Our Properties: Gamasutra GameCareerGuide IndieGames GDC Vault GDC IGF

GAMASUTRA The Art & Business of Making Games

SEARCH GO LOGOUT GAME JOBS
UPDATES BLOGS CONTRACTORS NEWSLETTER STORE SEARCH GO
ALL CONSOLE/PC SMARTPHONE/TABLET INDEPENDENT VR/AR SOCIAL/ONLINE GAME DEVELOPER ON GAMASUTRA

David Lightbown
My Account
My Profile
Blog Now
[LOGOUT]

PROGRAMMING
ART
AUDIO
DESIGN
PRODUCTION
BIZ/MARKETING

Latest Jobs
View All RSS
January 9, 2018
- Phosphor Games Studio
Mid to Senior Gameplay Programmer
- Blackstorm
Studio Game Engineer
- Naughty Dog
Web Developer
- Naughty Dog
IT Help Desk Generalist (Temporary Assignment)
- Trendy Entertainment, Inc.
Software Engineer - Game and Online Services
- Naughty Dog
Scripting / Systems Designer (Single Player)

Latest Blogs
View All Post RSS
January 9, 2018
- Endless Space 2 - E.N.F.E.R. Alternate Reality Game
- Splatoon 2: How it Encourages Movement
- Record over $3B AR/VR investment in 2017 ($1.5B+ in Q4)
- The limitations of matchmaking without server logic
- Minimalist container library in C (part 1)

Press Releases
January 9, 2018
Games Press
- Ankama's Nindash Is Coming for Mobile Phones
- UBISOFT ANNOUNCES THE FATHER'S CALLING AN...

Blogs

Classic Tools Retrospective: Tim Sweeney on the first version of the Unreal Editor

by David Lightbown on 01/09/18 04:04:00 am Featured Post

3 comments [Edit Post]

The following blog post, unless otherwise noted, was written by a member of Gamasutra’s community.
The thoughts and opinions expressed are those of the writer and not Gamasutra or its parent company.

Introduction

In recent years, retrospectives of classic games have been well received at GDC, but there have been very few stories about classic game tools. This series of articles will attempt to fill that gap, by interviewing key people who were instrumental in the history of game development tools.

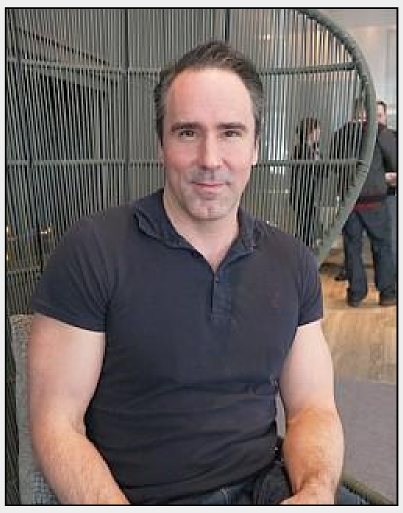
The first article in this series was an interview with John Romero about TEd, the Tile Editor he created at Id Software, that went on to ship over 30 games.

For the second article, I am fortunate to have the opportunity to speak to **Tim Sweeney** about the first version of the **Unreal Editor**, or **UnrealEd.** We spoke at the Unreal booth during the GamesCom 2017 conference in Cologne, Germany.

Fallacies and BSPs: "I'll write an editor"

DL: Thank you for taking the time to talk to me, Tim! Let's start with the early days of the Unreal Editor. I read that James Schmaltz – creator of Epic Pinball – showed you a game that he was working on, and when you saw it, you offered to build an editor for it. Is that correct?

TS: Yeah! He had been inspired by Bullfrog’s game, Magic Carpet. James is this insanely brilliant programmer, but he only wrote code in assembly language, he didn't want to learn C. [laughs] And so – in pure assembly language – he wrote this 3D engine that rendered terrain backgrounds and game objects. He didn't want to build an editing tool, so he actually built a BSP tree by hand and placed a capsule in the middle of this terrain. When I saw that, I was like "No, no, no, James, James... this is not how we do things." [laughs]



James Schmaltz and Magic Carpet, by Bullfrog

I said "I'll write an editor”, and so I set about building the user interface for the Unreal Editor, laying out the UI in Visual Basic, of all things. It had a text mode command-line interface to the C++ engine that was doing the rendering. Next I wrote the wireframe editor, and it went from there.

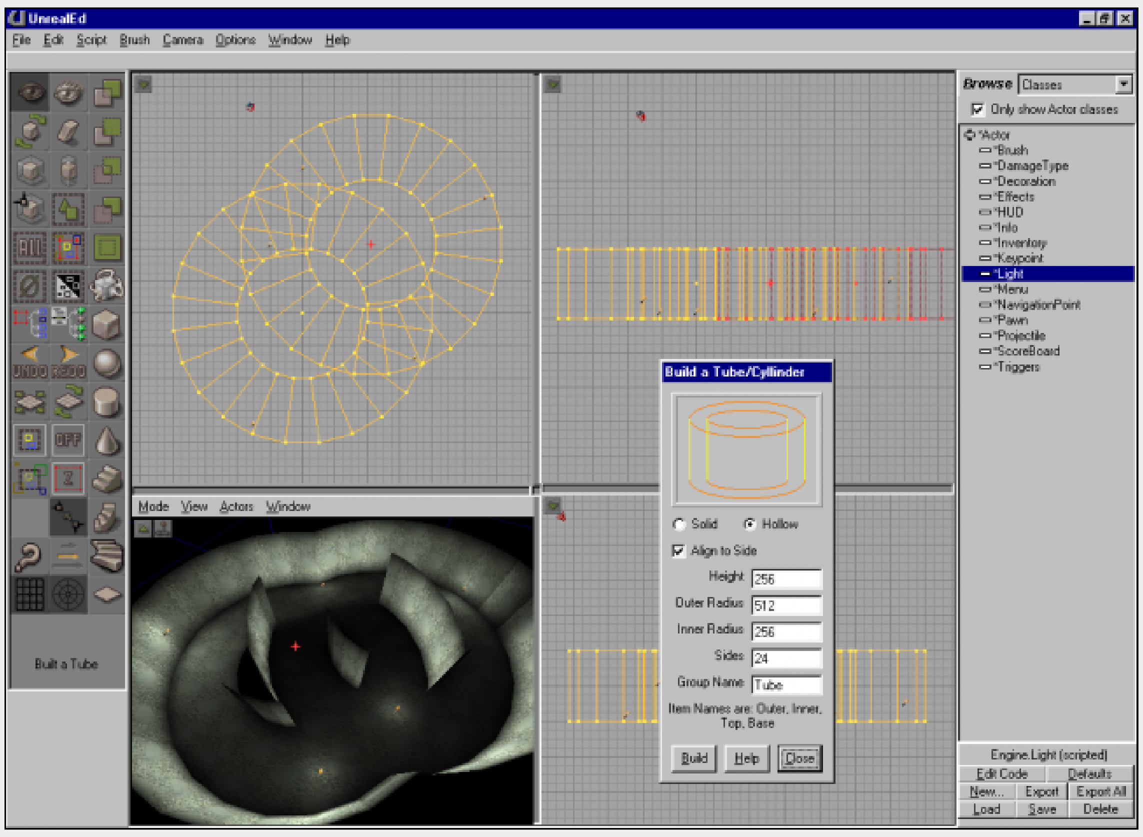
- ELECTRONIC ARTSâ€™ JADE RAYMOND JOINS ACADEMY OF...
- SWORDS OF CHULT UPDATE BRINGS NEW WAYS TO BATTLE...
- MY ARCADE DEBUTS NEW COLLECTIBLES LICENSED FROM...

View All RSS

So, it was a funny learning process. I thought I was just going to write this editor and integrate it into James's renderer. At some point, I said "Can you send me the code? I'd like to figure out how to integrate the renderer." So he sent me 30,000 lines of assembly code. [laughs] The 3D rendering engine had some elements of Epic Pinball in it and some other prior assembly code that he had been copy / pasting, and I was like "Oh my god, what is this stuff? I don't wanna touch this!" [laughs]

But I said to myself, before I figure this out, I'm just going to write a little texture mapper. So, I went through Michael Abrash's articles on texture mapping and looked at some stuff that Billy Zelsnack has shared early on. The texture mapping was actually pretty simple, so I said, "I guess I can figure this out".

The one real piece of wizardry in the early Unreal Editor – before I implemented lighting and things like that – was the real-time BSP tree creation. The idea is that you can reposition brushes in 3-D space, and then all the BSP work is updated completely in real-time.



Build a Tube/Cyllinder

Solid [✓ Hollow]

[✓] Align to Side

Height 256

Outer Radius 512

Inner Radius 256

Sides 24

Group Name Tube

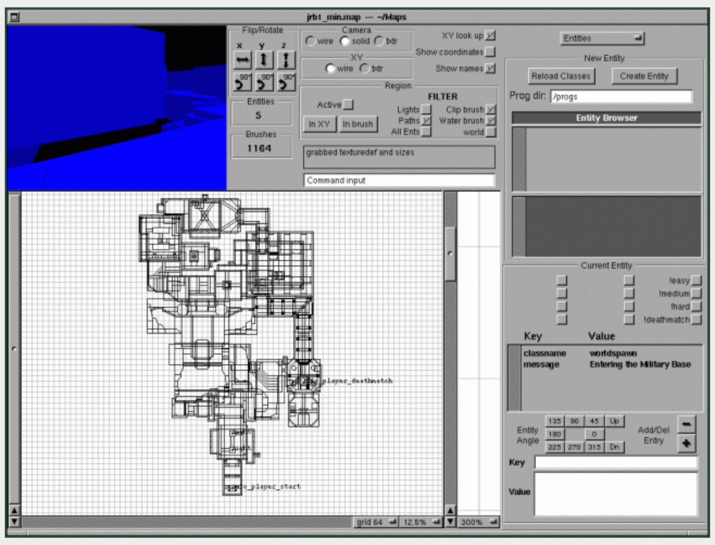
Item Names are: Outer, Inner, Top, Base

This had some mind-boggling implications, and I created this torus-building tool just to show off how cool it was. I got together with James – who, remember, had been building his BSPs by hand – and I said "check this out." I created two toruses that were interlocked, and subtracted them from the world, and he said "Whoa, no way? That's awesome!" It was a real example of programmer geekery.

"It was a real example of programmer geekery"

DL: Speaking of BSPs, it was my understanding that John Carmack was one of the first people to use BSPs in a game engine, and that the idea of working with BSPs was fairly new in the games industry at that time.

TS: Carmack had written this really advanced editor on the NeXT. I'd read all about it and I had seen screenshots of it, but I never actually used it. At the time, I thought to myself "Holy shit, Carmack wrote a real-time BSP editor!" What I didn't realize was that it wasn't actually real-time, there was this re-build process and all this other offline stuff. I didn't know that, and so I thought I had to create a completely real-time thing, and so I did. [laughs]



classname worldspawn

message Entering the Military Base

QuakeEd on NeXT and John Carmack

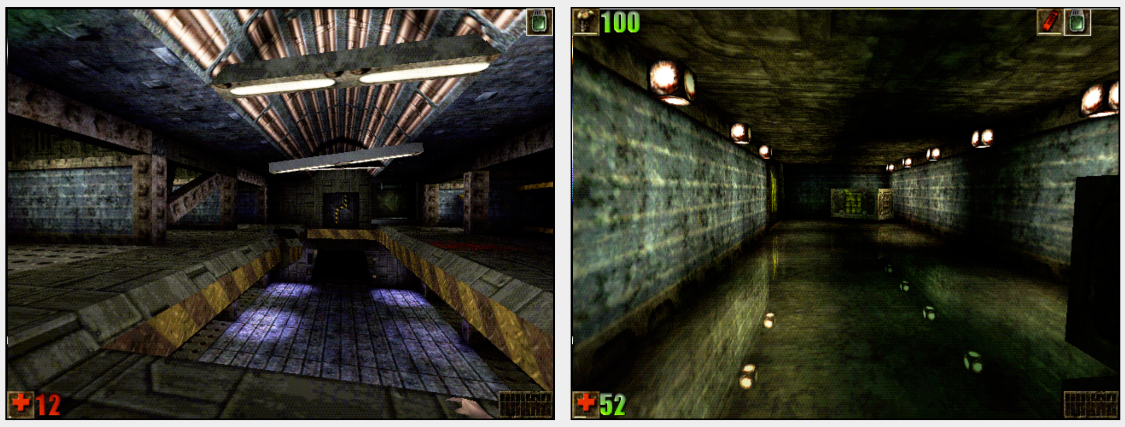
DL: [laughs] You thought that's how it was, and so you accepted the challenge.

TS: Yeah! A lot of the features in Unreal arose from fallacies of what I misperceived other people did.

"A lot of the features in Unreal arose from fallacies of what I misperceived other people did."

Also, a bunch of the former Future Crew demo-scene guys had formed a hardware company, and they had released some screenshots with incredibly realistic volumetric lighting in an indoor scene. There were some lights with really big spheres around them, and the volumetric lighting was really clearly clipped by all the geometry around it. It looked completely physically accurate. I was like "Oh my god, it's something I've never seen before, I have to figure it out!"

So I figured out that I needed to compute the line integral from the eye to every point on the screen. I learned some calculus in college, so I said to myself "I should be able to do this." So I figured out the formula for it with some crazy complicated trigonometry. I implemented it, but it was 100 times too slow. Then I realized, "Oh wait, I can do this in the lightmap space", because the lightmap is a discretization of geometry into bite-sized chunks. I did that with lightmaps, so it was real-time.



Examples of lightmap-based lighting from Unreal

I took a screenshot of it, and I emailed it to the guy I knew at the hardware company in Finland. He replied "Oh that's really awesome! But, our picture is just a rendering from 3D Studio Max because we couldn't figure out how to do that in real-time. [laughs]

DL: [laughs] Wow…

[Author's Note: The company that Tim is referring to is Bit Boys]

TS: So Unreal Engine was the first one with volumetric lighting, anywhere, I think... and it was based on that fallacy.

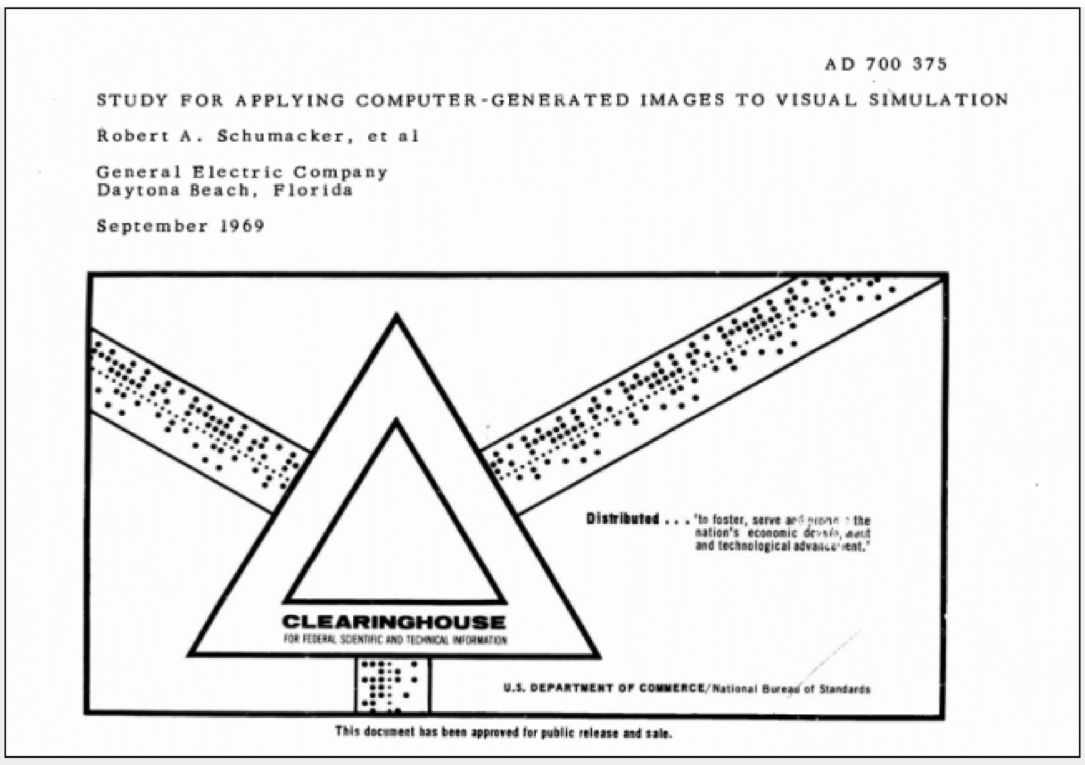
"There was this huge leap frogging process, where Carmack was implementing crazy new things, [and] I was implementing crazy new things"

It was a real cool time in the early history of the games industry, because 3D was just becoming possible. There were several software renderers that people had written, and nobody had really solved the large scale challenges of how to make lighting work in a large world, or how to make real time geometry work in a large world. There was this huge leap frogging process, where Carmack was implementing crazy new things, I was implementing crazy new things, and we were constantly releasing screenshots showing what was possible.

If you look at it, we – in about a four year period of time – re-created about 20 years of rendering research from the 1980s and 1990s that had previously only been possible offline and not in real-time.

DL: Right, like how the idea for BSPs was based on a paper written in 1969.

TS: Yeah, that's before I was born!



AD 700 375

STUDY FOR APPLYING COMPUTER-GENERATED IMAGES TO VISUAL SIMULATION

Robert A. Schumacker, et al

General Electric Company
Daytona Beach, Florida

September 1969

Distributed . . . "to foster, serve and promote the nation's economic development and technological advancement."

CLEARINGHOUSE
FOR FEDERAL SCIENTIFIC AND TECHNICAL INFORMATION

U.S. DEPARTMENT OF COMMERCE/National Bureau of Standards

This document has been approved for public release and sale.

TurboPascal and Maya: The inspirations behind UnrealEd

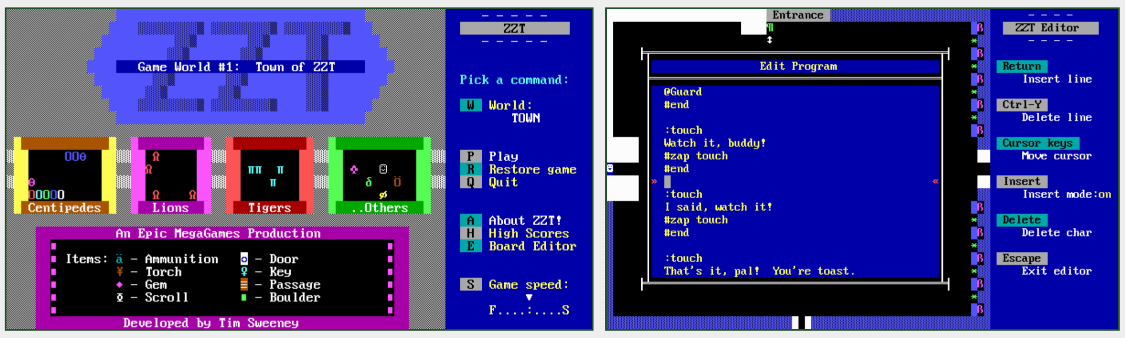
DL: What were some of the sources of inspiration for the design philosophy of the Unreal Editor?

TS: There were really several sources of inspiration.

If you look at ZZT from 1991, you'll see the core feature set of the Unreal Engine. It's basically a game engine with a game implanted in it. The engine exposes a little scripting language, which – while it was very basic – was a full scripting language that you could use to write little game scripts.

[Author's Note: For more about ZZT, check out the book by Anna Anthropy, published by Boss Fight Books]

It had a real time what-you-see-is-what-you-get interactive editor for creating levels. You could go back and forth with a few keystrokes, adding a level, and play-testing it, and iterating on it. That interactive workflow was really the key to it.



```
@Guard
#end

:touch
Watch it, buddy!
#zap touch
#end

:touch
I said, watch it!
#zap touch
#end

:touch
That's it, pal!  You're toast.
```

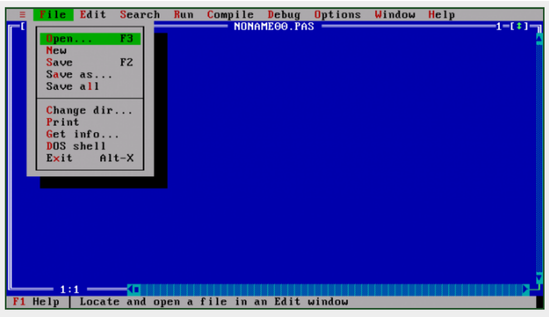
ZZT, in game mode and editor mode

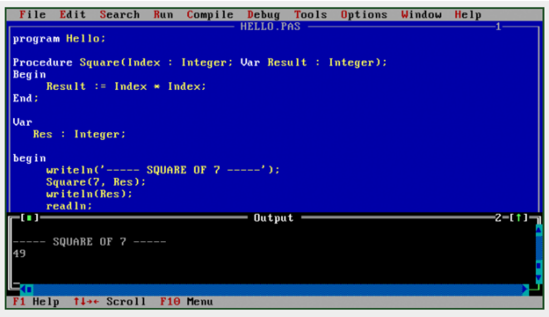
That was largely inspired by Turbo Pascal, which was the first toolset I used that I really loved. It was such an easy-to-use editor for creating code and compiling stuff. You would be typing code, and then several seconds later it would be compiled, and you'd be running it. The iterative process of creating stuff was just awesome, compared to everything I had used up until that point. If you look at ZZT's implementation, it really was like the text version of the Unreal Engine. The entire model for Unreal was driven by it.

There was one other really big inspiration that led to a lot of the design elements of the underlying engine, and that was Visual Basic, which was like Microsoft's clone of Delphi – the visual user interface editing version of Object Pascal for Windows that Borland had created – but I never used Delphi, I only used Visual Basic.

"That was largely inspired by Turbo Pascal, which was the first toolset I used that I really loved"

The idea was that you had this form editor, you'd draw some form elements and boxes and things like that, and then you'd click on it and bring up the code. The code would immediately be there and you just type it out and just go from there.


File Edit Search Run Compile Debug Options Window Help
NONAME00.PAS
Open... F3
New
Save F2
Save as...
Save all
Change dir...
Print
Get info...
DOS shell
Exit Alt-X
F1 Help | Locate and open a file in an Edit window


File Edit Search Run Compile Debug Tools Options Window Help
HELLO.PAS
Output
F1 Help ↑↓→← Scroll F10 Menu

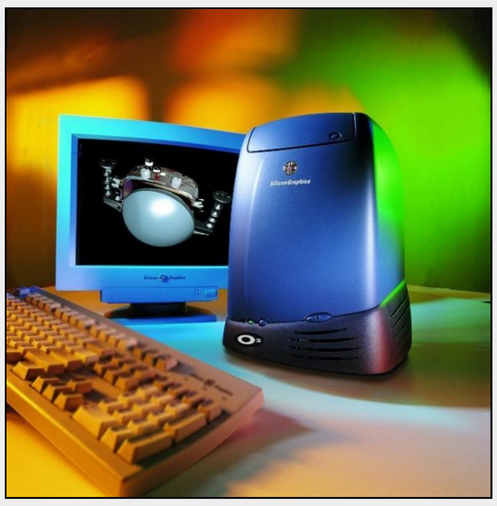
```
program Hello;

Procedure Square(Index : Integer; Var Result : Integer);
Begin
    Result := Index * Index;
End;

Var
   Res : Integer;

begin
    writeln('----- SQUARE OF 7 -----');
    Square(7, Res);
    writeln(Res);
    readln;
```

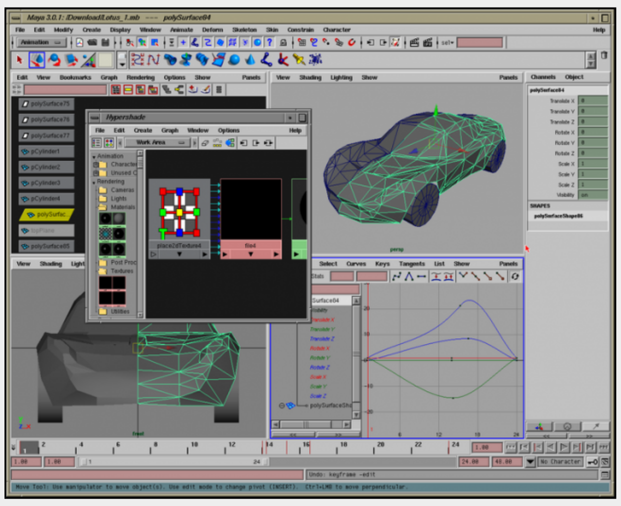
```
----- SQUARE OF 7 -----
49
```

TurboPascal, from Borland

I carried all those principles over to Unreal: you can drop an object in the level, double click on it and there would be the script editor, and you could go type some script down and change it. It would be just a few mouse clicks to be able to write code, create 3-D objects, and do everything all interactively in real-time.

The other thing that was really tied into the development of Unreal is that Epic bought a few Silicon Graphics workstations, and we had the first version of Maya up and running on them. Maya was a total piece-of-shit back then, but the one thing that was really cool was it had this interactive 3D mode where you had this blue, red, and black background space with these object outlines and wireframes that were completely in real time, which none of the other programs running and PC had really achieved at that point in time; they were still so lost in legacy code and legacy UI patterns.





So, the very first thing I did when I started building the Unreal Editor was creating a black background with the blue / red on it – copying Maya – and I wrote a line drawing routine and had 3D wireframe outlines for all the objects going around. That was the initial inspiration to prove that it was possible.

Visual Basic to Slate: The evolution of the UnrealEd interface

[Author's Note: At the beginning of the interview, I had set up a version of UnrealEd 1 running in a virtual machine on my laptop. I hand Tim the mouse so he can use it.]

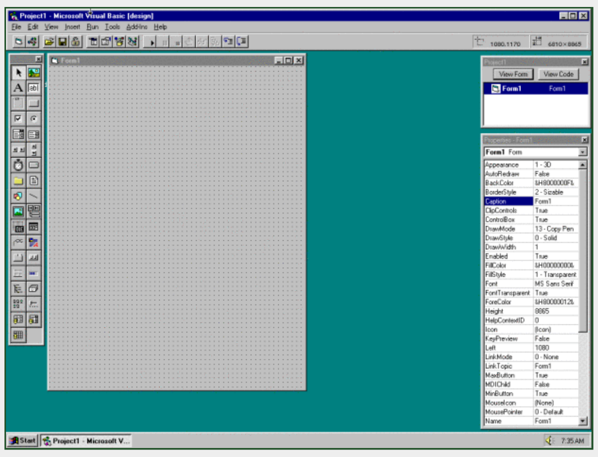
TS: Hey, neat, you really have it running here!

DL: Yes! So, I imagine that the version we're seeing here is all Visual Basic.

TS: Yeah, yeah!

[Author's Note: Tim is having a great time building a level from scratch. It's great to see, almost like seeing two old friends re-united after a long time apart]

DL: What led you to use Visual Basic for the interface of the first version of UnrealEd, and what were some of the other options that were out there?

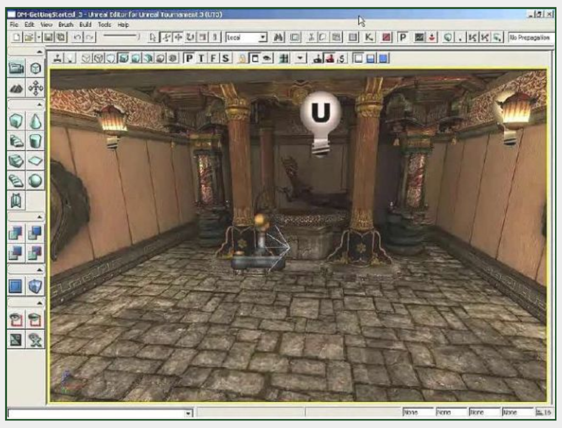


Alan Cooper and Visual Basic

TS: So, this was 1995. At that point in time, the state of the art in C++ user interface frameworks was just abysmal. There was Microsoft Foundation Classes which was the most miserable piece-of-shit API you can imagine. You'd start drawing some controls on to a window and it would be generating massive amounts of C++ code that had a bunch of comments saying "oh, here we're creating this control for you!" Then you'd move an object around and it would update part of the code but not other parts, and it would just get continually broken all the time, and so I said "I'm not touching this."

Visual Basic had this awesome user interface designer where you laid out all the controls and menu items and bits of UI in a really productive way. It was more productive than any like UI toolkit I had seen, mostly because it was just like this one really clean holistic program: you draw UI, you click on it, you add basic code of how it will interact. I found that it was much easier to build a UI that way, and then to interface it to C++ through this command line interface, sending text back and forth, as a way of marshaling data. I think that remained true for about a decade and was really only in the early 2000s that C++ UI toolkits got into a decent state, with Qt and others.

[Author's Note: The first version of Visual Basic was designed by Alan Cooper, who is often referred to as the "Father of Visual Basic". He is also one of the most important figures in the field of usability and user experience]



UnrealEd 3, which had elements of wxWidgets

DL: And then, over time, I guess there are pieces that started to be pulled out and replaced with other parts. How did that evolution go?

TS: After finishing Unreal Editor 1, I went off and started doing a whole bunch of next generation research that generally didn't reach fruition, while Warren Marshall jumped in and rewrote the Visual Basic portions of the Unreal Editor in C++ using wxWidgets which, at the time, was the best thing available. That was the basis of the UI framework in Unreal Editor 2.

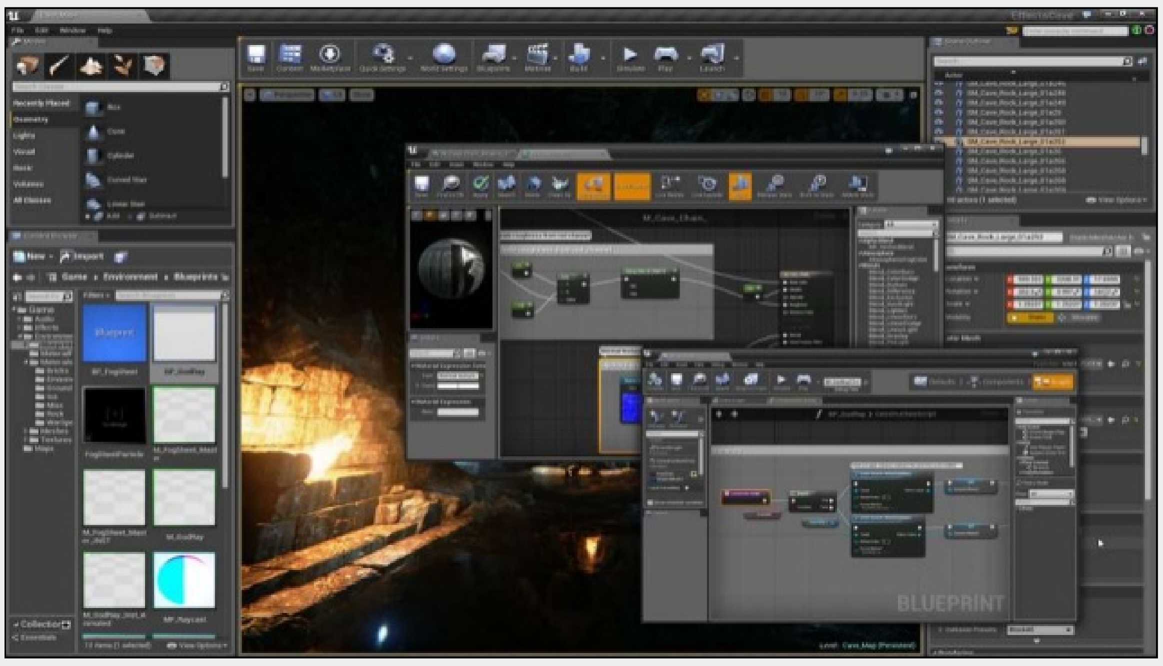
"By the time we got to the very beginning of the Unreal Engine 4 development cycle, we had five different UI toolkits"

By the middle of the Unreal Engine 2 generation, Visual Basic was totally gone from the process. There was this nicer, cleaner C++ framework there. That achieved basically the same UI but without the language challenges. The real problem was, over time, wxWidgets didn't improve and other UI toolkits came out, and so we kept integrating new ones for special tools. So, by the time we got to the very beginning of the Unreal Engine 4 development cycle, we had five different UI toolkits...

DL: That's not uncommon...

TS: ...including some crazy WPF stuff written in C#, and integrated in Unreal Engine 4 that just didn't work on Mac, for example. So there was this a giant mess we had at that point.

At the same time, Nick Atamas was prototyping a new UI layer in C++ and we eventually decided to adopt that, and that became Slate. So we re-wrote 100 percent of the Unreal Engine user interface, got rid of all of these plug-in UI toolkits, and re-wrote it in a unified way. So, that gave us the ability to scale up to what we have today.



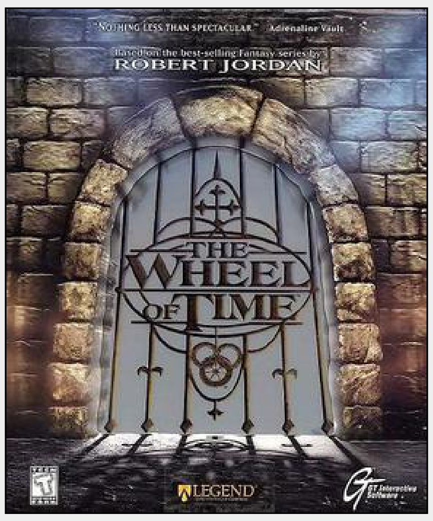
Screenshot of UnrealEd with the Slate UI

But still, nobody's really achieved the sweetness that existed back then with Visual Basic. Even the C# User Interface framework is just a massive complicated piles of XML and other craziness that's unnecessary. It feels like every generation that tries to do it does it in a more elaborate way and it gets worse.

Screenshots and XCopy: The importance of licensing

DL: Which companies were the first to use the Unreal Editor?

TS: Very early on – two years before we shipped – we already had licensees: Microprose was using Unreal Engine, and then Legend Entertainment was using it for Wheel of Time, and we were providing them with support.


"NOTHING LESS THAN SPECTACULAR" - Adrenaline Vault
Based on the best-selling Fantasy series by
ROBERT JORDAN
THE WHEEL OF TIME
LEGEND
GT Interactive Software

Wheel of Time, by Legend Entertainment

DL: They helped you out as well, right? That was sort of part of that relationship, working together.

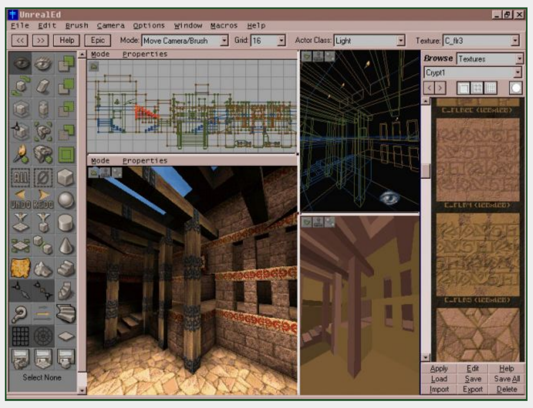
TS: Yeah, their license fees kept Epic afloat throughout the development of Unreal.

We really took licensing seriously at that point because it was paying the bills, and it led us to look at engine licensing quite differently than Id did. The joke -- at the time -- was that licensing from Id was like a quarter million dollar XCopy: you'd pay a quarter million dollars, and they'd use the DOS XCopy command to give you a copy of the source... and that would be it. [laughs]

"The joke – at the time – was that licensing from Id was like a quarter million dollar XCopy"

DL: [laughs] OK, so what led to Microprose and Legend Entertainment licensing Unreal Engine even before Unreal 1 was released?

TS: I think it was probably the fact that we released amazing screenshots of our game, but also screenshots of our editor early on, around 1995. That led those companies to call us. Microprose called us and they said "we're interested in licensing your engine!" and we're like "Engine? What engine? Oh! Right, our engine! It's very expensive." [laughs] And literally that was the conversation.

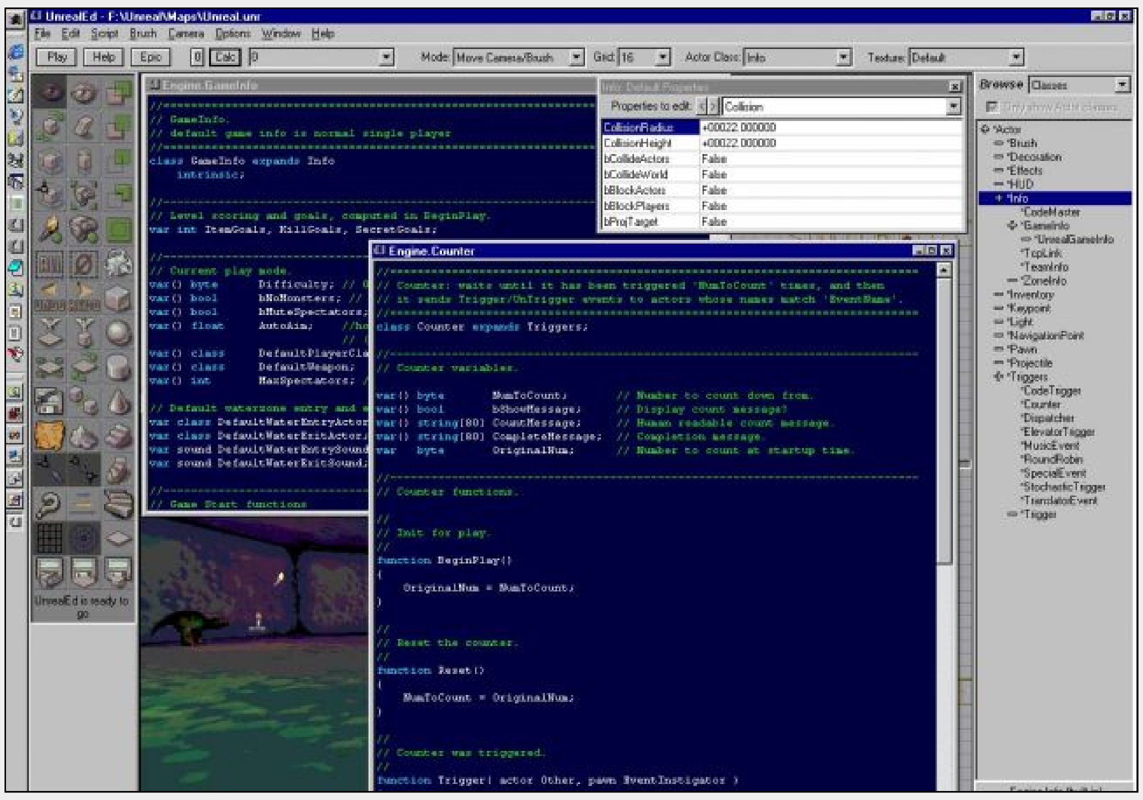


Two of the earliest screenshots of the Unreal Editor and Unreal

Dinosaurs and Lizards: The terminology and iconography of UnrealEd

DL: So, speaking of screenshots, here is one that was posted to Blue's News in late 90s. There are some subtle differences to the virtual machine instance that we have running here: for example, there are Play, Help, and Epic buttons in the top left corner, which were not in the final version.

Can you talk a bit about this screenshot?


UnrealEd - F:\Unreal\Maps\Unreal.unr
File Edit Script Brush Camera Options Window Help
Play Help Epic Calc
Mode: Move Camera/Brush Grid: 16 Actor Class: Info Texture: Default
Browse Classes
UnrealEd is ready to go

```
Engine.GameInfo
//=============================================================
// GameInfo.
// default game info is normal single player
//=============================================================
class GameInfo expands Info
    intrinsic;

//-------------------------------------------------------------
// Level scoring and goals, computed in BeginPlay.
var int ItemGoals, KillGoals, SecretGoals;

//-------------------------------------------------------------
// Current play mode.
var() byte      Difficulty; //
var() bool      bNoMonsters; //
var() bool      bMuteSpectators;
var() float     AutoAim;    //
                            //
var() class     DefaultPlayerCla
var() class     DefaultWeapon;
var() int       MaxSpectators; /

// Default waterzone entry and e
var class DefaultWaterEntryActor
var class DefaultWaterExitActor;
var sound DefaultWaterEntrySound
var sound DefaultWaterExitSound;

//-------------------------------------------------------------
// Game Start functions
```

```
Engine.Counter
//=============================================================================
// Counter: waits until it has been triggered 'NumToCount' times, and then
// it sends Trigger/UnTrigger events to actors whose names match 'EventName'.
//=============================================================================
class Counter expands Triggers;

//-----------------------------------------------------------------------------
// Counter variables.

var() byte        NumToCount;       // Number to count down from.
var() bool        bShowMessage;     // Display count message?
var() string[80]  CountMessage;     // Human readable count message.
var() string[80]  CompleteMessage;  // Completion message.
var   byte        OriginalNum;      // Number to count at startup time.

//-----------------------------------------------------------------------------
// Counter functions.

//
// Init for play.
//
function BeginPlay()
{
    OriginalNum = NumToCount;
}

//
// Reset the counter.
//
function Reset()
{
    NumToCount = OriginalNum;
}

//
// Counter was triggered.
//
function Trigger( actor Other, pawn EventInstigator )
```

Properties to edit: Collision

CollisionRadius	+00022.000000
CollisionHeight	+00022.000000
bCollideActors	False
bCollideWorld	False
bBlockActors	False
bBlockPlayers	False
bProjTarget	False

```
*Actor
  *Brush
  *Decoration
  *Effects
  *HUD
  *Info
    *CodeMaster
    *GameInfo
      *UnrealGameInfo
    *TcpLink
    *TeamInfo
    *ZoneInfo
  *Inventory
  *Keypoint
  *Light
  *NavigationPoint
  *Pawn
  *Projectile
  *Triggers
    *CodeTrigger
    *Counter
    *Dispatcher
    *ElevatorTrigger
    *MusicEvent
    *RoundRobin
    *SpecialEvent
    *StochasticTrigger
    *TranslatorEvent
    *Trigger
```

Screenshot of UnrealEd posted on Blues News in 1998

TS: That's definitely Unreal Engine 1, probably around 1998, because it has a bunch of Steve Polge's code that was used for coordinating multiplayer early on.

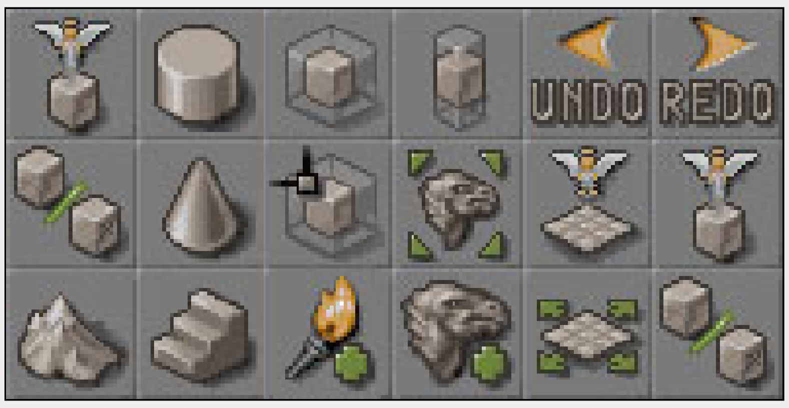
The "Epic" button was just a web page thing which everybody found annoying because they'd click on it accidentally and say "Ahhh, my web browser is popping up!" [laughs]

DL: [laughs] OK, can you tell me a bit about the iconography?

TS: I had drawn some really, really horrible icons for all of these things, and then I sent them to Dan Cook, the artist who did the artwork on Tyrian, our early shooter game.

He was supposed to draw an icon for a pawn, so he created a chess piece. I was like "no no no", and he said "OK, so tell me what a pawn is". So I said "It's like a monster, like really bad-ass thing" so he drew the head, which nobody is really sure what it is, some people think it's a dinosaur, a lizard, or something... but those have been our icons for a decade.

"Nobody is really sure what it is, some people think it's a dinosaur, a lizard, or something... but those have been our icons for a decade."



Daniel Cook and the icons he created for UnrealEd. The "Add Pawn" icon is in the bottom row, third from right.

DL: We spoke about the word “pawn” before. Was that something that you had come up with, or had you seen that somewhere else?

TS: I think that was Steve Polge's or James Schmaltz's.

DL: What about “actors”?

TS: Carmack had come up with some terminology calling things "entities", and I was like "No, we need some more flair."

DL: [laughs]

TS: We decided that we’d have something called "actors", because there are these actor-based programming laws defined in the 1980s in SmallTalk, which I'd really been enamored with at the time. It was all object-oriented. That seems like an appropriate start. So we came up with the idea of pawns, as well as instigators for defining who started a series of events, who gets credit for a kill, and all the other terminology.

Schmaltz-isms and the Voodoo Brain Virus: Creating UnrealScript

DL: Tell me more about how James and Steve were involved in the use and design of UnrealScript.

TS: James Schmaltz is like a super-brilliant jack of all trades. He was the best artist on the team, an amazing level designer, and could also program in UnrealScript as well as assembly.

DL: In the credits for Unreal, his name appears in a couple of different categories.

"He'd have this multi-line expression that's like: 'something dot instigator dot blah blah blah' and I'd replace it with something like... 'self'"

TS: There are very, very few people with that level of talent in the whole games industry, and he deserves all the success he has.

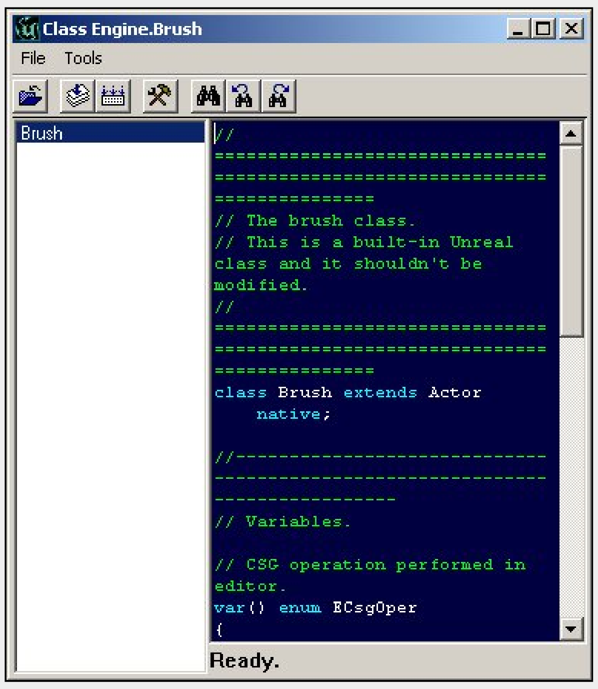
But he went from assembly language to UnrealScript, and he was writing this insane UnrealScript code where he would just type stuff until it worked, and so I'd go in at night and simplify his code. He'd have this multi-line expression that's like: "something dot instigator dot blah blah blah" and I'd replace it with something like… "self". [laughs]

DL: [laughs]

TS: We'd call those Schmaltz-isms. And then Polge would have these magic numbers in code, like "walk speed = run speed x 3.072". So I'd ask "is 3.072 some constant in physics I'm not aware of?" [laughs]

DL: [laughs]

```
pushl  %ebp
movl   %esp,%ebp
subl   $0x4,%esp
movl   $0x0,0xfffffffc(%ebp)
cmpl   $0x63,0xfffffffc(%ebp)
jle    08048930
jmp    08048948
nop
nop
nop
movl   0xfffffffc(%ebp),%eax
pushl  %eax
pushl  $0x8049418
call   080487c0 <printf>
addl   $0x8,%esp
incl   0xfffffffc(%ebp)
jmp    08048925
nop
nop
xorl   %eax,%eax
jmp    0804894c
leave
ret
```



```
//
==================================
==================================
================
// The brush class.
// This is a built-in Unreal
class and it shouldn't be
modified.
//
==================================
==================================
================
class Brush extends Actor
    native;

//--------------------------------
----------------------------------
------------------
// Variables.

// CSG operation performed in
editor.
var() enum ECsgOper
{
```

TS: UnrealScript itself was inspired by Java, at the time, which was seen as a successor to C++. They copied a lot of the design decisions and then they added a lot of new concepts on top it. There are the beginnings of some generic containers in UnrealScript that didn't exist in Java until several generations later.

I always thought when the C# language was designed, they must have really looked over UnrealScript, because I saw a few features from UnrealScript that surfaced in C#. It was always my hope that they had borrowed some of these ideas.

But, the more I dug into object oriented programming and really dug into SmallTalk and the state of the art research with metaclasses, I came to the realization that it was this kind of voodoo brain virus that didn't have any real theoretical underpinnings. Whereas, when you look into Haskell and Curry Howard iso-morphism, and other really advanced principles in programming, you see that there is a source and structure that should inspire us.

SoftDisk, Id, and million-dollar cheques:

DL: With their business focus and shareware history, did Jay Wilbur and Mark Rein have an impact on the engine, the tools, or the editor, or the resources that were put towards them?

TS: In the early days, Epic really worked because of the combination of me on the technical side and Mark on the business side. Mark would be going around the world and doing crazy business deals that get the cash rolling in. It was absolutely critical if we didn't have him, we would never survived those early days.

Mark Rein and Jay Wilbur

At one point, we were out of money, and what we were spending was completely financed by Mark's American Express card, which would eventually get taken away from him.

DL: [laughs]

TS: And so he flew off to meet with TG Interactive, and came back with a million dollar cheque. That saved us. That was repeated several times in our history. It's so important to have really great business guys out negotiating. He was the first president of Id, and -- after Mark -- Jay was the first CEO of Id.

DL: He was at SoftDisk before that, with Carmack, right?

TS: Yeah! It's funny because I had actually submitted my first game ZZT to SoftDisk. It was Jay Wilbur who was doing the deal for SoftDisk. I got $3,000 from SoftDisk as a result of that negotiation, so I knew Jay going way back to very, very early days.

The early days of Id Software. Jay Wilbur is on the extreme right.

The Id guys were so inspirational to work with early on. I went to this goofy shareware conference, and Id showed up. They were the darlings of the industry back then because they put out Wolfenstein 3D and it had been by far the biggest success in the history of shareware. They actually hung out with us and took us out to dinner, they were spending time with us. It was so cool to see that the superstars of the industry were just down-to-earth guys. John Romero is just the sweetest game developer ever.

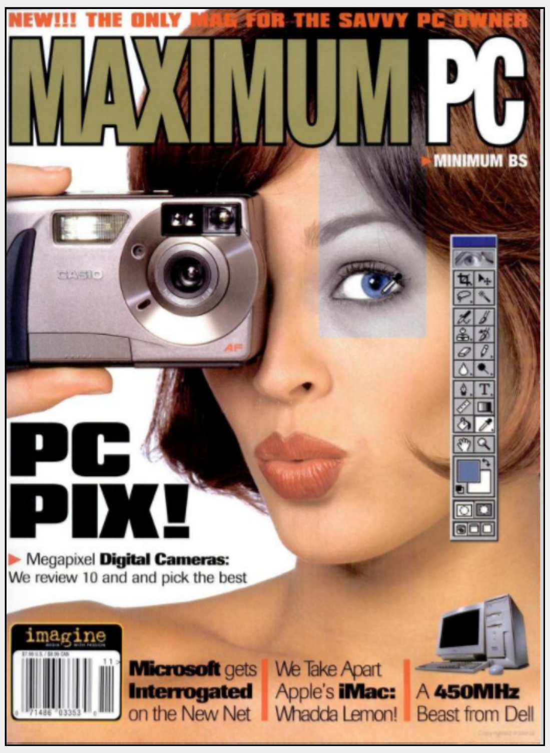
[Author's Note: Agreed. John Romero was very generous with his time during our TEd interview.]

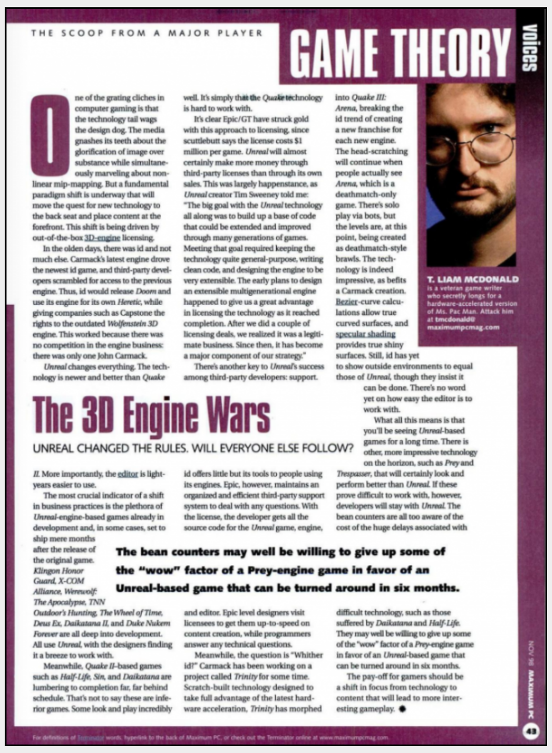
WYSIWYG and Ease-of-use: Tools perspective first

DL: So in the November 1998 issue of Maximum PC, there was an interview with you that also talked about some of the various technologies that were available at the time. In the article, they said "the [Unreal Editor] is light-years easier to use", and "Quake technology is hard to work with."

They also wrote "The technology [for Quake III: Arena] is indeed impressive", and "Prey and Trespasser look and perform better than Unreal. If these prove difficult to work with, however, developers will stay with Unreal."

So, was the goal from the start to make ease-of-use tool as a competitive advantage?


NEW!!! THE ONLY MAG FOR THE SAVVY PC OWNER
MAXIMUM PC
MINIMUM BS
PC PIX!
Megapixel Digital Cameras: We review 10 and and pick the best
imagine
Microsoft gets Interrogated on the New Net
We Take Apart Apple's iMac: Whadda Lemon!
A 450MHz Beast from Dell


THE SCOOP FROM A MAJOR PLAYER

GAME THEORY voices

The 3D Engine Wars

UNREAL CHANGED THE RULES. WILL EVERYONE ELSE FOLLOW?

One of the grating cliches in computer gaming is that the technology tail wags the design dog. The media gnashes its teeth about the glorification of image over substance while simultaneously marveling about non-linear mip-mapping. But a fundamental paradigm shift is underway that will move the quest for new technology to the back seat and place content at the forefront. This shift is being driven by out-of-the-box 3D-engine licensing.

In the olden days, there was id and not much else. Carmack's latest engine drove the newest id game, and third-party developers scrambled for access to the previous engine. Thus, id would release *Doom* and use its engine for its own *Heretic*, while giving companies such as Capstone the rights to the outdated *Wolfenstein 3D* engine. This worked because there was no competition in the engine business: there was only one John Carmack.

Unreal changes everything. The technology is newer and better than *Quake II*. More importantly, the editor is light-years easier to use.

The most crucial indicator of a shift in business practices is the plethora of *Unreal*-engine-based games already in development and, in some cases, set to ship mere months after the release of the original game. *Klingon Honor Guard, X-COM Alliance, Werewolf: The Apocalypse, TNN Outdoor's Hunting, The Wheel of Time, Deus Ex, Daikatana II,* and *Duke Nukem Forever* are all deep into development. All use *Unreal*, with the designers finding it a breeze to work with.

Meanwhile, *Quake II*-based games such as *Half-Life, Sin*, and *Daikatana* are lumbering to completion far, far behind schedule. That's not to say these are inferior games. Some look and play incredibly well. It's simply that the *Quake* technology is hard to work with.

It's clear Epic/GT have struck gold with this approach to licensing, since scuttlebutt says the license costs $1 million per game. *Unreal* will almost certainly make more money through third-party licenses than through its own sales. This was largely happenstance, as *Unreal* creator Tim Sweeney told me: "The big goal with the *Unreal* technology all along was to build up a base of code that could be extended and improved through many generations of games. Meeting that goal required keeping the technology quite general-purpose, writing clean code, and designing the engine to be very extensible. The early plans to design an extensible multigenerational engine happened to give us a great advantage in licensing the technology as it reached completion. After we did a couple of licensing deals, we realized it was a legitimate business. Since then, it has become a major component of our strategy."

There's another key to *Unreal*'s success among third-party developers: support. id offers little but its tools to people using its engines. Epic, however, maintains an organized and efficient third-party support system to deal with any questions. With the license, the developer gets all the source code for the *Unreal* game, engine, and editor. Epic level designers visit licensees to get them up-to-speed on content creation, while programmers answer any technical questions.

Meanwhile, the question is "Whither id?" Carmack has been working on a project called *Trinity* for some time. Scratch-built technology designed to take full advantage of the latest hardware acceleration, *Trinity* has morphed into *Quake III: Arena*, breaking the id trend of creating a new franchise for each new engine. The head-scratching will continue when people actually see *Arena*, which is a deathmatch-only game. There's solo play via bots, but the levels are, at this point, being created as deathmatch-style brawls. The technology is indeed impressive, as befits a Carmack creation. Bezier-curve calculations allow true curved surfaces, and specular shading provides true shiny surfaces. Still, id has yet to show outside environments to equal those of *Unreal*, though they insist it can be done. There's no word yet on how easy the editor is to work with.

What all this means is that you'll be seeing *Unreal*-based games for a long time. There is other, more impressive technology on the horizon, such as *Prey* and *Trespasser*, that will certainly look and perform better than *Unreal*. If these prove difficult to work with, however, developers will stay with *Unreal*. The bean counters are all too aware of the cost of the huge delays associated with difficult technology, such as those suffered by *Daikatana* and *Half-Life*. They may well be willing to give up some of the "wow" factor of a *Prey*-engine game in favor of an *Unreal*-based game that can be turned around in six months.

The pay-off for gamers should be a shift in focus from technology to content that will lead to more interesting gameplay. ◆

The bean counters may well be willing to give up some of the "wow" factor of a Prey-engine game in favor of an Unreal-based game that can be turned around in six months.

T. LIAM MCDONALD is a veteran game writer who secretly longs for a hardware-accelerated version of Ms. Pac Man. Attack him at tmcdonald@maximumpcmag.com

For definitions of Terminator words, hyperlink to the back of Maximum PC, or check out the Terminator online at www.maximumpcmag.com.

NOV 98 MAXIMUM PC 43

Maximum PC, November 1998

TS: Yeah, absolutely. You know, that was always my focus: to build an editor enabling level designers to build awesome stuff. From the beginning, it clear that making everything work in real-time and do super-fast iteration was the key to that. To be completely "what-you-see-is-what-you-get." Then, you're limited only by your ability to think of new ideas. We've always had a big focus on tools at Epic.

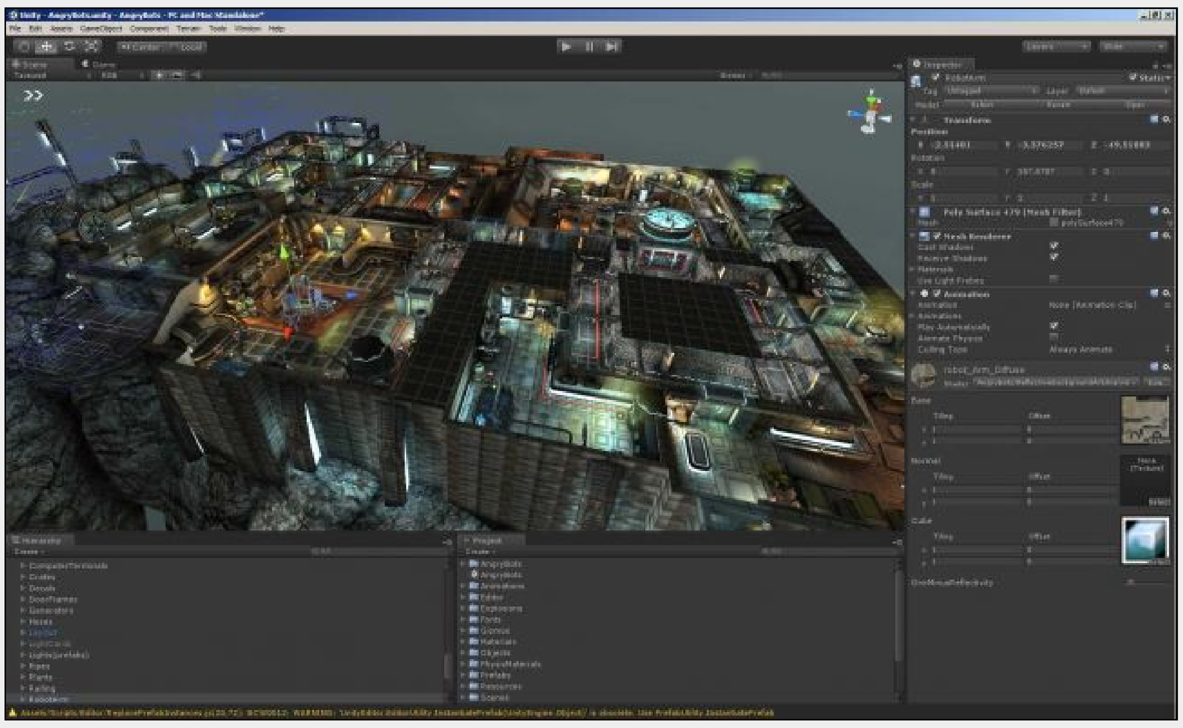
DL: What's the process that Epic uses to ensure that there's more attention and time and care put into making the tools easier to use?

TS: Development at Epic is a combination of the engine team, building systems, features, and tools, and the game teams consuming them while building games. There's this iterative process where the engine teams is out creating new ideas, and then sharing them with the game teams and getting constant feedback on what works and what doesn't work. That's really what has shaped our process: the fact that we have to satisfy our own internal game developers keeps the tools team honest.

"You can't treat ease-of-use as a stand-alone concept"

Not only do we want to build tools that are easy to use, but also ensure that they're making the right tradeoffs, and that you can actually produce the content that you need in the ship the game.

That's the flip-side of ease-of-use. You can't treat ease-of-use as a stand-alone concept. It's no good if the tools are super easy to get started with, so you can start building a game, but they're super hard to finish the game with, because they impose workflow burdens or limited functionality.



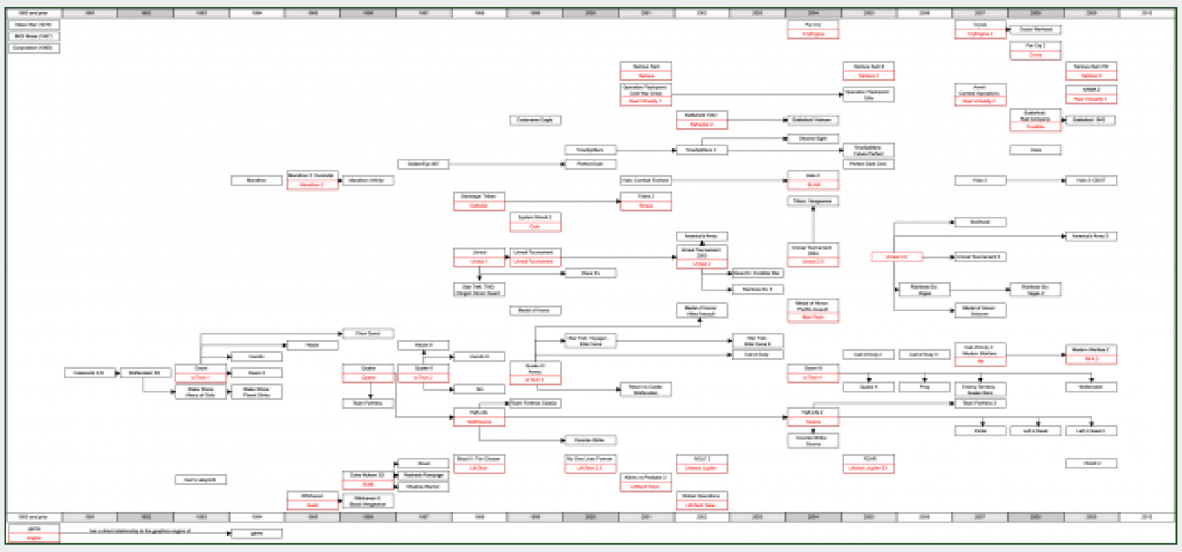
If you look at the last five or six years of engines, the competition between Epic and Unity has been defined by initial ease-of-use, where Unity has the advantage. Whereas over the life cycle of actually shipping a game, for performance, on multiple platforms, Unreal has had the advantage. That's because everything we do is aimed at being able to ship epic-scale games, which are the kind of games we build. That's actually quite a bit trickier than making it easy for a team of three people to create something quickly.

Now the Unreal Engine is about 20 times the code size of Unreal Engine 1. The tools are maybe 10 times the complexity, and necessarily so. People come into Unreal and they say "oh my god" because there are a lot of menu options. Then they go with Unity, and they say "oh so nice and easy." And they get to the point where they're trying to ship their product, and they go "oh shit, now we need to license source code, so we can add some features to the engine because they're not in the menus!" So, you know, that's the dichotomy.

Inspiration and Legacy: The impact of UnrealEd

DL: UnrealEd inspired some game developers like myself to not only start making games, but to build their own tools. What kind of impact do you think that UnrealEd has had on the industry?

TS: I think every game editor you look at nowadays draws some inspiration from UnrealEd. Because it was one of the earliest editors, we got a number of fundamental decisions right, like the way you deal with 2D grids, object placement, and navigation through the world. I think you can really trace the lineage that goes from the original Doom editor through to the Quake editor through Unreal. Now everything you look at is -- to some extent -- based on that.



A diagram showing the history of FPS engines, from Wikipedia. Click for larger verison.

Some of those are because of general influences like Maya that affected all of them, but some are really quite specifically Unreal, in the way that people structure class hierarchies, implement undo systems, or all of these other thorny problems of game development. Anybody who came into the industry in the early 2000s typically came through either the Unreal or the Quake path. Even though Quake was a much bigger game, I think more designers came in through UnrealEd because the tools were so gratifying.

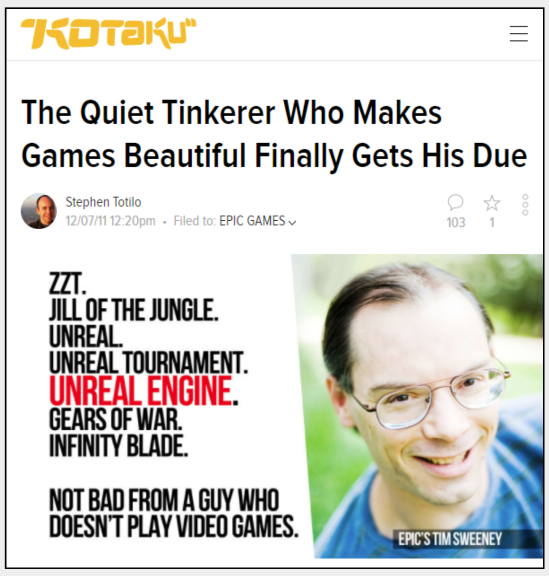
Multiplying and Dividing Productivity: Advice for game developers

DL: There is an interview that you did with Kotaku in 2011. I'm going to read a few quotes from you that I feel are related:

"We always approach game development from the tools perspective first. We build the tools we need, we build a user friendly set of tools and carry them forward."

"We think long term at Epic. We're like Intel. We look at what we're going to do in five to ten years from now, and set our direction that way, whereas most companies are like we need to ship our next game. we'll put all of our resources on this and then we'll worry about the next one."

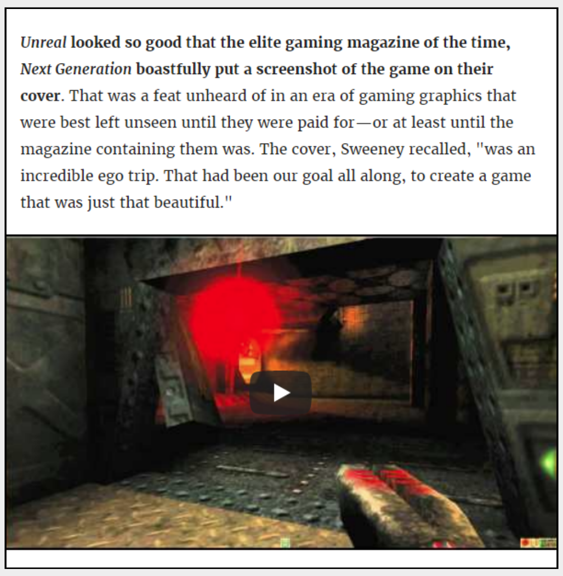
"Big game companies like EA or Activision don't make the tools investment, they don't do the sort of long term thinking we do and realize that we have to make game development processes as efficient as possible."



KOTAKU

The Quiet Tinkerer Who Makes Games Beautiful Finally Gets His Due

Stephen Totilo
12/07/11 12:20pm · Filed to: EPIC GAMES



***Unreal* looked so good that the elite gaming magazine of the time, *Next Generation* boastfully put a screenshot of the game on their cover**. That was a feat unheard of in an era of gaming graphics that were best left unseen until they were paid for—or at least until the magazine containing them was. The cover, Sweeney recalled, "was an incredible ego trip. That had been our goal all along, to create a game that was just that beautiful."

In my interview with John Romero, he summed up this all up very nicely, when he said: "Tools live longer than games do".

What advice do you have for game developers to avoid this mistake, so they can understand the long-term value of investing in tools?

TS: Well... don't "kind-of" build an engine. Either build an engine, or don't... right? So many companies are now crippling their production process by building engines that are perfectly fine with tools that are not fine at all. It's always the tools that kill people.

"So many companies are now crippling their production process by building engines that are perfectly fine with tools that are not fine at all. It's always the tools that kill people."

Look at all these internal engines... for example, Frostbite has more advanced rendering features than we have and they're producing prettier pixels than we are in a lot of cases, but Unreal developers can produce content much more productively, maybe 30 to 50 percent more productively, which means that a team half the size can produce a game that's just as good. They can iterate on it more, and polish it more than they could with these less polished toolsets. So, everybody should really make a conscientious decision to either fully invest in producing awesome tools for internal use, or not.

"I think every company needs to make the decision to either invest significantly more in tools, or just not bother."

DL: Because you feel that going half-way hurts developers.

TS: Yes. There must be some really fucking stupid accountants at the core of these companies, saying "Oh, wow, limiting tools development can save two percent of our budget." Then they end up increasing the budget by 50 percent, because they have to spend significantly more time and manpower to create their game. So, it creates this crazy false economy.

I think every company needs to make the decision to either invest significantly more in tools, or just not bother.

That applies to everything. It's not only for the 3d editor used to build your levels, but it's your build system, and it's your programming language, it's your production pipeline, it's the DCC tools you use, all of that.

Tools are supposed to have a multiplying impact on productivity, and when you find that they have a dividing impact on productivity, get the hell out.

DL: Perfect. Thank you for taking the time to talk with me.

If you're interested in game engines and tools, don't miss the Tools Tutorial Day at the Game Developers Conference in San Francisco!

This full-day gathering of tools developers will feature speakers from EA, Media Molecule, Infinity Ward, Ubisoft Montreal, Unity, Guerilla Games, Massive, Quel Solaar, Rebellion, Remedy, Blizzard, and Bungie.

Don't miss the inaugural Tools Tutorial Day, on Monday, March 19th, 2018.

Click here to find out more

Related Jobs

Phosphor Games Studio — Chicago, Illinois, United States
[01.08.18]
Mid to Senior Gameplay Programmer

Blackstorm — Mountain View, California, United States
[01.05.18]
Studio Game Engineer

Naughty Dog — Santa Monica, California, United States
[01.04.18]
Web Developer

Naughty Dog — Santa Monica, California, United States
[01.04.18]
IT Help Desk Generalist (Temporary Assignment)

[**View All Jobs**]