May - June 2010
$5.99
A 3D ATTACK PUBLICATION

unity creative

INTERSTELLAR MARINES

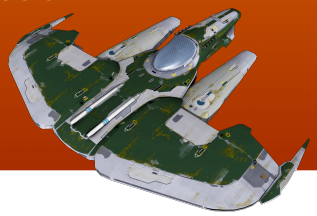
Interview with the creators of Interstellar Marines. Get to know the people behind the game and the company!
Get insight into their pipeline.

SPACE SHOOTOUT

Upcoming Space Shooter created by a 3 man team coming your way. Read what this game is all about and why it is different.

EFFECTIVE MARKETING

Indie developers are often dumbfounded as to when and how to start marketing. Learn what to do, how and when!



index

Issue One

editorial

Dear Readers,

Hello and welcome to the launch of Unity Creative Magazine. We at 3D Attack are so excited to bring to you our electronic publication dedicated to game artists and programmers using Unity to create their games. In this first issue you will find helpful, educational and informative articles, tutorials and reviews.

Our goal is to continue on this path bi-monthly. That's right, every two months we will bring you the best we have to give. Together with professionals throughout the Unity and Game Design and Development Communities, we are delighted to deliver this resource directly to your e-mail in-box.

As this is our first issue, you may see design and content change slightly with future releases. Other than that, we are ready to go!

If you have any questions, concerns, ideas or general feedback, feel free to drop us a line at 3dattack@3dattack.us

Here's to the start of something beautiful!

Tavy Pasieka
3D Attack
www.3dattack.us

CONTENT

Are you interested in contributing material for futures issue of Unity Creative? Does your company have something special to offer and you want to let the community know about it? Are you currently developing a game and want to share a bit about it? How about sharing what you know in a tutorial? Contact us with your ideas at 3dattack@3dattack.us

ADVERTISING

If you are interested in placing an ad in Unity Creative Magazine, we are happy to serve. Feel free to contact us a 3dattack@3dattack.us for pricing information and more.

COPYRIGHT INFORMATION

3D Attack
158 S. Saginaw St.
Chesaning, Michigan - USA
48616

INTERVIEW

Interstellar Marines Facts:

Interstellar Marines is a trilogy of three games, a high quality first person shooter experienced in a realistic and unpredictable future where first contact with another sentient species is slowly becoming reality. The games balance the military realism and cooperative action from tactical shooters with the character development and narrative depth from Role Playing Games.

INTERSTELLAR MARINES

Interview with Kim Haar Jørgensen, Game Director

Thomas P.: When did you decide to make this game and whose idea was it?

Kim Jørgensen.: The basic vision for "Interstellar Marines" came back in 1994, when Nicolai (Friend / Partner / Audio Composer) and I played an old Amiga game called "Hired Guns" from Psygnosis. "Hired Guns" featured a FPS-like click to move POV and offered two player cooperative split-screen at a time when most other shooters were desperately trying to clone "Wolfenstein 3D" from ID Soft. Back then we also watched way too much "Discovery Channel" and "X-Files" resulting in many long nights of talks about our ideas for "Interstellar Marines".

Several years later, Irrational Games released "System Shock 2", which featured two player cooperative multiplayer out of the box, something we have looked for in games ever since playing "Hired Guns". "System Shock 2" totally blew us away by shear Sci-Fi / FPS / RPG / Coop awesomeness and gave us additional ambitions for integrating simple RPG elements into the design of "Interstellar Marines". In 2003, Nicolai and I started looking into ways to bring our vision to life and the journey realizing "Interstellar Marines" began.

Thomas P.: What is your responsibility on this project?

Kim Jørgensen.: Being the Game Director on "Interstellar Marines", basically means that I'm the primary person responsible for both vision and ambition, and it should come as no surprise that "Interstellar Marines" has caused many years of sleepless nights where I'm kept awake in a kind of extreme creative intoxication of having this science fiction adventure so clear on the retina.

I am the guy everybody on the team can go to in the day or call at night. If they need general inspiration or detailed descriptions of the things that matter in "Interstellar Marines", like e.g. why a fork is a fork in our future and why audio counts for at least 50% of the experience, or why we rather have six hours of fun than twelve hours of boredom, or simply because they need to share creative ideas and feedback which can make "Interstellar Marines" an even better experience.

Thomas P.: When did actual development of the game begin?

Kim Jørgensen: In January 2004, Zero Point Software was founded by Gert Haar-Jørgensen, Nicolai F. Grønborg and myself. We immediately started developing ways to inspire people with the ideas and concepts envisioned for "Interstellar Marines" and to see if we could spawn some awareness of the concept. In May 2006 we released a pre-rendered game trailer presenting the game as if it were already made. Days after the trailer was launched on a few big gaming websites, hundreds of thousands of gamers started responding positively towards the concept and game publishers started calling us to setup meetings. This "proof of concept" phase helped us secure a few private investors and work building actual gameplay envisioned for our publisher demo began early 2007.

Thomas P.: How many people are currently working on this title?

Kim Jørgensen: Right now our 9 man strong development team at Zero Point Software counts the bare minimum core team for building "Interstellar Marines" one feature at the time; 1 Lead Artist, 1 Level Designer, 1 Animator, 1 Lead Programmer, 2 Game Programmers, 1 Sound Designer, 1 Composer and 1 hopefully very visionary Game Director.

Thomas P.: Are you funded by a sponsor? What's the financial situation like?

Kim Jørgensen.: To be honest we are still a few thousand Spearheads and Front-liners (Members purchasing premium account upgrades) away from being completely funded by crowd-sourcing, which is our ultimate dream. This means that we are still relying on our investors to pay the bills. They fortunately understand and believe completely in the potential of "Interstellar Marines" and our alternative business strategy called AAA Indie. We went bankrupt once (May 2009) and almost lost everything, this will never happen again. The journey forward is still up and down but one thing is absolutely sure, we will never give up before we have delivered our epic sci-fi trilogy to the world.

Thomas P.: Let's talk about the game. What's the story behind it? What's the players role?

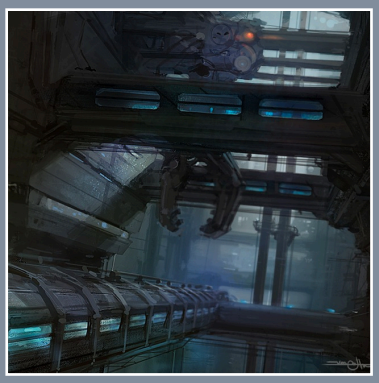
Kim Jørgensen.: "Interstellar Marines" is a first person shooter that is constantly moved forward by a credible story and a realistic vision of first contact with another sentient species. Gameplay is inspired by the military realism and cooperative elements from tactical shooters. Gameplay that dares introduce the players to simple role-playing elements and thereby provide them with personal and meaningful choices for training their weapons and character skills. Gameplay which allows each member of the team in coop the opportunity to choose exactly the type of skills that he or she believes can help to create the strongest team possible.

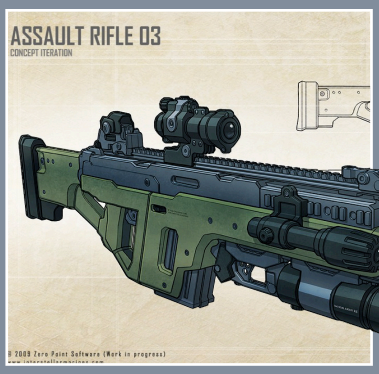
Absolute FPS immersion, meets cooperative gameplay, in a disturbing and realistic sci-fi universe, continuously driven forward by a captivating storyline and the reward of developing your character.

Thomas P.: I was a bit freaked out when I saw sharks in your FPS. Now I find it to be different and cool. Who came up with the design/idea?

Kim Jørgensen.: I have always been absolutely terrified of Great White Sharks, I respect them and I love them, but it would scare me to death to be forced to swim with them. We've had Great White Sharks and their

CONCEPT ART





genetic counterparts (Shark Creatures) in parts of our story from day one, as one of the early missions in the game is on-board a orbital research station where scientists are developing creatures for planetary warfare. Here we get a chance to play with what has to be one of the most frightening experiences in the world; being eaten alive! Watching "Alien Resurrection" back in 1997, presenting extremely cool underwater scenarios, kind of confirmed just how cool swimming with deadly creatures in space can be! :)

Thomas P.: The quality of the game, so far, looks AAA. How do you assure it stays that way?

Kim Jørgensen.: Part of our AAA commitment is ensuring that the features and technologies integrated in "Interstellar Marines" can compete with the quality of competitive games, but this is really only what is expected if you use the AAA quality mark. A bigger part of our AAA commitment is much more about changing people's expectation of sci-fi in games!

We're focusing on making the most believable and immersive sci-fi shooter you'll ever play. No laser weapons, no needle guns, etc. Our goal is that when you play "Interstellar Marines", you'll feel like you're in the Matrix and when you "unplug", you'll be wiping the blood from your mouth. Every detail from weapon feedback to communication with your friends is being focused on to archive this commitment.

Thomas P.: On your website you state "Published By The Community". Does it work well for you?

Kim Jørgensen.: Without our beloved community, and their support, we wouldn't be where we are today. We've been completely blown away by the commitment to and trust they have shown to us. It also pushes us to go that little bit further. And although those that have purchased upgrades to their accounts will always hold a special place in our hearts, it's not just they that are helping us out. It's every user and every visitor that helps give credibility to us and "Interstellar Marines". Thank you, all of you!

The financial support from the growing community is currently primarily used to show our potential investors that people from all over the world are willing to support this development strategy, with more and more people joining with each released playable preview.

It's important to the investors that our 'Sales Projections' calculated is proved, because it's a huge part of their decision to invest in "Interstellar Marines".

Thomas P.: You state that your game will have "Roleplaying Elements" - Can you tell us more about that?

Kim Jørgensen.: The RPG mechanics in "Interstellar Marines" is designed to achieve better character immersion and upgrade opportunities to both your character and your weapons, allowing the gameplay to be much more nuanced and varied than the typically "Gun and run" shooter.

Upgrades to e.g. your characters strength skills provide more inventory capacity; it increases the lethality of your melee attacks and allows you to throw grenades farther etc. That said, "Interstellar Marines" is really a hard core shooter at heart (FEAR, Half-life and Far Cry) controlled and defined by a few simple RPG mechanics balanced together for maximal replay value and better FPS immersion.

Thomas P.: You're using the "Unity Game Engine", care to share some thoughts on the engine itself? What do you like/dislike about it? What would you like to see in the future from Unity Technologies?

Kim Jørgensen.: Unity's power to deliver high quality game content on a web page, its PC & Mac platform compatibility, the fact that the technology will be available for consoles in the future, and their more liberal license pricing, all make Unity an obvious choice as technology platform for "Interstellar Marines". Utilizing the in-browser capabilities allows us to sell "Interstellar Marines" the exact same place people play it, online in their browsers at InterstellarMarines.com.

We absolutely love the Unity engine for its fast and iterative work flow, enabling developers to prototype fast and creative and with each engine update Unity move closer and closer to the best AAA game engines on the market. Unity allows everyone to test and play with the technology. This is what really drives the potential of Unity becoming a very exciting game engine alternative for many smaller game developers, even the ones developing bigger AAA quality games like we are trying to do.

Thomas P.: Will you make use of the features listed of Unity 3.0, or will you finish the game using 2.x?

Kim Jørgensen.: Our next playable preview slice called "Running Man" will obviously be released on 2.6, but we are already looking very much forward to the release of 3.0 which allows us to go crazy with deferred lighting, all the new FMOD audio features, as well as built in Beast global illumination baking for more advanced lightmaps etc.

We simply can't wait.. uhh and did I mention that we have already played around with a build of "Bullseye" on Xbox 360! Awesome stuff.

Thomas P.: When is the game supposed to be "finished"? Any estimates?

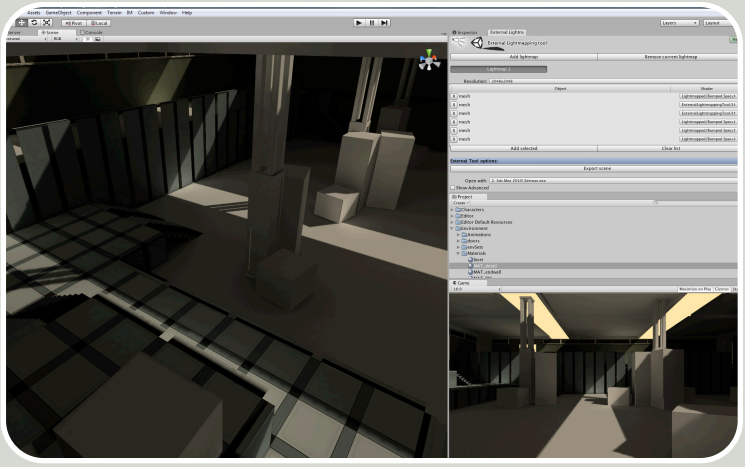
Kim Jørgensen.: We are currently focused on delivering small playable preview slices from "Interstellar Marines" as we develop the game, all designed to take us to competitive multiplayer, our big release for 2010. "First Contact" the first game in the trilogy is however still a few years away from the hands of gamers.

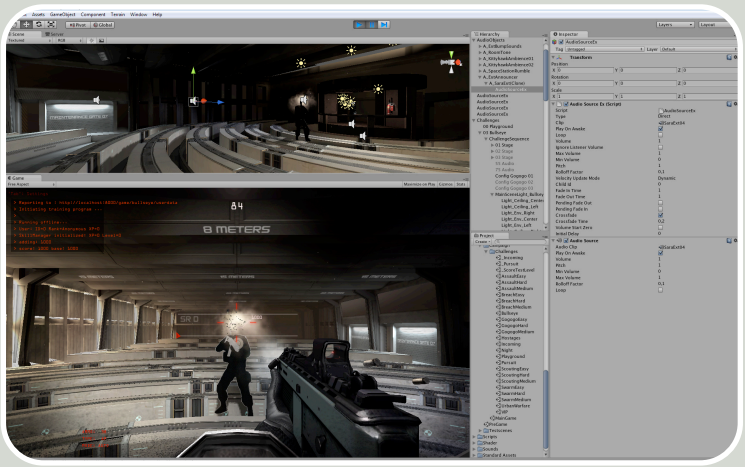
Thomas P.: Do you have any words of wisdom for the individual or small indie developer?

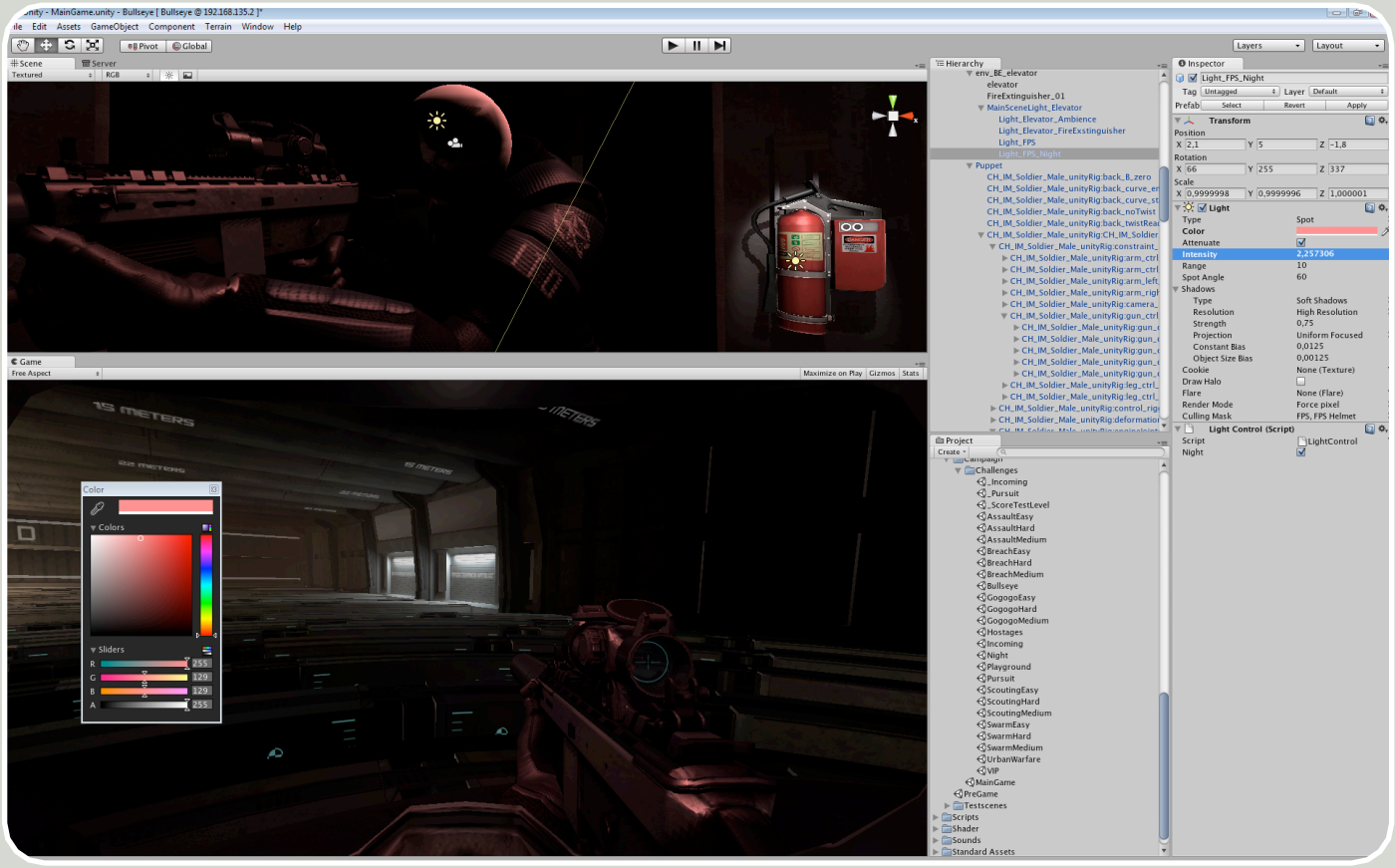
Kim Jørgensen.: Take bullshit from no one and always follow your own dreams, even if it means going where no one has gone before. Embrace failure and mistakes, because they will help you make better decisions in the future.

Thomas P.: Thank you for the interview Kim! Any last words?

Kim Jørgensen.:"I say we take off and nuke the entire site from orbit. It's the only way to be sure!"

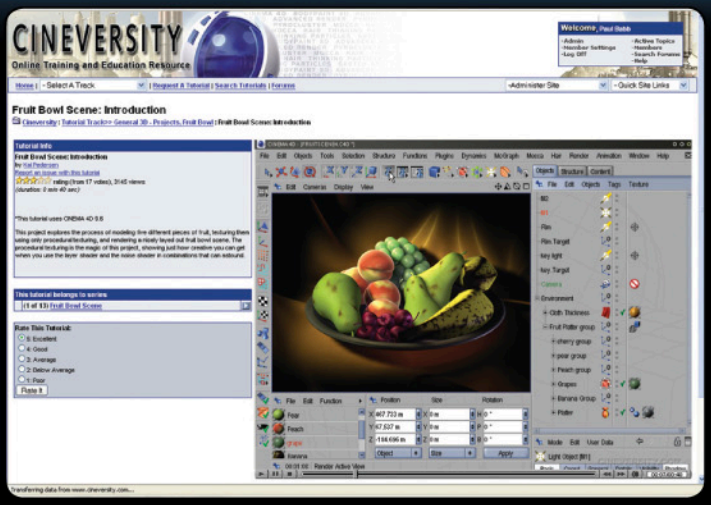






CINEVERSITY

Online Training and Education Rescource



$295 first year

$95 annual renewal

Online Video Tutorials

- Over 1200 member tutorials (250+ free tutorials)
- Production-ready presets and examples
- Exclusive members-only resources

Interactive, Online, Hands-on Training

- Learn in the comfort of your own home or office
- Interact live with your instructor
- Info and signup: training_us@maxon.net

www.cineversity.com

MAXON | 3D FOR THE REAL WORLD

TORNADO TWINS

The Netherlands has more to offer than cheese and happy cows. Let's see what the "Tornado Twins" are up to...
How did the Tornado Twins get started in the game business?

Nobody starts off as a pro game developer with a team of ten+ people and a budget that spans a million or two. More likely the reality is that you either spend years working your way towards the top in a respectable game company, or you start with bootstrapping 'the Indie way'.

Unity Technologies itself was struggling to survive in the early days of existence. And so were we. Many people in the Unity Communities know us as the 'TornadoTwins', but very few know our story. We started developing games at age 12 (many moons ago) and had gathered quite some knowledge on the subject half way through college. We wanted to be in professional game development pretty bad, but there were little opportunities to go around at the time.

Thousands of people stand in line to work at companies like Activision, Electronic Arts and Ubisoft, but very few people make it. A known game developer once said 'From the million people that want to make games, only two actually make it'. Our addition to that is: if you do happen to be one of the lucky ones, you'll be sure to start at the very bottom...

Surely, there has to be another way?

We had to come up with some type of plan. Half-way through college, we decided to take a crazy jump off into the deep end - turning the 'system' upside down. Instead of applying for a gamedev job, we registered our first company and made up our minds to shoot big. We had nothing much to show for, other than our high-school game projects.
We decided to set two rules from the get-go:
1. We go where the money is
2. We go where no game company has gone before

After a lot of brainstorming, we decided the perfect corporation to 'hire us' would be a bank: Banks have money, they have boring marketing teams, they need the creativity of game developers. All we needed was a way in...
An internship sounded like the way to go, plus the last year of college required us to complete one- so we killed two birds with one stone. Next, we networked our butts off. Trust me, it's not hard to find recruiters at all, but it's impossible to find the right ones.

Only days before our internship should officially start, we found a good recruiter that worked at 'ING bank', the 7th largest company in the world. He let us know though, that they "...only hired interns with a masters degree, no BAs". Bummer, but not a reason to give up. We let him know that we were okay with that, as long as he agreed to just meet us once. Days went by and we contacted him every other day. We must have driven him crazy. Finally he let us know there was an opening, but only for one person, not two.

It was time for another sneaky strategy. We quickly crafted ONE resume for the BOTH of us. When the interview was scheduled we both showed up! The recruiter was startled and said 'we only have room for one, but... come on in'. We replied: 'That's okay, one of us will just wait outside your office'. In the elevator up the skyscraper, we tried to keep a straight face and talk about anything other than the awkward situation we created. The excitement was incredible!

Once arrived at his office, there was no space for anyone to wait outside after all, so he agreed to interview us both at the same time. A typical 'twin moment' we'll never forget.

This was our only chance to blow him away, so we gave it all we had. Firm handshakes, well-tailored suits, witty answers and lots of laughs.
It worked. We were both hired.

The job description didn't have to do anything with games at all, but our plan was to just work hard to impress people and when the opportunity arose we would strike. Meanwhile we networked like crazy with the marketing people.

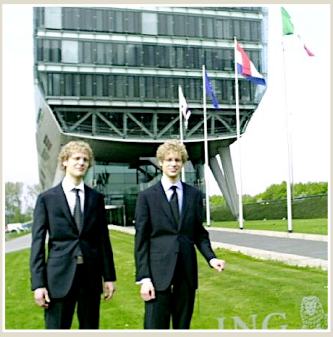
It paid off. All the reports we created were heavily designed and stood out way above the work of others (in terms of creativity, that is). We were noticed by one of the marketing guys that was over the new accounts for youth. Exactly the guy we needed!

Around the time the internship was over and we were graduating, this guy contacted us. He wanted our opinion on one of their online experiences, targeted to kids. Bingo!

At the meeting we came prepared with a well thought-through plan that exposed the weaknesses of their marketing efforts and proposed a solution in terms of a game. Half the game concept was already done for them, they 'just' needed to hire us to finish the rest.

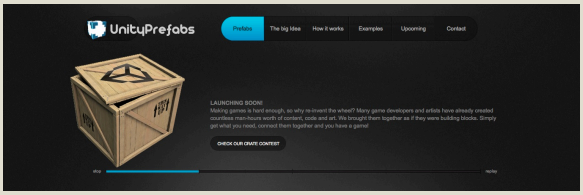
Everybody loved it.

I guess this is where you expect the 'happily every after' part. Unfortunately: not really.
At this very point in time, the economy had started to go down and budgets were cut. Again, again, and again. An official job offer therefore never came and many people we knew at the bank were laid off.

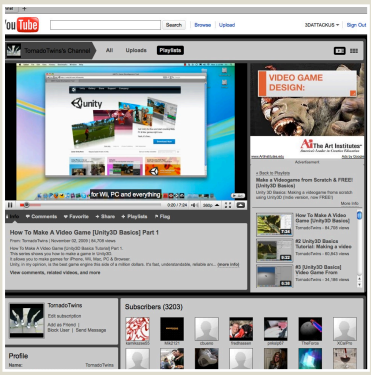


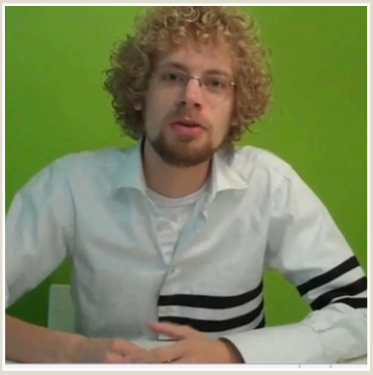
Tornado Twins

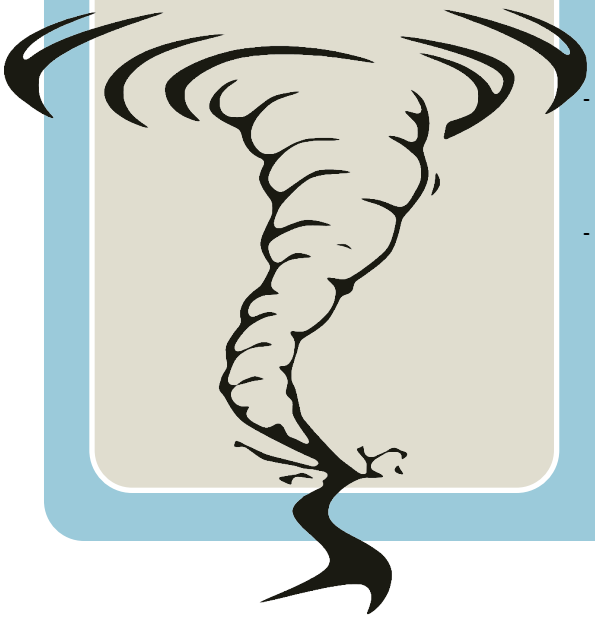
The TornadoTwins just launched 'UnityPrefabs.com', a website that hosts a professional library of pre-scripted Unity content. From waterfalls, characters to entire Facebook implementation scripts, it's there. Check out UnityPrefabs.com today!



VIDEO TUTORIALS







It looked like everything was just a bad idea in the first place: the presentations at the head quarters, the networking, it all seemed to have been a blurry dream.
Slowly we started to recognize the 'ING job' wasn't going to happen at all.

Did we shoot too high? Did we push it too far? Were we just no good at all? Should we just have settled with trying to get a job somewhere instead... like everyone else does?

But times change and so do opportunities.
After a couple of days of trying to get our ducks back in a row, we started looking back. First off, we were considered indies by one of the largest corporations in the world. If they think so, than now we are.

Second, what would keep us from doing it again? Enough corporations out there to give another go.

We decided to shoot even higher and aim international. We got hired by some of the most influential non-profits in the US and moved there. The fact that we worked with and for ING opened doors everywhere. The lessons we learned from our internship (such as "how to not come across as 'too creative'") turned out invaluable. And as a result, we now get to do what we love doing most!

If there's any advice I can give to Indies and people thinking about a game-dev startup, it would be this:

- **Don't quit your day job:** Work hard in the evenings and weekends, but if you have a shot at a steady income, don't let it go unless if there's another income ready to take over.

- **Never give up:** A dream will always stay a dream unless it's a reality. Simple, huh?

- **Feel like you're aiming too high?** Aim higher. Even if you can't make it to the moon after all, at least you've climbed the building.

- **Plan.** Those who fail to plan, plan to fail. And even if the plan still fails, you got further than without one.

- **Gain multiple skills:** Developing the actual games is only 10% of starting a game-dev company. You need to get marketing skills and most importantly: learn how to get sales!

- **Don't believe the first bump:** When you launch, sales are always higher than a couple months later. There will be a dip. A big one. Plan ahead for that and have a plan for what comes next.

- **Ideas aren't great until they sell:** Everyone can love your idea, but if there are no sales, don't infuse more cash into it. Let it go before it ruins you. Seriously.

- **Watch the TVshow'Shark Tank'** (Europe: Dragon's Den): It helps you think like an investor. Would you invest in your own idea?

- **Outsource:** You can outsource even on a zero-budget. There's no need to do what you're not good at, even on little resources. We've written a little book on it (www.tinyurl.com/theunplugged).

We hope that sharing part of our story has been an encouragement to you. If there's anything else you'd like us to write on next time, drop us a line on Twitter (@TornadoTwins). We're very interested in your projects and would love to connect with you!

Efraim & Ruben (a.k.a. TornadoTwins)

SPACE SHOOTOUT

TARGET LOCKED...

Get ready for a space shooter of the next generation.

Hello everyone. In this article you get an overview of the features and working processes of our upcoming space combat game “Space Shootout”. But before we begin with the more interesting part, we shortly want to introduce ourselves:

We are “Designation41”, a small software development company founded in february 2010 with its base in Münster, Germany. Currently the core team consists of 3 members: Chris B. (Concept/Author), Frank G. (Graphics) and David K. (Coding).

Working on our first official project we are aided by some freelancers and friends who provide us with additional stuff, mostly sound, music and additional graphics. At the moment, most of the work happens during our spare time, but we are ambitiously targeting the game market with the goal to upgrade our hobby to a more professional level. Well, that will do for now. Let´s start with the more interesting things. In “Space Shootout” you and your friends team up as members of mercenary star fighter squadrons and engage other teams in fast-paced dogfights, set in a distant future. The game is an action oriented, tactical third-person space shooter with easy accessible game controls.
It is clearly not a sci-fi flight simulator, where you have to press many buttons to simply get out of the hangar. The game is solely designed as a multiplayer game, for up to 16 players connected via LAN or Internet. A single-player campaign is definitely not planned.

"Space Shootout" comes with a set of maps, missions and playing modes. The last can be anything from basic “Free-for-all”-sessions to more complex scenarios with several mission objectives. In fact, the game is more laid out for tactical and mission-oriented team play than for simply “frag´em all up” scenarios. So many of the missions and maps will have special – sometimes unique – features, which also accounts for the maps. Some of them are vital for accomplishing mission objectives, others you can use as a tactical edge or just to pull a stunt to assure your position as the best hot-shot pilot.

Space is not only big, but also far from empty. During your missions you will encounter many things and objects, from ordinary space debris and asteroids to ship wreckage, lost cargo, satellites and even minefields. This includes even larger space ships, like civilian transports and freighters.

DESIGNATION
VIRTUAL ADVENTURES LIVING ENTERTAINMENT

The Designation41 Team

We are “Designation41”, a small software development company founded in february 2010 with its base in Münster, Germany. Currently the core team consists of 3 members: Chris B. (Concept/Author), Frank G. (Graphics) and David K. (Coding).

Such ships will be AI-controlled, but in some cases you can indirectly set and alter their actions via radio commands.
Coming to space ships, there are several different types of spacecrafts available for players: single-manned, small and medium sized fighters, as well as some „special craft“. The last category can be anything from heavy fighters and bombers to support craft like tech-shuttles and minesweepers. It will require a flight team of two or even more players, who take over positions as gunners, pilots, tech operators and the like, to actually use these ships to full capacity (The “Raptor” of the 2003 - 2009 Battlestar Galactica-Series makes a good example).

Depending on the fighter model you gain access to a selection of different weapon systems, from which you can choose the most suitable for the next upcoming mission or your preferred fighting stile. The arsenal includes guided and unguided missiles, torpedoes, mines and some classical energy weapons like laser guns, among other things. Every weapon has its own advantages and weaknesses and can even be designed for a very specific use. So even if you gain access to new weapon technology, your old armament will not become obsolete.

Beyond that, players also will be able to customize the appearances of their fighters for a more personal look. The game´s “Hangar”-section will provide a selection of skins and patterns, from which you can choose. This also includes unit and personal insignia and even some kind of “warpaint” as well.
Other gimmicks cover radio-voice sets and a squadron profile designer for example.
Currently we think about pilot-kits for another tactical note. These kits should give you some minor advantages, but they will absolutely NOT do the work for you!

Some additional background and flavor is provided by our own “home-made” sci-fi universe, which we designed as a neutral framework for all kinds of space adventures. It can be described as a “moderate” far future / space opera-setting, with versions of many classical topics and new innovative stuff as well. It is already full of background material right now, but will dynamically grow and evolve further with our forthcoming game projects.

In fact, for us, everything started with this long-term project, which originally was intended to be some kind of digital supported role playing campaign setting. But we soon realized that the whole thing was too much fun to restrict it only to that and over the next years the universe grew while the role playing part declined. The project became a setting for multi-media-use, be it short stories, audio books, artwork, music and sound and all kind of games, including computer games. The last part was only wishful thinking for years, but finally Frank came up with some serious plans to realize one of our favorite long-term goals: a tactical multiplayer first-person shooter, featuring – surprise - the UNITY engine. Although we had far enough ideas and concepts for artwork, design and gameplay, we knew that we lacked experience in other fields of work, e.g. animation. Nevertheless Frank put together some basic levels (very TRON-esque at first). The experience from these first steps was vital, showing us, that the amount of problems and necessary skills, basic and en detail, exceeded our expectations by far.
So we set up some kind of testing ground to avoid, that the game became some jury-rigged patchwork product, full of bugs and lose ends. Working with it proved so successful that the testing ground itself became actually a game project of its own. And so “Space Shootout” was born. [Chris]

Creating Assets for the Space Shootout proofs to be quite a hard task, besides all the fun we have. For one we want a precise audiovisual presentation, and second a guarantee, that the game will work proper on mid- and even some low-end systems.
To achieve that, we go through a strict design process, starting with shape studies and concept sketches.

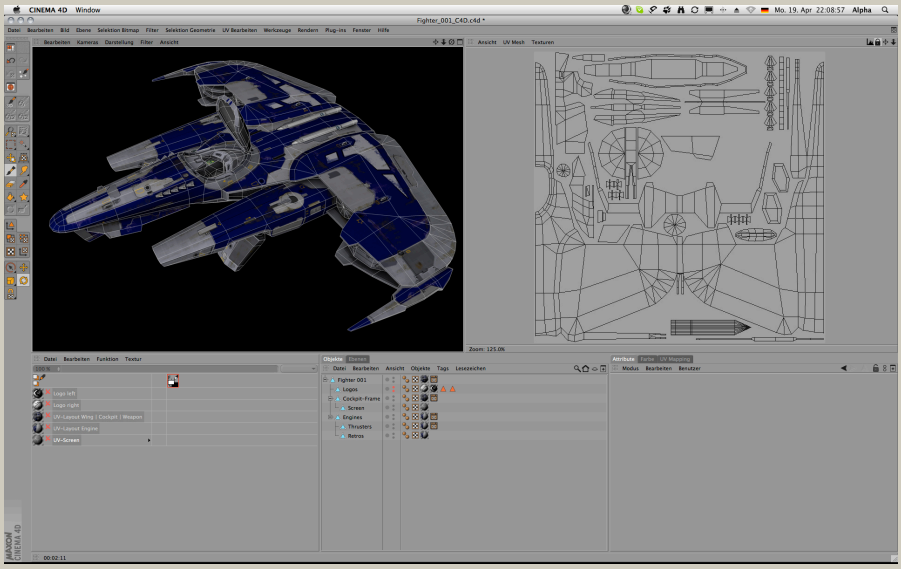
When the design is completed, we build a very basic 3D model, to see, how the shape behaves in 3D. Usually some form tweaking takes place during this process. We mainly use MAXON’s CINEMA 4D to create our models. Additional modeling work is done in Cheetah 3D and Pixologics ZBrush. In these early stages of the modeling process, add-ons like changeable engines, positioning of weapon load outs and other features of the model are planned and roughly designed.

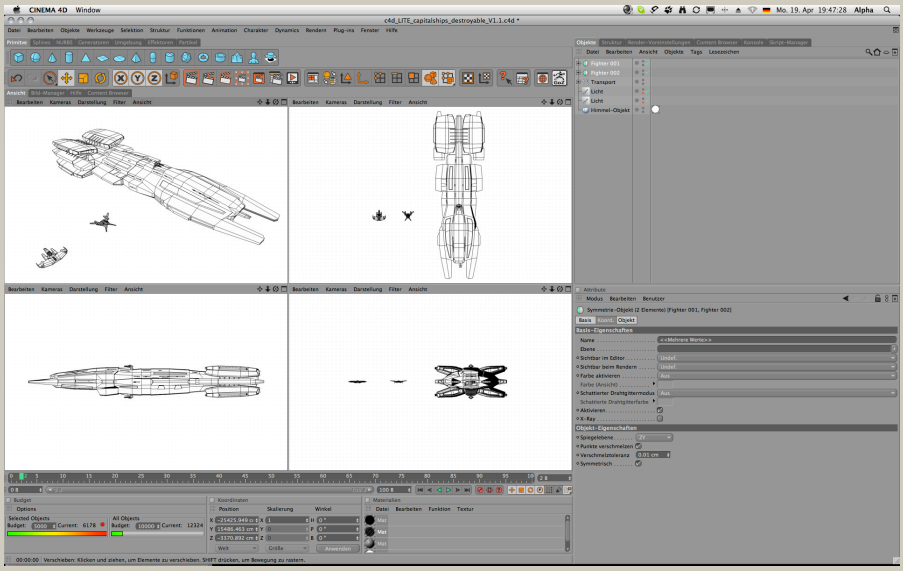
Once the basic model is completed with all its animation- and customization-features, we add details and the movable parts, such like landing gear, cockpit canopy and stuff. We decided to keep the player- and prop-models at around 3.5 to 5k polygons, so they won´t take up too much render-performance. There are at max 16 fighter models in the game, along with up to 8 larger vessels and the surrounding level. This puts us at around 300k polygons in a full scenario. Of course we use LoD systems and culling to keep the on-screen-polycount much lower.

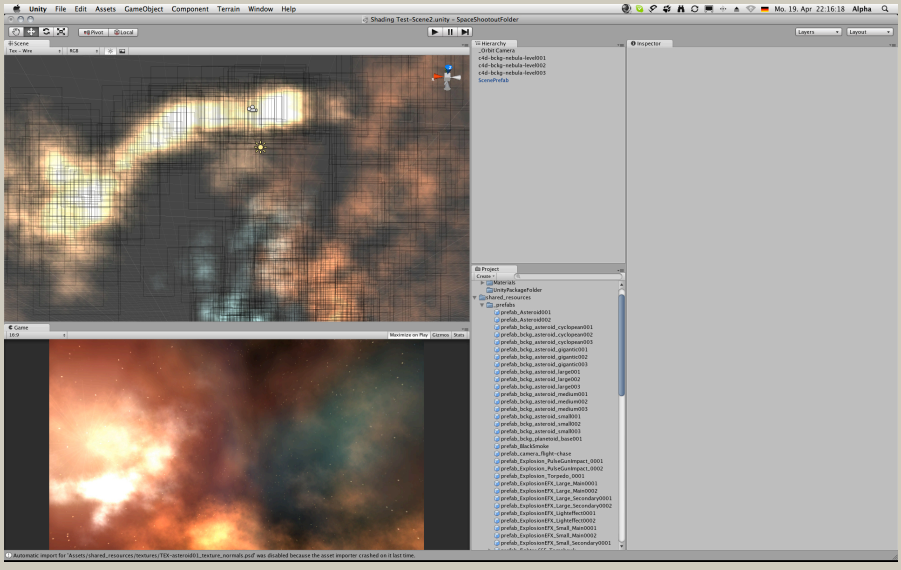
We also try to keep the draw calls as low as possible. Currently we are around a value of 200, but in the final game, this value may reach up to 350 to 400 DC´s in a full 16 player match. First tests even worked (at around 35 fps) on an G4 iMac. Target system, however, is any INTEL based Mac and standard Windows PCs, so we will not promise, that the final game will work on the G-Series of Macs anymore. 400 Draw Calls could be hard for such an old system. And yes, if it makes sense at all, we will release a ppc based version of the game, but my guess would be, that it will need at least any former high end G5 system.

Back to modeling: Personally i find it very important to already have a nice clean shading on the bare model itself. So, at this raw stage i import the model to UNITY, set the the import settings to the desired shading angle and check the model for shading issues. When encountering ugly triangles or broken shading, it can be easily fixed, by pulling some verts in your 3D App and check again. It also helps, to triangulate some parts of the model by hand. By doing so, you will get a smooth and great looking shading, that will support your model´s form and not only making it look round. I usually spend a handful of hours to get this part right. And it always pays off.

UV Layout is done mostly in MAXON'S'BodyPaint, however since Pixologic released the new UV Master a few weeks ago, we plan to switch to ZBrush, for its ease of use and excellent results in the final UV Set. Painting is done in Photoshop, after exporting the UV Set to a .psd file. The .psd UV Texture with all the mesh info is placed on the model to check for texture distortions and then imported into UNITY. It really helps during the painting, to have 2 Monitors, one for Photoshop and one for







UNITY. UNITYs excellent import features display changes to the textures on the model, as soon as you save in Photoshop, so changes can be viewed almost immediately. This makes it easy to place even the smallest details on your texture fast, efficient and with proper fit. To give your model a stronger appearance, we render ambient occlusion maps from C4D into the main texture. This gives the final appearance of the model a much more defined look with a lot of depth.

While painting in Photoshop, keeping a clean and sorted layer structure is very important. We use specific layers of the diffuse map to create the bump- and specular maps, which are mostly tweaked greyscale copies of the diffuse map with some specific changes for the best effect. When keeping the layer structures from your diffuse textures for bump- and specular texture creation, you have optimum control over the surface effects you want to create (such like differing raw metal from painted surfaces and so on). I usually have layers for basic structure, seems / bolts, paint (at best one for each color of paint), dirt, rust / damage, signs / logos and some special layers depending on the designated look i want to create. Using layer masks to create corrosion effects and scratches leaves the option, to create a brand new looking version of the model in no-time by just deleting the mask. I found this technique very helpful.

When everything looks ok, we create copies of the diffuse, bump, specular and transparency textures and combine them, using alpha channels, so you end up with a maximum of 2 textures for an object at best. We chose to create textures at 1024 by 1024 max to keep the RAM usage low, while still having a nice look. We also use mirrored UVs on our models, so we have the double amount of pixels displayed on our models for an even more detailed look.

The finished combined textures are copied into the UNITY project asset folder and are assigned to the according shaders. When putting everything together, it is usually the part, where you notice all those little mistakes you made, during the paint process, so after "finishing" i usually spend about 2 to 4 hours to finalize everything.

I am not the right man to ask, when it comes to coding, but i will also drop some words on that topic here. We use a modular framework as basic code structure, which allows us, to change, add or delete game features at ease, without distorting the rest of the game or the need to rewrite large parts of code. We also created an "in game database" which gives us access to lots of important variables while playing the game. By using this database, we are able to change flight characteristics, weapon damage and all other kind of variables in the mid of a running multiplayer game, with additional options to send these changes to all players or keep them on one player only for proper evaluation. This helps a lot, to create the right feeling for each game feature.

Well, thats it for the technical part of the game for now. I hope you enjoyed the infos and screens and we are looking forward to present a cool game to you in the not so far future. As a little eye candy we have enclosed a web based UNITY Player with one of our fighter models (pre-alpha stage). Have fun! [Frank]

According to schedule the „Space Shootout Lite“-version will be available about the end of 2010 as a free download. There will be no other limitations, no like online-registration, no additional game that incurs a fee. Just download the game and start blasting your friends into pieces.

The "Lite" version will include about three fighter models, one type of larger spacecraft, about three or four maps and a handful of mission types. In the near future, say 2011, there will be a commercial pro-version with MORE features and stuff in every aspect of the game.

Our new website at http://www.designation41.com is currently under construction and will be coming up soon. See you on the UNITY / THEUNITYARTIST forums soon and stay tuned for any info updates on our game there. [Frank/Chris]

Bye-Bye Mouse.
Hello Brass Monkey!

Brass Monkey is an SDK which connects multiple devices over a WIFI network. Imagine using your iPhone as a controller for a web based action game. Or, how about playing poker with your friends on their mobile devices while sharing one large screen for the table view. A protocol specification and implementation on multiple platforms allows one or many devices to be used to control any software which can connect to a network. Brass Monkey's event based protocol allows accelerometer data, touch events and custom data types to be sent to any end point. We encourage you to imagine the possibilities!

FEATURES

• multi-user support • works over wi-fi • sends touch events • sends accelerometer • low latency (10 - 30 ms round trip) • sends custom data • multi-directional communication • multi-device experience

PLATFORMS

• Unity3d • iPhone® (iPod Touch®, iPad) • Adobe® Flash® (coming soon) • Android™ (coming soon) • Windows® Phone 7 (coming soon) • *more available upon request.*

For SDK information and pricing, please contact Jon Leach at 978.853.4743 or leach@infrared5.com **brassmonkey.infrared5.com**

built by infrared5

beginning C#

part 1

Beginning C#

Starting a new language can sometimes feel like a daunting task. This includes both foreign languages as well as programming languages. The important thing to remember is it takes time and perseverance and in the end it can be one of the most rewarding experiences. Every couple of years I try to learn a new programming language, because I feel each language has a unique expressiveness of its own that challenges me critically and allows me to approach problems in different ways. Over the years I have found the quickest way to learn anything is with hands on experience with it.

Some things to keep in mind when learning a new language include:

1. **Start small** – Start with basic constructs and don't worry if you don't understand terms such as object-oriented, concurrent, dynamic. These concepts with come with time and practice and we will help you get there.
2. **Take the time** – Problem solving takes many hours of practice and patience, but if you are serious about learning a new language you will need to put in the time.
3. **Build a Library** – Find a selection of books for the language you are trying to learn. Take time to review the books to make sure they are compatible with your reading style.
4. **Demos** – Working with demos and discovering how other applications are put together is a valuable skill and one you will use often.
5. **Don't give up** – At times programming can become frustrating and you may feel like giving up. Some problems may seem impossible to solve. When you find yourself feeling this way take a break and let your nerves and brain relax.

The purpose of this series is to introduce the readers to the language and expressiveness of C#. From the beginning C# was designed from the ground up to support **component** concepts including events, properties and methods. Developers today do not develop the monolithic programs or class libraries of the yesteryear; instead they are developing self-contained components or classes that expose themselves through properties, events and methods. Keep in mind as we go through examples that C# is case sensitive, which is to say that "MyClass" is not the same as it's lower-case spelling "myclass".

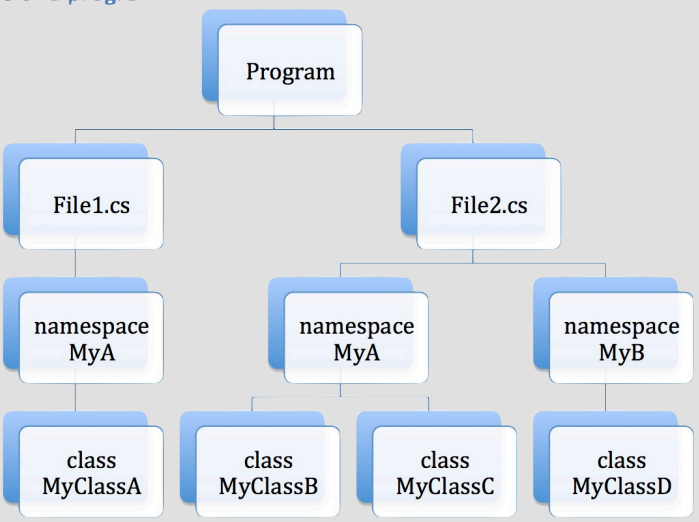
What defines a component?

1. Properties, methods and events
2. Design-time and run time attributes
3. Integrated documentation using Xml

Language Features

- Namespaces
  - Contains types and other namespaces
- Type declarations
  - Classes, structs, interfaces, enums and delegates
- Members
  - Constants, fields, methods, properties, indexers, events, operators, constructors, destructors
- Organization
  - No header files, code written (in-line)
  - No declaration order dependence

Structure of a program



The following program when executed will display a line of text in the console. This is your typical “Hello World” application, which every introductory article is obliged to recreate.

```
 1: //Our first program, printing a line of text
 2:
 3: using System;
 4:
 5: namespace UnityCreative
 6: {
 7:    class Welcome
 8:    {
 9:       static void Main( string[] args )
10:       {
11:          Console.WriteLine("Welcome to Unity
Creative!" );
12:       }
13:    }
14: }
```

Output

```
Welcome to Unity Creative!
```

Line 1: This line begins with //, indicating that this line (and remainder of the line) is a comment. Comments are used to document and increase the readability of the code. These comments help others (and sometimes even the author) understand the program. There is syntax for multi-line comments, however C# programmers typically use the single line style:

```
1: /* This is a multi-line
2:    comment */
```

The compiler ignores comments; and as such they do not cause the program to execute any action. It is considered good programming practice to comment your program.

Line 2: The using statement allows you to reference another namespace and use those types within your program. This allows the types within System to be used without pre-pending System to every reference (System.Int32). Namespaces group features into categories. The Console static class for example is contained within the System namespace, which contains code that you the programmer can re-use within your own program.

Line 3: The blank line and whitespace (spaces, tabs, new lines) add readability to the program. The compiler ignores these whitespace elements used to separate language elements.

Line 5: A namespace is a logical grouping of names/identifiers used within a program. Using namespaces you can logically group related features so they can easily be used within your program or other programs.

Line 6 and 14: The left brace “{“ begins the body of the namespace declaration, and the corresponding right brace “}” ends the body of the namespace declaration.

Line 7 - 13: The programmer defined class definition. The **class** keyword begins the class definition and is followed by the **class name.** By convention class names begin with an uppercase letter, and each word in the class name is also capitalized. This class name can contain letters, digits, underscores, and @. Identifiers (Class names) cannot begin with a number and cannot contain spaces.

Line 8 and 13: The left brace “{“ begins the body of the class definition, and the corresponding right brace “}” ends the body of the class definition.

Line 9: This represents the entry point in a console application. Console applications begin by executing at Main. The parentheses are an indication that this is a method, and in this case takes an array of strings. In a console application these parameters are passed on the command line and passed into this Main method.

Line 10 and 12: The left brace “{“ begins the body of the method definition, and the corresponding right brace “}” ends the body of the method definition.

Line 11: This is the line that instructs the computer to execute the action (print a string to the console). The Console class contained within the System namespace enables programs to output text to the standard output. Console.WriteLine outputs the text followed by a newline character. Every statement must end with a semi colon (;) which is called the statement terminator.

Escape Characters

To represent some characters in a string, they must first be escaped. Consider the double quote character. Since a string literal is defined beginning with a double quote and ending with a double quote, how can you represent the double quote within the string?

Escape	Definition
\n	New line, position cursor to beginning of next line
\r	Carriage return, position the cursor at the beginning of the current line.
\”	Double quote, used to represent the (”) character within the string
\\	Backslash, used to represent the (\) character within the string
\t	Tab

Strings Literals

C# supports both regular string literals and verbatim string literals. A regular string literal is the most common and consists of zero or more characters enclosed in double quotes, and may include single escape sequences as well as hexadecimal and Unicode escape sequences.

```
1: "Welcome to \nUnity Creative"
```

A verbatim string literal starts with the @ character followed by a double-quote, zero or more characters, and a closing double-quote. In a verbatim string literal, all the characters between the delimiters are interpreted verbatim, (with the exception of the quote-escape-sequence). The same string represented in the regular string literal can be represented as a verbatim string in the following example.

```
1: @"Welcome to
2: Unity Creative";
```

Here the newline character is interpreted verbatim. To escape a double-quote you use two double-quotes (“”), which will result in one double-quote within the string.

Data Types

In order to program in C# it is important to understand the types that the language supports and how to use them in program.

Data Type	Description	Example
object	The base of all types	object obj = null;
string	String type – sequence of Unicode characters	string str= “Unity”;
sbyte	8-bit signed integral type	sbyte val = 12;
short	16-bit signed integral type	short val = 12;
int	32-bit signed integral type	int val = 12;
long	64-bit signed integral type	long val1 = 12; long val2 = 34L;
bool	Boolean type; a bool value is either true or false	bool val1 = true; bool val2 = false;
char	Character type; a char value is a Unicode character	char val = ‘h’;
byte	8-bit unsigned integral type	byte val1 = 12; byte val2 = 34U;
ushort	16-bit unsigned integral type	ushort val1 = 12; ushort val2 = 34U;
uint	32-bit unsigned integral type	uint val1 = 12; uint val2 = 24U;

ulong	64-bit unsigned integral type	ulong val1 = 12;
		ulong val2 = 32U;
		ulong val3 = 56L;
		ulong val4 = 78UL;

float	Single-precision floating point type	float val = 1.23F;
double	Double-precision floating point type	double val1 = 1.23; double val2 = 4.56D;
decimal	Precise decimal type with 28 significant digits	decimal val = 1.23M

Type System

The type system in C# contains the following two basic types:

- Value types
  - Directly contain data and holds the data on the stack
  - Cannot be null

Value Types	
Primitives	Signed • sbyte (signed byte) • short • int • long Unsigned • byte • ushort • uint • ulong Character • char Floating-point • float • double • decimal Logical • bool
Enums	enum GameState : int { Unknown, Off, On }
Structs	struct Point { int x, y; }

- Reference Types
  - Contain references to object, where the reference is stored on the stack, but memory is allocated on the heap
  - Can be null

Reference Types	
Classes	public class Circle { }
Interfaces	interface IShape { }
Arrays	int[] a = new int[10];
Delegates	delegate void MyFunction();

Type Conversion

Think of type conversions as allowing you to copy a value into a variable or method of another type. For example you may have an integral type and want to pass it into a method that accepts double parameters. C# supports the following types of conversions:

- **Implicit conversions** – Implicit conversions are converted without explicitly defining the conversion. No special syntax is required because the operation is type safe and no data loss occurs. Usually these are widening operations, that is to say you are converting from a smaller integral the to a larger integral type (short to integer for example).

- **Explicit conversions** (Cast) - Explicit conversions include the type cast operator. In this case the source and destination are compatible types, but there is the potential for data loss to occur if the destination is a smaller size or is a base class of the source.

- **User defined** – (Conversion Operators) User-defied conversions are performed by special methods that you can define to enable either implicit or explicit conversions

- **Conversion with helper classes** – There are helper classes built into the framework for converting between different types. For example the **System.BitConverter** can be used to convert from integral types to byte arrays or even DateTime objects. Furthermore the **System.Convert** class or Parse method for built-in numeric classes can be used to convert strings into the numeric equivalent.

Allowed Implicit Type Conversions (Numeric Types)

The following table shows examples of the type and the allowed implicit type conversions. In other cases an explicit cast is required using (type) or one of the other conversion methods. The conversion from an integral type (int, uint, long) to float, or from long to double may cause loss of precision, but not a loss of magnitude. Also a constant expression of type int can be converted to smaller type (short for example) as long as that constant value falls into the range of the destination type.

From	To
sbyte	short, int, long, float, double, or decimal
byte	short, ushort, int, uint, long, ulong, float, double, decimal
short	int, long, float, double, or decimal
ushort	int, uint, long, ulong, float, double, or decimal
int	long, float, double, or decimal
uint	long, ulong, float, double, or decimal
long	float, double, or decimal
char	ushort, int, uint, long, ulong, float, double, or decimal
float	double
ulong	float, double, or decimal

http://msdn.microsoft.com/en-us/library/y5b434w4.aspx

```
1: // Implicit conversion from an integer to a long
2: int num = 123457;
3: long lNum = num;
```

Line 3: Here the implicit conversion from an integer to a long (a widening operation) is safe.

Allowed Explicit type Conversions (Numeric Types)

The following table shows examples of the type and the allowed explicit type conversions. The cast is used in instances where an implicit cast is not available. Also in an explicit cast may result in data loss or loss of precision.

- Decimals to integral types round towards the nearest integral. If the resulting value is outside the range of allowed value an error will be raised (overflow)
- Double or float values converted to integral types are truncated. If the resulting value (in a checked context) is outside the range of allowed value an error will be raised (overflow)
- Converting from a double to float, the value is rounded to the nearest float value. If the result is too big or small to fix into the destination the result is 0 or Infinity.
- Converting a decimal to a float or double, the value is rounded to the nearest float or double value.
- Float or double to decimal are converted to decimal representation and rounded to the nearest number after the 28th decimal place (if required). Numbers too small to be represented by a decimal will result in 0, if the number is too large to be represented by a decimal an error will be raised (overflow).

From	To
sbyte	byte, ushort, uint, ulong, or char
byte	sbyte or char
short	sbyte, byte, ushort, uint, ulong, or char
ushort	sbyte, byte, short, or char
int	sbyte, byte, short, ushort, uint, ulong,or char
uint	sbyte, byte, short, ushort, int, or char
long	sbyte, byte, short, ushort, int, uint, ulong, or char
char	sbyte, byte, or short
float	sbyte, byte, short, ushort, int, uint, long, ulong, char,or decimal
ulong	sbyte, byte, short, ushort, int, uint, long, or char
double	sbyte, byte, short, ushort, int, uint, long, ulong, char, float,or decimal
decimal	sbyte, byte, short, ushort, int, uint, long, ulong, char, float, or double

http://msdn.microsoft.com/en-us/library/yht2cx7b.aspx

```
1: // Explicit conversion from an double to a int
2: double dValue = 1234.5;
3: int iValue;
4: // Cast double to int.
5: iValue = (int)dValue;  //the value here is 1234
```

Line 5: The type operator (int) is used to explicitly convert the double value (1234.5) to an integer value store in iValue. For this conversion type the double value is truncated. Since 1234 can be represented in an integer no error is raised.

Historical Perspective

All this talk of numerical data types reminds me the first programming language I was introduced to (circa 1984). The language was Quick Basic and I was a 10-year-old set upon the task of writing a program that could do simple math calculations. The computer was an Osborne with a 10-meg hard drive and a 1200-baud modem. Bulletin Board Systems (BBS) were the "rage" back then and operated completely over modem (1 user at time, and busy signals). The first language was Quick Basic, Let's take a look at a simple program that can add two numbers together and output the result (next page).

```
1:  //-------------------------------------------------------------
2:  // Program 2: This program accepts two values from the console,
3:  //           converts the values to integer, adds them together
4:  //           and outputs the result
5:  //-------------------------------------------------------------
6:  using System;
7:
8:  namespace UnityCreative
9:  {
10:    class AddTwoNumbers
11:    {
12:       static void Main( string[] args )
13:       {
14:          //local string to store first number
15:          string numOne;
16:          //local string to store second number
17:          string numTwo;
18:
19:          //Output to the console "Enter first number:"
20:          Console.Write("Enter first number: ");
21:          //Read the input from the console
22:          numOne = Console.ReadLine();
23:
24:          //Output to the console "Enter second number:"
25:          Console.Write("Enter second number: ");
26:          //Read the input from the console
27:          numTwo = Console.ReadLine();
28:
29:          //Parse the string stored in numOne into an integer
30:          int num1 = int.Parse(numOne);
31:
32:          //Parse the string stored in numOne into an integer
33:          int num2 = int.Parse(numTwo);
34:
35:          //add the numbers
36:          int sum = num1 + num2;
37:
38:          //output numOne + numTwo = sum
39:          Console.WriteLine("{0} + {1} = {3}",
40:             numOne,
41:             numTwo,
42:             sum );
43:       }
44:    }
45: }
```

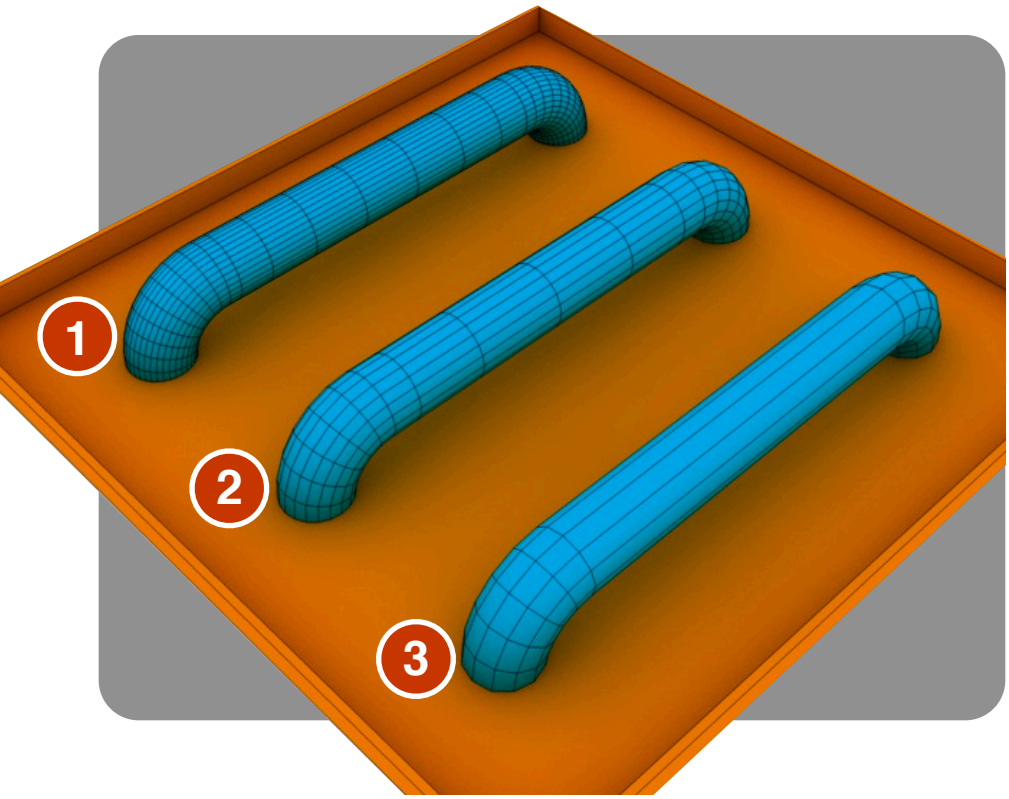
Next Issue

In the next issue our journey will take us from arithmetic and decision making operators through programming control structures. It has been both an honor and privilege to share this part of the voyage with you. If you have any suggestions or questions please feel free to contact me.

Shawn McCarthy is an experienced Business Systems Architect at First Data. Shawn graduated top of his class with a Master's in Computer Science and Engineering with an emphasis on Artificial Intelligence on May 16th, 2009. Shawn is also the Technology Lead and co-founder of an independent game development team (Six Times Nothing). You can reach Shawn at (shawn@sixtimesnothing.com) and follow the teams' projects at www.sixtimesnothing.com.

You can reach Shawn at (shawn@sixtimesnothing.com) and follow the teams' projects at www.sixtimesnothing.com.

Optimize, always!



Always optimize

Every artist should make it a good habit to optimize game models, textures and UV's, even when it's not necessary. This "habit" will be most valuable when you really need to depend on it. Your clients will thank you!

"A game artist should always try to optimize and think economically."

If you read this article, chances are you're a game artist just like I am. However, I wasn't always a game artist, I started out doing "regular" 3D work. For example, Before I got into Game Art I had been doing a lot of product visualizations, exhibition layouts and Architectural work for many years. Back in the days I didn't have to think a lot about optimizing my meshes or textures/uv's since there simply was no reason for me to do so. My models and scenes were never intended for "realtime" purposes. So, in essence, my workflow had to change dramatically.

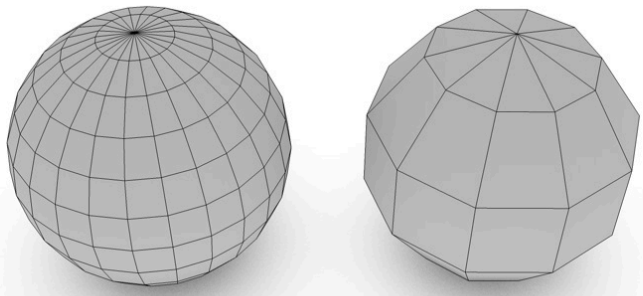
I have to admit, it wasn't easy in the beginning. I visited many forums and websites, and gathered tips and tricks from wherever I could and made them "law". A few years later I can honestly say that I love doing what I do as a Game Artist. Nowadays, I get hired to help optimize models, workflows/pipelines and also to "train/teach" people to think "green" in terms of resources and the likes.

Ok, enough of my past. Let's focus on your future. If you want to be a game artist you need to be aware of all the things mentioned above. Therefore, I will give some examples of what to do, and what not to do. If you have a look at the images above you will see 3 pipes on an orange ground. I rendered that image in wireframe so you can see the differences.

Clearly, the pipe labeled (1) has way too many polygons. This is what you get if you let your software package handle it for you. Way too many rotation segments and, overall, just not very "game friendly".

The pipe with the label (2) is looking better and has about half the amount of polygons compared to Pipe (1). Already much better, but still not perfect.

Now let's check pipe (3). This pipe segment has about half the amount of polygons of pipe (2). The object is still clearly recognizable as a "pipe" and it's shape is still readable. Pay attention to the outline of the object. The outline is the important part actually. You have to optimize your object, but try to maintain it's shape! Don't go overboard as seen below!



This is no longer a sphere!

Now, some folks may say that more modern computers and consoles can easily handle higher polygon counts. This is of course true, but that is still no free ticket to go insane on polycount. In the end it all adds up and, before you know it, your machine will drop frames.

Polygon count, however, is not the one and only performance killer in games. There are many other factors. For instance, let's talk about "Drawcalls" . The scene on the right shows a few cubes and two directional lights. Each directional light causes one additional drawcall per object/ cube. We have 12 cubes, so that sums up to a total of 36 drawcalls! Drawcalls will affect your performance, and even more so on your not so powerful devices as iPhone, iPod Touch or iPad.

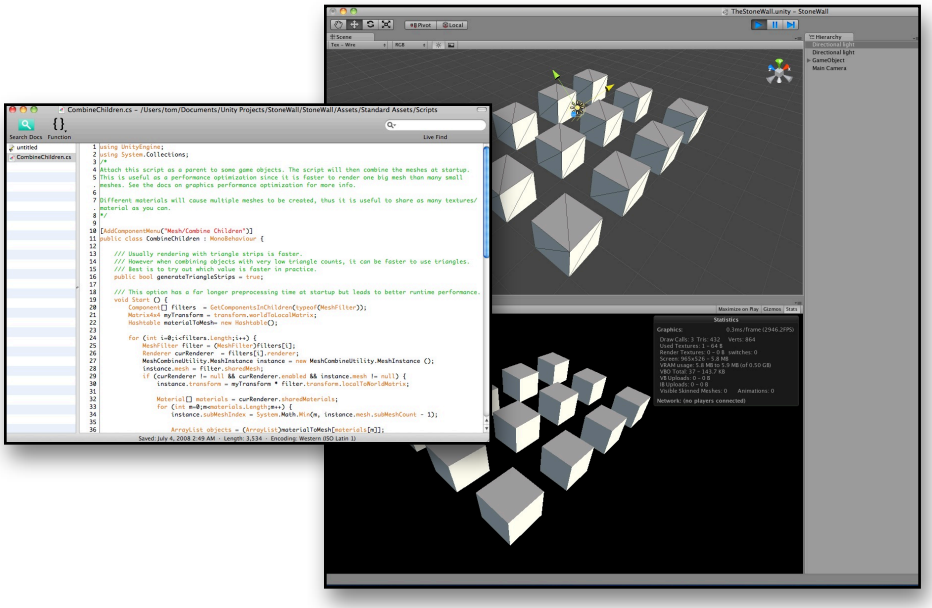
By using a script called "Combine Children" (found within the standard assets) you can drastically reduce drawcalls in this scene. I created an empty game object and made all cubes a child of this object. I then applied (drag&drop) the script (Combine Children) onto the empty game object. This greatly reduces the drawcalls in my scene down to 3!

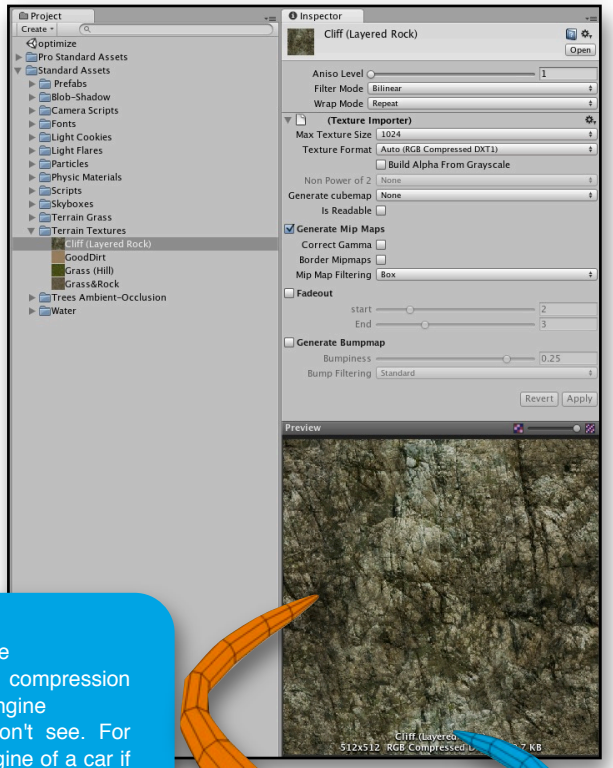
But wait, there is more! As the artist you should also be aware of "texture compression methods" and sizes/dimensions. Luckily, Unity makes that part very easy for us. Once imported, you can set things like "Max Texture Size" and "Texture Format (Compression Methods), as well as several other features.

In case I need to get up close to textures inside the game, I usually keep my textures fairly high. I test them by using different max texture sizes in the Unity editor.

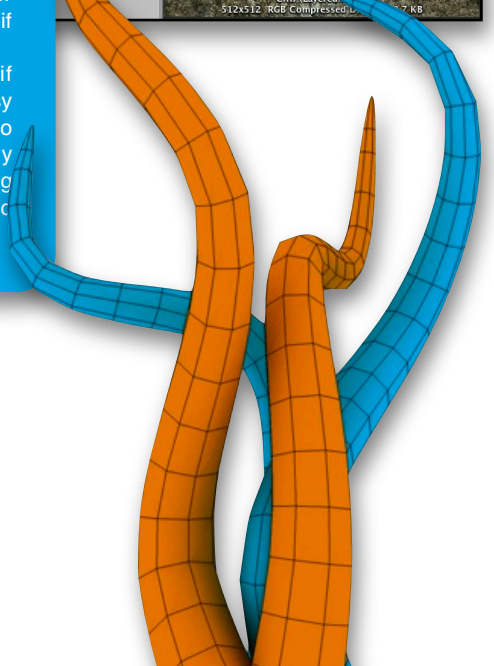
Everything mentioned above should become "second nature" to you as "Game Artist". Make yourself familiar with these matters. Learn from other artists and ask questions on the various forums. There is of course more to learn, but I hope this well help you on your way to becoming an efficient artist.

Thomas Pasieka





- Combine Meshes if possible
- Change texture size and compression method inside Unity game engine
- Don't model what you don't see. For instance, don't model an engine of a car if you never open the hood.
- Keep your meshes in "Quads" if possible. Triangulated meshes are messy and hard to fix/clean if you ever need to change something. Secondly, Unity converts the mesh to triangles during import so there is no reason you would triangulate it by hand.



collaboration using unity

An article on “Collaboration Options” by Erik Harg - Terra Vision

As a versatile game development platform, Unity has been adopted by both individuals and organizations of varying sizes. Working together with others on a game comes with its own set of challenges, that do not necessarily stem from the creative or technological development itself, but from the process of putting the team’s efforts together. This is true, even when using a tool like Unity, which excels at being accessible for almost all professions involved in game making.

In this article, we will discuss some of the problems your team may face when using Unity for collaborative game development. While we give a brief outline of the general and basic issues of collaboration, we will mainly focus on Unity-specific pitfalls, challenges, and solutions.

Collaboration Conundrum

Collaborating with others on your game project is rewarding: You can harvest the creative power of more people. You will not be the only one to test your game. You get someone to discuss important issues with, such as gameplay, mechanics, art direction, and the size of the drop shadow on the headlines of your menu.

All this is fine, until you start actually working on the same project, the same files. It is likely it will take less than a day before you figure out you will need a better way of sharing the project than taking turns sending the latest version back and forth. For your first project, you will now probably try to all work directly on the same files on a FTP-server, shared Drop-box account or something similar. It will not work.

Even if you manage to avoid overwriting each other’s work, you are at some point almost certain to at least overwrite your own, wishing you could turn back the time, and get some previous version of the file. If you have ever felt this, or the need for multiple people to edit the same files, you are in the market for a version control system (see sidebar). This will give you some protection (please do back up your files anyway), but it will also give you a neat and accessible history of who did what when. Starting to use version control systems does not have to be a pain, and as a Unity user you have a few options, which hurt just a little, but in different ways.

Version control for Unity users

Since version 2.0 of Unity 3D was released, those who pay up have had the option to use Unity’s own Asset Server for version control purposes. Version 2.6 also gave us support for using external version control systems. So, which should you use, and what should you be aware of?

Unity Asset Server

Using the Asset Server is easy. The server does not require much setup, and the client is right there, built into Unity. It is tightly integrated with Unity itself, so for the most part it means that it has a better idea of what is going on with your project than most external tools will. However, using a custom, proprietary system like the Asset Server means you will have very limited availability of third party software. If you rely on any existing project management systems, bug-tracking databases or other software that integrates tightly with a version control system, you will almost certainly be out of luck with the Asset Server. You will either have to write your own software (see Fig. 1) to do the integration, or rely on the Unity community to support your needs.

Looking at the Asset Server as a pure version control system also shows it is lacking in places. There is currently no support for standard features such as branching or merging of branches. This is not much of a problem for those having a few disjointed projects.

For those having dozens of projects, and where each reuses substantial pieces of code, which are subsequently refined and retuned, branching and merging is really useful. This would enable you to start one project as a copy of another, while keeping the history of the older into the new one. Let’s face it, we all make bugs, so when you fix a bug you found in the inherited code, you would like to back-port that to the older project as well. Using the Asset Server, there is no way of doing this besides copying the files directly, which will make it lose its version control history.

The Asset Server also has a few quirks that you should be aware of. Many Unity developers rely on AssetPostProcessor scripts to add setup scripts or colliders on certain named objects in your models. If you do, you should

SCM/VCS

A version control system (VCS) or a source code management (SCM), or even a software configuration management (also SCM) system, is a piece of software that keeps track of who changed what in source code, assets, and/or other files in a project.

The common functions of an SCM can be summarized like this: Each user has a local working copy of the project. Changes are shared with other users. Each user can get one or more changes from other users at a time. The SCM helps in merging differences when two or more users have changed a file.

Common VSC/SCM systems include: CVS, Subversion (SVN), Perforce, git, Mercurial, BitKeeper.

terra vision : Erling Skakkes gate 49B 7012 Trondheim post@terravision.no www.terravision.no +47 4000 3836 org.no 989 199 064 mva

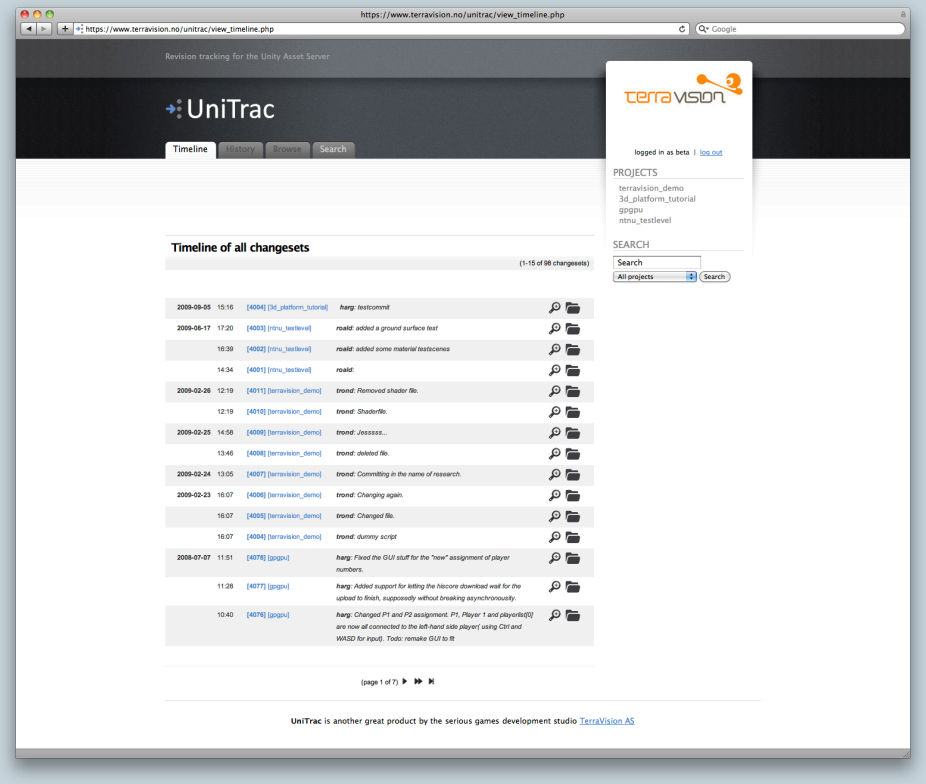


Fig. 1 – Unitrac, an internally developed web interface for the Asset Server

know that changes made by the post-processing script are not recognized by the Asset Server client. Each team member will have to reimport the model themselves to have the changes applied. When using native 3D model formats, this requires the modeling package to be installed on all machines, which is usually not the case when you have dedicated programmers and artists on a project.

This can be alleviated by sticking to FBX as your 3D model format. You can also trick the Asset Server into accepting a new version of the model, by making a small adjustment to the import settings, commit and have all team members update. You can then reset the import setting and repeat.

External version control

Since version 2.6, it has been possible to have Unity Pro set up special .meta files for each asset in the project. This enables standard, external version control systems to pick up almost all changes that happen inside Unity, by saving import settings and other meta-data to text files which can be committed, updated and diff-ed (see Fig. 2). Using an external SCM means you will have to rely on other tools than the internal Server panel to get updates, commit changes or view diffs and conflicts, but for many the benefits will be greater than the perceived cost.

Support for external version control systems means Unity users can now use industry standard SCMs like Subversion or Perforce, and perhaps even more importantly, the third-party tools that these systems can be integrated with. A Unity-using organization can choose to keep its existing SCM, or decide to pick a new one which has desirable features, and use it for both Unity- and non-Unity related work.

This simplifies introduction of Unity in established software development organizations. It also gives added value to developers using Unity, through the use of third party tools for project management, bug tracking and SCM administration.

The most common version control systems used with Unity are probably Subversion (SVN), Perforce and distributed systems like git. These all have different strengths and weaknesses, which are mostly unrelated to Unity, but for the sake of completeness, we will have a brief look at their main differentiation points.

Subversion or SVN is one of the most widely used version control systems, and is a free and open source software from CollabNet and the Apache Software Foundation. The open source community has long been supporting SVN, so a long range of mostly free utilities exist that integrate with SVN server repositories.

This includes popular tools such as Apache web server access, Trac bug-tracking and web interface and most software project management systems. SVN also has full support for branching and merging, and both well-designed graphical user interfaces and powerful command line tools are available. While free for open source use, and commercial teams with up to two members, Perforce is almost as pricey as Unity itself for larger teams.

However, it has become the standard SCM in many parts of the games industry. As Unity has gained traction with existing game development studios, the use of Perforce with Unity is also increasing. As a commercial product, Perforce has a very high level of support and maintenance of its software, and takes pride in being both flexible, very scalable and having the best tools. It has better cross-platform visual tools for repository administration than most systems, and it integrates directly with virtually all project management, bug tracking, build support, and testing tools. Perforce naturally has all the functionality you can expect from a modern, server-based SCM.

Both SVN and Perforce are based on a client-server model, which means each user updates from, and commits to, a single central repository.

```
fileFormatVersion: 1
guid: d31720323f9d14240aa0bd92c987ea6f
ModelImporter:
  importerVersion: 4
  meshes:
    scaleFactor: 1
    meshCompression: 0
    generateColliders: 0
    swapUVs: 0
  tangentSpace:
    generation: 0
    calculateNormals: 0
    smoothingAngle: 60
    splitTangents: 0
  materials:
    generation: 1
  animations:
    generation: 3
    bakeAnimations: 0
    animationCompression: 1
    animationWrapMode: 0
    splitAnimations: 1
  nodeInfo:
    AirTrap:
    - 100000
    - 400000
    - 3300000
    - 2300000
```

Fig. 2 – Example of a .meta file for an FBX model asset

This usually works great, but in some cases you want each developer to have more control over the version control process, and even the ability to have versioning of local work, or when not connected to a network. In recent years then, there has been a small surge of new SCM systems that are based on the idea of distributed version control, among these are git, Bazaar, Mercurial and BitKeeper. These generally give each user a local SCM system which keeps track of commits on the user's own file system.

They also enable the user to connect their local system to other users' systems, or even a central repository, and if permitted, be able to push their local updates to these other systems. These systems generally are more lightweight than SVN or Perforce, but still have much less support for third party tools and management systems. There is also a general lack of cross-platform, user-friendly interfaces for these SCMs, so you will most likely have to familiarize yourself with the command line syntax, to use these tools effectively.

Problems, pitfalls, and pains

Whether you choose the comfort of the Unity Asset Server or the flexibility of an external version control system, you are likely to meet some puzzling and potentially headache-inducing issues. We do not want you to have unnecessary headaches, so we will try to describe some of the more common and painful pitfalls you are likely to encounter, and give you some hints on how not to hurt yourself.

The first, and most common, issue when collaborating using Unity, is conflicts in scenes, prefabs, and other internal data files. Since these files are stored in a proprietary, binary format, no SCM (not even Asset Server) is able to handle these files properly. They cannot show the differences between two versions of a scene file, or merge your changes with mine, when you update to my latest commit. You are then left with the option of keeping my version or your own, modified version, of the scene file in its entirety.

Most smaller teams solve this by either just shouting out in the office "the level3 scene is mine now", or use built-in file locking in most SCMs. Some teams go so far as to deny all but one or a few people the right to commit scene files altogether. This can get quite complex when you have larger projects with more people, especially considering that prefabs suffer from exactly the same problems as scenes.

A somewhat better solution, is to ensure that your scenes are actually very minimal, to avoid frequent updates. You would rather maximize the use of post-processing and scene setup scripts that instantiate prefabs, add scripts and colliders to models, and set up the scene as needed. The setup scripts are stored as text, and are thus diff-able and merge-able for the SCM. This can however lead to depreciation of the built-in editor functionality of Unity. To still use the visual editor, you will need scripts to convert in-editor transforms and setup to snippets that can go into your scripts, or you will have to rely on setting up everything in script alone.

Even braver Unity users can take this idea one step further, and make your own, text-based scene and prefab file format. This requires all team members to stop using the ordinary scene format, and rather build the scenes procedurally from editor scripts. A starting point for such a system was recently released by TerraVision as an open source project at GitHub (http://github.com/terravision/UnityTextScene).

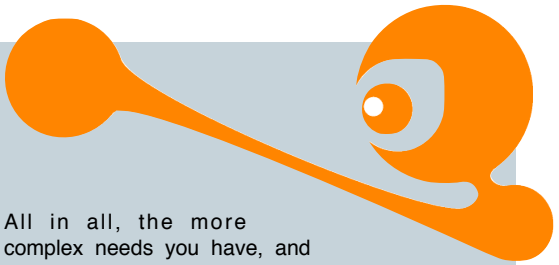
Pain can also come from using application-native model formats. As we noted in the section on the Unity Asset Server above, you can easily end up with local changes that are not detected by the version control system, and are thus not committed. For projects where some users do not have the modeling package installed, and you expect to have a certain amount of AssetPostProcessor scripts, it is probably recommended to stick with FBX model files.

While the support for saving .max, .ma, and .blend files directly in the project is one of Unity's party pieces, the ease of working with universally acceptable files such as FBX often outweighs the troubles of manual export. When using external version control, each Unity client is itself responsible for importing the native model files, since the imported models are not committed. This means all team members will need to have the modeling package installed, if you use anything but FBX files.

Likewise, and this goes for FBX files as well, if you make changes to materials generated by model import, these changes do not propagate to other users. That is, unless you explicitly make an editor script that checks for and prefers existing material files upon import.

A special case of puzzling behavior can come from using external version control systems and updating a project that is currently open in Unity. Seemingly because of the order updates are processed in, instance prefabs which contain references to new texture assets will likely lose the texture reference. If the Unity editor is closed while updating, all files will be available when the prefab is processed, and the references will remain intact.

The Unity editor brings with it another potential pitfall, when using external version control systems. If you rename or move any asset in the Project pane, your version control system will most likely fail to recognize this. You will be left with one file marked as deleted, and another as being brand new, and the history of the former file will be gone. Therefore, you should always move or rename files using the version control system, and not the Unity editor.



All in all, the more complex needs you have, and the more you want to use the finer features of especially external SCM systems, the more likely you are to be wanting more advanced editor scripts, special AssetPostProcessors and customized tools. This will require a bit more from the programmers to set up, and some more considerations from all users, but is well worth the efforts, to get the most out of Unity as a collaborative tool.

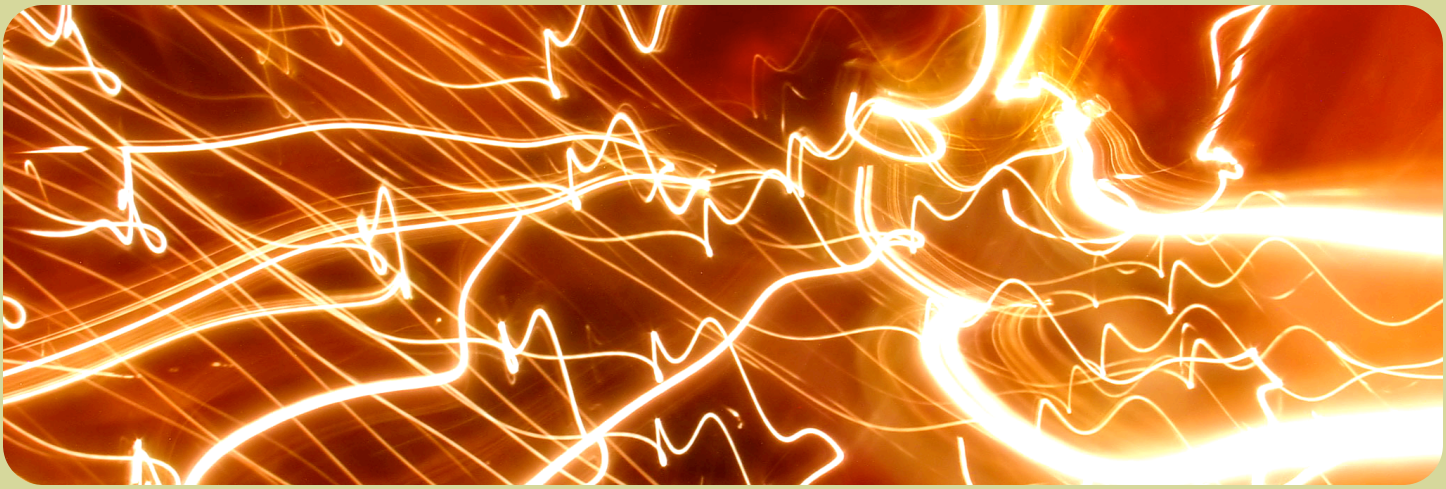
Conclusions

We have seen how a team working on a Unity project will benefit from using a software configuration management (SCM) system for version control. As a Unity Pro user, you can choose between using the integrated Unity Asset Server, and various standard, external systems, such as Subversion, Perforce or git.

The Asset Server is great for simpler use, but lacks in support for third party tools. External systems require user interfaces outside Unity, but are much more powerful, flexible and integrate with a range of tools for bug tracking, project management, build support and test administration. Regardless of your choice in SCM system, you will have to tackle problems such as diff-ing and merging Unity scene files and prefabs. With the right system for your organization, and perhaps some custom scripts to make your workflow a little easier, you can get Unity to work pretty well in even quite complex, collaborative projects.

effective marketing

By Manuel Saint-Victor, M.D.



As Indie developers, we are often dumbfounded as to when and how to start marketing. Starting too early doesn't seem to make sense, because when a game is in its infant stages as an idea, there is not much to market.

Furthermore, once it is time to market- what do we do then? What's the approach? How do we make the sale? How do we keep an audience interested and in the long run, provide value and nurture a mutually beneficial relationship?

In order to effectively market our games, indie developers must rely on techniques that have proven to produce greater return on investment. Techniques such as Inbound Marketing, Buzz Marketing, and Guerrilla Marketing focus on out-thinking the competition instead of outspending them. Inbound Marketing, as described by Hubspot founders Brian Halligan and Darmesh Shah, focuses on creating a Hub for your industry or desired niche.

By providing resources and building a community that comes to you, you are much more effective than the more time-consuming hunting, which inevitably leads to the prey fleeing. Guerrilla marketing emphasizes eye-popping, unconventional methods of promotion like running naked through the street with a logo sign covering your privates." Buzz Marketing, as described by VP of Half.com Marketing, Mark Hughes, derives its strength by getting people to do the talking for you by giving them something remarkable to talk about.

These the and associated emerging marketing philosophies can not only increase your support base and brand equity but also transform communities into social movements.

Market your brand - not just a single game

There are numerous advantages to marketing your brand rather than marketing a single game. First, your first game is likely to be only a ghost of the products that you develop later when you have mastered the craft. If you're great now, imagine how much better your 30th game will be. To quote Picasso, "It might've only taken me 30 seconds to draw this masterful sketch but it took 30 years to get to where I can do this in 30 seconds." Second , by marketing your brand, you begin to grow a relationship with your potential buyer and hopefully invited them to join in the benefits of your community.

Third, you are better able to quickly exchange information feedback on your next games and products. Last, you're also able to help others who are offering entertainment products by leveraging your credibility and endorsing them. This approach will provide longevity and allow your most recent success to be a catalyst to boost its co-brand members.

Be different

We've covered what to market, but not necessarily how to go about doing so, nor what your message should be. One of the first steps is to ask yourself: what's going to generate the most interest? What is unique with my game? What would make someone tell another person about my game? Start early, the marketing process should begin the moment you start designing your game. Determine what is actually going to pique people's curiosity. These things will be what you should focus on in your

design. Once you've nailed what makes your game standout, you can begin driving that message home. The imminent release of Star Wars Trench Run 2.0 by Infrared5 is a great example of this.

They've created a way to control the Unity Web Player game (which will be hosted on starwars.com) via the iPhone version of their game. The technology they've created to do this is called Brass Monkey. Using a mobile phone as a controller for a web game is not something anyone has seen before, and this has already started a buzz before the game has been released.

While Star Wars Trench Run 2.0 also includes other features like new levels, better effects and the ability to switch to different ships, Infrared5 is focusing on the Brass Monkey feature of the game. They are pushing the message that the future of the web and mobile is combining them in unique and compelling ways.

Figure out what's different in your game, and begin focusing in on that message in your own marketing.

Of course this same "stand out from the crowd" concept applies to your brand as well. Make sure that you get the message out there as to what makes your overall brand unique. Hone in on just those things, and you will see results.

Start marketing immediately

Start right now. Drop this article and get to it! Seriously, you might want to finish the article, but keep that type of urgency about marketing. You don't want your business to die a silent death.

Since we've already established that you're going to market your brand, we no longer have the issue of not having anything to market.During the months that you are working on your game, consider sharing your characters, settings, and so on. Don't worry about people stealing your prized idea because by virtue of putting a little bit out there, you've already taken the first steps to owning and generating your buzz!

As you build a game or two and get some visibility, it becomes easier to create a sense of anticipation for your next game. Visualize working the audience into such a frenzy that at midnight when you open the gates they melt your server. This requires that you've teased them with screenshots, character backgrounds and other tidbits to get them attached to and discovering the characters. Since you've grown enough brand awareness that they know what kind of product to expect from you, they'll want you to hurry up and deliver the goods.

Prepare for Success

As suggested in a discussion with Yilmaz Kiymaz (@VoxelBoy on Twitter), "consider having an active blog & forum for your game" where the community that forms around your game can obtain new information and also discuss the game amongst themselves. Update with relevant content and share your experiences building your game. Wolfire games is an example of a project that found success with this approach. They have a rigorously updated blog about their upcoming game, full of development-related information and articles regarding different aspects of game development (story, characters, gameplay mechanics etc).

Share notes and tips that may help others avoid your pitfalls. Another thing that comes to mind is to develop a relationship with popular games bloggers or people from game portals and have them post news about your game as development progresses.

Also take the time to get to know how you're valuable by helping share awareness of your favorite content on their site or event they are planning. Have stats software in place so that you can understand which articles resonate well with your audience. Have your Twitter account set up and begin finding and listening to your target audience.

Build your games with marketing in mind

Use social media libraries like dimeRocker as a means of helping your users engage their own friends across multiple social platforms to get them more immersed in your game.

Be sure to do this intelligently - when everyone goes for the same models, they become background noise. Users ignore them, or worse, they negatively impact your brand and subsequent products. Make it a win-win for the invited friends. Have both parties involved benefit with a micro-reward. Consider coming up with a way that enables invitations to have slightly different benefits based on the rank of the players involved.

This allows for subtle bragging to their friends every time they advance and invite them from a higher tier with more fringe benefits. The idea is to come up with approaches that don't sound like every other invite and both capture and sustain the invited parties' attention.

Manage relationships across all interaction platforms.

To maximize the audience that you are attracting, you need to cover Facebook, Twitter, YouTube, Linked in, Digg, Reddit, and Delicious - and that's just naming a few. Build your pages early. Take the time to learn how each community operates and find your friends in those communities. Don't be the sleazy sales guy.

Maintain your integrity at all times. Despite the appearance of these being separate systems, news of the smarmy salesman knows no boundaries and will . Grow your relationships by getting to know people in those networks. Carry them even further by inviting folks to stop by and

continue a conversation in another network if it's a better medium. If you're in the middle of a good talk with someone and you need to show them a video, send them a link to your YouTube channel with that video. Subscribe to their feeds during the talk and get to know their work. Vote up some of the stuff that you like and favorite some.

This should be done tastefully. Nobody likes a kiss-ass! Take the time to get to know their work well enough to be able to send others when interest and goals match.

Allow your users to enhance the games. Share these free enhancements. Just about every game nowadays allows the user to put her touches on the game and then (here comes the important part) share it with friends. Not the game, but her customization. The need to share their creation can be the fuel behind her action. Let your game come along for the ride.

Action in this way creates chatter. Chatter creates traffic, and traffic creates interest. By "interest," I mean that Uncle Google gets a sense of what people are interested in, including you. Many fishermen stopping by or discussing your site clues Google in that you might have something to do with fishing.

This gives you a boost when the hardcore fisherman decides he wants an iPhone game or Facebook game.

Get to know and understand your target audience. Immerse yourself in their world. If you anticipate selling games to fishermen or to skateboarding fans, then go to their most active sites and become an interested and involved participant. Make sure your interest is genuine and be sure to learn the culture of the environment before you post anything -- the only way to get street creed is to know the streets! Make sure that you are engaged and helpful for several months before the topic of selling your goods comes up and that you remain as involved in the community as long as you want to maintain your brand - which is forever! As Tadej Gregorcic, Co-founder of Motiviti Games pointed out to me, this also requires choosing a narrow enough segment so that you can become knowledgable in its thinking. Use demographics, theme, geographic location, and any other means to attract a small niche community that will be able to grow interest in your game into the general population. Think Guitar Hero, Band Hero, Lego Rock Band, etc.

Make your characters deep and with a rich enough history for folks to actually care. Create characters that people actually care about. Even if caring means they hate their guts! Take a hint from soap operas: It's the same story told over and over...and over.

It lasts for decades, but you can count on the characters to remain just as despicable no matter how many chances you give them. They'll keep making the same mistakes. As Calypso said in Pirates of the Caribbean, "Would you love me if I were any other way?"

How do you know people care about your characters? Well, that's one of the benefits of giving them a taste trough social media and the blog.

Muse Games put out some sneak peeks of their characters from upcoming games. Come up with a sideline story that involves the characters and see how people react.

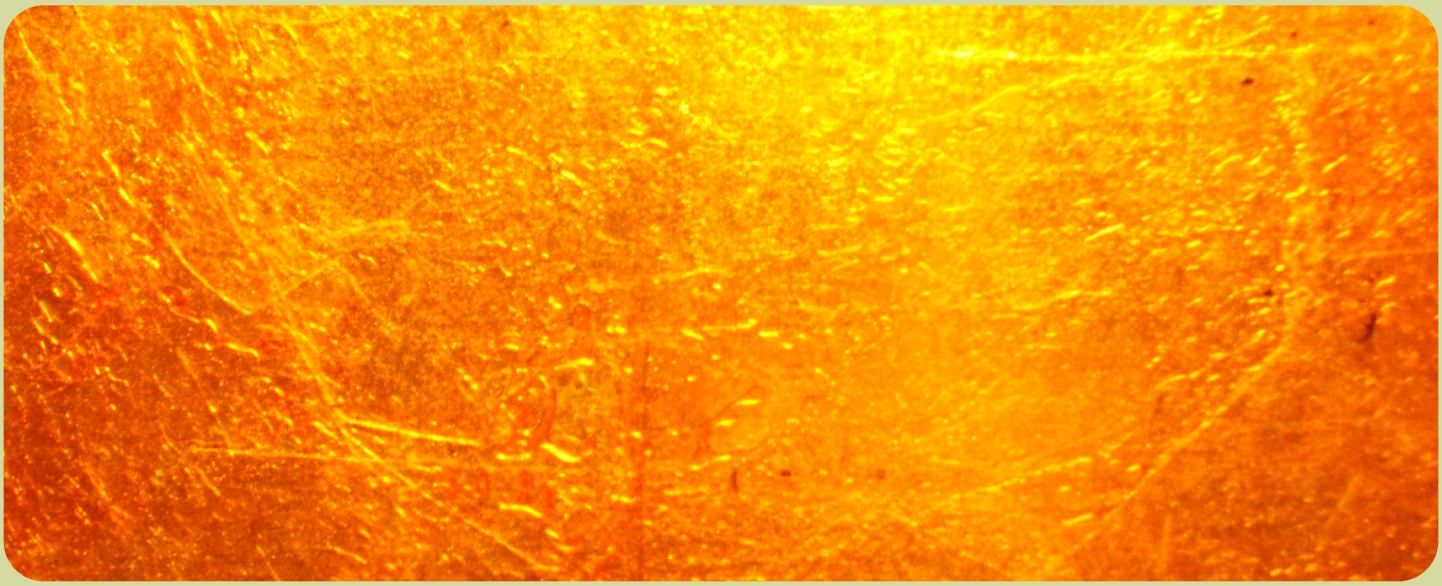
Do your own background research -- growing up which characters caught your imagination? Why did they catch your imagination? Also, remember that everyone likes characters that aren't as "real" as those you may meet in every-day life. Have some fun by exaggerating some traits.

Market in real life and where there's less noise. Give away T shirts with an attractive screen capture of your game, featuring the game's name in a prominent position, to friends. Encourage these friends to wear these T shirts to places where people outside your circle interact casually such as at the mall.

Take it further and have enough friends who don't know each other wearing the shirts in one area. Coordinate so that their traffic patterns create the appearance of increased popularity of the game instead of 2 guys from Marketing walking together.

Repeat this exercise with different people in your local community. This increased awareness, paired with the correctly targeted Facebook ad, is a much more powerful combination than either alone. Also keep in mind that as applications start to use more location-aware features that you'll have have advantages in searches and in visibility as users limit search to nearby areas. As Google hi-lights social search the size of your network will influence the likelihood that people will see your content. These are small, discrete factors that can combine to make a larger impact.

Get famous!
I'm being dead serious - yes I am. Grow a fan base and become a local celebrity. Help with community events and grow your audience that you can use as currency added to that of someone who may have a larger audience to share with you. Many people call it building a community, a tribe, or something else. When you move from site to site, it's vital that you don't have to recreate and regrow your support. The Internet enables you to have a portable team and make multiple arenas. By finding your group in each network you're ready to gain visibility when it's time. Your videos are in place on YouTube, there's mention of you at various sites and you're placing searches. As more people get



to know you from your game, you can move to someone else's home base and bring your community. This translates to more business and visibility for their game or product. That's why celebrities get free stuff - so that they can mention it and the company can get much more visibility. And as we all know, visibility translates to sales.

Participate heavily in panels that can showcase your work and your expertise. Make sure to give credit to the tools that you have used to get where you are with your product. If you prototyped with Protopack and it saved you hours, let folks know. This is how endorsements happen, and you help other Indies rise along the way. I think of it as a social responsibility.

Consider speaking at conferences and sharing your expertise as a means of nurturing your connection with your audience while growing your audience. This is a perfect time
to hear what they actually want you to share. You also gain the benefit of having several others with large audiences and instant cross-exposure.

Recycle!
When one of your games gains some visibility, you might want to consider boosting its sales by throwing in a free copy of another game. This game may have had some success in the past but not gotten the visibility it needed. If one of your games has some success, use the character in a new, similar game and increase sales of the previous game while giving the new game a boost. Lucas and their Star Wars brand have been doing this for over 25 years. How about Final Fantasy? They have like 13 releases at this point. Rock Band is another great example.

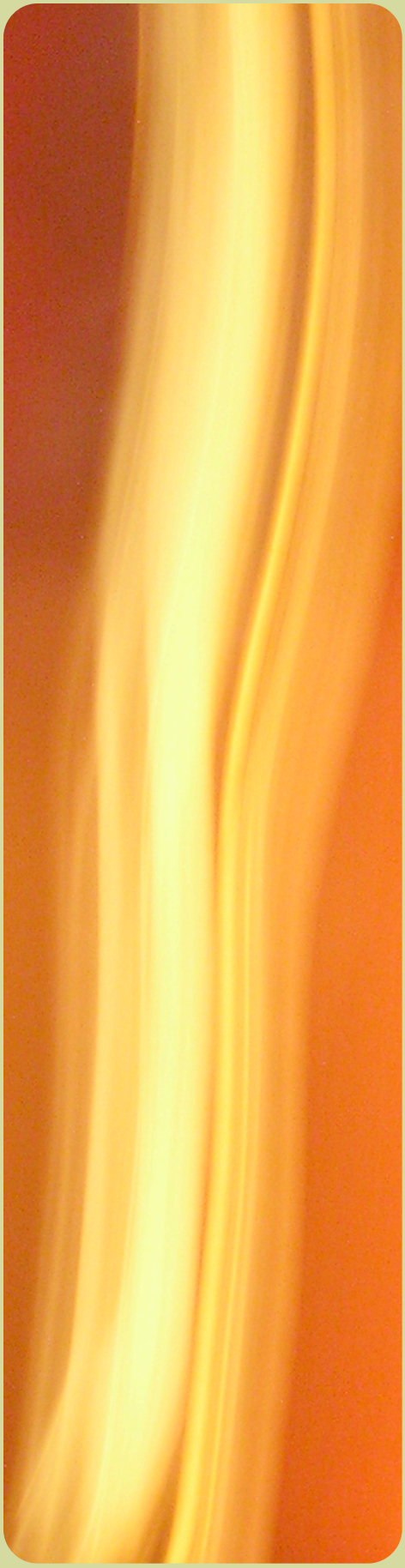
Network
Get to know others who are "coming up" just like you and put everything that you can into helping them get there. This requires that you find products that you like as well as people you enjoy hanging out with and nurturing those relationships. Certain things (like true interest and integrity) you can't fake and the second that people sense betrayal, the damage is hard to repair. Think about Tiger Woods or Toyota. Provide feedback, exchange resources, and refer talent.

Attend events like the Game Developers Conferences and SIEGEcon so that you can go from being a virtual friend to a real friend. There is a lot to be said about a real phone conversation and even more about a single face-to-face meeting. Indie game developer, Elliott Mitchell, known to many on Twitter as @MrT3D, has the following advice:
"Go to Events like GDC and PAX East to network, learn about marketing and show off your game on the expo floor. A group of game developers affiliated with the Boston Indies (Macguffin Games, Fire Hose Games and Dejobaan Games) made huge gains by effectively advertising / marketing their games at PAX East. They allowed the 60K attendees to play their games and networked with them in the expo hall. The Boston Indies also spoke on panels as often as possible to become the charismatic faces of their indie brands. Join and attend IGDA or similar industry networking events as well. In Boston we have an IGDA Chapter known as the Boston Post Mortem. IGDA board member, Darius Kazemi (@tinysubversions), organizes the Post Mortem along with a few others including Scott Macmillan (@mrmacguffin). Scott is also the primary organizer of Boston Indies group. It's largely through these groups I've gained knowledge and advice on making and marketing games to consumers and publishers."

Rinse and repeat
According to Buzz Marketing, by Mark Hughes, people see 1 in 3 of your "messages" and the average individual needs 9 exposures to your product before they actually move to make a purchase. That's 27 times that you have to be in a person's face before they'll buy in. Learn from your errors instead of considering them failures. Remind yourself that success might be just one attempt away but if you give up it's right here-right now. One of our greatest sales-persons, Zig Ziglar said "Remember that failure is an event, not a person".

These tips barely scratch the surface of what it'll take to launch and maintain an effective marketing campaign, but I hope they've sparked some ideas. What's important is to understand that if you're picking up some marketing skills as you develop your game, you can give your games a much better chance of being successful. We don't have to start from scratch when trying to figure out how to increase the spread of our games. Marketing is being done every day right in front of your eyes. Look closely and take time every day to pay attention to how it's taking place. You'll be able to use these techniques for your own benefit and for those in your community.

I also want to thank the team of folks that took the time to read this and help me think through it including Alex Shwartz (@gtjuggler) who has had to sit through several revisions of the site as well. Dr Akua G Asare and J Shakir Ramsey, and Claudia Vance from Gamr7, makers of Urban PAD, for grammar and style feedback. Elliott Mitchell and Yilmaz Kiymaz for letting me steal their words and wrap quotes around them. Tadej of Motiviti Games and Mike Derbyshire from dimeRocker their feedback and many others for the feedback. Most importantly, thanks to the Unity community for your support.



dimeRocker

SELF-PUBLISH & MONETIZE NEXT GEN SOCIAL GAMES

WHAT IS DIMEROCKER?

dimeRocker is an innovative online self-publishing platform enriched with turn-key features that make deploying your games to social networks quick and easy. We provide you with features that are often only present in widely successful social games, without the need of a web development team. dimeRocker offers fully scalable hosting solutions that let you quickly and easily integrate advanced features including:

LEADERBOARDS MARKETPLACES COMMUNITY RESEARCH & FEEDBACK
SOCIAL NETWORK HOOKS: INVITES, WALL POSTS, REQUSTS...
...AND MUCH MORE!

dR TUNE-UP PROGRAM

SUBMIT YOUR GAMES

IN THE FORM OF A PROPOSAL, PROTOTYPE OR FINISHED GAME

SELECTED GAMES COULD WIN A TUNE-UP INCLUDING:

- MONETIZATION AND VIRAL STRUCTURE ADVICE FROM AN INDUSTRY EXPERT.
- VIRAL SEEDING BY DIRECTING 10,000 USERS TO THE GAME.
- dR MONETIZATION ACCESS AND INTEGRATION ASSISTANCE.
- EARLY ACCESS TO dR ANALYTICS.

MORE DETAILS AT DIMEROCKER.COM

Facebook Facts

★More than 400 million active users

★50% of our active users log on to Facebook in any given day

★Average user has 130 friends

★People spend over 500 billion minutes per month on Facebook

★ More than 70 translations available on the site

★ About 70% of Facebook users are outside the United States

★ Over 300,000 users helped translate the site through the translations application

Unity and Facebook: Problems and Solutions

By Bob Ricci (Programmer of Crash Derby)
http://apps.facebook.com/crashderby

This article will discuss solving the most common problems that Unity developers can run into when tying their games into Facebook. When setting up a Facebook app to run a Unity game, there are a few pitfalls that seem like a deal breaker to most. These are the problems that we'll be addressing and discussing the best implementation for overcoming. I will be breaking these sections off into titled parts.

We will assume that you're already familiar with basic Unity scripting, basic Javascript, and that you are also familiar with Facebook programming on no more than a beginner level. I'm not going to get into the specifics of Facebook programming since there are many approaches and programming language preferences, and I want to keep this article applicable to them all. If you can do as much as pull a user's ID or name, it will suffice for this article. If you're not familiar with this process you should find and take a quick tutorial and come back. I'll wait. The main aspect that I'll be focusing on is setting up a solid method of communication between Unity and your HTML Facebook application page. Once that framework is down, your possibilities are endless.

A little about my experience with Unity and Facebook:

Crash Derby is a demolition derby game I developed under Eek! LLC (www.eekllc.com) with Ben Hanken. It is in Beta right now on Facebook, and will eventually be released for iPhone. Some of the features (so far) include syncing with your Facebook account and keeping track of your progress and development over time. You can earn "Crash Cash", experience, and levels that help you unlock new arenas to play in, and also to buy new parts for your car to enhance performance. Your friends names are brought into the game as other drivers that you compete against, and quirky comments are produced by them depending on game play. We have a leader board and stats profiles for all of our players as well. The game has the ability to post on players walls and they are able to invite friends to play the game and hire them as pit crew members for in-game bonuses.

When I created Crash Derby, I ran into a lot of problems with Facebook integration, and finding help or documentation on how to go about dealing with them was really difficult. This is why I thought that addressing those really annoying snags that can make or break a project was a good way to start off writing about Facebook and Unity integration.

Let's get started, shall we?

CRASH DERBY

Demolition Derby game coming soon from Eek! LLC. Play the BETA today!

Part I: Displaying a Unity Player on Your Facebook Application Page (FBML vs XFBML)

The first and most apparent problem is the player itself. Facebook has its own markup language called FBML. FBML is a great way to utilize the tools Facebook offers, and if you've ever worked with Facebook applications before, the preferred method. Unfortunately, FBML doesn't support embedding a Unity player. There are two ways to approach this. The easiest solution is to simply convert your application to a Frame canvas in your Facebook app settings. In doing so, you've now lost access to all of your FBML functions throughout your entire app, and you'll have to use XFBML (a version of FBML built to operate in frames or in applications) to do anything on your page. Unfortunately, there's no good way to avoid using XFBML when you want to have Unity interacting with Facebook (as we'll get into later) but that doesn't mean you have to sacrifice using FBML on other pages of your application that don't have the player on it. Or maybe you just want to host your game on your Facebook page as is without any in-game Facebook integration. In this case, a better way to deal with this would be to use Facebook's little known iframe object: "<fb:iframe>". By simply placing an iframe object on the page with the src property set to the url of your Unity Web Player page, you can render your Unity web player on the page in a frame and still keep your surrounding FBML objects intact.

In Crash Derby, I actually ended up going with the iFrame canvas method. I did this because my game has in-game interaction with Facebook, and uses the player's friends' names as well as posts high scores and level achievements to the user's wall. However, Crash Derby is a multi-paged app. We have user profiles and pages where users can invite other users, and it's very beneficial for us to use FBML on some of these pages. The way I accomplished this was another little known trick. By adding "?fb_force_mode=fbml" to the end of your url on any of your iFrame canvas Facebook pages, you can force the page to use FBML instead of XFBML. So any other pages you'd like to add to your application in which you'd like to use FBML, you can do so.

Part II: Sending Messages to Your HTML Page from Unity

Assuming you're going to be sending information back and forth, you'll need a way to send messages from Unity to your HTML page. Fortunately, Unity has a solution for that built right in. Unity has a function called Application.ExternalCall(), which allows you to send a message from the Unity Web Player to the browser document it's contained in. The function will run the Javascript function named the same as the argument you provide to the function. For example, if you want to run a Javascript function on the page called "GetUserName", you would simply call that from your Unity code like this: Application.ExternalCall ("GetUserName"). Say it's a particular user's name you're interested in. No problem. This Unity function supports parameters as well. Application.ExternalCall("GetUserID", "13") will call the Javascript function GetUserName, and also pass the function the argument "13". This could be a user ID, or literally any other data that the function might need from Unity to operate correctly. Your function on your html page might look something like this:

```
<script language="javascript1.1" type="text/javascript">

    function GetUserName () {
        GetUnity().SendMessage("CallReceiver", "GotUserInfo", userName);
    }

</script>
```

We'll come back to this code later since this is also the way we'll be sending messages back to Unity. But the main thing I'd like you to note here is that the function name is "GetUserName". Calling Application.ExternalCall("GetUserName") from Unity will execute this function.

Part III: Sending Messages to Unity From Your HTML Page

Let's take a look at the code we just saw and specifically look at what the function actually does.

```
GetUnity().SendMessage("CallReceiver", "GotUserInfo", userName);
```

GetUnity, a Javascript function that's included in your HTML page, is generated with your compiled Web Player build from Unity. You'll want to make sure you leave this function in your HTML page when you publish the page and Web Player. After the GetUnity function does its magic and retrieves the unity player object for us, we simply use the SendMessage function to send a message back to Unity. In this case, we're sending a message to the GameObject named "CallReceiver", and we're telling it to run the "GotUserInfo" function, and passing it a variable called "userName". I usually create a GameObject called "CallReceiver" in my main scene, and call the DontDestroyOnLoad(this) function in the Awake method to keep the object alive throughout the course of the game so my communication channel to Facebook stays open at all times. userName, of course, contains the name of the user that we request from Facebook. You should be starting to see now how all of this all works together. By sending messages back and forth, we can start to communicate with our unity project and load information to and from our Unity game back to our page.

Part IV: Putting it Together

Let's stop for a minute and put everything we've learned to practical use. Let's say we want to display a Facebook user's name on the screen. Set up a new Unity project, and create a GameObject. Name it "CallReceiver" and add a C# script to it.

Add the following to your script:

```
string userName = "";

public void GotUserName(string data)
{
    userName = data;
}

void Start()
{
    Application.SendMessage("GetUserName");
}

void OnGUI()
{
    GUI.Label(new Rect(Screen.width / 2 - 250, Screen.height / 2, 500, 50), userName);
}
```

That's all we'll need on the Unity end. Now let's set up your HTML page. Create a new HTML page and use the tutorial you picked earlier to get the Facebook app users's name. What we're going to do is write this name to a variable called userName. If you want to skip the Facebook programming for now, or you didn't take that turorial yet, you can simple assign a value to the variable yourself with the following code:

```
var userName = "Bob";
```

Of course this is cheating. Because essentially what you'd want to do is use your Facebook code to grab the user's name from the Facebook API and then assign that to userName, or simply write out the Javascript to the page dynamically, but I figured I'd cut you a little slack in case you're lazy. Since our goal here is communicating between Unity and the Browser, whether or not we have the actual data is irrelevant. Regardless, for those of you who are actually taking the time to get the user name from Facebook, you'll want to either assign that to your userName variable in Javascript, or if you're using PHP or .NET, you can simple create this whole block of code with your script and print it out to the page source.

Once that's finished, our entire code block on our HTML page should look like this:

```
<script language="javascript1.1" type="text/javascript">

    var userName = "Bob";

    function GetUserName () {
        GetUnity().SendMessage("CallReceiver", "GotUserName", userName);
    }

</script>
```

The next step would be to compile your game and then upload it to your web server. You'll notice if you play the project in the editor, nothing will display on the screen since Unity is not able to communicate with your HTML page when it's not being cased in a web player. You're going to want to build your game and target it for Web Player or Web Player Streamed. When unity creates your game's unity3d file, you will also see it's created an index.html file for you. This is the file that contains the code to display the player and also the GetUnity function I just mentioned. Note, that if you do not include this function, this example will not work!

The next step is to upload your unity3d file and your html file to your webserver. Navigate to your apps.facebook.com url that you set up for your game, and you should in fact see your facebook display name show up in the middle of the screen. Pretty cool, huh? Now this is a simple example, but from what we've learned so far we can essentially extract names and other information, bring them into our Unity game, and then while they're playing we can hand that information back to Facebook to either edit components of the page or to display end results of the game, etc. We can also do things like import names of friends or any other information we might want to use from Facebook to personalize the game more.

There are definitely more advanced ways to communicate with Facebook, and my favorite would be using web services to send and receive information directly to Unity from your Facebook app. This is a more advanced tutorial however, and would probably require an article all in itself. Crash Derby actually uses a combination of the methods you've learned here, web services, and also has a database back end where it keeps all of the individual user stats. I would certianly love to get more into how I developed the game, but as I mentioned before, it's way above the scope of this article. Maybe next time?

Until then though, there's one more thing I'd like to touch on that is a standard element of most Facebook games, but not so obvious to implement seamlessly with Unity. You might get a little lost at this part if you're not as familiar with Facebook programming. So now would be a good time to dig a little deeper into those tutorials and learn how to post on user walls using Facebook Connect.

Part V: Writing to the Wall

I've seen a lot of people throw their hands up in the air as far as figuring out a solution to this one. The problem is not that it's a difficult thing to do, but in fact has to do more with the nature of how Unity works.

Normally in a web environment, we can layer elements over each other with a simple setting of the z-index CSS property. So popping up a wall post prompt over the Unity player with a higher z-index setting should do the trick, right? Wrong!

The problem is that Unity is rendering 3D directly to the window. The Unity Web Player defines a box that Unity is instructed to render to. This means that every time Unity calls a frameloop, it's going to fight with that prompt you popped up, and in most cases lose and wind up with your wall post dialog behind the Unity player.

Now, we could always show the prompt below the Unity player or above it, but that's kind of tacky, and we want to give the impression that it's happening as seamlessly as possible within the flow of our game.

The solution is rather simple. Remember that GetUnity function we have available to us?

That gives us access to the designated area in which Unity is allowed to render its content. So what if we were to temporarily tell Unity that it's render area was incredibly small? So much in fact that the user can't see it. Let's give it a shot.

First we need to write our functions to make Unity invisible and make it visible again. Here are the simple functions to do that:

```
function ShrinkUnity() {
    GetUnity().width = 1;
    GetUnity().height = 1;
}

function ExpandUnity() {
    GetUnity().width = 720;
    GetUnity().height = 480;
}
```

This is pretty self explanitory. We're simply setting the boundaries of the player to 1 pixel in the Shrink function, and then setting it back to its original value again in the Expand function. Note, that when expanding, you'll want to set the width and height to what your original values were. In Crash Derby, our game window is 720x480.

Now, let's look at the code to show a wall post prompt in XFBML. Here's the actual code I use in Crash Derby:

```
function WallPost(postText) {
        ShrinkUnity();
        FB.Connect.streamPublish(postText,
        attachment, actionLinks, targetID,
        messageToUser, postComplete,
        autoPublish, actorID);
}
```

Notice that we first call ShrinkUnity, and then we execute our streamPublish function which tells Facebook to display the wall post dialog. Most of these arguments I've given the function are irrelevant. They will be set according to what you want to display to the user.

The one of interest to us is the one that says, postComplete. That particular parameter of the streamPublish function tells Facebook what Javascript function to call when the user is finished with the prompt.

This is important to us because if we're going to make our Unity window invisible, we need to know when to restore it. So let's set up that function:

```
function postComplete() {
    ExpandUnity();
}
```

Pretty simple, right? We're telling Unity to expand when postComplete is called, which is called by Facebook when it detects the user is done with the wall post dialog.

The only thing left to do is to call the WallPost function and pass it the postText string. We'll do this from Unity using Application.ExternalCall ("WallPost", postText). It's also recommended that you pause the game by sending another message to Unity while you're waiting for the user to decide what to do and then resume it when you're done. I've already given you all of the tools you need to figure that one out yourself, so I'll leave you to it.

I hope this has been an informative article, and I'd love to write some more in-depth and advanced articles in the future about Unity and Facebook integration. Please check out Crash Derby when you get a chance, and watch for our iPhone release later this year!

Game Review

Have you ever been walking about, minding your own business, when all of a sudden a bird swoops by and uses you as target practice? You curse the bird, why would it do that on your new jacket? The answer, because it's just too fun to pass up.

While a pigeon is not technically considered "Fowl", it matters little because the game more than makes up for the slight grammatical error. "Tilt to steer, Tap to poop" Is it really that simple? Of course it is, and that simplicity is where the game really shines. As a spiteful pigeon (for reasons unknown) you will reign your white terror on the unsuspecting park patrons. Your objective is to dodge the trees and rack up the highest combo you can, using only your reflexes and the power-ups that are scattered about the map. Powers such as a Taco that allows you to fire your droppings in a more spread out radius. Not only effecting more than one target, but hitting the targets multiple times allowing that combo number to skyrocket. While using power-ups your combo will never decrease if you miss a target, so this gives you the opportunity to rapidly fire your death dukey as much as you like.

Another useful power-up is the football helmet. This allows you to charge the trees head-on and take no damage. Not only are you invincible in this state, but you also get points and increase your combo for every tree you destroy. Mind you, these are just temporary and wear off rather quickly. The icon will flash, warning you that the power is almost gone. You have three chances to prove your worth. For every target you hit you will increase your combo, but if you miss even one your combo will restart at zero. This also occurs when you hit a tree. Hit three trees and it's game over. As your combo increases so will the speed of the game and the arrangement of the trees, making it more difficult to steer the pigeon.

The presentation of the game stays true to the game-play. Graphically it is simplistically unique and fun to look at. The block headed people roaming the park are humorous to watch and even more so when they become victims of the pigeon. The grass floor is bland, but you will pay little attention to it as your soar through the air. Trees are delightfully square from leaf to trunk. Menus are vibrant and colorful and easy to navigate. Submit your scores online using the feint option. This opens your game-play to more options like, chatting, achievements, challenging friends, and much more.

Musically the game can be a little redundant like most iPod titles. The same upbeat cartoony track you have heard a thousand times. This would not be that big of a deal if you could simply listen to your preferred music on your iPod as you play, but the game does not support this feature. If you choose to, you may also mute the game.

Content wise you don't really get a lot, but hey it's only a buck to purchase and the one game mode you do get is extremely addicting; a great time waster. It would be nice to have a bit more variety though. I hope the developers release some updates!

Some people will get bored of this game quickly and others will fit it into their free time as much as possible. In the end you're buying an attractive and addictive title for the bargain price of 1 dollar. So, if you ever wondered what it's like to poop and fly at the same time, this game is for you. If you're looking for a solid time waster than you cant go wrong. I'm going to give this one…4 out of 5 ATTACK POINTS!

Dylan Hillaker - Freelance Writer and Avid Gamer

Game: Fowlplay
Developer: Happynin Games
Platform: iPhone
Genre: Adventure
No. of Players: 1
Price: $1.99

Rating (4 of 5):

INTRODUCTIONS

By Robert Morris aka Nightfox

One of the toughest things for us to do as Game Designers, is deciding on how to begin our players adventure.

Do we thrust the player into the firefight, giving clues as to whats going on throughout the battle? Do we simply have them living comfortably in there two room flat, ready for a days work, not knowing that they would be the unwilling victim of a heist!

What do know to be a cliche intro? Is it alright to use them, and if not, when do we avoid them?

Lets start with tackling the idea of cliches.

Cliches are plot devices that we've seen used over and over again. The orphaned child who grows to become the avenging knight. The surviving member of a squad of space marines fighting his way out from the depths of the enemy base . These and much more are an example of what we've come to know and call cliches. They are the cookie cutter plot devices that drives a story along. For writers, especially game designers, it makes story telling tough on us. We know how our players react to a situation. We also know that players have dealt with many similar plots before and after awhile, they can sense a cliche plot device being used.

How do we get the player to follow the same plot device, without alerting them to the story hooks and seeing it as a boring cliche? What can we do to make a cliche interesting again.

As Game Designers, we have a multitude of tools at our disposal to make the story progress without the player identifying obvious cliches. We have our world, the NPCs that live in it, and its history. With these tools we can then adapt a simple story to better immerse the player into the game.

Lets actually go over a few examples of story introductions, you may recognize a few of them.

-1-

If you can imagine the movie announcer describing the world, you've hit upon a known style of introduction. In a galaxy far far away, in a land ruled by an evil overlord, these should sound fairly familiar.

How is this beneficial to the player?

Not to long ago, all the history about the players current situation, could be gleamed from the instruction manual. As time went on, many noticed that gamers didn't really read the instruction manual,

most people would rather play the game, than suffer a few minutes reading paperback.

So by utilizing this style of cinematic introduction, it allows us as the game designer, to introduce and explain the world to the player. The players would then be able to see the world in an opening cut scene, which for all intents and purposes, sounds a lot more inviting than opening book and reading a few pages of history.

-2-

Dropping the player right into the middle of it all.

Jameson finds himself amidst heavy crossfire, his squad dead. Caught deep behind enemy lines, where heavy artillery from his own allies begins pounding the area.

This style of introduction is very common to a majority of action games. Sticking the player right into the action. By using the environment and npc dialogue such as radio chatter or other forms of communication, we can introduce our player to the world and why it is in its current state.

It doesn't even have to be a war zone, the player could be a new visitor to a town, when suddenly they are ambushed by a number of bandits. Once defeated, a guard comes out, and is glad to see the player is alive and well, telling them of the dangers of traveling through town at knight. Thus not only has the player been introduced to the world, but they are also introduced to their first possible mission which could be to destroy the bandits.

-3-

Intro via character building. This style of character introduction is a bit of a hybrid. the player doesn't start with a prefabricated avatar, rather they themselves get to affect what the character will be. Some games even allow you to change both the sex and the looks of the character as well as being able to modify the characters intro history. Where were they raised, their race, their mentor, are they magical, technological, and other traits.

This effectively allows our players to not only develop their characters history, but to also learn about the world or universe that the player will take part in.

Players can be offered a wide variety of options to create their player, thus making them generally unique in the fact that with so many options, no two players should technically have the exact same character.

How can we expand upon this?

Player traits and skills can be affected from the very start of character generation. As they learn the history via building their character, different skills and abilities are rewarded accordingly. A character born upon an industrial world would probably have a great understanding of business if they were part of the upper class, or an understanding of mechanics and engineering if they were part of the workers class.

With their characters generated, they would then be put into the characters daily routine in life, from here we can feed in the story, thrusting them into major conflict which would then move the player from the comfort of their virtual home, and force them to take necessary action to see the story to the end.

-4-

The tragic hero, he who has a moral reason for doing what must be done.

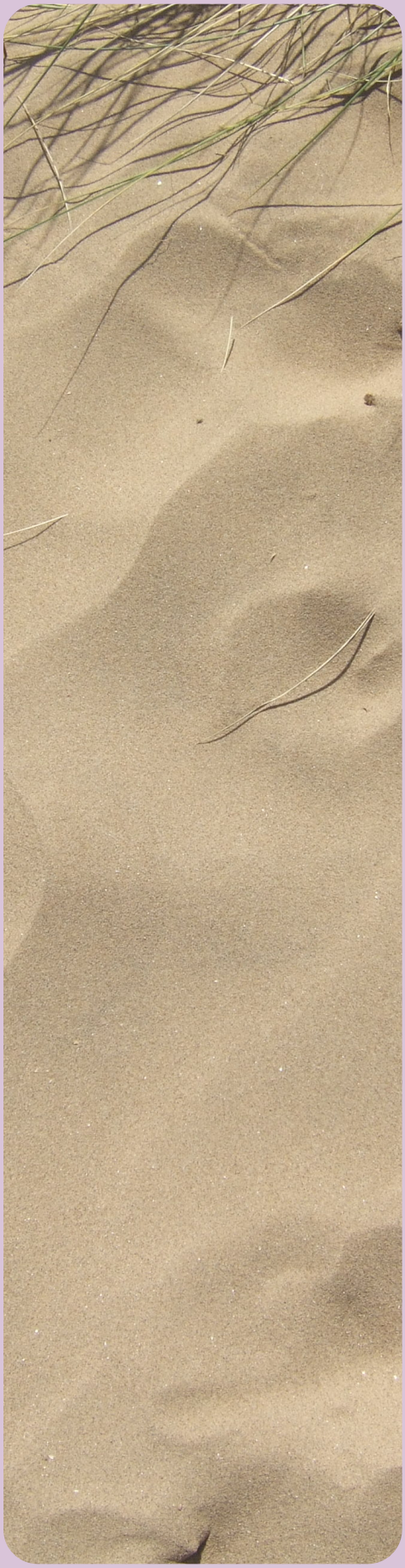
Jadeline, the daughter of an established family, watches in horror as her parents are brutally slaughtered by a rival lord. Her baby brother kidnapped and taken away.

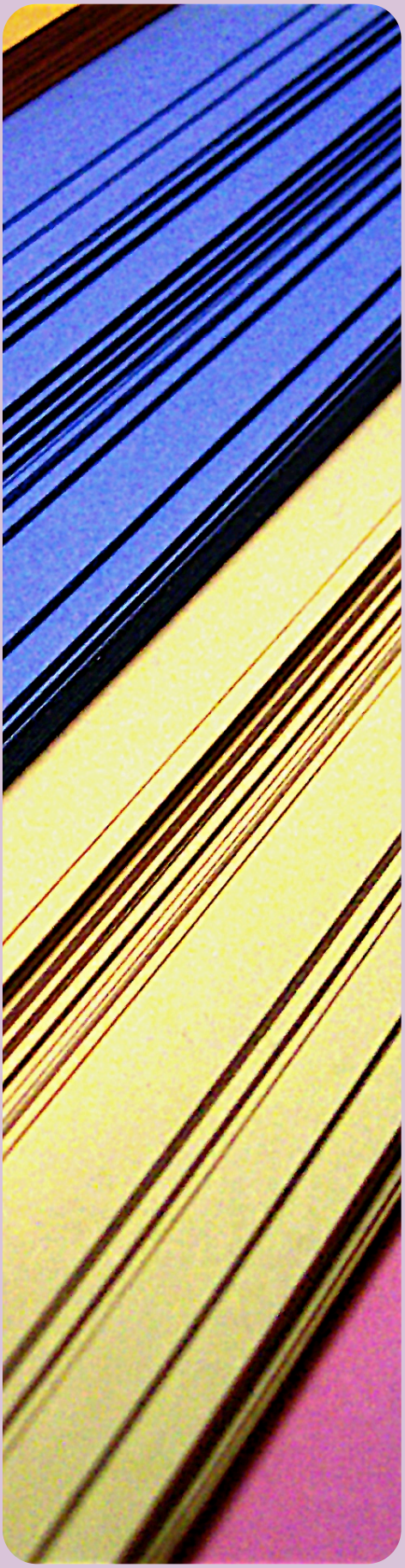
Here our players character has already had tragedy hit them. The player can learn this through external dialogue, either from the character herself, a journal, or gossip that can be heard amongst the NPCs that the player comes across, throwing the character into the terrible experience also helps them see it first hand and lets them understand the motivation behind it.

The tragic hero is a well loved and probably overused storyline. So many of us connect to a character who had it good in life, and then found themselves trampled into the dirt, only to rise slowly back to their feet and avenge themselves, their honor, or the people they lost.

Whether any of us have actually had an evil dark lord come in and destroy our family, I don't know, but it is a story we all like to take part in, nothing feels better then righting the wrong that was done to us.

These are but a few intros that we commonly see.





What is an example of an intro that we don't normally see?

A simple intro that establishes a foundation for our character in the world. Unlike the character generated storyline, this style of story telling takes place with an already known avatar which our players live through. They begin life doing what they do.

Here's an example

Jace, a mechanic for an auto company wakes up to the sound of his alarm clock. The player taking control from here on out, can see this avatars life through his own eyes, noting a family photo,, even seeing your characters children running about outside.

Now lets add a bit more to the into, perhaps integrate a tutorial.

Jace goes out, and mutters about his car not running properly, being a mechanic he has an idea of what maybe wrong. Thus starts the tutorial, the player is introduced to the problem, they are told via dialogue from the avatar, what skills and traits the avatar may have. After solving the puzzle they feel more connected to the player.

This would give a reasonable understanding of why the character can do what they can do. It doesn't take them out of the experience to do it.

Whats an intro, without a little foreshadowing. Foreshadowing merely helps us prepare the player for future events, we don't always need to use it, depending on the style of game were developing and how much we want to surprise our characters. Tutorials can be considered a tool for foreshadowing.

So lets add a little foreshadowing to Jace's story, to make it interesting it'll be used within a tutorial.

Jace finishes the car repairs just in time to hear his son call out for him, to play a simple game of hide and seek. The player is introduced to the hide and seek mechanic via a tutorial. Utilizing simple clues to determine where your son is hiding.

Once the story progresses, your looking for your son who is missing. You use the same clues that you were taught to use during the tutorial. Thus finding your son.

This style of introduction utilizes foreshadowing mixed with tutorials. It allows the player to learn the basic rules of the world which they then implement later on in the story. It helps to hide the obvious tutorial from the player, making the game feel more immersive.

This is a style of story progression that keeps the player well embedded into the story without breaking it.

What are other elements that make up an intro?

Atmosphere and Ambiance

-Atmosphere-

It is not just the weather, atmosphere can be static and or responsive.

A static intro, everything is predefined, and players are merely there for the ride until we finally give them control of the character. Best examples can usually be found in RPGs, were introduced to the world, the characters, and the possibly the antagonist through a CG intro. It can also be done via game play, where no matter what our decisions are, the same exact outcome will always happen.

Static- Jace goes to converse with his son, his son asks "Daddy will you play hide and seek with me?" your character then has an automatic response.

A responsive intro, is one where our players have a little more control over what the outcome of the intro is. Such as choosing sides in a war.

Responsive- You learn who your character is, why they are going on a mission, and they obtain the means to save the world. Right at the get go, you are given the choice to continue and help the good guys, or help the people you were fighting against just moments ago. This can still be considered the intro phase,, your still learning about the war, and the world, and now you have just initiated the beginning of the story.

-Ambiance-

Ambiance is the surrounding, from lighting, to sound itself.

The tone of the game sets the ambiance. It effects the mood of the player, and there isn't a right way, but there are wrong ways to implement ambiance. Ambiance should be constant. Ambiance in most games is static. A busy city will not only look, but it will sound busy.

Just as a lone haunted castle can feel deserted and sound deserted, not that it couldn't change, with a lot of games today, the world is becoming more and more adaptable to the players actions, whether the city suddenly goes quiet after a devastating aftermath, or police officers arriving to investigate the scream at the haunted house, their lights and sirens can be seen and heard.

(I would like to offer challenges, which would involve our readers participation, maybe the ideas offered by the readers can be posted in the forums, or the best idea being printed into the magazine along with the writers information giving them a little spotlight to shine in)

Here is the challenge, create an intro, use darkness to create a happy introduction.

How would you use both the atmosphere of the game, and the ambiance, to enhance the elements of this intro? Keeping it within the **T** for Teens game requirement.

Robert Morris aka Nightfox

wanted!

WE WANT YOU!

- If you are currently developing a game or application, we want to know about it.

- If you are a Game Artist, Developer or Programmer with an outstanding portfolio, we want to know about it.

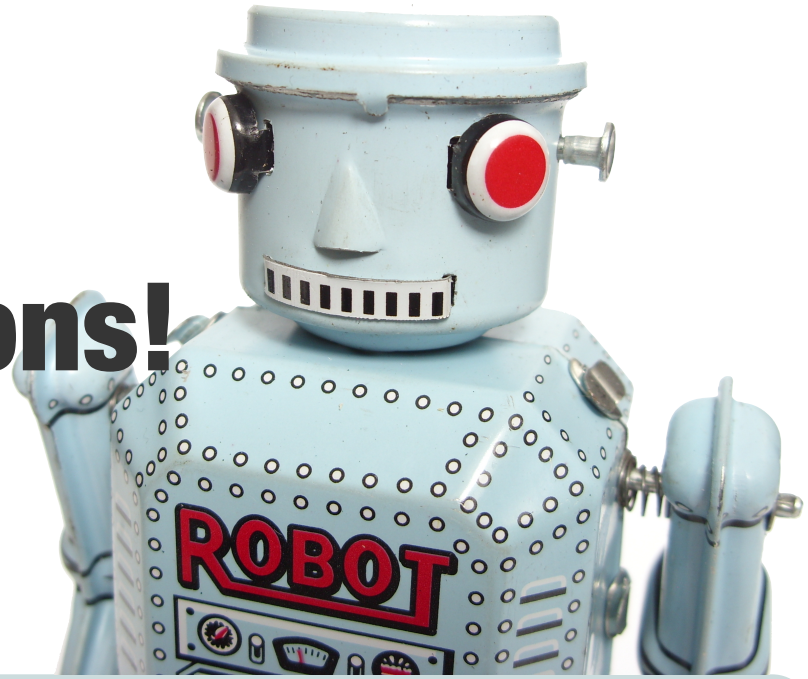
- If you have an idea for a tutorial, article or review, we want to know about.

If you are interested in contributing to Unity Creative with your knowledge, talent and/or ideas, feel free to contact us today.

3dattack@3dattack.us

Constant Interruptions!

by Jeff Murray



Gaming and gamers have changed dramatically since personal computers first crept into the average household. In my memories of playing the original Quake, sitting alone in the dark with my headphones on getting scared out of my wits, having a 'powerful' computer was a real novelty. Relatively few people had computers, let alone 3d games. The whole gaming experience was quite different, too, since most of the best selling titles were aimed squarely at what would be regarded today as 'hardcore' gamers.

With instant messaging, social networking and an almost constant flow of email, if you want to keep your players playing, you need to allow for your players to stop your game in mid-flow and to go off and do something else once in a while.

Most older games have some obvious differences to their modern day counterparts. There are no achievements in Quake, no combo bonuses and no support for task switching. As far as I remember, you couldn't even ALT+TAB out. The idea, back then, was that you loaded the game, you played it and you played it until either you completed it or your eyes fell out. Whilst not always a design choice and often due to technical limitations, there was little or no support for users leaving the game with a view to return.

The explosion in the popularity of casual games and social networking in the last 5-10 years has strongly influenced the way we play games today. In 2010, it is completely acceptable behavior to make calls or check email in the middle of a game. Incoming phone calls, emails, Facebook notifications, IM conversations… there are many reasons for taking a break. There are always exceptions to the rule, but try to understand your audience and the gaming experiences they may have become accustomed to. Playing games is about overcoming unnecessary obstacles to reach unnecessary goals, so we don't need to make it any more difficult for them to do what they don't have to!

Large-scale immersive experiences are generally played in longer chunks than their smaller counterparts, though this may well be on its way to changing. For example, almost all games purchased through the Steam network now have Steam's own social networking system built-in. You can, literally, chat to your friends through Steam in the middle of a fire fight in Call Of Duty. Whether or not you die is up to you, of course!

The current trends suggest that computers are being used for more than just concentrated game play, with unprecedented growth of social media channels and social gaming. Since it shows no signs of slowing, it's important to try and look after your players and their constant interruptions – I don't mean to labour the point here, but most of your players need their freedom.

As long as you are aware of the issues and you are aware of the fact that players may often be forced to leave your game behind, you should be able to find a strategy that suits your project. Try to think of what might happen at key points in your game if the player needed to take an important phone call, or if an instant messenger program popped up an alert window. Try to think of ways to make it easy for your players to drop in and out of the game world or give them regular breaks of easy gameplay to provide the opportunity for your players to wander temporarily. Modern gamers expect to be allowed to do what they want and it is our job to do our best in making them feel right at home.

Strategies to deal with interruptions

1. Make sure you can task switch without your game exploding.

It seems obvious, but there are still games that do not let you task switch. The last time I used a game engine that didn't have support for task switching built-in, for a casual game I was working on, it ended up costing a lot more time than it should have. Dealing with the game states and trapping window focus across different platforms and operating systems can be a pain to code and something that can trip you up because it seems deceptively trivial. Since you are using Unity, however, you made the right choice of engine for this. Your players are free to play the game and switch between their world and your world as many times as they like. The engine will just handle it. Join me in saying thanks for this little feature that will often go completely unnoticed by everyone!

2. Give players the option to play your game in a window or in full screen.

Talking of features that most people won't notice, Unity gives you full support for both full screen and windowed mode at just about any resolution a system's graphics card is capable of handling. It is a feature that may be taken for granted, but when you have to code this for yourself, across platforms and across the many different types of graphics cards, can be nothing short of a nightmare.

Thankfully, for Unity developers, including the option to play fullscreen or in a window may be nothing more complex than building extra UI to provide a checkbox. For your players, it may be invaluable and could mean the difference between them choosing to play your game as they chat to their friends, or choosing someone else's.

If you intend to take your game to any of the casual gaming portals, windowed and fullscreen support is a must, along with the ability to switch between windowed and fullscreen modes at any time during the game. With download games in particular, many game portals will demand both windowed and full screen capability because they know that most of their users prefer to have the option to do other things as they play. For the casual audience, this feature is a must.

Of course, switching between full screen and windowed mode can affect your UI and perhaps your games dynamic. For example, in a full screen game perhaps you don't need to display the mouse cursor whereas in a window it may make sense for the cursor to be on display. Unity helps you out by providing functions to deal with this. It's as simple as:

HYPERLINK "http://Screen-showCursor.html/"Screen.showCursor = false;

or

HYPERLINK "http://Screen-showCursor.html/"Screen.showCursor = true;

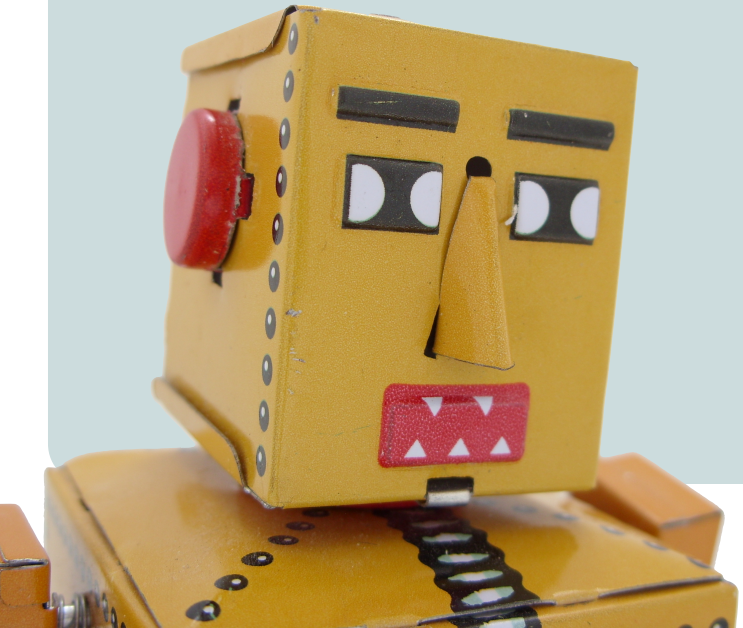
The screen class contains other useful variables and functions for dealing with resolution. Screen covers:

resolutions – returns a list of all fullscreen resolutions supported by the monitor
currentResolution – returns the current screen resolution
width – returns the current width of the screen window in pixels
height – returns the current height of the screen window in pixels
fullscreen – returns whether or not the game is running in full screen

This doesn't solve what to do with your GUI when the screen gets resized, though. When you change the resolution, your nice GUI could quite easily end up not fitting right and not filling the window anymore. That's why Unity have provided scaling for GUI elements using the Matrix4x4:

You use Matrix4x4.TRS(t, r, s) to create your matrix, where:

t is a translation vector value, move along x and y, but not z
r is a quaternion rotation value, rotate on z, but not on x and y
s is a scale vector value, scale on x and y, but not z



Here is a UnityScript example:

```
function OnGUI () {
	var tOffset	= Vector3 (0.0, 0.0, 0.0);
	var tRotation	= Quaternion.Euler(0, 0, 0);
	var tScale		= Vector3(2.0, 2.0, 1.0);
	var tMatrix	= Matrix4x4.TRS(tOffset, tRotation, tScale);
	GUI.matrix		= tMatrix;
}
```

And in C#:

```
void OnGUI()
{
	Vector3 tOffset		=	new Vector3(0, 0, 0);
	Vector3 tScale		=	new Vector3(2.0f, 2.0f, 1.0f);
	Quaternion tRotation	=	Quaternion.Euler(0, 0, 0);
	Matrix4x4 tMatrix	=	Matrix4x4.TRS(tOffset, tRotation,
tScale);

	GUI.matrix = tMatrix;
}
```

By setting this up, the GUI can scale to fit all resolutions. I tend to use it by building the UI in a default resolution first (below is an example of setting the GUI.Matrix up to work with iPhone sizes - since we used GUI.Matrix it will scale my UI at different resolutions, such as the iPad's 1024x768, to fill the screen).

GUI.matrix = Matrix4x4.TRS (Vector3.zero, Quaternion.identity, Vector3 (Screen.width / 480.0, Screen.height / 320.0, 1));

Once you have your default set up, you can arrange your GUI items with coordinates within that space (for the example here, to fill the window we could create a UI element 480x320 pixels).

With a little tweaking, you can easily make your game look great in both windowed and fullscreen modes at several different resolutions. Although there are other approaches, using GUI.matrix appears to be the most straightforward.

3. Try, as much as possible, to match the play lengths of your levels to the best and least noticeable amount of time.

Have you ever played a game where you didn't feel as though it was taking over your life, though somehow you could easily while away hour after hour? One of those 'just one more level' or 'just one more checkpoint' type games? It doesn't happen by accident!

Timing levels to 'just that right length' is an art that can make or break your game. For better or worse, the casual game explosion has helped to create a science of gameplay where the objective is to keep you playing, without letting you know about it.

Hardcore gamers often complain about the intrusion of casual game behaviors in the domain of 'serious' gaming; There is a whole argument against the 'watering down' of the hardcore gaming experience, though in order for modern titles to be successful with as many different types of gamers as possible, large scale game directors and designers have to work harder to try and deliver the benefits of a more casual game structure without dissolving the more 'hardcore' experience for the core market.

A great example of this, Call of Duty, is a highly successful first person shooter published by Activision. At first glance, there is a temptation to classify this a hardcore gamer's game. Though it may look and feel like a more hardcore game than, say PopCap's Bejeweled, there are an increasing number of traits that are shared between them. The evolution of

the Call of Duty series has been an interesting guide in ideal play lengths and of highly immersive linear game structures catering to both casual and hardcore gamers. The key to that success lies in carefully controlled lengths of intense, mild and low action sequences – in effect, giving players just enough of a dose of action to make them feel like there is time to play 'one more round' without making them feel like they didn't have their significant portion.

The break between rounds may serve as both a breather for the fatigued player or a chance to go and do something else before returning to the fight. There are many advantages to keeping your play in chunks, not least temping players to come back for one more round. For the developers, these play lengths allow the splitting up of the game into more manageable chunks both technically (making memory management, loading times, streaming etc. manageable) and in terms of allowing greater control over the game pace and flow and perhaps pace changes between areas.

Increasingly, in linear level designs such as the Call of Duty series, you can play for a short length of time, reach a checkpoint and choose to stop or continue. The game makes a point of telling you when you reach a checkpoint both as a reward and as an important signifier that they don't have to continue if they don't want to.

Players can go off to do something else, take a quick break and return, or try to reach 'one more checkpoint' before quitting for the night without it affecting their schedule (since it's only another 5 minutes, right?). Once you get into that 'one more checkpoint' cycle, it is easy to end up doing one more, then another then another until it's 5am and you have to be up for work in two hours. Not that I would ever do that, of course!

For a full breakdown (and some incredible advice on pacing) of Call of Duty and the pacing behind the level design, Gamasutra has a great article HYPERLINK "http://www.gamasutra.com/view/feature/3863/gameplay_fundamentals_revisited_.php"http://www.gamasutra.com/view/feature/3863/gameplay_fundamentals_revisited_.php

Allowing for short bursts of play or safe, quiet periods in-game also gives players the opportunity to take a deep breathe or do what they need to do outside of the game world. This is particularly important in multiplayer games.

In multiplayer games, the most expected 'out of the game' activity is communication with other players. Of course, no-one stops to chat in the middle of a melee attack. Slowing down the action at key points, safe areas or providing certain breaks in play helps players to have the chance to communicate with each other.

Timing is everything. When testing your game, pay particular attention to play lengths and how players feel about them. If at all possible, try different play lengths of the same levels on different people and gauge their reactions.

4. Store game state regularly, or if you can trap focus loss make sure you store it quickly!

Save points may be points at which you allow the user to save their game, or checkpoints at which you automatically save the game for them. Save point restrictions (such as distance between checkpoints) are a game designer's choice and should not be dictated by the technology. It is a game designer's job to take target player preference into account and design the game with any restrictions or possibilities in mind.

Making save points too far apart will frustrate players and making them too close will make the game too easy.

If you're working with Unity iPhone, this is something you really need to pay attention to. On mobile devices, players may be forced to leave the game for any number of reasons such as a dead battery, an important phone call or perhaps another program popping up with some kind of notification. If at all possible, players need to be able to quit out on demand and return to the last point without losing much of their hard work. Thankfully Unity provides everything you need to do that without too much work on your part.

Unity's PlayerPrefs system is a great option for saving data, though a quick search on the Unity forum may lead you to some alternatives that go beyond the scope of this article, such as writing to a text file or XML serialization (for more on XML serialization, there is an extensive post on the Unity Answers site http://answers.unity3d.com/questions/971/how-to-scrip-a-save-load-game- option).

If you don't fancy writing your own state saving code, there is also a straightforward third-party solution available for purchase called EZ Game Server. It is a turnkey solution for tackling data saving – HYPERLINK "http://www.anbsoft.com/middleware/ezs/"http://www.anbsoft.com/middleware/ezs/. You don't need a paid solution like this, of course, since Unity is capable of what you need right out of the box, but having a solution already built like this may end up saving development time and helping out your project. One of the great things about EZ Game Server is that it is so extensive.

As with just about everything, it depends on your exact requirements and there really is no right or wrong way just as long as it works for your project.

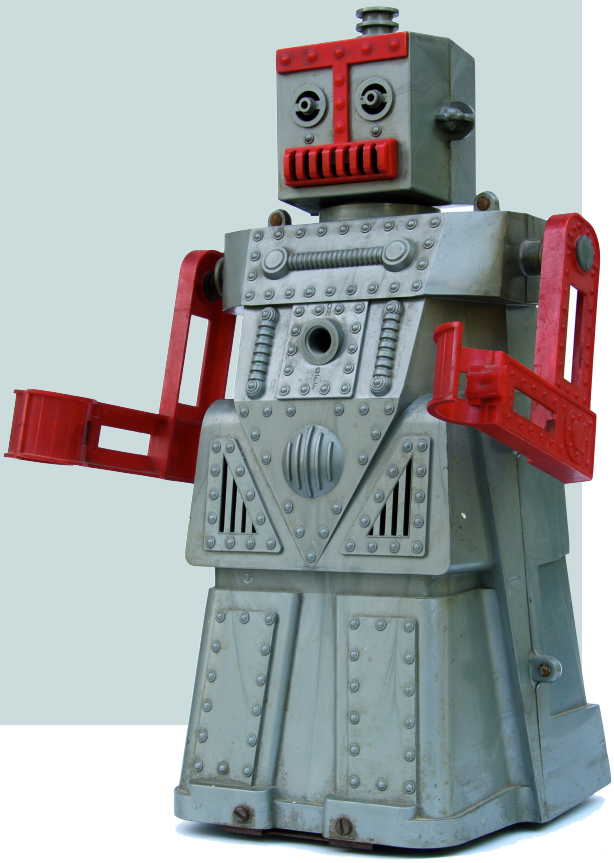
Here is a basic example of using PlayerPrefs, which will work in both C# and UnityScript:

Storing the player's position:

```
PlayerPrefs.SetFloat("Player_PosX", player.transform.position.x);
PlayerPrefs.SetFloat("Player_PosY", player.transform.position.y);
PlayerPrefs.SetFloat("Player_PosZ", player.transform.position.z);
```

Loading the player's position back again:

```
player.transform.position.x = PlayerPrefs.GetFloat("Player_PosX");
player.transform.position.y = PlayerPrefs.GetFloat("Player_PosY");
player.transform.position.z = PlayerPrefs.GetFloat("Player_PosZ");
```



Checking to see if this is the first time your game has been played (great for one-time tutorials or intro videos that you don't want to play over and over):

```
if (!PlayerPrefs.hasKey("init_prefs")){ //Is this first time?
  PlayerPrefs.SetInt("init_prefs", 1); //set 'did init' key.
  Debug.Log("This is the first time that the game has been run");
}
```

Checking to see if a high score has been beaten:

```
highScore=PlayerPrefs.GetInt("High score");
if (totalScore > highScore)
{
        PlayerPrefs.SetInt("High score", totalScore);
        Debug.Log("NEW HIGH SCORE!");
}
```

Note: In Unity 2.x, it is very important that you remember to read the prefs exactly as you wrote them. Trying to use PlayerPrefs.GetInt when you originally used PlayerPrefs.SetFloat will not work. Often, it will fail silently, meaning that you will have no indication of why your code isn't working properly and no error will appear about it. Check and double check that your get and set calls match exactly!

You may also want to look at a fantastic little chunk of free code from the Unity forums called 'Keys', written by Nick Breslin. It allows you to watch PlayerPrefs changes live (Nick calls it a 'PlayerPrefs Runtime Monitor').

http://forum.unity3d.com/viewtopic.php?t=26247

You can trap when the user quits out of the game (or when they are forced out) by using Unity's OnApplicationQuit function. On the iphone specifically, when an incoming call takes over the game this function will get fired and you can hook into it, to save your data.

Keeping game state and restoring it at the right times is key to saving your players from feeling like they lose out when they purposefully or accidentally destroy or postpone your game.

Players should be allowed to concentrate on playing the game and not concerned with the technicalities surrounding it. When they step out to post their latest cat video on YouTube, the key to a good player/game relationship is that they should not be penalized for daring to step away from it every once in a while.

Profile:

Jeff Murray is a freelance game designer and developer living in Ottawa, Canada. He has worked on just under 50 games including downloadable, browser-based and console titles. Jeff has worn many different hats, taking on the roles of Lead Programmer, Game Designer and Director on projects for companies like Microsoft, RealArcade, Hasbro and SpinMaster. Jeff is also an IGDA contributor and member with a real passion for good gameplay and tasty beer.

For more information, visit Jeff's blog at http://www.psychicparrotgames.com