lot of the frustration factor of previous games (7 Days, I'm looking at you).

All in all, 6 Days a Sacrifice is a fine plot-driven adventure game, with very little to fault it for. If you've played through and enjoyed the rest of the Chzo Mythos, and are itching find out how it all ends, you shouldn't hesitate to play it. In fact, why aren't you downloading it already?

GAME WRITING HANDBOOK

By Sven "FuzzYspoON" Bergstrom

Title: Game writing handbook
Author: Rafael Chandler (www.rafaelchandler.com)
Publisher: Thomson Delmar Learning
Level: Beginner to Intermediate
Genre: Game design
Cost: ~ $40
ISBN: 1-58450-503-6
http://www.amazon.com/Writing-Handbook-Charles-River-Development/dp/1584505036



Taking a look at game development opens up a whole array of aspects to consider. There is programming, art and even time-consuming testing. One aspect largely overlooked might be the writing part of game development. What compels you to buy a game? Is it the graphics or the sounds? Or is it the concept of the game? The story that leads you to desire the sequel, to complete the tale in which you played a part, is most likely a large factor in buying a game. This book deals with the game writing process - creating game narrative.

Packed with technical examples, example data, spreadsheets, information and even practical approaches to the in-game integration of story and cut-scenes, this book poses some great views on the processes involved. An author with such great references is another bonus; having worked as a writer for EA, Southpeak games, 1C and even Ubisoft, Rafael Chandler has a track record worth listening for. His portfolio, including titles such as Ghost Recon: Advanced Warfighter and Rainbow Six: Lockdown, gives him a lot to show for.

Covering a lot of topics in the book, there are a lot of aspects presented that are easy to learn about and offered in large detail throughout the book, including chapters covering:

- Writing a game,
- creating the concept,
- documenting the story,
- developing the context,
- creating the characters,
- structuring the narrative,
- organizing dialogue,
- creating cinematics,
- directing voice actors,
- knowing technical parameter,
- integrating dialogue,
- testing story content,
- understanding postproduction and
- working in the industry.

This book really covers some deep sections, but in a non-complex and simple-to-understand way. Enjoying this book was easy, and the learning curve is fantastic. Check out the author's site as well for more on him.

Battleships Forever

http://www.wyrdysm.com/

by Claudio "Chippit" de Sa

Battleships Forever plays out like most other tactical games. You select a fixed starting force, and must use it to the greatest effect to achieve your objective for the missions. What makes this indie offering unique is just how smoothly everything looks and works. Ships break apart, debris flies across the void and giant laser beams shred ships section by section.

Battleships Forever features all the little tricks and conveniences that you expect from a strategic game of its type, including waypoint systems, formations and unit grouping. Little is present that belies the game's simple indie origins, and it oozes polish and quality.

Most players will likely start off playing the short campaign, which will serve to introduce the player to the game and its mechanics. The game also features special skirmish missions which have their own unique goals, as well as a just-for-fun sandbox mode where you can define your battles exactly as you wish and import custom ships made with the included Shipmaker tool.

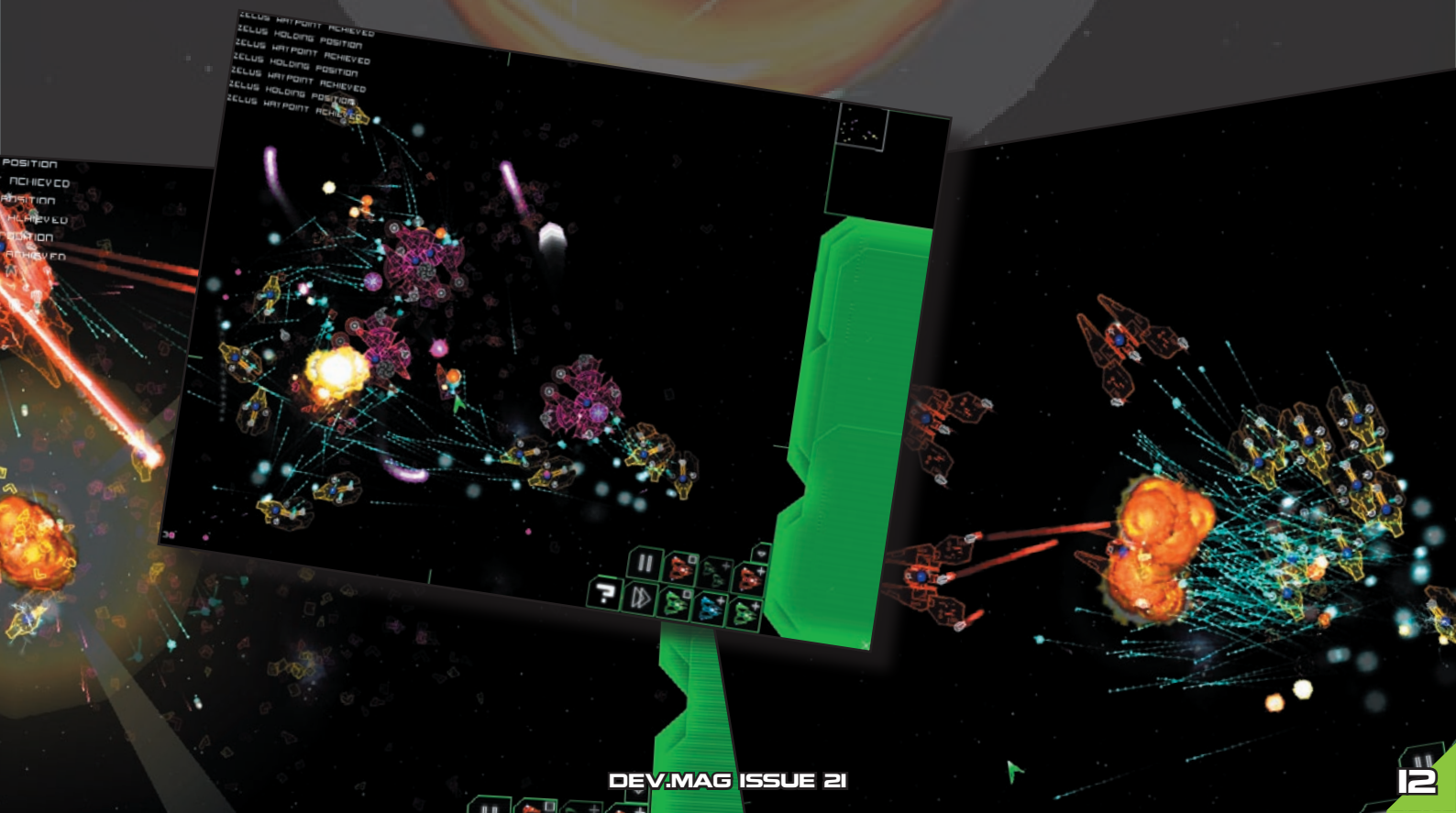
Default ships are outfitted with a unique and widely varied array of death-dealing devices that will, both spectacularly and realistically, rend entire sections from enemy ships based on exactly where each blow lands.

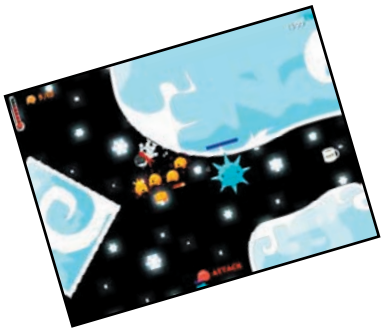
Individual sections of the craft can be destroyed, disabling any weapons or defensive systems mounted onto that portion. While destroying the core portion of the ship will cause the entire ship to explode, it is often heavily armoured or defended by other sections, and it may often be beneficial to target auxiliary sections of enemies before striking the heart of the ship.

For most missions you can select your fleet out of prebuilt vessels with various weapon layouts and physical designs, each serving its own unique purpose in furthering your goal for the mission.

The Helios destroyer, for example sports defensive weaponry that can destroy enemy projectiles before they can harm your ships. The Peitho battleship will utilize its large physical profile together with its shield generators to protect your fleet in a more passive manner. Others, such as the Hecate battleship or the smaller Enyo destroyer, bristle with intimidating weapons and will heartily take a more forward approach in your enemy's destruction.

The player will be challenged to use his limited and personalized fleet in every mission to the utmost effect in order to achieve whatever goal has been set out for him, but the simple pleasure of the game lies in commanding your fleet to blast enemy ships to pieces bit-by-bit and relish in the resulting fireworks. Commanding a fleet of vector warships has never looked quite so stylish.





FROZZD

http://www.yoyogames.com/games/show/20523

by Claudio "Chippit" de Sa

BY JESSE VENBRUX

The very first YoYo Games competition was held at the end of last year, with over 200 hundred winter-themed entries submitted and vying for the $1000 grand prize. The tantalizing 2 week long wait for the results eventually yielded a clear winner: Frozzd, a spacey platformer where the goal is to save the universe from its current icy predicament.

Players spend the majority of game time hopping between small planetoids trying to rescue as many Mubblies (a race of space-dwelling aliens) that are frozen in that area as possible. Each area is comprised of small frozen bodies of various shapes and sizes that the player can jump between, each exerting their gravity on the player. Hence, you'll spend quite a fair amount of time upside down.

Regions branch out as you continue playing, which presents the player with some choice in how to proceed. It is encouraged to visit all the available regions, however, since the accumulated score and power will only be beneficial in the impending conflict. This necessity renders the apparent sense of choice obsolete and transparent, but does not detract from the charming experience.

In a gameplay sense, the Mubblies serve as your only attack and defence. They sport the ability to launch fireballs at the Frozzd (The frosty enemies in the game), as well as thaw out other Mubblies that have been frozen. The player is able to command Mubblies either to concentrate their fire and thaw out frozen Mubblies, or to attack hostile enemies. Mubblies will swarm around you and it is up to you to keep them out of harm's way by either avoiding enemy contact and weapon fire, or shielding them from it yourself. Doing so will drain your suit's temperature more quickly than normal exposure to the cold environment, however, and will force you to find special pickups scattered around the levels in order to prevent yourself from joining in the unfortunate icy fate of the universe.

Utilizing your Mubblies to defeat enemies earns you score points. Score points are used to unlock new regions to explore and reveal progressively more Mubblies and new types of Frozzd enemies. Eventually, the player will encounter the source of the icy scourge and will have to use the accumulated Mubbly forces to defeat it once and for all.

Ultimately, Frozzd is an engaging platform experience and, despite its somewhat disappointing length, certainly deserves is competition victory.

SYNAESTHETE

www.synaesthetegame.com

By James "NightTimeHornets" Etherington-Smith

Synaesthete was a contender for the Excellence in Visual Art award at the 10th annual Independent Games Festival (IGF) and winner of the Student Showcase portion of the event. The word 'synaesthesia' refers to a neurological phenomenon in which one sensory perception involuntarily triggers another perception, such as letters and numbers being perceived as having their own associated colour. Obviously the game name suggests that this is the kind of experience the developers, Rolling Without Slipping, want you to have, blending visual and musical elements into one seamless experience.

Synaesthete puts you in control of the Zaikman, who has to navigate the game world - the collective unconscious - and zap monsters using his affinity with the musical soundtrack and a number of power-ups with are unlocked as progress is made. The game features original compositions of music ranging from ambient trance to, well, regular trance. The entire game world is a visualisation which pulses and animates to the beat of the music with some very cool looking textures, intricate and pretty effects, and bright psychedelic colours. The nomination for excellence in visual art is well deserved.

The game play mechanics are simple in concept but nearly as difficult to master as the spelling of 'Synaesthete.' The player is presented with three scrolling beat tracks, each a different colour and each representing an element of the music, be it beat or synthesiser. Each colour corresponds with a key which must be pressed in time to the beat if Zaikman is to stand any chance of destroying the array of monsters which populate the levels. The game is quite forgiving toward novice players, allowing them to concentrate on a single colour, while it rewards more experienced and coordinated players with a higher score. All this concentration on the scrolling beat tracks is a bit of a downside since the player has no time to really observe and appreciate the details of the monsters that assail them. After the completion of each area the player is given a thought provoking message, be it a quote from a famous poet, a line of lyric from a song or something you are likely to hear burbling from the mouth of an illicit drug addled trance party goer.

Synaesthete offers two difficulty settings, a 'trivial' 33 bpm and a 'tricky' 45 bpm. There are 3 distinct game world areas, each with 3 levels of their own. The final area is a mega boss fight in which the Zaikman will meet his greatest challenge and the player is likely to suffer from some form of repetitive strain injury. The game can be completed rather quickly the first time round but for anyone sufficiently entertained by the concept, graphics and trance music there is probably a bit of replay value, mainly in trying to master the key stroke combinations and achieving a new high score.

BLENDER TUTORIAL

Image File Formats

By Stefan "?rman" van der Vyver

This article refers to resources available at the "Contents" section of the Dev.Mag website (www.devmag.org.za). It is recommended that you visit the site and download these resources.

Consider that you manage to build something in Blender. It looks great, you are happy, and now you want to get an image that you can actually use outside of Blender. Confusion sets in. Panic is the next step and then a lot of time wasted because no-one ever showed you what that next step should be?

Here's an idea of image formats relevant to the gaming industry and what you need to know about them:

BMP

- The standard image format.
- Huge file size.
- No compression.
- Best quality.
- No transparency or alpha channel.
- Anti-aliased (explained further down below).

TIFF

- Big file size.
- No compression.
- Popular for printing graphics (both RGB and CMYK color mode).
- Supports alpha channel.
- Anti-aliased.

TGA

- Big file size.
- No compression (although you can choose to use compression if you do want it).
- Best quality.
- Full support for alpha channel.
- Does not display in Windows explorer thumbnail view, which makes it somewhat awkward to work with.
- Anti-aliased.

JPG

- Good image quality.
- Smaller file sizes.
- Variable image compression that leads to reduced quality the more you compress the image.
- No support for transparency of alpha channel.
- Anti-aliased.

GIF

- Very low file size.
- Indexed color palette (the picture is analyzed and colors reduced to a palette of 256 colors). One can also reduce the color palette even further, down to two colors if need be. File size reduces accordingly.
- Supports transparency (not alpha channel – explanation follows further down)
- No anti-aliasing.

PNG

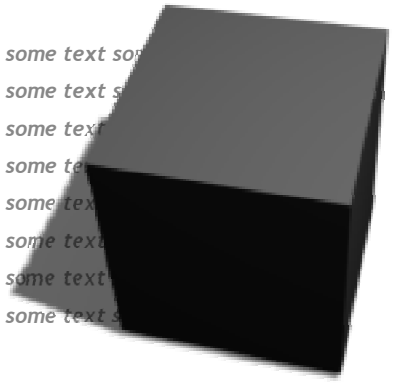
- Big file size.
- Compression as a choice, but not necessary.
- Supports full alpha channel.
- Anti-aliased.

What is transparency, and what is alpha channel?

Transparency is a "simpler" form of alpha channel. Alpha channel gives more advanced transparent information.

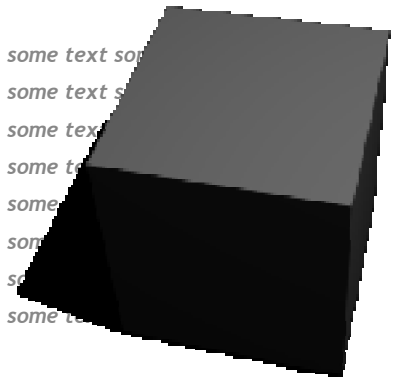
The following image is a cube that was rendered with a shadow. The shadow should be transparent, so that the background can be seen through it. The image has alpha channel information, allowing the white of the page to show through.

PERSONAL DISCLAIMER: *This list is by no intention comprehensive. Since I am the one writing this tutorial, I am relying on my own knowledge gained in a production environment to give you the best explanation that I can offer, with relevance to game*



Above is the image in PNG format, overlaid on some text. Below is the same cube, but in GIF format.

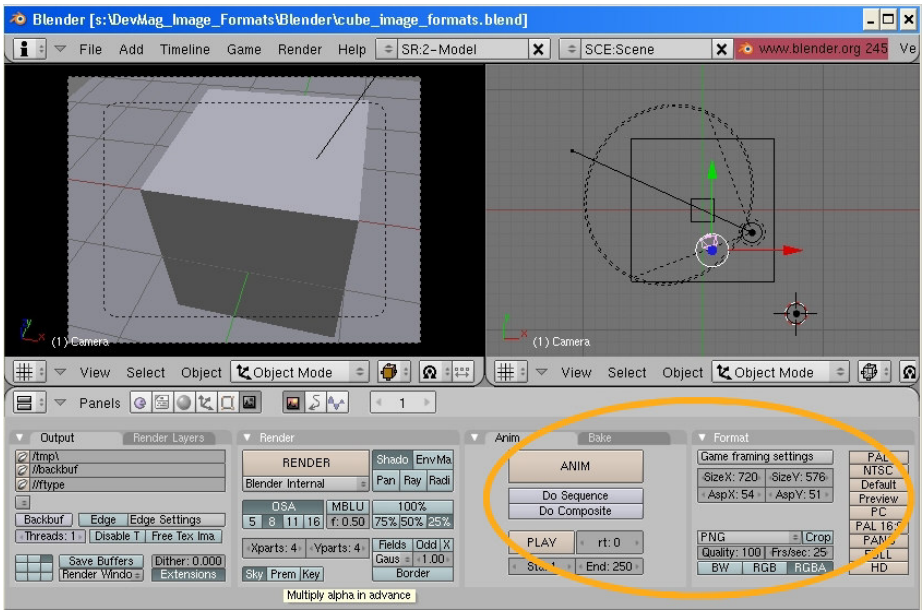
This has transparency so that the image has transparent areas. Now you can't see the text below the image, although the image has transparency inside the image borders.



If you don't get it immediately, read through it again to spot the difference. Once it makes sense, you probably won't forget it again. Another big difference is the smoothness of the edges (anti-aliasing). In most formats, color differences between areas are "smoothed over" by color blending the pixels between the different color areas. The GIF format does not support anti-aliasing, resulting in jagged, pixellated edges, especially where transparency is used.

Now it's up to you to decide what format is best to use in your projects. Some software will only allow you to use specific formats, based on a variety of factors. Certainly, for gaming, one would want to keep image sizes as low as possibly to increase game speed.

Now it's over to Blender. I included a file called cube_image_format.blend for use with this tutorial. This has a cube already set up, with a plane acting as the ground. Open up Blender, and let's get rendering. Upon opening Blender, you should see the screen as above.



The answer to our rendering needs lie mainly in the orange circle. That is where we indicate to Blender which format to save the rendered image as. We also specify whether the image should include an alpha channel or not. Blender cannot render and convert and image to GIF format. That has to be done in an external image editing application.

First, though, I need to explain the setup I have for the scene. The cube is a standard Blender primitive. It has no material assigned to it in this file. Below the cube is a plane (another Blender primitive). There are two settings on the plane object's material that we need to look at. RMB (Right mouse button) click and select the plane object in the right hand viewport. When selected, it turns pink.

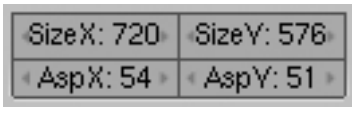
Click the Shading button to view the material assigned to the plane. What we want from the plane, is that it is in itself transparent, but that we can see the shadow of the cube that is cast onto it by the spotlight.

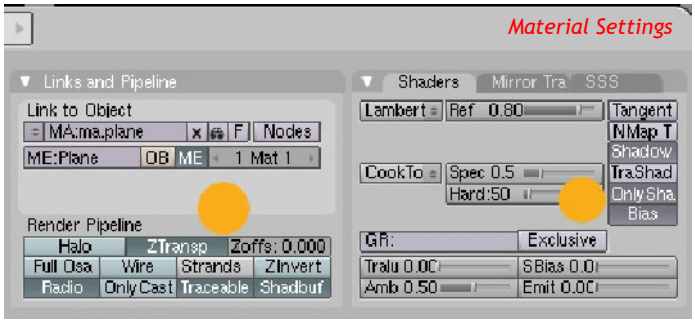
The most important setting here is the OnlySha button. This indicates that the material renders only the shadows that it receives. The second important button that needs to be activated, is the Ztransp button. The explanation for this is beyond the scope of this tutorial. Not activating that button may leave you with the shadow being the color of the background of your render window.

The color of the plane is not important for now, since the plane itself is not rendered. Click to the Scene button to get back to our render settings.

Buttons to note at this point is the *Oversampling* option to turn anti-aliasing on or off during the render. This speeds up render time (in the off position), but should really be enabled for the final render.

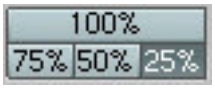
The next important setting is the rendered image size. All computer images are treated as rectangular images, therefore only an x and y dimension is specified. Ignore the AspX and AspY buttons for now.



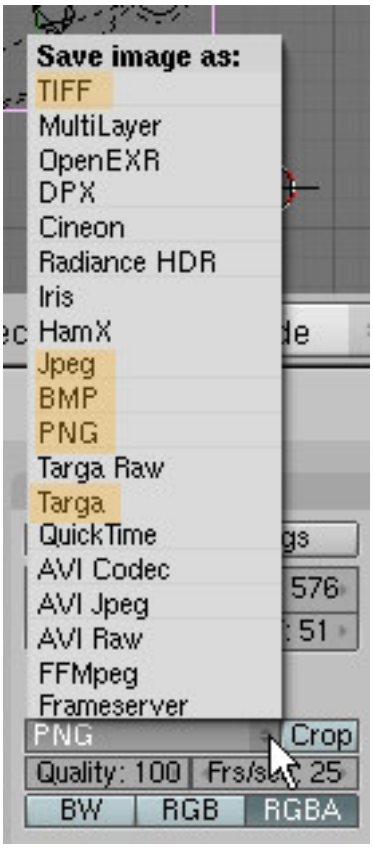


Enable the Shado button to render shadows. The Ray button also plays a role, but I set up the scene not to use it. Further explanation of this is beyond the scope of this tutorial.

The percentage buttons specify a render size as a percentage of the indicated size above. This is used to do quick test render, a smaller image taking less time to render than a large image.



Finally, we get to the image format options. Simply choose the image format that you want to render to, and press F3. Not so simple? I have highlighted the most relevant image formats.

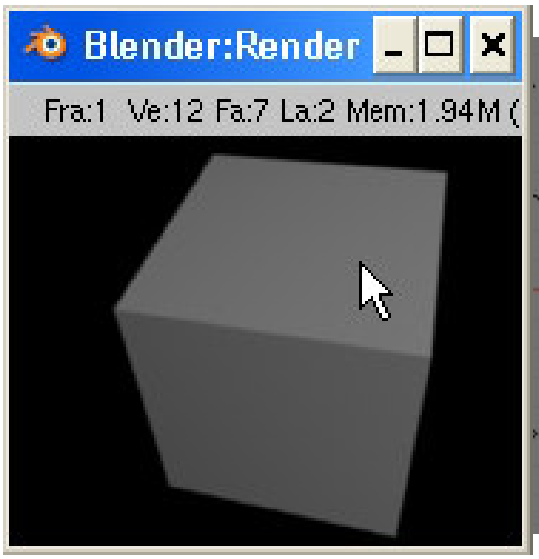


At the bottom of this format dialog there are two buttons that are important to this format selection. They are RGB and RGBA. RGBA indicates one's decision to render Red, Green, Blue as well as the Alpha channel.

Consulting the information at the beginning of this tutorial, it might make sense to you that, TIFF, PNG and TGA can use the alpha channel. Therefore "RGBA" needs to be enabled. BMP and Jpeg (same thing as JPG) on the other hand cannot use the alpha channel information. For these formats RGB needs to be enabled instead.

Go ahead and choose the PNG format. Enable RGBA.

Now press F12 for the image to render. I set the render to 25%. You may, of course, change that. The rendered image looked like this:



Where is the transparency?

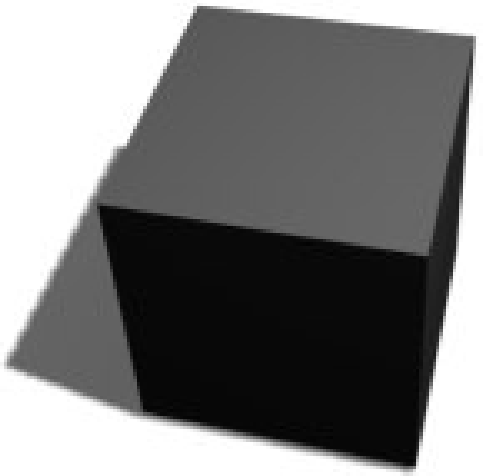
Don't fuss. It's there. The Blender render screen does not show transparency. Hit F3 and save the image to a location on your computer. From there you can open the image in a graphics editing application to see the transparency.

cube_render

Above is a Windows XP screenshot of the thumbnail of this render.
Should you want to change the background colour of your render, consult the online Blender manual that can be found at www.Blender.org.

It is my sincerest wish that this tutorial may have clarified some issues regarding image formats for you, and that you will find it easier in future to render image from Blender in the correct format.

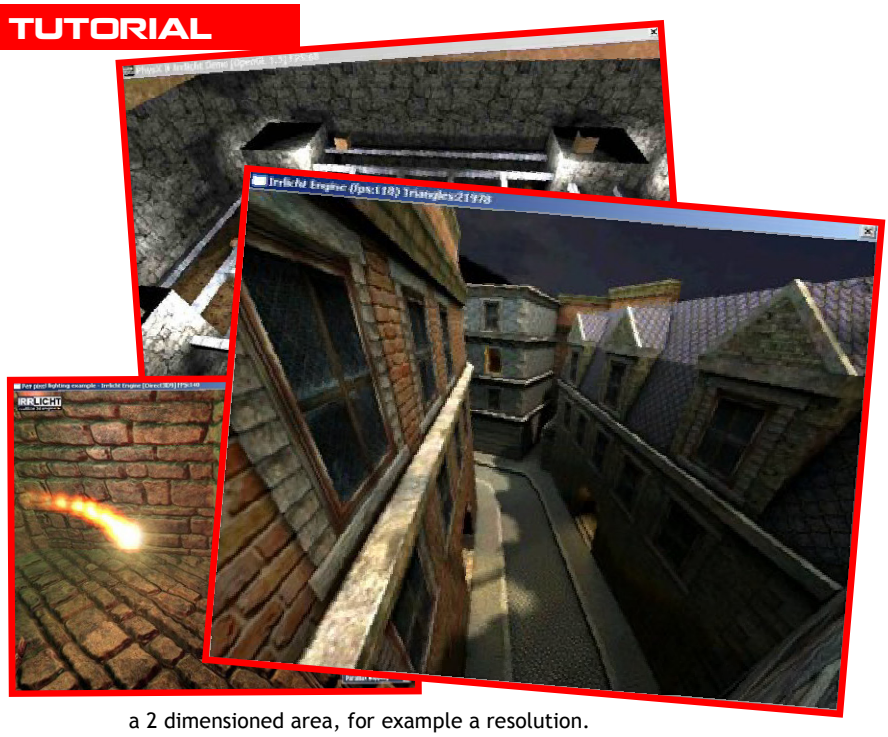
Happy Blendin'



IRRLICHT

Part 2 - Engine outline

by Sven "FuzzYspoON" Bergstrom



Well, having downloaded the Irrlicht SDK would have gotten you a number of great things. Tools of all sorts, a bunch of cool example programs and of course, the source code of the engine. Lets start outlining the engine and its concepts in this second part of our series.

Running through the examples is a good start for anyone looking to see what Irrlicht is about. They are a good start to understanding what could be the engine of your future game, and to see what the developers chose to showcase their free engine. This part 2 of the series is going to outline the engine, not what it can do, not what it cant do, but the layout and "concept" behind the design. That means there is no code sample specifically this week, but we are definitely going to be starting to use the engine today.

Lets start somewhere logical, the beginning of most Irrlicht based applications are going to require some sort of "handle" to the engine itself. The entire functionality of the engine is exposed through one simple object, the IrrlichtDevice. This device represents, for the simplest of descriptions, your hardware. It accesses all sorts of things, the operating system, the graphics hardware and even file access can be done with the engine itself. The device creation parameters are setup using a simple structure, and you populate a structure, and pass it to the function that will return a device for you to use. There are two create device methods, but the second, createDeviceEx is by far the best option. The options we hand the device tell it what we want from the graphics engine, and which mode we want the engine to start in. With 10 or so options, they are mostly self explanatory. Anti-aliasing, vertical synchronisation, full screen, high precision FPU and even stencil buffering are all Boolean options to setup in the creation process. Bits per pixel, resolution, and most other relevant options are all set with values according to the requirements you might have.

This brings about a good time to discuss variables and Irrlicht types. The Irrlicht engine has a namespace called irr::core which contains a huge amount of useful math, graphics, types and informative functions. These are all available to use so that type conversion can be simplified within your code and the code can be easily created, as well as easily understood. For example, when you create an IrrlichtDevice you are required to hand it some types you might not know about. **core::dimension2d< s32 >** looks daunting at first, but is easy to understand once you are familiar with the engine itself, in the language you choose. For this article, we are going to stay away from code conventions and focus on principles, so for now we need to understand the simplicity. Irrlicht has created a type that represents a 2 dimensioned area, for example a resolution. 1024x768 is a good example of this, and is easily handed to Irrlicht using the core namespace.

Once we have a device, there are a few namespaces that the engine exposes for us to use. The core, GUI, scene and video namespaces are all available directly through the device. The device has functions like:

getVideoDriver() return a handle to the video namespace. There is way too much to list what each namespace exposes, so briefly let's look at what is available.

Core :: useful conversion, type definitions and enumerations for irrlicht.

Scene :: All scene management is handled here, meshes, 3d objects, cameras and even animation is handled within this scene manager class.

Video :: This exposes your video drivers directly. Texture loading, management and rendering is done here. All 2D and 3D rendering happens within here.

GUI :: Graphical user interface, this should speak for itself. Fonts and windows are all managed in this namespace as well.

As you can see understanding the engine layout is really simple. If you need anything from the system the device can do it for you, or alternatively you may create references of each of the core elements of the engine and use those. The simplicity the Irrlicht engine brings out of 3D games and the creation of robust 3D applications is truly something to watch out for. With methods to handle most of the needed elements of 3-dimensional rendering the Irrlicht engine certainly puts up a great show for the cost.

This concludes our fairly short edition in this series, but understanding this is a great benefit rather then diving straight into code. Pick up the API reference and have a look at the functions and types that each namespace can offer you in the mean time, and don't forget there is a lot to learn if you are learning a new engine. Understanding the simplicity of Irrlicht will greatly improve the learning time of the engines deep functionality. Until next time.

GAME GRAPHICS DESIGN

Part 5: Simple Textures and Tilesets

By Rishal "TheUntouchableOne" Hurbans

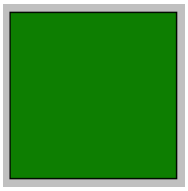
This article refers to resources available at the "Contents" section of the Dev.Mag website (www.devmag.org.za). It is recommended that you visit the site and download these resources.

This month is a particularly interesting and useful tutorial on creating graphic tilesets. A tileset is a collection of images that will be used to create the graphics for a game. The size of the images in a tileset is usually a number which is a power of two, such as 32x32 or 64x64. This is to ensure that the image will be compatible with all libraries and tools, such as OpenGL among others.

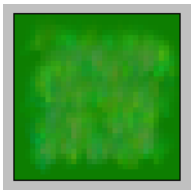
The purpose of a tileset is to have a complete image reference for your games that is easy to manage and use. This tutorial will not only to teach you how to create tilesets but to introduce them to you and give you a better understanding about them and how they can be used effectively.

In a typical tileset, grass, water, bushes and various other surfaces are useful to have for the surroundings and environment. This is dependant on the type of game you are creating and what graphics you need. We will create several tiles of different surfaces.

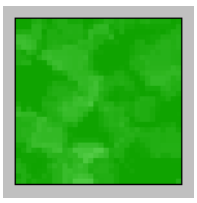
The first thing we will do as usual is create an empty image project in Photoshop. Use a size of 32 pixels wide by 32 pixels high. We will start with a typical grass tile. Two different variants of the tile will be shown as there is definitely more than one type of grass that could be used. This is to demonstrate that there are so many possibilities with regards to creativity for graphics.



For the first type of grass, change the colour of the Background layer to a dark lush green. The paint brush with the grass pattern was used in the previous tutorial, use it once again together with a lighter lime green and brush size of 8, now brush some random grass on the canvas. It would be best to keep the edges of the image clean of too much texture as it would make the final result look odd when tiled next to grass tiles of the same variant or of different variants when the images are used in a game. After applying a few different colours of grass brushes your image should look similar to the image alongside.



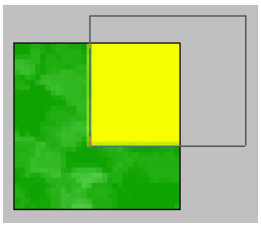
For the second type of grass, create a new Photoshop image project of size 32x32. Once again colour the background in a lush green colour. This time another brush tool will be used, choose brush number 95, it should be titled "Scattered leaves". Make the brush size 10 and select a lighter green, now brush the background with this texture. Once you are complete with brushing the texture, select Filter>Distort>Ocean Ripple. This is an excellent example of how the various tools in Photoshop can be wielded to create the exact effect you desire. The possibilities are endless with regards to the number of different types of grass tiles you can make so experiment with the different brush types, filters and blending. Your result should appear similar to the image to the left.



HINT *Remember to give your images short and meaningful names as you will be referring to them excessively when creating your game. It is good development habit to keep your images well organized and documented.*

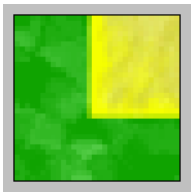
Another tile usually required is a "transition tile". Sometimes the player might cross from a grassy area to an area of different texture - such as hay, for example. In this case a tile needs to be created for every possible side of the transition. We're going to create a transition tile from grass to a type of dry grass/hay.

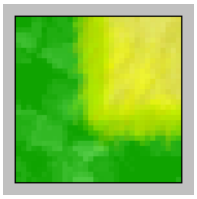
Open one of the grass tiles previously created and select the Rectangle tool. I used the second grass tile. Now select a yellow colour and create a rectangle as shown in the image below.



Select the rectangle layer you just created, right-click and choose Rasterize Layer. Now select a brush type of your choice and a different shade of yellow as the foreground colour. Apply the brush to the inside of the yellow rectangle while trying to keep away from the edges of the shape.

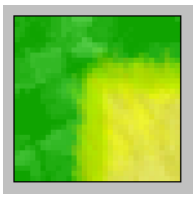
Now for the edges, select a colour in between green and yellow, the light lime seems to work well, select a good grass or leaf brush type. Now brush the edges of the rectangle so that it blends the two different textures together. It should look similar to the image below.

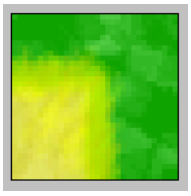




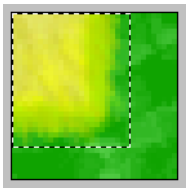
This transition texture looks pretty decent but it's only for the left bottom corner of the transition patch. We could go through the entire process again but that is too time-consuming and runs a big risk of inconsistency. So we will simply flip the entire canvas vertically and horizontally to get tiles we need.

Firstly, for the top-left transition tile, select Image>Rotate Canvas>Flip Canvas Vertically. Save the tile as file name of your choice. For the top-right transition tile, select Image>Rotate Canvas>Flip Canvas Horizontally. Save the file once again as something different. The remaining corner tile is the bottom right tile, so, as you have probably assumed, you will need to flip the canvas vertically again. Save the file once you're done.

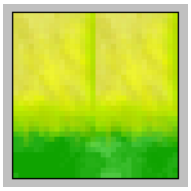




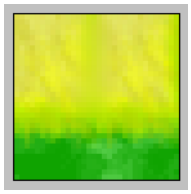
You now have the corners for the transition but another set of transitions are missing, the straight boundary tiles for top, bottom, left and right. This might sound like more work in brushing tiles from scratch but that's not necessary at all. Assuming the current tile is in the bottom-right, select the yellow grass area with the Rectangular Marquee Tool as shown alongside.



Now while that area of the image is selected, choose the Magic Wand tool, right-click on the image and select, Layer via Copy. A new layer should be created, select that layer and drag into position as shown to the left.

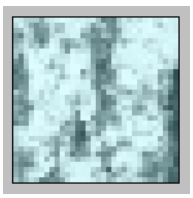


In the image I produced, there was a thin line of separation between the two yellow grass layers. If the image you created has the same problem, perform the following: Select the two yellow grass layers in the Layer Window using Ctrl and select the layers by clicking on them. Right-click on the selected layers and choose, Merge Layers. Now, select the Blur tool in the tools palette. Carefully apply the blur tool to across the images where there is a discrepancy. The fixed image should look as follows.



The bottom transition tile has been created. For the top transition tile, simply, flip the canvas vertically and save the file. For the right transition tile, rotate the canvas by 90° by select Image>Rotate Canvas>90° CW. Save the file. Finally, for the left tile, rotate the image by 180°.

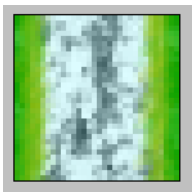
Other popular tiles in games are walls and stone floors. A very simple stone floor can be created with almost no work at all. Create a new 32x32 image project. Select the Rectangle Tool and draw a rectangle over the entire canvas. Now select the rectangle layer in the Layer Window and adjust the blending options of the layer. Change the pattern overlay to a texture that resembles a stone surface, make sure the scale of the texture is appropriate. You can also adjust the colour overlay to a light blue with a low opacity to add a further effect. Once you are complete with the blending options, you can save the file and there you have a stone floor tile.



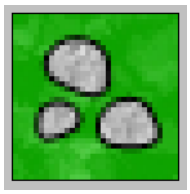
A tileset is made up of a variety of different types of tiles, which can depict different surroundings and areas in a game. In a RPG of ten levels, there could easily be more than a hundred different tiles for the many different elements of the game's graphics. There are also usually many transition tiles, making the game's graphics look pretty professional; they would be rather poor if the tiles just changed without any transition whatsoever. Other common tiles used in games are tiles depicting water, static objects and various other inorganic surfaces such as brick walls and shiny tiled floors. Go wild with creativity when creating graphics for your games. Your tile set may just make all the difference.

Some other possible tiles for the tileset we've created can be seen below together with a very basic scene using these tiles in a game.

You now know the basics about creating transition tiles and how to manipulate them without the extra effort of creating new tiles from scratch. If you believe your tilesets are up to scratch I'm sure the guys over at Game.Dev would be more than welcome to view and possibly even use you tilesets in their games. That's it for this tutorial. Good luck.

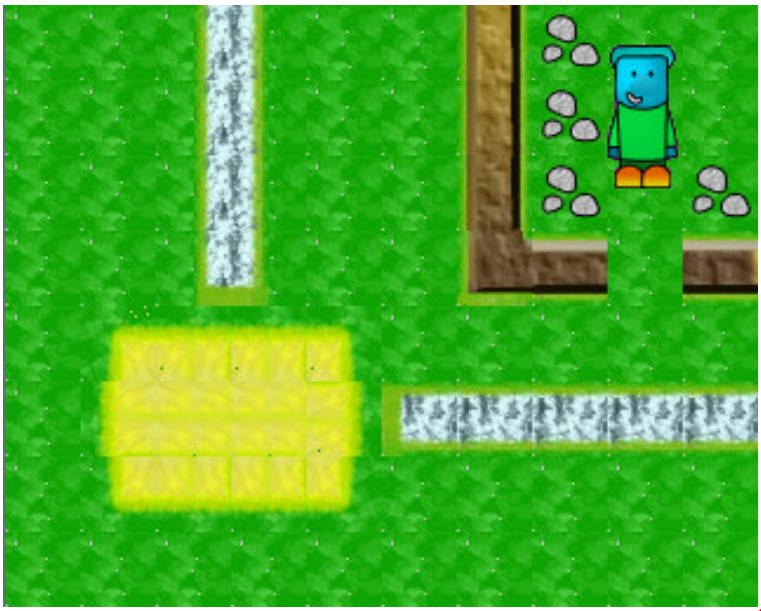


A rocky grass tile.



Stone path-grass transition tile.

Wall-grass transition tile.



POISSON DISK SAMPLING

Randomized object placement

By Herman Tulleken

One way to populate large worlds with objects is to simply place objects on a grid, or randomly. While fast and easy to implement, both these methods result in unsatisfying worlds: either too regular or too messy. In this article we look at an alternative algorithm that returns a random set of points with nice properties:

- the points are tightly packed together; but
- no closer to each other than a specified minimum distance.

Figure 1 shows an example of such a set, which is called Poisson-disk sample set. For comparison, two other sample sets are also shown: a uniform random sample, and a jittered grid sample.
Poisson disk sampling has many applications in games:

- random object placement;
- sampling for graphics applications;
- procedural texture algorithms; and
- mesh algorithms.

In this article we will look mostly at object placement and briefly at texture generation.

1.Implementation

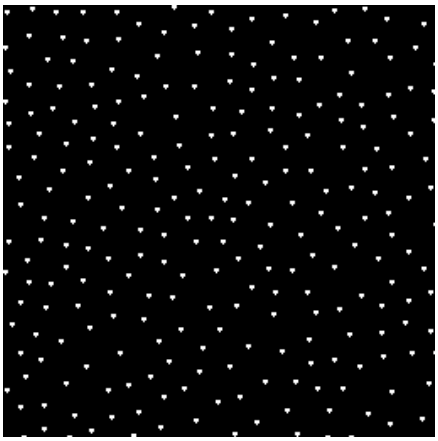
There are several algorithms for producing a Poisson disk sample set. The one presented here is easy to implement, and runs reasonably fast. It is also easy adapted for specific applications (described in the next section).

The basic idea is to generate points around existing points, and to check whether they can be added so that they don't disturb the minimum distance requirement. A grid is used to perform fast lookups of points. Two lists keep track of points that are being generated, and those that needs processing.

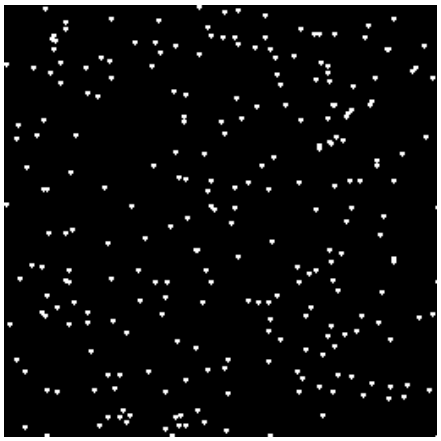
Here are the details:

1. A grid is created such that every cell will contain at most one sampling point. If points are at least distance *r* from each other, then the cell size must be *r*/ 2.
2. The first point is randomly chosen, and put in the output list, processing list and grid.
3. Until the processing list is empty, do the following:
    1. Choose a random point from the processing list.
    2. For this point, generate up to *k* points, randomly selected from the annulus surrounding the point. You can choose *k* - a value of 30 gives good results. In general, larger values give tighter packings, but make the algorithm run slower. For every generated point:
        1. Use the grid to check for points that are too close to this point. See below for more detail.
        2. If there is none, add the point to the output list, processing list, and grid.
        3. Remove the point from the processing list.
4. Return the output list as the sample points.

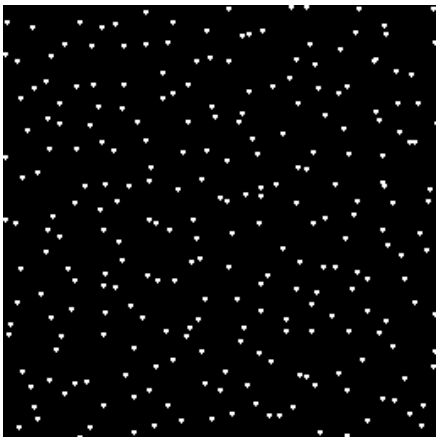
FIGURE 1



Poisson Disk sample points.



Uniform Random Points. The x and y coordinates of these points have been chosen randomly within the width of the image.



Jittered Grid. The image is divided into a grid, and one point is randomly selected from every cell in the grid.

Here is how all this look in pseudo code:

```
generate_poisson(width, height, min_dist, new_points_count)
{
    //Create the grid
    cellSize = min_dist/sqrt(2);

    grid = Grid2D(Point(
        (ceil(width/cell_size),          //grid width
        ceil(height/cell_size))));       //grid height

    //RandomQueue works like a queue, except that it
    //pops a random element from the queue instead of
    //the element at the head of the queue
    processList = RandomQueue();

    samplePoints = List();

    //generate the first point randomly
    //and updates

    firstPoint = Point(rand(width), rand(height));

    //update containers
    processList.push(firstPoint);
    samplePoints.push(firstPoint);
    grid[imageToGrid(firstPoint, cellSize)] = firstPoint;

    //generate other points from points in queue.
    while (not processList.empty())
    {
        point = processList.pop();

        for (i = 0; i < new_points_count; i++)
        {
            newPoint = generateRandomPointAround(point,
                                min_dist);
            //check that the point is in the image region
            //and no points exists in the point's neighbourhood
            if (inRectangle(newPoint) and
                not inNeighbourhood(grid, newPoint, min_dist,
                                    cellSize))
            {
                //update containers
                processList.push(newPoint);
                samplePoints.push(newPoint);
                grid[imageToGrid(newPoint, cellSize)] = newPoint;
            }
        }
    }

    return samplePoints;
}
```

The grid coordinates of a point can be easily calculated:

```
imageToGrid(point, cellSize)
{
        gridX = (int)(point.x / cellSize);
        gridY = (int)(point.y / cellSize);
        return Point(gridX, gridY);
}
```

FIGURE 2

Generating a new sample point

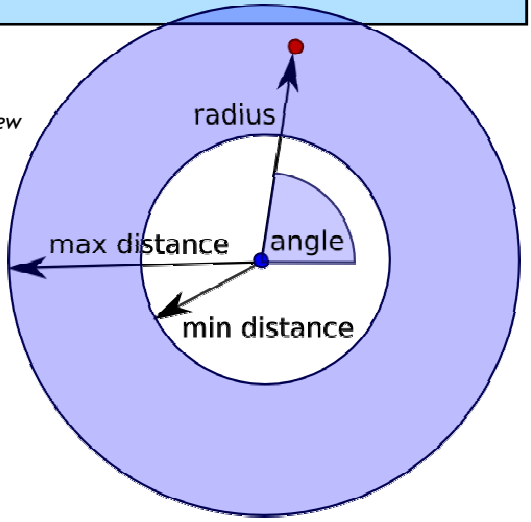


Figure 2 shows how a random point (red) is selected in the annulus around an existing point (blue). Two parameters determine the new point's position: the angle (randomly chosen between 0 and 360 degrees), and the distance from the original point (randomly chosen between the minimum distance and twice the minimum distance). In pseudo code:

```
generateRandomPointAround(point, mindist)
{
    r1 = Random.nex   tDouble(); //random point between 0 and 1
    r2 = Random.nextDouble();
    //random radius between mindist and 2 * mindist
    radius = mindist * (r1 + 1);

    //random angle
    angle = 2 * PI * r2;

    //the new point is generated around the point (x, y)
    newX = point.x + radius * cos(angle);
    newY = point.y + radius * sin(angle);

    return Point(newX, newY);
}
```

Before a newly generated point is admitted as a sample point, we have to check that no previously generated points are too close. Figure 3 shows a piece of the grid. The red dot is a potential new sample point. We have to check for existing points in the region contained by the red circles (they are the circles at the corners of the cell of the new point). The blue squares are cells that are partially or completely covered by a circle. We need only check these cells. However, to simplify the algorithm, we check all 25 cells.

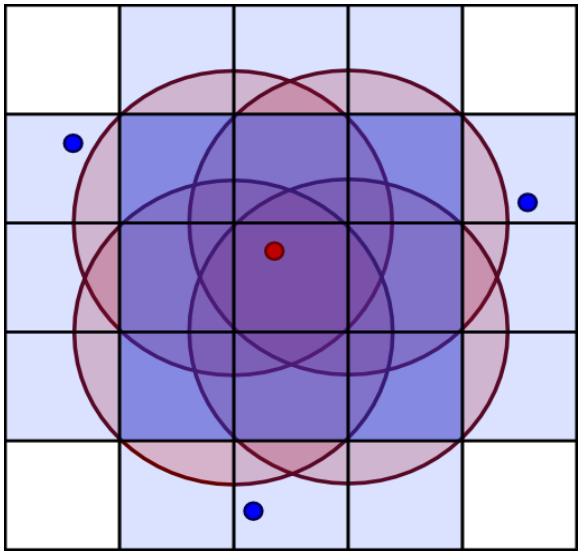


FIGURE 3

Checking the neighbourhood of a potential sample point

Here is the pseudo code:

```
inNeighbourhood(grid, point, mindist, cellSize)
{
    gridPoint = imageToGrid(point, cellSize)
    //get the neighbourhood if the point in the grid
    cellsAroundPoint = squareAroundPoint(grid, gridPoint, 5)

    for every cell in cellsAroundPoint
        if (cell != null)
            if distance(cell, point) < mindist
                return true
    return false
}
```

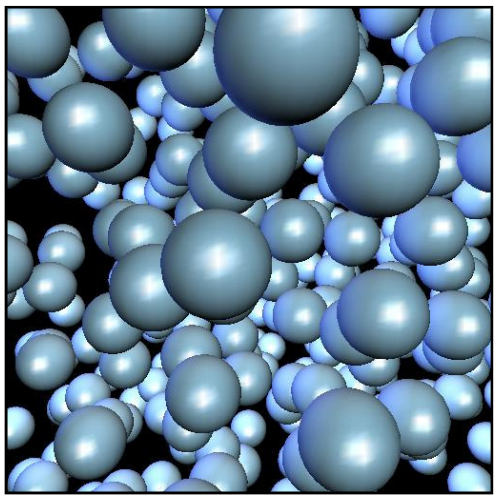
a. Implementation for 3D

The algorithm can easily be modified for 3D:

- Change all points to 3D points.
- Change the grid to a 3D grid. The neighbourhood of a point is now a cube of 125 cells around the cell of the point.
- Change the code to generate a new point to the following:

```
generateRandomPointAround(point, minDist)
{
    r1 = Random.nextDouble(); //random point between 0 and 1
    r2 = Random.nextDouble();
    r3 = Random.nextDouble();

    //random radius between mindist and 2* mindist
    radius = mindist * (r1 + 1);

    //random angle
    angle1 = 2 * PI * r2;
    angle2 = 2 * PI * r2;

    //the new point is generated around the point (x, y, z)
    newX = point.x + radius * cos(angle1) * sin(angle2);
    newY = point.y + radius * sin(angle1) * sin(angle2);
    newZ = point.z + radius * cos(angle2);

    return Point(newX, newY, newZ);
}
```

FIGURE 4



Spheres placed at points in a Poisson disk sample of 3D space.

2. Applications

a. Object Placement

Placing objects at the positions of a Poisson disk sample set is the simplest way to use this algorithm in games (Figure 5). Ideally, the algorithm can be built into your level editing tool with features that allows the artist to select the region to fill, and the models to fill them with.

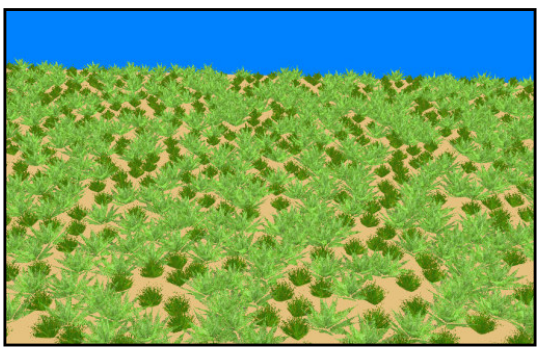


FIGURE 5

An example with shrubs placed at Poisson disk sample points.

One important variation of a Poisson sample set is one where the minimum distance between points is not constant, but varies across the image. In this variation, we feed the algorithm a greyscale image, which is used to modulate the minimum distance between points. To make this work, you need to modify the algorithm as follows:

- The grid must take a list of points.
- The cell size must be computed from the maximum the radius can be.
- The neighbourhood function should iterate through all points in every cell of a point's neighbourhood.
- Where a new point is generated, check the grey scale value of the image at that point, and calculate a minimum distance from the grey scale value:

```
min_dist = min_radius + grey * (max_radius – min_radius)
```

As an example, you can use Perlin noise to drive the minimum distance, giving you interesting clusters of objects (Figure 6). This method is especially useful for generating a field of plants.

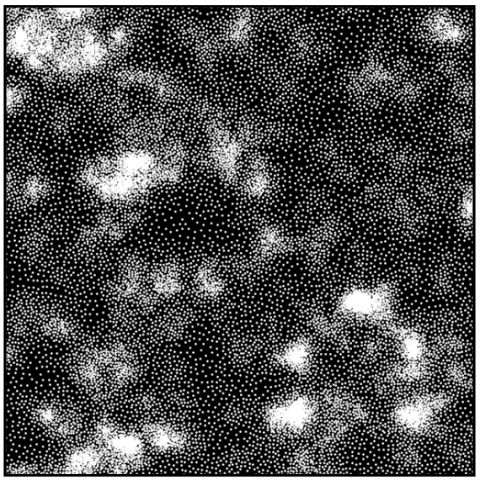
When using different radii as explained above, you might run into some problems. Take the following precautions:

- Ensure that your minimum radius is not too small. A very small radius might produce millions of points; a zero radius might prevent the algorithm from ever completing.
- Build in a bail-out feature in the algorithm, forcing it to end after a certain number of points have been reduced. Make it a function parameter, so that it can be modified according to the purpose. If you make a tool on top of this algorithm, also expose it to the user in the tool.
- Ensure that your maximum radius is not too big: if it is, no or very few points will be produced. This might seem obvious, but it can go wrong in a subtle way. Say, for example, you want to create a fall-off effect around a certain point. It would be natural to define your minimum distance array as follows:

```
min_dist[i, j] = dist((i, j), (x0, y0))
```

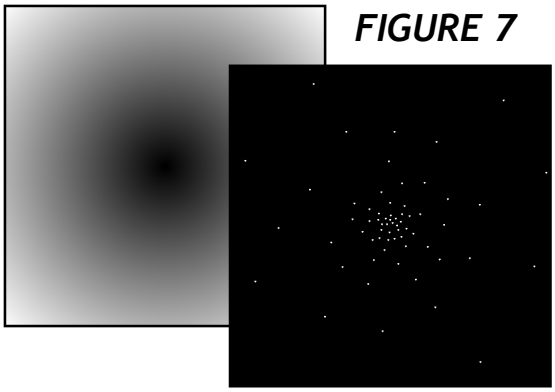
But because new points are generated at exactly this distance, many more points are excluded than expected, leading to a rather poor result. A better sample can be obtained by using the square root of the distance. (See Figure 7)

FIGURE 6

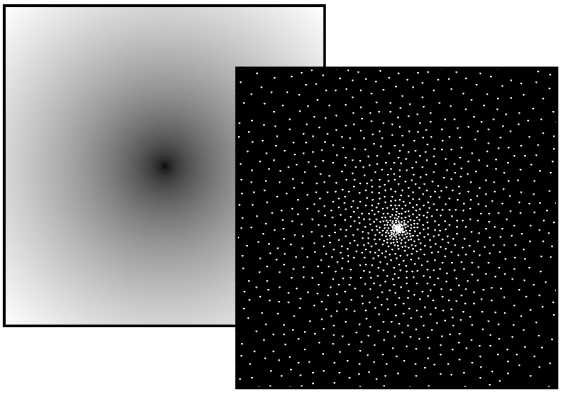


Poisson disk sample, where the minimum distance is driven by Perlin noise.

FIGURE 7



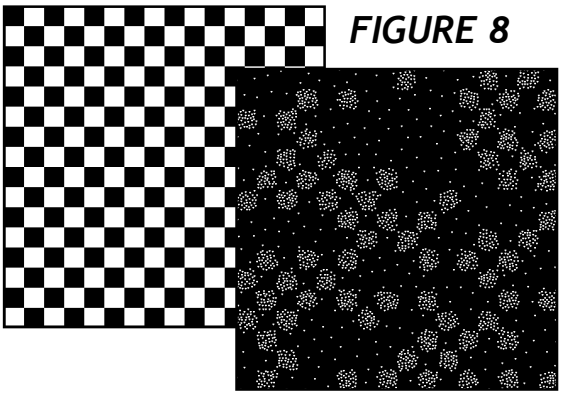
Using the distance from centre as minimum distance gives a poor result.



Using the square root of the distance from the centre as the minimum distance between points gives a much better looking sample.

In another situation, a section of large radii might be too close to other sections of large radii, so that no points are produced in sections of small radii (see Figure 8).

FIGURE 8

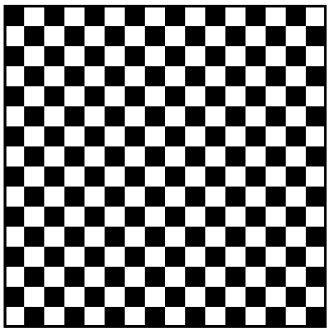


The high radius is too big, so that some low radius regions are skipped.

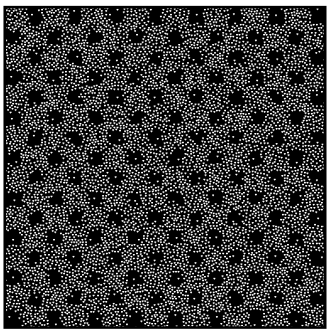
• For best results, your radius should vary smoothly across the rectangle. The algorithm respects sharp edges only ruggedly - see the checker board examples in Figure 9.

• Beware of the bleeding effect, as seen in Figure 9. You might want to run a dilation filter (use Photoshop or Gimp) on the radius grid before you do the sampling to compensate for this. Ideally the dilation should be a function of the minimum radius in a region, but in many cases you can used a fixed dilation.

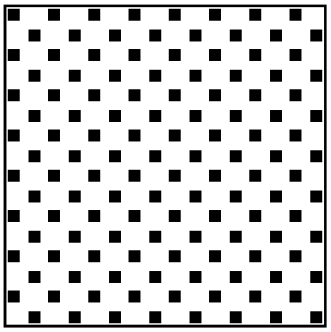
FIGURE 9



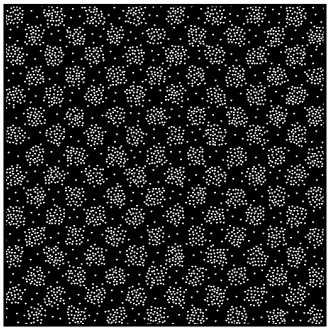
Checker board.



Low radius regions bleed into high radius regions.



Dilated checker board



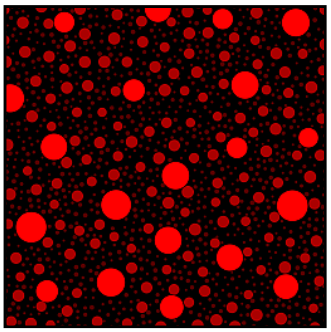
Artefact corrected.

b. Texture Generation

Poisson samples can also be used to generate certain textures. In Figure 10, the droplets on the bottle and glass have been created by combining three layers of Poisson disk sample points.

The modified algorithm creates three layers of points, each layer with with a smaller minimum distance between points than the next. In every layer, the algorithm checks that there are no circles in the previous layers. To do this, the look-up grids for all generated layers are used in a final step to eliminate unwanted points.

FIGURE 10



Raw texture (each layer is coloured differently).

The result. (Art: Luma Animation).

The local minimum distance of every sample point is stored, so that it can be used as a radius to draw a circle once all points have been found.

The raw texture is then further processed by clever artists through filters and shaders to produced the final result.

Poisson disk samples also form the basis of the procedural textures shown in Figure 11.

The first texture was produced by painting filled translucent circles for every Poisson sample point, for three separate samples. A different colour was used for every sample set, with small random variations.

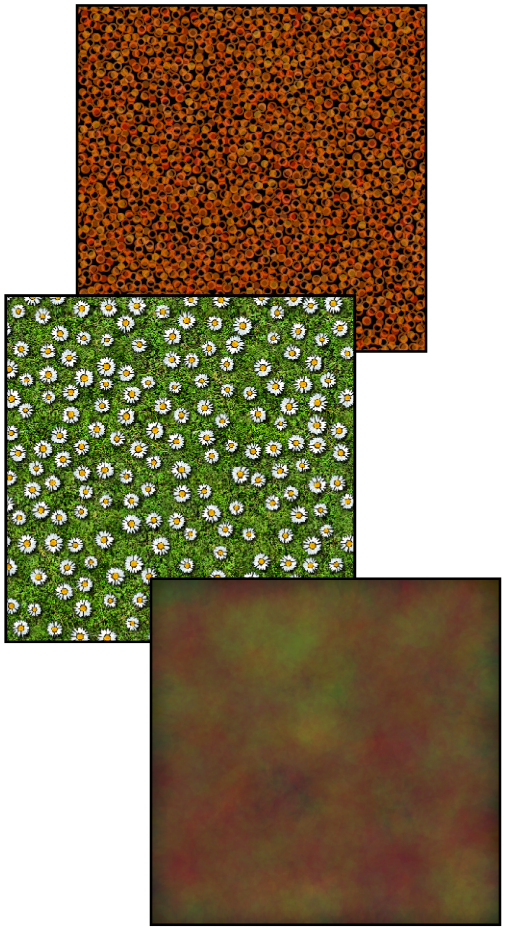
The second texture was produced by painting circles for two sample sets; one with filled circles, the other with outlines only.

The third texture was created by painting daisies (randomly scaled and rotated) onto an existing grass texture.

3. Download

You can download implementations for these algorithms from
http://www.luma.co.za/labs/2008/02/27/poisson-disk-sampling/
There are implementations in Java, Python and Ruby.
You can also download the "droplet generator tool".

FIGURE 11



Procedural textures generated from Poisson disk samples.

DB SAYS ...

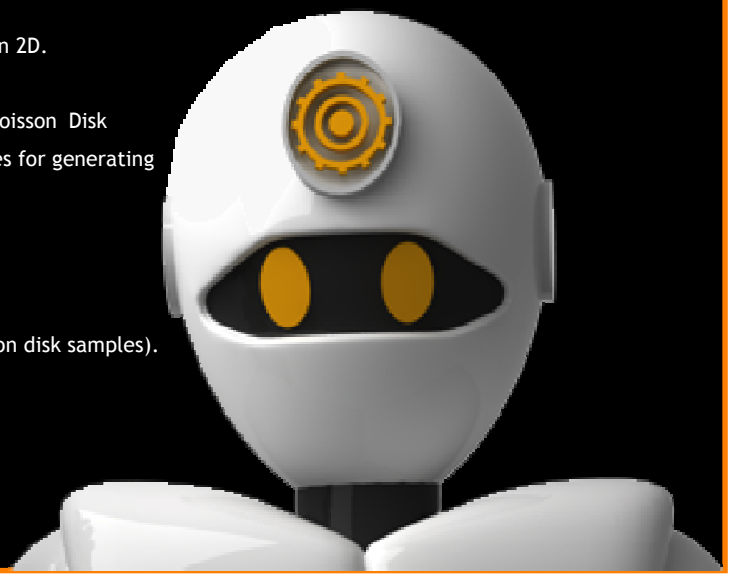
If you need more help with Disk Sampling, check out these sources for more information on the topic:

Bridson R. Fast Poisson Disk Sampling in Arbitrary Dimensions.
http://www.cs.ubc.ca/~rbridson/docs/bridson-siggraph07-poissondisk.pdf
The algorithm described in this article was taken from the paper above.

Dunbar, D.; Humphreys G. A Spatial Data Structure for Poisson-Disk Sample Generation.
http://www.cs.virginia.edu/papers/p503-dunbar.pdf
Describes a fast algorithm for generating Poisson-Disk Sample sets in 2D.

Lagae, A.; Dutré P. A Comparison of Methods for Generating Poisson Disk Distributions. An in-depth comparison between different techniques for generating Poisson disk samples.

Talmor, D. Well-Spaced Points for Numerical Methods.
http://www.cs.cmu.edu/~glmiller/Publications/TalmorPHD.ps.gz
A comprehensive exploration of well-spaced points (including Poisson disk samples).
Very mathematical!



Taking the blue pill for...

ADVENTURE GAME STUDIO

By Gareth "Gazza_N" Wilcock

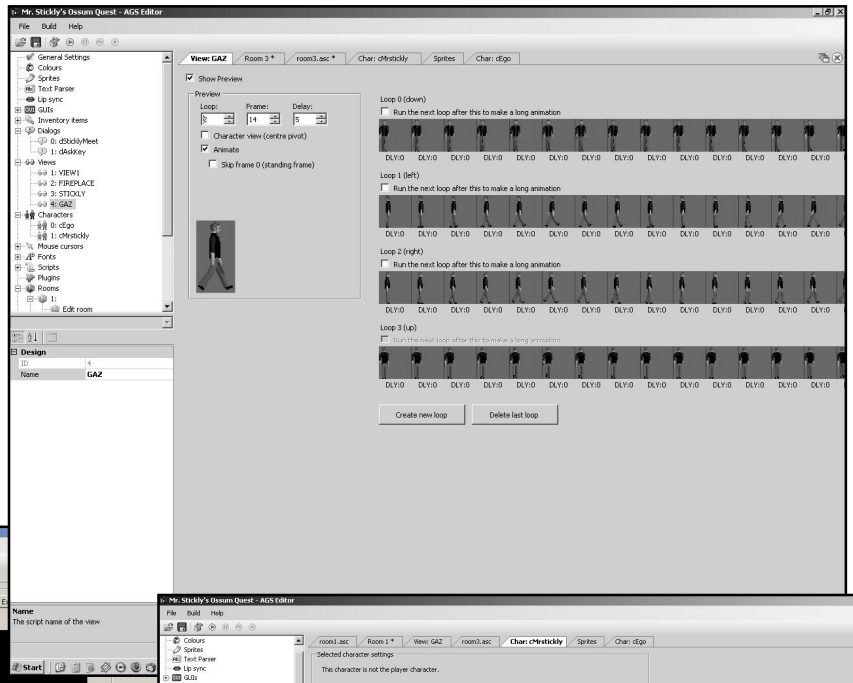
Back in the old days, when DOS ruled the operating system roost and memory management was king, lived the point-and-click adventure game. Among the earliest of genres, it was also among the most popular. Sadly, it soon fell out of favour with the growing PC game market and collapsed into obscurity, with only the odd commercial attempt to resurrect the glory days of using random items on other random items to see what worked.

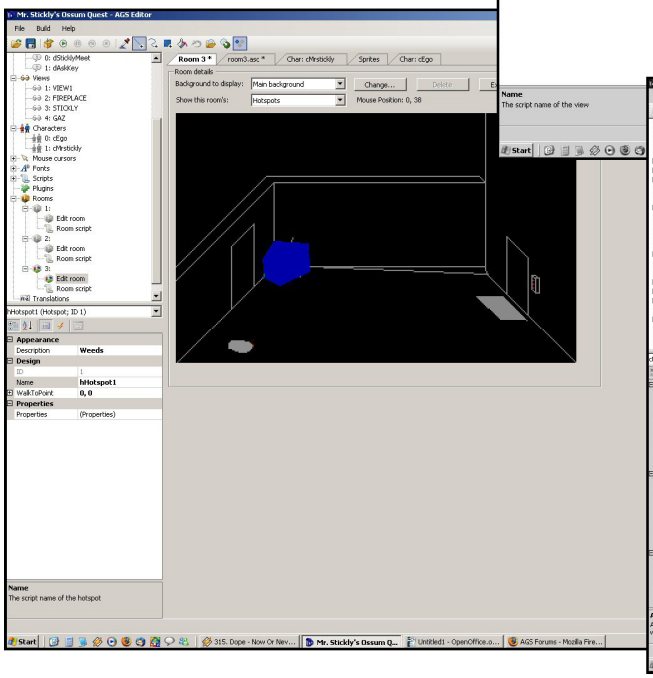
In the Indie scene, however, the point-and-click adventure is far from dead. It could be argued that this is in large part thanks to Chris Jones and his superb Adventure Game Studio package, which makes creating quality adventure games a whole lot easier. With version 3 just having been released, AGS is a powerful and flexible tool that allows you to construct your own adventures with ease.

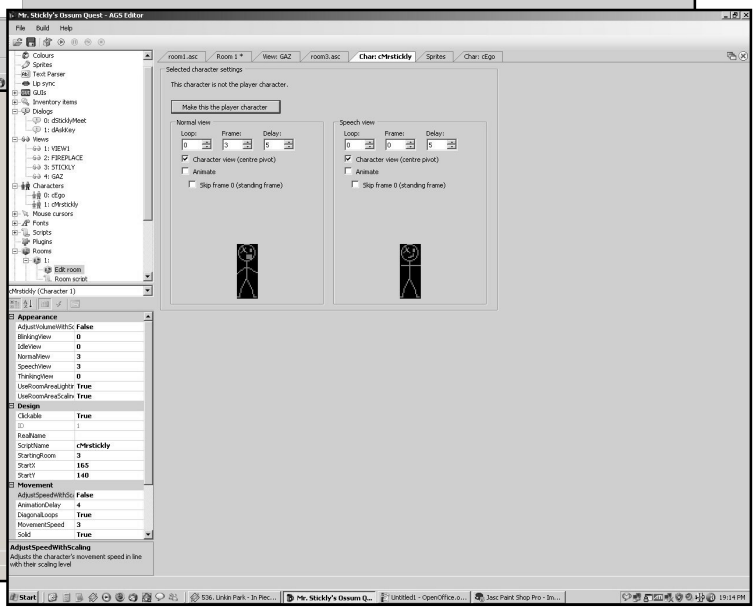
AGS makes use of a user-friendly GUI to manage game resources such as graphics and sound, and setting up rooms, characters, inventory items and objects is a relatively painless procedure. Unfortunately for some, most game interaction is controlled via scripts, so if you're one to shy away from a bit of programming this may not be the tool for you. Don't be too nervous though - the help file is robust and newbie-friendly, containing plenty of tutorials to get you up to speed even if you've never coded before in your life. In addition, the scripting may put newbies off, but it offers incredible flexibility for more experienced programmers, even to the point that non-adventure games have been built using AGS!

AGS has a highly active community built around it, and the official forums are simmering with activity. Everything from scripting help to community-made code modules (for the less technically minded) is available, not to mention being a great place to showcase your finished masterpieces.

If you love adventure games, and are itching to make your own without the hassle of coding your own engine from scratch, AGS is a fine choice. Unofficial Day of the Tentacle sequel, anyone?







GDC 2008

AN AFTER-ACTION REPORT

by Sven "FuzzYspoON" Bergstrom

What is the GDC, you ask? Well, what is the GDC not? There's so much going on at the annual Game Developers Conference that it's difficult to find the right place to start. Perhaps a quick history lesson: the year is 1987 and you are invited to a gathering of developers, 27 attendees and a living room. That's all it is. Yet, unlikely as it seems, this is the start of a burgeoning of interest and a great deal of planning to become unquestionably the most incredible gathering of game developers from across the world. Twenty-one years later sees the 2008 conference being held over a five-day period (18-22 February) in San Francisco.

Name any awesome game developer from years back and they'll probably be speaking at the GDC. Who wouldn't speaking at this grand construct of an event? Anybody who is anybody delivers content through an unparalleled medium with one focus in mind: to make games better.

The importance of team

Do you know how big this event REALLY is? Look more closely at the past. Where are all these great ideas coming from? Where can information be found on this technique? If you have read a game development article, it's likely that somewhere, somehow it originated from a branch of the information hub that GDC has become. The size and scale on which things are done doesn't happen by hoping to finish your epic FPS in a bedroom, but it does start with information. Hundreds and hundreds of people get together to build a well-oiled machine that delivers the goods.

Among the deliveries, a few great announcements for the indie game scene have surfaced in the shape of a taste of freedom. The freedom to release, market and even publish

your games on an already-networked hub of activity is a good thing for smaller bedroom-based developers. Of course, this is referring to the Microsoft keynote speech which informed developers about the freedom of the XBox live community releases. When their DreamBuildPlay competition was held, over 200 entries surfaced with a great outcome as stated by Microsoft.

This was all part of an attempt to stir up some creativity and get people looking towards the XNA platform for smaller games as well, and the announcement followed a quick chat with James Silva who Microsoft chose to represent the competition, was quite a shockwave through the minds of indie professionals and beginners world wide. Microsoft said that XBox would allow community-built games to be released through XBox live. The word used was "democratised", games being made available through the Live network would allow the community to control the content via a system which moves along the following lines: Create, Submit, Peer Review, Play. This means games that are worthwhile and which everybody is playing will be readily available as long as developers submit them. This takes the ten thousand players looking for something to do into your imagination – you do the math.

Another interesting announcement was that the XNA framework has been ported to the Zune – that's an MP3 player that can now play XNA-based games as well as use the music on the player as a resource for personalised audio and other cool features using sounds. The announcements were quite interesting among a host of other information, and previews, such as the Unreal engine from Epic Games running on the XBox 360, and a number of great sequels, including Fable 2 and, of course, Gears of War 2.

The IGF was also a big event, as always. Having been held and judged, the finalists and winners were announced at the GDC this year. A short wrap-up of the categories shows that there's great room for learning and getting recognized in the indie development scene, with prizes for efforts such as the

IGF 2008 AWARD WINNERS

Seumas McNally Grand Prize:

Crayon Physics Deluxe, by Kloonigames

Best Web Browser Game:

Iron Dukes, by One Ton Ghost

Design Innovation Award:

World Of Goo, by 2D Boy

Excellence in Visual Art:

Fez, by Kokoromi

Excellence in Audio:

Audiosurf, by Invisible Handlebar

Technical Excellence:

World Of Goo, by 2D Boy

Best Student Game:

Synaesthete, by Rolling Without Slipping

Audience Award:

Audiosurf, by Invisible Handlebar

Gleemie Awards:

First Place:

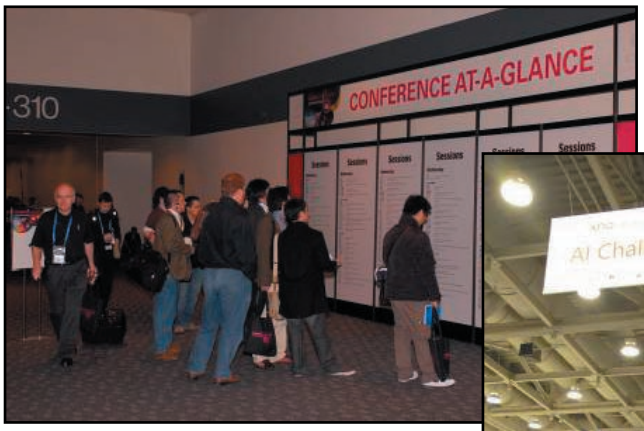
Desktop Tower Defense, by Handdrawngames

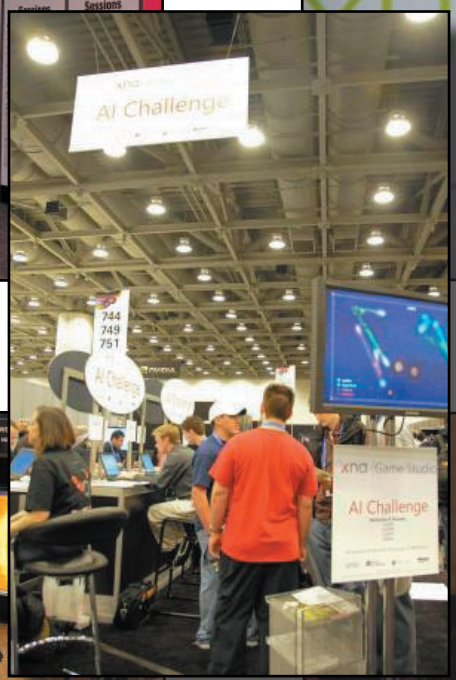
Second Place:

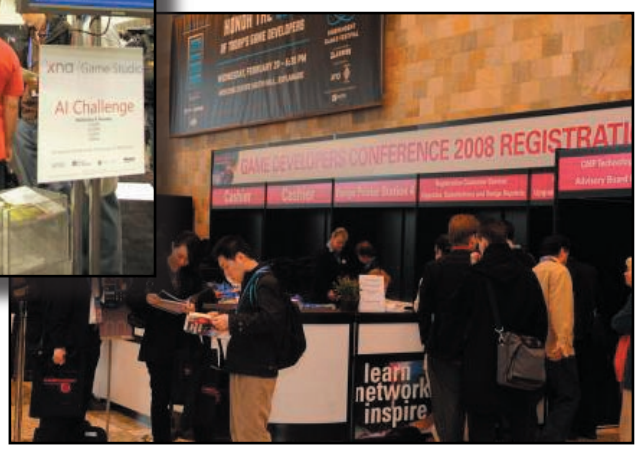
Skyrates, by Team Skyrates

Third Place:

Quadradius, by Quadradius







best web browser game, a design innovation award, excellence in audio, technical excellence and more. Check the boxout for a review of who won what.

The prizes included some huge cash prizes including the 20 000 US dollar award for the grand prize, as well as smaller prizes all round for the other categories. This definitely gives a lot of motivation to the indie communities worldwide and has inspired a lot of really well-polished, smaller games that can make it into the industry as a game of choice.

GDC is a showcase of well-placed advertising, great game marketing and all-round product promotion. Some interesting tech gadgets spotted included a PS3 Eye application, which tracks head movement and accurately updates a 3D scene, allowing a player to move as if looking around the world themselves. The other Eye application was a sketch pad of sorts. As the user sketched a drawing of a world, and a tank, the application created the sketch into a fully animated, collision-enabled game world that can be interacted with using the control pad.

All in all, another hugely successful conference was well-attended and well-managed. With content readily available on every conceivable page of the website, and even the notes available for download for a number of seminars, it's impossible not to learn something and get the most out of this year's conference.

DB SAYS ...

For a closer look at what GDC has been like this year, have a peek at these websites. They're full of awesome resources and cool information about the conference and the people who go there!

http://www.igf.com
http://www.gdconf.com