Creation of A Content Creation Pipeline

Bachelor thesis by Oskar Holmstrand

v1.0 2005-06-06

1. Abstract

This document describes the process of identifying and improving a production pipeline for the upcoming game/learning tool *Foreign Ground*.

Foreign ground is a first person game where the player is exposed to different situations typical for the Swedish international peace enforcing squads operating around the world, in this case Liberia.
Foreign ground is being developed by *Dataductus AB, Skellefteå, Sweden*, in cooperation with the Swedish military school, and LTU.

2. Foreword / intro

The reduced development time in todays game-projects have made the need for externally produced engines a reality. As a result of this, the production pipeline is defined in many ways, causing extra overhead in terms of learning and adapting. The selection of an optimal engine for the specific game is an often overlooked process, possibly resulting in production environments with low turnout.

By carefully analyzing and improving the internal use of the pipeline, efficiency can be increased – leaving more time for creativity rather than technical issues.

This thesis describes generic and specific processes used to pre-define the pipeline for Foreign Ground at Dataductus. Working on the project was me and a programmer, Fredrik Mäkeläinen. While working in close cooperation with eachother, we focused on different parts.

In this thesis, I refer to the Swedish military school as the customer.

3. Abbrevations/Words

LTU = Luleå University of technology.

WYSIWYG = What you see is what you get. Which in this case means that what you see in the editor is what the content will look like in-game.

BF1942 = Battlefield 1942. (see Appendix B)

UT2004 = Unreal Tournament 2004. (see Appendix B). Uses Unreal Engine 2.0.

UE2 = Unreal Engine 2.0.

HL2 = Halflife 2. (see Appendix B)

UDN= Unreal Developer Network

4. Index

Table of Contents

Appendix C, D, E are provided at the end of the document.

5. Process

My co-worker, Fredrik Mäkeläinen, and I decided to divide our work into three separate parts. The first step was to identify the potential engine candidates, and doing some basic tests of them. In the second phase, we would analyze the two most promising more throughly, leading to the choice of only one. The third and last part of our work was to improve and learn the pipeline in a way that would make working with it as easy and effective as possible.

Fredrik Mäkeläinen is a programmer, working parallell with me outlining a programmer workflow. This gave me the oppotunity to focus 100% on the graphics artist workflow.

5.1. Part 1 – Identify potential candidates

5.1.1. Pro's and con's

There are some important aspects to consider when identifying potential engine candidates for a specific project. First all the project specific variables(appendix A), and secondly all general workflow aspects should be included. If an engine is great at doing something, but the process of doing it is too cumbersome, an approach where the capabillities are less but the process is easy may be preferable.

Finally, the pro's and con's must always be weighed, together with the specific preferences of the involved team-members, to be able to make a decision which is as optimal as possible. This is a decision not to be made at a whim, since it is generally impossible to change it in hindsight.

5.1.2. The first list

During an initial meeting, we brainstormed about which engines that may be usable (appendix B). This resulted in a somewhat too large list for us to handle internally, so we also decided which engines should be delegated to LTU. It takes a considerable amount of time to understand the basics of a specific engine, so although the important variables may be easilly analyzed once accessed, the initial access does take some time. All in all, the method of access is something that may or may not differ between the engines, so some cross-knowledge is sometimes achieved.

5.1.3. Generic research

The biggest setback during this initial phase was that we had no document outlining the requirements from the customer. Since the requirements heavilly depended on the setting of the game(scenario), which wasn't decided at this point, it was more difficult to know how to judge the engines. This wasn't all in all bad though, since it also gave us a chance to steer the scenario selection from the technical perspective.

It should be considered the prime objective, to find an engine that with the most ease allows the production team to fulfill the requirements.

5.1.4. Our piece

The engines that fell on Fredriks and my table were HL2:source, UT2004(UE2) and Doom3. Farcry was also researched inhouse, but not mainly by us. At first, modding any of the engines seem very confusing, but by searching into the mod-community, a general principle can often be established without too much hassle. After getting the initial overview, I constructed workflow diagrams lining out the specific processes (Fig. 1, page 7). This was good in two ways – it made me see more clear how things correlated, and it made the information more digestible for others.

We started by constructing a template, outlining the captions and the contents of HL2:source. This could be used as a guide by ourselves and the others constructing similar documents. Having a common design on the documents is very important when later trying to compare them against each other.

The analysis resulted in three documents, each describing one respective game. To ease the overview of the documentation, we decided to keep the information brief; limiting ourself to technical keywords.

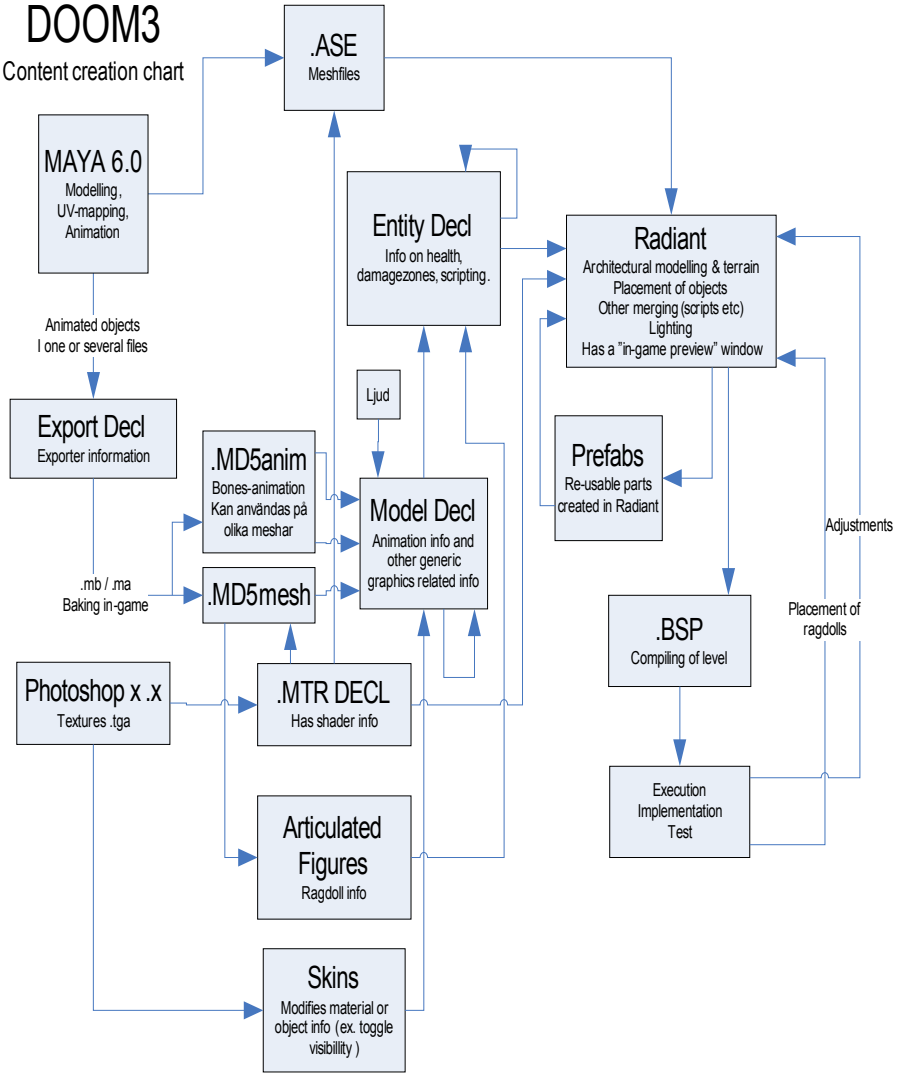

Fig. 1 Example Workflow, Doom3

5.1.5. The candidates

Phase one was concluded in a one day seminar, where we showcased the different engines to share the knowledge we had obtained. Our goal was to select two of them; two candidates that we would look into more deeply. One could clearly see that the engines not only were different, but were also in totally different steps in evolution.

The selection of two candidates was therefore an easy one:

- FarCry because of it's superior editor and visual abillities.
- UE2 because of it's very refined editor, low level control and massive actual community usage.

5.2. Part 2 – In-depth analysis

During the seminar mentioned above, a few glitches in the knowledge obtained about the selected engines was noticed. For FarCry, knowledge of it's ”PolyBump” technology in relation to regular tangent-space normalmaps was a specific area. Additionally, both Engines had to be tested in-depth when it came to character export and animation. This is something advanced that most regular modders don't do, so community knowledge of it is often sparse and flawed. The whole process had to function, because there was no mistaking that at one point or the other, the project members were going to have to do it all.

5.2.1. AI test

Meanwhile, the programmers were going to do a hands-on test, concering group movement in zig-zag formation that was actually going to be used in the game. The two main programmers would test one engine each. This would achieve a more practical approach to analysis that could root out how difficult it actually is to actually do something. It should be pointed out that it's not rare for a system to seem very easy and automatic on the surface, but actually be very complex or even impossible to work with. The lead star here is the word control. If there is a specific area we cannot control, then we must be sure that it meets all our requirements.

This was actually the case for FarCry. On the surface, it seemed extremely advanced (and it is), but the level of control handed out to consumers did not, after deeper investigation, seem to be sufficient for our task. We were going to modify the the game in a very specific way, so an almost wasn't good enough.

Lack of good documentation (or access to good documentation) for CryEngine kept the testing from progress that far. Good documentation is a very important thing. If you can't find out how to do something, you're never going to be able to do it. It doesn't matter how advanced the technology is, if it can't be accessed.

It should for the record be noted that the two programmers had totally different amounts of time to spend on their attempts, weighing in favor for UT2004. However, at this point, there was really no time to let question marks remain. We had to come to a decision.

5.2.2. Pro's n' Con's

The final decision was made at a meeting. Andreas(lead programmer) had put up a list of all the different areas of importance to us. We then went through the list, stating which of the two engines did it best. It was very often that we wanted to say ”well, both are good, neither is really better”, but a decision had to be made, so we had to decide. The result was a list of Yes's and No's.

On almost all areas, there was a discussion as to which and why and how each engine performed better or worse. It was not easy to choose.

We then weighed all the Pro's and Con's together. One method would have been to put scores on different categories, putting a very concrete value as to how much something was worth.

However, we decided to just weigh it all in our heads. The conclusion was made that UT2004 would be the most optimal engine to use. Personally, I would have preferred to work with the CryEngine – but game creation is a dish served by more than one chef.

The selection of the Engine had lead us to the conclusion of part 2, and onward to the most interesting part, pipeline definition!

5.2.3. UT2004 in very very short

Unreal tournament is a first person shooter. It is very arcade-style, and consists of levels where you fight bots or other players in different modes. It does not include a story driven single player mode per se. One of the game modes include vehicles which can be used freely.

An Unreal 2k4 mod is a directory with datafiles and code files, placed in a subdirectory under the UT2004 installation folder. By supplying the argument -mod=subdirectoryname, to either the editor or the game, the game will use those datafiles **before** trying to read the original gamefiles.

UT2004 uses in-house package-files for their data. These datafiles are operated from the editor, and each have a different extension. Textures are stored in .utx files, character/vehicle animation are stored in .ukx files, static meshes in .usx files and so on.

UT2004 level creation is based on solid bsp modelling, but makes extremely heavy use of static meshes.

5.3. *Part 3 – Pipeline definition*

The most important part of the process was defining the pipeline. This also included identifying possible problem areas, and attempting to circumvent or minimize these.

In this part, I will not describe the results. These can be found under their own caption. I will instead try to focus on the way I thought when producing the results. To get the most out of this text, you should probably look at the results simultaneously.

5.3.1. Shared content creation

One thing to emphasize through the creation of a workflow is the fact that the content will be created by several persons, working parallel with each other. Since much of the data is directly or indirectly dependent on each other, everything needs to be transferred efficiently between the content creators.

There is really no alternative than using a version management system, such as CVS or SourceSafe. This allows all the project members to be automatically updated with new versions, as well as making backups safer. It also allows marking of which objects are being edited, and by whom, preventing that parallel versions of a binary file is being created. It is impossible, or at the least very time consuming to manually merge binary files.

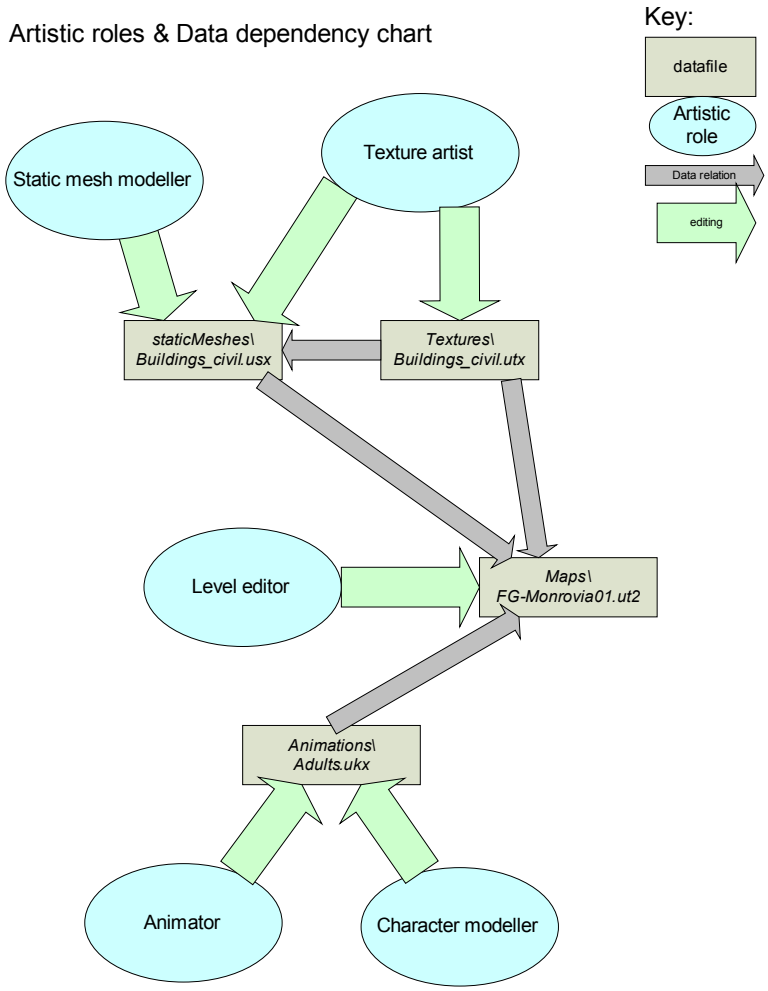

Fig. 2 Shared content creation

One may argue that a version management system is too cumbersome, and that all the artists just work on their own little personal pieces anyway. This could not be farther from the truth, as it very often arise problems with data that needs to be adjusted instantly by the only artist (or programmer) working late.

Since the correct usage of a version management system isn't trivial, as much thought as possible must be put into it's use in advance. One of the most important aspects of the version management system is the marking of editing. When one person work on a specific data-file, he/she marks in the system that he/she edits it. This prevents others from editing it at the same time.

The in-house pre-selected version management system was CVS, so I went along with that.

5.3.2. Common catalog structures, file naming conventions and location of data

By having a common catalog structure, including only the most recent versions of each file, you can avoid using old files. It also allows people to find data more efficiently. The actual construction of the catalog structure was done with a few other things in mind too.

Perhaps the most important thing was to try to fit the structure like a UT2004 mod. UT2004 has a structure which is pretty straight-forward. Which consists of a catalog for each type of data. While possible to change these paths, doing so would give room for possible errors, while contributing nothing. Therefore I decided to use the paths of UT2004 as a basis for all the other paths, instead of the other way around. I further expanded the definitions by using a document created by another group in the project – the shopping list for static meshes and items. By using the categories outlined in their document, the data could be placed in very logical, reasonable sized groups.

Since the whole data file will be updated each time a mesh inside it is updated, the size of a package should not to be too large. Another reason for avoiding big sizes is that if many objects are included in the same file, the chance that two (or more) people simoltaneously save the same file is huge. This is something that needs to be avoided.

The reason for not having too small package files, for example one package per object, is the way that the packages are handled in the editor. For each session you want to use a package, you have to open it. This is no problem if there are just a few packages, but if there are many, it will be very annoying and time-consuming.

By separating the work-folder (named Source) and the mod folder, the step from work to release is minimized. The work-folder consists of maya binaries, photoshop files etc.

The last thought I had in mind for the structure was to try to keep the system consistent. I therefore decided to structure the folders in the Source directory after the names of the datafiles, which data it's contents represented.

5.3.3. Workflow charts

Long areas of technical text are a nuisance to read (as you might have noticed by now...), and I really needed all the artists in the project to aquire the information. Therefore, I had to make toverview of the information easy. Hence; workflow charts. There are actually two types of workflows to consider...

5.3.4. Workflow charts, project-size

First of all is the project-size workflow issues. In a project, some things are just naturally more effective to do in a specific order. These interdepend very much with the fact that several people work together parallel. Since some data is dependent on other data, the order of which the data is created is important.

For our project, one of the most imortant aspects was the ability for whoever worked in the level editor with the level to be able to perform work simultaneously with the mesh artists. This required a list of the objects that were going to be used. Also, it required placeholder objects which the level creator could place even before the actual objects were finished. By successively replacing the placeholders with actual geometry, this allows for data to be automatically updated.

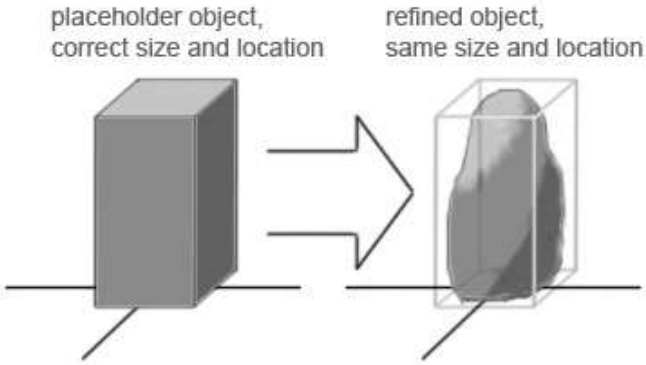

Fig. 3 Placeholder principle

Only after all the placeholder objects are completed, can the level creator seriously begin working. Any work he does before will not be at 100% efficiency. Incidentally, this required whomever creates the static meshes to be aware of this fact.

Of course, the first thing of all to do is to get the list of items completed, then design them all.

5.3.5. Workflow charts, detail-size

The details of how each individual artist works is a personal preference. When working together with other artists, some general principles must be obeyed so that one artist's work does not interfere with another's. Consider these questions ”when and where do I save?” and ”when and where do I export?” and ”when do I send stuff to CVS?”. The answers are very simple, but problems arise when the questions are never asked. Then there will be anarchy, and nobody will have control over the content.

So, basically – how the artist works within Maya is up to him/her (as long as the size & positions are correct), but the data needs to be put in the correct place so that others can work with it, so we need control over that.

5.3.6. From easy to hard to intermediate

When constructing an efficient workflow, everything mentioned in this document needs to be thought of simultaneously. There are also many technical aspects specific to whatever programs will be used that need to be obliged. The order I chose to produce the documents in was therefore, to first do whatever document I considered most easy. By choosing this first, I was able to quickly see if the general principles appeared to work, without getting caught up in just trying to fit it all together. This would of course not work if the easy documents would be directly dependent on the hard documents, as that would leave room for me to miss critical things.

I would then move on to the most difficult document, grinding it until it lappeared to work. The intermediate documents would then be a walk in the park.

5.3.7. Possible improvements

UE2 is a very refined engine, and it has a very polished workflow. However, I did identify a point where it could be improved. The animation pipeline included some irritating export/import bits which could be optimized away with some more automated plugins However, after weighing the time to create the improvement against the time that would be gained, Fredrik and I decided that it would be unlinkely to be worth creating..

Another point where the production pipeline could be enhanced was the CVS connection. If the CVS connection could be more automatic, the gain would be great. The first things that popped up were making the reserved edit marking, the update of the newest version, the package opening, the export, the package saving, and the package commit automatic.

The overview of items in CVS is also far from optimal. We searched for a long time to try to be able to configure it exactly the way we wanted it, but it failed. Two improvements It would be nice if we could see in the list automatically when someone had marked a file as reserved edit, (instead of having to manually query). The other thing would be to automatically update before going into reserved edit mode. (to make sure we don't edit an old file).

For CVS, the conclusion is that it really sucks when it comes to binary files. Just a few small improvements to the system and it would be bearable for binary folks, but alas, we are left in the dark.

6. Result

The result of my work can be seen in the documents I created. Some of the first documents regarding the indentification of candidates are however not public.

Artistic roles & Data dependency chart, see fig 2 (page 10),

This figure shows the generic artistic roles that an artist can take in the project. Each artist can be on several roles, it's just that these are separated from eachother.

Directory structure for foreign ground, see appenxid C

This document outlines principles naming conventions and the directory structures to be used in Foreign Ground.

Static meshes workflow UT2004, see appendix D

In this document, a very detailed workflow for static meshes is outlined. It includes version management, saving, updating, creating package files etc. It is intended to cover all things needed to get static meshes from Maya to UnrealEd, aswell as other aspects of mantaining an effective workflow in a multi-user production environment.

Animated meshes workflow UT2004, see appendix E

Although at first glance smaller, the Animated meshes workflow is actually more advanced. In order to keep the diagrams overviewable I removed version management, saving etc. This means that the animated meshes workflow requires the user to understand the static meshes workflow before viewing it.

7. Discussion / Errors

7.1. Lack of abstraction

By reading up on everything, I created a very good overview of the system for myself. One thing I should have done, was to more efficiently share this overview earlier in the process. The small picture in fig 2 (page 10) would most likely have been a good start. I did however create this very late in the process. I believe it was very difficult for others to start with all the nifty details I provided, instead of starting with the more abstract – then moving on to the concrete details..

7.2. Not much about the editor

I haven't tried much in the Unreal level editor. In fact, I'm still a beginner at the level editor itself. I learned the principles of the editor, but nothing about how to actually do it. This leaves room for some errors – if the principles and the execution in the editor differ.

7.3. Oh no, bugs!

Well so much for my nice functional workflow. It seemed like the direct export tool from maya sends the V-coordinates for textures flipped into the Unreal Editor. This means one of two things. We either have to create a script that automatically flips the coordinates temporarilly just before export, or we revert to using the functional but more cumbersome ActorX exporter. The solution I chose was to make a script that automatically flipped the UV's before export... It looked like this:

```
string $selexion[];
$selexion = `ls -sl`;

polyFlipUV -ft 1 -l 0 -ch 0;
polyMoveUV -t 0.0 1.0 -ch 0;

select -r $selexion;

(( export command here ))

polyFlipUV -ft 1 -l 0 -ch 0;
polyMoveUV -t 0.0 1.0 -ch 0;

select -r $selexion;
```

8. References

All of UDN is a great source for UE information:

http://udn.epicgames.com/Main/WebHome

Related bachelor theses,

These theses are in swedish, and are available through http://www.ltu.se.

Realistisk Interaktiv Miljö -modellering av miljö för realtids applikationer

by Simon Tingell

Game development process

by Fredrik Mäkeläinen

Realistisk Interaktiv miljö

by Erica Nilsson

9. Contact information

For feedback, further information etc, feel free to contact Oskar Holmstrand via http://www.oskarholmstrand.com.

10. Appendix A - Project specific factors

This is a list of the project specific factors for the ”Foreign Ground” project.

Character animation

The project was going to focus on character interaction, so having high level of realism on this was important.

Vehicles

We knew that there were most likely going to be useable vehicles in the game.

Visuals

Since the project is a playable prototype, it had to be visually stunning in order to convince whoever holds the money to invest in it.

Maya integration

Because knowledge takes too long to aquire, and expertise of Maya existed in the company, Maya integration was our focus – working with some other 3d package was not an option.

Dialogue

This was also related to character interaction.

Playback/logging

The customer issued a direct request to be able to review how the player played, in order to discuss whatever decisions he/she made.

Terrain

The scenario was going to be outdoor, most likely in a rural or jungle area. When the scenario was selected, it was an actual suburban area.

11. Appendix B – First list of engines

Halflife 2: Source by Valve

Possessing a strong emphasis on character interaction, HL2:source could be a very strong candidate.

Doom III by iD software

Doom3, also a state-of-the art engine renderwise, was added to the list, mainly because of its stunning visuals.

UT2004 by Epic Games

Being one of the most modded series of games in the history of gaming, UT2004 was without doubt worth looking into.

Full Spectrum Warrior by Pandemic Studios

The primary reason for this being in the list was that it was specifically mentioned by the customer, making it mandatory for us to research.

Operation Flashpoint by Bohemia Interactive

The primary reason for this being in the list was that it was specifically mentioned by the customer, making it mandatory for us to research.

FarCry by CryTech

Already looked at by members of the team, the FarCry & SandBox editor seemed to not only boast advanced lightning effects but also a very good editor with WYSIWYG.

Battlefield 1942 by Dice

With BF1942-II just around the corner, researching BF1942 might provide some useful insight into the new engine. Additionally, we knew that BF1942 featured vehicles and therefore the successor may be interesting.

Dice, short for Digital Illusions CE, is a AAA games-company based in Sweden.