Issue 24

August 2008

DEV.MAG

CREATE • DEVELOP

INSIDE:

Penny Arcade Adventures O Design vs Programming O How to make your own sounds for games O All you need to know about the Google App Engine O Interview with German Indie developer, Irrgheist O News + Reviews + Other stuff too!

CONTENTS

ED.PAGE

EDITOR
Claudio "Chippit" de Sa

DEPUTY EDITOR
James "Nighttimehornets" Etherington-Smith

DESIGNER
Quinton "Q-Man" Bronkhorst

CONTRIBUTORS
Rodain "Nandrew" Joubert
Simon "Tr00jg" de la Rouviere
Ricky "Insomniac" Abell
William "Cairnswm" Cairns
Danny "Dislekcia" Day
Andre "Fengol" Odendaal
Luke "Coolhand" Lamothe
Rishal "UntouchableOne" Hurbans
Gareth "Gazza_N" Wilcock
Sven "FuzzYspo0N" Bergstrom
Kyle "SkinkLizzard" van Duffelen
Chris "LionsInnards" Dudley
Herman Tulleken

WEBSITE ADMIN
Robbie "Squid" Fraser

WEBSITE
www.devmag.org.za

EMAIL
devmag@gmail.com

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

I've always wanted to write my very own story; this seems like as good a place as any to start. Okay, so once upon a time there lived a pretty little girl. Her name was Sarah.

It's been quite a juggling act this month; the post-holiday slump, numerous unforeseen obligations of some of our staff resulting in a bit of a slanted development cycle, plus the usual post-Game.Dev competition rush have all contributed to the veritably mad dash needed to get the magazine done on time. My own DreamBuildPlay entry currently taunts me from my taskbar as I write this, but it is actually down to a heroic effort from our designer that everything was ready to release at all – he really does deserve that candy.

Perhaps the major cause of our troubles this month was the push to get some strong content in for this issue. We sport a nice shiny review of **Penny Arcade Adventures Episode 1 – On the Rain-Slick Precipice of Darkness** (and I'll be damned if I type out that whole name again) headlining our reviews this month, with a smattering of other GarageGames-, and more specifically, **Torque** Engine-related content scattered throughout our virtual pages.

We also sat down with Irrgheist, the creators of H-Craft Championship, to hear their thoughts with particular focus on H-Craft Championship, their first game release and a title we've mentioned before. Additionally, we're starting off an excellent new multi-part tutorial with an introduction in the use of Google's web application engine for game creation. Keep a look out for the continuation of that one next month, since it promises to be an enlightening piece indeed.

You may also have noticed that the design and tutorial sections have vanished. In an effort to eliminate some ambiguity and the occasional overlap between the sections, we've decided to merge the offending sections into a new, larger, magazine division from this issue onwards. The new section – named Development – will contain all content relating to game development in any way, whether it is to share insight, tools or guides.

Finally, I'm proud to announce that Game.Dev will have a fairly strong presence at **DreamBuildPlay** this year, with at least two local teams currently at the XNA-shaped grindstone churning out their entries. You can look forward to a special piece nearer to the close of the competition detailing each team's personal experiences and thoughts.

And that's about all I have to say this month. We put a lot of effort into this issue, so if you're still here: get reading, already!

~ Claudio, Editor

Game Design Challenge.

http://www.gamecareerguide.com/features/

GameCareerGuide.com's Game Design Challenge is a weekly series of activities aimed at changing the way game designers tackle design problems and tries to encourage positive and proper game design methods. Every week they present a question and challenge people to send in their game design to solve a certain problem; their chosen winners are then revealed the end of that week. It's a fascinating experiment that occasionally results in really innovative solutions. Have a look at some of the past challenges and solutions, then give it a try yourself.

Xbox Live Community Games.

http://creators.xna.com/en-us/XboxLIVECommunityGames

Microsoft is finally getting ready to roll out XNA's biggest gun–community release on the XBL Marketplace. Soon, any game created in XNA will have the potential to be sold on the Live Marketplace together with other XBLA titles, earning the developer up to 70% of the revenue generated from its sale. Pricing for these titles will be determined by the developer, in the range of 200 to 800 Microsoft points, roughly $2.50 - $10.00. If you still don't have a Creator's Club membership now, you have no excuse.

Gamecues Launched

http://www.gamecues.com/

Sound effects are some of the trickiest yet most important factors in a game, something sadly overlooked far too often in the indie scene. This is where Gamecues steps in. Gamecues is a sound library specifically dedicated to game developers, featuring a veritable horde of high quality sound effects tailored specifically to the game industry. It's not free–as nothing of this calibre ever is–but a resource of such quality is invaluable.

Nonoba's Multiplayer API Kick Off is up.

http://nonoba.com/developers/contests/multiplayerapi-kickoff

Nonoba have launched a new competition to promote their multiplayer flash API, challenging developers to create a great game using their API and offering up to $20 000 in prizes. The grand prize, a clean $15 000, is a particularly good incentive to create something cool and fun. Just remember, you must use the Nonoba API, and your game must be a multiplayer title

Irrgheist is a small independent developer based in Germany, founded in 2006 by Philipp Lossack and Michael Zeilfelder. Their first release, H-Craft Championship (a sci-fi racer in a similar vein to the Wipeout series), was well received by indie circles for its smoothness. Dev.Mag writer Sven "FuzzYspo0N" Bergstrom sat down with Philipp and Michael and found out what they had to say.

Sven "FuzzYspo0N" Bergstrom

SO...

"EVEN THE SMALLEST TEAM NEEDS A CLEAR STRUCTURE WITH CLEAR AREAS OF RESPONSIBILITY"

Dev.Mag: In terms of development, what got you guys started?

Michael: Going independent is the only way I see to create the virtual worlds that interest me. You can't really work on your dreams while working for another company. Also, after switching companies a few times I found it increasingly frustrating to have to start from scratch every time afterwards, because none of the code I wrote belonged to me.

Philipp: Yes, I had similar reasons. On the huge teams you need them for today`s big games, a single person`s duties are very specific. Personally I prefer to see the whole frame of a small project, rather than a very small section of a huge one.

Dev.Mag: What made you choose the H-Craft game as a first title?

Michael: It was mainly Philipp's idea to do that. I liked playing such racers and it seemed like a good project to build up a solid technology base. For me it was very important to build up a toolchain which I could use for further games.

Philipp: Well, amongst other genres, I always loved Sci-Fi racers. It also seemed to be a doable task for a small team. And I finally got the chance to work on that vehicle design I had in mind for years.

Dev.Mag: What challenges were you faced with when developing this title?

Michael: The most underestimated task had been the GUI. You don't expect that for a racer, but I think this took at least a third of the whole development time. Just think about each dialog being reworked a few times and you see that even 20-30 dialogs will very soon cost you months.

Philipp: Don`t ever talk about GUI...

Dev.Mag: What did you learn about the creation of games through the development of H-Craft?

Michael: Lots of minor stuff; it's been some time since I last programmed a game all by myself and so I used completely new technology for nearly everything. For example, it was nice to see how little overhead is needed nowadays to write platform independent games when you select your libs [libraries] carefully.

Philipp: Even the smallest team needs a clear structure with clear areas of responsibility. At times we mixed things up a bit, and it didn`t help the progression.

Michael: I actually think that good communication is a more important key point. We used IM, e-mail, a forum and some folders to exchange documents, but that was still not enough. Stuff got often mentioned offhandedly and forgotten again. I hope using a Wiki next time will help somewhat.

Dev.Mag: If you could change things about H-Craft what would you have done differently?

Michael: I would start earlier to give out demos for testing. Also by now I wish that I had spent another month on it to code network support, but well, it seemed impossible to do that back then. I still hope we can fix that.
Philipp: It`s a matter of time, but a proper prototype would have helped to avoid errors. Basically we just started working on the game hoping the ideas on paper will work out. Fortunately most of them did, but some did not.

Dev.Mag: Can we expect updates or a sequel to this great game?

Michael: Maybe. I hope so. Let's say the chances are above 50%.

Dev.Mag: What does the future hold for the team in terms of indie game development?

Michael: I don't think we will work again in the same constellation like in the first game. Financing yourself for more than a year is very hard and we can't do that again right now. But there are other ways to continue and we're discussing that a lot. I do currently work as freelancer, which allows me to continue working with the Irrlicht engine and improving my toolchain.

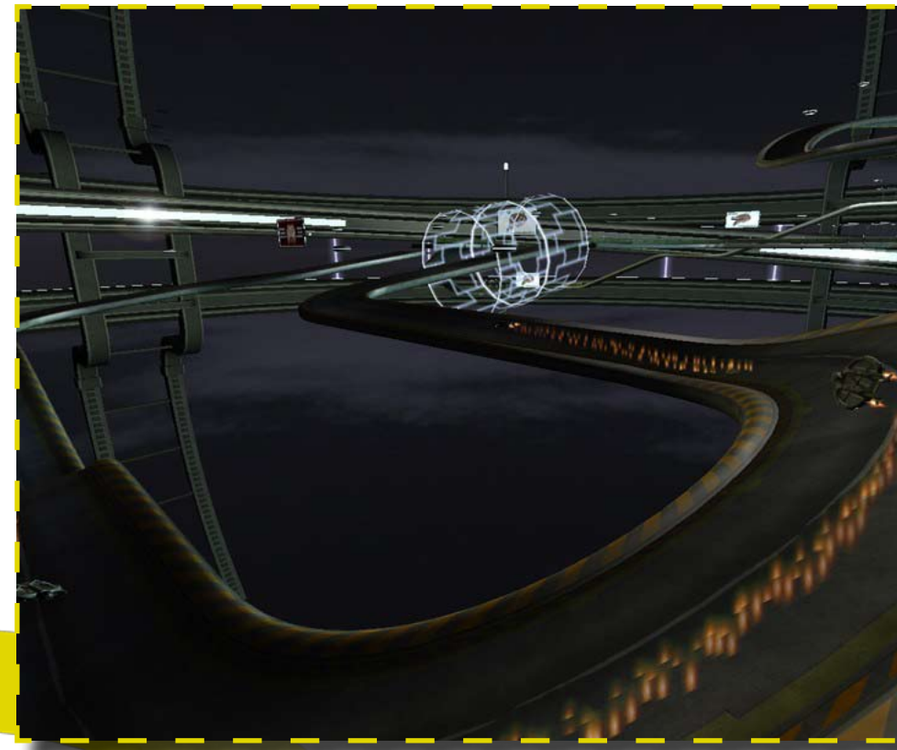
Philipp: Time will tell. Right now we discuss and try out various things, but no one really knows where it leads to.

Dev.Mag: If you had any tips for indie developers what would it be?

Michael: If you get offered a distribution contract then take it. Also use open source whereever you can, working with a community is very rewarding and I don't think you get that so much with any closed source stuff.

Philipp: Michael's advice has a history for us; it`s good advice! I`d like to add that indie developers shouldn`t entirely ignore the visuals. There are great small games out there, but many of them are obviously made by pure coder teams. That`s fine, but in my opinion they would profit from some sort of art direction. I am sure there are enough artists around who would be happy to join a team.

"THE MOST UNDERESTIMATED TASK HAD BEEN THE GUI."



TOOLCHAIN

My main development system was **Kanotix**, which is a Debian based GNU/Linux distribution.
For a compiler I used **GCC** (GNU Compiler Collection).
For porting the game to Windows I used **MinGW** (Minimalist GNU for Windows).
My IDE (Integrated Development Environment) was mostly **Code::Blocks**, but I also work a lot on the console and with scripts.
When I need some programmer art my favourite tools are **Wings3D** and certainly **Gimp**.

LIBRARIES

Irrlicht for 3D GUI

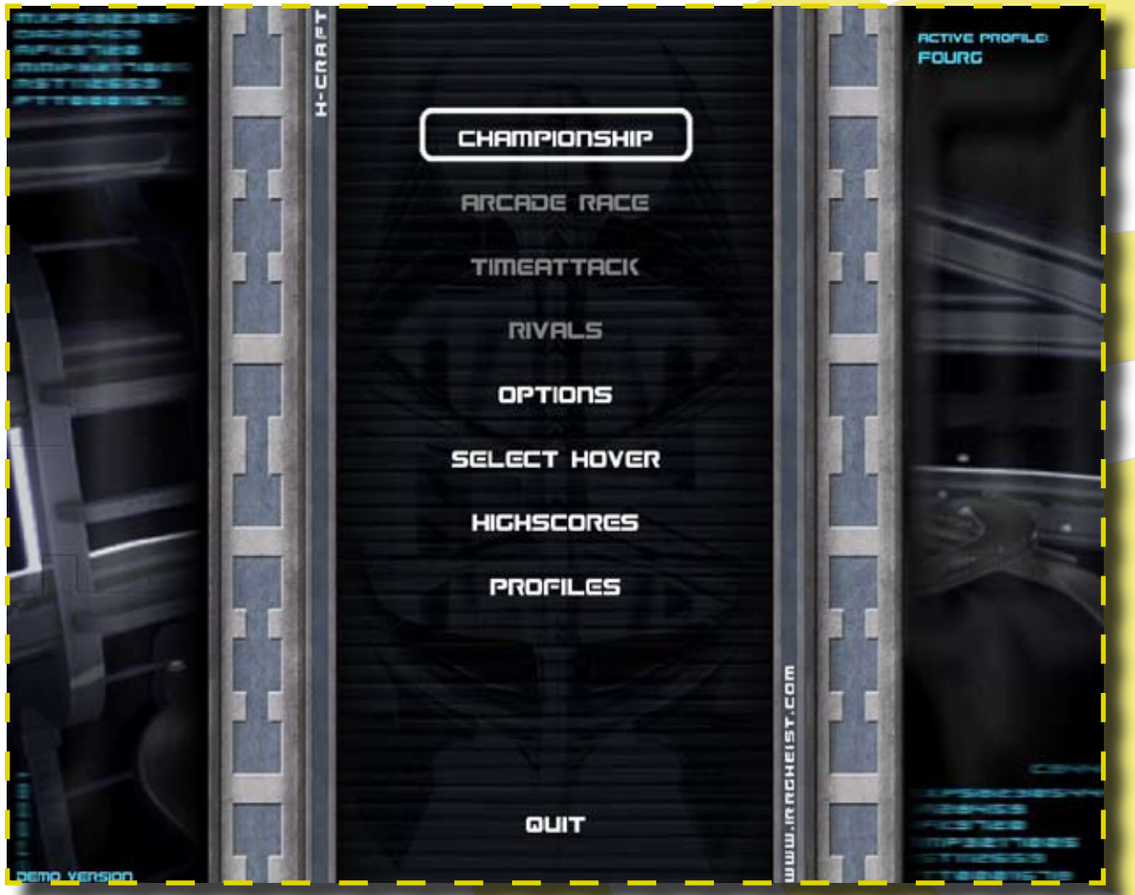
ALUT and **OpenAL** for sound.

Ogg **Vorbis** for *.ogg file support.

SDL (Simple Directmedia Layer) for joystick input.

FreeType for truetype fonts.

libcrypto++ for creating a distribution key system.



ON THE RAIN-SLICK PRECIPICE OF DARKNESS

EPISODE ONE

In most games, the opening cut scene generally consists of hints of a dark secret, a terrible evil that must be vanquished lest it destroy the world. This first episode of the Penny Arcade Adventures series of episodic games is no different. Suggestions of dark plots and nefarious deeds in the shadows of New Arcadia make you question whether this game actually has anything to do with the popular gaming humour web comic at all. Then your character's house gets crushed by a giant robotic fruit juicer, and the game's tone is set firmly to what you'd expect.

Precipice (to shorten the full title somewhat) was developed by indie developer Hothead games, in strong collaboration with the creators of Penny Arcade. Using the Torque engine, they were able to give the game a distinct comic-book feel, making liberal use of cell-shading to give it as close a resemblance to the web comic's art style as possible. They succeed quite admirably. Even the player character, whose appearance is fully customizable, looks as if she or he has stepped straight out of the comic.

As for the writing, let it be stated outright: the game is typically Penny Arcade in tone. While their roles are adapted to suit the game's story, almost all of the characters are straight from the virtual pages of the web comic, and Penny Arcade references and in-jokes abound as you trek through the streets of New Arcadia ("Wang's Chinese Restaurant," anyone?). That's not to say that it's completely inaccessible to those who aren't familiar with Penny Arcade, but fans will definitely get the most from it in terms of the humour. Be warned; if you aren't into more absurd humour or internet/gaming-culture, it's wholly possible that you won't find the game (or the comic) funny at all.

In terms of gameplay, Precipice of Darkness is an adventure game/RPG hybrid in the same vein as the cult favourite Anachronox. The game is essentially composed of two distinct play modes: exploration and combat. In exploration mode, you point and click around the game's four main locations, speaking to characters, filling your combat inventory by smashing crates and other possible containers of goods, and solving fun (but not particularly difficult) puzzles. Don't despair, action fans! A huge chunk of the game is spent in turn-based combat with the many, many different enemies standing in your way, and it's handled in a very interesting way.

"THE GAME IS TYPICALLY PENNY ARCADE IN TONE."

The combat mechanic is where the RPG element of the game kicks in, but it's a little different from the RPG's you may have played in the past. Firstly, combat is completely separated from the exploration element–characters begin each battle in perfect condition, so there's no need to heal characters or dispel status effects between battles. Secondly, it has a rather novel take on traditional turn-based combat. Each character has three fixed actions that they can perform while in a battle: use of inventory items collected during exploration, a standard attack, and a special attack. To use these actions, you must allow the characters to gain "initiative" for them, much like the Active Battle Time system in Final Fantasy. Unlike FF however, your attack options charge sequentially. Your Inventory initiative needs to top up before your Basic Attack initiative starts charging, and your Basic Attack needs to be full before your Special Attack can charge. Using any of a character's attack options, regardless of level, brings the total accumulated initiative for that character back to zero. Special Attacks require you to complete a small character-specific mini-game to determine the total damage dealt, which becomes progressively tougher the stronger the character gets. This adds quite a bit of strategy to the battles: do you heal a character who is close to death and lose all your initiative? Or do you wait a little longer and attempt to deal a devastating final blow to your enemy with a perfectly executed special attack? Death blows could net you a spectacular shower of gore and an Overkill bonus; a permanent increase to your weapon damage. In addition, if multiple characters have a special attack charged, you can initiate a "team up" attack which deals massive damage while circumventing the Special Attack mini-game entirely. Add in the backup from three support characters (who can occasionally be summoned into combat to hit enemies with their special support attacks), and the ability to block enemy attacks by hitting the spacebar at the right time (sometimes resulting in a free counterattack from the defending character), and you have a rather fast-paced and dynamic combat system that manages to stay entertaining and challenging all through the game.

There's very little to criticize about Precipice, but it isn't perfect. The game is stable, except for crashes encountered while attempting to use a certain support character during the final boss battle. One might find the game a bit too short, which should be expected given its episodic nature, but it may leave the player feeling a little unfulfilled in terms of play time. Nonetheless, the game was highly entertaining throughout and never felt stretched out or padded, so its relative brevity can be forgiven. Special mention must be made of the excellent musical score, which might have you humming a lot of the combat music a week after completing the game.

For those who enjoy Penny Arcade and Adventure/RPG games, Precipice comes highly recommended. For those who don't, it still comes highly recommended, but it is suggested that you play the demo to determine if it resonates with you first. Bring on Episode 2!

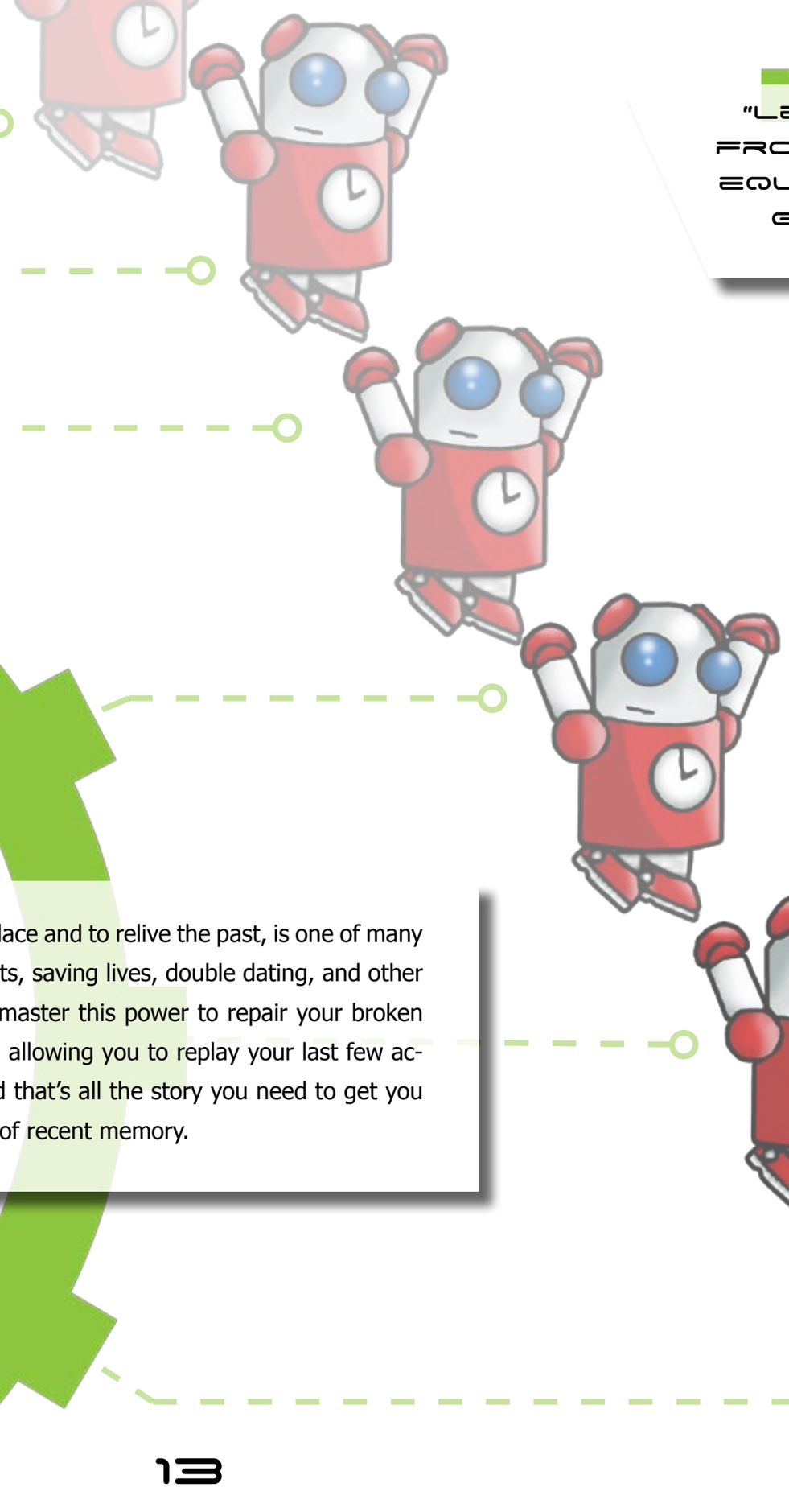
Gareth "Gazza_N" Wilcock

"YOU HAVE A RATHER FAST-PACED AND DYNAMIC COMBAT SYSTEM."

Chronotron

The ability to go back in time, to be in more than one place and to relive the past, is one of many a childhood fantasy. We dreamed of stopping accidents, saving lives, double dating, and other honourable things. Well in Chronotron, you have to master this power to repair your broken time machine, which has become stuck in loop mode, allowing you to replay your last few actions to aid you in finding all of its missing pieces–and that's all the story you need to get you started in one of the most original puzzle/platformers of recent memory.

"LEVELS RANGE FROM BASIC TO THE EQUIVALENT OF JUGGLING NAILS WITH YOUR EYES."

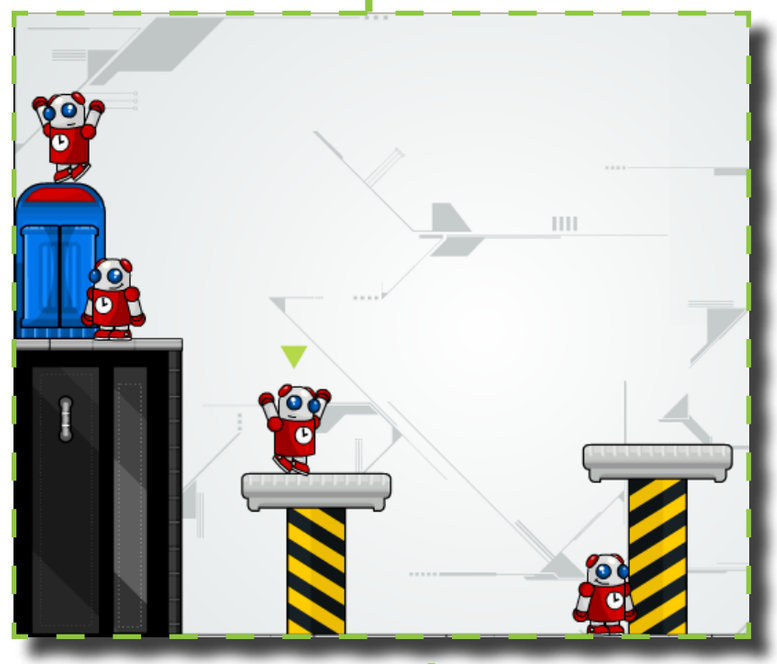


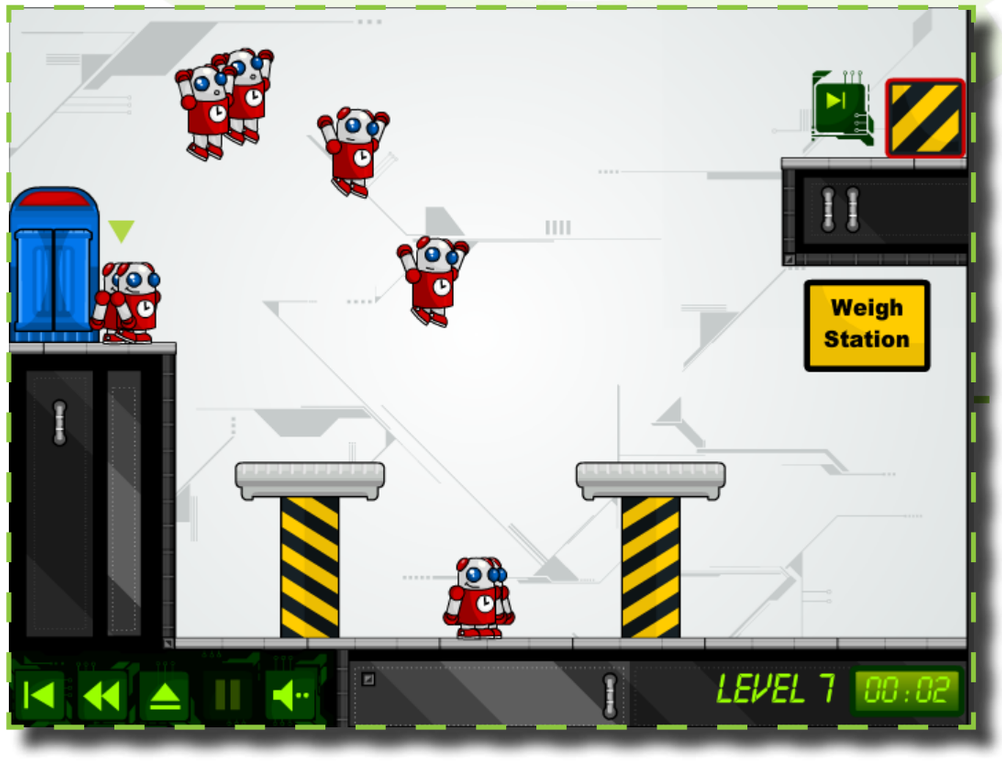
You are greeted by a nice title screen when the game starts up. Clear, detailed graphics display a cute little robot with bug eyes and a clock on his chest. He is your history-faring counterpart on this adventure. The graphics are crisp, nicely detailed and attractively cell shaded. The animations are a bit basic, but they get the job done. They are forgiven and forgotten, though, when you get to the highlight of the game; the gameplay.

The game places you in a level with just your avatar, the time machine and a few levers and obstacles. Each stage requires a piece of the machine to be retrieved, but it will require several trips to do it. For example, if the piece is on a ledge with a platform beneath, you will need to stand on the switch to raise it for a bit, and then return to the machine and rewind time. When you step out again, your past self will go on to do everything you did on your last instance, while you are free to stand on the rising platform to retrieve the objective. Some basic jumping is required, but this usually focuses on accuracy and timed leaps, rather than monotonous hopping from ledge to ledge. These levels range from basic to the equivalent of juggling nails with your eyes.

The most impressive thing is the way the game consistently produces difficult, but not illogical, puzzles that challenge you to master timing, platforming accuracy and move-plotting. This is truly a masterpiece, a fun, quirky and detailed puzzler (with suitably funky electro music) that deserves to be given a try.

Chris "LionsInnards" Dudley





Garage Games, the creators of the popular and robust Torque Game Builder have released a new and exciting platform for the gaming industry dubbed "Instant Action." If only faced with the name, one would be duped into thinking it is a simple casual flash portal. It shares similarities to a flash portal (such as Miniclip), but it's more like its big brother.

INSTANT ACTION

"SERVICES SUCH AS INSTANT ACTION WILL NO DOUBT BECOME SOMETHING TO LOOK OUT FOR IN THE FUTURE."

Instant Action provides 3D gaming action in your browser. It brings together the social and accessible nature of the web and the competitiveness of 3D gaming. At the moment the games on offer are a bit sparse, but enough to satisfy any gamer. The most popular game at the moment on Instant Action that Garage Games is flaunting is Fallen Empires: Legions. It is an FPS that is similar in style and play to Tribes. The larger maps have sprawling landscapes. Another popular game is called Rokkitball, a futuristic sport that is a cross between football, rocket launchers and magno-beams and it supports up to 8 players.

The other games on offer include Marble Blast Online, a firm casual favourite that has you rolling a marble through puzzles. Think Tanks is a neat strategy game involving tanks. An arcade shooter in the form of ZAP (Zero All Productivity) is also present alongside an action puzzler called Cyclomite. Amongst the games that are still in development is a racing game, a hover tank combat game, an aerial dogfight game and a frenetic strategy game. As you can see, there are games on Instant Action that will satisfy any gamer's needs!

The market that Garage Games are aiming for is quite wide and will no doubt appeal to a lot of people. Hardcore gamers who want a more accessible and inexpensive gaming experience, international gamers with no access to a console and former core gamers who now do not really have the time and money to invest in serious gaming will find a home at Instant Action.

The interface and presentation of Instant Action is done really well. It is very intuitive and easy to use, allowing you to jump into a game very quickly. The current social networking features include adding friends and having a real-time chat with them. There are loads of expansion packs and additional doodads that can be bought. It includes new avatars, new models for games, map packs and so forth. At the moment the only problem with something like Instant Action is the bandwidth usage. People in countries with high connection costs and small data caps (*cough* South Africa *cough*) will definitely need to steer clear of Instant Action.

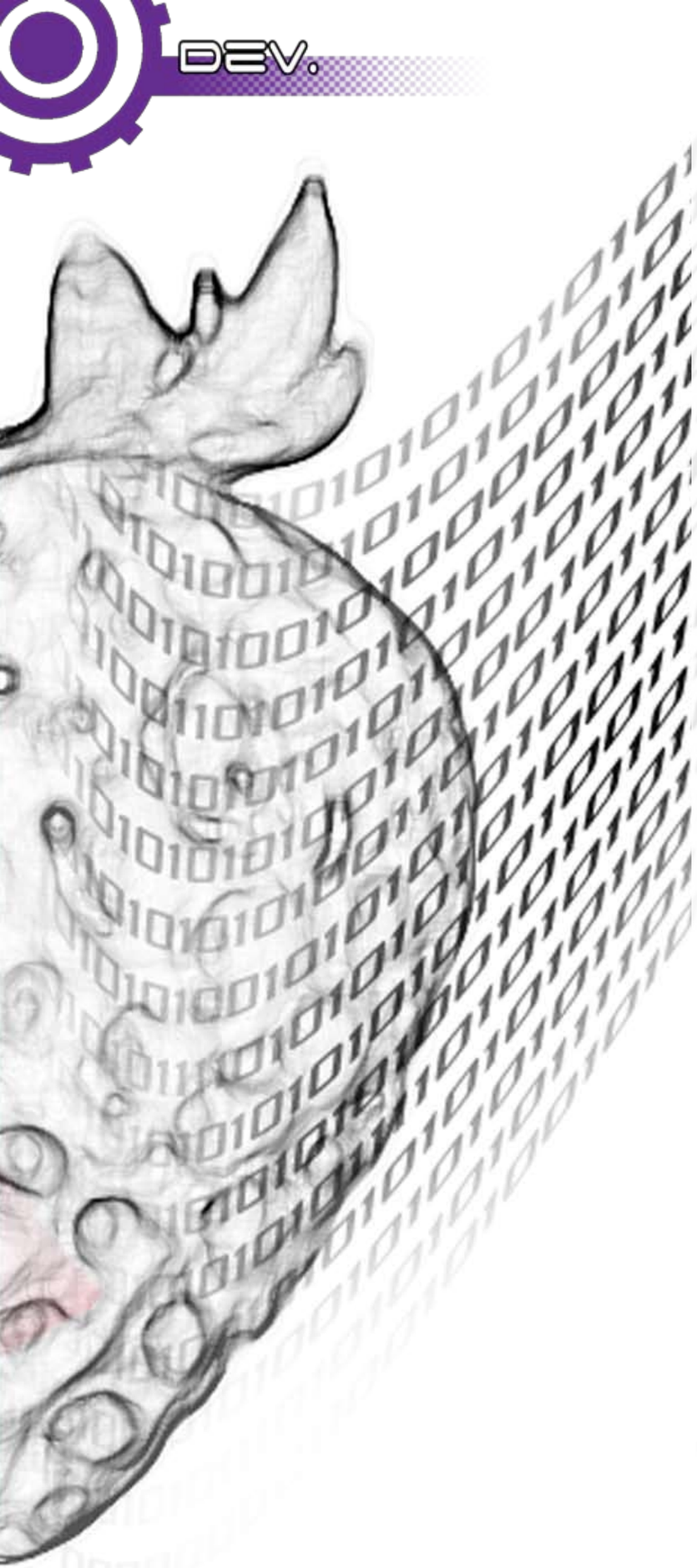
Instant Action is still in its early stages of development; the Open Beta started early in 2008 and at the time of writing, the service is still in the beta stage. With the continuing trend of falling broadband costs coupled with increasing broadband penetration, services such as Instant Action will no doubt become something to look out for in the future. Heck, who wouldn't love instant 3D action in their browser?

Simon "Tr00jg" de la Rouviere

Design vs. Programming

Rodain "Nandrew" Joubert

Have you ever slaved away at a new project for ages, using every trick that you know to make it visually and technically stunning, only to have it fall flat when you present it to those philistines that you want to test the game on? It's a heartbreaking experience, and there's generally nothing left but for you to seek out some equally avid programmers who will give the congratulations that you deserve. Even then, the victory may be hollowed out by their reluctance to play your game after they've finished admiring your frame rate.



It's a tragic truth in the game development industry that there are way too many budding developers who, at early age, find themselves with a innate gift for programming, only to develop this talent without any due regard for the accompanying skill of game design. While there are a multitude of coders out there whose dedication and expertise is truly awe-inspiring (demo groups, in particular, produce some fascinating examples of coding acrobatics), there is also a sad legacy of games out there that are technically excellent, but fail miserably on the level of raw fun.

The mindset that leads people to this problem in the first place is the same mindset which prevents developers from potentially erasing it. Getting stuck in the "code trap" is frightfully easy, and escaping can be difficult without an external guide. Sometimes, simply changing your thought patterns can instantly see you making much better games; it's just a matter of taking that step.

The number one obstacle is short-term selfishness. If you're making a game with the primary motive of indulging yourself, writing some fancy routines and stroking your own ego once you've pulled them off, then you'll receive your reward in full before the game has even reached the consumer. When players realise this, they'll switch off.

This isn't saying that selfishness is entirely negative: there is a correct way to congratulate yourself, but it entails creating a game aimed at the player–not the developer–and collecting the hard-earned praise afterwards for a game well crafted. Indulge the end user, and they'll return the favour later; this is what good game design is all about. Becoming a top designer, therefore, involves learning to predict what a player wants and finding a way to give it to them. People who don't invest effort in improving this skill run the risk of producing consistently poor games.

Game developers also need to be open to learning. This applies not only to their particular speciality but also to everything else that is related to their work. In an article entitled "Toward the future of game design," The Escapist quotes experienced game developers on the importance of possessing a wide range of skills. Richard Dansky of Red Storm Entertainment had this to say: "...take a little bit of everything. Learn some math and statistics; you'll need it for balancing. Learn to write, or your ideas will never make it out of documentation. Learn some psychology, so you can understand both your characters and your players. Learn a little economics, because on some level, pretty much every game is about resource management. Take some lit theory, so you can understand narrative and have a grounding in the great stories that underpin so many games. Take some history, because it does in fact repeat itself, often at 30fps. Learn some programming and some art, so you can talk to the other members of the team and more importantly, listen to them. In other words, make your base as broad as possible, because sooner or later it all comes into play, and you'll want as many arrows in your quiver as you can manage."

Specialising in a single development discipline (such as programming) means that you're severely restricting your arsenal if you're working by yourself. Although it's not necessary to become a grand master of aspects such as design, art and marketing, it helps immensely to be just a little knowledgeable in these disciplines. As Dansky points out, even being part of a team requires you to have some understanding of what your colleagues are up to.

An online article entitled A crash course in game design & production emphasises another cornerstone of game design: planning. The designer's job often begins long before the programmer types even a single line of code, a fact which is drilled into the reader from the start. The author mentions, by means of introduction, that "it's going to be a while (at least 6 weeks) before we write a single line of code."

One of the examples given in the document is the hypothetical game design process of creating the original Pac Man: while it's easy to sit down and say, "I want a yellow guy in a maze who eats dots and gets chased by monsters," and promptly begin coding, there's far more detail that needs to be considered before the game can be practically implemented. Questions include: how big is the maze? How fast do the monsters move? How fast does the player move? Are there any gameplay events that may change monster behaviour? How will we score? What should the controls be like?

Planning your game down to the last edible dot means that you can begin programming with the knowledge of exactly what you need to do, meaning that you'll avoid the problem of overdoing your code or, worse still, coding yourself into a corner. Using the above points, one can begin constructing a grossly simplified representation of the differing concerns that programmers and designers face:

The Designer's Questions:

What do I want to add to this game?

Why does the game need it?

How can I make it fit in with the rest of the game?

Will project/coding constraints permit me to add it?

Will the player enjoy it?

The Programmer's Questions:

What does the game design require of me?

What's the best way of implementing this?

Can I justify my code's importance to the design?

Am I leaving the code open for later design needs?

The designer's fourth question is naturally of considerable importance, and requires co-operation with the programmer and other team members. Almost all of the programmer's questions depend on a solid design scheme. Should a developer find themselves in the position of both designer and programmer, then the issue of communication becomes moot: however, they are still required to ask themselves both sets of questions in order to maintain a reliable project.

Reflect critically on your own game development thus far. What has improved in your projects over the time that you've been developing? Is it the quality, originality and entertainment value of the finished product, or perhaps an increased ability to complete what you've started? Chances are that you have the right ideas about game design and are well on your way to developing that bestseller one day.

If the technical level of your finished product has increased but your games consistently meet a poor reception, step back for a moment and ask yourself if you're following a good design philosophy. Try following the basic points outlined here. Get in some practice by entering design-related competitions such as those on Game Career Guide (http://www.gamecareerguide.com/), or read up on design articles both on the Internet and in Dev.Mag.

Remember: design isn't just something that game developers are trying to force on those hapless programmers who just want to get on with the coding. Good design practices are an important part of ANY major project, and even if you take your programming into a career that has nothing to do with game development, you're going to go a lot further if you know how to think like a designer.

Finally, this article isn't the be-all and end-all of good design practice. There are a few references to good design practice in here, but you'll probably be a lot better off checking out other Dev.Mag articles and online resources if you want to get more insight on techniques and design experience.

What you definitely should learn from all this chicken scratch is the fact that design IS a separate entity from programming and it IS important to figure out. Use this article as a springboard for readjusting your outlook when it comes to games, and remember to consider design whenever you embark on a game development venture. Once you get the attitude right, you'll discover that the knowledge–and the adoring fans–will soon follow.

MYTHBUSTING

MYTH: Players appreciate good programming.

Unless your players are themselves programmers or developers, it's unlikely that they're going to stop and directly appreciate the coding that has gone into the game they're playing. Certainly, a solid code base is important to reinforce a well designed game–in fact, players will be rather nastily reminded of the underlying program if they encounter a particularly severe bug–but throwing a few bloom effects into a sub-par game will rarely persuade anyone to give something a second shot. Putting some appropriate wizardry into a good, solid title is a much better idea. Don't program to impress players. Design to impress players, and then write the necessary code to back it up.

MYTH: Programming is independent of game design.

Ideally, programming should bend to the will of good design, though it's also true that design needs to respect the limitations of realistic programming. To put it succinctly: the two are bound to one another. Putting forth a design for the Most Unrealistically Awesome Game of Ever is going to fall flat on its face if the project is way beyond any logistical and technological feasibility. Similarly, the idea that you can start coding up your Most Uber Awesome Game of Ever without a well detailed plan of what it will ultimately need usually ends in grief, code rewriting and project cancellation. Your code needs to follow the direction and spirit of your game's design from the start, so that the code can deal with exactly what the game is likely to throw at it; anything more is a waste of effort, anything less will most likely cripple the project later.

MYTH: Design is easy.

As the Game Career Guide puts it: being a designer requires you to think like both a game designer and a player simultaneously. This can be a real headache, and while the designer may not be faced with the particular challenge of getting a game to work, they need to make it work in a way that is appealing and logical to a player. They need to take the constraints of available technology and the overall game style to make something that is new, interesting and well-balanced. It's not only the major gameplay decisions that have to be figured out: every corner of a level needs to have a purpose, as does every item placement, every enemy and every situational puzzle that the player may (or may not!) come across. The designer also needs to be comfortable when dealing with matters like statistics, balancing and game difficulty. With every single concern above, they need to ask themselves the same question: what would the player think? Or, more accurately: what would ANY player think?

MYTH: Designers are all about art and drawing.

Not so. As explained already, good game design goes beyond sketching cool monsters and locations to fit into the gaming universe. Designers need to justify a multitude of game development choices: it's all very well that you're faced with a spike-tailed gremlin, but where should the player face it? What resources should be at their disposal to help fight the gremlin? What purpose does the tail serve? Would another monster not do better? Does a gremlin even have a place in a World War 2 shooter? Design choices may be based around situations that the player barely gives any notice to at all: do we want five health potions present in the secret bonus room, or six? How injured is a player likely to be at this point? Would the advantage of discovering the secret room be considered too unfair? For that matter, how easy will it be for players to discover the secret room? Does it make sense in the game world? The questions just keep coming, and designers need to answer them all.

MYTH: If you can program well, you can design a good game.

Sadly, this is not strictly true. Game design is a job in its own right, in the same way that you can be a 3D artist or programmer. There's certainly nothing that stops you from being both a good programmer and a good designer, but many people overlook the fact that despite being something which is often referred to as a "soft learning field," good design comes with experience and effort. Like programming, people can enhance their design skills through practice, reading up on design articles and even entering Internet competitions. Starting a project with the aim of building your gameplay around that new lighting effect you learned–rather than the other way around–may well cripple the end result. That is unless, of course, your work incorporates light-based gameplay that stemmed from a carefully considered design choice!

IN CASE OF EMERGENCY
BREAK THE MOULD

The Need Continues...

Design Documents

Well here we go again: last time we had a look at how game design documents can benefit us; how they can help us with the overall process of game development; and perhaps provided enough motivation for us to use them for once. In the second installment we are going deeper, into the actual contents of a design document, specifically aimed at games. Software development at all levels should have design documents but games are a unique and interesting beast when it comes to completing a project. There are team dynamics that will never exist in other areas of software development; there are constant improvements and developments in the area of hardware and graphics capability; and of course, there is the competition. Sticking to your guns on a design is where the focus should be set to ensure that games are made, instead of unfulfilled dreams. More important than getting through a design document is having one of value. This is what this article aims toward; providing some clarity on the concept of the design of a game.

Each section will refer to a section in the supplemental example design document which will be available on the dev. mag web site. Again, there are no rules when it comes to the creation of a document covering the design of a game; this article aims only to lay some guidelines in the current game dev world.

Sven "FuzzYspo0N" Bergstrom

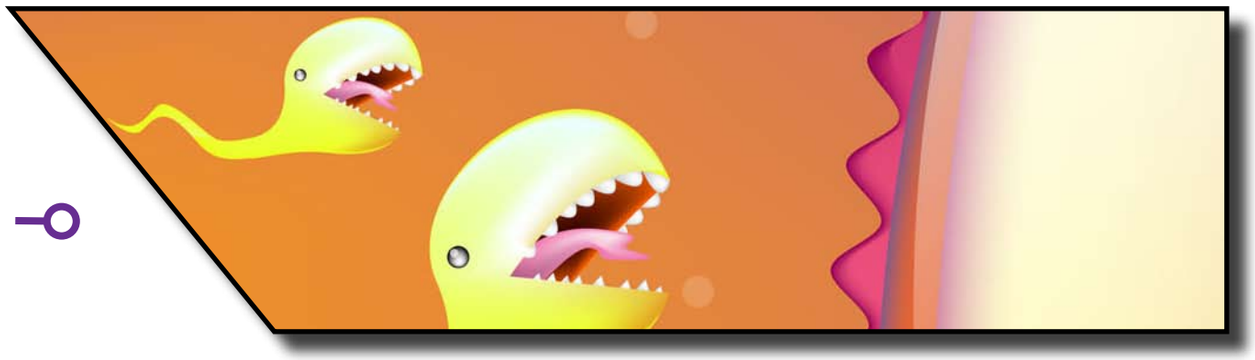
The Birthplace.

The bedroom has long been known as a hotbed of dream activity; indeed dreams are formed at the mere spark of a good idea. The internet can provide easy access to necessary development components and one can easily put a small team together; suddenly, making a game in your spare time seems like something simple to achieve. Unfortunately, a good idea and a team of eager and willing collaborators is not enough to make dreams a reality. The birth of a game should be in its design and its documented processes; the actual working through of what is to be accomplished, and how. This is the birthplace of a game.

The Conception.

The vision of a game brought to life is exciting, but what does it take to generate a design that is realizable and practical to implement? The game design document should start with something important and relevant to any game; the game concept. The concept is what makes the game stand out, and is based on the original idea. The game concept itself should manage to explain all aspects of the game in a condensed form. Keeping the relevant things in the fore and leaving the details for the body of the document. Explaining how the game works is easier in detail, but the ability to describe the overall gist of a game within a few short paragraphs will ease the game into being. The example design document contains good and bad examples of the aforementioned skill, and using it as a reference is up to you.

This condensed form is not to say that the game concept section of the document should be short at all, but it is aimed at explaining, in as brief a manner as possible, the game in its entirety. Some things seen in commercial design concepts are relevant information to the developers themselves, as well as the business and marketing sides. Brand analysis of the game and its characters; competitors games that rival the new game concept; stepping stones that this game has take to make its way into the actual design process; the purpose of the game; goals of the game; and of course, the story concept in full. For example: "Game Information : <game name> is a third person action adventure game featuring stealth, puzzle solving, and fighting game play. The game will be developed for the Xbox360 and PC platforms." This, is the art of conceptualising a game into a design document. As with most documents, the concept is somewhat of an introduction to the rest of the design.



The Growth.

Seeing as the document now has an introduction we can start to lay some meat on the bones. Extracting the major points from the first section and expanding this will start to give more of an idea to each person involved as to where the team is headed. It also helps to give perspective into certain aspects of the game that were unrealistic, and exists to prevent the addition of features and changing large amounts of the original concept. Of course, it is your game; you may change what you like, but it is important that once a team have decided the path they want the game to take, that the outlining of the game itself does not introduce unnecessary game play dynamics and start to remove from the original concept. Sticking to the concept, helps to keep the game development plausible and it also helps to lessen the likelihood of a failed attempt. Do not be afraid to strip entire aspects out of the game whilst the design process is in motion; making a realistic game design is important, but keep in mind what games are for, fun and entertainment.

As the document grows, you start laying down things such as: a detailed game concept; how the game handles input; how it displays information to the user; how the game play comes together; how the menu structure will be laid out; how the art direction will correlate with the game concept; how the game modes are broken down into parts; how each part works and the resulting effects; how the game creation process will be expanded by creating the required tools; how the tools will be used to further the development of further games in the series and/or the modding of the original game; how the controls will be used by the player and how the controls affect each element of game play; how the player interacts with the game environment; how each piece of the game falls together to make something of a playable experience; which characters will be introduced into the game and at what point in the game they become important; how AI or multiplayer games should work; how the enemies react to the player; how vehicles should work; how boss levels or major events should happen; a description of effects and camera motion; how the sound is meant to come across to the player; whether there is sound at all; whether the lighting controls the game play at all; and much more.

The above list is minimal and spans many genres. Getting across the important aspects of the design should be viewed as though one was never going to see or interact with those developing the game itself. Explaining every possible detail sounds boring, but it will make sticking to and finalising a game much more feasible than a random attempt at explaining an idea.

The Birth.

All in all, there is a lot of space for making up whatever suits you when it comes to the design of a game, but when all is said and done, sticking with the tried and tested works more often than not. Reading as much as you can about the topic is often a good choice, and trying different approaches is always something that's acceptable. Focus on details and focus on games. Don't just make designs, make games!

MOUTH TO MICROPHONE

Rodain "Nandrew" Joubert

One aspect of game creation that constantly seems to stump the average hobbyist developer is the matter of sound creation. Nowadays, experienced players can go onto the Internet, download a few indie games and easily pick up on what one may call 'stock effects;' sounds that appear in a whole host of games because developers frequently resort to the same online libraries to get their beloved game noises. Favourites include 'Famous Bird Chirp' and 'Ubiquitous Cow Moo.'

This is not strictly the result of laziness (even though we're all admittedly lazy at heart). Many developers out there are very intimidated by the idea of sound engineering; a widely held misconception which usually prevents people from trying it out. While professional audio manipulation is definitely not something to be taken lightly–those who make a career of it can tell you about the painful investment in time and equipment it requires–there's nothing that stops the casual enthusiast from jiffy-fixing a little bit of sound magic to suit their own needs. Anybody can go into Paint or GIMP and make crude, but serviceable sprites to serve their gaming ends. The same really does apply to sound; it's just not advertised enough.

DEV.MAG READER: "Right, then! Give me a sound tutorial!"

Well, if you insist. There is one quick way to do things, and it only requires you to cover three bases:

(1) Believe in yourself. This sounds hideously corny, but it's true. The number one obstacle that prevents most developers from sound crafting is the idea that it's too difficult. This is simply not true. If you want to create A-class audio, that's one thing but working with the basics is another matter entirely. Take a moment to acknowledge that you have the capability. If you have doubts, remember that this article is aimed at doubters. Take a deep breath, and move on to the next step.

(2) Secure basic equipment. We're talking really, really basic here. You only need a microphone. Nothing fancy. Just an ordinary, run-of-the-mill microphone. All you want is something that can plug into your computer and let you record sound. If a twig and a wad of bubblegum achieve this, then so be it.

(3) Get a sound editor. One which comes strongly recommended is **Audacity**. It's a tiny, 2 MB download and it's free. It's also reasonably powerful.

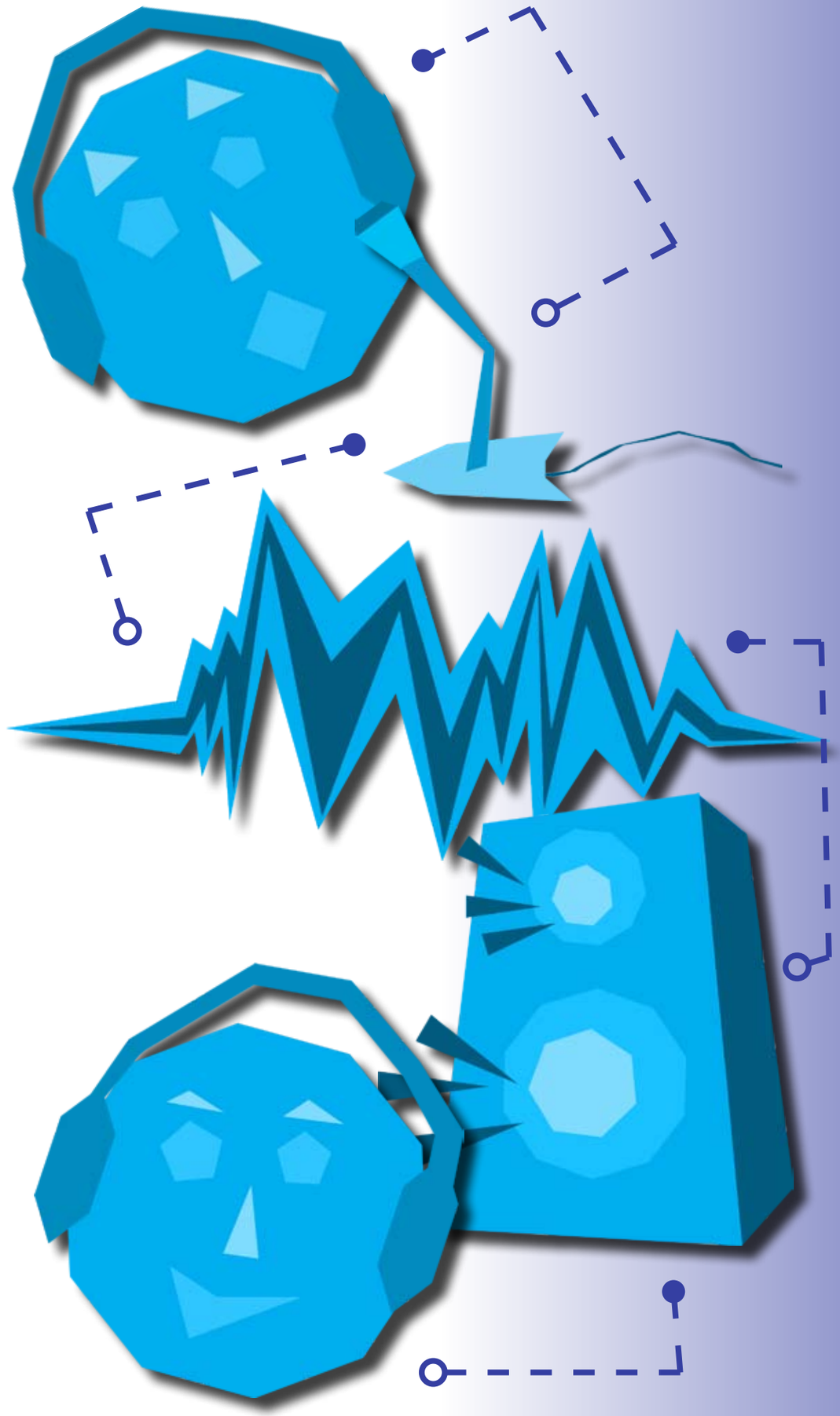
Ready? Great. Be sure to take notes!

STEP ONE: Press the big red button in your sound editor.

STEP TWO: Say something.

STEP THREE: Stop the recording.

Congratulations. You've just made your first sound effect. In less than 300 words, you've been given a complete guide to basic audio generation. Pat yourself on the back; you've done well.



DEV.MAG READER: "Hey! That was the worst tutorial ever!"

Maybe, but basic sound generation is all about your attitude. It has nothing to do with learning advanced techniques; these come through experimentation and real tutorials. This article isn't a tutorial. It's an attitude shifter. Tutorials are nasty and intimidating; they imply steps, and studies, and the need to pay close attention.

This article requires none of those (though your attention would be quite welcome).

In short: the tutorial section is officially over. Now it's time to really learn something.

DEV.MAG READER: "This isn't very useful."

Let's tell you why this really is useful:

(**1) You've just created some original material**. That's a big step beyond what most developers have, and the fact that you uniquely generated it means that you're running a 0% risk of somebody suing you for millions because you stole their intellectual property and sold it as part of a blockbuster game.

(2) You're now able to tailor sound effects to your game's specific needs. This is incredibly important because while there are a lot of free sounds on the Internet, the people who made these sounds can, at best, only guess what may be suitable for any number of games out there. When we're talking about any number, we're talking about a frightfully massive number. What's the biggest number you can think of? Chances are that the number of potential games is bigger. It's probably best if you didn't even try, you'd only end up hurting your brain.

To give a (somewhat true) example of the problems with pre-made sound libraries, we have a look at the story of Joe Developer. Mr. Developer (he gets a lot of flak for his weird surname) wants an explosion sound effect for his laser gun's wall impact. He goes to his friendly local directory of 1000 Free Sound Effects to hunt some down. About 970 of these are in the "completely wrong" category: bird whistles, water-going-down-drain noises and other various effects which are clearly meant for something else. This leaves 30 explosion noises, about 25 of which are in the "way off" category: they're explosions, but sound either like sparklers or nuclear meltdowns. Not cool.

Of the five remaining, four are in the "getting warmer" category; they're in the neighbourhood, but they're not quite right. They may be good enough to sound right on their own, but plugging them into the game and giving them a test drive reveals that they don't have that particular je ne sais quoi. The one remaining effect tends to go into the "closest fit" category–it's also not exactly right, but it stands out from the rest so you may as well go for it–and hey, it may even be quite serviceable, but things like explosions are one of the culprits of generic sound, so don't be surprised if you hear it coming back at you from some other game later. Moreover, you may want explosions for several different weapons in your game. Do you repeat the same noise for all of them? Or do you select one of the other, less desireable noises? You may even want a sound effect later for something that isn't quite as generic as an explosion, and you'll be hard pressed to find anything at all in your free directory. That's where your unique approach of sound generation comes in.

DEV.MAG READER: "My mouth doesn't sound like a laser explosion."

Really? Imagine a laser explosion in your head (not an actual explosion in your head, just the sound). Get a very clear idea of it. Then think, "It kinda sounds like ..." and try to get the closest noise using your mouth. My idea right now is kinda like "kwisk!"

So, I'm going to say "kwisk!" into my microphone. This will sound really, really stupid. When you're in this situation, try do it alone with the door closed and the lights dimmed. If you get caught by somebody, either claim to be testing the mike or explain that you're clearing your throat in a really weird way. Aside from that, just grin and bear it. Things will be a lot better once you've got the sound onto the computer. You may need to try this several times to get just the right tone and speed. It's a matter of making a few recordings and selecting the best one.

DEV.MAG READER: "My mouth still isn't a laser explosion."

Fair enough. A lot of sounds are just too abstract to instantly be replicated by our vocal chords. That's why you've got the software to help you–use it. Adjusting pitch and speed alone will give you considerable flexibility and offer some very convincing sounds. Checking the effects menu of your sound recorder will yield a lot more than that, and you're free to just open tabs and fiddle. You don't even need to understand the stuff that's thrown at you (though it helps).

Remember; even if you still pick up that your flashy new sound effect was originally a rather scratchy "kwisk!" you should pay heed to the fact that you are actively looking for the flaw. A current project of mine works exclusively on mouth-based sound effects which occasionally sound so convincing that players ask me how I find free effects to match my game so well. Being in the know, however, I can still hear the strangeness in the noises which others miss out on.

For the record, these sound effects include: robot voices; explosions of various sizes and lengths; slime effects; throws; bounces (various materials); glass shattering; laser guns; mechanical motions; swinging blades; several fancy pickup noises and many esoteric sound effects which are almost impossible to find in standard libraries. Many of them came from the family of "swish!", "shlock!" and "shwee!"

DEV.MAG READER: "That's neat, but surely this can't work for everything?"

To be honest; it doesn't.

MANY OF THEM CAME FROM THE FAMILY OF "SWISH!", "SHLOCK!" AND "SHWEE!"

DEV.MAG READER: "Thanks for being honest."

No problem. I already said that this isn't a tutorial. I'm not trying to make you into a maestro. I'm encouraging you to experiment and arguing for the viability of home-brewed sounds. Using the onomatopoeic mouth approach is just something for you to occupy yourself with while I make my point. It's not some deus ex machina. If you're not meticulous, your resulting sounds may sound quite cartoony–great for more light hearted games, but perhaps less appropriate for gritty gun-fests.

Starting with your mouth, however, can lead you on to other things: one day, when you're looking for that particularly crunchy set of footsteps, you may be encouraged to put other props near the microphone. After some deliberation, you could get a box of (clean) kitty litter, put on some boots and record one or two stomps for the right effect. After learning how to successfully use props, you may even be inspired to try more complicated stuff. Or stay where you are and refine your technique. The choice is yours, and either way you'll be getting better at making sounds without even thinking about it. Do it for long enough and you'll look back in wonderment at how scared you were before you started sound engineering (admit it, you were scared).

Basic sound effects are not difficult to make. They just aren't. You could have a lisp. You could be a 13-something boy whose voice is breaking murderously. You could be a repeat winner of the "World's Worst Orator" trophy. It's not a problem. All you need to do is overcome your shyness, speak up and aim at the microphone. From there, things will pick up. All you need to do is try a little.

A few more tutorial-like things:

Do you want to get your sound over to the computer more effectively? Here are a few basic pointers to help you along:

(1) Speak up. Your computer needs something to work with, and you shouldn't let shyness get in your way. Just don't scream, or you'll get that nasty fuzzy effect in the recorder. Experiment to see which volume works best.

(2) Beware of microphone "popping". This is sound distortion caused by the harsh enunciation of sounds (anything with the letter "p" is usually a culprit), so avoid spitting out your words. Speaking at an angle to the microphone sometimes helps.

(3) Keep the microphone at a reasonable distance. Choking hazard aside, swallowing the mike won't do you any good; it picks up your breathing easily and there's more risk of popping.

(4) Give yourself plenty of recording time before and after the sound. Don't race; you can always trim it afterwards. More importantly, being too eager with the record/stop button may have you cutting off sound at the beginning or end of your effect, forcing you to record again.

(5) Make sure you're in a quiet environment. Background noise may not seem prevalent in the recording software, but you'll certainly notice it more when the full sound effect is played in-game.

Google App Engine provides a compelling offer for deploying web applications. In this article I give a brief overview of the technology. In next month's issue, I will show you how to implement a simple game on GAE.

Using Google App Engine for Web Games

Herman Tulleken

The web is a wonderful platform for indie-games. The biggest advantage is certainly that distribution is so much easier, but the recent explosion of web technologies has opened the doors to possibilities that have not been available before. Here are some examples:

- Microsoft provides Silverlight, and Sun, JavaFX, as alternatives to flash.
- GarageGames provides InstantAction – Torque Game Engine games in the browser.
- Several social networks (such as FaceBook and Flickr) come with API's that allow developers to hook into their networks and allow their applications to take advantage of the user relationships and tagged content.

And of course, the features and support for traditional web technologies such as JavaScript and Flash has steadily improved as well.

What is Google's App Engine?

Essentially, the GAE allows you to use a set of powerful, yet easy to use API's, and run your web app on Google's infrastructure. You can get started for free – and the free quota is more than enough for a beginning: 500MB of disk space, with enough CPU and bandwidth for about 5 million page views per month and 2000 mails per day.

And really, it is hassle free. The SDK is easy to use, and a simple update script makes it a one-step process to deploy your app on the web. The API's provide useful services, including authentication using Google accounts. The API's are well designed, and well documented (and of course, the search feature is excellent!). It has also been demonstrated that it is possible to run your GAE application on other hosts, so you won't be locked into Google technologies. AppDrop is a proof-of-concept that runs GAE apps on Amazon's EC2 (Easy Computing Cloud. See http://appdrop.com/).

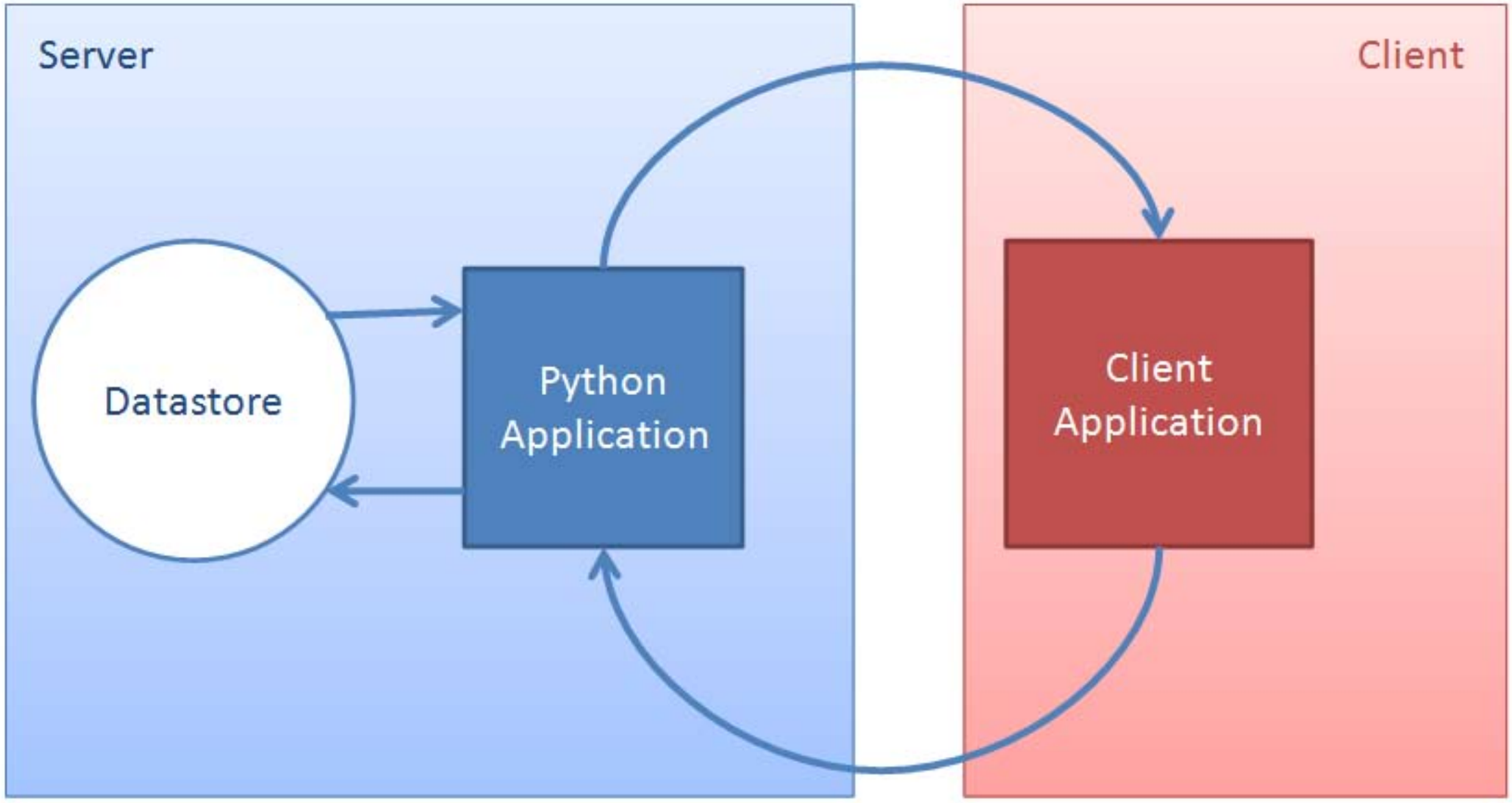
There are some drawbacks however. At this stage Python is the only language supported (although, you really should know Python). It is the scripting language of many game related tools: Blender; XSI Softimage; RealFluid; Panda3D; to name a few.

The server only works with HTTP request cycles. So you probably won't use GAE for your real-time MMOG. The API's are not infinitely powerful, and I particular miss some features in the image manipulation API. However, Google is sure to develop them further, and of course, the fact that the API's are simple makes them quick to learn.

What you should know before you get a Google App

Because of some limitations at this point, you better make sure you do not do something that you will regret later.

- When applying for a Google App Engine account, you need a mobile number to which Google SMS'es a key that you need to sign up.
- For every mobile number, you can only get 10 applications.
- You cannot delete any of your applications.
- You cannot change the name of any of your applications.
- You can either configure an application to be verified against arbitrary Google accounts, or Google accounts from your Google App domain only. You cannot change this preference once the account has been set up.
- You can link your app with an existing domain (so even if you chose some ugly sub-domain of appspot.com, you can still have pretty URLs if you own a domain name).
- The current release of GAE is still a preview release. Among other things, it means that if you exceed your resource quota, you cannot buy more. You can apply for additional quota, but it is not guaranteed that you will get it. So GAE is not quite business-ready yet.



How does it work?

Every Google app consists of a client and server. The client simply communicates with the server via HTTP requests, and can be implemented in any technology (see the links at the end of the article for some examples). It is very common to use the browser as the client application, in which case the Python application spits out HTML to GET requests. These HTML pages can then access CSS and JavaScript files in the usual manner, and can contain forms (or JavaScript widgets) that allow the user to interact with the application.

The server part is a Python application that runs on Google's servers. Essentially, the server application sets up a mapping between handlers and URLs. A handler is simply a class with methods that gets executed when a HTTP request is made for a certain URL. For example, the URL '***/view_high_scores***' might be mapped to the class ***ViewHighScoresHandler***. This class will implement a method called get, which will read the high-scores from the database, and then render out an HTML page that shows all the high scores. This page is then sent to the requester's browser, where it is displayed.

Google API's

The API's provided by GAE takes a lot of the grunt work out of developing a web application. Below I give a brief overview of the API's.

Datastore

The data store is where all your data is saved. Data is retrieved from the datastore with SQL-like queries through Google Query Language (GQL). Because it is designed for scale, the datastore is not a relational database, and some typical SQL constructs are not supported (such as the join operation).

To define a record type (a "row" in the record "table"), you simple define a Python class with the appropriate properties. You always have your data as an instance of this class, so it is very straightforward to work with it. Except for the usual types (string, integer, and so on), the data API also comes with some useful high-level data types, such as dates, time, lists, and geospatial data.

Google Accounts

Instead of rolling out your own user management system, you can simply build on top of Google accounts. This means you do not have to implement a registration/log-in system, and users with Google accounts can automatically log in. In particular, this means you do not have to worry about saving (or rather, not saving) passwords, or how to handle the "forgot my password" case. You only have access to a users email and nickname, so you still need to implement parts for storing avatars, or other user data. In some cases you might prefer to not use this API; you can still roll out your own user management from scratch.

Other API's

URL Fetch allows you to make use of other web services. Simply put, it allows your app to make HTTP request to other applications. For example, you might implement a treasure hunting game using Google's Map API, or you might serve dynamic images to illustrate a game narrative from Flickr.

The ***Mail*** API allows you to send mail using Google's framework.

Memcache is provided to boost performance of your apps. Essentially, it allows you to have data for quick access in "memory" instead of the datastore.

The ***Image Manipulation*** API provides some fundamental image manipulation services, mostly elementary transformations such as scaling, cropping, and rotation.

How can having a web server enhance my games?

Provide dynamic and user-created content. Using a web server for a data driven game allows you to make automatic updates and roll out episodic content seamlessly. As a bonus, you can let players develop their own content – so enriching their own experience, and help you to provide more content for no extra cost.

Provide large, persistent worlds for games with many players. In LAN multiplayer games, the world goes away (or, in some cases, to sleep) when the players break up; but the world in an online game can go on regardless of the players, making it much more like a real place than can be done with LAN-type multiplayer.

Build a community. The impact of a game can be greatly enhanced when the players are given means to exchange information, compare their skills, and create challenges that do not form part of the original game design. Not only do players spend more time in the game, but they also recruit other players much more aggressively.

Allow different kinds of multiplayer experiences. Facebook games such as (chess and Scrabulous) are good examples of what is typically possible: player's can invite each other to join, challenge other players, and turned based games can stretch over many days as players play whenever they have a free moment.

Keep track of player statistics. This can both enhance game play (by providing the user with a history of his progress, records, and so on), and be useful for analysis (to tweak the game for better gameplay, or making business decisions for increased profit).

So how can I use Google App Engine for my Games?

Implement your game as a full-fledged web application. This is perhaps not a good idea for games that require intensive rendering (but this really depends on the client technology you use – look at what Garage Games did with InstantAction), but 2D adventure games, puzzle games and turn-based strategy games are perfect candidates for web deployment. The links at the end of the article are all games running on GAE.

Build a portal web application to embed a stand-alone game. In this case, your game still resides on the web, and is accessed through the browser, and hence is bounded by the same constraints as the option above. However, here the game is a standalone application, and does not make HTTP requests. Instead, the web site in which the game is embedded provides extra features for players, such as forums.

Build a web application to support a stand-alone game. Here you can deploy your game on the desktop, but a web application provides extra features for players as above. Build your game as a desktop application that communicates with a server through HTTP requests. In this case, you needn't worry about rendering, although updates over the network will still be limited. However, you can still have high score tables and player ranking that update over the internet, and support at least limited interaction between players.

Conclusion

I would not yet use GAE for business critical software, but it is definitely worth looking at, especially for projects with a tight budget (such as indie games!). Like many of Google's other technologies, this one is a delight to work with!

Resources

Using GAE with other technologies

AJAX

http://code.google.com/appengine/articles/rpc.html

Flash

http://aralbalkan.com/1318

Games apps on Google App Engine

http://www.ajaxbattle.net

AjaxBattle is a multi-player real-time strategy game for 2 to 4 players. It is a web-based version of the old X-Windows game xbattle.

http://uboggle.appspot.com/

uBoggle is a app to play Boggle – the popular word making game.

http://mnk.appspot.com/

This game lets users set up most types of x-in-a-row type games – can be played against another online player, or the computer.

http://gaesudoku.appspot.com/

Sudoku.

http://games.wtanaka.com

Card and board games.

http://www.guessasketch.com/

A multiplayer game where players guess what other players draw.

http://achi.appspot.com/

A tic-tac-toe-like board game.

http://photomunchers.appspot.com/

http://jamendogame.appspot.com/

These two games are really methods to get users to tag images and music, respectively. They illustrate how to tap into other networks.

Object Pascal has a lot to offer...

www.PascalGameDevelopment.com

The cloth physics simulations we covered last issue were only a small part of Blender's simulation capabilities and, in fact, a subset of the soft body physics system that we'll cover now. The soft body simulator allows you to more accurately represent objects that are malleable and otherwise not entirely solid; basically anything that may deform under certain conditions is fair game (which is basically any solid object in the real word). So let's see how it all works, shall we?

BLENDER

SOFT BODY PHYSICS

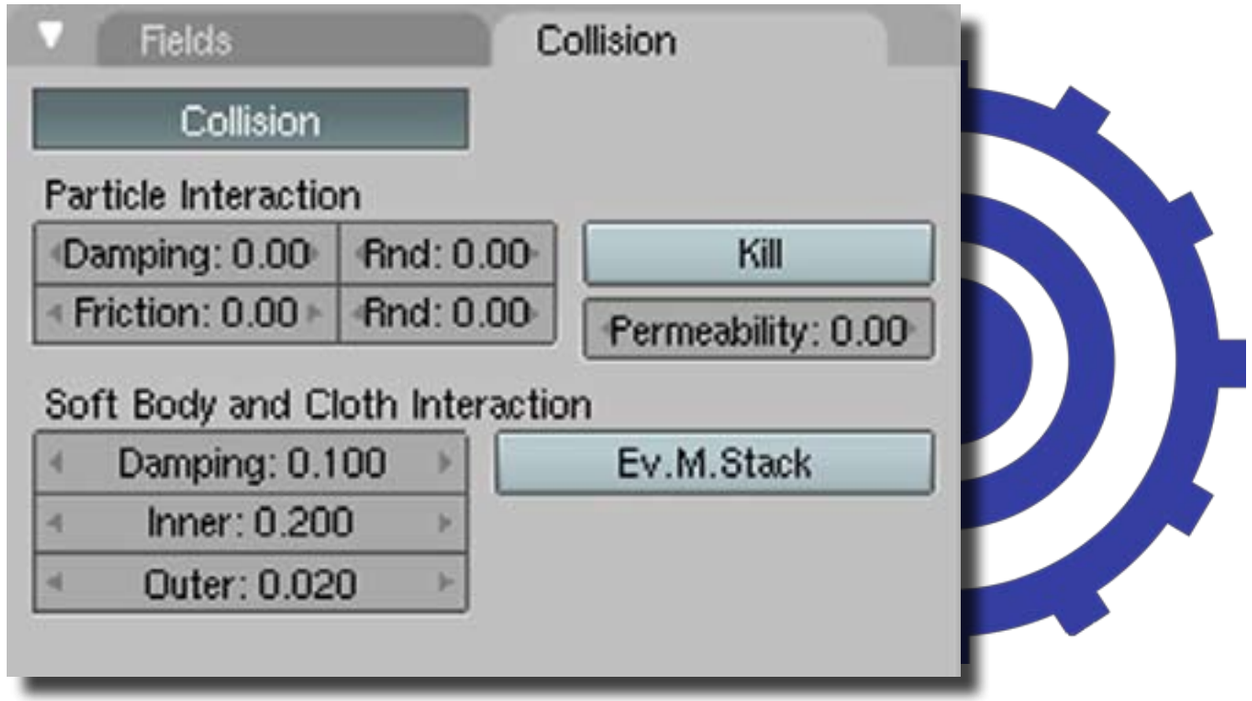
Claudio "Chippit" de Sa

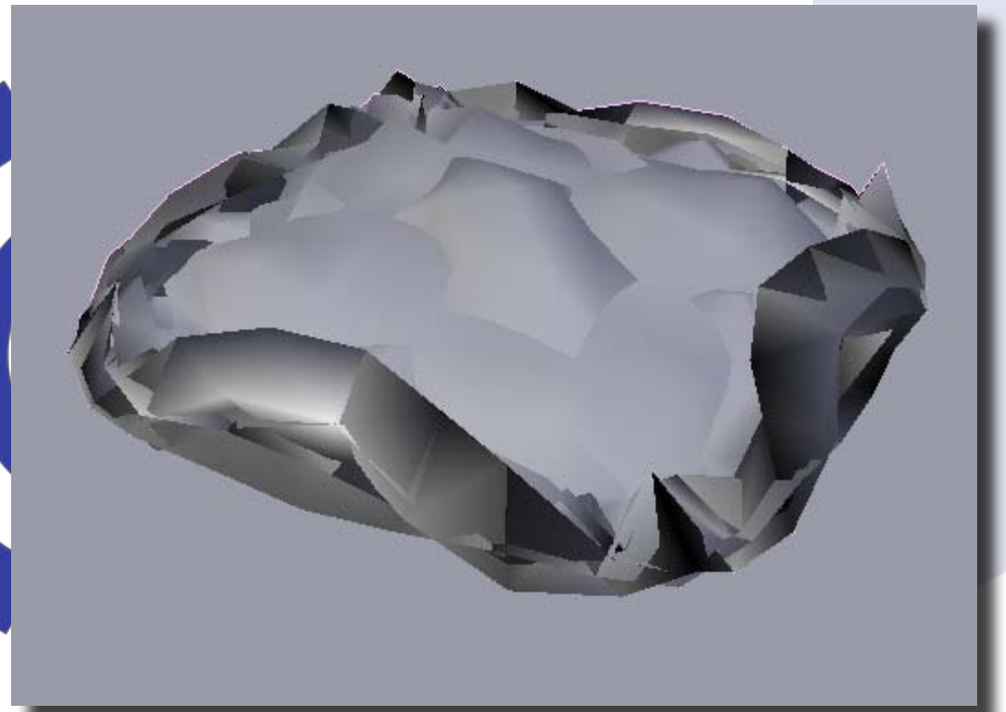
Setting up

Add a cube to a blank scene, then, in edit mode, subdivide and smooth it a few times. This will give us a nice highres mesh with rounded edges to use for our deformations. We're going to try and make this behave like a blob of jelly.

With the new cube selection, navigate to the soft body physics options. You'll find them in the same tab that the cloth physics options were located: Physics Buttons under the object menu. Click the soft body button to enable simulation on the selected object, disable 'Use Goal' (I'll explain why in a bit) and you're set to go. Hitting Alt+A in Object Mode will send the cube falling under the effects of gravity and you'll observe all the glory of the soft-body system.

Well, no, not really. However, you'll recall that this unexciting event is pretty much exactly what we started with last month. What makes it different, however, it was will happen when the cube collides with something (or something collides with it). Add a plane a little under the cube, run the animation again and see what happens. Remember to enable collision on the plane, just like you did with the flagpole last month.



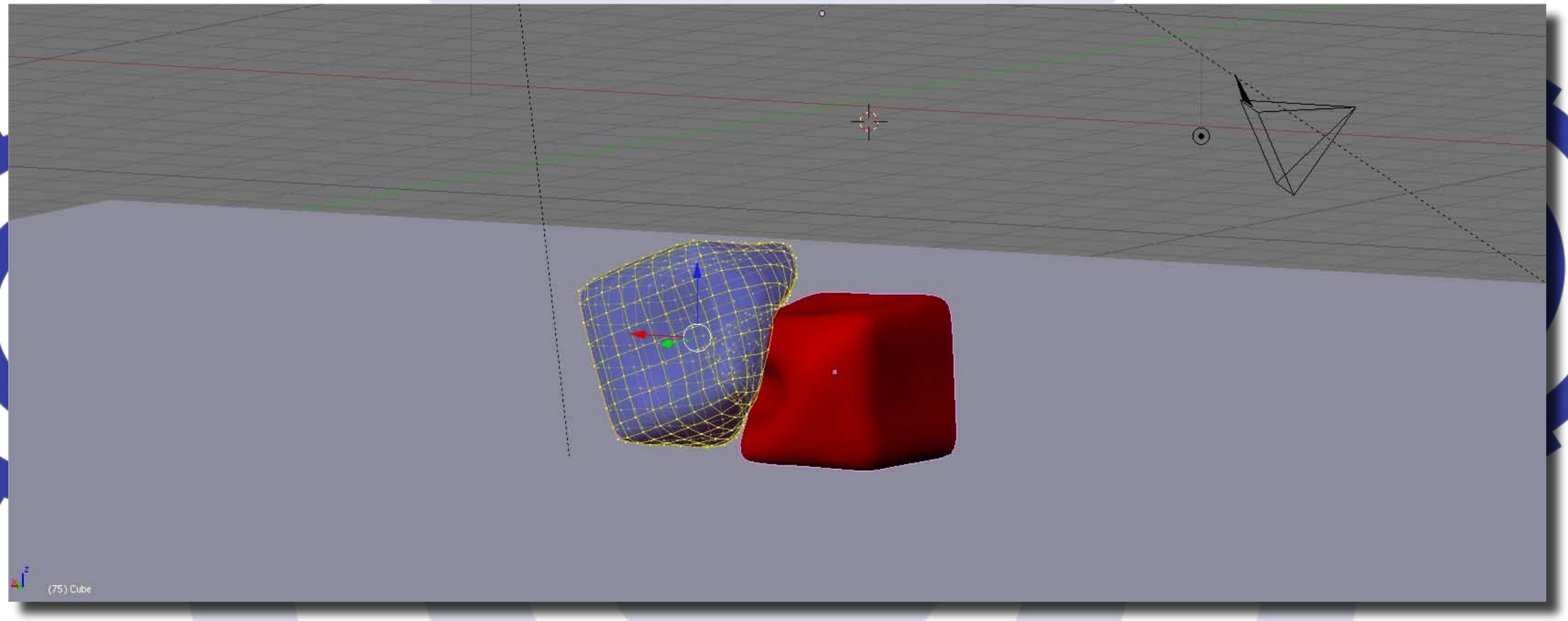


Settings and tweaks

Oh, but this doesn't look right, does it? It's collapsed in on itself. This is where all those values in the soft body tab come into play. We'll need to adjust those variables in order to obtain the results we want. To do this it's best to first understand how exactly the soft body system works.

Soft bodies work by defining 'springs' between the edges of the mesh. These springs will help the edges retain their correct length and orientation relative to each other and, effectively, allow the mesh to deform while still preserving its basic shape. Most of the options in the soft body tab adjust how the springs behave in different circumstances.

Refer to the image for an explanation of the major options in the tab, and then set your values to match those pictured. Animating now, while exponentially slower, reveals behavior far closer to what is expected of an object such as this.



Making it prettier

Like the cloth physics engine we examined last time, soft bodies use the vertices of meshes to deform objects with the Blender particle system. As such, just like with cloth, soft bodies are also affected by particle fields such as wind, and can be animated using traditional keyframe methods. To do this with soft bodies, you'll re-enable that 'Use Goal' option we turned off earlier. This makes vertices stick to predefined keyframe positions, but still behave like a soft body. The variables there define the weighting between animated positions and physically simulated behavior. Mess around with these values if you're animating soft bodies using keyframes.

Additionally, all softbodies, and in fact most of the simulated objects, can also interact with each other, as long as all objects involved have collision enabled like we've done with the floor. The final render you see here is with two identical soft bodies, using the settings above, colliding with each other. Some of the material settings I've used for this final result are out of scope of this tutorial, but, as always, this file can be downloaded from the Dev.Mag website if you're curious as to how I've obtained this output.

Simon “Tr00jg” de la Rouviere

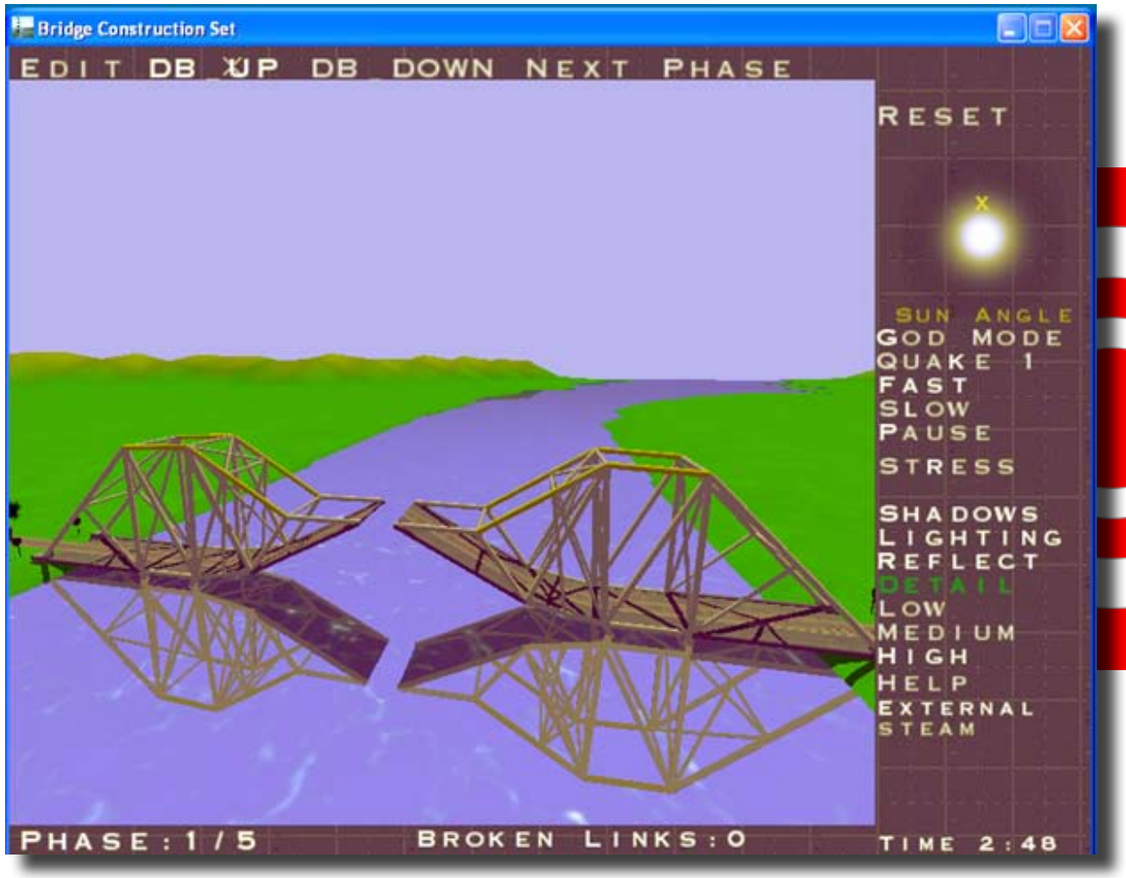
Taking TORQUE For a spin

We’ve trawled the internet for you and selected a handful of games powered by GarageGames’s Torque Game Engine. Here’s what we thought of the ones we took out for a spin. Three games; three viewpoints; one tailpiece. What will we think of next?

Bridge Construction Set.

When you watch Discovery Channel (or something similar) you can't help but stare in awe at the vast valleys and rivers that mankind have conquered via a bridge. With Bridge Construction Set it confirms just how difficult it really is to build a bridge. As you might have gathered, Bridge Construction Set is all about building bridges. The game play is quite intuitive–build a bridge. The first few levels are really easy, which is to be expected. The later levels really challenge your spatial thinking as you have to juggle your limited budget with the varying weights of materials stretched across humongous ravines.

The graphics aren't spectacular, but the game doesn't really need great graphics. The UI is tad unintuitive at first, but in the end Bridge Construction Set is a really fun game for the intellectually tuned. Masochists will love the later levels and sadists will really enjoy sending cars and trains into the abyss.



Kingdom Elemental: Tactics

Every gamer has been faced with fantasy lore. Some are good and highly engrossing while others just confuse you with unpronounceable elfin names and all sorts of wild creatures that missed the sign to the local gene pool. Kingdom Elemental: Tactics starts by accepting that most fantasy lore actually sucks. The narrator mocks fantasy, the game and sometimes the player. This is actually really refreshing alongside the strategic fare.

Kingdom Elemental: Tactics is a mix of real-time strategy, turn-based strategy and RPG on very small maps. Your task is to defeat the spawning nasties on each level with a variety of units at your disposal. There are no unit-production facilities, just units. A coin is given to you at the end of a level. These coins are used to get new unit types or new unit skills. Defeating the enemies requires a variety of units, which the game provides sufficiently. The most useful feature in the game is having the ability to pause the game, allowing you to assign orders to your units amidst the frantic battling. The interface and controls are very simple to grasp and do their job really well. Although it is nothing spectacular, the game play is solid and it is certain that any strategy fan will enjoy Kingdom Elemental: Tactics.

Tube Twist: Quantum Flux Edition

When firing Tube Twist up for the first time, you can't help but wonder if you stumbled upon an alternative version of the classic, The Incredible Machine. TIM was a game in which you placed various gadgets and objects across the screen to solve a certain goal, like getting a ball into the basket or letting the cat catch the mouse.

Tube Twist does indeed cite TIM as inspiration and upon further investigation it seems like Tube Twist is a simplified, streamlined version of TIM. Gone are the cats, monkeys and tennis balls; in Tube Twist, your job is to guide energy balls (aka Macrotrons) into their colour-matching reactor tubes to extract the Macrotrons' energy. In simple English, you must guide coloured balls into their tubes by placing various pipes, accelerators, and so on.

While the concept seems a tad dull, Tube Twist is a super solid puzzle game. If you have any propensity toward any form of a puzzle game, you will definitely love Tube Twist. As with any great puzzle game, it starts out simple and gradually introduces you to new items every few rounds, ramping up to something like orgasmic puzzle pizza.

With all the crazy ball accelerating, it can sometimes get too crazy to follow a specific ball. A slow-motion mode would've been great to track the balls progress in order to see where you could perhaps change its course next time around.

GEAR COUNT:

OVER 9000!

ONLINE

DEV.MAG

CREATE • DEVELOP • EXPERIENCE

www.devmag.org.za