Issue 26

October 2008

DEV.MAG

CREATE • DEVELOP • EXPERIENCE

rAge 2008

INSIDE: Part 2 of our LUMA interview A Introversion speaks about Multiwinia A Audacity explained A More Blender A Did someone ask about Level Design? A Quad Trees A News + Reviews + Other stuff too!

CONTENTS

CAELESTIS
Claudio "Claudius" de Sa

DEMITTO CAELESTIS
James "Calamitas" Etherington-Smith

PULCRITUDO
Quinton "Voluptarius" Bronkhorst

SERVIOS
Rodain "Venustas" Joubert
Simon "Tr0jan" de la Rouviere
Ricky "SecusObdormio" Abell
William "Cairnswm...us?" Cairns
Danny "RecitoPessime" Day
Andre "Fengolus" Odendaal
Luke "FrigusManus" Lamothe
Rishal "IntegerProprius" Hurbans
Gareth "GazzanusEnios" Wilcock
Sven "TergumChucciaios" Bergstrom
Kyle "ErepoCaudos" van Duffelen
Chris "LeoPenitus" Dudley
Herman Tulleken

NUNTIUS
Robbie "Squid" Fraser

DOMUS
www.devmag.org.za

CELER NUNTIUS
devmag@gmail.com

This magazine is a project of the South African Game.Dev community. Visit us at:
www.devmag.org.za

I've stealthily hidden a purple ninja somewhere in this issue. If you find him; stop taking drugs immediately! Also. Burning ducks. Why has no one made this a basis for a game yet?

DreamBuildPlay is done, rAge preparations are nearing conclusion and we're all looking forward to a crazy weekend where most of the Dev.Mag staff and Game.Dev community will actually meet each other, some for the first time. It's odd to consider that, even though Dev.Mag has been running for nearly three years, a large majority of the people who contribute to this magazine on a monthly basis are still faceless pseudonyms to me. Such is the curious nature of an online venture, where contributors are so disjoint yet still rather intimately connected.

And all the above means the busiest month of the year will be over by the time you read this. Although that last statement was a bit of a hopeful conjecture on my part, mostly because I cannot imagine a month that was busier than the last. More accurately, I cannot imagine the aftermath of such a month; finishing the largest game project I've ever been involved in as well as preparing for what is, quite possibly, the most important event of the year qualifies any month as Freakishly Busy. No person should need to endure many of these every year, and I shall certainly look forward to a break after the rAge dust storm settles.

However, in hindsight, all the effort was well worth it. By the end of it all, Game.Dev as a community will have two complete Xbox games under its collective belt, both of which were entered into a high-profile competition where even the slightest honourable mention will have a potentially gargantuan effect for our little fellowship of developers. Needless to say, we're all incredibly excited and hopeful. And proud.

That's enough blathering, though. Now comes the fun part, where I get to tell you about all the things that you'd already find on the index page, just in many more words. Most importantly, we have two feature interviews this month – a first for us. One is an excellent chat with Mark Morris of Introversion on Multiwinia, and another being the conclusion of the Luma interviews we started last month. We've got another audio-related piece discussing the interrelationships between bwumphs and blonks and other words that make spellcheckers cry, and our tailpiece goes back and looks at the history of Game.Dev's regular competitions, discussing what aspiring designers could learn from them, whether or not they have participated.

That's it for this month. Read, enjoy and get out there and make games!

Oh, and finally:
rAge!

~ Claudio, Editor

Luma Arcade on InstantAction

http://blog.instantaction.com/2008/07/blur-game-forme.html

The local team over at Luma Arcade has been slaving with the arcade racer BLUR, a game slated to be released into open beta on GarageGames' InstantAction games portal soon. The game, currently in a private beta testing phase, puts players behind the wheel of one of two selectable vehicles in an adrenaline fueled 8-player race. Be sure to try it out when it's available for public play.

Shred Nebula Design Documents released

http://www.gamecareerguide.com/features/603/documents_of_newly_published_xbox_.php

In an unprecedented move, James Goddard of CrunchTime Games released both design documents that were used during the Xbox Live Arcade pitch of his game, Shred Nebula. Both documents - a '60 seconds of gameplay' essay and the actual pitch/design document - are freely available to download and view at the site above. These should provide a valuable insight into how the usually hidden internal processes work for XBLA and, by extension, for most publishing deals.

Bioshock Postmortem available at Gamasutra

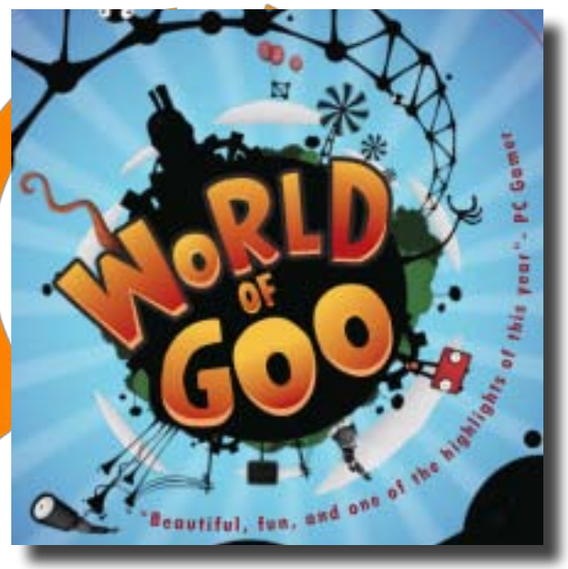
http://www.gamasutra.com/view/feature/3774/postmortem_2k_boston2k_.php

In a first for Gamasutra, notable postmortems and other articles from the vaunted Game Developer magazine will be published on the site. The first fruit of this new arrangement is this Bioshock postmortem, detailing the creation of the game from the perspective of project lead Alyssa Finley. Worth reading.

Retro-Remakes competition 2008

http://oddbob.wordpress.com/

Retro Remakes, a site dedicated to the retro gaming scene and retro-styled games, have launched their 2008 remake competition. The contest, running till 6 December, challenges developers to submit a freeware entry under any of 6 categories for over £5000 total prizes to be claimed by the first 10 places in the grand prize, winners of each category, and a special judges prize to be handed out at their discretion. Entries are open until 2 December.



World of Goo is Gold

http://2dboy.com/2008/09/09/pretty-big-news/

After a long, long wait, and a massive amount of evolution from the original Tower of Goo prototype, World of Goo has finally gone gold. Made by a tiny two-man team, World of Goo is a construction-based puzzle game that won the 2008 IGF Innovation award, an honour previously bestowed on XBLA's Braid and PSN's Everyday Shooter.

Torque X 3D Engine now bundled with Softimage|XSI Mod Tool Pro

http://www.gamedev.net/community/forums/topic.asp?topic_id=508463

The powerful, professional modelling and animation package made by Softimage, more popularly known for its use with Valve's Half Life 2, is now included absolutely free with a Torque X 3D engine license. The Torque X Engine contains flexible game authoring tools built on XNA and, with the inclusion of the XSI Mod Tool, should become a tool that every XNA developer would want to have at hand. With indie licenses available for only $250 it isn't far out of reach either.

"As always, we will live or die by our next title, and in this case it's Multiwinia."

Simon "Tr00jg" de la Rouviere

NOT SO INTROVERTED

Having thoroughly enjoyed playing through the preview code of Multiwinia (Dev.Mag issue 25) and bringing our readers a first look at the game, Dev.Mag decided to have with a chat with **Mark Morris,** the Managing Director of Introversion. Perhaps he can help us find our missing Darwinian…

Dev.Mag: Why did you decide to take Darwinia into multiplayer?

Mark: Just after we won the 2006 IGF (Independent Games Festival) award for Darwinia, we were talking with Microsoft about getting Darwinia onto Xbox Live Arcade. They were keen, but wanted us to include a multiplayer mode within the package. At first we thought this would be a quick job, but over time, we realized that we were really onto something. Multiwinia was really great fun to play and we decided to turn it into the next major release from Introversion. We had to work really hard to get such a variety of maps and different modes, but we are really pleased with the results!

Dev.Mag: How did the design process for the game modes work?

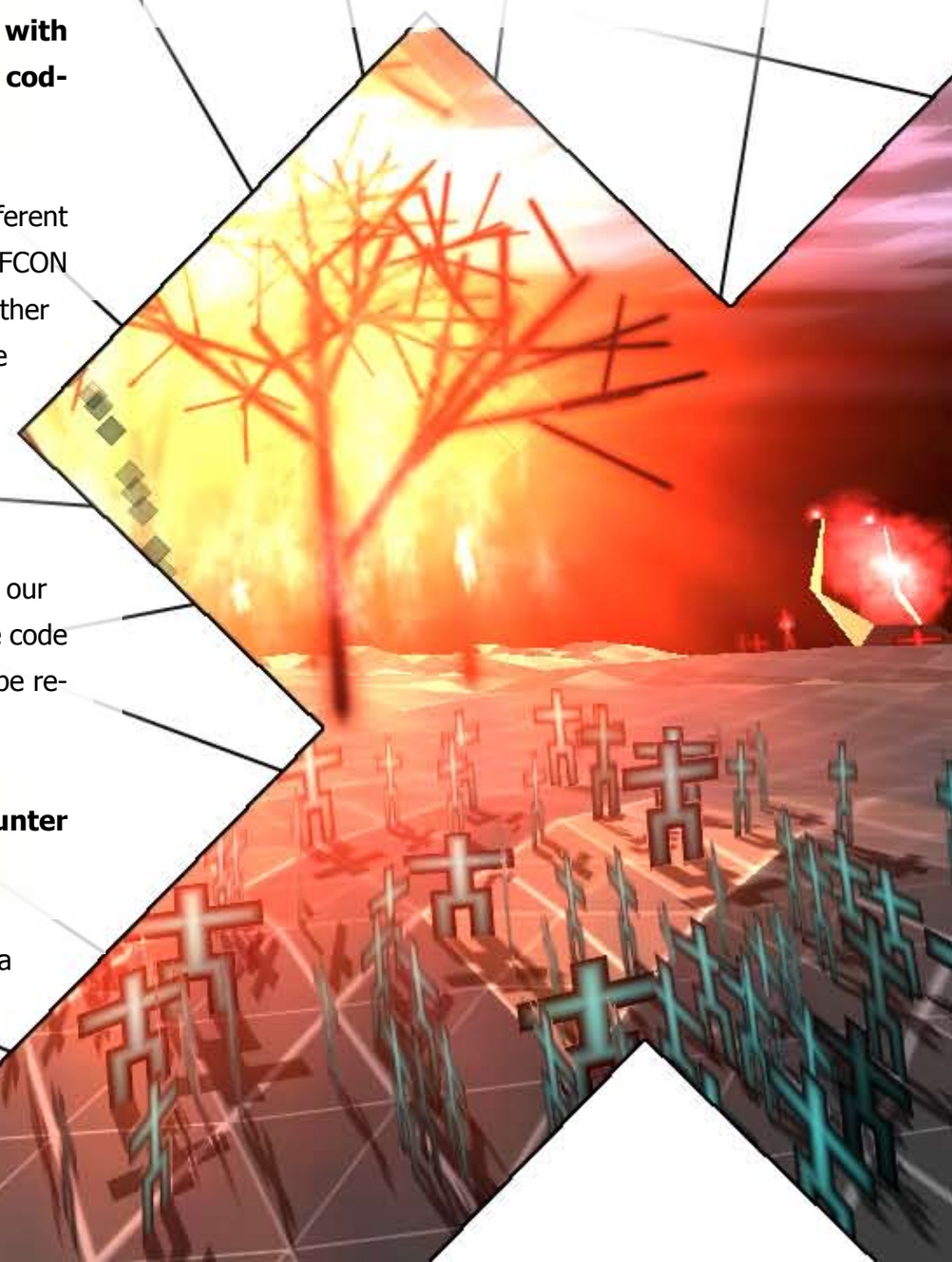
Mark: Design at introversion is really iterative. Some of the modes were really obvious, like King of the Hill; but the more complex modes, like Assault, Rocket Riot and Blitzkrieg, took us a long time to get right. Basically we would try something, see if it was fun or not, and develop it. Multiwinia is all about fun and action, and if we weren't having fun with a particular mode, we would try for a while and then scrap it if we couldn't get it right.

Dev.Mag: How much did DEFCON help with creating Multiwinia, from a design and coding perspective?

Mark: Multiwinia and DEFCON are very different games, so from a design perspective, DEFCON didn't influence Multiwinia, other than a rather unsubtle nod to DEFCON, included as a crate power-up. That said, we had learnt a lot about networking and game stability from our work on DEFCON. The problem was that we had to go back to the original Darwinia code, which wasn't as robust as our latter work. The result was that much of the code that we had written for DEFCON needed to be rewritten for Multiwinia.

Dev.Mag: What hurdles did you encounter with the development of Multiwinia?

Mark: The biggest challenge with Multiwinia was making sure that there was enough variation in the maps and the different game modes. We spent most of 2007 coming up with, testing and rejecting lots of game modes before settling on the final six. We're really pleased with the end results, and I'm sure that everyone will have their own favorite modes and maps. I personally love attacking in Assault.



Dev.Mag: What was it like working with the Xbox 360?

Mark: The 360 is a really great platform to work with. The console itself is basically a PC, but they do some clever stuff to get as much out of the hardware as possible, this meant that we were able to get a basic port up and running quickly, but the real challenge is getting the performance up. Of course we are still a little bit off the Live Arcade launch, so you'll need to ask me the question again in a few months!

Dev.Mag: Was it difficult to adapt the Xbox controls?

Mark: Multiwinia was designed with the 360 controller in mind, but it's Darwinia that is giving us the most trouble. We are nearly there, but we did think it was going to be a much easier job than it actually turned out to be. I think we have gone through about four or five control revisions and I'm still not sure we are there yet!

Dev.Mag: You've been going for quite some time. Do you foresee a shiny future ahead for you guys?

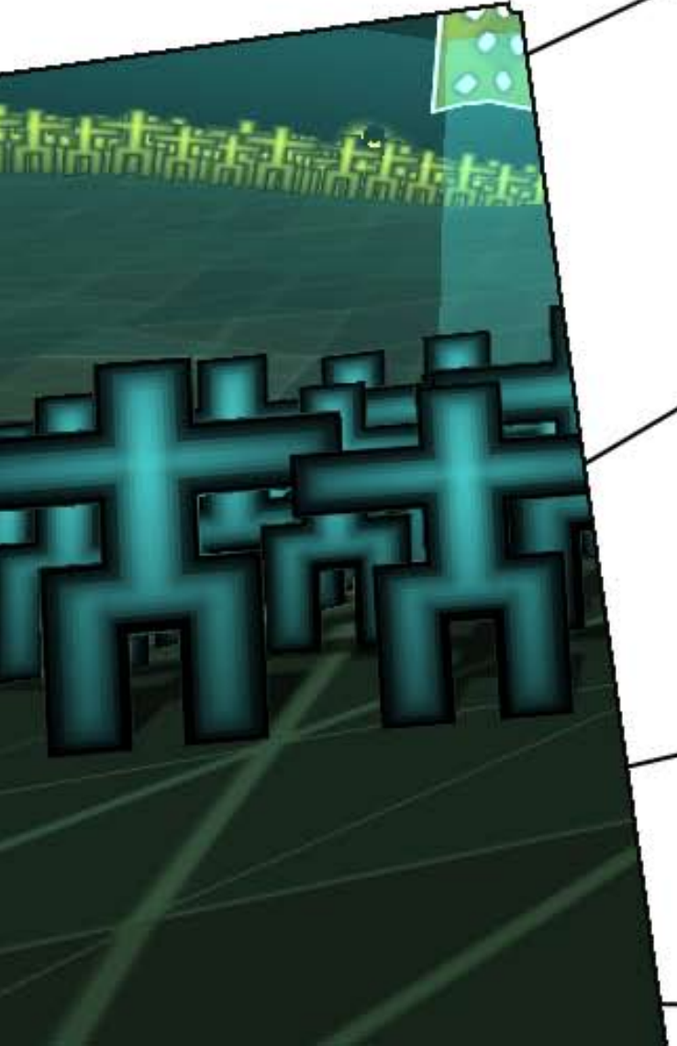
Mark: We're in a very strong position now. We're pretty well known in the industry, and our back catalogue of games still sells in reasonable numbers. We have a few ports in the pipeline, which should help to support the main effort; developing great new games. As always, we will live or die by our next title, and in this case it's Multiwinia. Keep your fingers crossed for us!

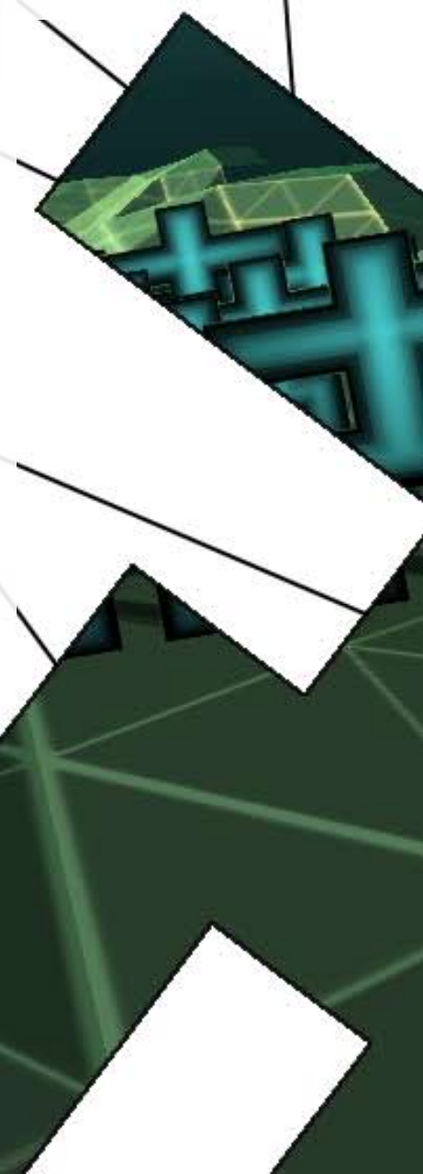
Dev.Mag: First you rattle our collective gaming minds with Subversion, and then we hear about Chronometer. When are we going to get some info on this? When do you plan to release it?

Mark: *smiles roguishly*

Dev.Mag: How does your design process work in general? How do you get your wild ideas onto paper and then into a solid game?

Mark: We have a very relaxed and free flowing design process. Chris has most of the big ideas and then he'll go away and jam for a few months. It probably takes him about six months, working part time, for him to come up with the core of a game. At that point we bring in Gary; he is our Chris multiplier. Gary is great at seeing the themes that Chris has created and jamming around them. Show Gary a map, and he'll come back a week later with six. During this time, the other members of the team will be adding new features or extending ideas as directed by Chris. Once we start to run out of money, we all get scared and we work like mad for a few months to get the game bug free. Sound simple?





Dev.Mag: Do you plan on releasing Darwinia merchandise? Those Darwinians are a hot commodity.

Mark: We already have a store full of cool Darwinia stuff. Check out **http://store.introversion.co.uk**

Dev.Mag: The entry point for great indie games is getting higher and higher? Don't you think it is discouraging for beginners?

Mark: Indie games are getting bigger and better, and there are more and more tools out there to support the development effort. I'm talking about the likes of XNA. This does make it a lot harder for new people to get set up and producing games, but I don't think that is a bad thing. It is very, very hard to make a great video game, but if you want it enough, you'll get there!

Dev.Mag: After working on the Xbox, do you still see the PC as your main platform?

Mark: Yeah, we will always be principally a PC developer. The PC is an open platform and that means that nobody pushes us around or tells us what to do. We are fiercely independent, and whilst I'd love to see our stuff on other platforms, the big boys need to understand that they can't have a hand in the design process. That said, Microsoft have been really helpful and have given us loads of feedback on how to improve Multiwinia.

Short and Sweet

MARK MORRIS

Peanuts or Raisins?
Diablo 3 or StarCraft 2?
DRM or No DRM?

Mark: Listening to DRM-free MP3's whilst eating peanuts and playing StarCraft 2.

Last issue, we featured Luke Lamothe from Luma, talking about their latest offering, BLUR - this issue, we finish off by talking to BLUR's lead artist, **Chris Cunnington**, and creative director, **Dale Best,** about some of the challenges they faced bringing out this anticipated offering!

BLURRING The lines

Sven 'Fuzzyspoon' Bergstrom

CHRIS CUNNINGTON

Dev.Mag: As lead artist on the BLUR project, which aspects brought new challenges?

We came into BLUR just off the back of MINI#37. With MINI we had been very restricted by TGE in the way we produced the road & pavement sections, not allowing us to make flowing, curved roads with textures that followed the curves. So heading into BLUR one of the first things I wanted to re-engineer was our road production. The outcome of a couple of weeks work was a modular system, that allowed for completely flowing surfaces. The drawbacks were that for each single road section, I had two sections, one you see and one you don't. The visual improvement over MINI#37 completely outweighed the drawbacks of handling double the art assets. Otherwise, we generally pushed the bar as much as we could, as we always will, allowing a huge visual leap!

Dev.Mag: With developing for a browser environment, were there any art limitations, compared to desktop games?

The beauty of the InstantAction platform is that it is able to load any game engine – obviously with a few tech changes – into the browser. So when we set out creating BLUR, we never limited ourselves at all with any thoughts of, "It's an in browser game." We created it as if it was a standard PC game. We actually only got it into the browser very late on in the project. The only limitation we then set on ourselves was file size. We wanted to keep it small and neat so that people don't need to download huge amounts of data.

Dev.Mag: Working on worlds with good detail has its perks. What was your favourite map or scene to create?

Hmmm, that's a tough one! Each level has its own unique quality that jumps out for me. SkyCity was the first to be built, and I really dig the shiny clean feel of this world. ParadiseCove was planned to be more a gritty, graffiti-cum-old–European, coastal town; but it got changed slowly into the world it is now. Now it is awesome; who can complain about a level filled with Palm Trees? Reactor Station was great fun to make. We wanted an industrial, piping filled world, but went wild with it when we added a huge glowing reactor to the scene. So yeah, worlds with more detail are awesome, even though racing games still don't let you get to the level of detail needed for a modern FPS.

Dev.Mag: As an artist, what 'ideal' map would you have made?

Hmmm, well, for a racing game, I pretty much got my wish with ParadiseCove, making an island beach level. Moving away from racing games, I am very keen to start building more open, free worlds, with a more sandbox feel!

Dev.Mag: Tips for aspiring 3D game artists?

Be prepared! Game art is not all that it appears on game-artisans.org. A regular day is easily split between working out bugs with your art pipeline, asset management, and somewhere in between, making new art assets.

Dev.Mag: Any last comments?

As with most games, everyone always looks at the quality of the art, but let's not forget the programmers. As a game artist, I have to say a big thanks to the programming guys. Without them so many things would not have been possible visually. Being able to ask for a custom tool to be made to help your workflow, and watching it being developed, is the coolest thing ever!

DALE BEST

Dev.Mag: What part did you play in the overall production of this game?

I oversaw the creative direction in terms of style and game play, as well as the direction of the levels and cars.

Dev.Mag: What aspect slowed you down most as a creative director?

Once I've had my third cup of coffee, there's no place for slowing down. We had milestones to meet as a requirement by our publisher, and that's that.

Dev.Mag: What aspects of the game were swayed by InstantAction integration, in terms of design changes?

Well, the whole idea around InstantAction is that of 'pick up and play', so that is a major consideration in the game design. Pushing the game into the, 'hard to master' territory also is key, because you want to keep your players coming back. We feel the game is nicely balanced in this regard, and continue to get feedback from the closed beta stage it's currently in.

Dev.Mag: If you were to make another racing game for InstantAction, what would you choose?

LumaArcade won't be making another racing game for a while, unless we have to. We may incorporate a racing component into new games we develop if the design requires that. BLUR will continue to grow over time, with new car packs that users can buy, and new tracks as well. The game will grow, it won't be replaced. We are happy with BLUR, and wouldn't do it another way. The old school arcade quality is spot on for the platform, as far as I'm concerned.

Dev.Mag: Any tips for aspiring creative directors?

Well, I just kind of moved 'organically' into this position, and usually that's the case. It becomes more of a managerial role, but I'm still hands on, which is cool. Just do what you enjoy doing. Take it seriously enough to do well, but try to keep a balance in your life. When you start getting symptoms of carpal tunnel syndrome, you know you need to get out more!

http://www.gamedev.za.net/filecloset/data/files/589/GlyphHunter.zip

GLYPH HUNTER

"THE ADDICTION LIES IN THE DIFFICULTY."

There exists a theory; that to truly express your hatred of the zombies in Glyph Hunter, would bring about a black hole of contempt, capable of devouring the universe and replacing it with something even more frustrating. There is conjecture that this has already happened.

Chris "Braaaaaaains" Dudley

That said, this is not a horrible game in any way. While a mixture of dungeons, monsters, and swords, is hardly the most innovative combination in the industry, for local indie developer Rodain "Nandrew" Joubert to take those elements and create a game as enjoyable as this one is quite a feat. Such a lot of quality has been packed into this title, that it is easy to get drawn into the action and forget the generic 'magical quest' storyline.

The meat-and-potatoes of the game involves hacking, slashing, and various combinations of the two deadly assaults. The tutorial gives you the gist of the simple gameplay, throwing a few enemies at the player to get them started. After that, the game lets the player get to the carnage, with the occasional pop-up informing them of their progress. For the most part, the gameplay is cut-and-dried; hack through monstrous hordes to a certain point, flip a switch, slash all the way back through more monstrous hordes.

Along the way, tension and addictive frustration mounts as the player watches their life slowly dwindle. Mana steadily drops as they frantically attempt to dispatch enemy mages who are lobbing flaming balls of death. Desperate whimpers escape as hordes of respawning zombies claw for the jugular. It's at these moments that the game shines; it creates tense, enjoyable fights that entertain and challenge (boy-oh-boy, do they challenge).

This is where the aforementioned zombies come into play, with their unsettling obsession of having a chunk of cranium for dinner. The zombies don't merely gang up, they swarm. They amass. No matter how many times the player shoves the business end of a sharp metal object into their faces, they pick themselves up and stagger on, with a chorus of "You're never gonna keep me down!" A quick fireball will turn them into a zombie flambé, and send their tortured souls back to hell permanently; but mana has to be used sparingly if the player is to make it past the ghouls, and to the haven of the next save point. Once there, they can rest assured with the knowledge that the next section will prove to be far more difficult.

The sense of euphoria that the player receives upon slipping into safety with their avatar on the verge of a grisly death is so immensely satisfying, that they will suddenly find themselves unable to tear away from it, striving to reach 'just one more checkpoint." The addiction lies in the difficulty though - be warned - this is not the most relaxing of games, but still a high quality production, coming out of the local dev scene.

Warning: side effects of Glyph Hunter may include hair loss; the invention of new swearwords; and spontaneous bursts of animalistic battle cries.

http://lostgarden.com/

Lost Garden

Claudio "Chippit" de Sa

Solid game design knowledge is traditionally kept close to the hearts of those who possess it; rarely does one find people willing to divulge their insights into the art and even rarer still are those who are willing to do so regularly, for no gain whatsoever other than the simple act of seeing others benefit because of it.

Daniel Cook is one such person. Also a contributor to the well-known development related sites GameDev.net and Gamasutra.com, Cook – under the moniker Danc – helms a game design blog named Lost Garden, where he regularly posts insightful game design essays and thoughts, provides free hand-drawn art for game prototypes, and occasionally challenges his readers to create game prototypes based on his design and theme.

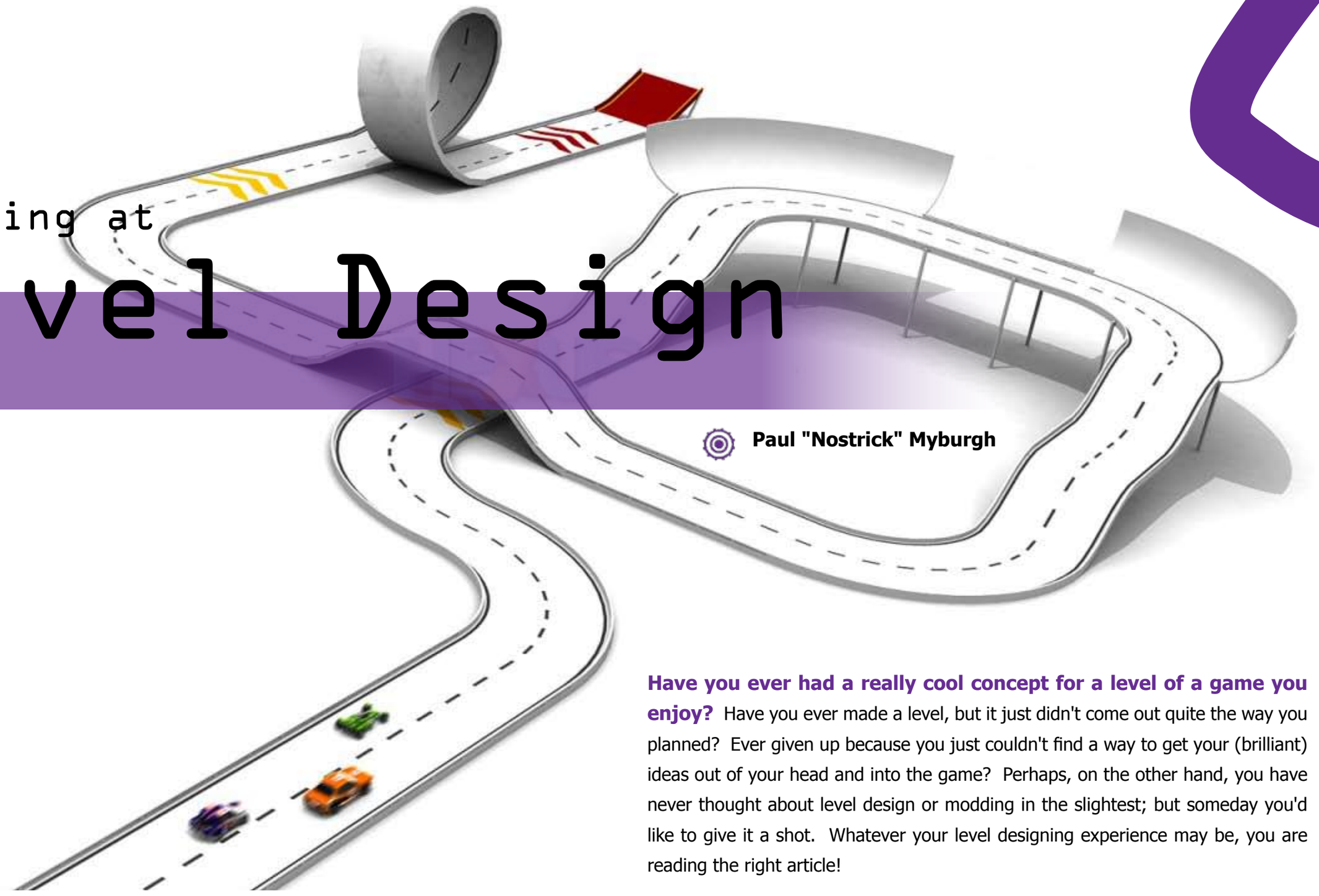
One such challenge, concluding just last month, tasked developers with creating a game in which world shadows are an inherent part of the game dynamic. A player would need to lead mushrooms harvested from the world back to a 'home' point. Since the mushrooms would shrivel and die in the sun, the player would need to hug dynamically changing shadows created by the changing time of day to achieve the best performance.

The game design insights posted on Lost Garden are also well presented and offer incredibly handy knowledge for the indie developer. In fact, all the offerings on the site will be invaluable for an independent game developer: design challenges to test your skill; design essays to impart useful tips; insights and knowledge; and artwork to facilitate your masterpiece creation. All in all, a priceless resource that should live on anyone's bookmark list.

"GAME DESIGN INSIGHTS POSTED ON LOST GARDEN ARE WELL PRESENTED AND OFFER INCREDIBLY HANDY KNOWLEDGE."

Looking at

Level Design



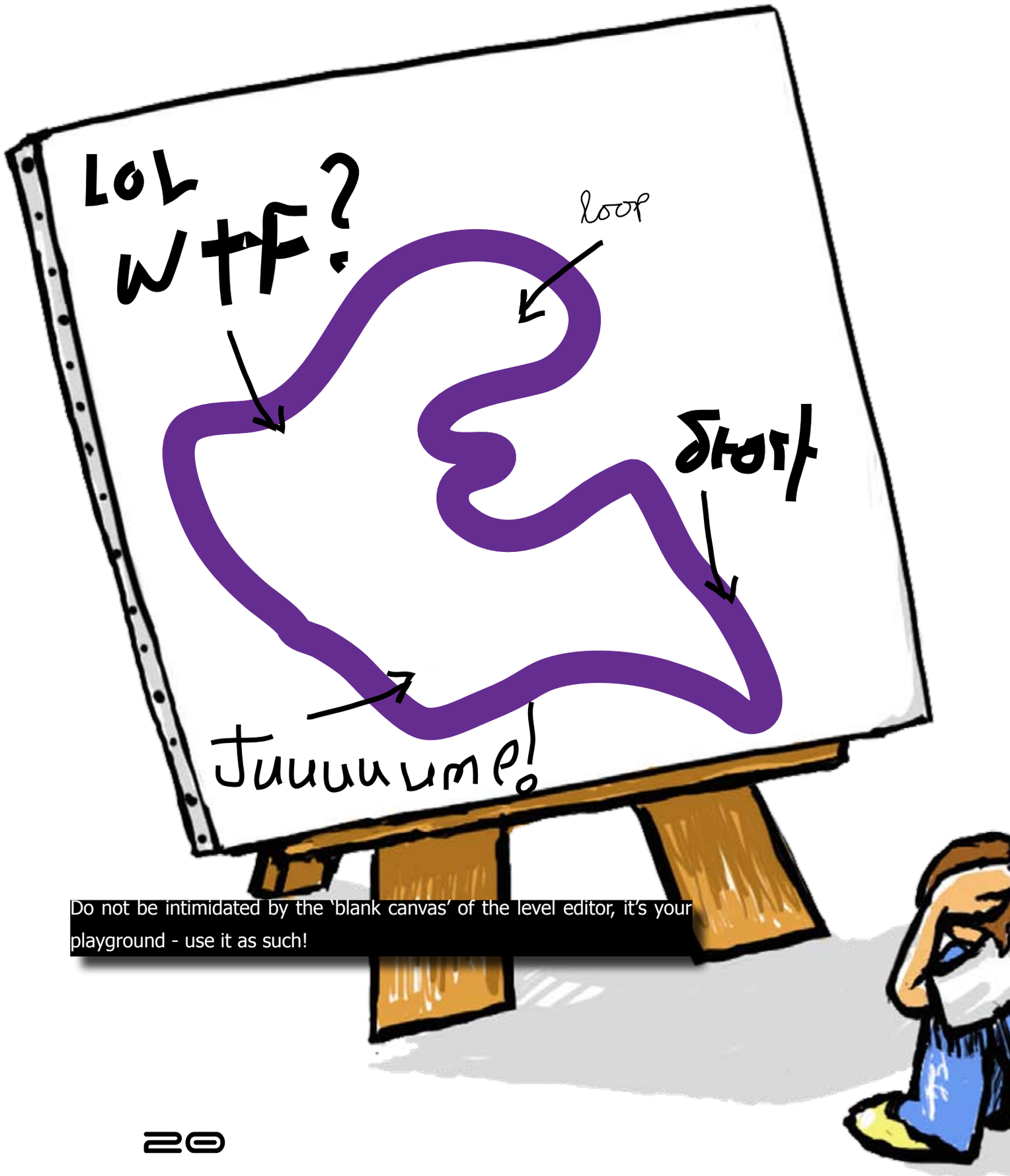
Paul "Nostrick" Myburgh

Have you ever had a really cool concept for a level of a game you enjoy? Have you ever made a level, but it just didn't come out quite the way you planned? Ever given up because you just couldn't find a way to get your (brilliant) ideas out of your head and into the game? Perhaps, on the other hand, you have never thought about level design or modding in the slightest; but someday you'd like to give it a shot. Whatever your level designing experience may be, you are reading the right article!

Here we aim to give you a grasp of the design aspect of level creation; it's a little something on how one might go about the design process, the process by which you bring your ideas into being, to create a playable and fantastic level for all to enjoy! Although simplified, this explanation should be effective enough to get you on your way. To explain this process, we are going to use the example of building a level (or track) in Track Mania. Explanation will be as generic as possible, so that you may take these basic concepts and apply them in other games as well.

To understand what makes a good level in a game, it's always best to know the game well beforehand; experiment with the game play mechanics, and take note of what makes the levels fun to play. There is no better example to follow in level design than that set by the creators of the game. The most downloaded and highly rated user levels are also great examples, so you might want to find a good website with a database of levels pertaining to your game of choice. Start by gathering ideas from these designers, and taking that inspiration to fuel your own creations.

Once kitted up with a bit of knowledge and inspiration, it's time to get started on the actual building and design process. We have an idea, now how do we go about making a track? Do not be intimidated by the 'blank canvas' of the level editor; it is your playground, so treat it as such. Just go at it with everything you have, and don't stop to think twice. Begin by laying down the basic level design; similar to sketching outlines on a drawing before you clean up and shade in. In Track Mania, run tracks all around, add loops, corners and little jumps. Just let loose; allow that idea in your head to flow down your arm, into the mouse and onto the screen!



Do not be intimidated by the 'blank canvas' of the level editor, it's your playground - use it as such!

Always begin rough; treat it as a draft. Don't become concerned over little things such as how the player is going to take the first corner, or perfecting the first jump. This is just a distraction and will be detrimental to your master plan; leave it for later.

Once you've experimented a bit with the rough draft, and found something that works, we move on to the next step; the flow. This is where play-testing begins to play a role, as refinements to the roughness of the level take place, shaping it into something more playable. This is a very important step. Ask yourself; could someone else race on this track (or play through this level) without becoming too frustrated? Make sure that the overall flow of your level provides a fun experience, while still providing a challenge.

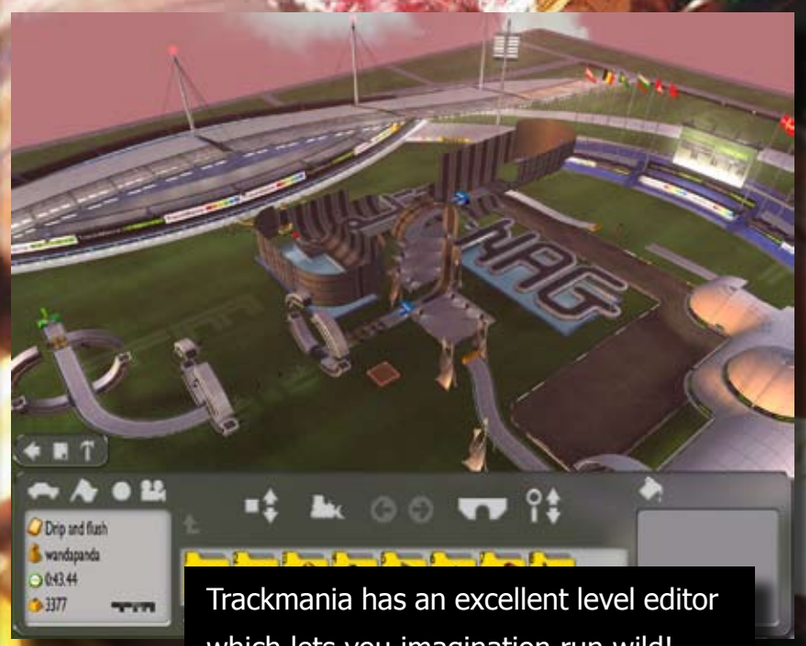
By first creating the 'skeleton' of your level, you will find it easy to go back and add more interesting and fun ways to link areas together, and fine tune certain areas for an even better experience. You should constantly have your mind focused on how the first-timer to your level would experience it and how they will either grow to like it or hate it. Never, ever, forget about the end user.

Once finished with fine tuning the mechanics of the level, the beautification begins. Try to add hints, such as which way one should be turning next, or in which direction you should be facing to make a jump, by strategically placing surroundings. As much as some people find beautifying a level/track unimportant, I'd say it plays a big role in the overall experience, but remember - don't overdo it! Sometimes less is better than more!

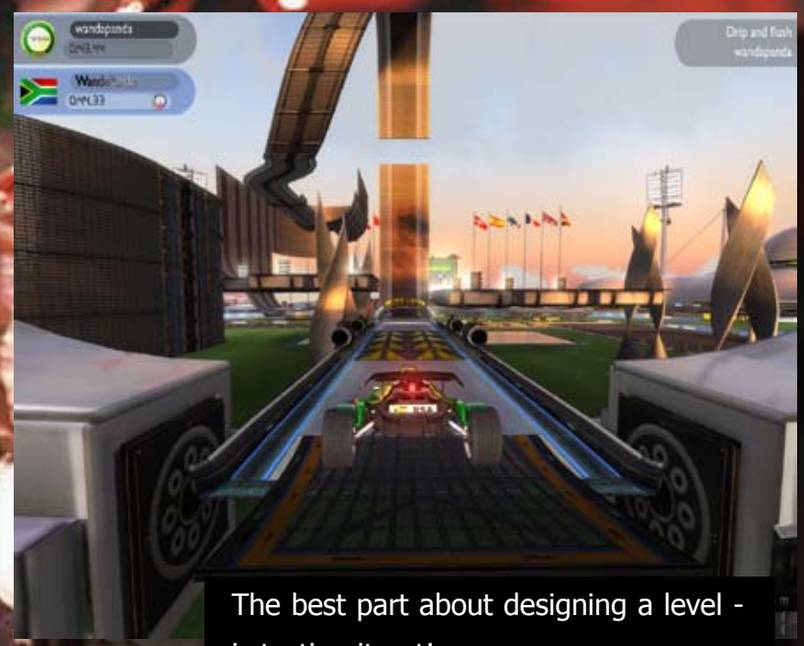
Finally, and most importantly, have lots of fun! It should be something you do in good spirits and enjoy as you design and play-test your very own creation. If you are having fun, you will be more involved and proud of your creation, and you might find yourself tweaking and play-testing to perfection right through the night!

So, find a fun game, get those creative juices flowing and just go at it. You might be surprised by the awesome things you can come up with. Happy level making everyone!

When design ideas run low, nature is often our best resource



Trackmania has an excellent level editor which lets you imagination run wild!

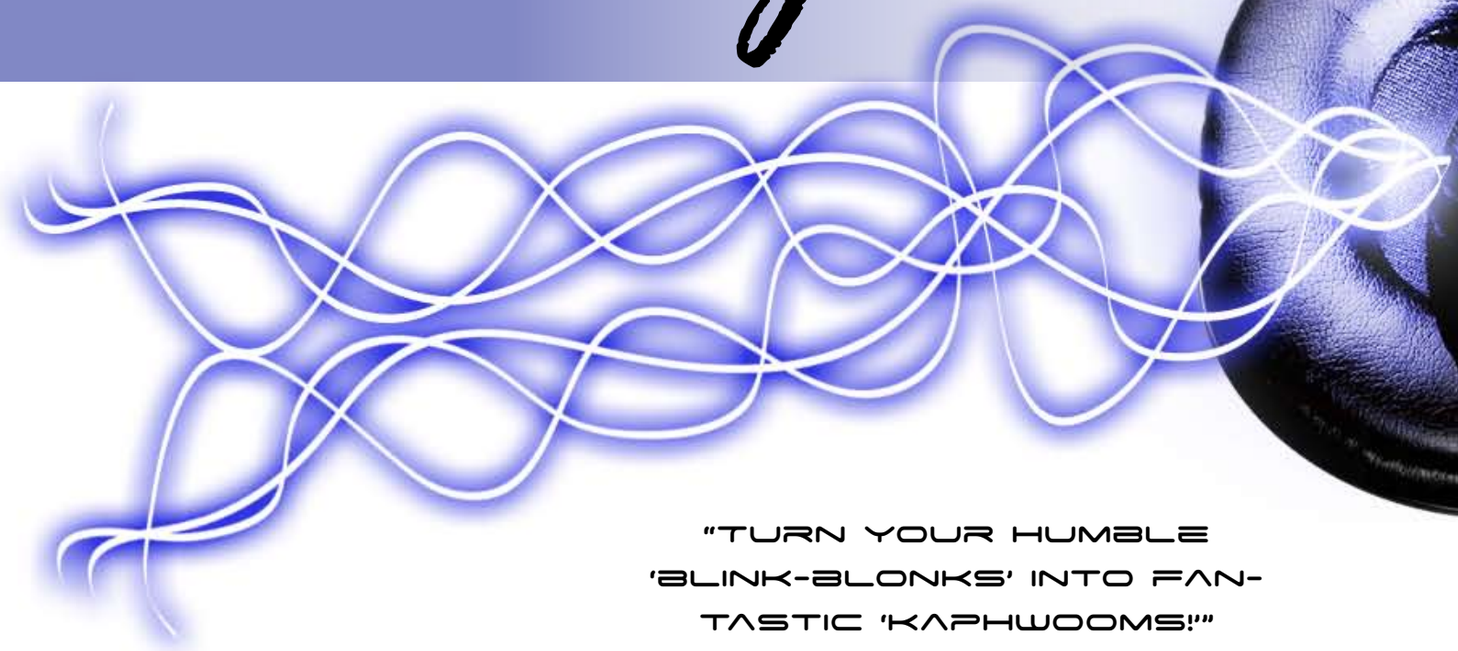


The best part about designing a level - is testing it out!

IN CASE OF EMERGENCY
BREAK THE MOULD

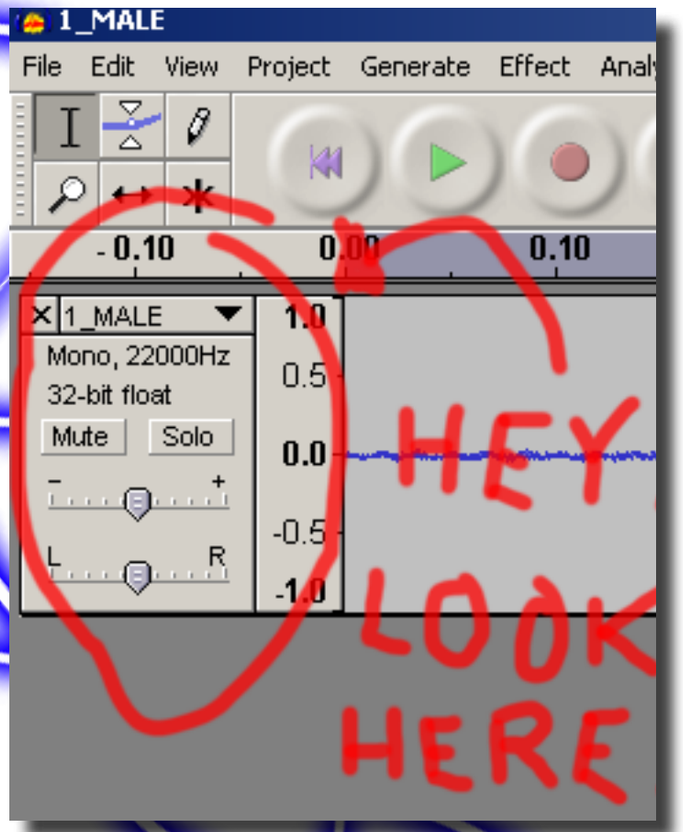
Interested in making your own sound effects for videogames?
This month we'll be looking at Audacity and a few of the common effects that can be used to turn your humble "blink-blonks" into fantastic "kaphwooms!"

Looking at Effects with Audacity



"TURN YOUR HUMBLE 'BLINK-BLONKS' INTO FANTASTIC 'KAPHWOOMS!'"

This article deals with the mildly technical side of sound production in Audacity, so it'll assume that you already have a wave file loaded up and ready for priming. In terms of format, you may want to look for a file that's based on the standard PCM WAV structure (it's the most common WAV file format, so you're probably using it already) with a 16-bit, 44100Hz quality (these details will be shown in the info box to the left of the file when you load it up in Audacity.)



You can also opt for a stereo sound format, but mono means a smaller file size and will usually do the job just fine unless you specifically need two channels.

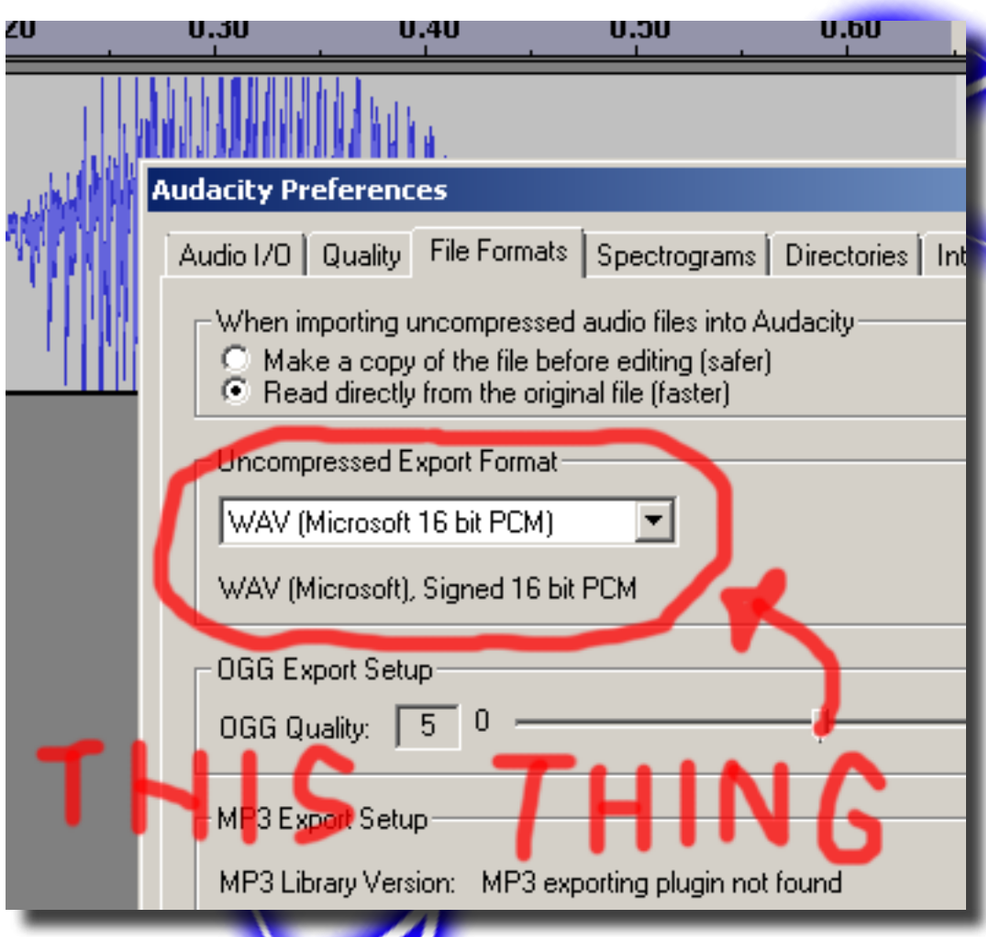
So, to clarify, you'll prefer these settings for an input file:

- PCM WAV format;
- 16 bit;
- 44100Hz (or similar);
- Mono.

Higher quality is optional, but isn't always necessary.

If you really can't find an input file to meet these requirements, no biggie – you can still try to convert them to the desired format by clicking on the filename in the left info box and selecting the necessary properties in the pop-up menu. Also check the Edit -> Preferences -> File Format menu to make sure that your uncompressed export format is set to the 16-bit PCM.

Maybe some of you remember that funky little sound idea thing back in Issue 24? If you haven't checked it out yet, don't panic – you'll be able to understand this article just the same, but try and have a gander at it anyway. Also, be sure to grab your own copy of Audacity from **http://audacity.sourceforge.net/**. Right, now that we've got all the details out of the way, let's look at the interesting stuff.



The Audacity effects

Select all, or part of the file that you've loaded up in Audacity (this can be done by clicking and dragging the mouse over the desired file component). This is going to be the section of the file that you apply your effects and filters to. Now select Effects from the top menu. You'll see a whole list of neat things that can be done to the innocent sound file which is now under your control.

We won't be looking at every effect that's on display, but a few of the simpler ones will be covered. What follows is a description of several effects, giving you their job, their potential for game development and a few useful pointers to get you going in the right direction. Let's go!

Change Pitch

Technical Description: This effect alters, well, the pitch of the sound. How high or low the notes are, so to speak. You can plug in a spoken sentence and decrease the pitch for a deep, manly-man voice or conversely increase the pitch to make it sound like a chipmunk. Wheee!

Game Use: This is one of the most commonly used effects to tweak sounds. You may want to change the pitch of an in-game explosion. Perhaps you have a piano somewhere in your game and you want to get several pitch variations of the same sound clip. Or you have a nice cartoony game where you want to take a standard set of sound effects and make them all cutesy by swinging the pitch up.

Hints: This filter is great if you want to speak in your game but need to mask your voice or simply make it sound cooler. Lowering or heightening the pitch ever-so-slightly will greatly improve the sound in your own ears, and you could potentially voice several in-game characters simply by altering the way you speak each time and applying a pitch effect.

Change Tempo

Technical Description: This is the counterpart of changing pitch. Do you want the same sound to play much faster or a little bit slower than the one you currently have? Tempo can make a "bwooooooooooom" into a "bwumph," and vice versa.

Game Use: Maybe you want a quick and tiny explosion noise but only have the Manhattan Project on hand. Or maybe your character is using a weapon with a high rate of fire and you've only got sound effects which last for at least two seconds. No problemo! Increase the tempo until you're able to justifiably go "powpowpow" for as long as your character needs.

Hint: By doing some extreme compression or extension of sound effects with the tempo changer, you'll actually start hearing some very, very weird things. This can be rather neat if you're looking for some exotic and/or sci-fi sound effects, so give it a shot.

Change Speed

Technical Description: This function is the equivalent of changing both the pitch and tempo in one go. A higher pitch delivers a faster tempo, and vice versa.

Game Use: If you're going to be using both pitch and tempo change for a sound effect, this can be handy for doing it with one effect.

Fade In / Out

Technical Description: Gradually brings a sound clip from zero volume to full, or vice versa.

Game Use: A very specific effect which is probably most useful for tailoring background music or longer sound effects (power up and power down sequences, for example). Can replace amplify in certain circumstances.

Echo (Echo)

Technical Description: Adds a basic echo.

Game Use: Handy for dramatic announcements or sound effects in a cave.

Hints: Most uses for the echo will involve decreasing the default delay time. A high delay time may sound good in certain areas (experimenting to find exotic sound effects is great) but generally it just makes the effect rather confusing.

Reverse

Technical Description: Puts the sound clip back-to-front, so that the end plays first.

Game Use: Can be used for a wide range of funky effects. If you're keen to experiment with a sound, try reversing it and see how it comes out!

Hints: If you know how to use stereo tracks in Audacity (it's a more convoluted process than you may experience in some other programs – check Audacity's help file for more details), you can reverse one of the channels and leave the other one playing normally. This occasionally grants a really cool effect.

Amplify

Technical Description: Increases or decreases the volume of the selected sound clip.

Game Use: This is mainly to place emphasis on a certain portion of a sound. For example, if you want an explosion to start off loud and then trail off, you can use amplification to tweak various parts of the sound. You can also use it to get all the sound effects in your game on acceptable volume levels. There's no point in your water sound effects drowning out the sound of loud bangs, after all.

Hints: If you're recording your own sounds, don't rely too heavily on amplify to fix your volume. Making sounds louder will increase the chance of background noise becoming noticeable. Conversely, screaming into the microphone and then reducing the volume probably won't get rid of the distortion that sometimes crops up. Use amplification for minor tweaking only. Also note that there are several more advanced amplification tools in Audacity (Normalize, Compressor, Equalize, etc). Learn to use these if you want to fine-tune, or perform finicky volume tasks, otherwise you should be able to ignore them.

Remove Clicks

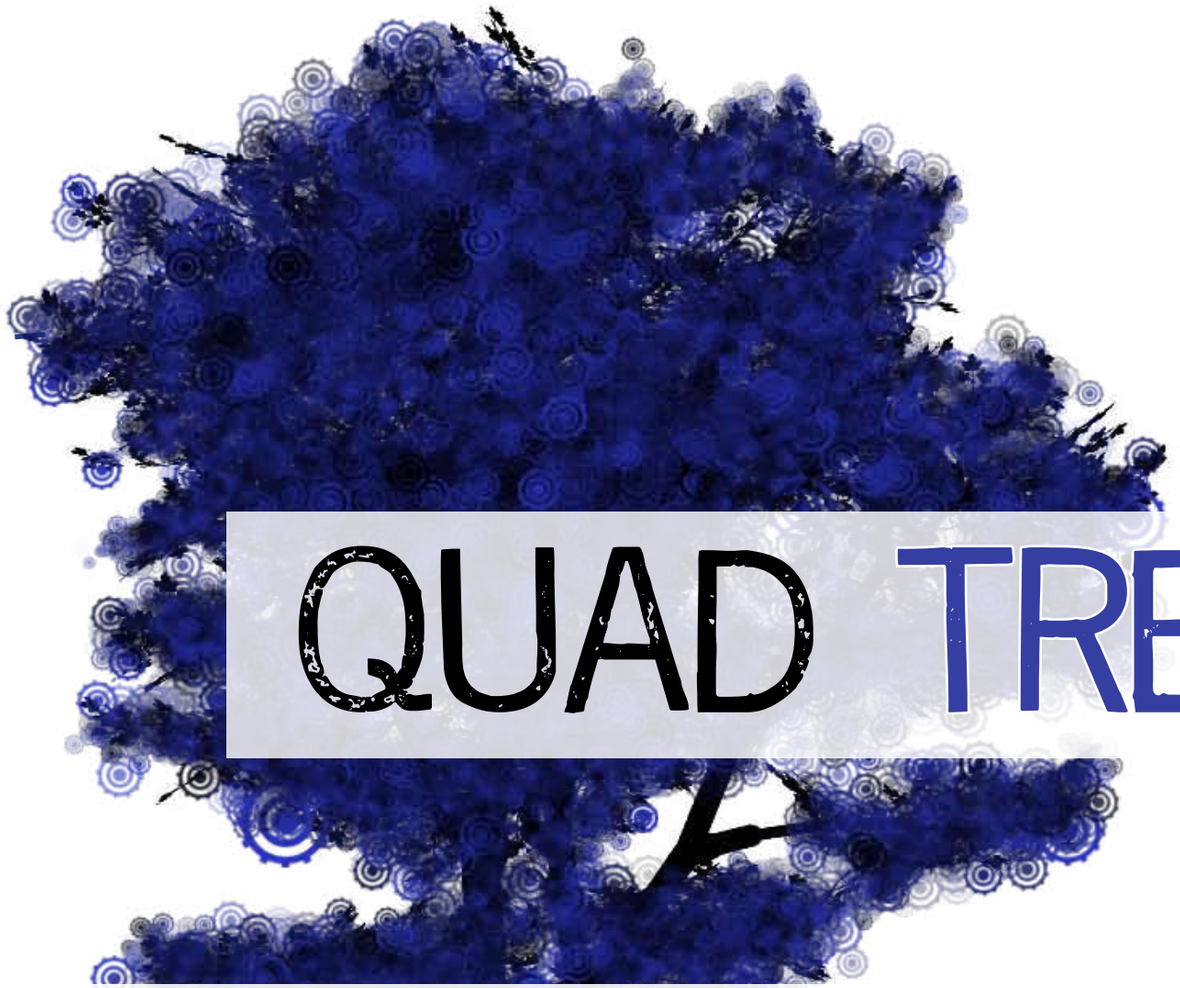
Technical Description: This is the easiest way to remove that horrible crackly effect that one typically encounters when using amateur recording equipment. Make your sounds crisper and less 'polluted' with this filter.

Game Use: If you've got 'clean' sound effects mixed with crumbly messes in your game, it helps to apply this effect to the culprits.

Hints: This isn't a perfect effect, so try not to rely on it too heavily. Rather make an effort to generate a smooth sound before it goes into the Audacity editor. If you truly feel confident, try using the Noise Removal effect instead (this will require you to capture a separate noise profile and then use it to remove noise in other parts of your clip).

In Conclusion

These are just a few of the simpler effects in Audacity. Fiddling with some of the more complex tools can lend more interesting effects, but the point of this tutorial is to provide you with the basics, to be able to confidently tweak your own sounds and get rid of the more glaring problems in your files. If you seriously want to go into sound editing and file fixing, it's worthwhile to consider finding a more powerful and/or specialised application to do the job. Many programs come with sets of filters and tweaks that can offer you a wider variety of sound wizardry (useful in generating robot voices, impacts and other common effects). However, do not to underestimate the power of a well-recorded sound and a few choice effects – after giving it a shot, you'll wonder how you ever settled for your database of 1001 Free Sounds for your day-to-day gamecrafting.



QUAD TREES

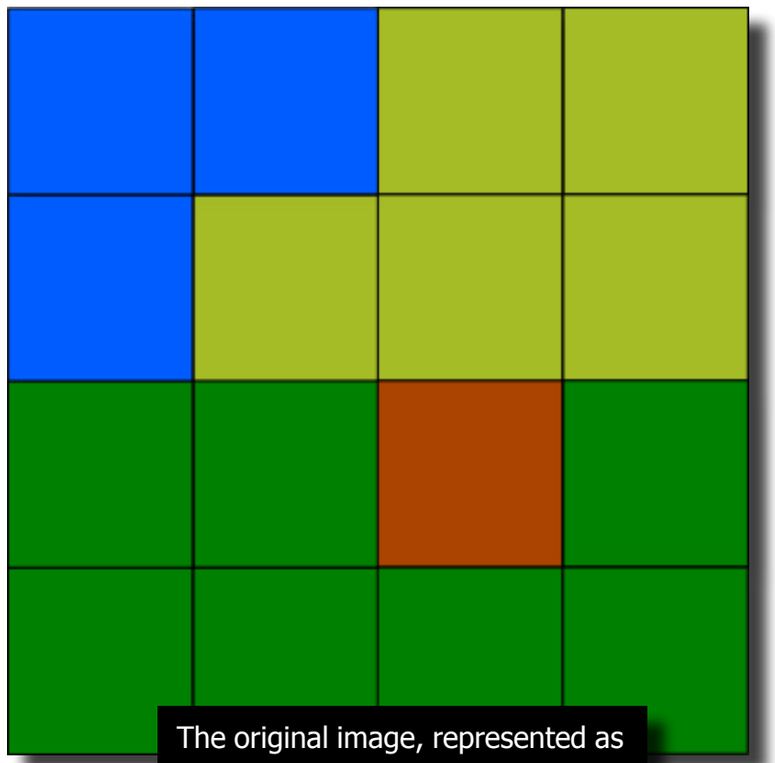
Herman Tulleken.

Quad trees are 2D data structures, useful for efficient representation of 2D data (such as images), and look-up in a 2D space (where are those monsters?). In this tutorial, we focus on the implementation of quad trees that represent 2D data efficiently; that is, where quadtrees can be used to compress data. Although quadtrees can be used with a variety of data types, we will focus on colour data (images) in this tutorial. In part 2 we will look at more general applications, such as occupancy maps and force fields. We also look at the properties of data that determines whether it is suitable to represent as a quadtree.

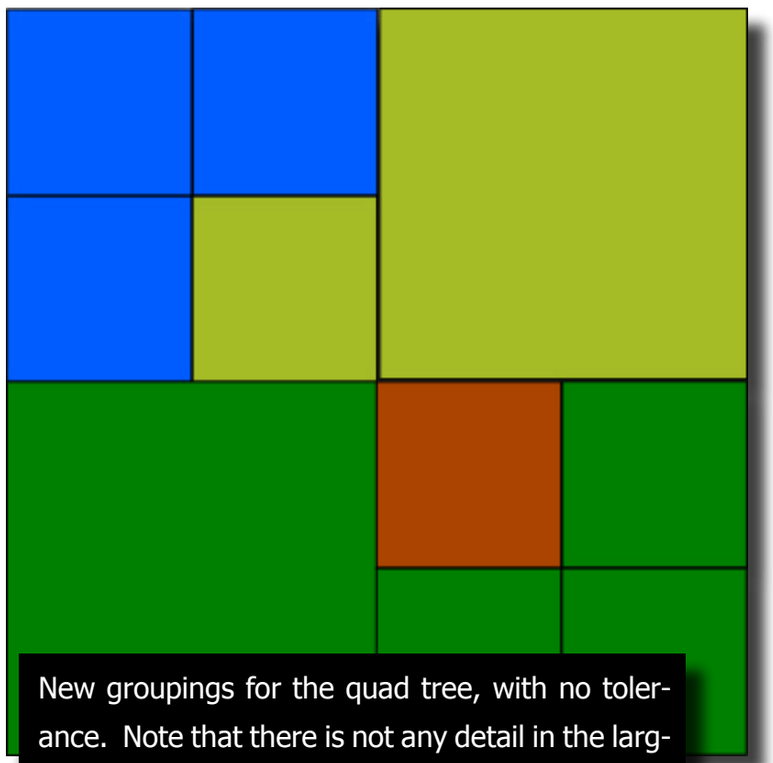
Quadtrees take advantage of the fact that cells in a grid are often the same as adjacent cells – for example, a red pixel is very likely surrounded by other red pixels. Thus, we do not need a data point for every pixel, as we do when we use a grid – we can use a single point for an entire section of the grid.

Implementation

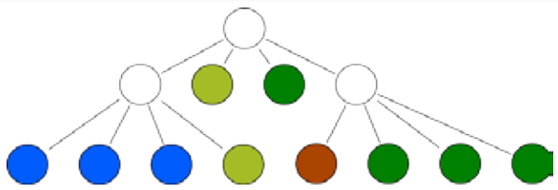
Every node on a quad tree has exactly 4 or 0 children. A quad tree is always constructed from a grid that contains the raw data. The root node represents the entire grid. If the grid does not have enough detail, no children are necessary, and the entire grid is represented by one data item contained in the root. If, however, the data is interesting enough, every quadrant of the grid is represented by a child node. These nodes can be further divided based on the squares of the original image they are to represent, and so on, until every piece of the image is represented by a single node.



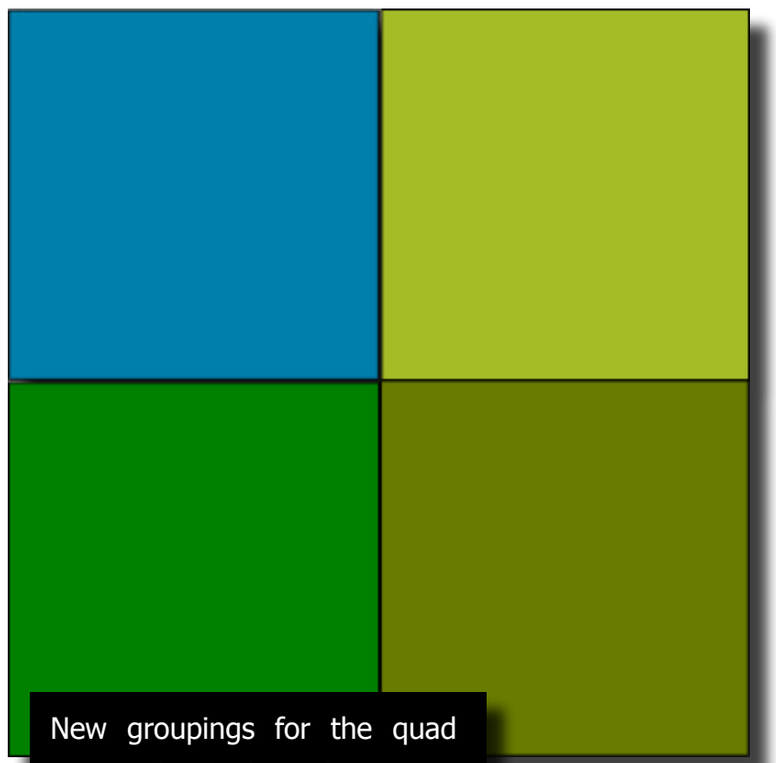
The original image, represented as a grid.



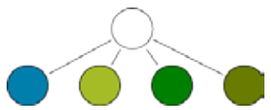
New groupings for the quad tree, with no tolerance. Note that there is not any detail in the larger blocks, so we do not need to subdivide them.



The tree representation.



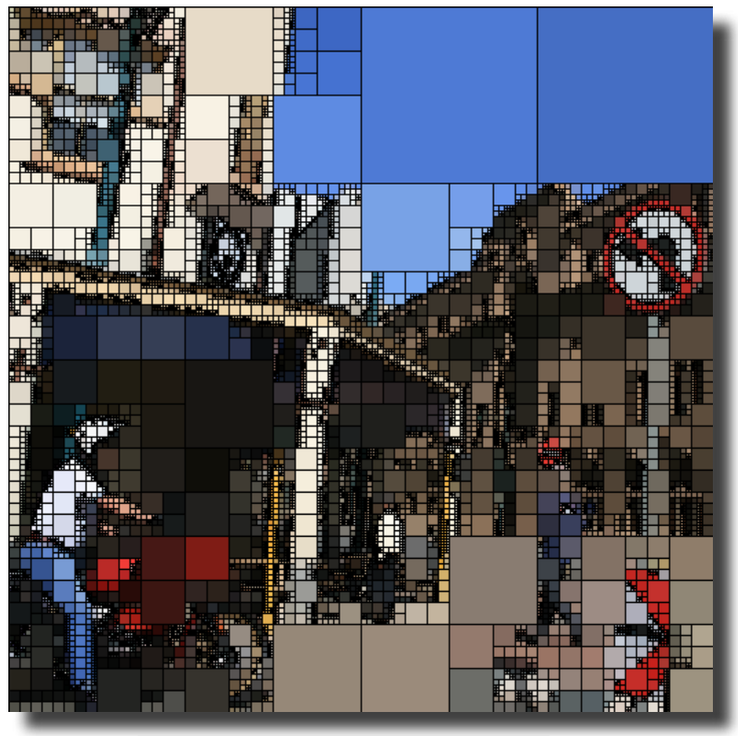
New groupings for the quad tree, with a higher tolerance.



The tree representation.

Original

Quad Tree

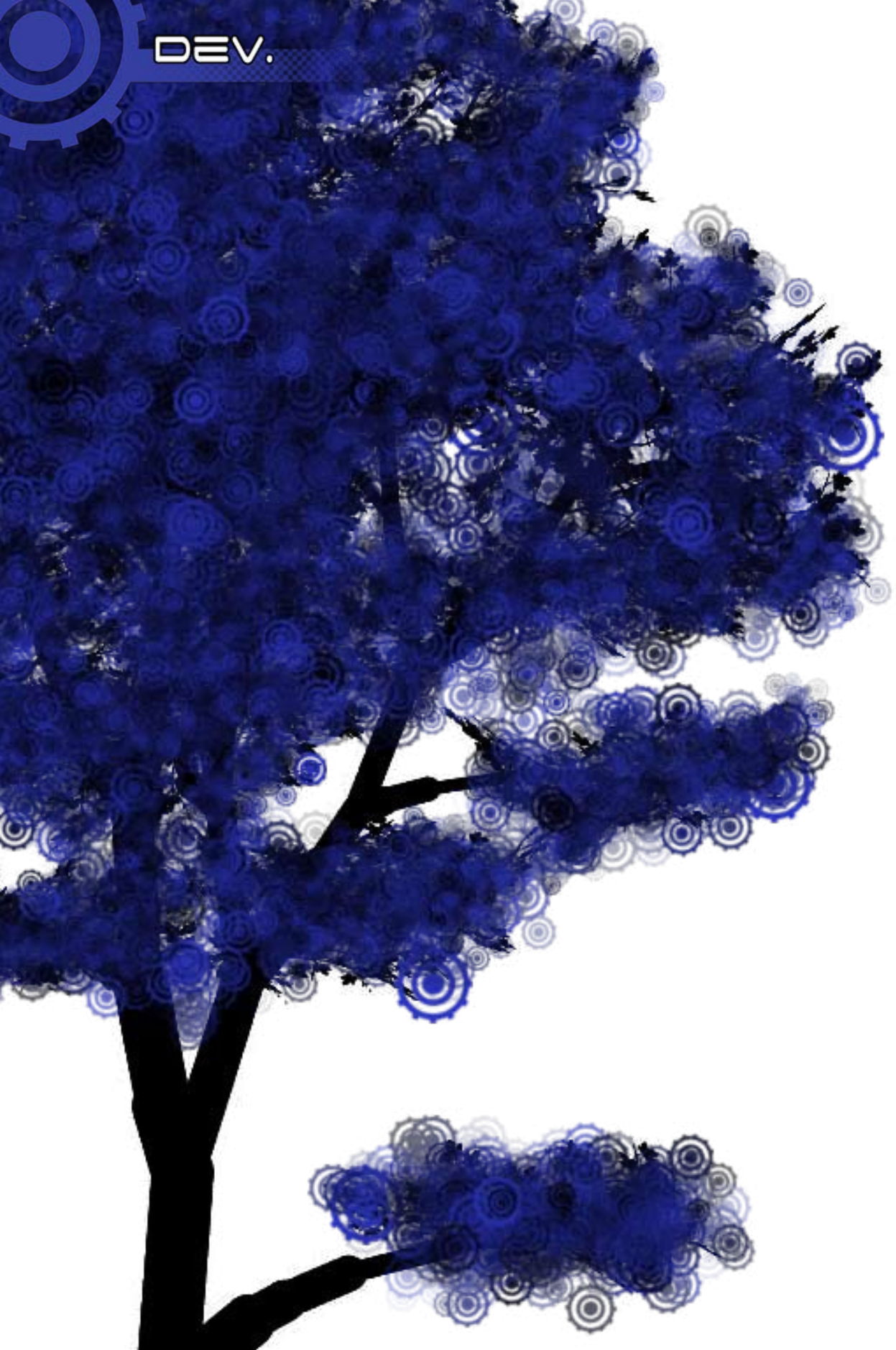


Quad Tree with Rectangles

To implement a quad tree, you need to do five things:

- **define a QuadTree class;**
- **define a Node class;**
- **implement a detail measuring function;**
- **implement a construction algorithm;**
- **implement an access algorithm.**

These are explained below.



Defining the Quad Tree Class

This class is very simple: it stores the root node and the algorithms, which are explained in the sections below. Here is how that would look in Java:

```
class Quadtree
{
  Node root;

  QuadTree(Grid grid){…}

  Color get(int x, int y){…}
}
```

Defining the Node class

The node class is where most of the work is done. It should store its four children (possibly all Null), and be able to handle construction and access functions.

```
class Node
{
  Node children[];
  int x, y;
  int width, height;

  Node(Grid grid, int x, int y, int width, int
height){…}

  Color get(int x, int y){…}
}
```

Detail Measure Algorithm

This algorithm calculates the amount of detail on a rectangle of the grid. How that detail is measured, depends on the application. For images, the average Manhattan distance between colours and the average colour is a crude measure that often works well. The Manhattan distance between two colours is defined as:

$$d = |r1 - r2| + |g1 - g2| + |b1 - b2|,$$

Where 'r1' is the red component of colour 1, and so on. Note that the entire grid is passed to the algorithm, with extra parameters to indicate the boundaries of the piece we actually want to measure. We also define a help function to calculate the average of a rectangle in a grid.

```
// Calculates the average color of a rectangular region
of a grid
Color average(Grid grid, int x, int y, int width, int
height)
{
  int redSum = 0;
  int greenSum = 0;
  int blueSum = 0;

  //Adds the color values for each channel.
  for(int i = 0; i < x + width; i++)
    for(int j = 0; j < y + height; j++)
    {
      Color color = grid.get(i, j);
      redSum += color.getRed();
      greenSum += color.getGreen();
      blueSum += color.getBlue();
    }

  //number of pixels evaluated
  int area = width * height;

  // Returns the color that represent the average.
  return Color(redSum / area, greenSum / area, blueSum
/ area);
}

// Measures the amount of detail of a rectangular re-
gion of a grid
Color measureDetail(Grid grid, int x int y, int width,
int height)
{
  Color averageColor = average(grid, x, y, width,
height);

  int red = averageColor.getRed();
  int green = averageColor.getGreen();
  int blue = averageColor.getBlue();
  int colorSum = 0;

  // Calculates the distance between every pixel in the
region
  // and the average color. The Manhattan distance is
used, and
  // all the distances are added.
  for(int i = 0; i < x + width; i++)
    for(int j = 0; j < y + height; j++)
    {
      Color cellColor = grid.get(i, j);
      colorSum += abs (red - cellColor.getRed());
      colorSum += abs(green - cellColor.getGreen());
      colorSum += abs (blue - cellColor.getBlue());
    }

  // Calculates the average distance, and returns the
result.
  // We divide by three, because we are averaging over
3 channels.
  return colorSum / (3 * area);
}
```

Construction Algorithm

The construction algorithm can be put in the constructor of the Node class. The constructor of the actual QuadTree class simply constructs the root node. The algorithm is simple: the detail is measured over the part of the grid that the node is meant to represent. If it is higher than some threshold, the node constructs four children nodes, each representing a quadrant of the original rectangle. If the detail in the rectangle is lower than the threshold, the average colour is calculated for the rectangle, and stored in the node. The threshold value passed to the function is often determined empirically – that is, you change it until you get what you want. Obviously, the smaller it is, the more accurately the quadtree will represent the original data, and the more memory and processing time will be used.

```
// Constructs a new Quadtree node from a grid, and pa-
rameters
// that indicate the region this node is to represent,
as well as
// the threshold to use to decide wether to split this
node further.
Node(Grid grid, int x, int y, int width, int height,
threshold)
{
  this.x = x;
  this.y = x;
  this.width = width;
  this.height = height;

  if (measureDetail(grid, x, y, width, height) <
threshold)
  {//too little detail
    color = average(grid, x, y, width, height);
  }
  else
  {//enough detail to split
```

```
    children = new Node[4];

    //upper left quadrant
    children[0] = new Node(data, x, y, width/2,
height/2);

    //upper right quadrant
    children[1] = new Node(data, x + width/2, y,
      width - width/2, height/2);

    //lower left quadrant
    children[2] = new Node(data, x, y + height/2,
      width/2, height - height/2);

    //lower right corner
    children[3] = new Node(data, x + width/2, y +
height / 2,
      width - width/2, height - height/2);
  }
}
```

Access Algorithm

The access works as follows: if the node from which the method is called is a leaf (a node without any children), that node's colour is returned. Otherwise, the call is delegated to the child node of the correct quadrant. The method is shown below:

```
// Returns whether this node has any children.
boolean isLeaf()
{
  return children == null;
}

// Gets the colour at the given pixel location.
Color get(int i, int j)
{
  if isLeaf()
  {
    return color;
  }
  else
  { // Decides in which quadrant the pixel lies,
    // and delegates the method to the appropriate
node.
    if(i < x + width/ 2)
    {
      if(j < y + height / 2)
      {
        return ((Node) children[0]).get(i, j);
      }
      else
      {
        return ((Node) children[2]).get(i, j);
      }
    }
    else
    {
      if(j < y + height / 2)
      {
        return ((Node) children[1]).get(i, j);
      }
      else
      {
        return ((Node) children[3]).get(i, j);
      }
    }
  }
}
```

Real-world Implementation

The typical real-world implementation will differ in several respects from the simple implementation described above:

- Integers will be used to represent colours in the raw data and Nodes, rather than Colour objects;

- Intermediate values of colours will be stored as floats, where component values are scaled to lie between 0 and 1;

- Whatever detail function is used, its output will be scaled to lie between 0 and 1. (This makes it easier to determine a correct threshold);

- Adding several hundred (or even thousand) red, blue and green values will cause overflow problems. The summands are often scaled before they are added (for example, the division by the area can be done before they are added to the sums);

Calculating the average is expensive; where it is used as part of the detail measuring algorithm, it will be calculated separately, and passed to this function, so that the value can be reused.

Debugging Tips

- Implement a way to visualise the quad tree, even if the quad tree does not represent an image. In addition to the normal visualisation, also implement visualisations that:
  - render every square in a different colour (see example algorithms **http://arxiv.org/abs/cs.CG/9907030**);
  - render outlines of blocks over the normal visualisation.
- Implement a visualisation of the error of your quad tree representation against the original;
- For benchmarking, implement node count methods for counting:
  - all nodes;
  - all leaf nodes.

  This is useful to make sure that a quad tree is indeed a more efficient representation of you data (for example, white noise will be better represented by a grid).

Resources

http://www.ics.uci.edu/~eppstein/gina/quadtree.html A list of quadtree resources from Geometry in Action.

Downloads

You can download examples of implementations in Java and Python on code-spot:
http://www.code-spot.co.za/2008/10/06/quadtrees

Object Pascal has a lot to offer...

www.PascalGameDevelopment.com

BLENDER

RIGID BODY PHYSICS

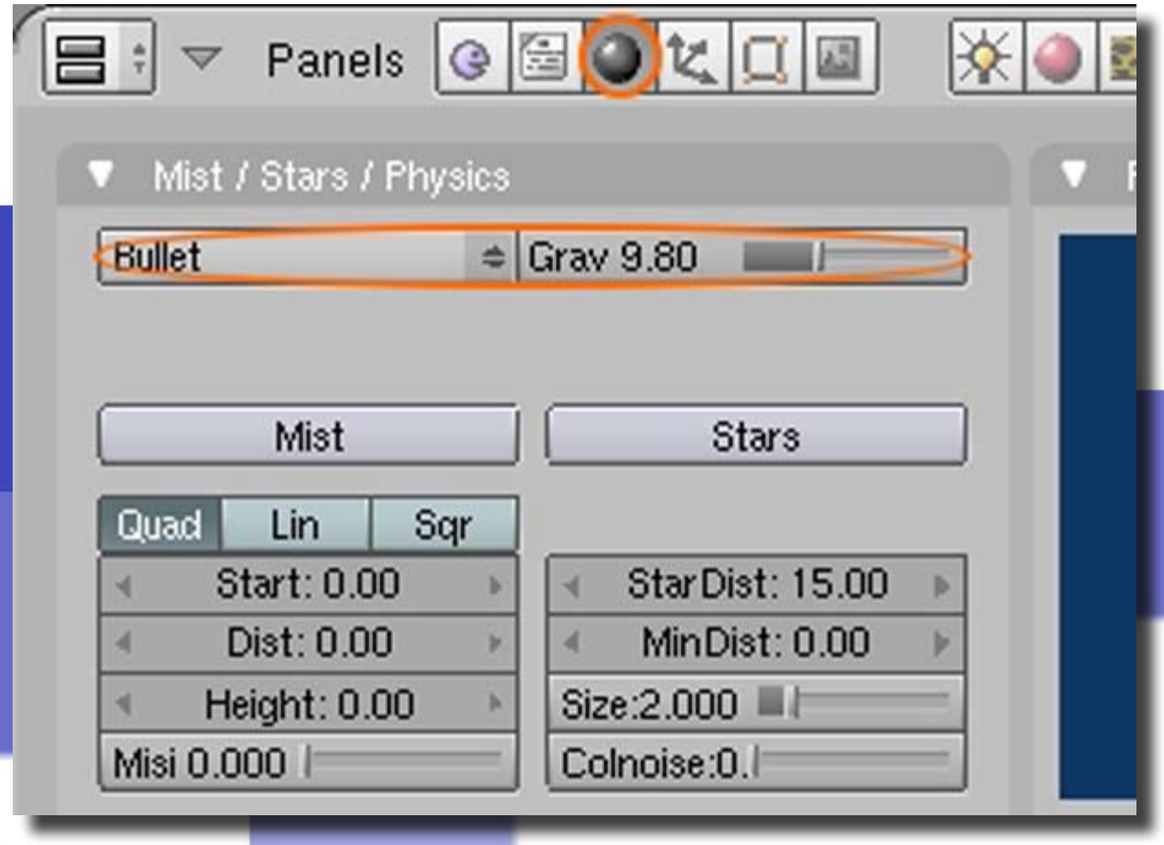
Claudio "Chippit" de Sa

Rigid body dynamics is a term used to describe the motion and behavior of solid physical objects; simulating such things as friction, gravity and collision responses, in the six degrees of freedom that describe the state of most physical objects. A movement along any of the three dimensional axes is referred to as 'translation', and is described by the nautical terms of heaving, swaying and surging. 'Rotation' around the axes is described by roll, pitch and yaw. Rigid body physics ignores the potential deformations of objects during collision and motion, and focuses only on linear momentum (movement and force in a certain direction) and angular momentum and torque (governing rotation and rotational force). This is commonly used in the current generation of video game engines and is also simulated by Blender's Game Engine.

Blender Game Engine

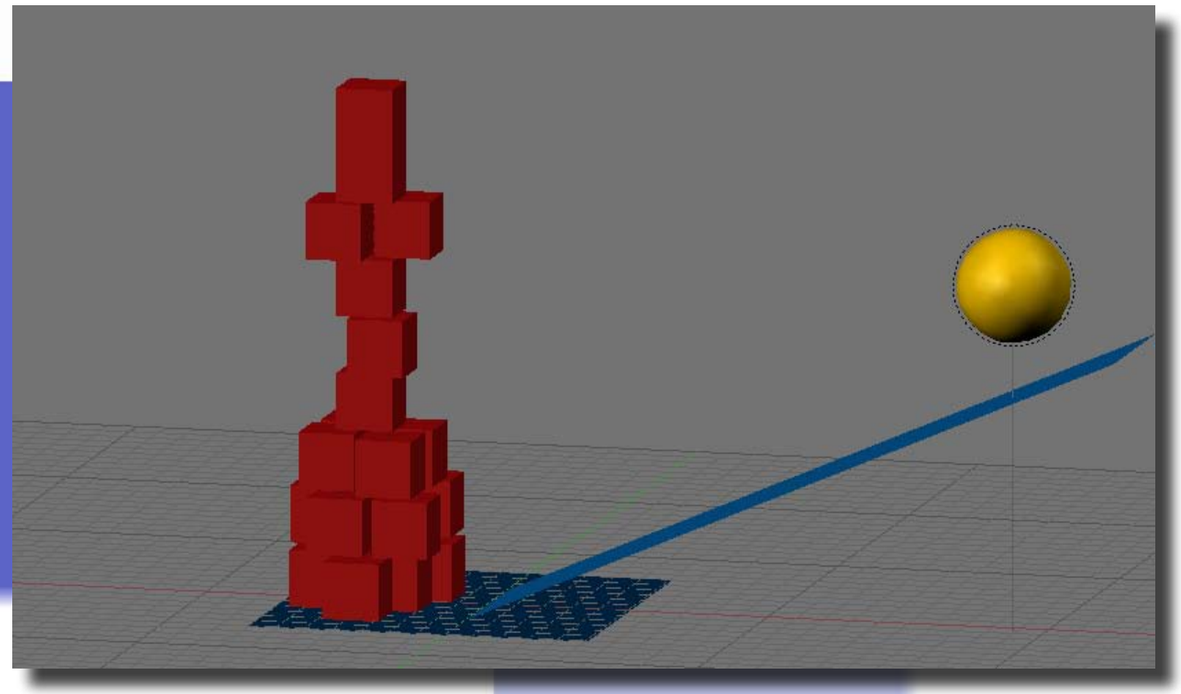
An underused feature of Blender, the ability to create games in its engine, has been steadily increasing in flexibility and power, as is true for most other aspects of Blender's functionality. While the more advanced features of the BGE are reserved for future issues, and out of the scope of this tutorial, we'll be using some of its basic functionality to create rigid body physical simulations. The 'record to IPO' feature and the physics engine itself.

To get started, you need to ensure that Blender is set to simulate using the Bullet physics engine – the most advanced engine currently implemented. This should be the default setting for all Blender versions newer than 2.42. If you don't have it, it's probably time to upgrade. The relevant options are grouped under the Shader context menu -> World buttons. In the 'Mist/Stars/Physics' tab you'll find the choice of physics engine, as well as a global gravity value.



Scene set up

Physics engines need objects to operate on. Whilst the bullet engine can operate on complex objects; boxes, spheres and planes are the fastest and simplest to use. I've set up a quick example scene using a haphazardly stacked pile of boxes, a ball, a plane to represent the floor, and another plane for the ball to roll down.

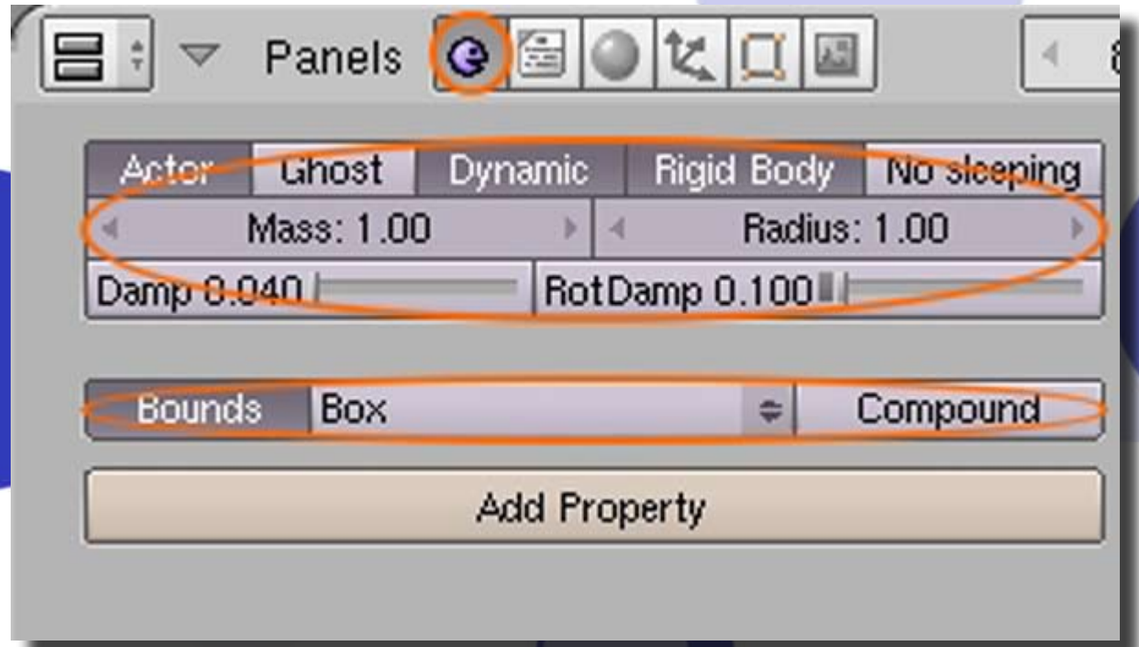


The key to a successful rigid body simulation is - as with the other simulations previously covered - a good setup. Every object needs to have certain properties defined, the most important being their collision type and their mass. All the relevant options are visible in the Logic context. Note that before you place a large quantity of objects in your scene, you will need to define physical properties for each item individually. If you plan to have a lot of items in the scene with identical or similar properties, it is often best to create a single one, set up its collision properties, then make duplicates of it.

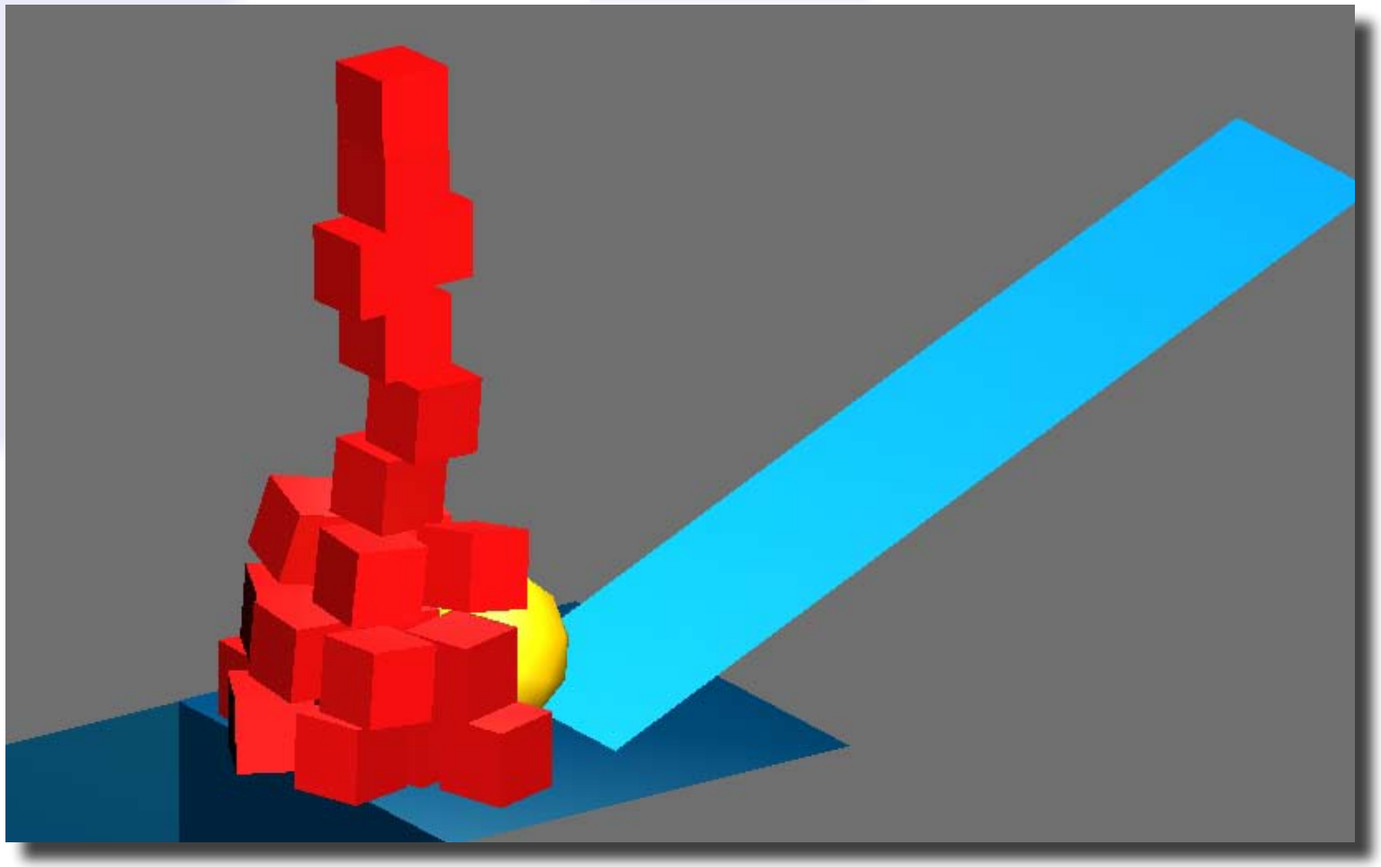
Object Properties

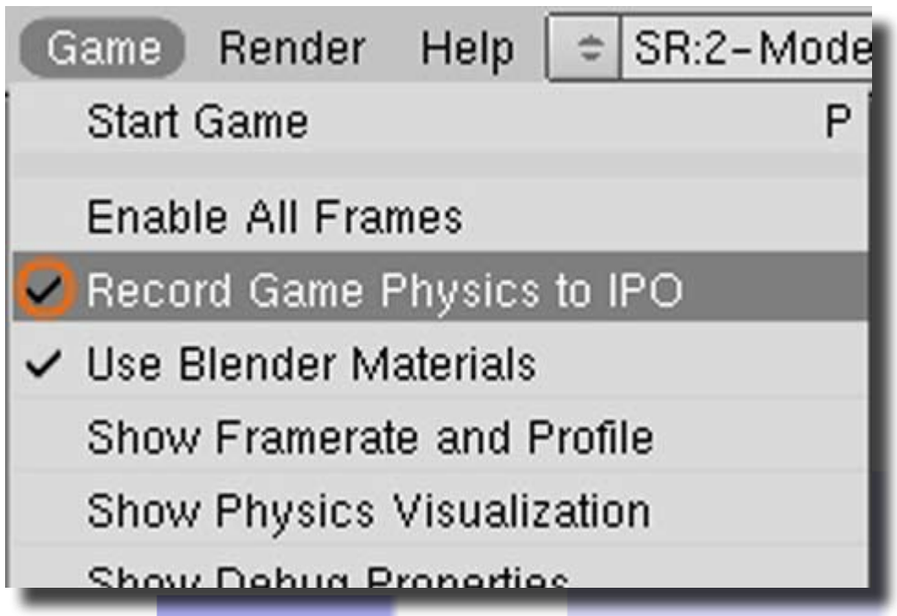
Rigid body objects are actually comparatively simple to set up and use. A quick overview of each relevant option follows:

Actor: This object is active and will be evaluated by the game engine.
Ghost: This object will not collide.
Dynamic: Object obeys the laws of physics.
Rigid Body: For accurate rigid body calculations, including angular velocity and momentum, necessary for rolling motions.
No sleeping: Object will never enter a 'rest' state where simulation stops running on it.
Mass: Total mass of the object.
Radius: The radius of the sphere that represents this object's bounds. Only necessary for a spherical collision setting.
Damp / RotDamp: Damping values for all movement and rotation.
Bounds: Specify object collision bounds for physics. If you do not specify one of these, the collision engine will use the actual mesh. Choosing one of these usually speeds up your computation time, however, so it is recommended where possible. Within the bound menu are three settings.
Static triangle mesh: For complex shapes that do not move, commonly used for terrain or static obstacles.
Convex hull polytope: Will use the smallest convex hull as the collision mesh for this object.
Sphere/Cone/Cylinder/Box: Uses the specified shape as a collision mesh for the object.
Compound: The object is made of multiple compound shapes. Used for more complex simulations where objects are tied together with child/parent relationships, to make more complex shapes.



Because our planes aren't going to move, we don't need to do anything to them; they'll collide with objects automatically. The other's need a few changes, however. The boxes should have 'Actor', 'Dynamic', and 'Rigid Body' enabled, together with Box bounds. The ball is the same, but with spherical bounds. Be sure to set the radius setting to match the size of the sphere. You'll notice that the radius is represented graphically in the Blender 3D view, though it's often easier to see in wireframe mode.



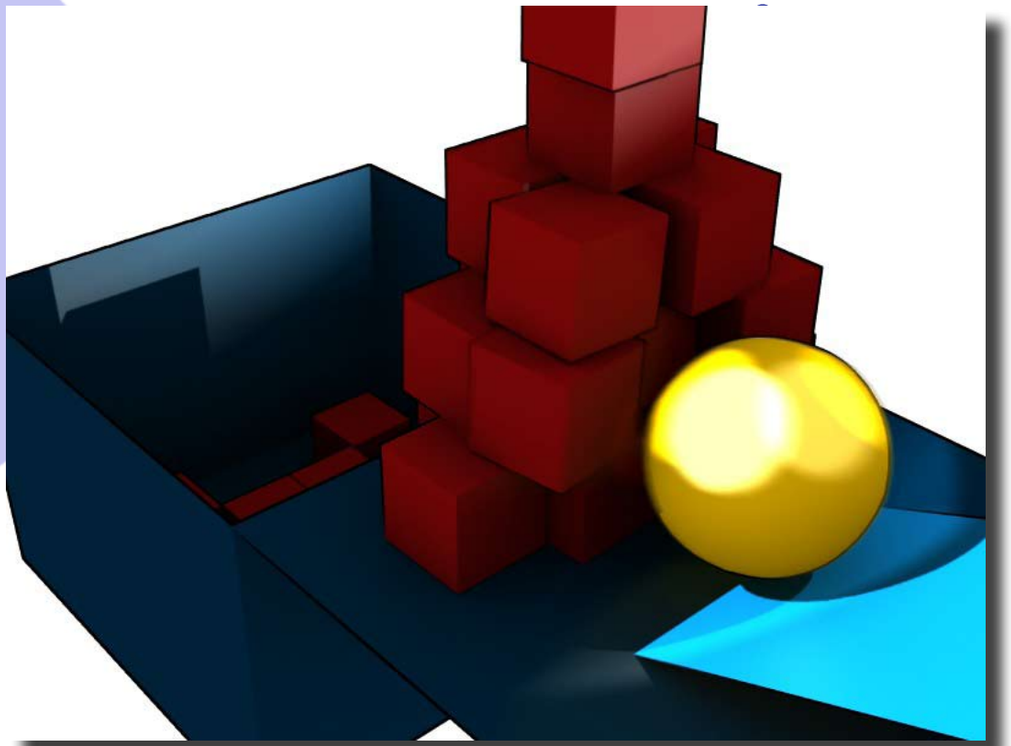


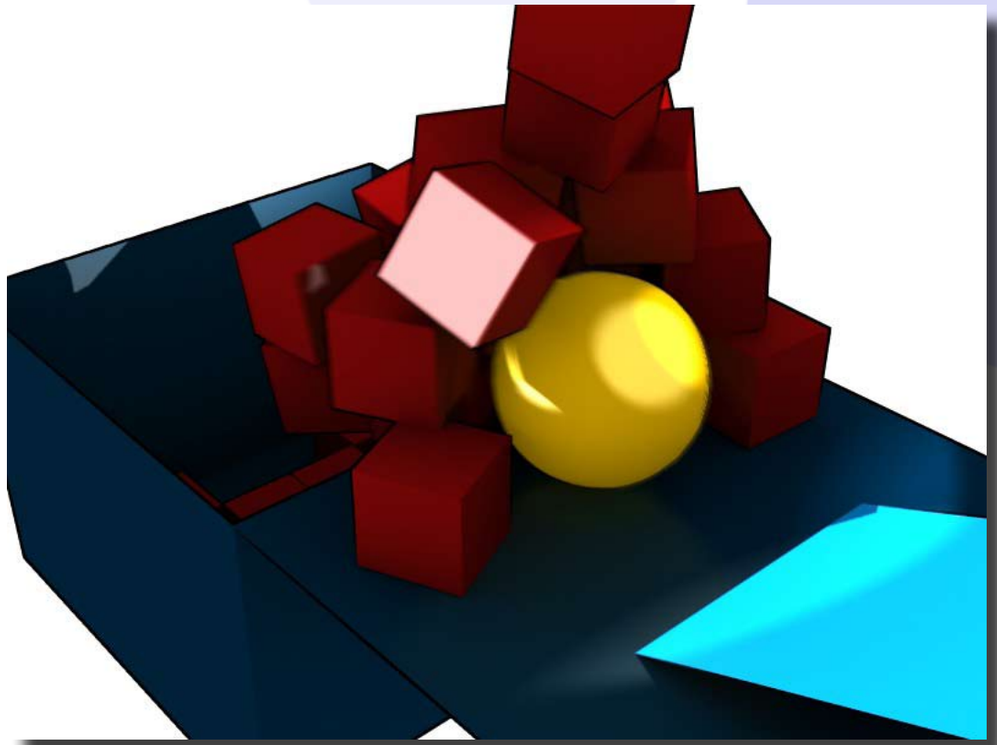
Tying everything up

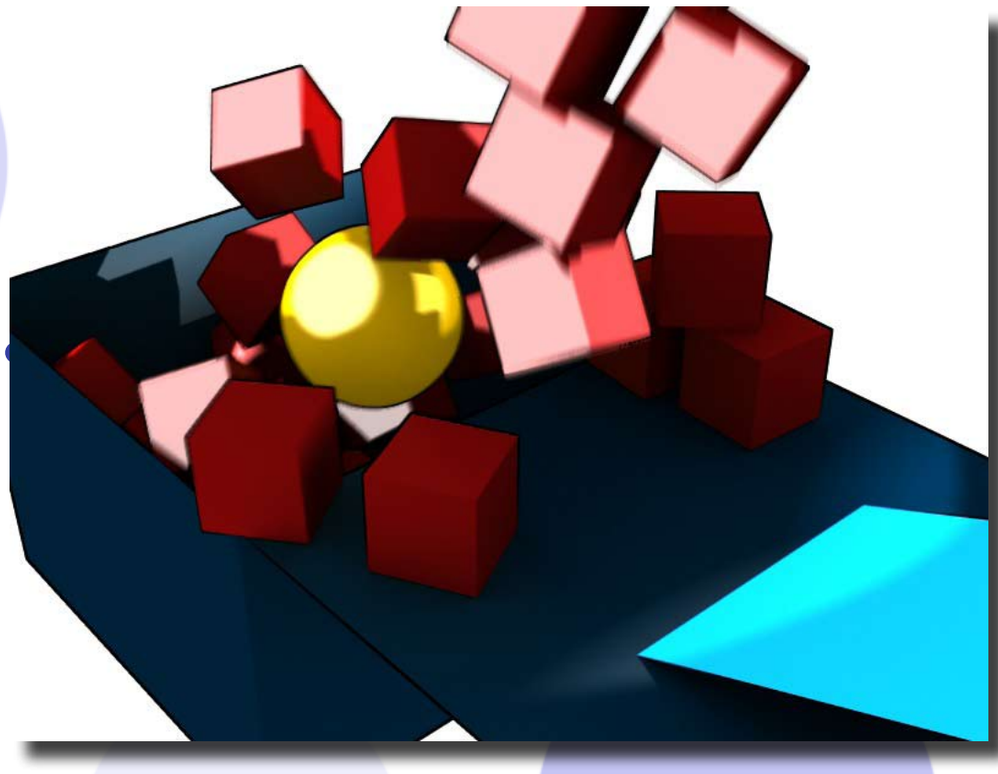
Once you're confident that all your objects are set up, simply press P or select Start Game from the Game menu in the main Blender menu bar to start the simulation. If all goes well, you should see the ball roll down the slope and strike the boxes placed at the bottom, all in real-time. Press Escape to end the game simulation when you're done.

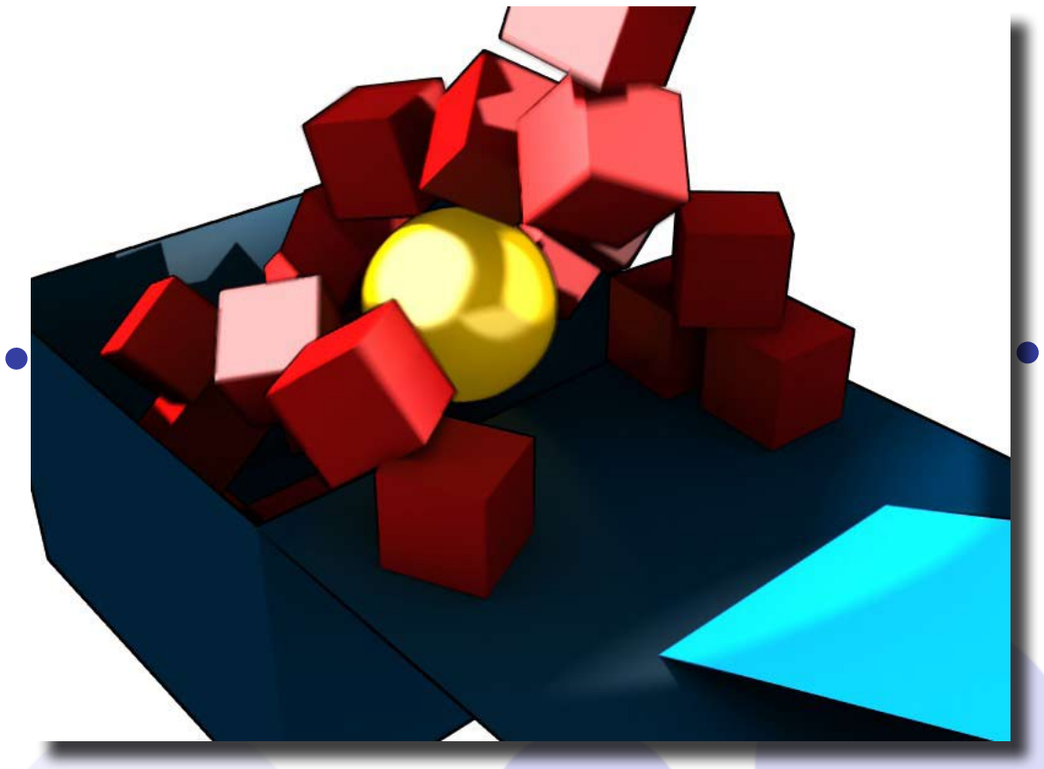
Now, it's all well and good that everything is moving as it should, but it still won't do any good if we want to render this as a proper scene. The BGE provides a Record Game Physics to IPO feature that will take all movement in a game and record it to all the involved object's IPO curves, for tweaking and proper rendering. Enable the option in the Game menu then run the simulation again.

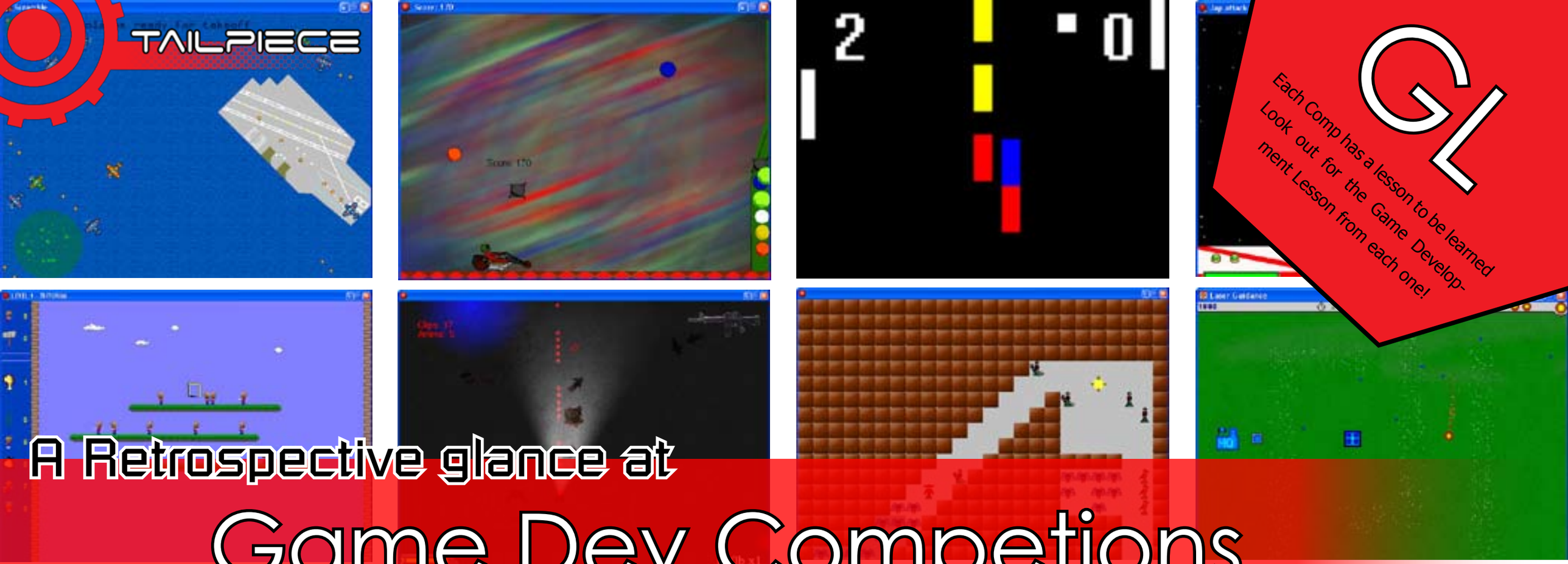
Once it completes, you'll see that you can step through the baked animation just like any normal keyframed animation. You can also edit the IPO curve, just as you would do otherwise. Note that the IPO curve also defines where an object will be at the beginning of the game simulation. If you wish to move an object after you've already recorded a previous animation to IPO, you'll either need to clear the IPO curve or add a keyframe with the new position at the starting frame. That's all for this month. Next time we'll get to actually using the BGE for what it was intended. Interaction and making games! Have fun till then.











A Retrospective glance at Game.Dev Competions

Part 1

Rodain "Nandrew" Joubert

Compared to most other game development competitions, Game.Dev's fondly-named "Comps" have always stood out on one particular front: each new incarnation has always set out to challenge, direct and develop entrants within the field of game development. Instead of the oh-so-typical "create a game about kitties and/or mudkips" mentality that many mainstream events focus on, the Game.Dev competitions have always sought to home in on an aspect of game development that people don't always consider, and try to train new developers in the techniques that it describes. Although some may frown upon this method and drop out as a result, those who engage with the competitions often emerge from the experience as more mature and insightful developers.

Since January 2005, Game.Dev has worked to inspire and lead game developers with these competitions, and some truly intriguing titles have come about as a result. What follows is an overview of early Game.Dev Comp history, along with the lessons that people have learned along the way. Read on and be inspired.

Comp1 Make a game. Any game. Go for it.

GL

It's possible to make videogames, whoever you are, whatever your experience level.

The Game.Dev competitions had very humble beginnings, as most things tend to. Comp 1 started off as a simple idea posted on the NAG forums in January 2005, before Game.Dev itself even existed. The concept was basic and the criteria were broad: make a game, any game, and post it on the forums for judgement. The competition eventually produced five games, most of them using the recommended development tool, Game Maker. These entries were crude compared to later offerings, but they proved one thing: there was an interest in game development amongst gamers (who would have thought?). Even though some people scoffed at the idea of such a 'childish' tool being used to craft games, anyone who bothered to download this free application and take the time to fiddle about with it was generally able to produce results by the time the competition came to an end.

Comp2 Circles vs Squares

After a month of downtime (and the creation of its own forum), Game.Dev decided to host its second competition, themed around circles versus squares. The group was still rather young and wide-eyed at this point, but the competition provided several entries from members who would later become influential components of the Game.Dev group. In any game, it's important to look at the fun factor first – you can get to the rest of it later. Comp 2 produced some hearty entries from people who used circles and squares to their best effect to create a fun and engaging experience, limited to a simple graphics set and forced to figure out how they can make their game stand out from a field of similar-looking entries.

GL

Great games can be made with the simplest of graphics.

Comp3 Remakes

GL

It pays to study the classics.

Game.Dev's Comp 3 asked gamers to do a remake of famous old-school games (aside from a few horribly cliché ones such as Pong and Tetris). The results were interesting, to say the least. Some opted to take the classics and improve upon old dynamics with the availability of better development tools and greater processing power. Others took even more creative routes and merged several classics to create an entirely new game using rules from each. This competition was possibly the first to display the game development maturity of entrants: the top games homed in on the most fun aspects of these bygone offerings, proving that they understood what made great games great; adding improvements in the correct places to make these titles even better. After all, everybody knows that PacMan is famous; not everybody truly understands why. To excel in game development, Game.Dev wanted entrants to analyse the games they play more critically, and adopt that special 'game developer' mindset that's critical for anybody who wants to do gamecrafting for a living.

Comp4 Simple Rules, Complex Game

Too often, game developers try to make a good game by adding more bells and whistles. Not enough variety in your project? No problem, just add more enemies and abilities... Right? Wrong. A flawed game doesn't become better simply because you add more features – it's the core dynamic, that little kernel of your game which defines it and makes it special. This was an exercise to create a few rules that the game developer could twist and manipulate to generate a massive variety of gaming scenarios, and exercised the creativity and flexibility of developers. This particular competition produced one of the finest games of Game.Dev's early era – an offering titled Roach Toaster which stood head and shoulders above the rest of the entries up until that point and raised the bar for all competitions that followed. It didn't have mind-blowing graphics. It didn't have a load of flashy scripted events. It didn't even have sound effects. It just had a basic roach generation algorithm and a few well-balanced roach busting tools that were meticulously considered; providing a player with a simple experience that felt like an epic.

Comp5 Action!

Action games are difficult. They tend to be real-time and a lot of control leaves the developer: you can't force the player to take a turn, deal a specific amount of damage and tailor the enemy response to provide a balanced counter-attack. Every split-second matters; which means that the developer needed a lot of help to make sure that the game felt 'just right' no matter who played it. By the time Comp 5 came about, a flood of new developers had entered the forum and it fell upon the established crowd to help them get into the swing of things. This revealed a trait about the community which has successfully lasted to this very day: an openness and friendliness which is crucial for allowing good game development. Whether an entrant was a development veteran or a complete newbie who had just learned the concept of 'player.x + 1', feedback from the community was inevitably constructive and helped make early, clumsy offerings into golden games by the end of the competition month. Those who posted early drafts excelled in this competition, because instead of relying on a single developer to playtest and hunt for bugs in their title, these entries had the feedback and collective expertise of at least a dozen enthusiasts to back them up.

Comp6 Polishing an Old Game

Game.Dev's Comp 6 decided to go in a slightly different direction and forced entrants to look at previous work for inspiration. Most new game developers are quick to generate a fun or quirky title, but tend to lose steam after they've finished a "full go" of the game or realised that there were too many extra resources to generate easily. Comp 6 was very much a discipline competition – people are often reluctant to revisit their old creations, favouring a hop to new titles, rather than lingering with the old. But polish is important for good game creation, and most of the successful games out there weren't simply done with one take – they repeatedly changed as development progressed, and no matter how heartbreaking it may be to throw away a particular piece of code or artwork and start again, it's necessary to allow growth in your game where it's needed. Successful competitors were also generally able to modify their games quite easily – they'd left enough room in the design for change, rather than creating a static game with no opportunity for expansion. Remember to plan ahead when designing your game – you never know how it may change at the end, and it's much better to modify a small amount of code rather than being forced to restart the whole mess.

TAILPIECE

Comp7 Style.

Have you ever played a game with that certain X factor that made it really special? That feeling or vibe which turns an average Joe game into something a little more involving? Style is an elusive aspect of game development and competitors found it difficult to define. In all respects, this was the most advanced Game.Dev Comp to date – not only were people required to craft a game, but they had to grasp an abstract concept and try make it show in their final work. To ease the process, this competition was once again oriented around remakes, to ensure that game developers had a springboard to launch from instead of floating about in a haze. Many of today's remakes often have some sort of revamp or stylish factor to make them more appealing to players, whether it's a particular colour theme, the type of sound effects employed or even just a funky change of art direction. Once consolidated in a remake, these sort of ideas can be carried over and used in original games, to put your own unique stamp on your work.

Comp8 Consume!

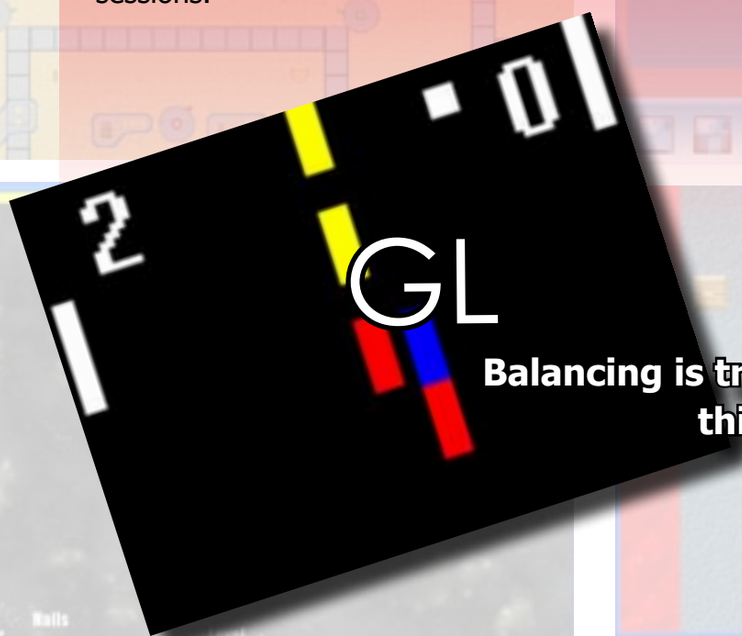
By this time, Game.Dev had evolved even further and was beginning to look at other development groups for inspiration and ideas. The idea for this Comp came from Experimental Gameplay, a site known for its interesting prototypes, which was holding a similar competition at the time. There were two major points in this competition: firstly, the description was simply 'Consume', affording a great deal of flexibility to entrants keen to pump up their creativity. Secondly, each entrant was required to submit two games instead of one. The result was that developers had to learn the skill of prototyping – rapidly conceptualising and establishing the framework for potential games without getting bogged down in details or long-term development. Prototyping is an incredibly important skill in game development: new developers often try to make "the next big game" and end up getting bogged down with a concept that often isn't all that good. A far better idea is the rapid generation of several minor game concepts, allowing the developer to gather a broader range of experience and browse through an entire collection of ideas to see which one works the best.

Comp9 Google it!

Comp 9 was odd in the way that the lesson it taught was quite dramatically different from the one which was originally thought up. After more than a year of competitions (running one every two months), Game.Dev decided to engage developers a little more and have them looking in rather exotic directions. The premise for this one was therefore quite creative: entrants were each given three words to use, and all of their in-game graphics needed to consist of imagery extracted from Google image searches based on these three words. This was meant to be another competition which focused on the generation of good gameplay, while forgetting about complicated graphics. Unfortunately, for some developers, this task was a little too restrictive – they found themselves developing games that they didn't feel entirely comfortable with, and the results showed in these cases. Discipline is important in game development, but it's also important to remember that game development is an expression of your own creativity and enjoyment, and that the best titles are created when the developers love what they're doing.

Comp10 Management Games

By the time Game.Dev had hit the double digits for its competitions, it had gained enough influence and enough of a following, to attract a sponsorship of a R10 000 (just over $1000) cash pool for Comp 10. This was met with considerable enthusiasm from the community, and to do justice to this cash sponsorship, it was decided that the competition would run for an extra month, focusing on a particularly challenging subject: management games. This genre, more than most, requires developers to carefully think out their game design in advance, considering every addition to their game and how such an addition would affect the rest of the objects already in play. Although it was the players who would ultimately be keeping track of resources and variables, it was the developer who needed to pay meticulous attention to ALL of these values to ensure that the game remained consistently challenging and fun to play. Planning was key, and the winner of the competition (a game titled "Fast Food in Space") exemplified this principle by providing players with a management game that kept developing and offering new challenges as play-time increased, ultimately providing a steadily rising difficulty curve, which managed to keep gamers hooked for multiple playing sessions.



That's it...for now

This concludes Part 1 of the Game.Dev Comp series. Check next month's Dev.Mag for the second half of the series, where we investigate more contemporary competitions, and see where they've taken participants following their first tender steps nearly four years ago. If you're from South Africa and are interested in entering Game.Dev's latest competition, keep an eye on the Website **(www.gamedotdev.co.za)** and scout about on the forums for news of the most recent offering.

GEAR COUNT:

IT'S TIME FOR OUR FAVOURITE GAME! HO-DOWN!

GO CALL YOUR MOM.

OHSNAP!

WWW.DEVMAG.ORG.ZA