Issue 27

November 2008

DEV.MAG

CREATE • DEVELOP • EXPERIENCE

They're here...

INSIDE:

CONTENTS

FARMER
Claudio "Baah" de Sa

SHEPARD
James "Ram" Etherington-Smith

SHEARER
Quinton "Ewe" Bronkhorst

SHEEP
Rodain "Quack Quack" Joubert
Simon "Lamb Chop" de la Rouviere
William "Possibly not a sheep" Cairns
Danny "Rock the Flock" Day
Andre "Mary had him" Odendaal
Luke "Bleating the Cold" Lamothe
Gareth "Ate one for dinner" Wilcock
Sven "Woolyspoon" Bergstrom
Chris "Woolen Mittens" Dudley
Herman Tulleken

HOUSEKEEPER
Robbie "Shear the fun" Fraser

FARMHOUSE
www.devmag.org.za

POSTBOX
devmag@gmail.com

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

COMBINE Orange and Duck.
"There are only two ways that combining these two objects is possible, and neither of them are particularly comfortable for the duck. So let's rather not, then."

So DreamBuildPlay is over. Unfortunately, neither of the Game.Dev offerings placed in among the finalists (we needed more sheep, evidently), but the experience both teams gained from the journey will likely prove more valuable than any prize we could've won from the competition. We've created the first two complete Xbox 360 games in the country, both of which were proudly on display at the **rAge expo** held last month for people to see and play.

This is a major stepping stone for recognition of our fledgling industry in a more global light. Both these games have the potential to be released on the XNA Community Games service (which will likely be live by the time you read this), both for personal profit (yay, making money for the stuff we love doing!) as well as the subsequent acknowledgement.

Other than that, things are pretty much as normal here. I apologise that we're a bit late this month, something for which I am mostly to blame. Other responsibilities meant that most of my Dev.Mag time was otherwise occupied. That is also partly the reason that there is no Blender tutorial this month. Additionally, there will be **no December issue of Dev.Mag**, with that month being our traditional break. Rest assured, we'll be back in mid-January.

Now to this month's content. The highlight of the month is our promised double-feature on the Game.Dev DreamBuildPlay entries, chronicling the journey and lessons as they happened. We've also managed to snag copies of 2DBoy's World of Goo and Introversion's Multiwinia for review. Things are tied up at the end by the conclusion to our Game.Dev Comp roundup on the final 10 competitions.

Oh, and **sheepies**!

~ Claudio, Editor

Flash development tools abound

http://www.mochiads.com/resources/
http://www.kongregate.com/labs

Interest in Flash development grows with every excellent offering that is produced for it. Many sites attempt to harvest potential new developers, and there are two new contestants on the block. Firstly, Mochiland, a development community that showed people ways to make money out of their games, has assembled a large host of valuable flash development resources on their website in an attempt to help new developers get started. Secondly, the popular Kongregate flash portal has created their own basic game development tool under Kongregate Labs. It seeks to increase interest flash game creation by simplifying the process, as well as offering 'new developer' prizes as incentive.

New Lost Garden prototyping challenge

http://lostgarden.com/2008/11/fishing-girl-game-prototyping-challenge.html

Lost Garden has just launched their new prototyping challenge, this time involving a casual flash game. The premise of this competition is quite simple, yet, as always, the execution thereof will be of great importance. The post details the play dynamics of a fishing game, and Danc hints that it may be somewhat related to the Mystery Project he's started with developers in his local area. As always, artwork for the game has been included.

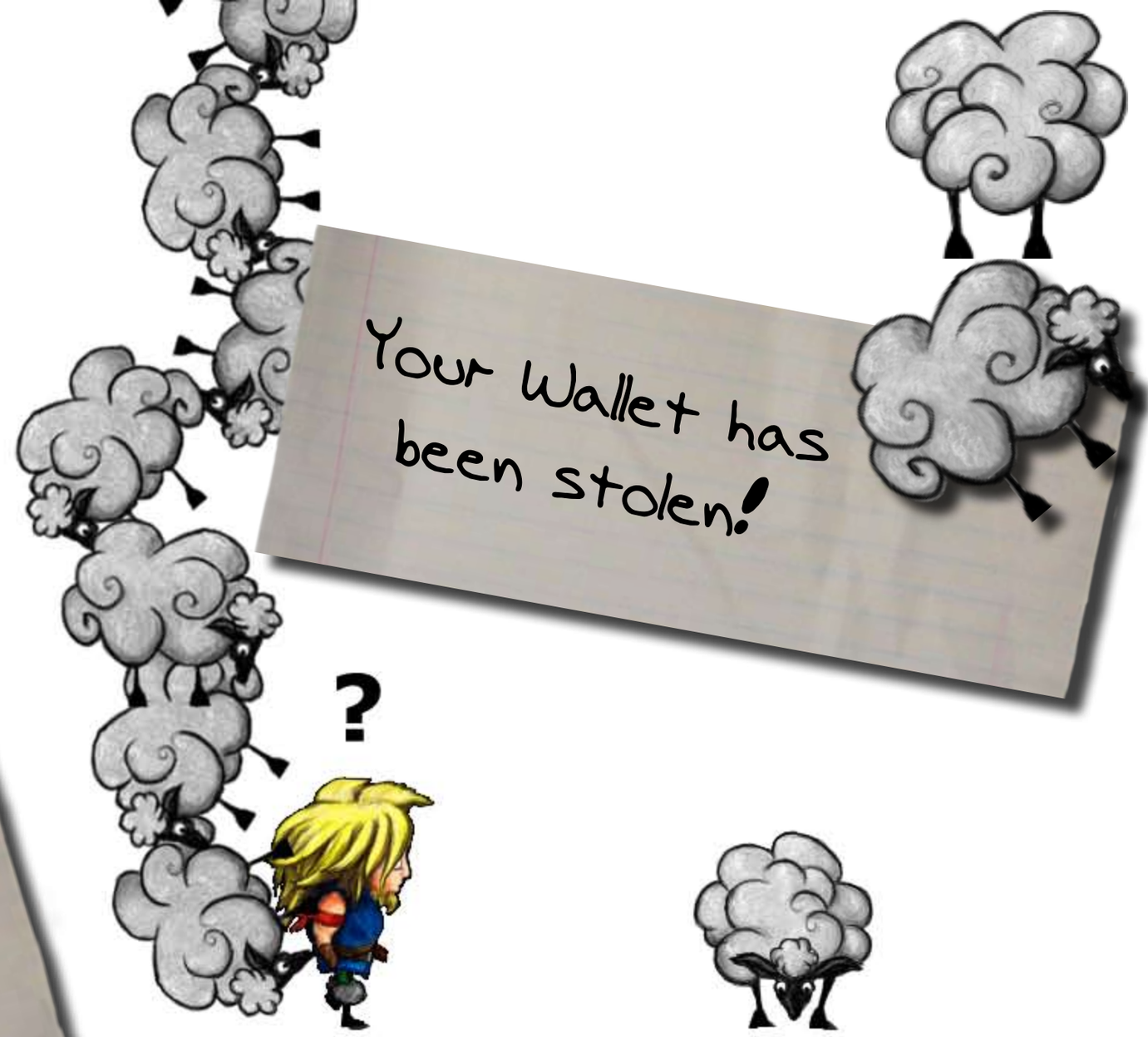
Soundsnap hosts gargantuan sound library for free

http://www.soundsnap.com/

Soundsnap, a free community based sound-effect site with a library of nearly 100 000 sound effects (ranging from weapon sounds to music loops to ambient effects) was recently launched. It is entirely community driven, with all sound effects available created or recorded by its members. The site contains varied and high-quality audio samples which will be invaluable for game developers, and is a site that should reside in every developer's resource bookmark list.

Ultimate Level Quest

Claudio "Chippit" de Sa



We've come a long way, crafted what is probably the most complete game I've ever created in under 3 months, entered it into a huge, global competition, and came away sane and with a product that we're proud of. Are there better ways to spend sleepless months? Probably, but few of those result in such a sense of accomplishment as seeing groups of strangers, gamers and non-gamers alike, playing your game in the largest expo in the country. And laughing and having fun.

So we've achieved what we sought to do, and learnt much in the process of getting here: The value of development tools in larger projects (like our UEditor), the massive benefits of working in a group, quite a few intricacies of puzzle design, and the practice of iterative design. And, of course, the general experience gained from creating an entire Xbox game from scratch.

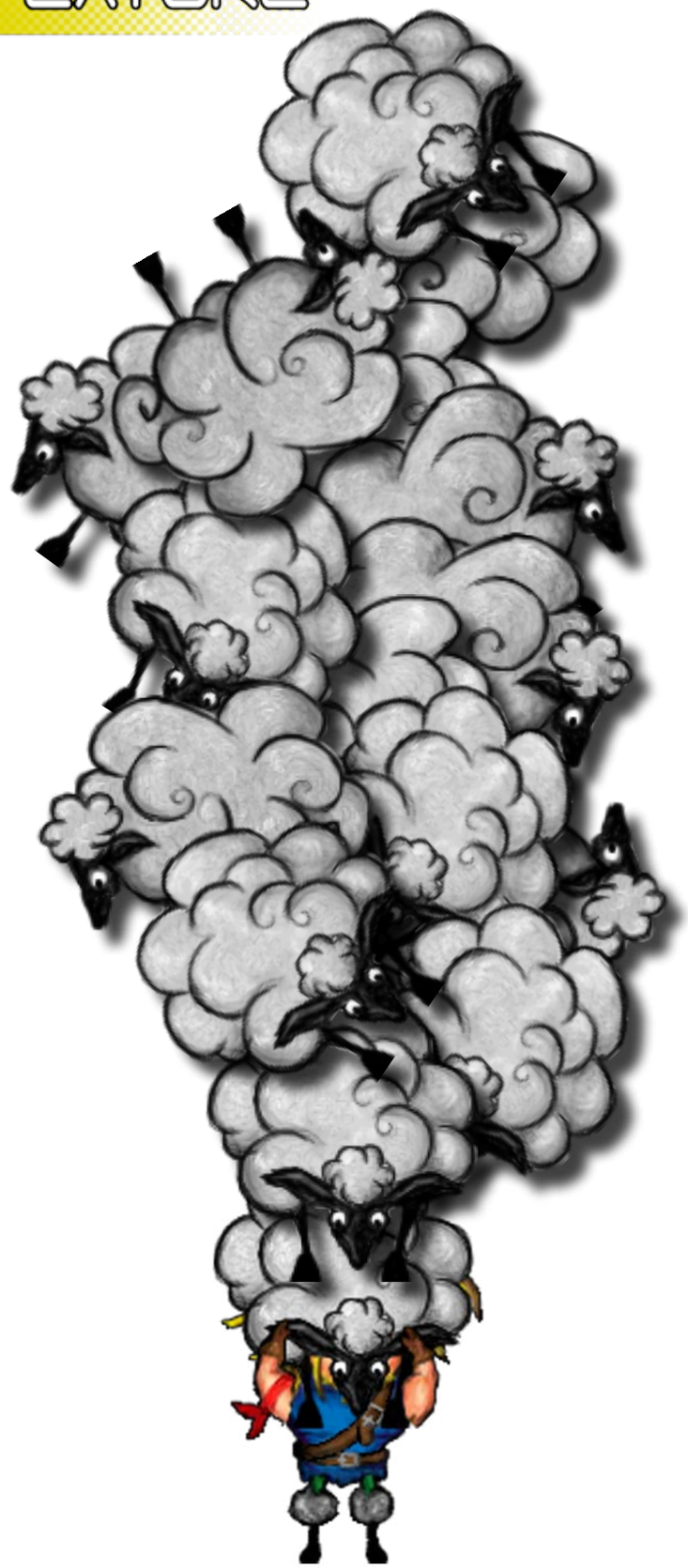
But let's go back to the beginning, soon after we decided to work on UQ for DreamBuildPlay. We had the original game, which we believed was successful enough, and now we had a new challenge of making it work on the Xbox. The largest obvious challenge was the completely different control scheme. We couldn't use a text parser input or the traditional point and click systems that worked so well for the PC adventure games of old, since both of these would be incredibly awkward on a 360 controller.

We eventually settled for a system not unlike that which was used in the later LucasArts adventures Grim Fandango and Monkey Island, where the avatar's position is essentially your cursor and the player can only interact with objects in the avatar's immediate vicinity. In our system, nearby objects were highlighted and placed in a ring around the player for selection by the user. We added an additional twist into the mix by giving the player the ability to highlight all objects on the screen that are interactive, regardless of distance to the player. This essentially removed all artificial 'pixel-hunting' challenge that some adventures used to ramp difficulty and required us to adjust our puzzle design accordingly. We couldn't hide things in obscure places on the screen to make our puzzles harder, so we had to add additional steps and/or complexity to the puzzles in order to achieve the same effect.

As it turned out, however, our puzzles ended up too difficult because of this. With both Azimuth and I being a more veteran adventure gamer crew, the puzzles we did have made the difficulty curve a rather formidable obstacle. One of the largest focus points we have at the moment is to include simpler puzzles at the start of the game so that players aren't immediately stuck without any idea of what to do. There is little more frustrating for a player than to be overcome by a game's difficulty right from the start, and this is something we urgently need to address.

Once we had a control scheme we believed could work, I started on a preliminary version of the game tile engine and what eventually became the UEditor. At the time, it was simply a tile map editor, but I quickly worked on ways to make it easy to use, allowing it to place pretty much anything we needed on the map, including interactive and static world objects. Eventually, it grew into the tool we'd also use to craft and test in-game dialogue as well. The dialogue editor used a flexible and powerful tree-based system, which could also control special game events and access persistent in-game variables and even player inventory items. Essentially, all actual in-game content is created almost exclusively in the editor, which made the entire game extremely modular and easy to extend. Levels, complex merchant dialogue, action responses and more were all crafted from the UEditor and imported directly into the game.

The graphical style of the game underwent some of the most drastic iterations. Initially we settled for an 8-bit pixel-doubled style as something of a homage to old Sierra titles. Difficulties coupled with a few concerns about the general acceptability of something like that led us to a slightly higher fidelity graphical style and then, eventually, to the painted style we finally settled on. The transition between the different iterations took place over quite some time, and the final change was rather a bit too close to deadline for my personal liking. However, seeing the final product look like it does did vindicate the effort.



All the framework and testing and design considerations took a considerable amount of our 3-month deadline. In fact, it was only when the deadline counter started ticking past 3 weeks did we realise that it was time to create an actual game. By this point, we had a framework for level creation, most of the requisite art assets and the underlying engine that would power everything, but we didn't actually have any game content or puzzles. The last 2 weeks of the deadline were spent in a frantic rush to make a fairly complete offering for the competition, as well as clearing out all the bugs that reared their heads.

At this point we looked to the original UQ prototype extensively for inspiration on setting and puzzle designs. Many of the puzzles were adapted and extended from the original ones. We lengthened many of the originals, adding extra sub-puzzles and layers of complexity in the mix. The result was a collection of puzzles that were quite devious and very much in the traditional adventure game spirit, with many convoluted solutions that we hope are fun to solve, even if they may be a bit too challenging to stand on their own without some sort of training puzzles for preparation. The game was essentially two meta-puzzles: The first involved getting money for your character to use to purchase goods at the store, and the second was to get membership into the adventurers' guild. Each of these was achieved by solving a web of smaller puzzles that would eventually lead to the primary solution.

By crunch time, we had completed a game that was quite a bit longer than the original prototype, with a far more flexible dialogue system and engine powering it. There were still bugs to work out and testing to be done, which took up most of the last two or three days (thank the heavens for the deadline extension, both because of this and because of upload difficulties), but the game was in a more complete stage than anything I'd made before, and we were proud to submit it to DreamBuildPlay.

But, as much as I'd like to say the journey is over, we're now planning to work hard to get this game in a complete, sellable form so that it can be placed on XNA Community Games (which launches near the end of November for all Xbox Live members) as soon as we can. There are quite a few things that need to be done to achieve this: A rework of the initial puzzles and tutorial sections to make the game simpler, rich Marketplace and Live features (like a demo mode and user presence information), and general tweaks to improve playability. We're confident we have something we can sell and, in fact, hope to use the game and the resultant engine as a stepping stone for future adventure titles on XBLA.

"SpaceHack is a Hack-and-Slash, Rogue-like top down shooter. In space." I have lost count of the number of times I have uttered that phrase in the past few weeks. Add the number of permutations (like adding SHMUP or mentioning Bullet-Hell if the person I'm talking to looks like a gamer) and I'm surprised I still feel enthusiastic about the game at all. I always forget to emphasise the random generation of EVERYTHING aspect…

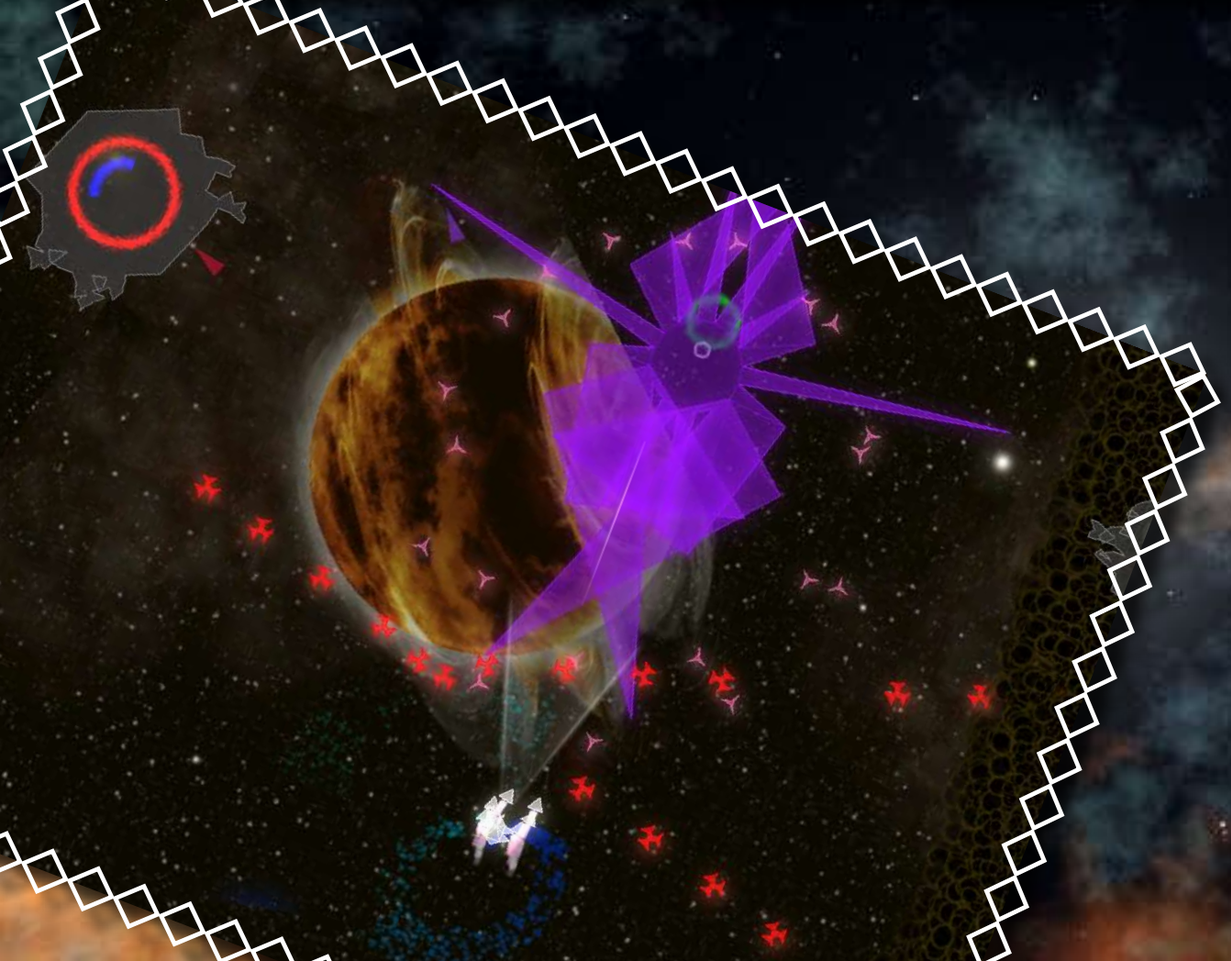
Danny "dislekcia" Day

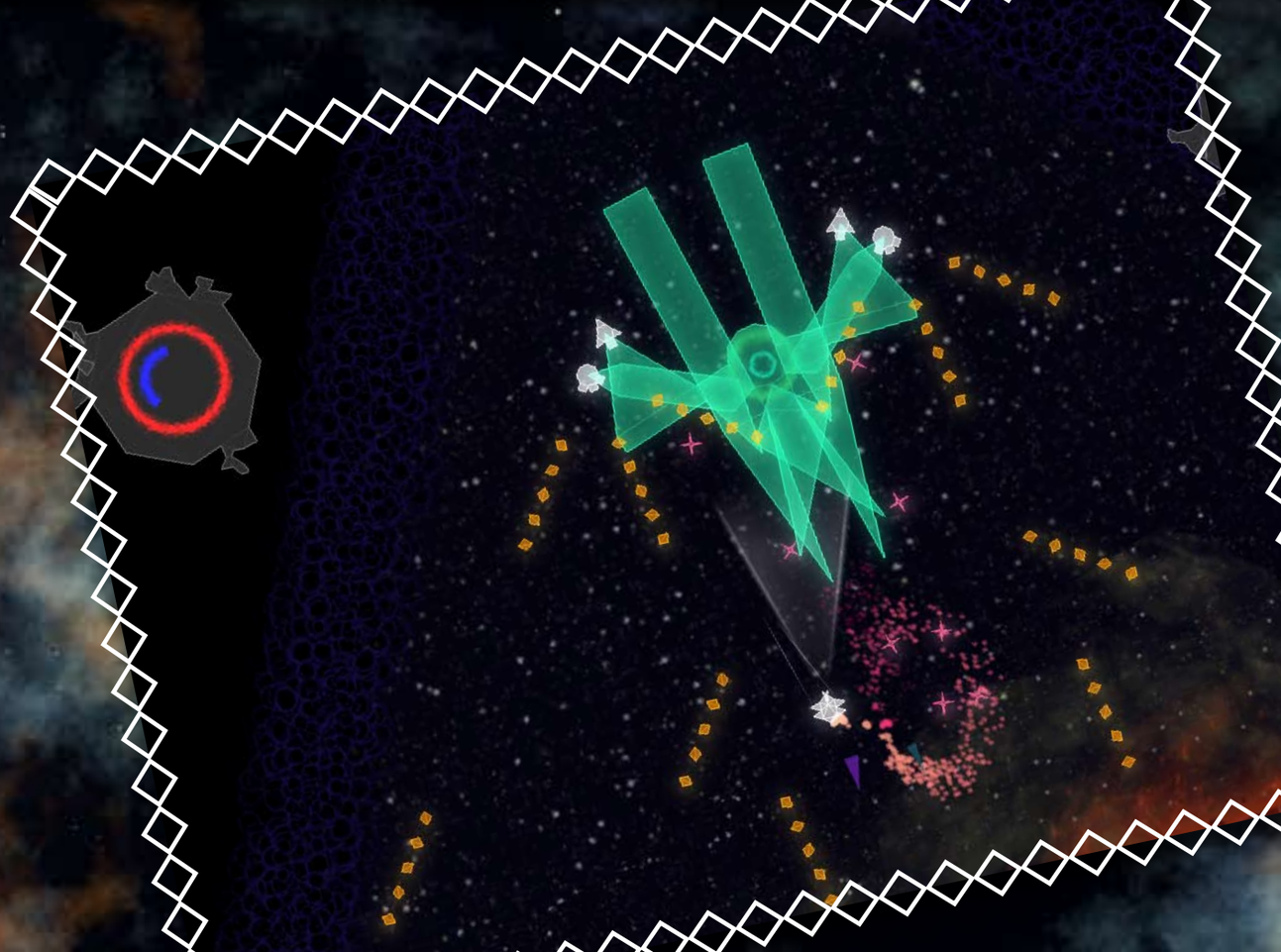
But, hand-in-hand with most of those explanations have come the smiles of people playing our game for the first time. Exclamations of surprise; coos of "cool" at the enemy shapes the generators create; screams of triumph and defeat. All of these blend together into one simple message: People are having fun with this; they're enjoying something we feel isn't done yet, so we have to be on the right path here.

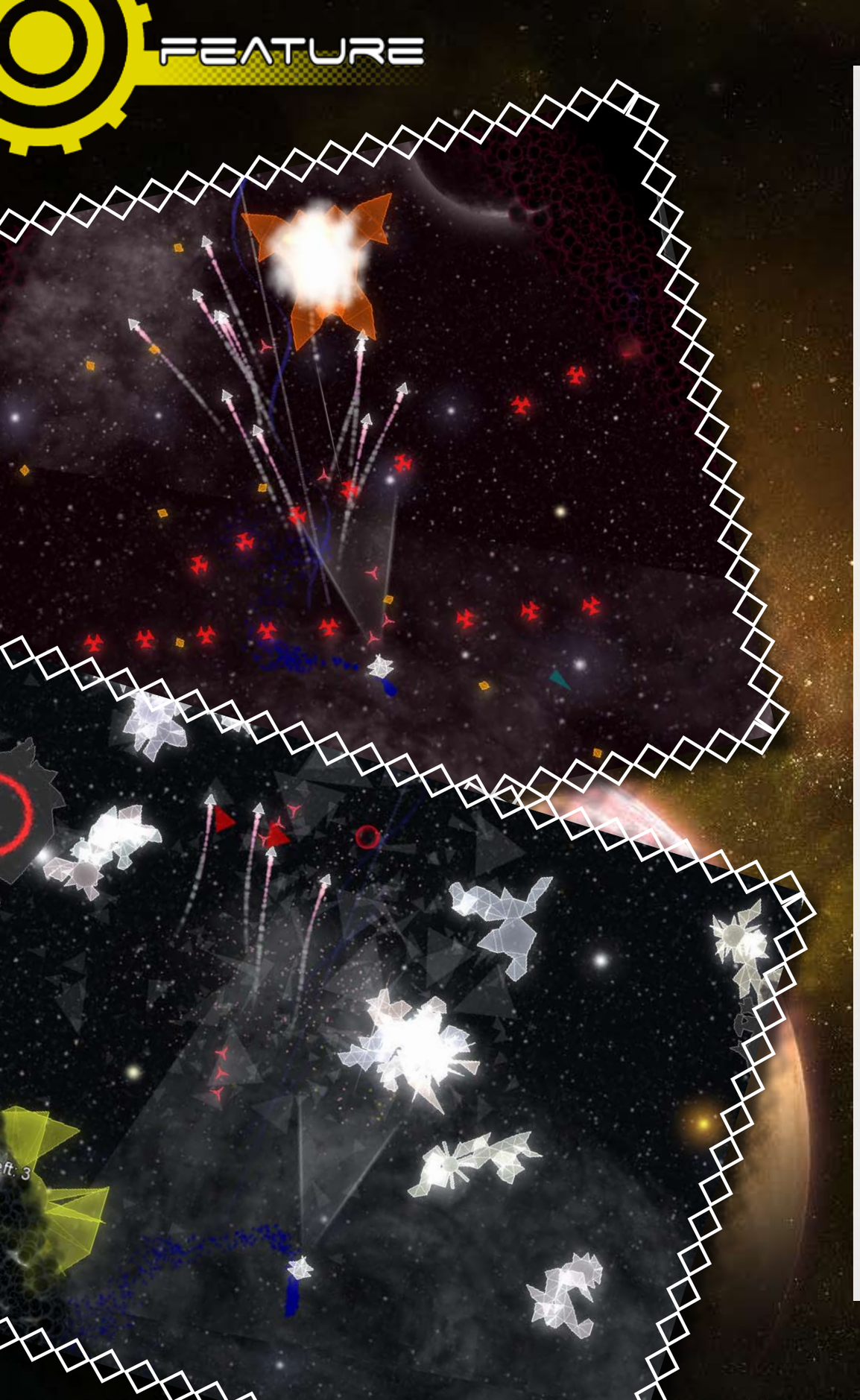
Along with that enjoyment comes the interest. People take notice when your indie game runs on an Xbox 360. They take even more notice when you drop in choice phrases like "infinitely replayable" and "randomly generated story." SpaceHack has opened a lot of doors locally – doors that I plan to keep wide open to the rest of the stuff that will emerge from Game.Dev. SpaceHack and Ultimate Quest are but the first of many. But I'm supposed to be talking about our game, so I'll reign in the big picture stuff and start again.

SpaceHack began as two things: A racing game; and a random ship generator for a grandiose space opera in the vein of Starcontrol 2. I'll look at the racing game first because it's the least intuitive.

At some stage late in my university career (so we're heading back to the scary days of late 2004 here folks), I had an idea for a 2D racing game with a different control scheme: Instead of having a player steer a car via discrete full left/right and complete brake/accelerate binary key-presses, why not have a player control a car's orientation on the road absolutely via the mouse? That would be an analogue turning system, providing much more control (I feel that a controller's analogue options make driving games much better). If the player kept smoothly dragging the mouse in a direction, they'd corner accurately and responsively. If they jerked the mouse right or left quickly, the car would go into a drift and behave differently, leading to gameplay possibilities via the controls. The idea was to have angles that caused the car to "let go" of the road, so drifting and spinning would be very important parts of the game.







I started prototyping the idea and quickly got a system going where the player's car would be stationary on the screen and the entire map would rotate around the car according to the motions of the mouse. I ran into issues with the car physics though, and my impetus quickly ran dry. Sarcastically, I added a shooting capability and a few arbitrary targets and filed the prototype away as yet another of those rainy day projects that could take some work when the inspiration to do something new was lacking. I revisited it a few times over the next few months, adding a star-field for grins and at one point spending an enjoyable few hours coding Robotech-style missiles just to see if I could. Everyone liked the missiles. That went into the book for later.

Around the same period, I was spending non-digitally enabled time writing down story ideas for a Starcon 2 style game, on actual physical paper. I was doing a lot of travelling due to my then girlfriend's family – marriages were going down, people were hitting important age milestones, that sort of thing. So there was lots of sitting around in hairdressing salons and idly daydreaming about galactic civilisations, the rise of sentience and the stabilising effects of odd anomalies like the Earth's moon, punctuated by obligatory "ooh's" and "aah's" at the latest sculpted and flower-adorned majestic head ornamentation.

Once people got used to the idea of being married and the rampant ageing had calmed down, I got back to my regular base of operations and decided that I hated doing art for games and this dramatic space opera would require far too much of it. So I started messing with algorithmic approaches to generating spaceships. Prototyping revealed that the simpler methods were best and the people that saw the resulting collections of polys were amazingly quick to give them recognisable characteristics: Calling them squid-like or crab-like as often as they observed that this particular one should belong to a race of robots. Both the huge galactic story and the random generation prototypes promptly went into the Big Stack of Game Ideas™ and didn't go anywhere for a while.

Now, I think it bears explaining that I am a Hack-and-Slash aficionado. I loved Rogue and ADOM. I have a MUD character with 4000 hours of existence. I played Diablo to death. Diablo 2 destroyed more hours of my life than I am ready to admit to yet. Darkstone, Fate, Sacred and eventually Titan Quest all scratched that itch. I'd always had ideas around a H&S of my own, but never any real foundation to build on, until one day the racing game prototype stuck in my head during a Game.Dev organised DevLAN about procedural generation. Why not take that control scheme and build an action-heavy H&S designed to be played to completion in short sessions? Similar in execution to PlasmaWorm's Strange Adventures in Infinite Space (play that game if you haven't already done so).

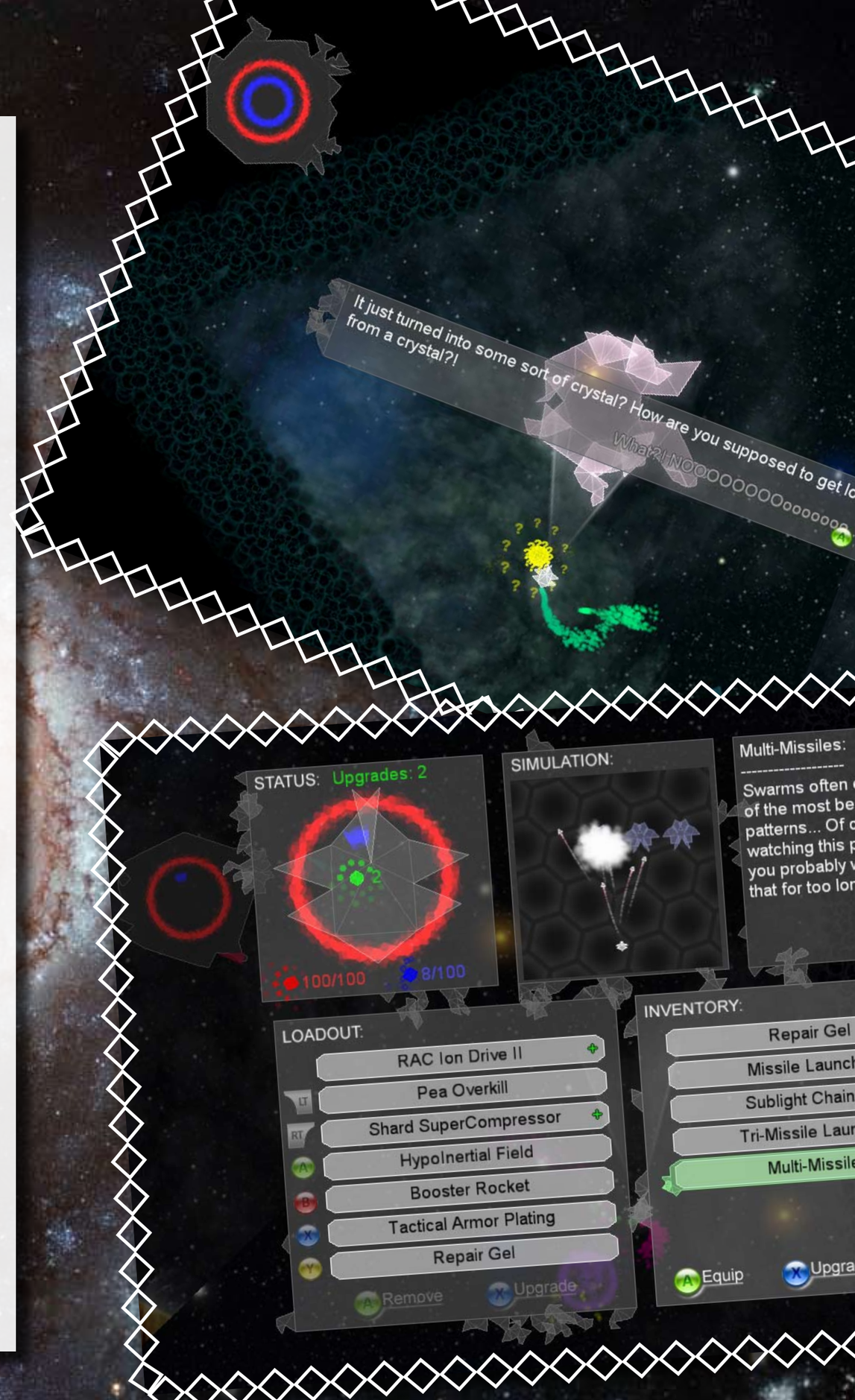
That would serve as the germ concept that eventually became SpaceHack. Every once in a while I'd add to the game a little, driven by inspiration, or a session of "what if the game was already done, what would it do?" mental masturbation. That's how SpaceHack got its unique elements: things like combining skills and items into items with branching upgrade trees to streamline the play experience, but still give players that thrill of random drops, and customisable upgrades with options to explore differently next game; or the weapon preview scene, to show players exactly what a newly acquired item would do when either equipped or upgraded, (that way it wouldn't need huge and possibly ambiguous, or worse, confusing text descriptions that bothered me in other H&S games).

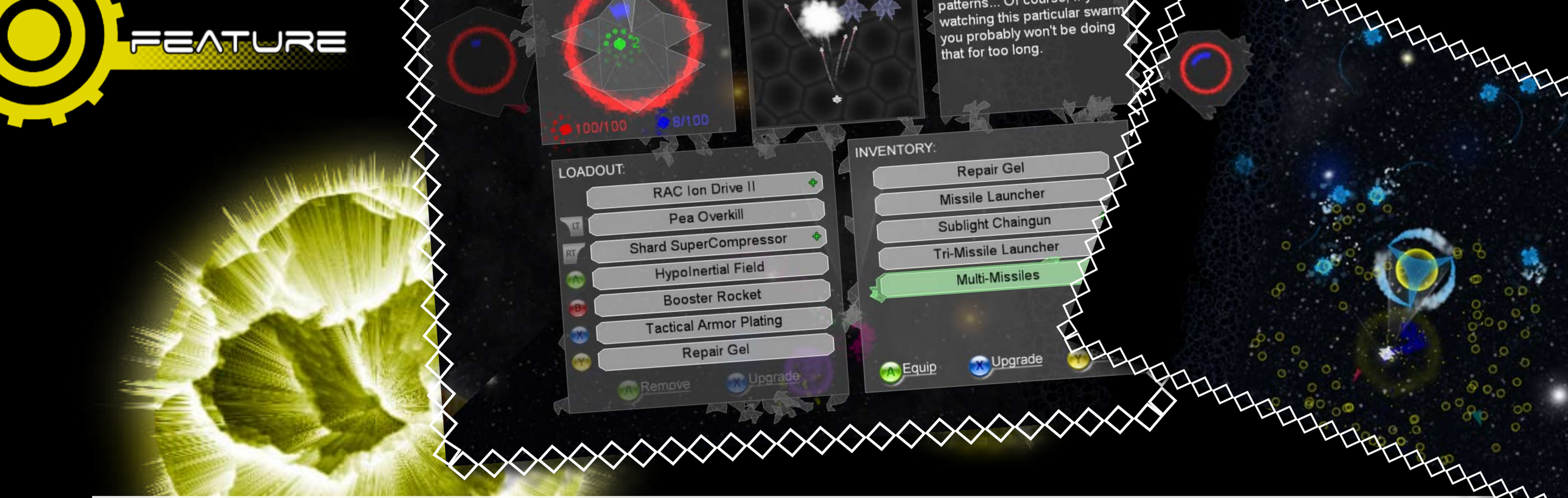
That was why I picked the concept as my entry into the first DreamBuildPlay competition in 2007. I wanted to make it a console game and the idea (then called Void Escape because I suck with names) really made sense in the console space – a platform devoid of a good H&S for me to play. DBP proved a perfect excuse to test out my ideas on procedural generation in a wider setting – the game would need randomly generated maps, events, enemies, weapons and there was even scope for a random story system. But as things turned out, time was not kind to me. A design contract went a little south earlier in the year and meant that not only was I struggling to pay for food for a while; I was also limited to just under 4 weeks to finish Void Escape before the DBP deadline.

I cut features like crazy and used everything I'd already learned about .NET from my previous DirectX 9 work, but it wasn't enough to get anything nearly complete out there. I slept roughly 14 out of 120 hours in that final crazy stretch before submission. Void Escape didn't place anywhere, but it did lay the engine groundwork for a lot of what would eventually become SpaceHack.

That November when I started my company, Quarter Circle Forward, to formally take over from my previous contract-based consulting, DBP 2008 was on the company schedule as something we were going to do. Whoever "we" ended up being at that point. Ideas for the next competition entry constantly spiralled their way into my design file: a much expanded Monochrome with a film noir story and multiple endings; a good way to do RTS on console natively instead of porting mouse/keyboard issues; a bevy of puzzle game concepts that I hadn't seen done anywhere else; numerous game mechanics inspired by, or hinted at in other games I played during that period.

The part of the file marked "Possible DBP" grew large. Thus it was that in May 2008, when my longtime friend and nearly-longtime housemate Marc "Aequitas" Luck quit his job to live off his savings and finally make games, DreamBuildPlay was high on the list of what we talked about.





When the competition was announced in June 2008 and the theme completely failed to help us make what had become an increasingly difficult decision as to which game to do, we ended up choosing Redshift. What better way to make a splash as a small studio than to finally make a workable console-based RTS? We began working: Aequitas was tasked with learning about shaders and getting his teeth into XNA, while I built a map system that would index by time as well as position and started on the scene framework we'd need for the game.

Aequitas made good progress. This was his first real game project (sure, he'd been to DevLANs before and produced the odd Game Maker prototype while there, but this was in a completely different league), so there was a lot for him to get up to speed on. 3D programming in particular, hence being set the mission to understand shaders – I'd worked with him at university before, so I knew he was up to the challenge. And up for it he was; pretty soon we were attempting to write shaders that would draw the smooth curves we'd need for unit prediction paths in the game – he'd supply the shader code and I'd work on the math.

A few weeks passed and we learned a lot, but made very poor progress. Redshift just didn't feel like it was gaining any momentum at all. I was running into a lot of stupid issues with the engine (at one point I simply couldn't get textured quads to render correctly at all, yay), and our forays into parametric line shaders proved only marginally reliable. It was time to re-evaluate the entry and see what we could get done realistically in the time we had. Void Escape and Monochrome started dominating our conversations, but most of the people we asked still liked the sound of Redshift. We were right back to where we'd started, unable to reach a decision…

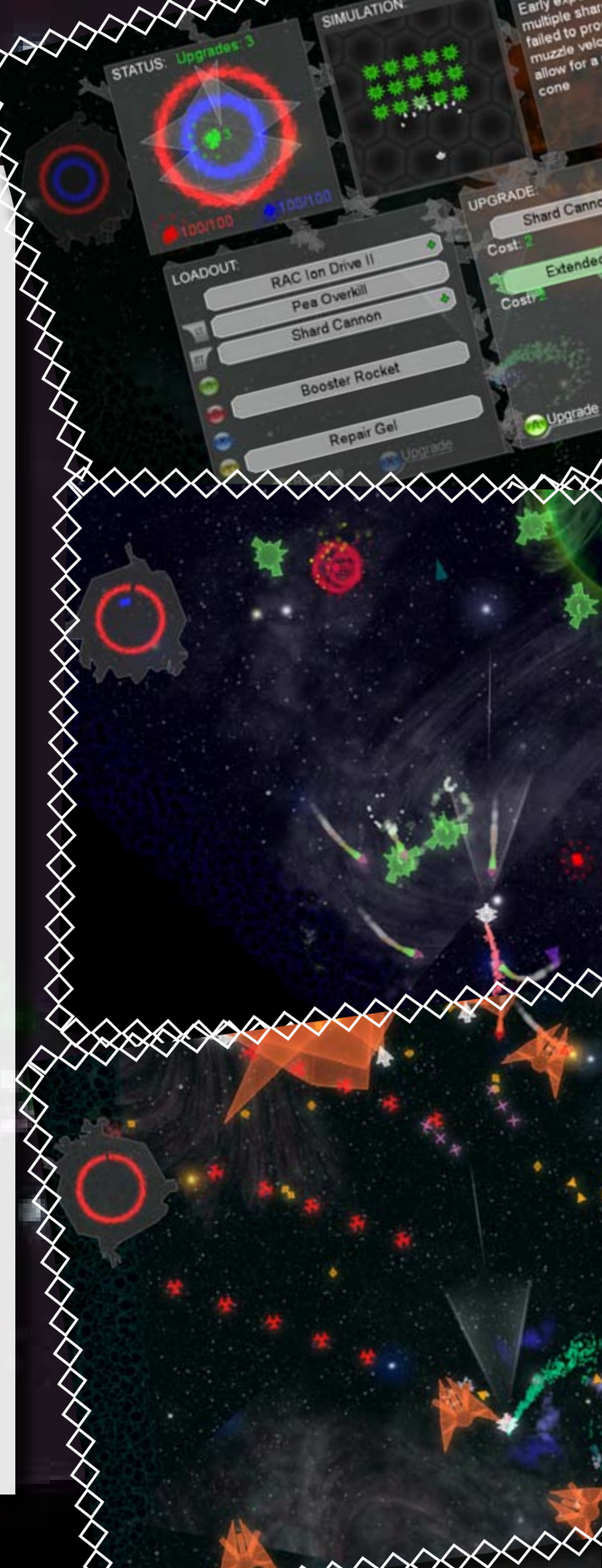
After a rather bad day we gave ourselves a week to mess with Void Escape, see what kind of progress we made, and pick our best shot at the contest that weekend. We never went back to Redshift after the first couple of days. Void Escape was renamed and just didn't have the bottlenecks that Redshift's design did; building on an already working (if horribly incomplete and limited) proto-system made all the difference in the world. Every time I wanted a specific type of functionality in the engine or scene system, I'd start working on coding it, only to find out that I'd already done it a year ago in that sleep-deprived death crunch. Plus, I apparently still comment under pressure, so I wasn't lost in the code at all. There were some glaring omissions, like a relative-to-parent positioning system for objects attached to other objects, but congratulating my previous self's foresight became a habit as we got further into development.

Working with someone else on the project is what made SpaceHack possible. Aequitas really proved himself in those short months. I started off handing him tasks with the idea that I'd take his outputs and integrate them into the game and engine myself, but after I spent nearly a week optimising some slowdowns with my quadtree implementation and was too busy to really pay a lot of attention to his work, he ended up integrating the entire particle system by himself – quickly turning it into one of the fastest parts of the game. I levelled up my management skills and let him get on with what he needed to do. Our previous theoretical game design ramblings proved invaluable at this stage: he would get what I was going on about and not only figure out what I would need programmatically to get it done, he'd add his own understanding to that and produce stuff that could logically do more without getting dangerously stuck in feature creep.

Aequitas understood SpaceHack from a player's perspective instantly; we knew how the other spoke about games, which meant that there was (and still is) no design document for the game. We had a few dedicated design sessions to handle story or broad thrusts of enemy design, but in general most of the game evolved organically from the core concepts as they started coming to life. We'd discuss the game on the fly and things would just magically emerge as cool and functional after that. On top of the shader-heavy implementation code, like the particle system (when he showed me 40 000 particles onscreen simultaneously, all animating independently without any slowdown I freaked out a little), or the geometry instancing that we ended up doing to kill a hitch we were getting with enemy bullets (of which there are usually MANY), Aequitas turned the preview system from theory into reality and handled all of the enemy and boss firing patterns while I worked on the random systems and eventually player items and story.

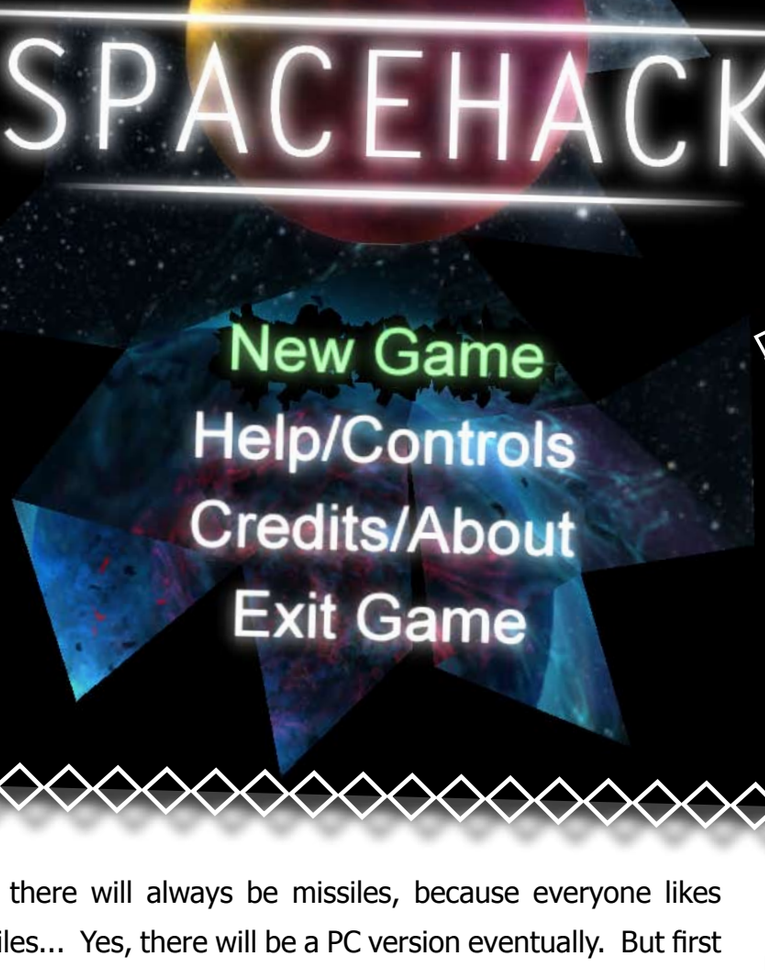
I think the only regret either of us has regarding the game is that we want there to be more of it! The random generation systems that populate the game-world with maps for the player to explore, put enemies in those maps and give the player a story to follow currently only have a fraction of the content we'd like them to have available. The version we submitted to DBP has over 80 boss and enemy behaviour segments (which the game combines in any order according to heuristically determined difficulty levels that depend on how well the player is currently playing), roughly 30 story events (unfortunately only 1 major story arc though; the nanomachine one didn't get done in time), and over 40 player items to find or unlock.

So while the generators can take all that content and produce a game that is already very replayable and different each time (it takes an average of 3 plays through the game to see everything that can happen and many more than that to get all the items at least once), we had to cut tons of ideas and neat things we wanted to do because we simply didn't have time. Pretty much all of the gameplay in the DBP demo version of SpaceHack was done in the last two weeks as individual segments, and is turned into a playable game on the fly as you're playing it. We want another few months to first produce editors for each type of system (story, enemy, player item) and then build as many objects in each as we possibly can so that you'll be able to fire up the full version of SpaceHack and play it a hundred times and get a different experience every time.

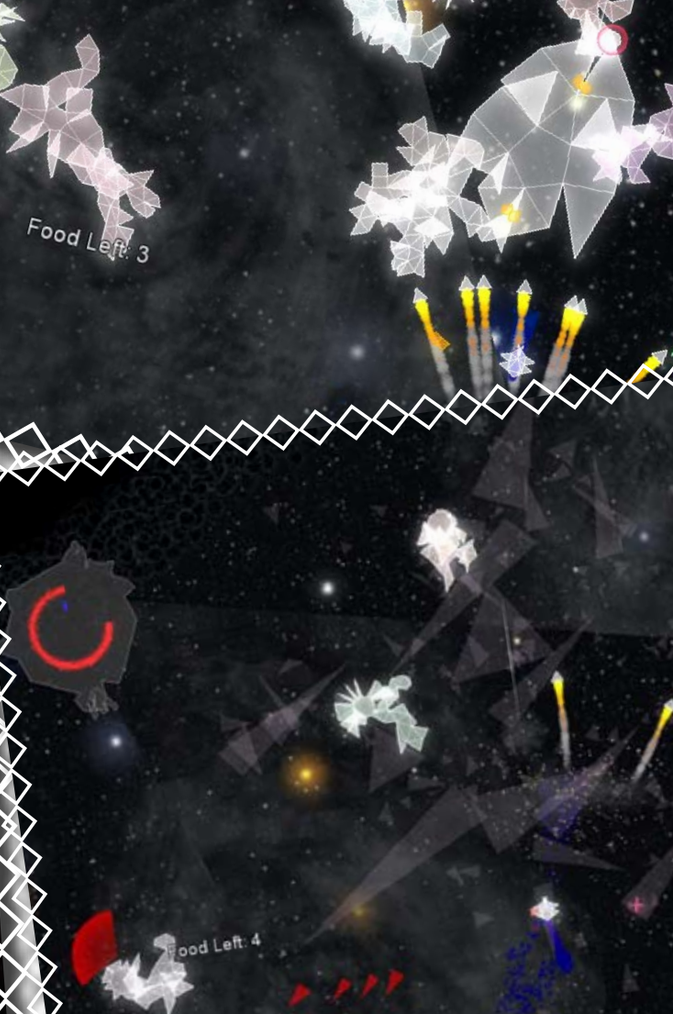


FEATURE

SPACEHACK



But there will always be missiles, because everyone likes missiles... Yes, there will be a PC version eventually. But first we need to figure out how a small studio doesn't go bankrupt if you don't have cool stuff to sell on eBay.



"LIKE TAKING A JOG THROUGH THE COUNTRYSIDE TO AVOID THE COMPLETE AND UTTER OBLITERATION OF YOUR SPECIES."

Run, Dino, Run

Chris "LionsInnards" Dudley

Nothing quite gets the blood pumping like taking a jog through the countryside. There could be many reasons why one would choose to do this: it could be to lose weight; it could be to get fit; or it could be to avoid the complete and utter obliteration of your species.

In Dino Run, the player takes control of a small, yellow, raptor like critter that was enjoying a beautiful day of sunshine, frolic and fun – up until the giant meteors started to pummel his homeland. Now he's desperately trying to find shelter to avoid the unstoppable cloud of gaseous death that threatens to cause his extinction; and the game tasks the player with getting him there.

Developers Pixel Jam, have created a little time machine here: a jaunt back to the days when difficulty stood for something other than how many shotgun blasts an enemy can take to the face; when "open ended" meant that one of the sides had fallen off your arcade machine; and an economic crisis was when you had run out of pocket money. It harks back to the glory days of Sonic – memories of rushing through the stages as fast as possible, grabbing rings out of the air and basking in the glorious speed will come rushing back.

It is a refreshingly straightforward game, one that has no ambiguity or deeper meaning, no next-gen shine and no ambitions to re-invent the wheel. It is pure, unadulterated fun, which will simply eat away at the player's time as they speed past the obstacles and race to the end of the level. And that's exactly what it is: a race. As the player skips and jumps through the deceptively detailed levels, an unmerciful cloud of death tries to catch up to you and put an end to your fun. The feeling of panic that grips the player as the screen turns dark is intense; the rumbling of the cloud fills them with dread as they try to scramble away over the rocks to safety.

The gameplay is almost flawless. A few minor gripes about slightly sticky controls occasionally rear their heads, but are immediately shushed by all the other awesome things that make this game truly special. Levels in which the dinosaurs form part of the structure of the terrain; the way the developers have included little touches such as dinosaurs drowning in tar pits (although it is hard to feel pity for them, as the player needs to hop on their helpless heads to survive); and the numerous game modes that cater for quick and easy play show the amount of effort put into this title. There are many different unlockables to be gained by replaying the levels, secrets to find and critters to munch; so multiple playthroughs are a must.

It's a game that shows that when a core concept is fun, it doesn't need Triple-Whammy-Game-Feature-X. It's polished, it's retro, it's challenging and fun, and it has dinosaurs. What more motivation could there be?

multiwinia

Gareth "Gazza_N" Wilcock

NEWS FLASH!

As of October 14, Introversion has released Patch 1.1. This adds several fixes and features to the game that were missing in our review copy. These include the addition of the lobby chat and game passwords that we complained about in the review. Now you good folks have no reason not to play this game. GOGOGO!

Anybody who's played Introversion's seminal strategy/action game Darwinia will know the joys of having thousands of little flat men marching across the map in a long green column, leaving the glittering red digital souls of dead viruses in their wake. It could be assumed that two or more such armies colliding would make for one epic battle. Well, it seems that Introversion agrees, because (as our preview in Dev.Mag 25 revealed) they've been hard at work on Multiwinia, a brand new stand-alone multiplayer pseudo-sequel to Darwinia. So now that it's finally been released, how does it measure up?

The premise behind the game is fairly simple. The Darwinians, a race of self-improving virtual beings created by Dr. Sepulveda, have evolved even further since the events of Darwinia. Unfortunately, dwindling resources have led them to go the way of all supposedly intelligent beings: they've split up into differently coloured factions of "Multiwinians" and have started blowing each other into pixellated chunklets. It's up to the player to take control and lead their group of Multiwinians to victory, either against friends on the Internet, or against the AI in single-player.

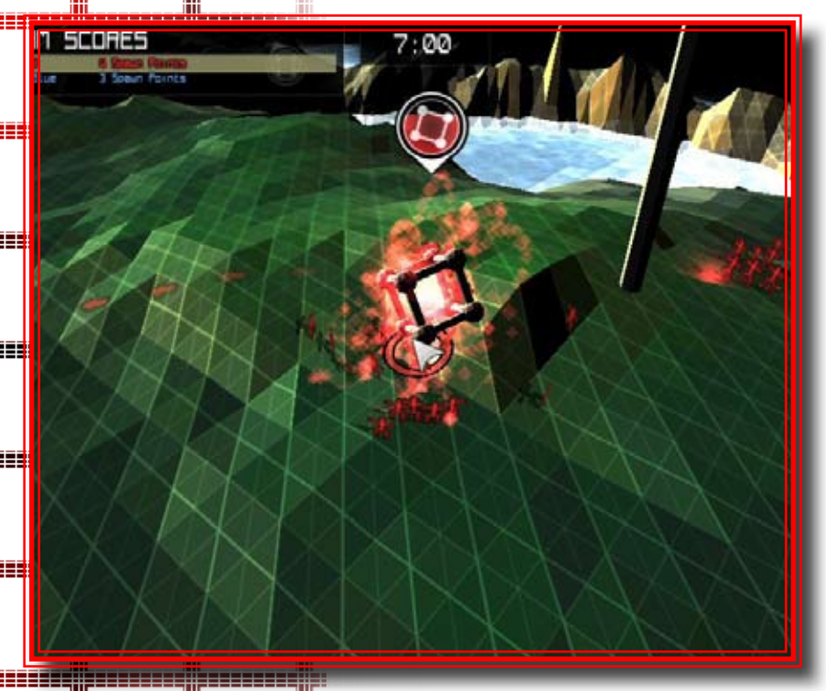
The game is broken up into six different modes, each with varying objectives. These consist of Domination – the objective is to capture the most spawn points across the map. King of the Hill – the player must hold circular control areas on the map to generate points. Capture the Statue – a CTF variant using giant statues that require multitudes of Multiwinians to transport. Blitzkrieg – each team must sequentially capture and hold a series of flags leading to the others base. Assault – teams must alternately attack and defend an objective on the map. A unique mode, Rocket Riot – the player captures and holds solar panels across the map until their giant rocket is fuelled, filled with 100 Multiwinians and launched before the opponents can launch theirs.

There's plenty of variety in the game modes to cater for any player, from simple kill 'em all slaughter fests to more complex challenges. All game modes are time limited, meaning that from the moment a match begins it's a frenetic all out race to complete objectives, or at least get ahead of opponents, before the clock hits zero.

Multiwinians are used to accomplish pretty much everything in the game world. They are injected onto the map at set intervals, either via large stationary portals, or at player-captured spawn points scattered across the level. In some modes, capturing and holding additional spawn points is vital to success, as the more spawn points controlled, the more reinforcements received each spawn cycle.

Control of the Multiwinians comprises exclusively of movement orders. Don't despair over this supposedly limited control, however; the Multiwinians themselves are fairly autonomous, and once in position will automatically and intelligently attack enemies, man machines, open crates, and generally interact with anything of interest within close proximity. Multiwinians can be manipulated individually, in small groups (using an expanding selection circle), or can be guided en masse by converting a single Multiwinian into an officer through a simple right-click. Officers, in turn, have two functions: they can act as non-mobile pointers, directing any nearby Multiwinians to a location; or they can gather Multiwinians into a mobile squad, which will follow the officer around the map in tight formation. Squads allow concentrated firepower and better coordination, but also move more slowly, are less autonomous, and are highly vulnerable to rear and flank attacks.

"THE DYNAMISM AND UNPREDICTABILITY LENDS A UNIQUE FLAVOUR TO EACH MATCH."



Their tightly-packed configuration also makes them prime targets for the grenades of non-squad Multiwinians. All in all, control is tight and effective, allowing you to quickly get massive hordes of Multiwinians to where you need them to be, while still allowing precision tactical control where necessary.

At this point one could be forgiven for thinking that Multiwinia is nothing but a game of logistics – capturing spawn points and setting up officer driven 'supply routes' to the front lines in such a way that the enemy is eventually overwhelmed. Whilst the game is primarily based on that concept, Introversion has decided to throw in a little something extra to spice things up: supply crates. Not unlike the classic Worms, these crates are para dropped randomly around the map as the match progresses. Once opened by a group of Multiwinians (the larger the group, the faster they open), crates provide the player with one of a vast selection of randomized powerups that can easily turn the tide of battle if used correctly. These can be anything from stationary gun turrets, APCs and personal shields, to a nuclear missile barrage which, hilariously, makes direct reference to Introversion's own DEFCON.

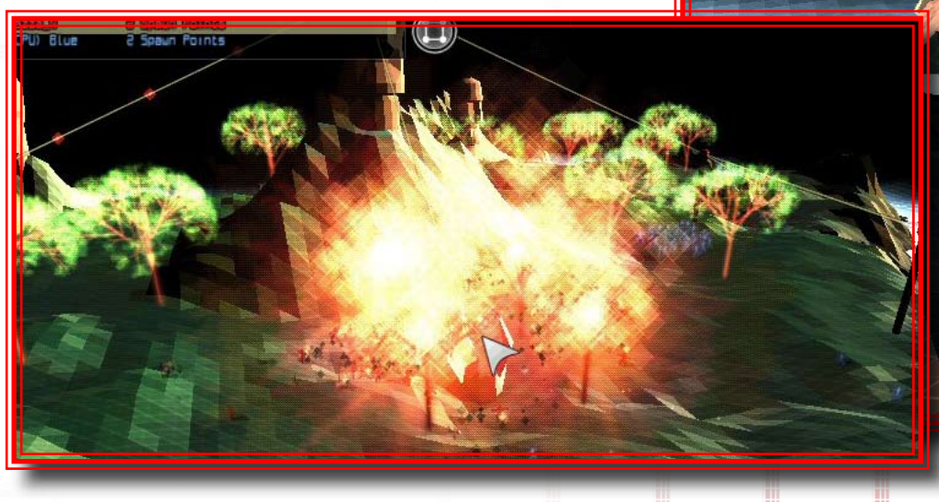
Unfortunately, crates can also contain nastier surprises such as viruses or other more elaborate hindrances to your war effort. Some may complain that the crates unbalance the game, but the dynamism and unpredictability they add lends a unique flavour to each match, forcing players to adapt their strategies and think on their feet every time they play. The game also provides the option for crate drop locations and spawn quantities to be balanced in favour of the underdog, meaning that losing players can be given a fighting chance should they find themselves the victim of one too many meteor showers.

Asides from one or two minor path finding issues with squads, the only major criticism that can be levelled at Multiwinia, oddly enough, has nothing to do with any of the usual suspects. It's the lobby system for multiplayer. While setting up games is dirt simple, the lobby system lacks any form of chat or password protection. Not only does this mean that random people tend to unintentionally blunder into a game intended to be private, but one is unable to tell them so. We were forced to discuss game settings over IRC, or via the in-game chat (yes, there is in-game chat, mercifully) once each match had ended. We also found that people with more unorthodox network

"MULTIWINIA IS SLICK, POLISHED, QUIRKY, AND ABOVE ALL, ENORMOUS FUN."

setups were unable to connect to multiplayer games at all, which was somewhat unfortunate. Fortunately, Introversion has confirmed that they're working on these niggles, and that they'll be fixed in an upcoming patch.

Lobby hassles aside, Multiwinia is exactly the kind of enjoyable yet unorthodox experience we've come to expect from Introversion. Hardcore players of strategy games may scoff at the supposed imbalances produced by the supply crates, but those willing to accept Multiwinia for the casual arcade-like romp it is, will reap hours of enjoyment from it. Multiwinia is slick, polished, quirky, and above all, enormous fun. Highly recommended.



http://www.shotbeakgames.za.net/RoachToaster.zip

ROACH TOASTER

"THOSE IN THE MOOD FOR A MINESWEEPER REPLACEMENT SHOULD LOOK UP ROACHTOASTER."

Chris "LionsInnards" Dudley

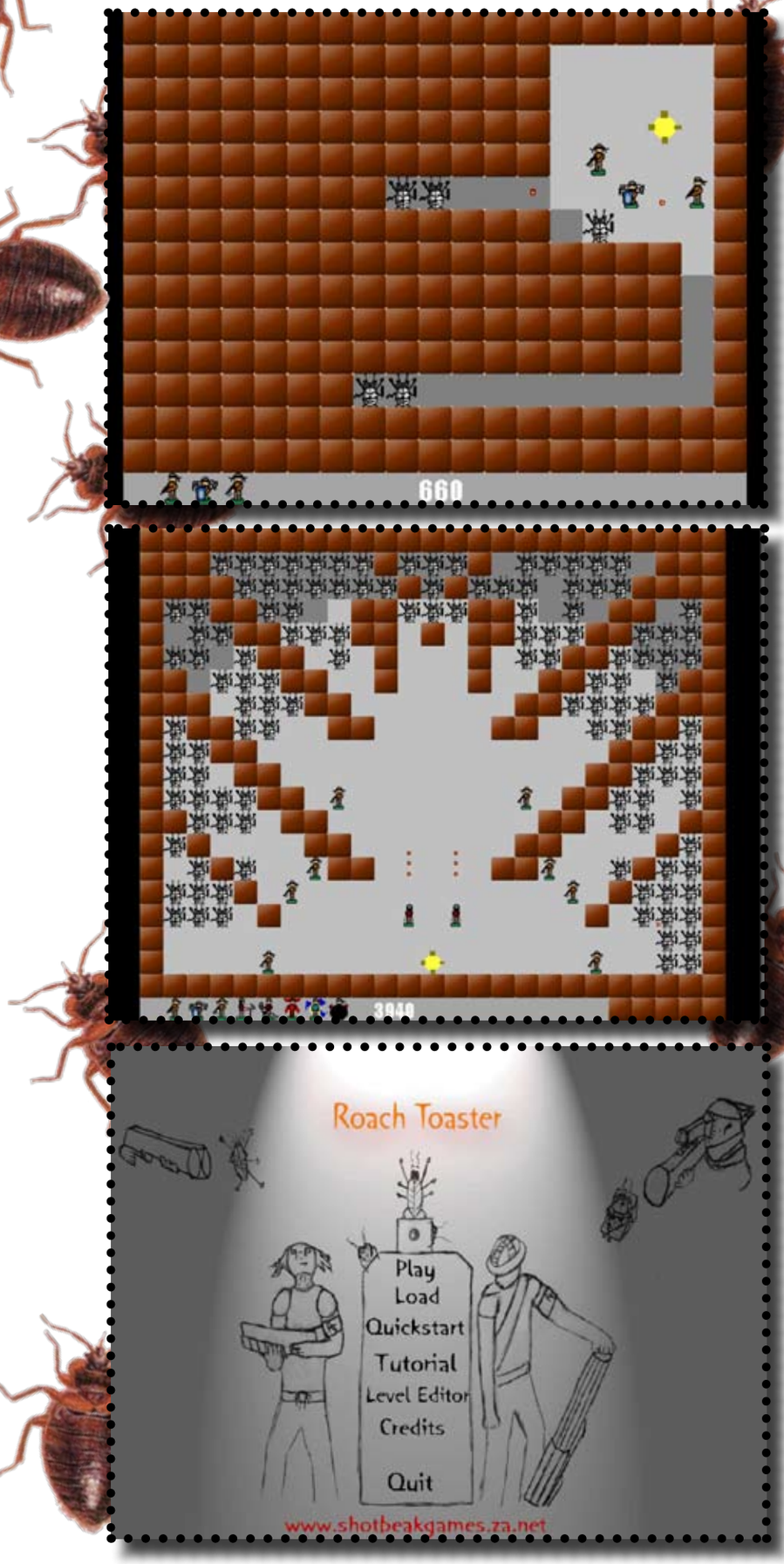
There comes a time when a gamer requires a certain type of game. They have work to do, its demand to be completed buzzing at the back of their head like an overzealous mosquito. They know it must be done, but before they can start they just have to play something, anything, to quench their game-parched thirst. They need a game that can be played for short bursts or extended periods and remain constantly fun. They need a game that is satisfying, challenging and entertaining. They need a game that involves shotguns, killer roaches and references to David Hasselhoff in a bikini.

"GAMERS NEED A GAME THAT INVOLVES SHOTGUNS, KILLER ROACHES AND REFERENCES TO DAVID HASSELHOFF IN A BIKINI."

That game is Roach Toaster – a creation by Game.Dev community regular Simon "Tr00jg" de la Rouviere, and a competitor for the "Most Likely To Get You Fired For Being Played At Work" award. Roach Toaster has similarities to both Minesweeper and Tower Defense; the first in terms of its casual appeal, the latter for its emphasis on smart decisions and planning. As the concise tutorial informs you of what you need to know, it becomes apparent that this is in a genre of its own. The premise is simple really: roaches are invading and the player is asked to dispatch them by strategically placing different classes of gun wielding soldiers onto the gameplay grid, in the path of the roaches. The rapidly swarming vermin multiply to fill all grid spaces around them, so some forethought as to where they might be headed is a good idea. The game's complexity becomes apparent when the player is introduced to the currency and playable area systems.

Any feelings of being overwhelmed will dissipate soon enough. Once the player has played a round or two (which generally last between five and ten minutes – great for a coffee break) the basics seem natural. Before long, players will master setting up defenses, forming blockers and clearing out roach holes. Amusing dialogue boxes add a bit of story and humour to the levels, and subtle changes to the level structure force adaptation.

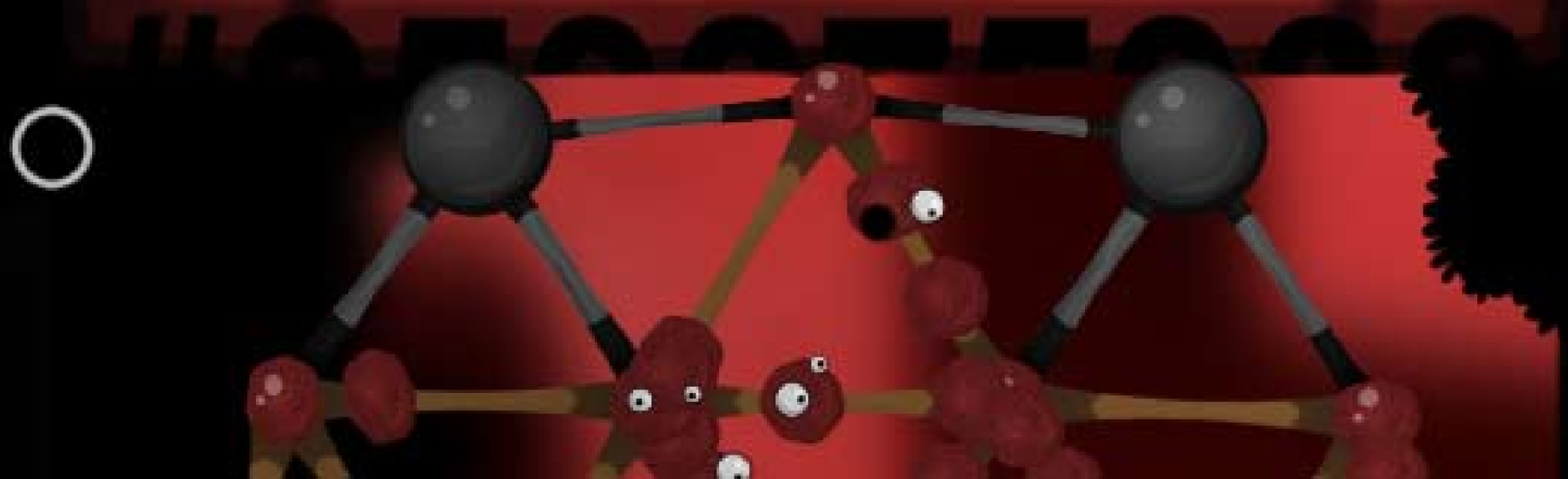
The few flaws that occasionally rear their head are frustrating, but rarely game breaking. Occasionally the AI will get confused and do odd things, like firing at a wall or ignoring an approaching army, but this happens very rarely. A couple of other minor issues will undoubtedly result in some nasty words being uttered, but are outshone by the sight of the final roach being gunned down, or a well planned assault working out perfectly. Those in the mood for a minesweeper replacement should look up RoachToaster. The rewarding action, silly writing and included level editor make this indie title shine above the rest.



World of Goo

Rodain "Nandrew" Joubert.

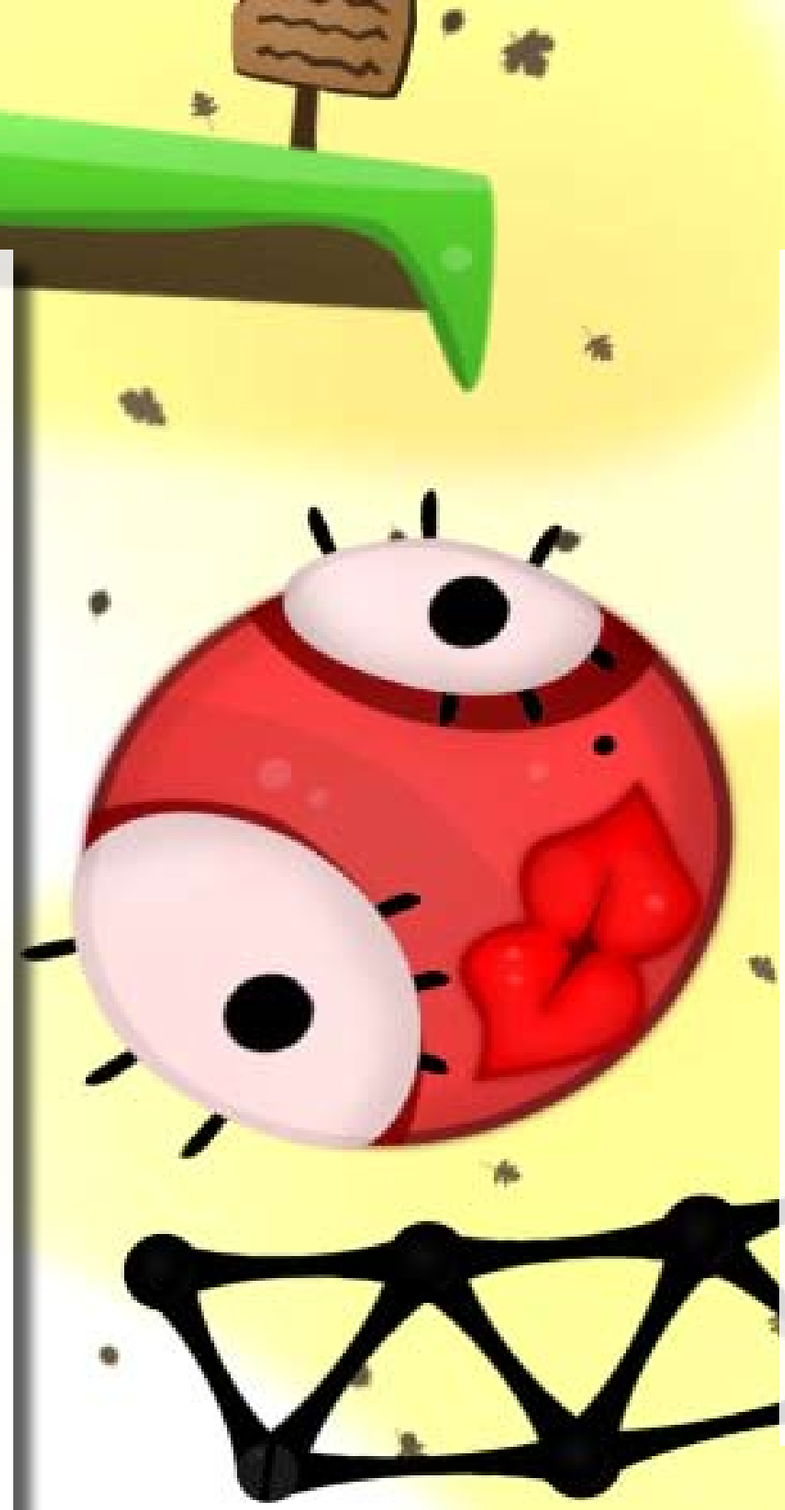
I must admit, I've been a philistine. I played the original Tower of Goo back when it was on Experimental Gameplay and thought that it was entertaining enough. It didn't blow my mind or anything, but it was worth the time spent playing and I gave it a mental 'thumbs up', before moving on to the next funky prototype in my "To Play" list.



A while later, I heard about World of Goo. My friends were going insane about it – the game had gone mainstream, it seemed. They pleaded, cajoled, threatened and wept in their efforts to get me to play it, but I just shrugged my shoulders and said, "It's another physics game. Maybe I'll get around to it later. Maybe."

Today, I shed gooey tears of remorse when I consider my doubting ways, because World of Goo really is one of the best titles I've played recently. This game is a raw, sticky delight from beginning to end – a masterpiece consisting of gooballs, wicked humour and some really neat backing music. Combine this with nice physics, inspired level design and a visual identity which looks like Tim Burton's take on a Worms game and you have a squishy, stretchy masterpiece which towers over the competition.

Throughout the game, the basic premise remains the same – the player given a bunch of gooballs and the job is to connect them with one another to form a structure that can reach a pipe at the end of the level. Any gooballs that aren't used in your construction go into the pipe. If enough of the little globs have been gathered, the level is won.



World of Goo doesn't stop there though. Different species of gooball are introduced as the game progresses, each one with their own particular perks and drawbacks. Some require fewer bonds to form structures. Others float. Some can even catch fire. And others are, well, others just really need to ease off on the makeup.

Environmental obstacles also play an important role. High barriers, spinning blades and the ever-present Pits of Doom™ keep players on their toes, and make for a set of interesting challenges without becoming frustrating. Smooth play is facilitated with the presence of little "time bugs" on some levels. These critters exist in a limited quantity and allow players to go back in time by one move, removing that ever-dreadful experience of, "Oops, I made one bad move after placing hundreds of gooballs, and now I have to start everything all over again!" Thank you, developers, for this small kindness extended to end users.

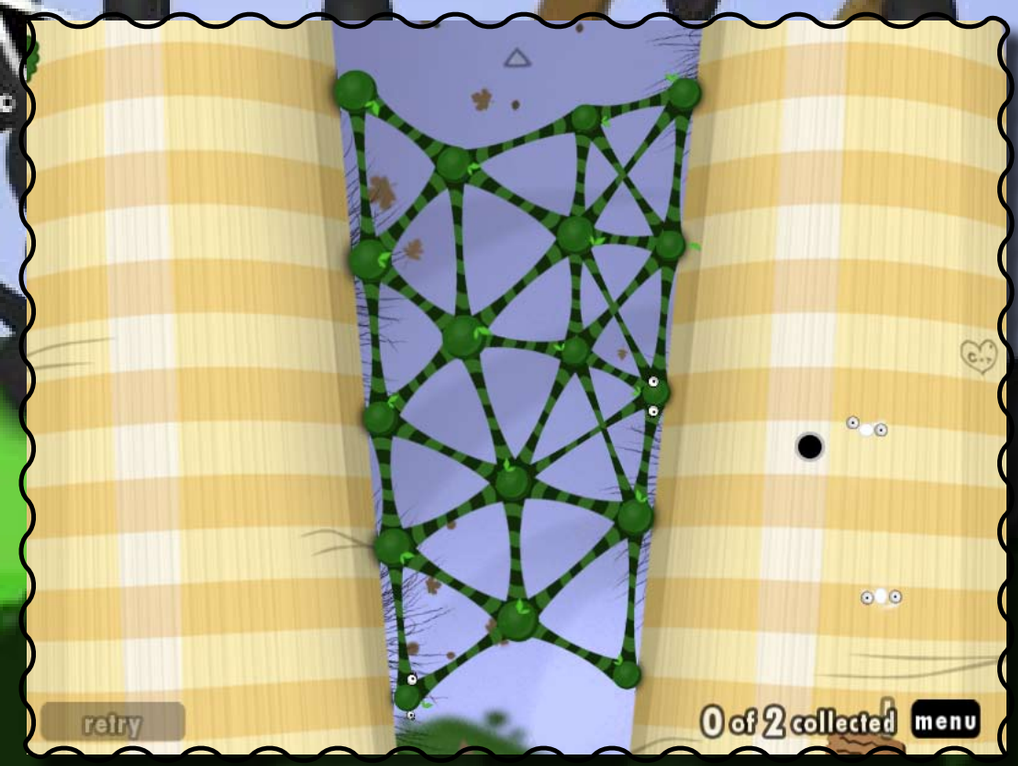
"HIGH BARRIERS, SPINNING BLADES AND THE EVER-PRESENT PITS OF DOOM™ KEEP PLAYERS ON THEIR TOES."

REVIEW

If I were to stop slobbering over this game for just one moment to critique it from a game development perspective, the following strong points would show up for me:

- The feature creep is well-balanced and introduces new concepts to the player at a respectable rate. More importantly, each new perk and improvement actually makes sense in terms of the story and gameplay, and doesn't make previous abilities or scenarios obsolete.
- The game focuses on a few base points and polishes them to perfection, rather than trying to overextend itself or throw on fluffy extras that are poorly implemented. Everything in the game feels like it has been very carefully tested. It's great to see how the developers have expanded upon the original Tower of Goo prototype to create a product that capitalises on the original's strengths.
- The meta-game elements are superb. Upon finishing a game, the player still has more goals to accomplish – attempts can be made to acquire OCD achievements, for example, finishing levels within a limited number of moves, or saving a certain amount of gooballs. There's also a sandbox mode that utilises all the extra gooballs from the main game mode which allows you to build as high a tower as possible using an open playing field and all the goo you can muster. Basically, anybody playing World of Goo is able to set goals and unlock achievements above and beyond the basic "get to the end of everything" scenario – and the developers have made this rewarding.

I give this game five gooey stars. It's a brilliant title which will keep you slushily entertained for hours, and it has a lure that will have you wanting more long after that final gooball has been drained away.



IN CASE OF EMERGENCY
BREAK THE MOULD

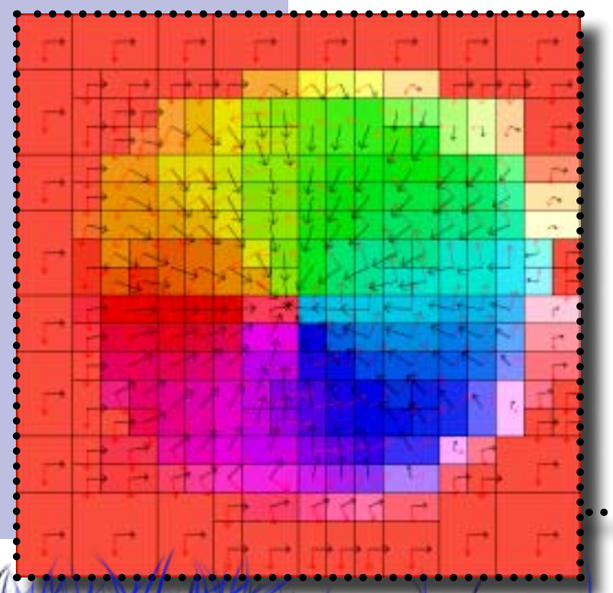
QUAD TREES

Herman Tulleken.

In the previous issue, we looked at a simple implementation of quadtrees – the kind that can be used for compression of 2D data. In this issue, we look at when to use quadtrees, how to choose a threshold, issues that might arise with specific applications, and how you can modify the quadtree algorithm for some applications. The images provided demonstrate most of the principles discussed here, but it is important to remember that defects that are very visible in images might be totally invisible in other applications.

Quadtrees are by no means the only 2D space-efficient data structures – typical image compression algorithms can easily outperform quadtree compression. Here are a few things to consider when deciding on whether a quadtree is the appropriate data structure for your application.

Is random access required?	Can quadtrees deliver a significant amount of compression?	Is the detail level too uniform?	Does the data have large regions of uniformity that compress well?	Does the data need to be updated regularly?	Is the blockiness of the resulting data acceptable?
Quadtree lookups are faster than lookups into some other compression structures. If you only need to access pixel data sequentially, some other algorithm might be better (run-length encoding, for example). If you only need to access data infrequently, you might consider an algorithm that is slower but provides better space efficiency.	This will need to be checked the solution is implemented, but it is possible to make some rough calculations. If savings are not significant, you are wasting your time. Do not use quadtrees to gain incremental savings – rather use another data structure. A related question: is the amount of data huge? If it is not, a 2D array will do perfectly.	Quadtrees work best when the detail in your data is non-uniform. If it is too uniform, then using quadtrees might not be the best structure – a low resolution grid will be easier to implement. However, quadtrees are still a good solution when the uniformity of the data is unknown in advance. Also note that the uniformity of the resulting tree can be tweaked by using a non-constant threshold (covered later).	If not, you might not get the compression you want with quadtrees, and you should consider using another compression algorithm or no compression at all.	Although you can implement updateable quadtrees, it can be very tricky to get right. If you need to update your data, another structure might be preferable.	For images, it is generally not – high tolerance values must be used to achieve a good result. However, for many applications high levels of blockiness is unnoticeable and therefore acceptable. If you use force fields to steer agents, for example, you can get away with a threshold that is quite low, because an abrupt change in force only leads to an abrupt change in acceleration, not speed or position. You only see the effects of the force field at one point in time (per agent), so it is hard to visualise the entire field. For high fidelity simulations, or simulations of particles (such as water or smoke), using quadtrees to store force field data might be inadequate. There are however, some ways to counteract blockiness – see image.



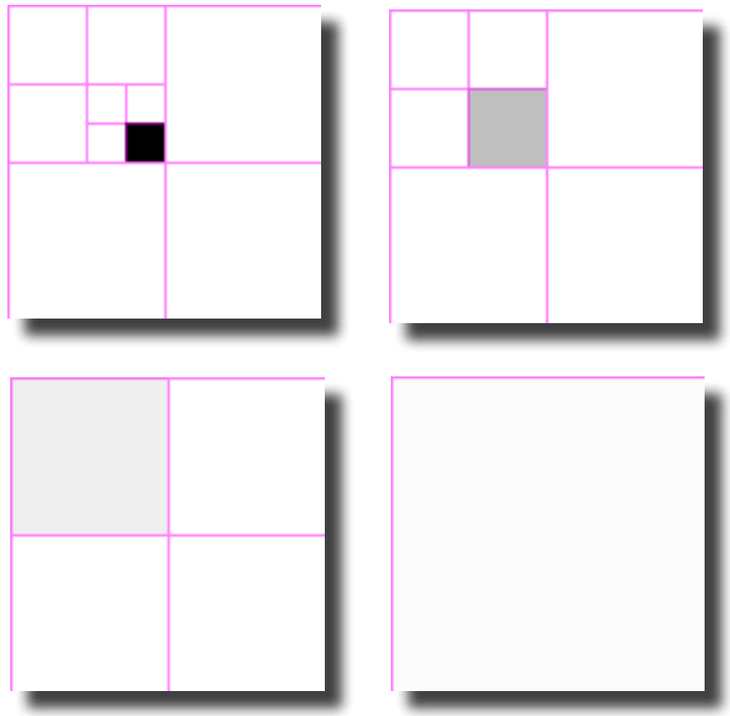
Choosing a threshold

There are no hard and fast ways to determine the correct threshold value (not without some sticky mathematics). Some experimentation is required, and after playing with your data you will get a very good feel for how threshold values affects the result. If you put some method to this madness, you can determine a good threshold value quite easily. The tests below can help you choose a threshold, but they will give you more insight into your algorithm – especially if you use exotic detail measuring functions. The tests below are also very good to use as regression tests, making sure that you do not tweak your algorithm and cause some unforseen anomalies.

It is important to adjust your test data to your application. Use grids that are comparable in size, and judge the results on how they affect things in your application, not how good it works on an image. But do use images to train yourself to see your data, and interpret it visually.

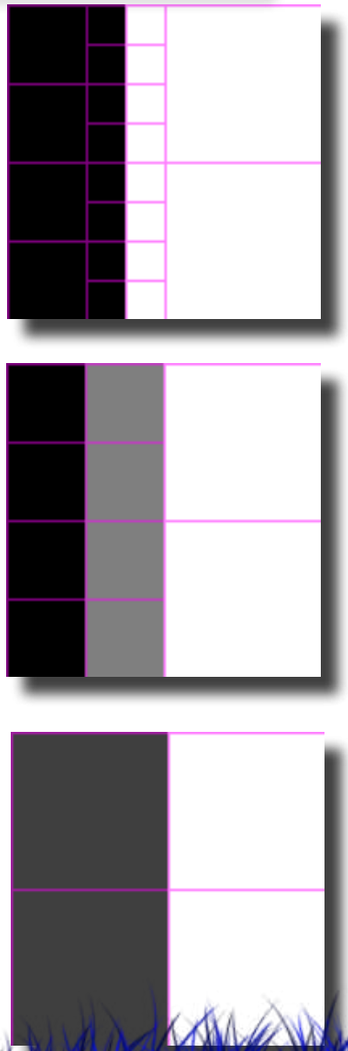
The single pixel test:

This tests the effects of quadtree compression on high resolution detail. The basic idea is to put a single pixel with a different value in a homogeneous grid, and do some quad tree compressions with different threshold values. Depending on your needs, you can obtain a good maximum value. If you are not concerned with high resolution detail, you can skip this test.



The vertical straight edge test:

This tests the effect of quadtree compression on edges. Divide a plane vertically in two parts, and fill each part with different values. It is important to know that where you do the division is important. Test with different placements, but the worst case scenario occurs when the division is one pixel left or right from the halfway mark of the rectangle.



The diagonal straight edge test:

This test is the same as above, except that you divide your plane diagonally. Test with different angles (you only need to test from vertical (90 degrees) to halfway horizontal (45 degrees) because of the symmetry of the algorithm.

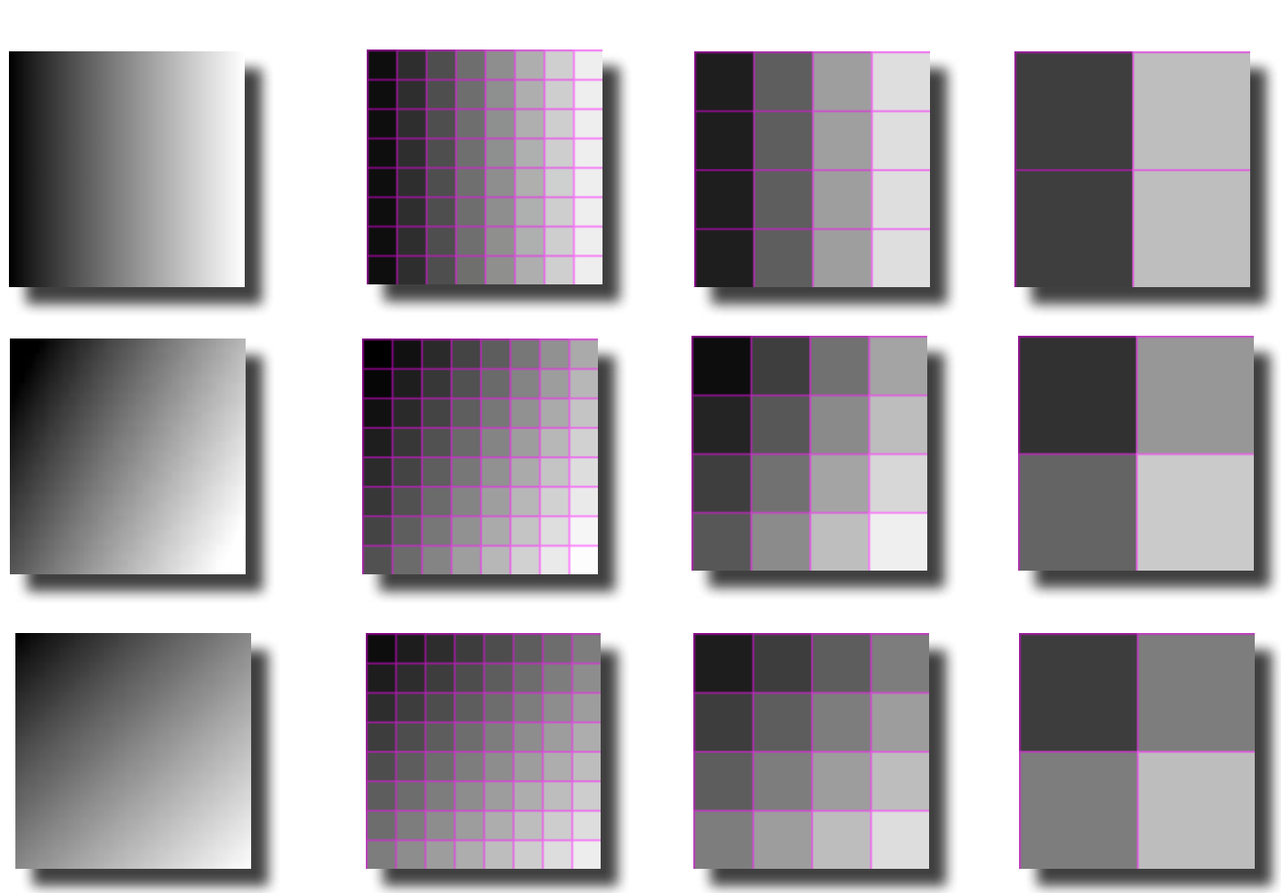
The gradient test:

This tests the effect on smoothly changing data. You fill a test rectangle with a gradient. Like the edge tests, you need to test gradients at different angles.

0 degrees

22.5 degrees

45 degrees

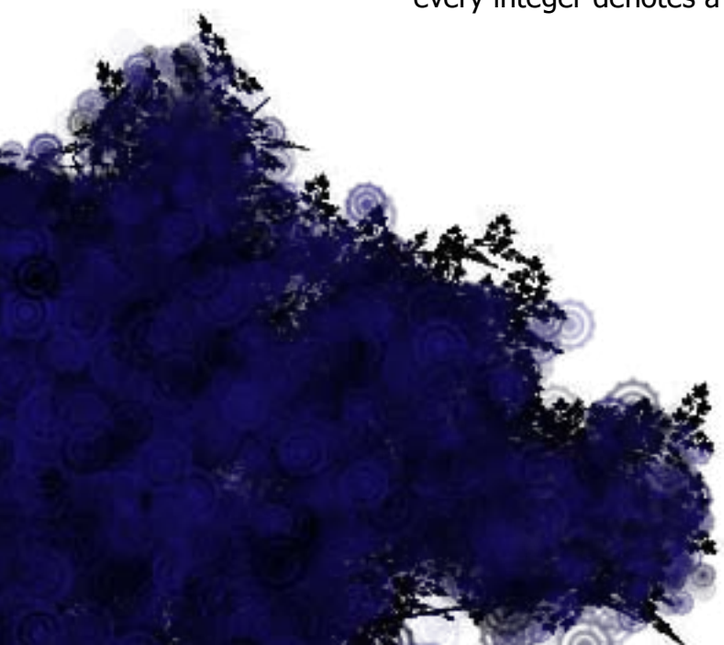


The circle test:

This tests the effects of quadtree compression on curved edges. Put a circle in the centre of your grid, and check the result. Test with circles of different radii.

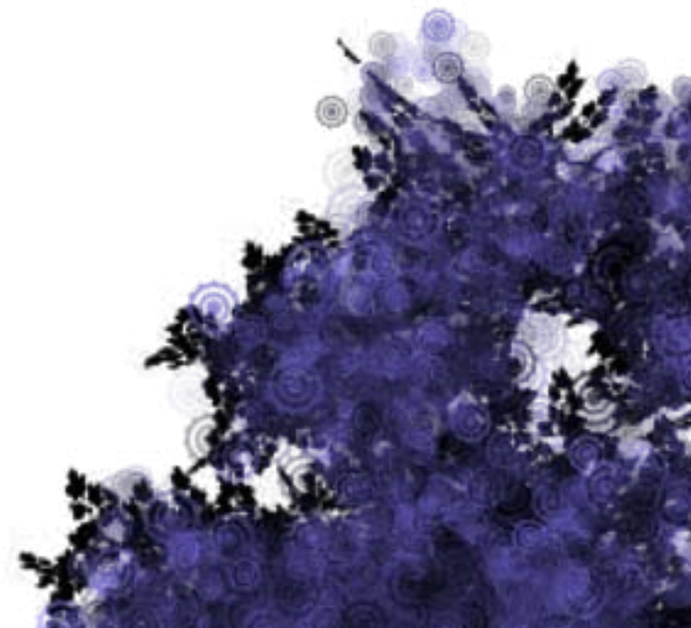
Handling Discrete Data:

In some applications, every cell might contain a value from a finite set (for instance, an integer between 0 and 10). A typical example is in a tile map for a game, where every integer denotes a tile type. (In a previous issue of Dev.Mag we explained how to procedurally generate such maps with Perlin noise).



Lossless compression:

Often, we would like the exact map to be retained in the tree. To do this, we need to set our threshold close to 1, so that only exact matches will be grouped together. If you make it 1 exactly, floating point errors might result in a maximal tree. You will only get decent compression if there are large areas covered with the same tile, so quadtrees are not suitable if this is not the case.



Lossy compression:

In some cases, we might not worry too much about the exact layout for the map – especially if the map is merely for decoration. There are two cases to consider:

- The sizes of the values are significant. For instance, you may have ten tiles. Tile 0 is water, tile 9 is land, and the tiles in between are each a mix of water and land, from more water to more land. Essentially, it means that the tile set is ordered, and that the order corresponds to the value used to represent it.
- The sizes of the values are not significant. This is the case, for example, when you have ten tiles – water, land, fire, grass, etc. There is no clear meaning of "halfway between fire and water". In this case, there is no ordering of the tiles, and the values used to represent them have nothing to do with the tile that they represent.

The methods of handling these two cases differ a lot from one another.

Maps from Ordered Tile Sets:

In this case we can store floating point values. When you get a query, you simply convert this into a suitable tile. You can round the value to the nearest integer, and use that tile, but this does not provide any benefits. Instead, take one of the following approaches:

- Return a random tile, biased with the fractional part of the value. For instance, if the value in a quadtree node is 1.25, you might return 1 with probability 0.75, and 2 with probability 0.25. This approach works well if your threshold is high. When using randomisation in this way, it is important that your generator takes an argument (dependent on the coordinates of the pixel you are querying), and returns the same number for that argument every time. A simple way of doing this is to generate a small square grid of white noise, and do lookups into the grid to get random numbers.
- Work out a new interpolated value (see how to do this below), and round this value to find a tile. This will not only reduce the blockiness, but regions will be more contiguous.

Using the above, we might use a lower threshold, resulting in better savings.

Maps from Unordered Tile Sets:

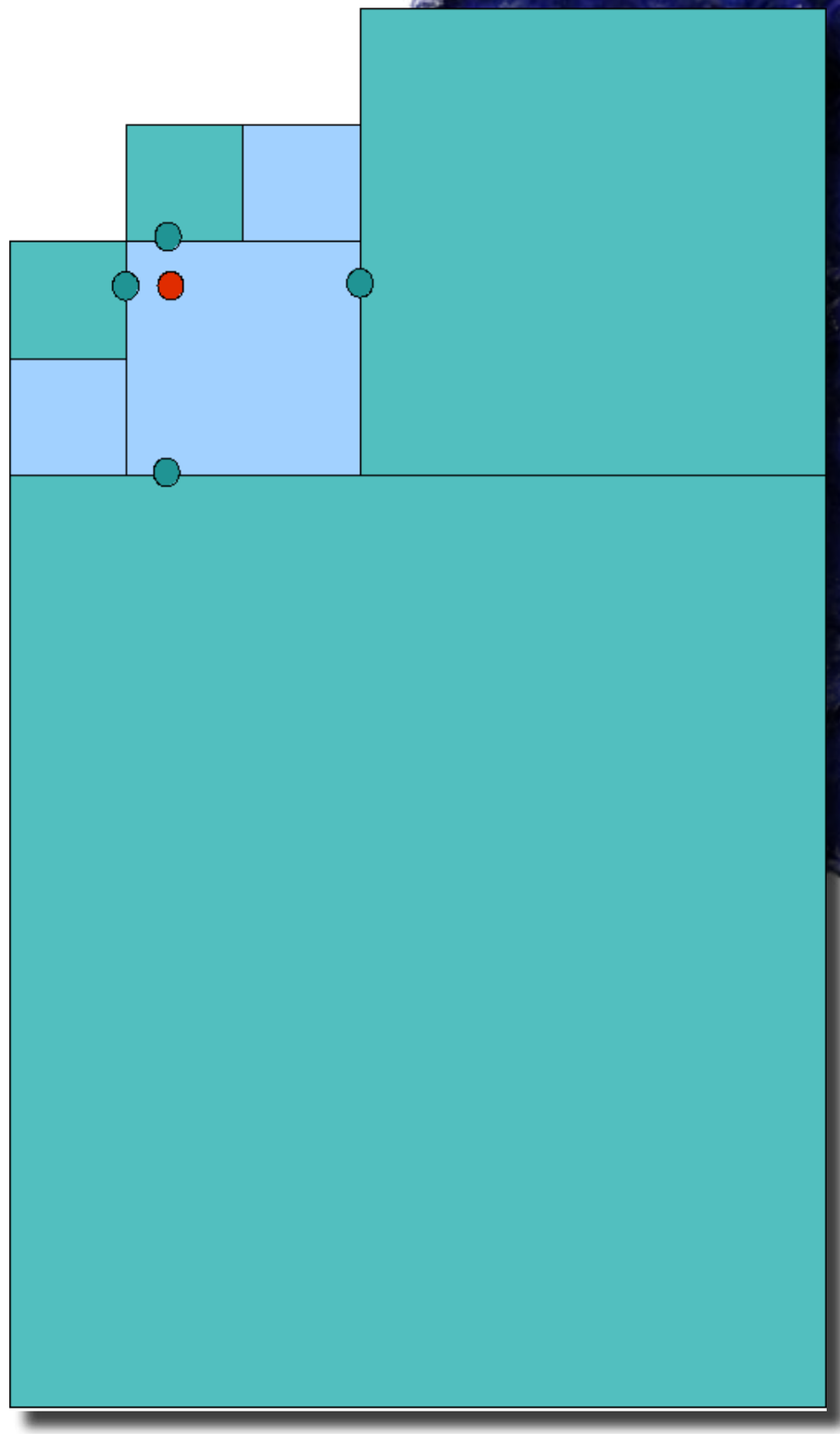
Here we need to take special care not to lose information. We cannot simply use averages – for example, the tile halfway between tile 0 and tile 2 is generally not tile 1. To handle this case, we need to do several things.

First, we choose the maximum number of tiles that can be depicted by a node. The algorithm is easiest if this is a power of two, especially if we choose 2. Call this number N. Now we cannot store single values of tiles. Instead, we store two vectors: one containing the tiles represented in that region; the other one the percentages of each tile in the region. These two vectors are both of length N (the last one need only be of length N – 1, since the values must add up to 1, we can leave out one value and calculate it instead). We can now use this to bias randomly selected tiles. Note that if N is higher, more detail can be dismissed, but you need to store more data per node. Lower values will result in bigger quadtrees, but lead to more efficient storage per node. You need to experiment to find the magic value for your data set. This method is so complex, that I would only recommend it if you really need the extra space that it might provide.

Changes to the basic algorithm:

Interpolation:

This approach is suitable for data characterised by large contiguous regions of smoothly changing data. It is possible to get a result that is completely smooth, but the algorithm is very complicated because all the squares can be different sizes, and any square (represented by a tree node) can be surrounded by a very complex arrangement of other squares. There is a simple way to get a crude result that works for many applications. For every pixel, you find the closest top, bottom, left and right neighbouring nodes.



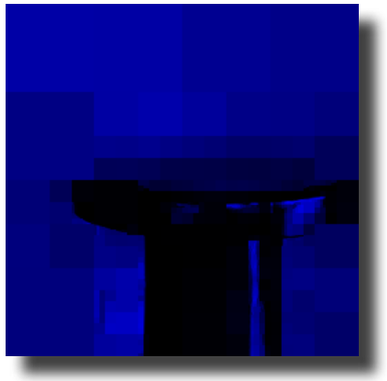
Now you have five values. The new value can be calculated from an average of the five colours. You can even weigh the average according to how far the pixel is away from the centre, like this:

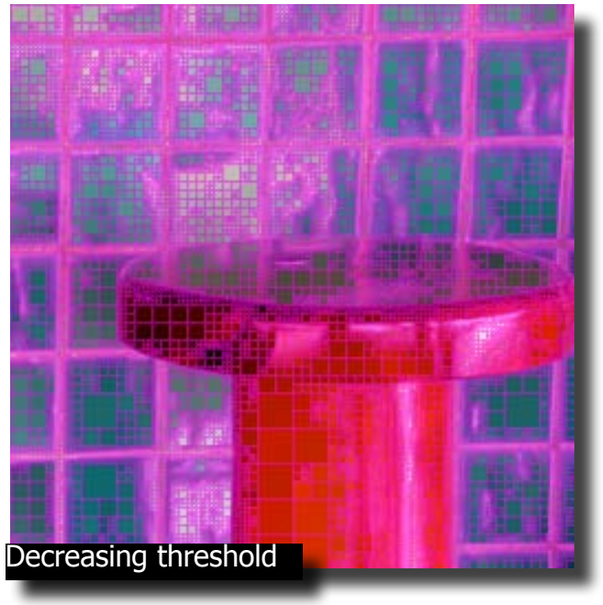
$$v' = (D - d)/D * v + d/(4*D) * (v_{top} + v_{bot} + v_{left} + v_{right})$$

D is the maximum offset from the centre in that node, and d is the offset of the pixel from the centre of the node. The node values are denoted by v, v_top, etc. This interpolation procedure results in some anomalies near the edges, but is fine for many applications.

Multi-channel Quadtrees:

If we represent force fields or images with quadtrees, we store a vector value in every node. For images, we store the RGB vector; for force fields, the xyz force components. When the channels are not highly correlated, we can often get better savings by using separate quadtrees for every channel. In images, this is not generally the case, and using separate quadtrees will be very wasteful. Artefacts are also introduced that are very noticeable in images. In force fields, the up-down dimension is often not correlated with the other two dimensions, and significant savings can be made by using one quadtree for the up-down dimension, and another for the other two dimensions.

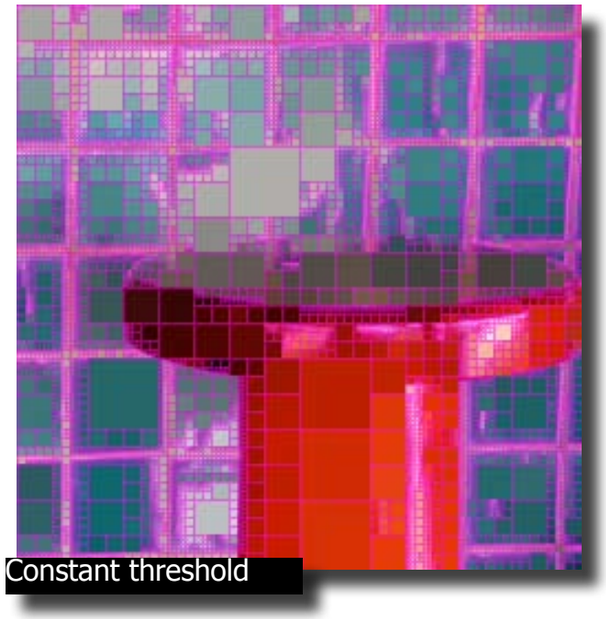




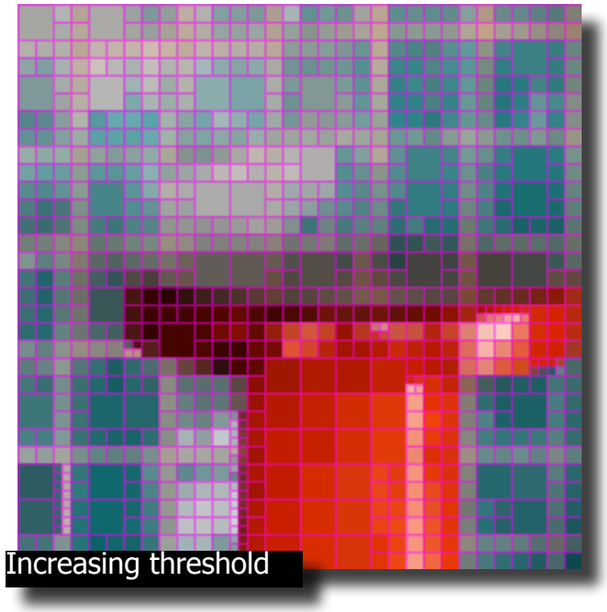
Decreasing threshold

Changing Threshold:

Instead of using a fixed threshold, you can use a threshold that depends on the depth of the tree where you are processing. This can be used to increase / decrease the uniformity of the node sizes.



Constant threshold



Increasing threshold

Gradient domain quadtrees:

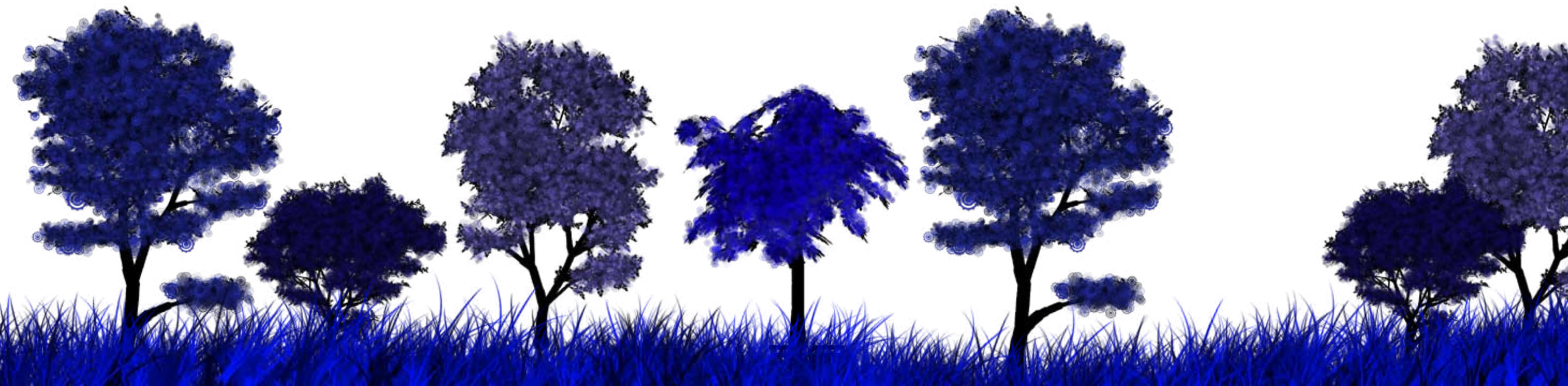
In certain circumstances you need only differences between pixels, and not their actual values, for example, when using force fields. As a body moves through the field, you need only update the value with the new difference. If, in addition, your data is characterised by lots of smoothly changing regions, and edges are not extremely important, you might get better results if you do your processing in the gradient domain: essentially working with the differences between pixels, instead of working with their actual values. It is convenient to start with two grids, one for vertical differences, and one for horizontal differences. Each of these two grids are then stored in a quad tree. The formulas that can be used (when v(x, y) represent the value at indices x and y in the original grid) are:

v'(x, y) = v(x + 1, y) – v(x, y) (horizontal)
v'(x, y) = v(x, y + 1) – v(x, y) (vertical)

You will need to store some extra data to be able to reconstruct the original grid (usually the first row and first column). You might also store a very low resolution quadtree of the actual values, to calibrate every once in a while. In images, the artefacts are very noticeable; other applications are much more forgiving.

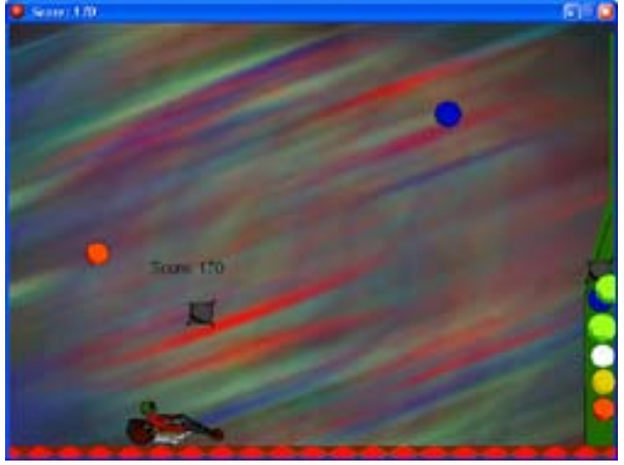
Download:

You can download code that implements some of the ideas from: http://code-spot.co.za/2008/11/15/quadtrees/



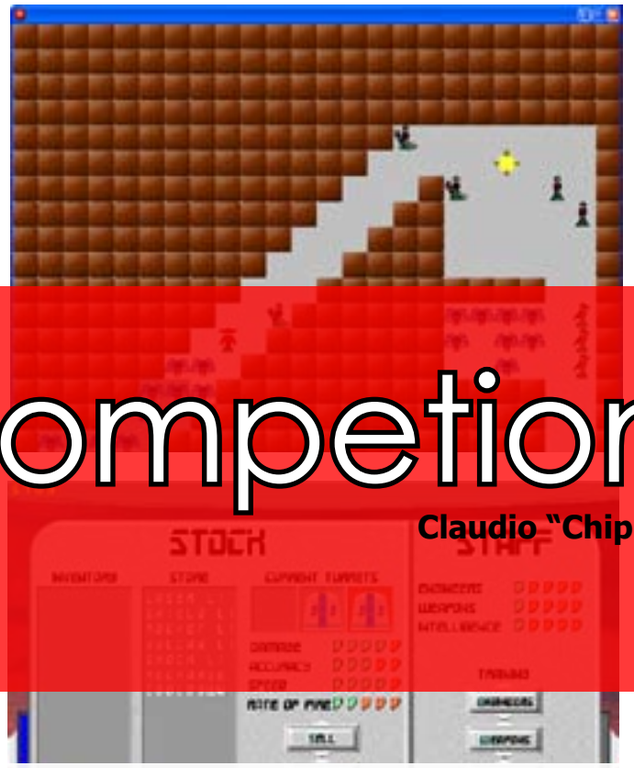
Object Pascal has a lot to offer...

www.PascalGameDevelopment.com



GL

Each Comp has a lesson to be learned
Look out for the Game Development Lesson from each one!



A Retrospective glance at Game.Dev Competions

Part 2

Claudio "Chippit" de Sa

The Game.Dev Comps have evolved considerably since they started, with sponsors and prizes being obtained, and more experienced (and larger) community facilitating the creation of even more advanced games. In part 2, we take a look at all the competitions held since August 2006; 10 competitions over 2 years.

Comp11 Tutorials

Comp 11 was unlike the other competitions held to date, and still stands as the only competition that didn't bid for the creation of an actual game. Instead, comp 11 sought to create a succinct tutorial describing the technique, effect, skill or system that developers had learned and used during the creation of their games. This change was justified two-fold: firstly, it served as a way for developers to share tricks and tips; secondly, it sought to correct the post-rAge slump that threatened to throw off the regular 2 month schedule that the competitions had been maintaining. Being little more than a warm-up for the return of the 'regular' format in Competition 12, this comp only saw 2 entries.

Comp12 Single-PC multiplayer

Following immediately after Comp 11, without the customary month-long gap, Comp 12 tasked developers with the creation of a multiplayer game that is playable on a single system. No other constraints were imposed so entries varied from real-time co-operative entries with a split mouse/keyboard control scheme, to turn-based tactical offerings. What made this Comp unique was the introduction of additional players into the design equation. Past games had dealt with single players only, so the new challenge here was to create a game that was fun for all parties involved. This required some new, creative thinking on the part of the developers, but precipitated interesting and fun results. In true Christmas spirit, this competition had a modest but tangible prize up for grabs, sponsored by none other than Danny "dislekcia" Day himself.

Comp13 Polish Mk 2.0

Not unlike Comp 6, this competition once again challenged developers to look back on and improve one of their past titles, with the aim of producing games of retail quality. The twist this time around was that the title to be polished was not a choice of the developer himself, but rather a result of a collective vote by the community. The competition sought to teach the value of using and analysing feedback from players and using it to improve the offering. Additionally, the competition elegantly introduced the Pareto, or 80-20, principle; that is, the old maxim claiming that 20% of your project takes 80% of the time. This is especially true for the final polish stages of gaming, where much time is spent play-testing, tweaking, then testing some more. This grueling, arduous task is often too much for developers who lose interest and abandon their projects when it is no longer fun for them to play, as is inevitable when you're playing a game to death. However, this essential step is often the difference between a diamond in the rough and a retail hit.

Comp14 Demos

This was another unique Comp, once again asking not for a game, but for the demonstration of tricks; graphical or otherwise. The format was in the vein of demo-scene offerings, where each entry was designated to be a small application showcasing a piece of eye-candy. While the results were never judged, there were many small entries made by developers who had used the opportunity to experiment with effects or techniques where they would usually dedicate their time to tweaking gameplay. While this may have appeared a little counterintuitive to the Game.Dev ideal of gameplay above graphics, previous competitions had highlighted that presentation and polish affect critical reception in an efficacious manner. As such, the competition sought to develop skills and tools which may be used in future productions.

Comp15 Guerilla learning

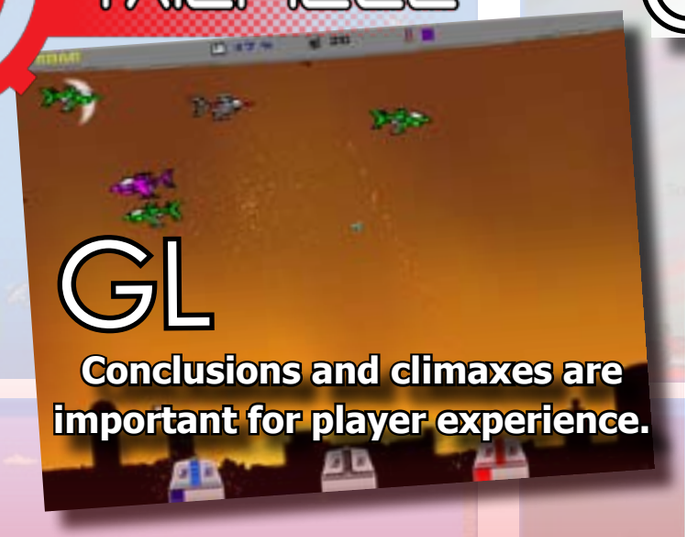
Comp 15 was another of those milestone competitions. It featured a huge prize sponsored by local education initiative Mindset Learn (http://www.mindset.co.za/learn/default.asp). The competition challenged the community to compete for a share of the R10 000 prize pool by creating a guerilla learning experience for players – that is, a game that can teach players valuable lessons by integrating them into the game itself, rather than grinding the learning experience through reward and punishment, as is typical for 'Edutainment'. The games that resulted were not edutainment in any way, but rather interesting and fun games that taught skills by exposing the player to their concepts as part of the dynamic itself. As with the last sponsored competition, Comp 15 ran over two months and saw a huge response from the community with 15 hopefuls churning out entries for the Mindset judges to evaluate. Evil_Toaster, winner of the previous sponsored competition, swarmed in and took this one with his title, Cartesian Chaos, and subsequently received publishing interest with Mindset. The competition succeeded in highlighting the capacity that videogames hold for learning experiences and will hopefully contribute to a change in the general sentiment regarding educational games, both for consumers and developers.

Comp16 Text only

Comp 16 went back and borrowed some ideas from Comp 9, once again limiting players by certain rules in an attempt to broaden the results. Things were slightly less restrictive this time around, with only a single constraint imposed: that all graphics need to be entirely composed of, or derived from text. Again, like Comp 9, this was an attempt to focus entrants on gameplay without needing to worry about graphical presentation. As it turned out, this restriction was better received than the Comp 9 incarnation, with ten extremely varied entries being submitted, those ranging from a real-time enemy-blaster, to a rhythm game, to a text adventure. All of these had well-developed gameplay as a result of limited graphical focus; succeeding in demonstrating that graphics are only needed as a catalyst to solid game dynamics, not a substitute.

TAILPIECE

Comp18 Lost Garden



In a move against the current generation's obsession with sandbox gameplay and supposedly infinite playtime, this competition called for games that would end – in ten minutes. Comp 17 sought to highlight the importance of cleverly planned events to orchestrate emotion and define the experience. Pacing is essential to elicit the required player responses during the course of the game. This Comp's fixed timeframe means that it was incredibly important to achieve balanced and carefully planned gaming experiences, for the purposes of the competition. As such, this turned out to be one of the most interesting Comps in Game.Dev history, along with Comp 19.

If you're a regular reader, you'll already be familiar with Lost Garden, the amazing game design blog run by Danc; and you'll be familiar with his prototyping challenges. Comp 18 released Game.Dev developers on their choice of three prototype challenges from Lost Garden: Playing with your Peas, SpaceCute and CuteGod. This means that developers were given fixed frameworks and designs, and challenged to make them fun. In effect, this highlighted the fact that ideas are cheap; quality games come not from good ideas, but the combination of an idea and a solid implementation of that idea.

Comp 19 was, quite possibly, the greatest exercise for the Game.Dev members design muscles. It challenged the familiar death concept in games and urged developers to take the current axioms regarding it as a game dynamic and turn them upside down. A lot of innovation in gaming is rooted in the simple reexamination of systems that were previously taken for granted. This competition sought to seed such innovation by encouraging players to change death from a penalty into a critical game mechanic.

Comp17 Ending

Comp19 Death

Comp20 No Text

To mirror the theme from Comp 16, Comp 20 took that concept and flipped it around, requiring developers to create a game that had no text whatsoever. This meant that language couldn't be relied on to convey information such as objectives, rewards or controls, and these must be afforded to the player in other manners. Developers were encouraged to experiment with other ways to reward correct behavior, like using subtle audio or graphical cues to signify correct player responses. The competition turned out some interesting titles, some succeeding quite respectably in telling stories and conveying objectives without the use of any language whatsoever.

That's it folks!

This concludes the Game.Dev competition series. If you're interested in entering one of the competitions and you reside in South Africa, keep an eye out on the Game.Dev website (www.gamedotdev.co.za) and on the forums for competition announcements. By the time you read this, Comp 21 should be well underway. We look forward to seeing new faces.

SHEEP COUNT:

36 SHEEEEEEEEEEP!

ONLINE

DEVMAG

CREATE • DEVELOP • EXPERIENCE

www.devmag.org.za