EAI Endorsed Transactions
on Serious Games

Research Article ICST.ORG

Game engines: a survey

A. Andrade[1,*]

[1] Virtual Campus Lda. Av. Fernão de Magalhães, 716, 1 PT 4350-151, Porto, Portugal – http://virtual-campus.eu

Abstract

Due to hardware limitations at the origin of the video game industry, each new game was generally coded from the ground up. Years later, from the evolution of hardware and the need for quick game development cycles, spawned the concept of game engine. A game engine is a reusable software layer allowing the separation of common game concepts from the game assets (levels, graphics, etc.). This paper surveys fourteen different game engines relevant today, ranging from the industry-level to the newcomer-friendlier ones.

Keywords: animation, game engines, game design, serious games, physics, rendering.

Received on 15 October 2015, accepted on 15 October 2015, published on 05 November 2015

Copyright © 2015 António Andrade, licensed to EAI. This is an open access article distributed under the terms of the Creative Commons Attribution licence (http://creativecommons.org/licenses/by/3.0/), which permits unlimited use, distribution and reproduction in any medium so long as the original work is properly cited.

doi: 10.4108/eai.5-11-2015.150615

1. Introduction

One might think that each new game to date, particularly the ones with innovative game mechanics, is developed from scratch as a singular entity, with no relation to previous games. That was in fact true for the first video game platforms, such as the *Atari* or the *Commodore*, in the beginning of the 80's [1]. Back then each game had to be developed from the bottom up to make optimal use of the display hardware. Even if in other platforms the display hardware allowed for slightly more freedom, the memory available would often still not provide enough room for an overly generous, data-driven, game framework. Moreover, the rapid development of arcade hardware then booming, also meant that most game code would be thrown out after its original use, as later game generations would use completely different game designs to take advantage of the extra resources. Thus, game designs through the 80's were for the most part based on hard-coded rule sets and a limited number of levels and graphic data.

Although some "construction kits" and similar game creating software were available in the 80's, it was not until the mid-90's that game engines became common practice, especially associated with the rise of 3D first-person shooter games (FPS). The popularity of games as *Doom* and *Quake* was such that rather than working from scratch for each version, core portions of the software started being licensed (as the id Tech engine) for other developers to create their own graphics, characters, weapons and levels (the "game assets") [2]. This separation of game-specific rules and data from their basic and abstract concepts (as collision detection or the concept of game entity) also meant that teams could grow and specialize, improving productivity and game features. In the longer run, it ended transforming the typology of game development teams where we will now typically find several times as many artists as actual programmers [3].

By 1998, releases as *Quake III* and *Unreal* were already being designed with this abstraction in mind, completely separating the engine from the content [4]. By then, the practice of developing in-house game engines and licensing them also proved to be a useful auxiliary revenue stream for developers. Furthermore, the use of such game engines also allowed for faster and easier development of game sequels, which is a valuable advantage in the competitive video game industry.

While first-person shooters remain the predominant users of third-party game engines, any other game genre development is nowadays taking advantage of these frameworks – from role-playing games to action and adventure games, sport simulations, strategy games, etc. Interestingly, game engines are also the tool of choice for technical simulations and visualizations and serious games. Thus we will frequently see them applied to training in areas such as the military or medical fields.

The community of hobbyist game development can be traced back to the beginnings of the video game industry, acting at a rather small and often non-commercial scale, and focusing on innovation. By the end of the 80s the videogame industry grew stronger and got controlled by big publishers and retailers making it harder for independent developers to be commercially published [5].

By the 2000's decade, the readily availability of game creation tools and new online distribution methods allowed

*Corresponding author. Email: antonio_andrade@virtual-campus.eu

for a comeback of independent video game publishing (now commonly referred to as indie games) [6]. In addition to catering for niche markets, a number of these indie game titles also achieved significant commercial success — *The World of Goo*, *Minecraft*, *Braid*, *Super Meat Boy* or *Flappy Bird* are a few examples of that. Thus, this community has come to demonstrate how it is currently possible for individuals to create games without an industry-level setup.

2. What is a game engine?

So far it seems clear that a game engine is a piece of software that enables the creation of video games. However, we still do not have a proper definition of what can be expected in such frameworks. The main goal of a game engine is to abstract common video game features allowing for code and game asset reuse in different games. For that end, the following functionalities are typically found on a game engine:

- A rendering engine, for 2D or 3D graphics;
- Input handling (for keyboard & mouse, touch devices or other hardware, etc.);
- Game loop (the internal routine than recalculates game events every frame);
- A physics engine, with collision detection and response;
- Sound;
- A scene graph (which manages graphical elements and their relationships on the screen);
- Animation (for 2D sprites or 3D models);
- Memory management;
- Process threading (allowing for multiple, parallel processes).

Other functionalities might include:

- Scripting;
- Artificial intelligence;
- Networking;
- Streaming;
- Localization support;
- Multi-platform publishing.

More often than not, these features are supported by the aggregation of diverse third-party libraries (which in this context can be designated as middleware). Bringing them together under the same abstraction layer, under the same editor interface or Integrated Development Editor (IDE), with transparent interoperability and ease of use is thus the game engine developers' goal.

Currently, two major trends can be observed in game engines. Firstly, game engines are now much more likely to use high-level languages such as Java, C# or Python, which often results in a productivity gain for developers. Considering current hardware, the translation overhead for such languages has become negligible and game performance is mostly related with the power of the available graphic card.

The second major trend for some game engines is a common goal to enable cross-platform publishing while keeping the same code base. Most game engines are thus currently not only able to publish for desktop environments but also for the major mobile device operating systems and for different game console systems.

3. Game engines

The definition we have seen for a game engine was kept broad enough. In fact the exact features and workflow will vary from one system to another. As we have seen, some industry-level game engines have grown to be so powerful and flexible that any kind of game can be developed therein. Other are purposely kept simpler and with a narrower scope, either in their target user group or in their supported game mechanics. A few of them do not even require prior programming knowledge, allowing for newcomers to get started or artists to quickly prototype an idea.

We will now survey a number of game engines, going from the generic examples to the more specific. Although I intend to include the most relevant engines in each target group, it would not be possible to enumerate every one of them so bear with me if I happen to leave out any platform you feel should be included. Nonetheless, I am confident that the engines surveyed give you a good overview of the available options, targeting from hobbyists to industry-level developers.

3.1. Unity

The Unity game engine was first announced at Apple's Worldwide Developers Conference in 2005 and has since then produced major shifts in the video game industry. Firstly, it has been notable in its efforts to broaden its target platforms range. Specifically, at the time of this writing it is able to export to 15 platforms (with the same code base): *iOS*, *Android*, *Windows & Windows Store*, *Windows Phone 8*, *BlackBerry 10*, *OSX*, *Linux*, *PlayStation 3*, *PlayStation 4*, *PlayStation Vita*, *PlayStation Mobile*, *Xbox One*, *Xbox 360*, *Wii U*, and all major web browsers through a proprietary web player. Moreover, in *Wii U*'s case, it is the only official software development kit (SDK). One should, however, keep in mind that publishing for video game consoles generally requires a developer license from that system's owner, which often is costly and requires a preexisting game portfolio.

Also notable is Unity's price point as it is being offered for free for indie developers or companies with an annual turnover not exceeding US$100 000. A Pro version is also available, which includes a number of features for advanced use such as lower-level access to graphic procedures as well as a profiler and a GPU profiler – very useful for resource-usage optimization.

Figure 1. Unity interface.

Unity consists of a visual editor and an IDE, allowing for rapid prototyping. Simple scenes (with no complete game mechanics) can even be accomplished without a line of code. Its workflow is component-based, meaning it is modular and flexible enough for complex mechanics to be built out of smaller pieces of logic.

In practice, the basic game entity is called a *GameObject*. A *Scene* is the container for all *GameObjects*. *GameObjects*' spatial properties, graphical behaviour, game mechanics, etc., are each defined in different components. In addition to the built-in components (for rendering, animation, sound, etc.) the programmer can define more logic by coding new script components (which all extend the basic *MonoBehaviour* class). The built prototype can be tested directly inside the visual editor (with close-to-none compile time) and changes in the code base can generally be seen in real time.

Scripting inside Unity can be done in *UnityScript* (which is a custom language with a *JavaScript*-like syntax) or *C#* — the recommended option. Under the hood Unity transparently uses different middleware to support its features, namely *Mono* for scripting, *DirectX* and *OpenGL* for rendering, *beäst* and *Substance* for graphics, *fmod* for sound, *PhysX* as physics engine, *RakNet* for networking, etc. Moreover, Unity continues to integrate features easing the integration of social and in-game monetization aspects into its framework. Another aspect making Unity a great reference to start with game development is the size of its community and the availability of a marketplace where game assets and components can be exchanged (to quicken development or serve as learning material).

Unity is being used for both smaller-scale mobile games as well as AAA industry-level productions, which demonstrates its versatility. A few examples developed with Unity include *Temple Run*, *Bad Piggies*, *Monument Valley*, *HearthStone* or *Wasteland 2*.

Official website: http://unity3d.com/

3.2. Unreal engine

Unreal Engine 4 is the current iteration of one of the first major game engines to have come out to the public. Unity also supports multiplatform publishing, namely using *DirectX* (*Windows*, *Xbox One*), *OpenGL* (*OSX*, *Linux*, *PlayStation 4*, *iOS*, *Android* and *Windows XP*) as well as *WebGL* (for HTML5 web browsers). Epic games has also recently revised Unreal Engine's pricing plans, providing its complete technology for free for non-commercial use, and for all other cases licensing the software for a small subscription and royalty fee. Although initially developed to support the *Unreal* first person shooter, and arguably not as rich-featured as Unity, the Unreal Engine has grown to be a very powerful engine capable of supporting any game genre (including 2D environments). Besides, like Unity, the Unreal Engine also has a considerable community and asset marketplace.

Scripting inside the engine is done using the C++ language – which might feel like a drawback compared to the higher-level languages Unity provides, for instance. This seems to reflect the overall feeling of the editor itself which, for instance, also only recently introduced live preview of the game changes ("hot update"). Perhaps attempting to balance that trait, this latest version of Unreal Engine is introducing a rather innovative feature designated *Blueprint Visual Scripting*. These Blueprints consist of a node-based interface to create gameplay elements from within the editor. From its documentation, the goal of this approach is to allow for designers to prototype and experiment with virtually the full range of concepts generally only available to programmers. This tool is described as being so powerful that entire games can be built using it exclusively.

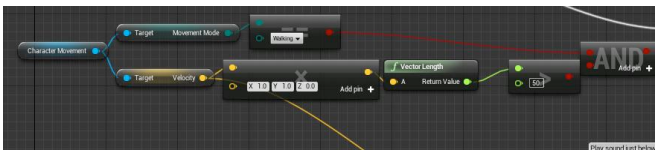

Figure 2. Unreal Engine's Blueprint sample.

While the Unreal Editor might come out as a less newcomer-friendly, the general community thoughts seem to point that it is also more capable of producing impressive visuals when compared to other engines such as Unity. Games developed with Unreal Engine include *XCOM: Enemy Unknown, Tony Hawk's Pro Skater HD, Mass Effect 2 and Gears of War 3*.

Finally, another differentiating feature found on Unreal Engine is that its source code is open for subscribers. This allows both for community improvements to be included as well as for a learning experience for the curious developer.

Official website: https://www.unrealengine.com/

3.3. CryEngine

CryEngine is another industry-level game engine. It was developed by Crytek to support all of its games, starting with *Far Cry* and all its sequels. It is currently being licensed through an affordable subscription and allows for desktop and console publishing.

When compared to the two previous engines, CryEngine seems to have the steepest learning curve, with a less friendly editor interface, a smaller community and no asset marketplace. Nonetheless, CryEngine is known to produce state-of-the-art graphics and performance. In fact, in August 2014, Ryse: Son of Rome, developed within CryEngine, won the SIGGRAPH Award for Best Real-Time Graphics. CryEngine scripting is done in C++ or Lua languages.

Official website: http://cryengine.com/

3.4. Torque

Torque comes in two flavours: Torque 2D and Torque 3D. This is another example of an in-house engine released to the public. However, it also ended up being rendered free and open-source by 2012. Both versions of the engine are scriptable using TorqueScript, a C++ like language. Although its features could be compared to the previously reviewed engines, the most interesting feature is the price point and the free access to the engine source code.

Official website: http://www.garagegames.com/

3.5. Flixel

Flixel is a free and open source game-making library written in ActionScript 3. Although it provides most of the constituents and heavy-lifting expected in a 2D game engine, it does not include a visual editor and is intended for developer use. Nonetheless it is considered a good entry point for game development, particularly for those with some experience with Flash or ActionScript. Flixel's author developed the library while working on his own games, which include *Gravity Hook* and *Canabalt*.

Official website: http://flixel.org/

3.6. GameMaker: Studio

GameMaker: Studio is a commercial game creation system and engine supporting many genres, and targeting at desktop and mobile environments as well as the latest PlayStation consoles. GameMaker is built with a non-programmer user in mind, providing a drag-and-drop editor that allows the creation of games by abstracting the underlying logic. This is generally called a visual programming language (VPL) and was originated by the MIT's Scratch computer language learning environment.

Nonetheless, GameMaker also provides its own scripting language (Game Maker Language) for more advanced functionalities. Also important to notice is that GameMaker's
editor is mostly target at 2D graphics and, although it supports 3D, any 3D functionality, will have to be custom scripted. Hence, GameMaker's advantage is its easier learning entry point when compared to Unity, for instance.

Official website: https://www.yoyogames.com/studio

3.7. Construct 2

Construct 2 premise is somewhat similar to GameMaker's: enabling non-programmers to create 2D games. Thus, Construct also uses a visual, drag and drop editor for all logic development, based on events and behaviours. Like GameMaker, Construct 2 can be extended and scripted using JavaScript. Furthermore, although Construct also claims to publish into the major desktop and mobile platforms, its primary target is HTML5/JavaScript. Therefore, any non-browser version (i.e., mobile, desktop or console) is contained by a DOM and JavaScript enabling wrapper. This architecture generally hampers performance. Nevertheless, Construct should still be considered an option for newcomers not interested in learning programming skills, and part of the community claims the editor as a friendlier interface when compared to GameMaker.

Official website: https://www.scirra.com/construct2

3.8. Stencyl

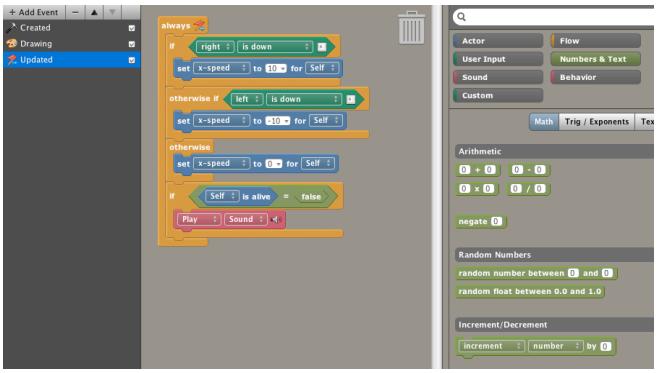

Figure 3. Stencyl's visual programming language

Stencyl is yet another 2D game creation platform. It provides both authoring tools (to edit graphics and tile sets) and an IDE to edit game logic using VPL modular pieces in a fashion similar to Construct 2. Unlike GameMaker and Construct, the Stencyl IDE publishes to Flash when targeting desktop or browser platforms.

Official website: http://www.stencyl.com/

3.9. Visionaire Studio

Visionaire is a commercial environment for the creation of 2D point & click adventure games that was primarily developed with simplicity and usability in mind. This editor requires no programming knowledge, as users are able to define all actions visually. Yet it also allows the use of Lua as scripting language. Audio and video are also supported out-of-the-box, as well as the localization of dialogs.

Visionaire Studio allows publishing for both desktop and mobile platforms and it has been used in the past for game such as Deponia or The Whispered World. Wintermute Engine was a free and open source alternative to this engine but unfortunately there have been no updates since July 2012 so support may start lacking.

Official website: http://www.visionaire-studio.net.

3.10. eAdventure

eAdventure is a free and open-source game engine and visual editor for point & click adventure games developed at the e-UCM. Its main target is serious games development and thus provides a number of unique features in the field. Specifically, eAdventure provides a built-in assessment mechanism, support for real-time adaptive learning scenarios and integration with different educational standards for learning management systems integration. Another interesting feature regarding this engine is its commitment to accessibility providing tools to include colorblind players.

Official website: http://e-adventure.e-ucm.es/

3.11. RenPyl

RenPy is a popular free and open source game engine for the creation of visual novels, supporting branching stories, save file systems, rollback to previous points in story, scene transitions, etc. It is developed in the Python language and

allows both for Python scripting and a simplified RenPy scripting syntax.

Official website: http://www.renpy.org/

3.12. RPG Maker

RPG Maker is a commercial game creation software aimed at the creation of the traditional, top-view, 2D role-playing games. RPG Maker provides tile set authoring tools, tools for editing characters and items and a simple scripting language to program the game logic. RPG Maker only supports publishing for the Windows operating system.

Official website: http://www.rpgmakerweb.com/

3.13. Phaser

HTML5 is a rather modern medium for game publishing, and hence still rarely used by the industry. Its main advantage is that any modern web browser supports HTML5 and JavaScript, thus cross-platform support is naturally achieving without using wrappers or browser plugins (as is the case for the Unity Web Player).

Phaser is a free and open source HTML5 + JavaScript (and TypeScript), generic 2D game engine. It supports WebGL and Canvas rendering and most features you will require to develop in this context. Most importantly (and uncommon for HTML5 game engines), Phaser has a game development company behind it – meaning the code is tested on a daily basis and optimized at every opportunity, as well as an active community (with more than 100 contributors). The documentation is extensive and accounts for more than 300 examples.

Official website: http://phaser.io/

3.14. Turbulenz

Turbulenz is another open source game engine developed with optimized JavaScript. Although it is intended for browser use, mobile-targeted bundling options are also available. In addition to the common 2D features, Turbulenz also supports 3D rendering and physics. Although games developed with this engine can be freely distributed, Turbulenz itself provides a publishing platform (therefore Turbulenz is also well backed). Although Turbulenz seems more feature-rich when compared to Phaser, its documentation and setup seems much more complex.

Official website: http://biz.turbulenz.com/developers

4. Summary

A few other engines were left out, either because they did not provide more value in the same category as the ones already presented (e.g. Shiva3D), or because they are too exclusive or expensive to be considered (even though they are indeed comparable feature-wise), such as Source Engine or Gamebryo.

For simplicity's sake, the survey is summarized in Table 1.

Table 1. Summary of game engine survey

Game Engine	Game gender & Renderer	Publishing Target	Starting Price for Commercial Usage	Scripting Language	Supported Platforms
Unity	Generic 2D / 3D	Windows; OSX; Linux; PlayStation; Xbox; Wii U; iOS; Android; BlackBerry; Windows Phone; Browsers; etc.	Free	C#	Windows; OSX; Linux
Unreal Engine	Generic 2D / 3D	Windows; OSX; PlayStation; Xbox; iOS; Android; Browsers	Free	C++	Windows; OSX; Linux
CryEngine	Generic 3D	Windows; OSX; Linux; PlayStation; Xbox; Wii U; iOS; Android	€9.90/month	C++ or Lua	Windows; OSX; Linux
Torque	Generic 2D / 3D	Windows; OSX; iOS	Free	TorqueScript, similar to C++	Windows; OSX; Linux
Flixel	Generic 2D	Flash; iOS; Android	Free	ActionScript 3	Windows; OSX; Linux
GameMaker: Studio	Generic 2D	Windows; OSX; Linux; iOS; Android; Windows	Free or $99.99	Visual Programming Language or Game	Windows; OSX

		Phone; Browsers		Maker Language	
Construct 2	Generic 2D	HTML5	€99.99	Visual Programming Language and JavaScript	Windows
Stencyl	Generic 2D	Windows; OSX; Linux; iOS; Android; Flash	Free, $99 /year or $199/year	Visual Programming Language	Windows; OSX; Linux
Visionaire Studio	2D Adventure	Windows; OSX; Linux; iOS; Android	€49	Not required.	Windows; OSX
eAdventure	2D Adventure	Windows; OSX; Linux; Browsers	Free	Not required.	Windows; OSX; Linux
RenPy	2D Visual Novel	Windows; OSX; Linux	Free	Python	Windows; OSX; Linux
RPG Maker	2D RPG	Windows	$69.99	Not required.	Windows
Phaser	Generic 2D	Browsers	Free	JavaScript	Windows; OSX; Linux
Turbulenz	Generic 2D / 3D	Browsers	Free	JavaScript	Windows; OSX; Linux

There is no such thing as “the best game engine”. However, when choosing a game engine one should ask himself a few questions: What kind of game am I developing? What is my prior experience and how much effort am I willing/able to put into its development and/or learning a new platform? Is this an isolated project or am I getting into game development for the long run?

For someone looking to get into game development for the medium or long run Unity or Unreal Engine are the recommended choices, enabling one to develop from the simplest projects to AAA-level games. For someone just looking into quickly getting a working prototype running or not so acquainted with logic programming, some other platform would be preferable.

Newcomers should also take the time to experiment with a couple of different options and get a feel of different editor interfaces and game abstractions. In any case, before making a final choice one should evaluate how active is that game engine’s community, how much support he can expect from it and what is the engine documentation’s and samples’ quality.

References

[1] BRABEN, D. (2011). Classic game postmortem. [Online]. Available: http://www.gdcvault.com/play/1014628/Classic-Game-Postmortem.

[2] CARMACK, J. (2004). “DOOM 3: The Legacy.” Transcript of video retrieved June 2004 from the *New DOOM website*. [Online]. Available: http://www.newdoom.com/interviews.php?i=d3video.

[3] WORCESTER Polytechnic Institute (2005). [Online]. Available: http://web.cs.wpi.edu/~id111x/c05/slides/intro.pdf.

[4] BLESZINSKI, C. (23 February 2010). History of the Unreal Engine. [Online]. Available: http://www.ign.com/articles/2010/02/23/history-of-the-unreal-engine

[5] DARLING, S. (February 1985). Birth of a Computer Game. In *Compute!*, 48.

[6] STUART, K. (27 January 2010). Back to the bedroom: how indie gaming is reviving the Britsoft spirit. [Online]. Available: http://www.theguardian.com/technology/gamesblog/2010/jan/26/casual-gaming-indiegames

[7] MECHNER, J. (2011). *The Making of Prince of Persia*.