Modding Support of Game Engines

LUKAS SCHREINER and SEBASTIAN VON MAMMEN, Dept. of Computer Science

Julius-Maximilians-University, Würzburg, Germany

This paper aims at showing how different game engines support modding. To this end, the concepts of modding and game engines alongside their historical evolution are introduced first. Additionally, some well-known game engines and mods are presented, and their unique features are described. Further, the effects of using game engines and those that modding has on games, developers, and players are laid out. Particular focus is placed on the effects on competitive games. These effects include an enhanced lifespan for games and the requirement for more development effort. Finally, the modding-support of selected game engines is illustrated and compared with each other. These engines are the id Tech engine(s), Unreal Engine, the Source engine, and Unity. The paper is then concluded with an outlook on future research possibilities and recommendations to developers.

Additional Key Words and Phrases: game engines, game mods, content creation, user-generated content

1 INTRODUCTION

Some of the most popular games started as modifications, or mods, of existing games. Prominent examples of this are the Counter-Strike [56] series and DOTA 2 [59]. The former was originally a mod for Half-Life [55] and was released as "Half-Life: Counter-Strike", and the latter was a mod for "Warcraft III" [41]. Both were later re-released as independent games. A short history and description of both games can be found in section 2.5. Complementary to mods that became successful titles on their own, some games are known for their active modding communities. Examples of this are The Elder Scrolls V: Skyrim [39] and Minecraft [49]. Both games are more than nine years old and are still extended by numerous new mods on a daily basis. Playing these mods is very popular—the download count for Skyrim mods, for instance, amounted to more than 500M until 2019 [29]. Before the 1990s, developing mods was much more complicated as developers did not share their code or interfaces. With the emergence of game engines in the 1990s and developers actively supporting mods, their popularity rose quickly and the modding community emerged [21, 25].

In this paper, we analyze current implementations of modding support in game engines by identifying and comparing the respective functional features. It builds on a literature survey which highlights the impact of modding support and sparks novel research directions. Additionally, the paper aims at helping developers to decide whether and how to provide modding support. To this end, section 5 contains some developer guidelines deducted from the results. Section 2

Authors' address: Lukas Schreiner, lukas.schreiner@stud-mail.uni-wuerzburg.de; Sebastian von Mammen, sebastian.von.mammen@uni-wuerzburg.de, Dept. of Computer Science
Julius-Maximilians-University, Würzburg, Am Hubland, W¨rzburg, Bavaria, Germany, 97074.

sets out with a concrete definition and classification of mods, as well as a short history of game modding. It also outlines the effects of modding for the developers, the players and the games themselves. Following a similar structure, due to their importance for modding, we briefly outline the history and give a definition of game engines in section 3. Regarding concrete analysis, we limit ourselves to the four representative game engines: id Tech engine, Unreal Engine, Source Engine, and Unity. In section 4, we detail the support of modding by these four game engines and compare them based on aspects such as their general modding-friendliness as well as the associated versatility. Afterward, we discuss our findings and deduce guidelines for developers. We conclude the paper with an outlook on future research opportunities.

2 MODDING

In this section, the terms "mod" and "modding" are explained, considering also three categories of game mods. Afterward, various effects of modding and moddable games are laid out. To illustrate the concepts and impact, we conclude this section by introducing three well-known mods that were later developed into full games.

2.1 Definition

The terms "mod" and "modding" appear in different contexts. However, this paper only considers modding in the context of video games and game development. According to the definition by Finch [21, 24], "mod", i.e. the shortened form of modification, refers to edits to a game made by its players—similar to fan fiction. These kinds of mods are mostly available free of charge [21, 24]. There are no limits in terms of the size and scope of a mod. They are ranging from minor changes and bug fixes all the way to entirely new gameplay and experiences [21, 24]. However, the border between a mod and an entirely new game is not well-defined [21]. The creators of mods are usually referred to as "modders" [33]. When modding emerged, games were typically not designed with modding support. Therefore, practically all mods of the 1980s were unauthorized, and modding was related to reverse-engineering, or hacking, the games. Although this context has changed over time, as shown throughout this paper, some of the early hacking spirit can still be found in nowadays modding communities [20]. Other related usages of the term include "modding" of computers and game consoles. For PCs, this includes modified designs but also hardware upgrades and processor overclocking for better game performance. Modding of consoles is often closely related to hacking and can enable features that were not intended by the manufacturer. While both of these can improve the gaming experience, we do not consider them part of the definition of modding in the remainder of this paper [33].

2.2 History

It is subject of debate which mod was the first, but different sources point to several games from the 1980s. Lode Runner [53] is mentioned as the first game with an editor and Ms. Pac-Man [45] as the first mod. Some even consider SpaceWar! [54] from 1962 to be the first mod [21]. As noted in section 3.1, the first game engines and tools to modify games appeared in the 1990s [21]. One of these games was ZZT [51] from 1991, which was sold together with an editor and its own scripting language [21]. Similarly, the developer of the first-person shooter (FPS) DOOM [46], id Software, released multiple tools and instructions for creating mods [21].

2.3 Types of Game Mods

Game Mods can be divided into different categories. While some alter the gameplay, others make visual changes or improve the game controls. This section introduces three types of (software) mods that directly relate to games, as presented by Scacchi [33].

2.3.1 User Interface Changes. User interface changeing mods consist of three subcategories. The first subtype allows players to make changes to their in-game characters, including new attire or entirely different avatars. Customizing the interface is allowed by the second subcategory. While these two types of mods mostly introduce cosmetic changes, the third subtype can influence the gameplay: This is achieved by changing the information presented to the user, for example by adding information about events outside of the standard view [33].

2.3.2 Game Conversion Mods. Game conversion mods are likely the most common type of game mods. Usually, these mods only change portions of the game by adding or modifying existing parts. These changes can include player characters, non-player characters, resources, objects, the virtual environments, or level designs. Additionally, these mods can alter the game mechanics and gameplay rules [33]. Aside from these partial modifications, there are also complete conversions of games. These could be parodies of the original game or entirely new games with different gameplay. While game conversions usually result in new games, the modified games are sometimes used for educational purposes [33].

2.3.3 Machinimas. Machinimas are a special type of game mods. Instead of changing gameplay, they are used to make movies inside of video games. They started in the "Quake movie" community which, like the name suggests, created movies using Quake [47] [26]. At first, players recorded their avatar's movement manually and copied the movement data into the demo-files of the game. These files were then distributed, and owners of the game could play them back on their PC. Therefore, it was not the game that was modified, but rather the demo files and tools used to manipulate games [26]. As games started to add functionality to record in-game footage, the games became the target of modification. Additionally, players did not control the characters anymore for recording but created scripts to do so [26]. Over time, the popularity of creating machinimas grew, and Epic Games even held a contest on creating mods and machinimas using the Unreal Engine [43] [26]. With the inclusion of cut-scenes into newer games, film-recording functionality became part of the games' code bases. This allowed machinima makers to use scripts for animation and camera control and, thereby, modify the games themselves. This meant a shift in focus away from play performance towards changes in perspective and guidance of the games. However, not all tools provided through the newer game engines found approval by machinima makers, and the community keeps creating their own tools [26].

2.4 Effects of Modding

Modding can have various effects on the developers, the development life-cycle, and as a result, on the game and its players.

2.4.1 Effects on Development. Since developing mods is usually less time-consuming and less effort than creating an entire game, they are well suited for single developers who want to join the game development industry. Gabe Newell, the founder of Valve, recommends "making content using the MOD tools that are out there" as the best option for people with no prior experience to game design and development [21]. For the actual developer of a game, providing modding support means additional investments in terms of backend functionality, APIs, documentation, possibly GUIs and

graphical editors, as well as testing of the respective modding pipelines [28]. As a positive outcome of this invest, a game developed with modding features implies a modular code architecture. This not only empowers the players/modders to change contents and mechanics but also facilitates the processes of development, design and maintenance of the original product. Modding can also save development time, since modders might fix bugs or add features that would have had to be addressed by the developers, otherwise. The respective mods could even be integrated into the original code branch. Along these lines, moddable games can last much longer due to the user-generated contents that do not require additional investments by the creator of the game and still drive the sales, even after a long period of time. This especially applies to mods that are also used to fix and improve the game. However, the developer runs the risk of overburdening the modding community, which can quickly trigger the impression that the developers do not care about the game [28, 30].

2.4.2 Effects on P lay. Since installing mods for a game is usually optional, players can choose which mods they want and do not want to use. This means that players rarely have a disadvantage just by modding support of a game. However, since mods often alter the gameplay, they can be an issue in the context of competitive games. Mods could be exploited or specifically designed for cheating and gaining unfair advantages. Augmenting the information about the current game state is one way to do so. For instance, retrieving the whereabouts and resources of one's competitors in a capture-the-flag first-person shooter would give one player an unfair edge [33]. Next to providing according "wallhacks" that effectively let the player look through walls, "aimbots" simplify the task of target acquisition, pointing and shooting. These well-established examples underline the opportunities for unfair game mods in competitive settings [20].

However, modding can have positive effects on players. Depending on the type of a mod, players can benefit from enhanced gameplay experiences based on interface customizations, or play an entirely different game with game conversion mods, increasing the original game's replayability [28, 33]. It also offers players the opportunity to shape existing games in ways they imagine. Without mods, games can only be played as the developers intended, giving the players less freedom [23].

2.5 Relevant/Special Mods

We illustrate the observations so far by providing three examples of popular mods: Counter-Strike [56] and DotA [59] which both started as mods but became so successful that they were later recreated as independent games, as well as Garry's Mod [44], which is based on Half-Life [55] but takes a unique role as a so-called meta-mod.

2.5.1 Garry's Mod. As a meta-mod, Garry's Mod, can be viewed as an open-ended physics-based game, but it resembles more an experimental sandbox with all the features of a comprehensive modding toolkit and is, thus, used to create completely new games [33]. Additionally, Garry's Mod is often used to create machinimas as it allows for animation of objects and recording [36].

2.5.2 Defense of the Ancients. Defense of the Ancients (or DotA for short) was conceived as a mod for Blizzard's Warcraft III [41] from 2003, and itself based on a mod for StarCraft [40] named Aeon of Strife. While Warcraft was a strategy game, DotA combined elements of Real-Time Strategy and Role-Play Games [13, 24]. This combination marked the beginning of an entirely new genre, nowadays known as multiplayer online battle arena (MOBA) [34]. The DotA mod was extended and maintained for a few years by different modders, and later called DotA-Allstars [13]. In 2009, Riot Games, with employees formerly involved with DotA or Blizzard [12], released League of Legends [52], which built on the gameplay of DotA but was an independent game with new characters and gameplay elements. In 2013,

Valve also released a MOBA game titled DOTA 2 in cooperation with DotA's last maintainer IceFrog. This game can be considered a sequel to DotA as it directly adopted numerous characters and gameplay elements [34]. Paving the way for MOBA titles such as DOTA 2 and League of Legends, modding needs to be considered a major contributor to the growth of the eSports industry [34].

2.5.3 Counter-Strike. Counter-Strike [56] was created in 1999 as a mod for Half-Life [55] by an independent developer. Since Valve had not released an SDK for Half-Life at that time, the creator had to figure out things like character and weapon creation himself [37]. While Counter-Strike is an FPS like its base game, it implements a different gameplay. It is a multiplayer game where one team plays as a group of terrorists and the other team as counterterrorists. Generally, the teams have to shoot each other to win, but there are other goals such as planting bombs or rescuing hostages [37]. Early versions already received as many awards as Half-Life itself and gained a large player-base. This success was furthered when Valve supported the development starting in 2000 [37]. Since then, multiple successors were developed and published by Valve [33]. The latest installment, called Counter-Strike: Global Offensive, even became one of the most played games on Valve's platform Steam and one of the biggest E-Sport titles [31].

3 GAME ENGINES

The rise of game engines in the 1990s drastically simplified modding and propelled its success. Therefore, this section briefly elaborates about the short history of games engines, introduces a definition of the term, and describes four well-known game engines that represent the state-of-the-art.

3.1 History

Early computer games were written from scratch, i.e. there were no large libraries or frameworks the games could be built upon, and newly developed code snippets could rarely be reused for other games due to the heterogeneity of hardware and operating systems [23]. With their standardization, the opportunity for code reuse increased and professional game developers started creating and refining their libraries and toolsets. In the 1990s, this reached a point where a game's contents could be widely kept separate from its code base as has been emphasized by id Software's FPS DOOM. This game's core code base managed user input, game logic, physics simulation, graphics and audio rendering, and the contents of the game, including meshes, textures, level designs, audio files, but also the game rules were added on top, to flesh out a concrete game instance [25]. This separation allowed other developers to license the game engine and create new titles without programming the core/back-end part. Due to the reduced efforts, smaller studios and independent developers could also produce modified versions of existing games. This was often enabled by tools that the original developers provided free of charge—the modding community was born [25]. In the late 1990s, the two games Quake III Arena [48] and Unreal [43] were developed with the goal of code reuse and modding. These games' engines featured scripting languages such as Quake C for easier customization. Today, licensing game engines has become a veritable business model.

3.2 Definition

While some engines attempt to support development of any kinds of games, others are built for specific purposes and are more focussed on specific gameplay experiences or tooling pipelines. For example, an engine for race car games can offer various optimizations in terms of continuous rendering progressions across a race track or adapted driving physics, whereas an engine tailored to the adventure game genre would provide extensive functionality for dialogue

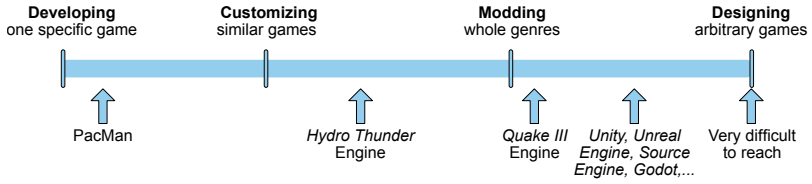

Fig. 1. Game engine reusability gamut, adapted from [25].

scripting and inventories, etc. More versatile engines provide highly configurable views and rendering perspectives and broad tooling amenities. Yet, the broader the targeted spectrum of contents and mechanics, the harder it is to reach competitive efficiency in specific domains [23, 25]. Elaborating about game engines, Gregory emphasizes a framework's comprehensive integration of functionality, a wide spectrum of supported games, and the separation of code and contents [25]. Accordingly, he orders game engines along a spectrum similar to the one shown in Figure 1. We adapted the gamut of game engines with respect to the categories' abilities: Following Gregory's wording that customization could yield numerous very similar game titles and modding could cover whole genres, we emphasized the aspects of development of single games and design of arbitrary games at the far ends of the spectrum. Games developed without a game engine can be found on the very left of the spectrum, while the far-right would be the (possibly) unachievable concept of an engine with unlimited possibilities. High-level, easy-to-use design opportunities typically introduce performance losses in one way or another. However, a vertical refinement process of the designed product's efficiency, i.e. stepping through all the different system layers (operating system, drivers, frameworks, subengines, and individual programming components) to harness cross-layer and cross-component optimizations, whether manual or automatic, could counteract this disadvantage. Therefore, we would not assume that reaching the right-hand side of the spectrum is impossible, but consider it a rather difficult enterprise with costs quickly outweighing the benefits. Reading the figure from the left to the right up to the mentioned, state-of-the-art engines also represents the evolution of game development over time, so far. And each step toward straight-forward game design, i.e. the right end of the spectrum, bears advantages in terms of productivity: Code re-use, infrastructure re-use in terms of code tooling but also regarding the asset pipeline, the distribution of work across team members or contractors, as well as bringing together assets and contents favor cost reductions and increases in productivity and quality.

Game engines have to provide various functions. These may include but are not limited to input processing, rendering of graphics and sound, support for complex assets such as multi-texture materials, rigged animations, 3D soundscapes, various special effects engines, physics computation, artificial intelligence models and algorithms, especially integration of state machines, behavior trees and various path finding support methods, network access to support multiplayer games, etc. Usually, these functions are handled by individual sub-engines. Not all game engines support all of these features, and the supported features are not always equally pronounced [22].

3.3 Popular Game Engines

To underline our analyses, we briefly describe four game engines that (1) have reached considerable popularity, often even beyond the game development community, that (2) are available to the broad public, and not exclusive to their creators, and (3) allow the creation of moddable games (which will be further elaborated on in section 4). For other engines and comparisons of other aspects, see for instance [22] and [25].

3.3.1 id Tech Engine. The first iteration of the id Tech game engine was released in 1993 by id Software. While the milestone versions of id's engines are numbered, all of them are known by the name of the primarily developed game title, e.g. the Doom Engine was followed by the Quake Engine, the Quake II Engine, etc. [18, 32]. The Quake Engine was the first game engine with support for fleshing out 3D environments which reached beyond perpendicular walls on an otherwise flat floor. It marks a milestone in game engine history [19] and earned itself the version title id Tech 2. It supports various PC operating systems and major game consoles of the time of its release [32]. id Software used their engines for their own games and licensed them to other developers, as well. However, this changed with version 5, which was the first version that was not released as open-source and exclusively available to members of ZeniMax Media. Nowadays, id Tech engines lost popularity and newer versions are often considered not as innovative as previous iterations [18, 32]. While the id Tech engine allowed created games to be modded, modified versions of the engine itself were also created. Thus, the newest Source engine developed by Valve still contains parts of the Quake II engine, which their GoldSource engine was based on [18, 32]. This family of game engines is described in section 3.3.3. id Tech engines are generally a bit more to the left on the reusability spectrum [25]: They mainly serve to create FPS games [2], which were often called "DOOM clones" according to the name of the genre-defining game created with the first id Tech engine [18].

3.3.2 Unreal Engine. The Unreal Engine was released in 1998 together with the game Unreal and became the biggest competitor to the id Tech engine in the FPS genre at the brink to the new millennium [18]. However, its newer versions are still widely used today, with version 5 being the latest installment. Since 2015 the engine is free to use for everyone until generated profits exceed a certain threshold, including free access to its source code [35]. The Unreal Engine is based on C++, supports C++ natively and a visual scripting language called Blueprint. There is support for a wide variety of devices, including modern smartphones and game consoles. Since the engine is open-source, it can be modified by other developers. This allows game creators to create mod editors specifically for their games [10, 22].

3.3.3 Source Engine. The Source engine was created between 1996 and 1998 when Valve developed Half-Life [55] and adapted the Quake engine to their needs. All later versions of the Source engine still contain parts of the original id Tech 2 engine [36]. The initial name for this engine was GoldSource, or GoldSrc, and Valve only released new features under the "Source" branch to avoid breaking mods and tools for Half-Life, which, at had quickly mobilized a considerable community. The name was changed with the release of Half-Life 2 [57], which made use of the engine's advancements since the first installment. However, this was the result of continuous development and not an entirely new engine. Until today, only one new major version was released. All development in-between was integrated continuously and used in later games [36]. Due to the active modding support, multiple games like Counter-Strike [56] and Team Fortress 2 [58] emerged from mods for the Half-Life series. Nevertheless, the engine is also still used as the basis of many Valve games like DOTA 2 [59] [36].

3.3.4 Unity. Unity was first announced in 2005 and has since become one of the most popular game engines. It supports one of the broadest spectrums of target platforms among current game engines, including PCs, smartphones, and game consoles [22]. It is often used for mobile games like Pokémon GO [50], one of the most popular handheld games in recent years [38]. However, there are also some bigger games developed with Unity for PCs, such as Cities: Skylines [42], a city-building simulator that comes with modding support and various editors [3, 27]. Unity is considered beginner-friendly, as its interface is simple to use, and it provides many assets and tutorials. However, scripting in

Unity is done using object-oriented languages like C#, JavaScript, or Boo. Many other game engines use C++ instead, therefore switching to a different, more advanced one is not easy [22, 25].

4 MODDING-SUPPORT OF GAME ENGINES

In this section, we discuss the modding support of the different engines listed above (section 3.3). We summarize the results of our research in tabular form and conclude this section by means of a comparative analysis.

The id Tech engine supported modding from the very beginning, and mods of DOOM [46] became the early FPS-games. Since id Tech engine versions 1-4 were released as open-source, other developers could adapt the engines to their needs. Since the newer versions cannot be licensed anymore, there is no way to obtain or modify them. Older versions used the GNU General Public License, which allows modifications and distribution. However, id Tech engines are not as versatile as other engines and creating anything other than FPS-games is hardly possible [18, 25, 32]. Nevertheless, some game conversion mods for DOOM [46] and the Quake series [47] were released. These include Aliens TC, which moved DOOM to Aliens' space horror setting by completely changing the game's assets, and the original version of Team Fortress based on Quake, which introduced eight different classes to reward cooperative play which can still be found in more recent FPS games [24].

Epic Games actively supports modding and claims that the Unreal Engine was "designed with modding in mind". Since the engine is open-source, modders can gain full control over all aspects of the engine. Additionally, Epic Games provides a subforum dedicated to modding as well as tutorials and documentation [10]. The Unreal Engine's license permits mods and games to be distributed to anyone through means the developers choose. However, if mods are sold, the same royalty fees apply as they do to game sales. Modified versions of the engine itself can only be distributed, but only to the Unreal Engine community through official channels [10]. The current Unreal Engine also ships with various editors that can, in part, be used for modding. These include, among others, a user interface editor, a level editor, a scripting editor and one for animation. As developers can modify the engine, tailor-fit modding editors can be offered for specific games. One example is Bus Simulator 18—an editor can be acquired through the Epic Games store that allows various mods such as liveries for the busses. However, there are also comprehensive game conversions of games created with the Unreal Engine such as Red Orchestra, a WWII FPS based on Unreal Tournament [7, 15, 24]. With the release of Half-Life, Valve started to create an eco-system for mods and expansion packs. They also purposely did not alter the engine code after the release to avoid breaking modification and focused on modding support instead of releasing new games. The modding support was key to the success of the games created by means of the Source engine. Since its release, Valve has been providing the Source SDK for modders [5, 36]. Using the source engine for mods is free as long as the mods are distributed for free. However, selling a game or mod requires license fees for tools that are included in the engine [1]. The Source SDK can be modified and redistributed freely, which allows the creation of custom editors [14]. Due to the Half-Life series's modding friendliness, a wide variety of mods has been created. They range from comprehensive game conversions like the already mentioned Counter-Strike title over ports of titles that were originally created by means of different engines like Team Fortress (originally Quake Engine) and more recent conversion mods like Black Mesa to small gameplay changes. One of these changes was to allow Half-Life players to carry a weapon and a flashlight simultaneously. To our knowledge, this mod is the only example of balancing in-world realism (using material likely found in the game's environment) and immersion as it makes the game less scary [21, 24, 36].

Unity is innately not as modding friendly as the other engines on this list. Therefore, third-party developers have developed and published assets on Unity's Asset Store that make modding easier. These are usually frameworks that

have to be added to the game in advance and then allow players to load mods during gameplay [27]. If developers decide not to use any of these frameworks, they have to come up with their own system to load and integrate mods. This requires different handling for different types of data. While Unity can read JSON files and other files containing text, developers must explicitly handle more complex data such as 3D meshes [27]. Without the use of additional frameworks, Unity's AssetBundle API can be helpful here. If the game contains such AssetBundles, these can replaced by AssetBundles mods [27]. In terms of scripting, some of the frameworks allow loading C# or Lua scripts. Doing so at runtime is usually not supported by Unity since all code is compiled and bundled before the game is published. As a result, it is harder to implement support for script mods without additional modding frameworks [27]. Therefore, any modding implementation for a game made with Unity requires detailed documentation on how to implement different features in a mod [27]. The aforementioned title Cities: Skylines [27] is a representative example of modding support by a Unity-made game. It support User Interface Changes in terms of using other assets and level designs as well as Game Conversion Mods in terms of goal and rules of play. The latter mods are introduced based on C# scripts. These scripts are compiled when launching the game by an included compiler. To support User Interface Changes, the game offers a set of editors to create maps, themes, assets, and scenarios. These editors run inside the game and do not rely on the Unity editor. Cities: Skylines has a dedicated forum for this topic and detailed documentation for the coding part in a wiki section to help developers create mods [3, 4]. Unity's terms of service strictly prohibit any modification to the engine or redistribution thereof. However, creating mod editors inside games or modifying games using licensed versions of Unity is permitted. Any game or mod created with Unity entirely belongs to the developer and can, therefore, be distributed and sold without being regulated by the license [8, 16, 17].

4.1 Comparison

To allow for a more direct comparison between the engines' modding capabilities, we aggregated relevant aspects in Table 1.

We identified the relevant criteria listed in the table during the course of our investigations for the following reasons. *Reusability/variety of games* immediately relates to the game engine gamut presented in Figure 1—it lays the (soft) boundaries of tackled game designs and, thus, also determines the scope of potential modding efforts. We consider it a valuable insight whether an engine is built on a positive *modding philosophy, or modding friendliness*, since mods may benefit from all the services provided by a game engine, even those that we could not clearly identify or those that might not have played a role in modding, yet. Another reasons why the modding philosophy behind an engine should be considered lies in aspects such as a sustainable life-cycle. The concrete *documentation and infrastructure* an engine provides for modding is the first hands-on criterion that affords modding or moddable game design. Next, its supported *programming language(s)* may hinder or support modders in unfolding their creative ideas. Domain-specific scripting languages may be less performant and expressive than hardware-oriented languages such as C/C++ but often come with a flatter learning curve and faster development cycles. Visual scripting adds another layer of abstraction and makes programming more accessible still. These aspects of coding accessibility are important as modding differs from development in its focus on design of game contents/mechanics, which would benefit from simple and quick changes of the codebase, if necessary. These in-code changes might not only affect the codebases of specific games but potentially target the *engine* they are built on, or its *editor(s)*. There are numerous reasons this might be beneficial to modding at large such as the introducing shader programs to realize new graphical styles, a spatial sound engine to extend the spectrum of sensory cues in a FPS game, or simply to provide convenience functions for various modding goals.

Feature \ Engine	id Tech	Unreal	Source	Unity
Reusability/Variety of games	Mostly FPS ("Doom clones") [2, 18]	Early games: FPS Today: rather broad [2]	Generic Shooter and FPS [2]	Very broad w.r.t. genre and device [2]
Modding philosophy/ friendliness	Active modding support [25]	Active modding support ("Designed with modding in mind") [10, 25]	Modding SDK and editors available since early versions; Active support for Half-Life mods [36]	No built-in modding support; third-party frameworks exist [27]
Modding documentation and infrastructure	Source code of versions 1-4 [32]	Forums, tutorials, engine documentation, and engine source code [10]	Engine documentation, engine source code, and wiki [6, 11]	Documentation of engine and third-party modding frameworks [9, 27]
Programming languages	id Tech 1-3: C/QuakeC; Starting with id Tech 3: C++ [18, 25, 32]	C++/Blueprints (visual scripting) [10]	C++ [6]	C#, Javascript, Boo, and Lua (dep. on ext. framework) [25, 27]
Engine/editor customization	Recent versions no longer open-source [18, 32]	Engine mods as custom editors [10]	SDK may be modified and redistributed [14]	Engine mods not permitted; editor plugins for modding feasible [14, 27]

Table 1. Comparison of Modding Opportunities of Popular Game Engines

The difference between Unity and the other three game engines is quickly noticable. While id Tech engines, Unreal Engine, and the Source engine all allow for easy modding and even endorse developers to create mods with tutorials and documentation, Unity does not actively support modding. However, it possible to create mods for Unity games. Another noteworthy point for modders is the difference in programming languages used. Aside from Unity, the listed Engines are all based on C or C++, which is are generally considered challenging languages due to the direct control the programmer has to exercise at a system level, e.g. considering memory management, but – for the same reason – also very powerful and efficient. Despite the advantages, inexperienced programmers might prefer modding Unity games, use its recently introduced visual scripting editor, or consider Unreal Engine's visual scripting blueprints. All four engines could be used for mods that do not pertain to coding. However, especially with Unity, the developers of the original titles must invest into planning and implementing ways to access and exchange assets. Reducing the necessary investments quite a bit, the other engines provide comprehensive resources and built-in tools to do so. Custom mod editors for specific games can be built for all engines. In the case of the older id Tech engines, the Unreal Engine, and the Source engine, these editors can be modifications of the engines themselves. For Unity, however, custom editors must be created as part of the game or as a standalone application.

5 DISCUSSION & GUIDELINES

Modding support of game engines varies a lot. While some engines provide everything needed for modding, others set modding aside. These differences can be interpreted in multiple ways. Like the Unreal Engine, some engines

that were created when modding emerged, have maintained their support. Only the id Tech engine series dropped its support, due to an overall change to exclusivity. Another observation we can make in the context of the four engines we have investigated is that less versatile engines are more likely to support modding (Table 1). This might be due to the fact that the resulting mods are also more limited and support to modding endeavors is, therefore, easier to implement. Another reason could be that engines that thrive in genre-specific niches depend on the favor of the respective gaming communities. Looking at the enthusiasm for specific games and the creative potential modders exhibit, their influence in these communities might be similarly great. Vice versa, in community-driven genres, specialized engines might thrive in the first place and, therefore, naturally support the overall success and popularity of the respective genres, including modding. Section 2.4 and section 4.1 allow us to infer some concrete guidelines for developers aiming at moddable games. In general providing modding support has a positive impact on games and game development, so long as the required initial invest is feasible in the scope of a given game development project. In the case of competitive games, modding should be limited to maintain a fair gameplay. Therefore, as a first step, a developer (person or studio) should make an informed decision about whether the envisioned title should be turned into a moddable game or not (the decision steps are summarized in Figure 2). Based on our preceding elaborations, there are strong reasons to support modding such as aiming at a specific genre such as MOBA or fps, establishing or serving a gaming community, or if the game is not so much defined through existing contents but by means of its potentials, see for instance Garry's Mod (section 2.5). Next, there are reasons that apply to games and game development in general such as the improvement of the game's quality, maintenance, extensibility, as well as the opportunity to benefit from user-generated contents and prolonged periods of sales. If it is determined that modding would be beneficial and feasible for a given project, an analysis about modding types should follow. The overarching categories of User Interface Changes, Game Conversion Mods, or Machinimas offer numerous opportunities for modding whose potential impact and associated feasibility/costs should inform the developer about prioritizing his efforts. If the project is already tied to a specific game engine due to preceding development efforts, established codebases or game design pipelines, the developer needs to investigate about the best possible support options—relying on built-in support of the engine or resorting to extensions/plugins or additional stand-alone applications. Alternatively, based on its strategic importance, modding support should be considered a major factor for deciding on a specific game engine. In this context, developers need to be

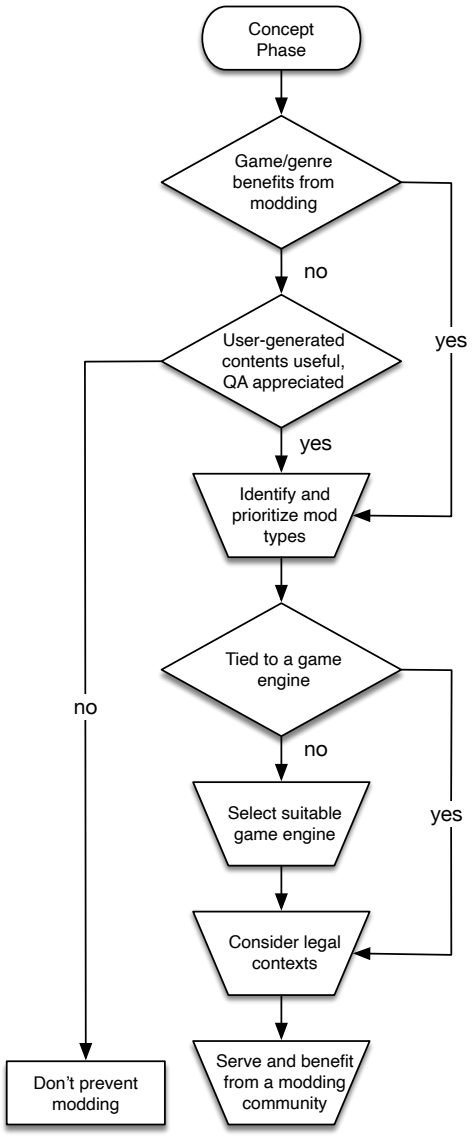

Fig. 2. Guided steps towards moddable games.

careful when selecting an engine with respect to its legal implications. Their licenses may be subject to change, which has been the case oftentimes in the past, or might enforce different rules for developers from different countries. Along these lines, licenses of any other shared intellectual property, including sound or graphics assets, also need to be considered.

Even if a considerable modding community has grown around a game, its developer should not intentionally offload work to modders. Rather, he should attempt to act as an important, professional service partner of the community, which provides services in return for the modders' efforts and all the players' financial investments. An exit strategy could always be to leave a formerly flourishing project to its community, if it does not pay for itself any longer. Even games that do not officially support modding can attract modders' interest. Unless the respective mods are malicious, bias competitive play or pursue illegal methods of data processing or asset use, developers should not try to lock them out. As one implication, the developers should consider the legislative context, understand their legal responsibilities and act accordingly in order to ensure sustainable benefits.

6 CONCLUSIONS

Based on an informal definition and categorization of modding (User Interface Changes, Game Conversion Mods, and Machinimas), we briefly elaborated about its impact in terms of a game's development and play. We presented the popular mods Garry's Mod, DOTA 2, and Counter-Strike to underline our deliberations. Next, we emphasized the link between game engines and modding, resorting to the game engine gamut by Gregory [25] and adapting it to stress the increased ease-of-use to design games as opposed to develop the required codebases. We introduced four well-established and widely known game engines (the id Tech engines, Unreal Engine, the Source engine, and Unity) to link them to modding and to present five aspects of game engines that determine modding support at different levels (codebase, editor, asset pipelines,...) and at different stages of the development life-cycle of a game. We could show that modding support differs significantly and, therefore, advise developers to consider this important aspect early on when entering the concept phase of the game development process.

Previous studies already underlined the positive effects of modding [28, 29]. We did not add new insights in this regard. However, our taxonomical rationale for modding support in combination with a systematic analysis of several state-of-the-art game engines provides an informative perspective for developers of games and game engines alike. Both, game engines and modding, have been subject to research since the early 2000s, especially since mods like Counter-Strike became popular among players. However, this research is usually conducted separately from each other, and therefore the modding support of individual game engines is rarely considered in scientific contexts. As shown in section 2 and section 3, the history of both is strongly linked and hence should be examined in more detail. The Source engine especially requires more research as it plays an important role in the history of modded games, but is hardly mentioned in the scientific discourse. Of course, our limited analysis of four game engines should, of course, be extended to ensure the correctness of the inferred insights and guidelines, as well as to identify additional supportive criteria or hindrances for modding. Another important direction of potential future research is the connection between the used game engine and a mod's success since section 2.5 shows that many successful mods are based on games created by means of the Source engine.

REFERENCES

[1] [n.d.]. Distributing Source Engine Games (Steamworks Documentation). https://partner.steamgames.com/doc/sdk/uploading/distributing_source_engine

[2] [n.d.]. Internet game database. https://www.igdb.com/

[3] [n.d.]. Modding - Cities: Skylines Wiki. https://skylines.paradoxwikis.com/Modding

[4] [n.d.]. Modding API - Cities: Skylines Wiki. https://skylines.paradoxwikis.com/Modding_API

[5] [n.d.]. SDK Docs. https://developer.valvesoftware.com/wiki/SDK_Docs

[6] [n.d.]. Source. https://developer.valvesoftware.com/wiki/Source

[7] [n.d.]. Tools and Editors. https://docs.unrealengine.com/en-US/Basics/ToolsAndEditors/index.html

[8] [n.d.]. Unity & modding legal questions. https://forum.unity.com/threads/unity-modding-legal-questions.255624/

[9] [n.d.]. Unity User Manual. https://docs.unity3d.com/Manual/

[10] [n.d.]. Unreal Engine: Frequently Asked Questions. https://www.unrealengine.com/en-US/faq

[11] [n.d.]. Valve Developer Community. https://developer.valvesoftware.com/

[12] 2009. DotA & LoL. https://web.archive.org/web/20091027011423/http://www.leagueoflegends.com/articles/dota_and_lol Archived version.

[13] 2010. Official DotA: Frequently Asked Questions. https://web.archive.org/web/20101111183559/http://www.getdota.com/faq Archived version.

[14] 2013. Source SDK License. https://github.com/ValveSoftware/source-sdk-2013/blob/master/LICENSE

[15] 2018. Bus Simulator 18 Editor. https://www.epicgames.com/store/de/product/bus-sim-18/home

[16] 2020. Unity Software Additional Terms. https://unity3d.com/legal/terms-of-service/software

[17] 2021. Unity Terms of Service. https://unity3d.com/legal/terms-of-service

[18] Gavin Annand. 2020. The History of the id Tech Engine. https://fat-studios.medium.com/the-history-of-the-id-tech-engine-4dbf7c70ef5b

[19] Thilo Bayer. 2010. 14 years of Quake Engine: The famous games with id Technology. https://www.pcgameshardware.de/Spiele-Thema-239104/News/14-years-of-Quake-Engine-The-famous-games-with-id-Technology-687947/

[20] Benjamin Beil. 2014. *Modding / Leveleditoren / Editor-Games Skripte und Praktiken digitaler Partizipation.* Biermann, Ralf and Fromme, Johannes and Verständig, Dan, 207–232. https://doi.org/10.1007/978-3-658-01793-4_10

[21] Erik Champion. 2012. *Introduction: Mod Mod Glorious Mod.* ETC Press, Pittsburgh, PA, USA, 9–26.

[22] Eleftheria Christopoulou and Stelios Xinogalos. 2017. Overview and Comparative Analysis of Game Engines for Desktop and Mobile Devices. *International Journal of Serious Games* 4, 4 (12 2017). https://doi.org/10.17083/ijsg.v4i4.194

[23] Kevin R. Conway. 2012. *Game Mods, Engines and Architecture.* ETC Press, Pittsburgh, PA, USA, 87–112.

[24] Greg Finch. 2011. The Top 10 Game Mods Of All Time. https://www.vice.com/en/article/4x45mp/the-top-10-game-mods-of-all-time

[25] Jason Gregory. 2018. *Game engine architecture* (3rd edition ed.). CRC Press.

[26] Friedrich Kirschner. 2012. *From Games to Movies: Machinima and Modifications.* ETC Press, Pittsburgh, PA, USA, 149–166.

[27] Sean Kumar. 2015. Creating A Moddable Unity Game. https://www.turiyaware.com/blog/creating-a-moddable-unity-game

[28] Daniel Lee, Dayi Lin, Cor-Paul Bezemer, and Ahmed E. Hassan. 2020. Building the Perfect Game - An Empirical Study of Game Modifications. *Empirical Software Engineering* 25 (07 2020). https://doi.org/10.1007/s10664-019-09783-w

[29] Daniel Lee, Gopi Krishnan Rajbahadur, Dayi Lin, Mohammed Sayagh, Cor-Paul Bezemer, and Ahmed E. Hassan. 2020. An Empirical Study of the Characteristics of Popular Minecraft Mods. *Empirical Software Engineering* (09 2020). https://doi.org/10.1007/s10664-020-09840-9

[30] Loke. 2019. Modders aren't supposed to fix bugs. https://lokeloski.medium.com/modders-arent-supposed-to-fix-bugs-76e0ea66cf00

[31] Ilya Makarov, Dmitry Savostyanov, Boris Litvyakov, and Dmitry I. Ignatov. 2018. Predicting Winning Team and Probabilistic Ratings in "Dota 2" and "Counter-Strike: Global Offensive" Video Games. In *Analysis of Images, Social Networks and Texts*, Wil M.P. van der Aalst, Dmitry I. Ignatov, Michael Khachay, Sergei O. Kuznetsov, Victor Lempitsky, Irina A. Lomazova, Natalia Loukachevitch, Amedeo Napoli, Alexander Panchenko, Panos M. Pardalos, Andrey V. Savchenko, and Stanley Wasserman (Eds.). Springer International Publishing, Cham, 183–196.

[32] Muhammad Saqib. 2020. id Tech - Series of Game Engines written in C/C++. https://www.mycplus.com/featured-articles/id-tech-game-engines/

[33] Walt Scacchi. 2010. Computer game mods, modders, modding, and the mod scene. *First Monday* 15, 5 (05 2010). https://doi.org/10.5210/fm.v15i5.2965

[34] Mike Stubbs. 2019. The incredible rise of Dota. https://www.redbull.com/int-en/the-history-of-dota

[35] Tim Sweeney. 2015. Unreal Engine is Now Free! https://www.unrealengine.com/en-US/blog/ue4-is-free

[36] Michael Thomsen. 2012. Ode to Source: A History of Valve's Tireless Game Engine. https://www.ign.com/articles/2009/09/22/ode-to-source-a-history-of-valves-tireless-game-engine

[37] Jens Wiemken. 2003. Phänomen Computerspiele: Counter-Strike. *spielbar.de* (2003). https://www.spielbar.de/fachartikel/145757/phaenomen-bildschirmspiele-counter-strike

[38] Nick Wingfield. 2016. Unity Technologies, Maker of Pokémon Go Engine, Swells in Value. https://www.nytimes.com/2016/07/14/technology/unity-technologies-maker-of-pokemon-go-engine-swells-in-value.html

LUDOGRAPHY

[39] Bethesda Game Studios. 2011. The Elder Scrolls V: Skyrim. PEGI 18, ESRB Mature.

[40] Blizzard Entertainment. 1998. Starcraft. PEGI 16, ESRB Teen.
[41] Blizzard Entertainment. 2002. Warcraft III: Reign of Chaos. PEGI 12, ESRB Teen.
[42] Colossal Order. 2015. Cities: Skylines. PEGI 3, ESRB Everyone.
[43] Epic Games. 1998. Unreal. ESRB Mature.
[44] Facepunch Studios. 2004. Garry's Mod.
[45] General Computer Corporation. 1982. Ms. Pac-Man. PEGI 7, ESRB Everyone.
[46] id Software. 1993. DOOM. PEGI 16, ESRB Mature.
[47] id Software. 1996. Quake. ESRB Mature.
[48] id Software. 1999. Quake III Arena. ESRB Mature.
[49] Mojang Studios. 2011. Minecraft. PEGI 7.
[50] Niantic Labs. 2016. Pokémon GO. PEGI 3, ESRB Everyone 10+.
[51] Potomac Computer Systems. 1991. ZZT.
[52] Riot Games. 2009. League of Legends. PEGI 12, ESRB Teen.
[53] Douglas E. Smith. 1983. Lode Runner. Rerelease: PEGI 3.
[54] Steve Russell. 1962. Spacewar!
[55] Valve. 1998. Half-Life. PEGI 16.
[56] Valve. 2000. Counter-Strike. PEGI 16, some sequels PEGI 18.
[57] Valve. 2004. Half-Life 2. PEGI 16, ESRB Mature.
[58] Valve. 2007. Team Fortress 2. PEGI 16, ESRB Mature 17+.
[59] Valve. 2013. DOTA 2. ESRB Teen.