Issue 28

January 2009

DEV.MAG

CREATE • DEVELOP • EXPERIENCE

INSIDE:

We chat to the creator of Iji O Get your fix of collision detection and learn some handy flash O Reviews: Penny Arcade Adventures Episode 2 O Killer Worm O Iji O Loads more!

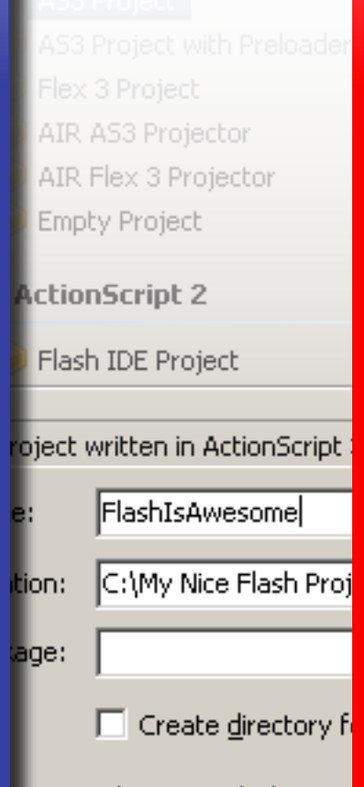
REGULARS

OPINION

FEATURE

REVIEW

DEVELOPMENT



TAILPIECE

HEADMASTER
Claudio "Dumbledore" de Sa

DEPUTY HEADMASTER
James "Professor McGonagall" Etherington-Smith

HE-WHO-MUST-NOT-BE-NAMED
Quinton "Voldemort, bitches" Bronkhorst

STUDENTS
Rodain "Harry Potter" Joubert
Simon "Ron Weasley" de la Rouviere
William "Neville Longbottom" Cairns
Danny "That one dude" Day
Andre "George Weasley" Odendaal
Luke "Fred Weasley" Lamothe
Gareth "Draco Malfoy" Wilcock
Sven "Luna Lovegood" Bergstrom
Chris "Herwhiney Ranger" Dudley
Herman "Herman Tulleken" Tulleken

CARETAKER
Robbie "Argus Filch" Fraser

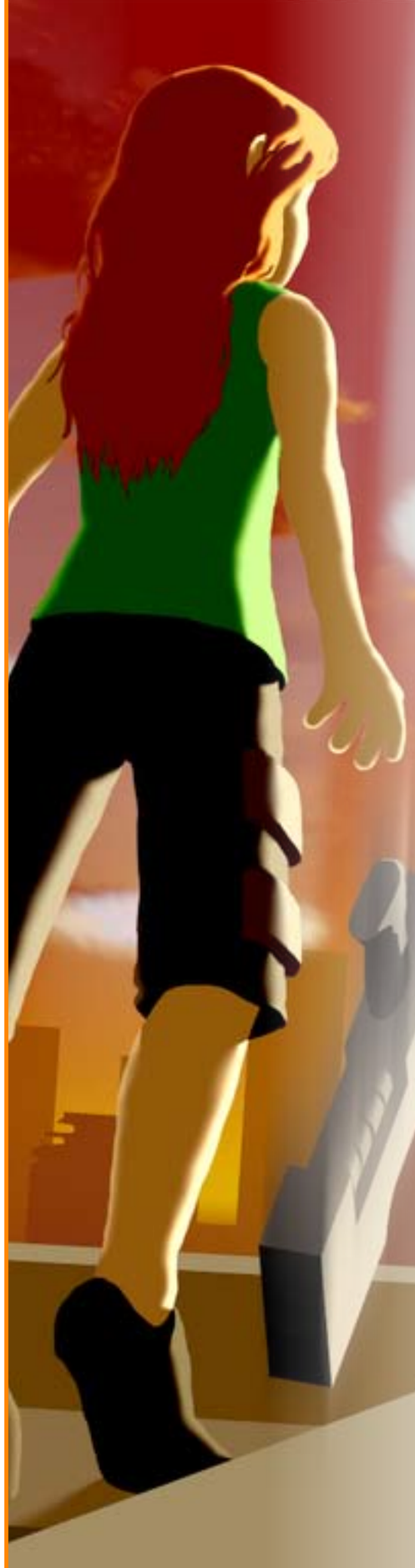
HOGWARTS
www.devmag.org.za

HEDWIG
devmag@gmail.com

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

SNAPE KILLS DUMBLEDORE
LOLOLOLOL. Oh wait, this is old.

:<



So we're all back from our extended holidays, thrust back to the tedium of the daily working grind, all the festivities already forgotten. Which generally means news (and, in fact, everything) is a bit light in comparison to our usual offerings.

Not to fear, however, the long break hasn't dulled us or our dedication and contributions to the magazine. In fact, recent growth and developments have led to the possibility of some rather drastic (and long coming) future changes to accompany our change of hosting. Yes, that's right, we'll be changing our hosting service soon, and I apologise in advance if this causes any issues with our website and the availability of our magazines.

That also brings me to a special but unfortunate request: Reliable web hosting that can offer the service we require (and promise to our readers) costs money; not a lot, but enough that help would be appreciated. As such, sometime next month or late this month, we'll be adding an extra link to our website that will allow you, our readers, to **donate to the magazine** and help keep it running. This is **not obligatory at all**. The magazine will stay free (probably forever) and you're more than welcome to continue reading and downloading our issues as you've always done. However, for the generous among you (or those who feel our work is worth something other than your readership), the donation option will be made available in the near future.

On the content side of things, we have a fairly special feature/review pair this month, and our cover this month (drawn by **John Nesky**) hints to its content: a review of a fantastic Game Maker platform title, **Iji**, and an accompanying interview with its creator, Daniel Remar.

That's all I have to babble on about this month. Get reading, and enjoy!

~ Editor

Claudio "Chippit de Sa

Blender game engine competition

Blender

To encourage experimentation with Blender's recently overhauled and improved game engine, a competition is being hosted on the official Blender site, calling for developers and development teams to use Blender's tools to create a fully featured game in one of the 3 categories: Best graphics, best gameplay, and a third, special category restricting the use of python scripting. The deadline for entries is 16 March, which gives you time to put your skills to the test.

Atari's lite-C game programming language version 1.5 available

Atari

The newest version of Atari's Windows-based game programming language, created by Conitec in association with Atari, has been released to the public. It has a 24-workshop tutorial that aims to teach anyone how to use the software, and attempts to be easy to use and understand even for non-programmers, by handling complex importing, collision detection, lighting and other 3D game considerations transparently. It is that is free for non-commercial purposes.

Lost Garden fishing prototype results

Lost Garden

The results for the most recent Lost Garden prototyping challenge have been revealed, and they include a surprise result: For the very first time, a gold medal was awarded to an entry, made by one Andre, a game that took the prototype and design further than any other before. Far enough to have it accumulate a 4.1 aggregate rating on Newgrounds, and ultimately to sell it for a neat sum of $4000. Who said these prototyping challenges lead nowhere?

2009 IGF finalists announced

IGF

The 22 finalists for 2009's Independent Games Festival have been announced, including DreamBuildPlay finalist and grand prize winner, CarneyVale Showtime. IGF is a huge event in the indie calendar, and we'll be carefully watching developments in this area. Expect a huge roundup on all the IGF entrants in the next issue.

Global Game Jam in Cape Town

Afrigraph

The Global Game Jam, a 48-hour international game prototype creation competition being held roughly simultaneously in numerous venues all over the planet, is taking place in Cape Town this year too. The University of Cape Town has offered their computer science labs for the purposes of the competition, set to be held from 30 January to 1 February. With tons of popular development tools available for participants to work in (including both Game Maker and XNA), the Jam promises to be a fun evening for like-minded developers to team up and frantically cobble something playable together.

"THIS IMBECILIC AND DOWNRIGHT HARMFUL OPINION WILL BE FED TO THE GENERAL INTERNET POPULACE."

The other day, a member of the Game.Dev community was accused of producing a racist game. Surprised and intrigued by the rumour, I investigated the link provided by a friend and stumbled upon a Website that claimed to analyse games on a more "academic" level than the average review or blog post.

I was equal parts relieved and annoyed to find that the blog post I was sent to consisted of distilled garbage – the result of some self-aggrandising, pseudo-intellectual lunatic who believed that cockroaches and pest exterminators were a metaphor for racial oppression and wholesale slaughter of native Africans rather than, say, a damn pest extermination simulator. Reasons for this line of thinking included factors such as the roaches' dark colours and the game developer's background as a white South African citizen (a national identity which immediately seems to raise the racism flag amongst less-than-enlightened foreigners, along with thoughts of mud huts and riding lions).

After two posts in the comments section which gained me little ground (the Website admins didn't seem interested in retracting their statements or apologising for their sensationalism), I made a quick facepalm and moved on to more enlightened parts of the Internet.

I think it's sufficient to say that I was unimpressed with the blog post. They accused my colleague of producing a racist game based on the most ludicrous of evidence, then tried to justify their statement instead of looking back on it and perhaps saying, "Actually, my bad." I'm not even going to go into the journalistic responsibilities that I feel were quite thoroughly violated (a hint, ladies and gentlemen: if it's not a forum, and it's not framed as an opinion column, don't call your damn blog posts "academic" and then start writing sensationalist rubbish).

What upsets me most, however, is that at the end of the day this imbecilic and downright harmful opinion will be fed to the general Internet populace, 90% of which will never even bother scrolling down to look at the rebuttals afterwards. Voices like this exist all over the Internet in a whole variety of genres. Game development, in particular, has its fair share of individuals who cry wolf in this manner, with opinions hiding in everything from academic essays to humble MySpace posts made by angst ridden teens.

One of my friends drew an analogy between this and the "feminist conspiracy" regarding the shape and function of the red and blue ovals in Portal. Then there's the stuff about ninja games being a Japanese conspiracy to train gamers for their secret army. One really doesn't have to dig that deep in order to find a whole menagerie of ludicrous theories that relate to just about any paradigm, class or society. They make Jack Thompson look completely reasonable in comparison, and I'd throw my chips in with him if these people were the alternative.

Sadly, there actually are problem games out there which promote some very negative messages, and not in an ironic or over-the-top manner either. Army of Two, for example, is incredibly politically slanted and several reviewers have pointed out the rather shameless pro American bias that it contains. Then there are those ubiquitous under-the-radar games which clearly and unapologetically promote hate speech, intolerance and gender-based violence in a much more severe manner.

Maybe if we had more self-styled watchdogs keeping an eye out for the real problems instead of latching onto messages that only their own guilt or obsession lets them see we'd have a slightly better Internet society on our hands. Or maybe not. At the very least it would keep them occupied with real problems instead of contrived horrors.

Racist cockroaches, anyone?

Simon "tr00jg" de la Rouviere

Talking about.... Iji

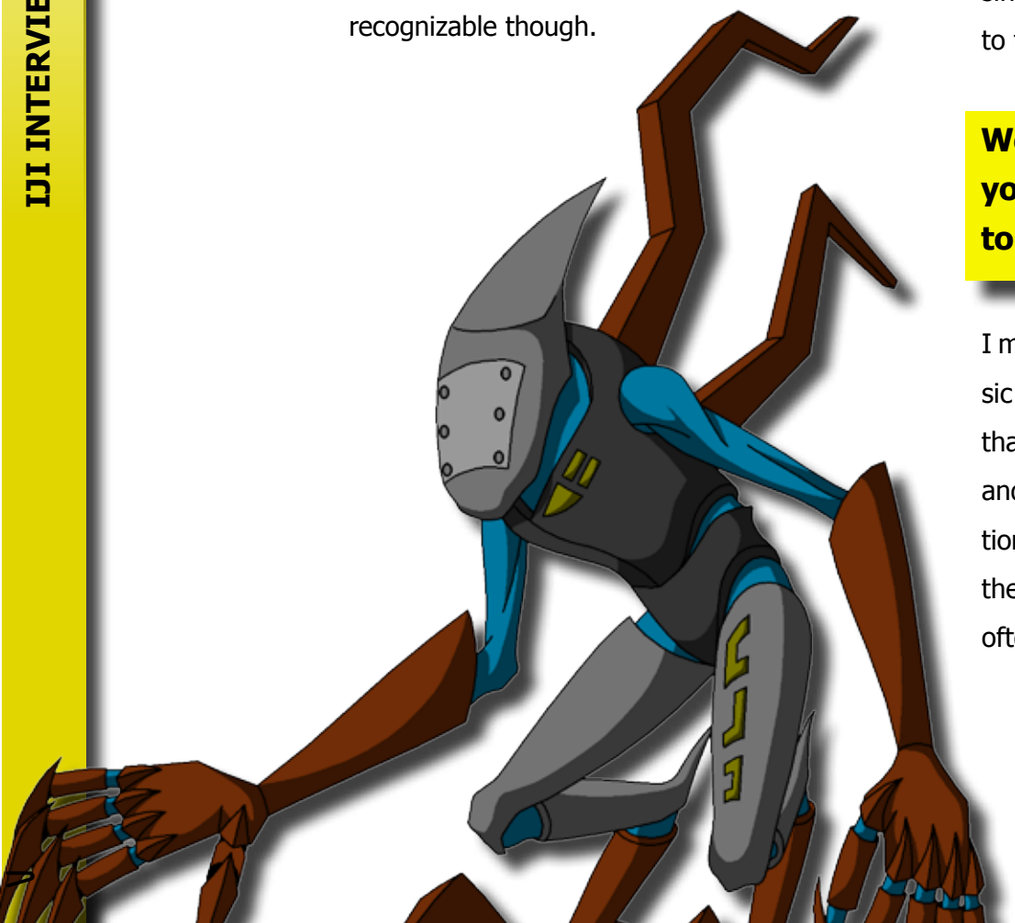
To accompany our review of the game, Dev.Mag sat down with **Daniel Remar**, the sole creator of Iji, and probed him with questions about the development of the remarkable game.

What inspired you to make the story-driven platformer, Iji?

The main inspiration comes from my old discontinued comic, which in turn was inspired by another creation of mine that stemmed from playing Operation Carnage. Iji was at first supposed to be a survival horror kind of platformer, but changed rapidly once I begun working on it. It's hard to say when or why it turned into what it is now.

Where does the name come from? Why Iji?

All the names were pretty much randomized, and don't have any special meaning. I wanted the main ones short and easily recognizable though.



Iji took 4 years to develop. Why? Were there several design iterations?

There were no real design iterations; it just flowed from start to finish. It took so long because of the huge amount of content in the game, and the amount of bug fixing I needed to do. Recording the voices and drawing the cutscenes are examples of things that took several months each. Of course, it didn't take four full years; I often took month-long breaks to keep my motivation up. Building and tiling a level also usually took a few weeks, but Sector 5 was done in three days. Some of the many moves, secrets and alternate texts and events also took a long time since they often caused conflicts, and I had to test them thoroughly.

Were there any particular goals you wanted the music in the game to achieve?

I mainly wanted heavy or industrial metal music mixed with ambience, recorded live rather than in MOD format. Machinae Supremacy and their various works were a big inspiration, and one of the reasons I like them is the optimism and believing in oneself that is often seen in their lyrics and musical style.

Were there any specific guidelines you passed on to the composers who created it for you?

There was a lot of experimenting and testing with the music long before I wrote the general guidelines for the sound and feeling of each song. A mini-soundtrack was produced for the first demo of the game, which gave us experience in understanding each others' tastes. While the final soundtrack was being composed and recorded, I let Chris do pretty much what he wanted since I trusted him, and he knew what kind of style we were after. He liked driving, positive tracks rather than dark, depressing ones, and I thought that what he came up with fit the game perfectly. Only one song in the final batch (Organ Smash) was left out of the game. It's included in the high-quality soundtrack download though, so don't miss it.

Did you create any custom tools or systems or was it all made in Game Maker?

To make the game's resources I used Photoshop for graphics, Blender 3D for the characters, and Goldwave for editing the sound and voices, but the only "system" I could say I created was the polygon rotation and forward kinematics animation tool for the final boss. Unlike the other characters, it animates in real time with polygons, rather than using rendered sprites, since it's so large. Unfortunately this made it slow on some computers.

How did you find working with Game Maker? Did you upgrade in the middle of the development?

It's all GM5, since it was too bothersome to switch to GM6 in 2005 due to the many incompatibilities, the loss of certain functions, the poorer sound and music handling and the loss of compatibility with Windows 98, among other things. I always work in GM5 unless a certain game would be impossible or more difficult to do without it, such as Garden Gnome Carnage's rotation and surface effects. GGC also started out as a GM5 game though.

It seems that you took some inspiration from System Shock, Deus Ex, Blackthorne and Another World. Any other games that helped shape the final product?

The inspiration for the gameplay was largely the games you list, but I got inspiration from so many others (most of them subconsciously) that I can't list them all. There are references to everything from Tyrian to Doom in there, but I was also inspired by movies like Nausicaä.

Of all the games you took inspiration from, which is your favourite?

Of the direct inspirations it's System Shock 2. I personally don't think Another World and Blackthorne are fun games, but they're inspiring.

You used the same polygonal character animation that Another World used. How did you go about creating and animating the characters?

I built and rigged them in Blender 3D, and animated them frame-by-frame which was so tedious it nearly stopped the development of the game. I rendered them from a parallel perspective in flat colors, and fixed the frames up in Photoshop. It produced better results than trying to animate them by hand, at least. Since they are only seen from the sides, I could "cheat" a lot while animating them, such as having body parts cut into each other or dislocating parts to make sure they were drawn in front of others.

What are your plans for the future?

There are some games I want to make which will only take a few months each, but I won't have the kind of spare time I had as when I worked on Iji anymore. I won't make anything the size of Iji again, it's too draining.

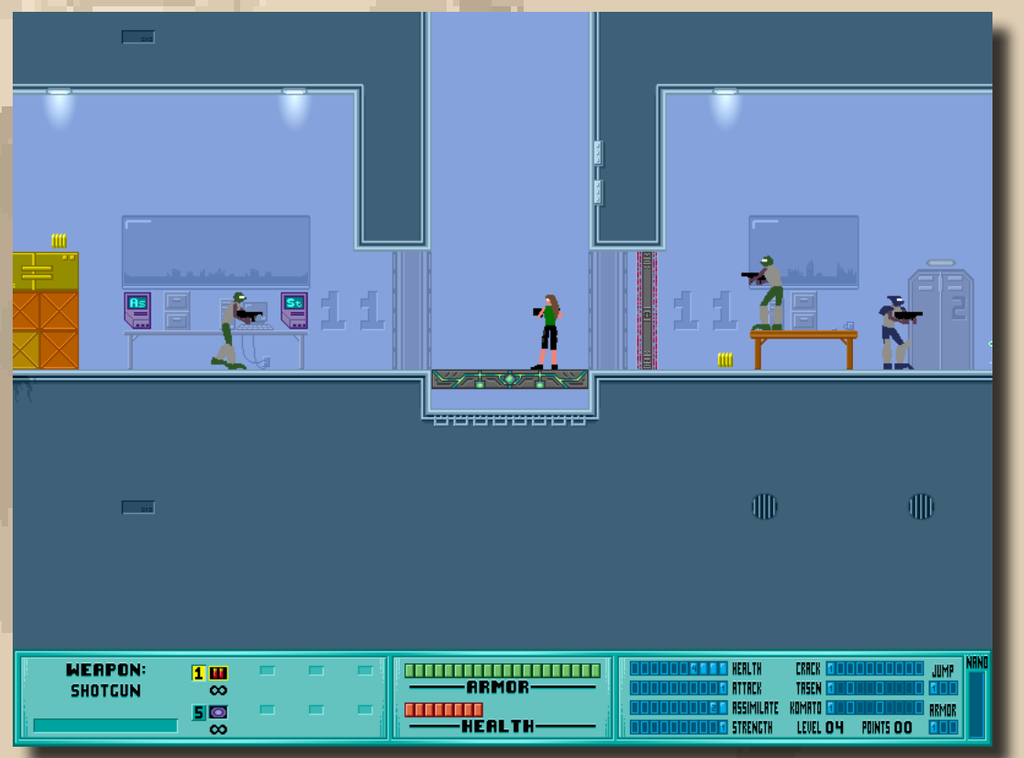
SHORTS

Favourite Platform?

Either the PC or N64, they're the ones I've played the most over the years.

Bad in-flight movies or cold coffee?

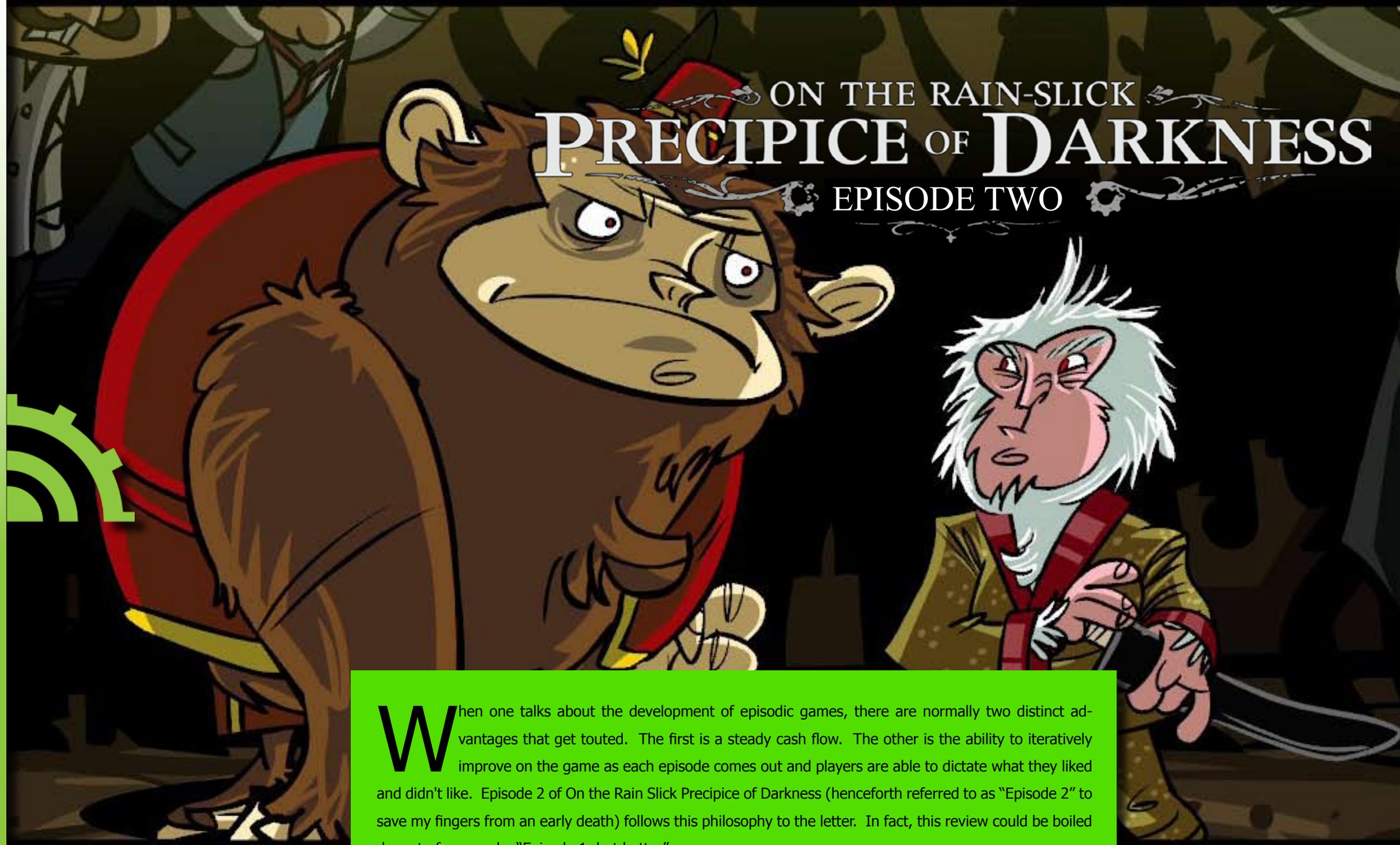
I neither fly nor drink coffee; so between failed homemade cake rolls and being too cheap to buy clothes without holes in them, I'll say failed homemade cake rolls.



"THE DEVELOPERS HAVE TAKEN ALL THE PLAYER FEEDBACK ABOUT EPISODE 1 TO MAKE EPISODE 2 BETTER."

ON THE RAIN-SLICK PRECIPICE OF DARKNESS

EPISODE TWO



Gareth "Gazza_N" Wilcock

When one talks about the development of episodic games, there are normally two distinct advantages that get touted. The first is a steady cash flow. The other is the ability to iteratively improve on the game as each episode comes out and players are able to dictate what they liked and didn't like. Episode 2 of On the Rain Slick Precipice of Darkness (henceforth referred to as "Episode 2" to save my fingers from an early death) follows this philosophy to the letter. In fact, this review could be boiled down to four words: "Episode 1, but better".

"FUN TO PLAY AND BRILLIANTLY WRITTEN."

Given that we published a review of Episode 1 not so long ago (Dev.Mag Issue 24), I won't go into too much depth regarding the mechanics. Suffice to say that at its core Episode 2 is Episode 1 – same characters; same basic plot premise; same interface; same exploration/puzzle-solving/combat dynamic. Naturally, they've substituted certain elements with newer ones – the characters have new weapons and attacks that you need to improve; the Special Attack minigames are different; and the enemies are tougher. Powerups and status items have undergone minor changes to slightly alter your battle strategy. Otherwise, the gameplay remains the same, but given how much fun Episode 1 was, this isn't much of a drawback.

What the developers have done is taken all the player feedback about Episode 1, and expanded and refined on those elements to make Episode 2 better. For example, the game feels much longer, with more characters, and locations to visit than in the original. Along with this comes more enemies and more, tougher (but far from impossible), puzzles. The game is crammed full of more of the hilarious animated cutscenes that made the original so entertaining. The combat mechanics may be the same, but the developers have attempted to make fighting more dynamic. Enemies will switch positions on the fly, enter combat at different times and from different directions, and flee from battle to lure you to larger and stronger groups of adversaries. Blocking has been made easier as well, by increasing the visibility of the "block" cue.

Technically, Episode 2 can be played without having played Episode 1, but this would be a mistake. The presence of certain characters, inventory items and plot points make very little sense unless you've been introduced to them in the first game. Additionally, by importing your Episode 1 character, you gain access to special items that were won or purchased in that game, but didn't have much use. These can be combined with others in Episode 2 to obtain additional benefits.

So yes, Episode 2 is more of the same, but with extra polish and more content. It's still as fun to play and brilliantly written as one would expect, and the changes and additions keep it from feeling too much like more of the same. If you enjoyed Episode 1, you'll enjoy this continuation of the series.

Iji

If one were to attempt classification, Iji could be described as a "Nonlinear Action RPG Platformer". It borrows thematic and gameplay elements from System Shock and Deus Ex, marries them with tactical platformer gunplay à la Blackthorne, and presents it all using the same basic polygonal graphics that made Another World so visually distinctive.

Whew! Quite a mouthful!

"A LOT DEEPER THAN ONE WOULD EXPECT."

Gareth "Gazza_N" Wilcock

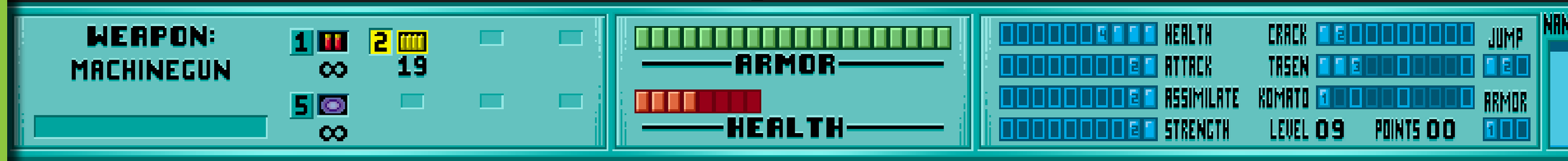
"IJI AWAKES FROM A COMA TO FIND THAT THE MILITARY COMPLEX HAS BEEN COMPLETELY OVERRUN BY ALIENS."

Sadly, a side effect of relating a game to existing titles is that it makes the reader think that the game being reviewed consists of nothing but cheap borrowings from the games it's being compared to. This does the game a great disservice, especially if it does a particularly good job of melding all those borrowings into a cohesive and entertaining whole. Iji takes all its influences, mashes them together very nicely, and adds some little touches of its own that make it highly distinctive; but enough of that – it's time to tell you why.

Iji's story begins with the title character and her family touring the military research complex where her father works. This is rudely interrupted when a hostile alien race hits the Earth with a somewhat sudden and unprovoked orbital bombardment. Six months later, Iji awakes from a coma to find that the military complex has been completely overrun by the aliens, and that she has been infused with reverse-engineered alien nanotechnology by the few surviving scientists. It's up to her to strike back against the invaders and, through whatever means available, get them off what's left of Earth.

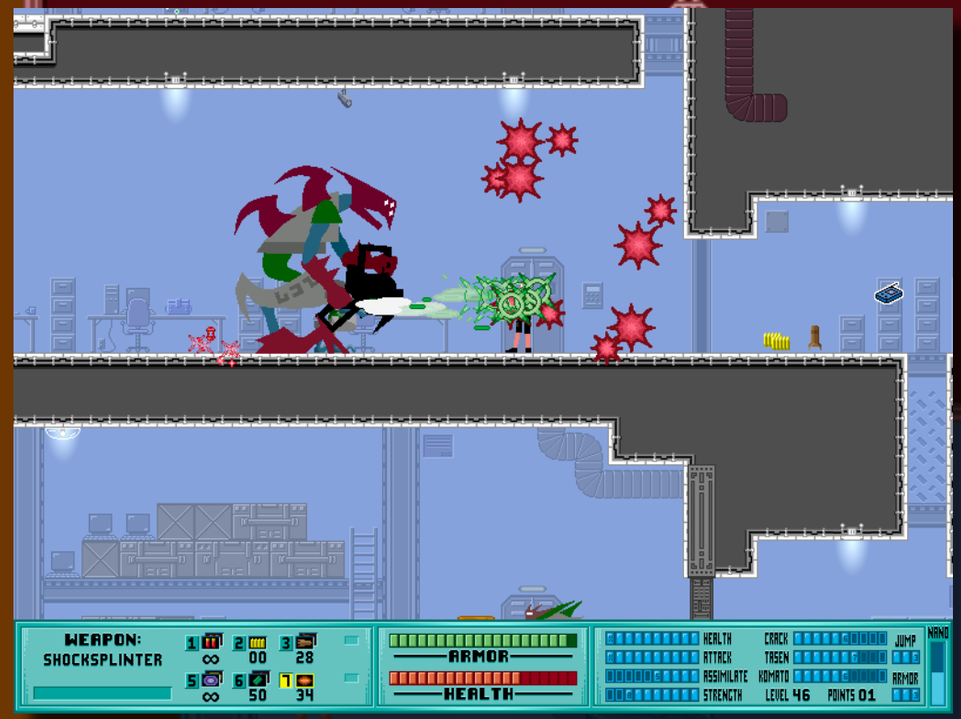
So yes, Iji is part action platformer; and it's little different from any other action platformer you may have played. You guide Iji through the ten Sectors of the military complex, jumping from floor to floor and shooting increasingly nasty aliens with a growing range of weaponry. Her repertoire of moves is pretty basic – jumping and crouching.

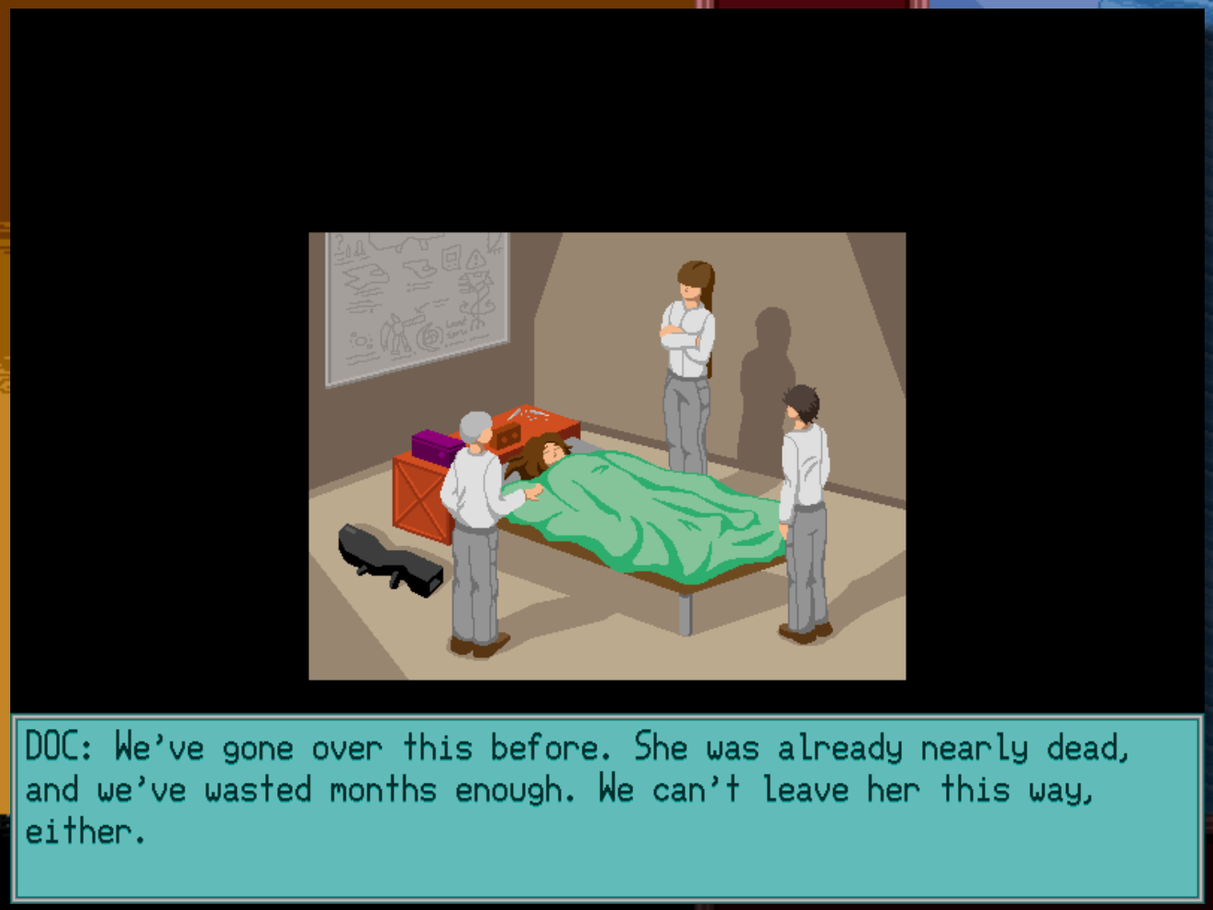
Machinegun ammo +4



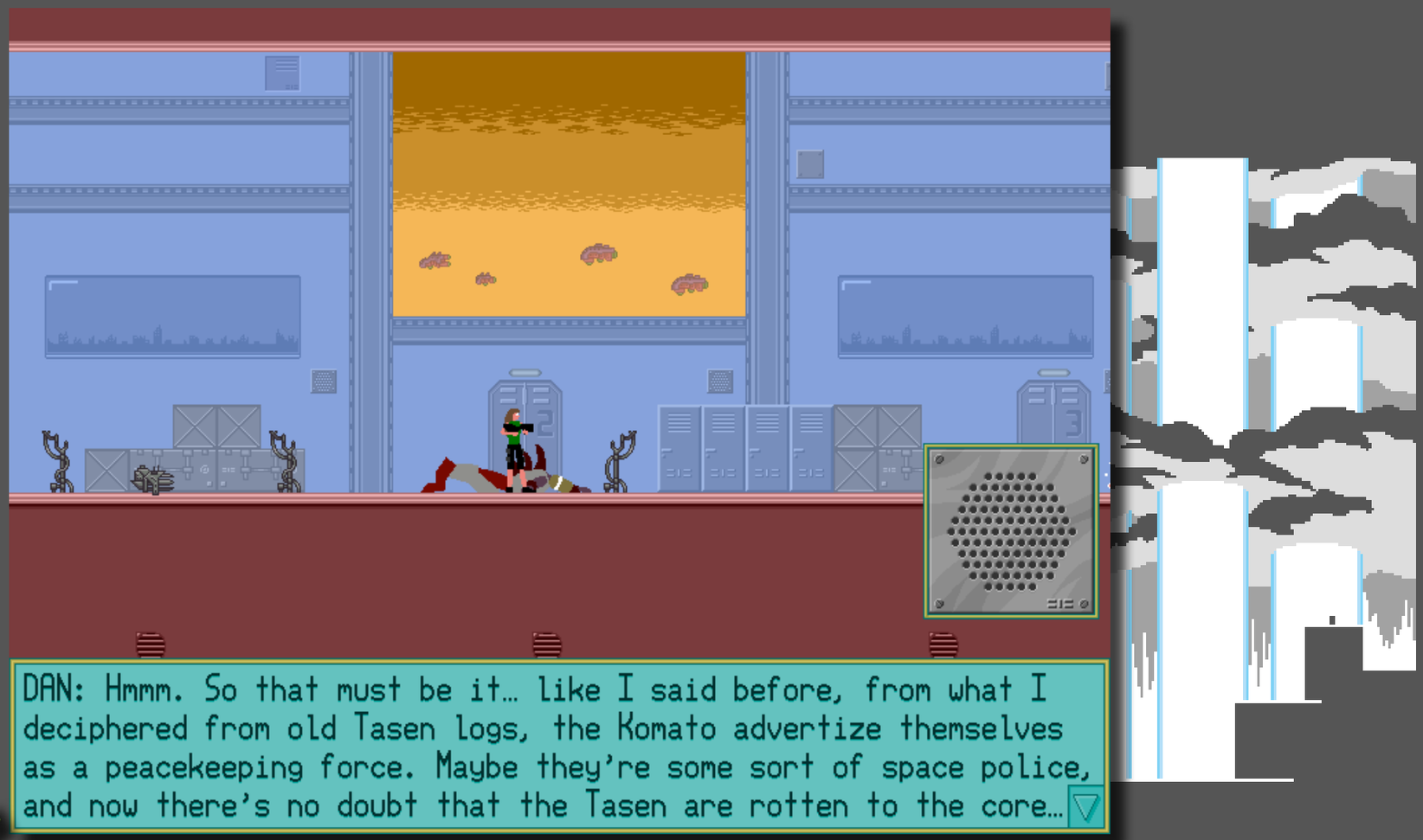
Crouching allows nastier enemy projectiles to zip harmlessly over her head, and is ideally performed behind cover to prevent fire from weaker weapons and melee attacks from reaching her too. However, unlike in other games of this type, she is unable to fire her own weapons while either crouching or in mid-jump. This can become annoying in some situations, but eliminating these abilities makes tactical placement and weapon selection all the more crucial to your survival. You can't just pull off the old "crouch 'n' fire while the aliens shoot uselessly over your head" or "jump to the next floor level, fire at enemy in mid-jump, fall into cover" maneuvers. This forces you to think your firefights through a little more carefully, especially against larger, stronger adversaries.

Fallen foes leave behind residual nanite swarms, known in the game as "Nano". Once enough Nano, which can also be found as free-floating, nonlethal swarms, is collected, Iji gains a Level Point. As you progress through the facility, you'll find special Upgrade Stations where you can trade Level Points for increased aptitude in the specific skills indicated by each station. Health and Attack skills provide more hit points and weapon effectiveness respectively. Strength allows you to shatter certain doors and send enemies flying with Iji's Mighty Foot. Your aptitude with alien weaponry needs to be increased to a set level before you can use certain guns. Cracking gives you access to the hacking mini-game, which allows you hack equipment containers, doors, and even lets you fuse weapons together to create more powerful variants. Choosing which skills to augment is an immense challenge at points, as all of them are incredibly useful and can dictate your path through the sectors, with all the benefits that brings in terms of access to weapons and collectibles.





"IT'S DIFFICULT TO REFRAIN FROM GUSHING ABOUT IJI."



SECOND OPINION

by Chris "LionsInnards" Dudley

Iji is like a delicious stew. It feels as though a pinch of everything was flung into the mixing pot, but with the grace and skill of a master chef. Slices of role-playing and story float around in the platforming sauce, complimenting the A-grade chunks of action, all garnished with a delicate sprinkling of emotion. This is a full meal, and one that was crafted with the most basic of tools, Game Maker 5. It is the kind of game that makes one's own creations with the program appear amateurish; as prison gruel is to haute cuisine.

Iji's story deserves special mention too. Presented through a mixture of semi-animated slideshow cutscenes, in-game dialogue and enemy notebooks (which also provide some humourous gameplay tips and "recipes" for new weapons) – it's certainly a lot deeper than one would expect from a game of this type, presenting some very interesting questions to the player regarding war and politics. Most intriguing is the effect that the player's actions have on Iji herself and on certain aspects of the story. While the overall plot remains the same regardless, taking either a pacifistic, avoidant stance or an aggressive, kill-'em-all strategy alters in-game dialogue and the contents of enemy logbooks, and determines the presence of certain friendly characters later in the game. Iji's in-game utterances will reflect her stance as well, with her either uttering apologies when killing someone, or letting out an impassioned "DIE!" when dispatching a foe. It's a nice touch, making one feel as if Iji herself is evolving as a character, and adds yet another tangible reflection of a chosen play style.

There remain two other honourable mentions one must make about Iji. Firstly, its replayability is superb, courtesy of both its nonlinear design as well as a wealth of hidden areas and unlockable content. Secondly, an absolutely incredible soundtrack that creates an amazing atmosphere for each of the levels – the final boss music and the credits track in particular must be heard to be believed. These two little touches of polish raise Iji above and beyond the regular Indie offerings, and make the game far better for it.

It's difficult to refrain from gushing about Iji. Daniel Remar has done himself proud with this game, creating something both fun and inspirational. Not bad for one man using Game Maker.

Killer Worm 2

Killer Worm drops the player behind the wheel of a giant killer worm, ready to lay waste to the highly nutritious inhabitants of the game world. Murder! Destruction! Pixilated slaughter! And all the player has to do to achieve this wormy goodness is lift a single finger. The controls couldn't be simpler; the leap action is bound to left click, and aiming is governed by the mouse.

Chris "LionsInnards" Dudley

At first, Killer Worm is quite difficult to classify due to the combination of different elements from various genres. There is an RTS-like resource system demanding that one plays smart and makes tactical decisions, and a serious dependence on the score to keep the player motivated – reminiscent of old school arcade games.

The strategy aspects come in to play when the player has to manage a hunger-meter, which requires the player to strike a balance between exposing themselves to danger, and grabbing themselves a slice of civilian tartare – all while reveling in the cheesy b-grade feel of the game's visuals and music.

The Killer Worm franchise is now in its third iteration, and it has improved noticeably. The basic elements of the first game formed a great base for Game.Dev community regular and developer Unc1354m to expand on, as his original version of the game was lacking in a few areas. The most glaring was the repetitive and restricted action available to the player; a problem easily remedied in his sequels, in which range and angle were taken into consideration when coming in for an attack. This served as a great way to break a lot of the monotony of the first game, and to show off the updated animations and growth systems.

Killer Worm is a great reference to show how evolving your ideas can pay off. Unc1354m should be commended for his dedication to his titles, and is a shining example for new developers disheartened with their own. Everyone is encouraged to follow his example and pour their all into their pet project – you never know what it could blossom into.

WHEN WORLDS COLLIDE

The Basics of Collision Detection in 2D
PART 1

Claudio "Chippit" de Sa

Almost every video game needs to respond to objects touching each other in some sense, a practice commonly known as collision detection. Whether it's simply to prevent the player character from walking through the walls of your maze with a simple collision grid array, or if it's to test if any of the hundreds of projectiles fired by a boss character in a top-down shoot-'em-up have struck the player's ship, your game will likely require a collision detection system of one sort or another.

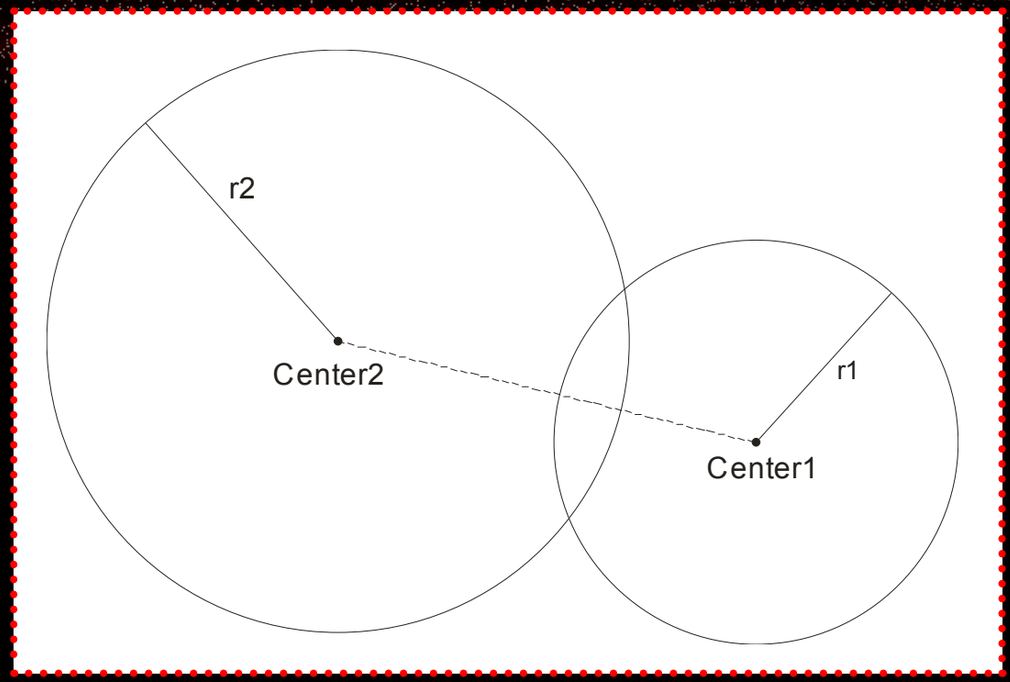
So there comes a time in almost every game's development cycle when an important choice needs to be made: how accurate should the collision detection be, and which method should be used to achieve that accuracy. This is a decision that is not made lightly, since it can drastically affect both gameplay and performance of your game.

Unfortunately, there's often no avoiding the mathematics behind collision detection. However, anyone with so much as a secondary-school maths education will be able to follow the collision detection explanations in this article.

Bounding sphere/ circle test

< A sprite with visible circular collision bounds

The simplest of all methods for detecting intersections between objects is a simple bounding sphere test. Essentially, this represents objects in the world as circles or spheres, and test whether they touch, intersect or completely contain each other. This method is ideal when accuracy is not paramount, for objects roughly circular in shape, or in instances where these objects do a lot of rotations.



Each object will have a bounding circle defined by a centre point and a radius. To test for collision with another bounding circle, all that needs to be done is compare the distance between the two centre points with the sum of the two radii:

- If the distance exceeds the sum, the circles are too far apart to intersect.
- If the distance is equal to the sum, the circles are touching.
- If the distance is less than the sum of the radii, the circles intersect.

```
CircleCircleCollision
Input
    Center1   x/y pair of floating points    Centre of first circle
    R1        Floating point                 Radius of first circle
    Center2   x/y pair of floating points    Centre of second circle
    R2        Floating point                 Radius of second circle
Output
    True if circles collide

Method
    // Calculate difference between centres
    distX = Center1.X - Center2.X
    distY = Center1.Y - Center2.Y
    // Get distance with Pythagoras
    dist = sqrt((distX * distX) + (distY * distY))
    return dist <= (R1 + R2)
```

The above method can be optimized somewhat by comparing the square distance with the square of the sum of the radii instead, saving a comparatively slow square root operation, as shown below.

```
    // Get distance with Pythagoras
    squaredist = (distX * distX) + (distY * distY)
    return squaredist <= (R1 + R2) * (R1 + R2)
```

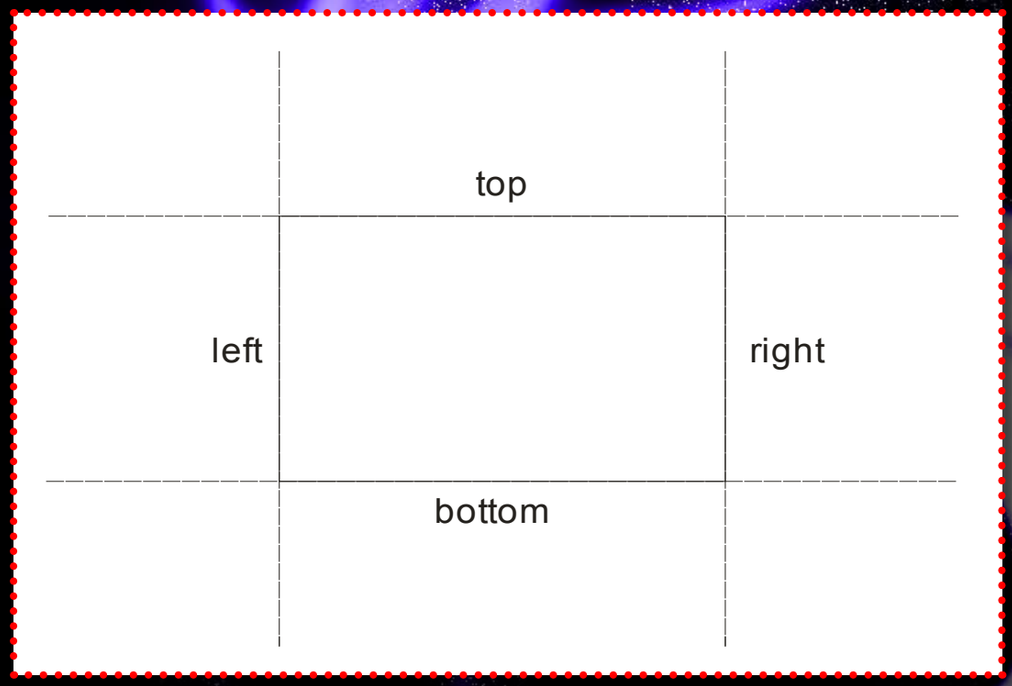
Bounding box/ rectangle test

<A sprite with visible rectangular collision bounds

The second obvious solution to the problem is to represent obstacles as axis-aligned rectangles. This method is ideal for smaller objects that are roughly rectangular and because it is incredibly fast to process.

The method that will be described uses a contradiction to determine whether the rectangles intersect. Because it is simpler instead to determine whether rectangles do not intersect, the function will calculate that and return the negation of its result.
Rectangles will be defined by their left-, top-, bottom- and right-edges. To determine whether two rectangles do not intersect, one simply has to check for any of the following conditions:

- Rectangle 1's bottom edge is higher than Rectangle 2's top edge.
- Rectangle 1's top edge is lower than Rectangle 2's bottom edge.
- Rectangle 1's left edge is to the right of Rectangle 2's right edge.
- Rectangle 1's right edge is to the left of Rectangle 2's left edge.




```
RectRectCollision
Input
    Rect1     Rectangle      First Rectangle
    Rect2     Rectangle           Second Rectangle
Output
    True if the rectangles collide

Method
OutsideBottom = Rect1.Bottom < Rect2.Top
OutsideTop = Rect1.Top > Rect2.Bottom
OutsideLeft = Rect1.Left > Rect2.Right
OutsideRight = Rect1.Right < Rect2.Left
    return NOT (OutsideBottom OR OutsideTop OR OutsideLeft OR Out-
sideRight)
```

The above can then be condensed into a single line as follows.

```
    return NOT (
         (Rect1.Bottom < Rect2.Top) OR
         (Rect1.Top > Rect2.Bottom) OR
         (Rect1.Left > Rect2.Right) OR
         (Rect1.Right < Rect2.Left) )
```

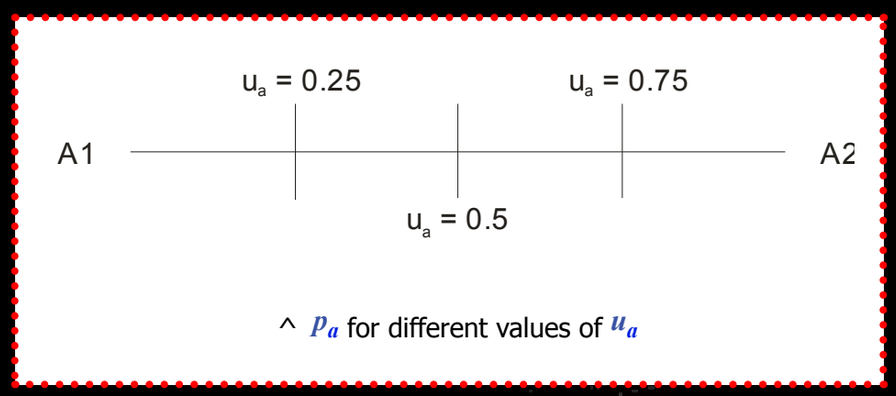
Line-Line intersection test

Occasionally, one would need to represent objects in the game as simple lines (or perhaps a compound group of lines). Testing collision between two lines is slightly more complicated that the two methods described above, requiring some algebra to develop an algorithm, but is still otherwise fairly simple.

All lines used in this method will be represented as two points along the line (or the two endpoints if the lines are to be considered line segments instead of lines of infinite length). With a line A of infinite length, and two points on that line, A_1 and A_2, we define point P_a, any point on line A as:

$P_a = A_1 + u_a (A_2 - A_1)$ with u_a being any real number.

That is, P_a is a point U_a-percent along the line from A_1 to A_2. Note that, since Ua is any real number, this point (and therefore the intersection between the lines) can lie on any point on either side of A_1 and A_2. Conversely, if u_a is between 0 and 1, P_a is between points A_1 and A_2 (this is important later).



^ p_a for different values of u_a

Similarly, we define point P_b, any point on line B, as:

$P_a = B_1 + u_a (B_2 - B_1)$ with u_b being any real number and B_1 and B_2 as two points on line B

Solving for the point where $p_a = p_b$ (the intersection between these lines), we get two equations:

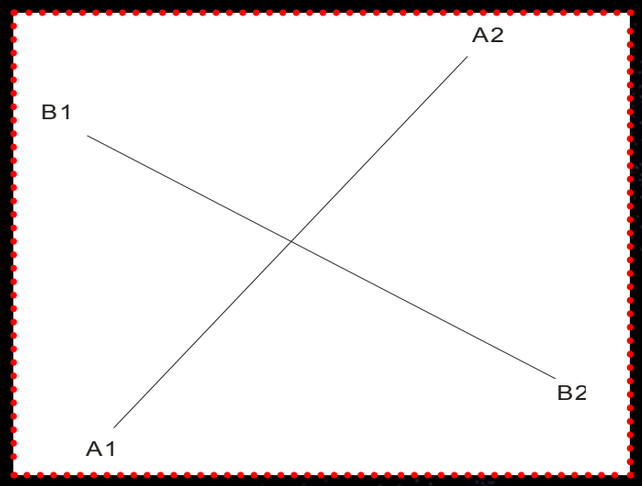
$$x_{A1} + u_a (x_{A2} - x_{A1}) = x_{B1} + u_b (x_{B2} - x_{B1})$$
$$y_{A1} + u_a (y_{A2} - y_{A1}) = y_{B1} + u_b (y_{B2} - y_{B1})$$

Solving the above for u_a and u_b gives:

$$u_a = \frac{(x_{B2} - x_{B1})(y_{A1} - y_{B1}) - (y_{B2} - y_{B1})(x_{A1} - x_{B1})}{(y_{B2} - y_{B1})(x_{A2} - x_{A1}) - (x_{B2} - x_{B1})(y_{A2} - y_{A1})}$$

$$u_b = \frac{(x_{A2} - x_{A1})(y_{A1} - y_{B1}) - (y_{A2} - y_{A1})(x_{A1} - x_{B1})}{(y_{B2} - y_{B1})(x_{A2} - x_{A1}) - (x_{B2} - x_{B1})(y_{A2} - y_{A1})}$$

You'll notice that the denominator for these two equations is the same. Thus, if the denominator is 0, both u_a and u_b are undefined, and no collision between these lines exist (the lines are parallel). The algorithm for simple infinite-line length checks simply needs to calculate the value for this denominator to determine whether a collision exists.



```
LineLineCollision
Input
    LineA1    Point     First point on line A
    LineA2    Point     Second point on line A
    LineB1    Point     First point on line B
    LineB2    Point     Second point on line B
Output
    True if lines collide

Method
    denom = ((LineB2.Y - LineB1.Y) * (LineA2.X - LineA1.X)) -
         ((LineB2.X - lineB1.X) * (LineA2.Y - LineA1.Y))
    return denom != 0
```

However, there will be occasions when either one or both of the lines to be tested are not lines of infinite length. In this case, as mentioned above, ua and ub become really important. If ua is between 0 and 1, then the collision occurs on a line segment of line A, between points A1 and A1, and, similarly, if ub is between 0 and 1, the collision occurs on a line segment of line B, between points B1 and B2. More often than not, you'll want to perform both these checks in your collision routine.

And that's all we have space for this month. Next issue we'll look at getting more information out of some of these collision routines, as well as covering a few more complicated algorithms.

Rodain "Nandrew" Joubert

Adobe The FLASH

Flash for Free: How to become a Flash developer without spending a cent.

Read ↓

Not copyright → infringement

Many of you have experienced it. You go online, log into your instant messenger or e-mail account and receive a link from some excited friend saying, "ARARRRAARARR PLAY THIS GAME ASBFLARGAFUG www.insertrandomlinkhere.com!!!!!". Chances are it's either an Internet booby trap (if you have those kind of friends) or an awesome browser-based something brought to you by Adobe Flash; and if you're a keen game developer, you've probably slobbered at the idea of creating some of these cool games yourself.

MEANWHILE...

MOST PEOPLE BELIEVE THAT FLASH DEVELOPMENT IS SOMETHING REMOTE OR INTIMIDATING, A CRAFT WHICH IS DIFFICULT TO GET INTO AND EVEN MORE DIFFICULT TO PERSIST WITH. HOWEVER, THIS NEED NOT BE THE CASE - ALL YOU NEED IS A GOOD SET OF GUIDE POINTS TO WORK FROM AND, OF COURSE, THE RIGHT SOFTWARE (ON TOP OF THAT, KNOWING JAVA WILL HAVE FLASH PRACTICALLY FALL INTO YOUR LAP).

THIS IS A COMPREHENSIVE KICK-START TO FLASH WHICH WILL TAKE YOU THROUGH THREE BASIC SECTIONS:

1. SECURING (FREE) SOFTWARE
2. PROJECT STRUCTURE AND IDE ENVIRONMENT
3. LEARNING BASIC CODE WITH ACTIONSCRIPT 3

AFTER GOING THROUGH THESE THREE BASIC SECTIONS, YOU SHOULD HAVE A WORKING IDEA OF WHAT DEVELOPING IN FLASH REQUIRES FROM YOU. FROM THERE, IT'S ONTO INTERNET TUTORIALS AND OTHER THINGS TO GET TO GRIPS WITH CODING AND ADVANCED TOOLS. LET'S GET STARTED.

1. YOUR TOOLS

Like many programming languages, Flash requires three primary components: (1) the IDE (or "where you slap the code in"); (2) the SDK (or "the files on your hard drive which let your computer understand the code stuff") and (3) the debugger (or "the program which runs the final stuff and helps you see if you wrote it all right-like").

For a coding environment, we'll be using the excellent and totally free FlashDevelop IDE **(http://www.flashdevelop.org/)**. It's only a few megabytes to download and has all of the features that one may expect from a competent IDE, including possibly the most awesome code completion system that you'll ever lay eyes on. This is the program that will be visible to you, the user, whenever you want to show off your coding biceps and make something fantastic. Install it, but don't double-click that little icon just yet! We want to get some other stuff onto your system first.

The FLEX SDK is a group of files which needs to sit on your system before you can develop in Flash. These files are used in building the final Flash applications after you've done the hard work of laying down the code, and will be used by the FlashDevelop IDE when you get around to opening it up.

There are numerous SDK downloads at: **http://opensource.adobe.com/wiki/display/flexsdk/Download+Flex+3**

Once you've picked something to download (hopefully a fairly recent build under the "Adobe Flex SDK" category), grab the brand spanking new ZIP file that appears on your hard drive and extract it to a place on your drive where you won't forget about it (for example, C:\flex_sdk_3). Seriously, don't forget where you've put it.

Finally, you'll probably also want to look at downloading some runtime files and debuggers. Browse through the files on this page: **http://www.adobe.com/support/flashplayer/downloads.html**

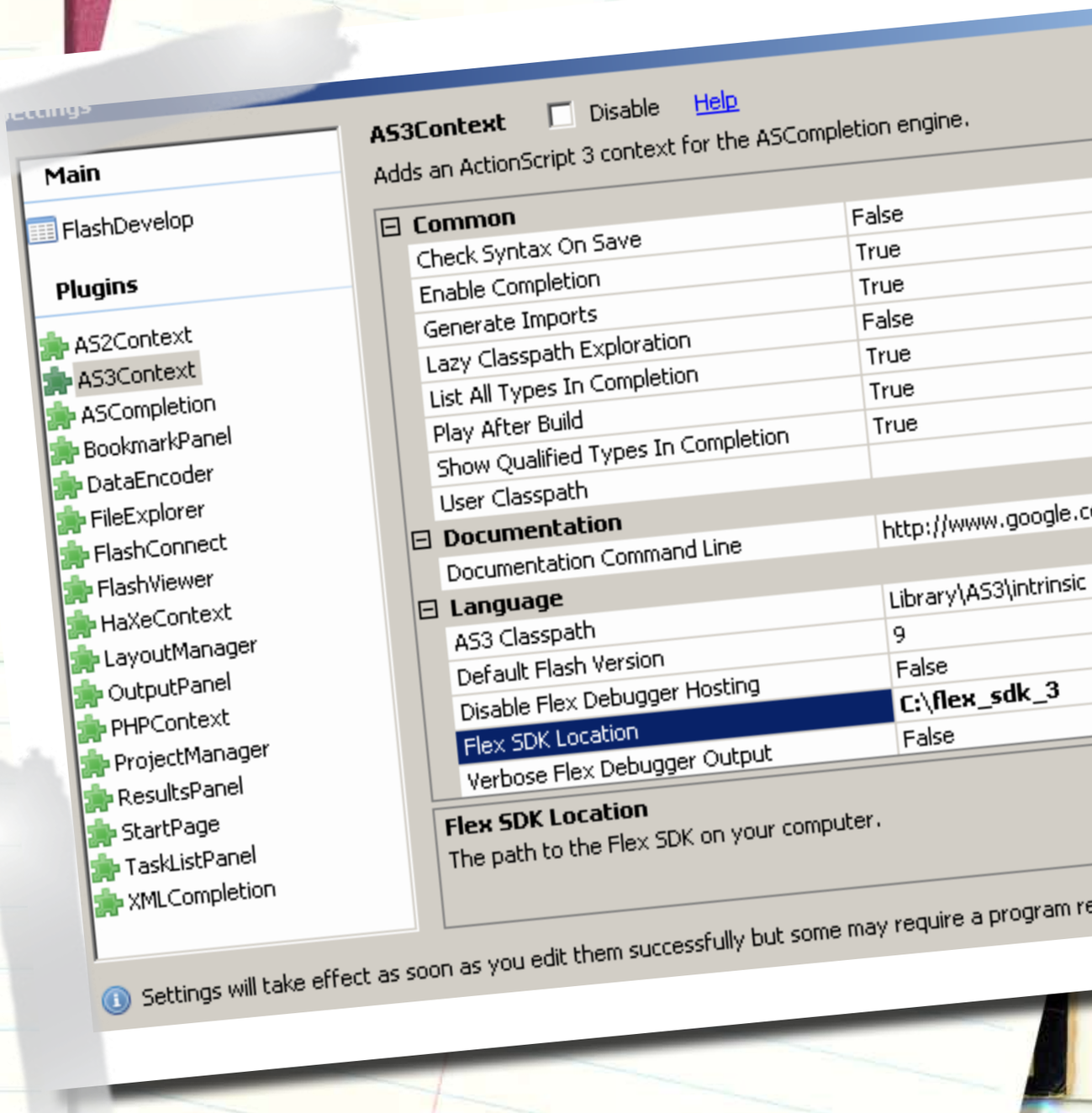
Recommended downloads are the Flash Player 9 ActiveX control content debugger and the Flash Player 9 Projector content debugger.

After that, you're pretty much sorted for basic downloads. Install applications as necessary.

2. YOUR WORKING ENVIRONMENT

There's a universal consent that part 2 is by far the most boring, and people are tempted to pull an Underpants Gnome trick (don't ask) and go carefully through steps 1 and 3 while leaving a big set of question marks over 2. Please don't skip this section. Your programs will screw up, you'll fail to compile properly and the world will explode. Worst of all, you won't fully understand why all this happens, and many, many programming woes emerge purely because the user doesn't know what's going on.

Right, go ahead and open FlashDevelop. Then take a moment to pause and read through the next few paragraphs before doing anything else. There are a few things that need to be set up in FlashDevelop before you even open an AS3 project. First off, look for Tools in the menu bar and proceed to Program Settings. Select the AS3Context plugin and scroll down to the "Flex SDK Location" field. Now, remember earlier when we extracted the SDK files onto the hard disk? Type that location into the text field, or click on the ellipsis (three little dots) and browse for it. Hopefully you haven't done something stupid, like forgetting where you extracted those files to. If you have, slap yourself on the face and go back to step one. Everyone else – congratulations! You've just told FlashDevelop where the SDK is. Now it can actually figure out what to do with the code when you click that big green "GO!" button.

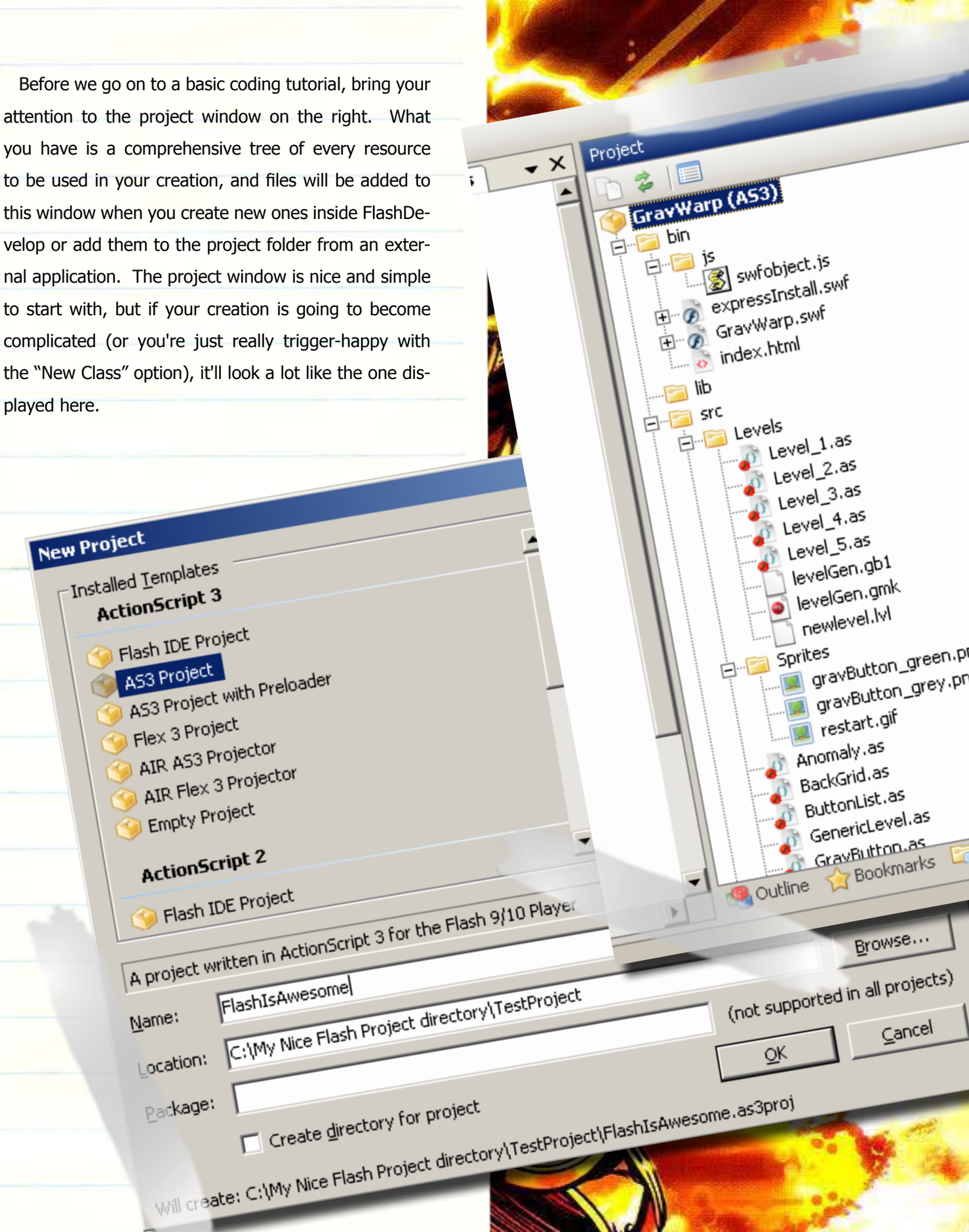


For beginners who aren't familiar with this concept, take note: FlashDevelop is just a development environment. This means that it can watch you code, help you auto-complete, and generally provide a comprehensive structure to organise and streamline your development experience. It cannot, and never will, be useful in debugging, compiling, interpreting, linking, running, fishing, baking or writing philosophy essays. It needs the SDK for that stuff in the same way that a really awesome refrigerator can provide shelving and storage for your food but won't ever prepare dinner for you (unless you count the stuff that always grows right at the back). If you try to make a Flash program without an SDK present, you'll come short very, very quickly.

A secondary, albeit slightly less critical action to perform is establishing a folder on your hard drive for all of your Flash projects. After all, your projects don't consist of one or two arbitrary files that can just be shoved anywhere and recognised easily. They're entire directories (and sub-directories) of project shortcuts, source code and embedded files which all contribute to your final product, and it's preferable if you don't just shove them into the mess of C:\various_rubbish. So go ahead – create a nice neat "Flash Projects" folder somewhere on your drive. You'll thank me for it later.

Now that you've got all this stuff out of the way, go to the Project > New Project menu option. There's quite an array of project types that you can start up here, but it's a good idea to start up a basic AS3 Project. Type in an appropriate project name and directory (remember the folder you set up just now?) and then continue. You should now have a fresh new workspace to operate with.

Before we go on to a basic coding tutorial, bring your attention to the project window on the right. What you have is a comprehensive tree of every resource to be used in your creation, and files will be added to this window when you create new ones inside FlashDevelop or add them to the project folder from an external application. The project window is nice and simple to start with, but if your creation is going to become complicated (or you're just really trigger-happy with the "New Class" option), it'll look a lot like the one displayed here.



3. YOUR "HELLO WORLD"

Standard AS3 projects in FlashDevelop are divided into three primary folders:

BIN – no, this is NOT where you throw your rubbish! The BIN (or binary) folder contains the final product, the sum of your efforts, the stuff that an end user will pick up and look at. The two most important files for you to worry about over here are index.html (a Webpage template that can test how your project appears in a browser – if you know HTML, you can fiddle about with this) and a .swf file named after your project (only visible after your first build/run, so don't panic if you don't see it yet). The latter is your executable Flash program.

LIB – you can worry about this later. You know, when you're pro and stuff. It contains things.

SRC – this is where all your hard work takes place before you send it to your executable, and will be the folder in which you spend most of your time. SRC can contain many things. For a start, it holds "Main.as", the entry point of your program and the place where you'll begin your coding. Later on, it'll hold all of your various source code files, embedded images / sounds and, well, whatever else you want to put in there (ooh, look, someone smuggled in a Game Maker 7 file!).

Hopefully you have a slightly better idea of how things work in FlashDevelop now. Feel free to hate me, but knowledge is good. Seriously. Besides, we're getting on to the cool bit now.

Right, now for the implementation of all your hopes and dreams! By now you should have a nubile young AS3 project parading about on your screen (if you don't have one, you obviously didn't read part 2. Didn't I just tell you to read part 2?). Double-click on the Main.as file to open it in your editor window (the file is squirreled away inside your project's SRC directory, remember?). You'll be presented with what us fancy types like to call a code stub – this is FlashDevelop's way of saying, "You're lazy and/or stupid, so we've written out some framing code for you already. Now you connect the dots."

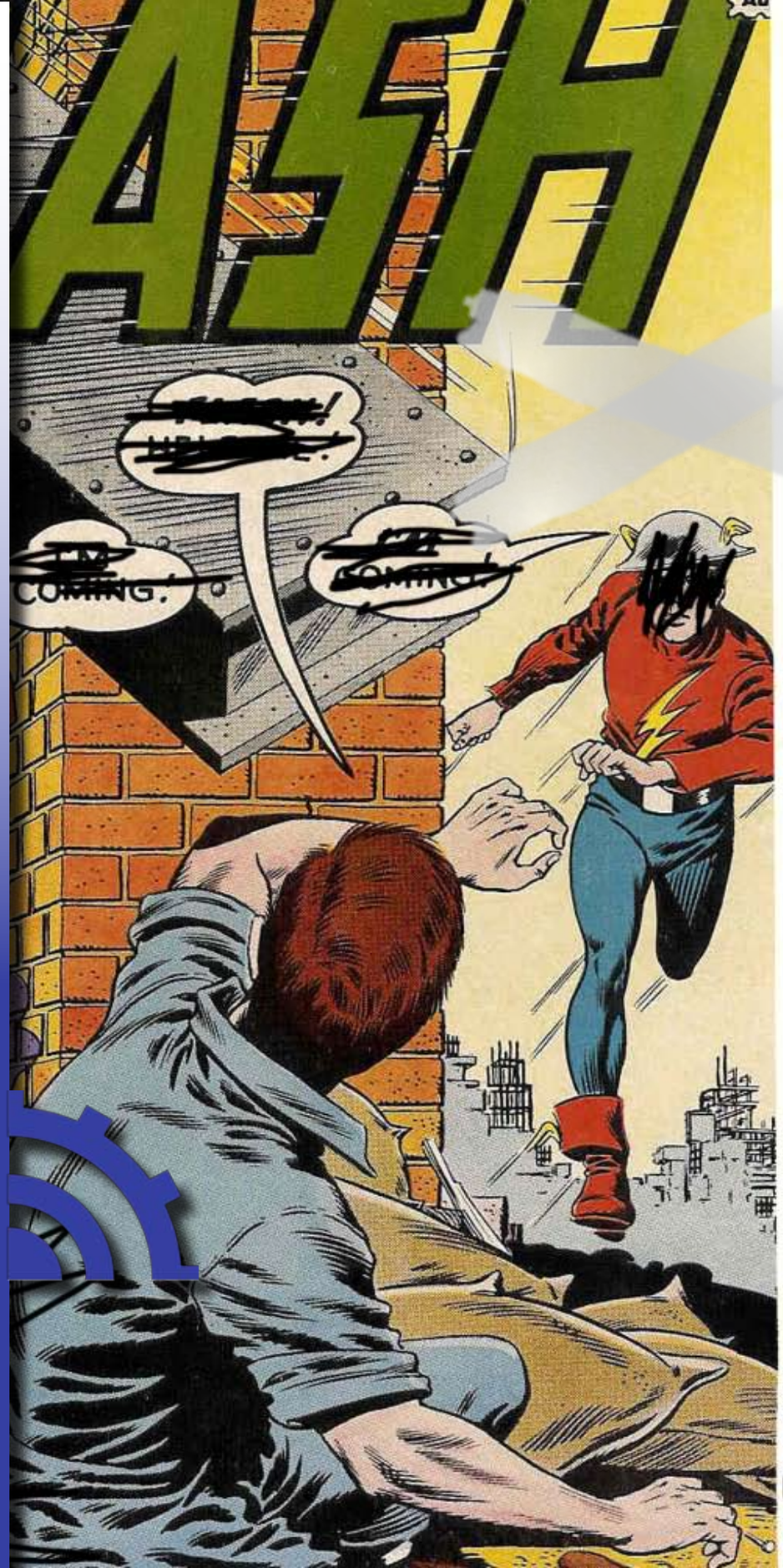
FlashIsAwesome - FlashDevelop

File Edit View Search Insert Project Tools Syntax Help

Debug

Main.as

```
1  package
2  {
3      import flash.display.Sprite;
4      import flash.events.Event;
5
6      /**
7       * ...
8       * @author DefaultUser (Tools -> Custom Arguments...)
9       */
10     public class Main extends Sprite
11     {
12
13         public function Main():void
14         {
15             if (stage) init();
16             else addEventListener(Event.ADDED_TO_STAGE, init);
17         }
18
19         private function init(e:Event = null):void
20         {
21             removeEventListener(Event.ADDED_TO_STAGE, init);
22             // entry point
23         }
24
25     }
```



You'll be presented with one or two "import" lines (in simple terms, each one extends the commands and/or objects that you have access to in this particular file), then a public Main class which contains two default functions: one called Main() – (code that runs as soon as the program starts) – and another called init() – (code that runs just a short while after the code in Main()).

You may notice that the code refers to something called "stage". "What's that?" you may wonder. Well, AS3 applications are based on a great big parent/child structure. This "stage" object is a universal instance (visible to most default object types) which serves as a great big canvas on which you can stick everything else. So if you want a soccer ball to appear in your product, you'll write something along the lines of "stage.addChild(soccerball_thingie);" in your code. If you want ten soccer balls to appear on-screen, you'll add ten of these "children" to the "parent" stage. This is all done after you've successfully created a soccer ball object, of course.

So, let's make our first project stick with the classic "Hello World". Actually, heck, let's be adventurous! We'll make it write "Hello Flash!" in nice friendly letters.

Go to your init() function just below the point where it says "entry point" in comment marks. This is where we'll be writing our code. "Why not do stuff in Main()?", you may ask. Well, Flash has a few funny problems with initialising stuff before it's properly added to the stage, and the Main instance has to be added just like everything else. Don't worry about this technicality too much – the code stub provided will make sure that it automatically adds itself before it does anything, so just stick to that friendly little "entry point" reminder until you know enough ActionScript to screw about properly.

Let's throw in the following lines of code:

```
var myText: TextField = new TextField();
myText.text = "Hello Flash!";
stage.addChild(myText);
```

These lines, in order, do the following:

(1) Create a new reference to a text object, then instantiate it. This means that somewhere in the tiny little universe of your Flash project, there will now be a little man with a sign pointing at something and going "Look! I've got a textbox here that you can use!"

(2) We've just told the little man to go with a pencil and etch something into the textbox – because it doesn't have anything in it yet.

(3) Now we've grabbed our stage object and, as mentioned earlier, added the text object as a child. So now our little man is no longer floating around – he's been put onto the stage to perform for a great big audience, using his talents as a Professional Textbox Tamer to wow the crowd. Amazing stuff.

Right, let's attempt a project build so that we can run this thing. Go to Project > Build Project in the top menu.

Uh oh. Looks like the program threw some errors.

Fortunately, these ones are easy to fix. Go to the "import" lines in your code again. Remember that it was mentioned how these had this sort-of ability to extend the command set of your program? Well, we need to extend it now to recognise these text objects. S o write this line with the other imports:

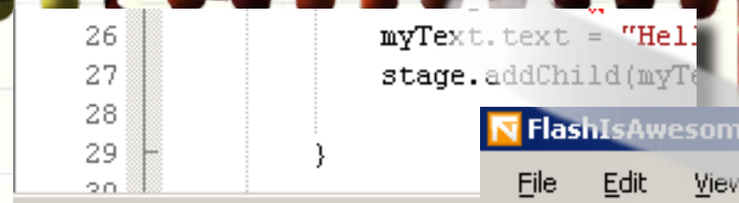
```
import flash.text.TextField;
```

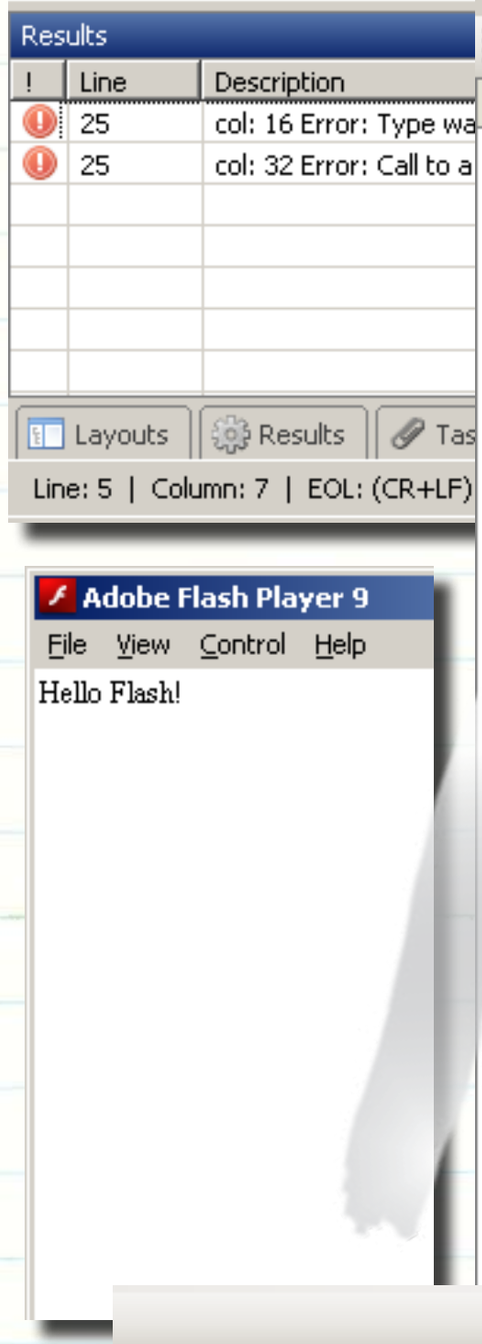
There we go! Try building the project again. It should go without a hitch.

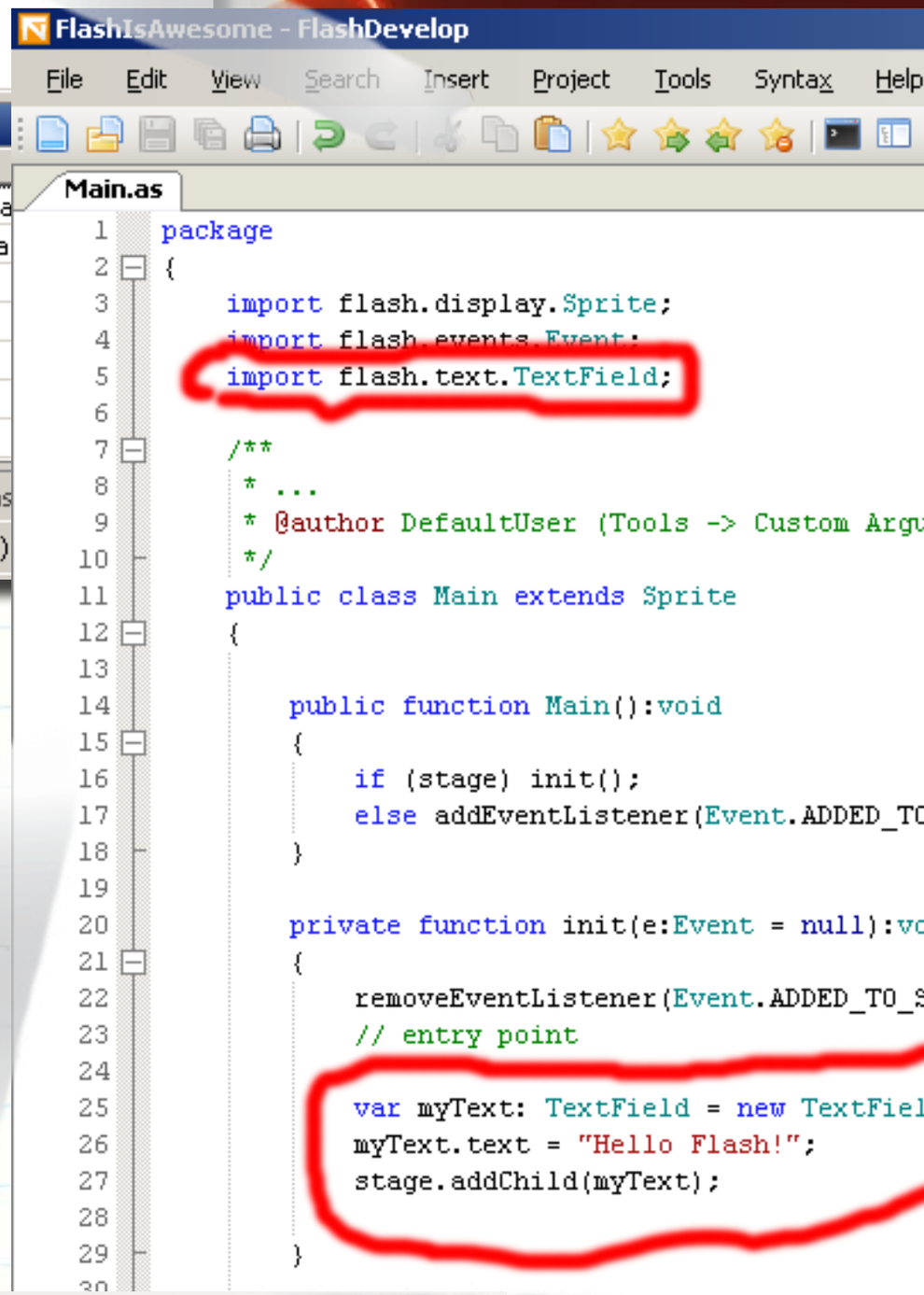
Once the build is complete, run your project.

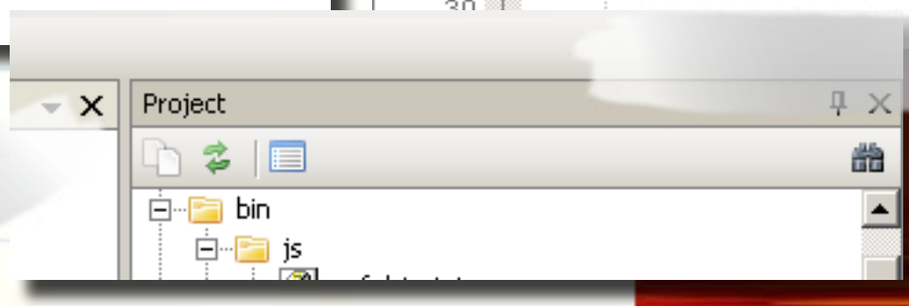
Brilliant, you've made your first Flash application, and you did it without loads of money!

As a final parting shot: tweaking some of the output properties is really easy to do with FlashDevelop, and you may want to remember these for future reference. Click on the rightmost icon at the top of the Project window to get access to project properties. This will offer various output options (and many other adjustments) which developers should probably have a look at.









AT THE END OF THE DAY

Now, use this beginner's guide as a springboard, look up coding tutorials on the Internet (make sure that they deal with ActionScript 3, and not the Adobe Flash creator application or something else) and learn how to create your own Flash games like a pro. It's a fun and rewarding language that, as you may soon discover, is powerful enough to do a great many things and reach a whole lot of people. Happy scripting!

THE FLASH

WE'D LIKE TO THANK DC COMICS AND ADOBE FOR PROVIDING THIS OPPORTUNITY. SORRY.

2009!

Celebrations! And a happy new year to all of our readers! With 2009 well on it's way (can you believe it's the end of January already?) some of the Dev.Mag crew threw together some thoughts and anticipations for the year ahead. We're looking forward to many different things that will be coming this year, but most of all, we look forward to sharing it all with you!

2009! What they have to say about it

Simon "Tr00jg" de la Rouviere

My wish for 2009 is to see more exploration platformers in the vein of Knytt. I love getting lost in the ambient exploration of other worlds. I love discovering interesting new environments and just letting my imagination run loose on why the world is like it is. If I were to develop one, I would centre it on the idea of an old lost civilization/city. At first you only see its ruins, but as you go on, you slowly reawaken the city piece by piece to its former glory.
[Play Aquaria, wouldya? – Ed]

Rodain "Nandrew" Joubert

Independent Games Festival 2009

To be frank, the annual IGF should be on EVERY aspiring game developer's froth-at-the-mouth list for 2009. Now in its 11th year, the Independent Games Festival is a great showcase for most of the top indie titles out there. In recent years, it has featured and honoured gems such as World of Goo, Crayon Physics Deluxe, Aquaria, Audiosurf, Darwinia and Braid.

If you're fortunate enough to be within reach of the IGF (March, in San Francisco), snap up the opportunity to attend and ogle. For the rest ... well, this year's finalists have just been announced, so hike on over to **http://www.igf.com/** and check out the quality of the games featured. Chances are that you won't be disappointed.

Claudio "Chippit" de Sa

Xbox Community Games is the big landmark name for this year, and you'll probably be hearing a lot about it in the coming months. We're all anxiously awaiting the day when we can gladly and truthfully say that a Game.Dev creation is available for sale on a worldwide scale; whether this game is SpaceHack or Ultimate Quest (or another local project) is irrelevant, or, in fact, whether it's on XBCG or Steam, or any other distribution platform available to a wide audience yet accessible to indies.

Tied in with that is anticipation for DreamBuildPlay 2009. We're hoping to see an even greater Game.Dev presence there this year, and many plans for projects are already floating around long before the start of the competition. Are we getting ahead of ourselves? Probably, but we don't care; we enjoy it so much.

Robbie "Squid" Fraser

Going to study IT and hopefully acquiring some nifty programming tricks. There, and with my new 360 controller, I can look forward to getting involved with XNA and learning C#. There's also the new possibility of making some cash writing for cellphones (yay, monies!). Then there's rAge 2009, which promises to be an awesome weekend as always. I also cannot wait to see more of SpaceHack, the game has loads of potential and I can't wait to see where it goes. Finally, the evolution of Dev.Mag itself; the magazine keeps going from strength to strength. It's actually rather exciting.

IN CASE OF EMERGENCY
BREAK THE MOULD

Object Pascal has a lot to offer...

www.PascalGameDevelopment.com

FEAR COUNT:

WE MANAGED TO TERRIFY 5 PEOPLE THIS MONTH.

3 CHILDREN.

ONLINE

DEV.MAG

CREATE • DEVELOP • EXPERIENCE

WWW.DEVMAG.ORG.ZA