CECILIA: A Toolkit for Visual Game Content Exploration and Modification

Philipp Fleck[1], Michael Hochörtler[1], David Kastl[1], Georg Gotschier[1], Johanna Pirker[1] and Dieter Schmalstieg[2]
[1]Graz University of Technology, [2]University of Stuttgart

***Abstract*—We investigate the idea of a toolkit for visually exploring and modifying game content, addressing questions such as how to identify relevant in-game data, how to make use of the data to create in-game visual representations, and what benefits these representations have. To that aim, we build a toolkit on top of the .NET platform employed by Unity in order to explore and add custom content without access to the game's source code. Our visual modifications use live objects in the game as data sources. The results appear as an integral part of the game world, which is generated with the original Unity rendering engine. This capability enables visual exploration for debugging, playtesting, modding, streaming, and data-driven analysis of games, as we demonstrate with several examples.**

***Index Terms*—Content exploration, Modifications, Accessibility**

I. Introduction

IT is a known fact that online games acquire extensive amounts of telemetry data created by users while playing [1]. The collected data is used for analytics procedures which aim to improve the gameplay experience. Use cases include operations (*e.g.,* server load balancing in massive online games) and maintenance (*e.g.,* detecting and fixing errors in the game code), but also monetization [2] (for example, marketing research for in-game shops).

In addition to developers and publishers, other stakeholders have important interests in using in-game data for various purposes. Players may want to maximize their experience or training success. Modders create game extensions in a community effort. Journalists – including bloggers, streamers, and so on – report on game events, such as e-sports competitions, and reach out to spectators (who may or may not be players themselves). In addition, the huge success of computer games as a mass media has sparked the interest of the research community. For example, behavioral science [3], [4] and sociology [5] have discovered games and gamers as a domain worth investigating. Serious games and gamification [6] are considered important topics in education research today.

All of these stakeholders have specific needs regarding the addition of new functions to an existing game. Many of these needs involve displaying game-related information in a timely and efficient manner. For this purpose, extending the standard visual presentation of a game with additional *data visualizations* [7] is often the method of choice.

Visual game content exploration and modification have several compelling use cases throughout the lifecycle of a game project. For example, developers can benefit from rapid prototyping tools for game development, especially for in-game debugging and playtesting. Modders may develop visual modifications that make the player's life easier (for example, accessibility features for the visually impaired) or match the player's personal preferences. Journalists may use such tools to create extended spectator views for live e-sports coverage or for analytic investigation in retrospect. Researchers can instrument games for their investigation of player behavior and social computing.

Many genres of games, such as strategic or business simulations (see Figure 2), already make use of data visualization as part of their in-game interface. However, these data visualizations are hard-coded into the game and usually cannot be customized beyond the game designer's intention.

Some online games expose their network streams in a known format [8], which can be used to visualize game-related data outside of the actual game. Unfortunately, this approach is limited to the information sent over the network and does not lend itself to "immersive analytics"[1] [9] using visual representations directly embedded (and therefore implicitly linked to the activities) in the game.

A more powerful method is the use of libraries for game modifications, or *mods*. Games that are "moddable" expose an application programmer interface (API) to load and execute extensions. By modding, visual modifications can be developed and deployed on demand, as long as they conform to the mod API.

Unfortunately, relatively few games include mod support [10]. In contrast, other software categories, such as office products, commonly include end-user-friendly scripting capabilities that require little or no coding skills to customize or extend the features of the software product. We speculate that end-users would benefit from a standardized mod API for games, especially, if it allows the creation of immersive visualizations directly in the game.

In this paper, we investigate this idea further. As a case study, we select a leading game engine, *Unity*, which is reported to have an estimated market share of approximately 37% [11]–[14], with more than 750.000 games published [15]. We use the case study to address the key questions in creating visual modifications: (1) How can we explore relevant in-game data? (2) How can we make use of the in-game data to create in-game visual modifications? (3) What are the benefits of visual modifications?

[1]The term "immersive analytics" is used loosely here, since the games we consider mostly show 3D worlds on a conventional flat screen and not in virtual reality.

Fig. 1. Enhanced versions of games installed from Steam without access to source code: (A) an extension of the game *Subnautica* with an added game mode and visualizations for the tracking of creatures, (B) original version without visual modifications, (C) high-contrast accessibility mode added to the game *Risk of Rain 2* (modified version on the right, original game on the left).

Our system, called CECILIA (an acronym for "*combining existing common intermediate language* with *immersive analytics*"), addresses the first two questions using runtime code modification (so-called " weaving") to generate visual modifications in Unity games without mod support or access[2] to source code.

By weaving into the .NET platform employed by Unity [16], visual modifications can be conveniently created with our authoring tools that have been integrated into the Unity editor. With this toolkit, we demonstrate how to support the entire workflow that a mod developer must address. In summary, CECILIA makes the following contributions:

- It supports game content exploration and identifying relevant game data.
- It enables the creation of visual modification to be displayed in the game scene.
- It facilitates interactive control over the visual modification and its properties.

We demonstrate our work with a variety of use cases encompassing enhancements for debugging, gameplay and accessibility, and we report on a qualitative user study conducted with Unity developers.

II. Related work

A good portion of human-computer interaction research in the last 30 years has focused to some extent on games and game-related interactions [17]. Games are used not only to investigate player behavior, but can also serve as a vehicle for other research, for example, on medical conditions such as Parkinson's disease [18]. Among the four major paradigms [19] on how games are used in human-computer interaction research – operative, epistemological, ontological, and practice –, CECILIA contributes primarily to operative and practice research, since it allows studies that involve games that are otherwise unattainable (*i.e.,* they lack mod support). For example, CECILIA can help add spectator modes [20] or improve usability for users with special needs [21]. Without CECILIA, a game must have built-in mod support to create such extensions.

A. Game analytics

Without doubt, video games generate a lot of data that is of value to various stakeholders. Unfortunately, games often make it difficult to collect data for third-party research endeavors. Most of the data is collected by the original game developers using proprietary game mining tools [22]. The main motivation to use data mining is to look for an optimal monetization strategy [2].

Other ways of collecting data are surveys, playtesting [23] or gathering publicly accessible data such as replay files [8]. Games with mod support may also be instrumented to collect specific data of interest. This approach is sometimes taken by researchers who want to study the social phenomena around games [24] or even train an artificial intelligence [8] to play.

For example, Bauckhage *et al.* [25] use the collected data to investigate the loss of player interest in games over time. Drachen *et al.* [26] analyze player activity, causes of death and player frustration. Mertens [27] describes the reasons why games are often released unfinished. Other topics of interest include the search for an optimal game development process [28] or modding practices [29], [30]. Wallner and Kriglstein [24] provide a summary of gameplay analysis research.

In all these cases, the data are collected first and then analyzed with software tools which are external to the original game. Such an approach fits a pure analytic tasks, but is less appropriate for other use cases, such as e-sports broadcasts or game accessibility, where the main focus lies on the live game. These and other uses cases clearly benefit from presenting visualizations directly as oart of the game world.

B. Game mods

The community formed around popular games often has a strong interest in developing game modifications. Popular game engines with good mod support are the *Unreal* engine [31] and the *Source* engine [32]. In contrast, the widely used *Unity* engine [33] does not have built-in mod support, making modding more difficult [34].

[2]Please see Section VII for a discussion of dual use purposes of game modifications

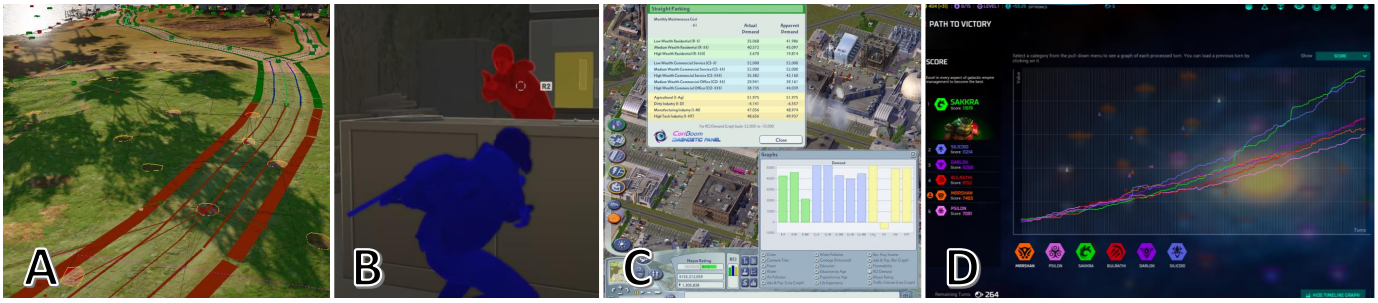

Fig. 2. Examplatory visual modifications in games across genres: (A) Debug of *Assissin's Creed Origin*; (B) Accessibility enhances of *The Last of Us, Part II*; (C) Citizen's demands visualized as bar charts in *SimCity 4*; (D) Path to Victory line chart visualization in *Master of Orion: Conquer the Stars*.

According to Scacchi [35], mods can be categorized into four different types: *User interface customizations* enhance the user's experience of the game, for example, with additional convenience functions or stylistic changes. *Game conversions* alter the game experience by adding completely new content, such as new maps or even new game modes. *Machinima* mods are built to record and replay game sessions, for documentation, dissemination, or even for artistic expression. Finally, *hacking* mods alter the system around the game, such as to bypass anti-cheating measures or to enable modding in games that do not support it out of the box. We classify CECILIA as a combination of hacking mod and user interface customization, where the former enables the latter.

Although modding a game shares many characteristics with debugging it, modders are rarely given access to the game's source code and must contend with the capabilities exposed through a game's modding API or log files recorded of game runs [36]. The sheer complexity of a game project can make it difficult to reproduce a bug and collect data about it [37], even if a source code-level debugger is available. To make the mental demands of developing/debugging a visual mod more manageable [38], CECILIA offers a dedicated workflow to support content exploration. As explained in Section III-A, its features are based on DLL manipulation. Other tools like BepInEx[3]are technically similar in their code manipulation abilities, but have no means for exploring the actual game content.

C. Accessibility Mods

A visual accessibility mode can be considered a special kind of visual modification for users with low vision or color blindness. Very few commercial games offer such features, *e.g.,* configurable text size or high-contrast rendering, as can be inferred by checking databases from organizations such as the *Family Gaming Database* [39]. Yuan *et al.* and Westin *et al.* discuss the low number of games featuring accessibility modes [40] and point to new concepts [41] in game accessibility. Fortes *et al.* [41] summarize the evaluation methods used in game accessibility. Miesenberger *et al.* [42] elaborate on the importance of game accessibility for inclusion. The gap between games released without accessibility features and the part of the gaming community in need is significant. CECILIA can help fill the accessibility gap, as we show in the results section. Our accessibility examples are inspired by Mangiron *et al.* [43] and Garber *et al.* [44].

[3]https://github.com/BepInEx/BepInEx

D. In-game visualizations

If we want to analyze game data immediately, for example, during e-sports broadcasts or for game debugging, displaying data visualizations directly inside the live game is the most effective approach. The survey by Bowman *et al.* [7] discusses several data visualization archetypes that are frequently used in games to support players. They point out that players tend to have a high visual literacy and benefit from visual analytics tools in complex games, such as economic simulations. Dillman *et al.* [45] describe a taxonomy of data visualizations in games that essentially implement "augmented reality" (AR) overlays in the game world, serving as examples of how AR designs could work in the real world.

Researchers and practitioners of in-game visualizations can benefit from the wide variety of extensions available for Unity, which include several toolkits for generating 3D data visualizations inside Unity. DXR [46] uses a grammar of graphics to express visualizations of tabular data. The result of the visual encoding process is stored as conventional game objects in Unity. IATK [47] and U2VIS [48] take a similar approach. IATK achieves the best performance by using GPU shaders instead of static game objects to render the visualization.

Despite such software support, in-game visualizations other than those built in for playing the game (see Figure 2) are rare or non-existent. This situation must be expected since games are mainly designed to entertain the players. However, alternative uses of games, for example in research, can hugely benefit from more advanced visualization. With CECILIA, our aim is to fill this gap.

III. Overview

CECILIA covers three tasks (Figure 3):

- T1 *Content exploration*: Assuming access to the data stored in the game objects, the exploration step comprises *identifying* the relevant in-game content and preparing it for later use.
- T2 *Visual modification creation*: The desired views (typically, new game objects containing data visualizations

or additional rendering layers) must be defined and connected to the data sources identified in T1 for later display during game runtime.

T3 *Runtime control*: The desired views must be instantiated and their parameters adjusted in response to new game events and user input, while the game is running.

A developer-friendly interface is provided by integrating the tools for all three tasks into the Unity IDE. For the creation of visual modifications, the tasks T1-T3 are commonly carried out in sequence. For other types of modifications, a single task may be sufficient (for example, T1 for recording). In the following, we focus on the main goal of creating visual modifications.

A. Content exploration

Content exploration (T1), *i.e.,* identifying which game objects or properties to use in visual modifications, can be challenging. CECILIA must provide sufficiently powerful abilities for handling this exploration. In practice, tools similar to those offered by a conventional visual debugger are required. The exploration phase starts the game with CECILIA loaded, but without adding any custom visual modifications yet. In this phase, the developer's objective is to become familiar with the game objects and learn enough about the internal mechanics of a game to move on to the visual encoding phase. The content explorer of CECILIA is a kind of runtime visual debugger that enables the inspection of game objects much like in the Unity IDE itself, but without access to its source code (see Section IV for more details). Figure 5 shows an example of the debugging interface. The main features of the content explorer are the following:

a) Scene graph and inspector view: A graphical overlay shows the current scene graph and the properties of game objects in a manner very similar to that of Unity IDE. The scene graph view lists the currently selected game objects; the inspector view enables property modification at runtime.

b) Debug console: The debug console makes it possible to display Unity's log messages in the compiled game. The messages are familiar to Unity developers from within the IDE, but are not normally accessible outside of the IDE.

c) Pause game: The game can be paused by setting the game time scale to zero. In pause mode, the scene and its game objects can be inspected, visual modifications can be instantiated, etc. Games relying the standard *Time* object in Unity will pause as expected. For games that do not use standard time, other mechanisms, such as hooks into the game loop, can be used.

d) Debug camera: This feature lets the user freely control a debug camera with mouse and keyboard to view the game scene from an arbitrary vantage point. It is best used after pausing the game, so that the modified camera does not interfere with other game functions.

e) Game object selection: While the debug camera is active, the user may select arbitrary game objects in the scene by clicking, provided that the objects contain collider components. Selecting an object in the game view will also mark it as selected in the scene graph hierarchy and disclose its internals in the inspector view. To visually indicate selected game objects, they are retrofitted with a halo-like outline upon selection. Moreover, selections can be saved and loaded to better support extensive debugging sessions.

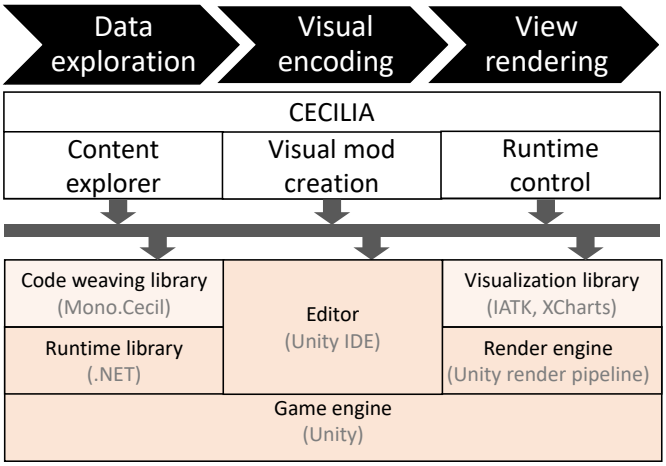

Fig. 3. System overview: The workflow of CECILIA (top row) consists of three stages: content exploration, visual modification creation and runtime control. These components are built on the standard elements of Unity (orange) and as plug-ins (light orange) for code modification (Mono.Cecil) and visual encoding (IATK, XCharts).

f) Data recorder: Changes to property values of game objects can be recorded and saved in a database for later analysis or external processing. Filters can be used to identify the desired data source selected for recording. In addition, the recording frequency and interval must be specified. Technically, co-routines are spawned to sample selected data-streams (*e.g.,* position of the player) concurrently and at different rates. This might come at the cost of scalability, depending on the hardware.

B. Creation of visual modifications

CECILIA integrates several ways to create and render visual modifications in Unity (T2). It supports 2D and 3D data visualizations as well as general-purpose rendering extensions, such as adding new layers to Unity's layered rendering system.

For 2D data visualizations, we integrate XCharts [49], a chart library for Unity. XCharts allows for the creation of interactive, real-time updateable data visualizations. If no integration with the 3D scene structure is required (*i.e.,* the data about the game to be visualized is of abstract nature, such as general performance measurements), XCharts provides a fast way of creating data visualizations.

For 3D data visualizations, we extend IATK, which lets the user specify a set of 3D graphical marks from one or more data sources. The generation process uses a grammar of graphics, which expresses a series of transformations from raw data to graphical objects. Each transformation consists of a view-frame in which to place the graphical output, a geometric object to be generated, and mapping from the properties of the input data in the graphical properties of the output object. By stacking multiple such transformations, complex and compound data visualizations can be built. IATK comes

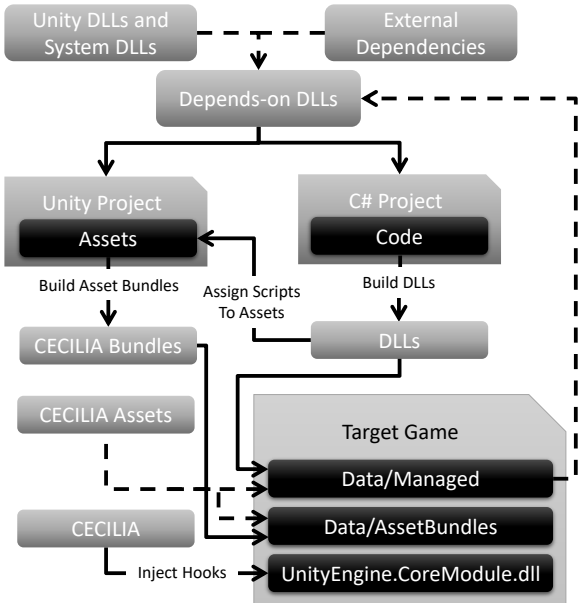

Fig. 4. Workflow of adding visual modifications to an existing game.

with a library of common data visualization patterns, such as bar charts, scatter plots, parallel coordinates, and visual links. Moreover, it enables interactions with the data visualization, such as linking and brushing, for the selection of data points.

For all data visualization, CECILIA supports far-reaching customization of parameters while the game is running. A data visualization is connected to one or more game objects for raw data input, but also refers to additional user-defined parameters for customization of appearance and styling (*e.g.,* line width or color gradients). For this purpose, every data visualization can be equipped with a *customization interface*.

The customization interface may be created in the Unity IDE and then added together with its host data visualization into the target game. However, in most cases, it suffices to just expose a list of customization parameters for the user to change in an inspector-style window. For this purpose, we have developed a utility class that takes a list of properties of the data visualization prefab, which should be exposed in the customization interface. The utility class relies on reflection to identify the properties referred to in the list and builds a corresponding customization interface at runtime.

Finally, for general-purpose visual modifications, we allow bypassing XCharts and IATK entirely and using custom rendering primitives and shader code instead. Such code may be useful for gameplay modifications that are not directly data-driven (such as in-game cinematics or accessibility interfaces).

C. Runtime control

To instantiate a visual modification at runtime, we provide a runtime control interface (T3). The interface allows the user to manage the lifecycle of visual modification. Visual modifications can be created, connected to game objects, modified via their customization interface, and finally deleted.

All visual modifications prepared in the previous step are stored as Unity prefabs, so they can easily be inserted as new

TABLE I
SUMMARY OF UNITY FEATURES USED BY CECILIA.

Feature	Description
GameObject	Entity in the virtual environment
Scene	Scene graph of the virtual environment
Prefab	Stored hierarchy of Game Objects
Assets	Scenes, game objects, scripts, etc.
Asset bundle	Assets compiled into a file
Assembly	CIL code in a dynamic link library (DLL) file

game objects at runtime. To establish the connection between a newly instantiated visual modification and its target game, the user must specify one or more game objects as parameters, either to attach visual modification to the game object, or to track game object properties, or both. For this purpose, we retrieve and display the hierarchy of game objects in the scene from the Unity runtime system. After the user selects an object, we intercept the resulting event and connect visual modification to the chosen object.

D. Example workflow

Let us look at an example of a moving game entity (*e.g.,* the karts described in Section V, for which we want to display a trail indicating the entity's recent movement. We add the CECILIA toolkit to the game, explore the game content with the CECILIA debug UI, and identify the relevant game object. The debug UI supports selecting the character by clicking on the screen or, alternatively, filtering game objects by type or name. Filtering can be performed interactively in the debug UI, but also programmatically at runtime (see Figure 9) to connect visual modifications to dynamic collections of game objects. Once a game object is identified, reflection is used to reveal the various components and attributes of the game object. We select the position element of the transform component of the character for our visualization.

We set up an IATK visualization displaying a poly-line with a fixed number of segments. Adding a visualization requires writing a few lines of C# code that instantiates a prefab contained in the CECILIA toolkit. The vertices that define the polyline are connected to the entity's position. The vertex array is configured as a fixed-size buffer. When a new vertex is added, the oldest one is discarded. We connect the vertex to the game character so a new vertex is added whenever the character's position changes. The poly-line is drawn relative to the character's feet, slightly above ground. It is rendered with depth buffering enabled, so that the trail appears as a decal over the ground, but it properly occluded by other objects or higher terrain.

The resulting modification consists of additional files that contain the toolkit and the bespoke modification. These modifications are stored in a set of files, and the original Unity application is modified to load these files at startup (see Section IV-B for more details). In the modified application, the visual modification is now available and can be invoked automatically or on demand.

Fig. 5. *Magic the Gathering: Arena* was extended using CECILIA. (A) The unaltered game is depicted from the player's perspective. (B) The debug UI allows for game object manipulation on the fly. The blue arrows indicate the manipulated colors of the cards. (C) A hidden test menu is presented, which was dormant in the game and activated by enabling the corresponding game object within the hierarchy. In (B) and (C), we altered the camera to a different angle than used in the original game.

IV. Implementation

The previous section has outlined the goals and workflow of CECILIA from a user interface perspective. In this section, we explain the technical process for creating visual modifications (Figure 4). CECILIA injects code modifications into existing Unity applications. The injection leverages the fact that Unity relies on the *common intermediate language* (CIL), a form of byte-code used by .NET or Mono, which has a known structure. The original application code is modified so that the CECILIA toolkit is automatically loaded, and event handlers are added that trigger the creation of custom visual modifications as in-game objects.

We also considered modifying native C/C++ libraries [50], which would potentially allow us to target other engines and platforms (see also the discussion in Section VII). Android and other mobile platforms rely on `IL2CPP`[4] compilation, which produces native C/C++ libraries. Decompilers like `CPP2IL`[5] revert native libraries back to managed ones that are accessible by CECILIA. However, dealing with native libraries is more difficult, and we considered the wide adoption of Unity sufficient for the purposes of our study.

We start this section with an overview of Unity's software architecture. Then, we explain how to modify Unity assemblies and how to prepare asset bundles in the required code format. We conclude the section by reviewing the aspects of our user interface for code modifications.

A. Unity software architecture

Unity applications use an entity-component design pattern [51], [52]. This pattern follows a paradigm of composition over inheritance and is also known as *composite reuse principle* (not to be confused with the entity-component-system pattern, or ECS). Each entity, a so-called *game object*, aggregates one or more *components* which define its behavior. The game objects populate a three-dimensional environment, the *scene*, represented as a hierarchical scene graph. Game objects typically consist not only of components, but refer to other game objects to build hierarchical structures. Recurring game object hierarchies are defined as *prefabs*. When creating new game objects, their behavior is defined using *scripts* written in C#, which call the native API functions of Unity. Scripts are compiled into assemblies using *Mono* [16] or .NET in newer versions. Moreover, game *assets*, such as game objects, images, geometric models, materials, etc., are organized into *asset bundles* and stored in a proprietary file format. Scripts are compiled into assemblies, which can be shipped together with asset bundles. Table I summarizes the features used by CECILIA.

B. Code modification

"Weaving" is the process of modifying the code of an existing application that is given as CIL code. Tools such as Postsharp or Mono.Cecil are commonly used in large software applications to add patches or enforce save-guards, such as range checks. However, existing weaving tools only support the low-level mechanics of code modification, which makes them hardly suitable for the creative process of generating visual modifications.

To support a convenient workflow, CECILIA wraps the weaving mechanics in a simple mechanism. Recall that the code of .NET/Mono applications is kept in assemblies, while application data is kept in asset bundles. A visual modification typically consists of new code and new game content, which are stored in a new assembly file and asset bundle, respectively. In addition, we must also add code and content of the CECILIA library itself to the game. Both can be achieved by extending the list of files that the original application loads upon startup. Moreover, existing code must be modified to invoke the visual modification automatically or on demand. For that purpose, we intercept certain standard procedures, such as instantiations of a given class or handling of a given event, and patch them to invoke the CECILIA functions.

The steps to generate the new and modified files are fully automated, based on the choices the modder made in the earlier steps of working with CECILIA (exploring the game content, defining the visual modification, connecting visual modification to the game content). When the development of visual modification is complete, it can be injected with a single click for testing in the live game. The CECILIA code itself has negligible impact on the performance of the target game. However, the custom code introduced to create a custom visualization or other game modification might

[4]https://docs.unity3d.com/6000.1/Documentation/Manual/scripting-backends-il2cpp.html

[5]https://github.com/SamboyCoding/Cpp2IL

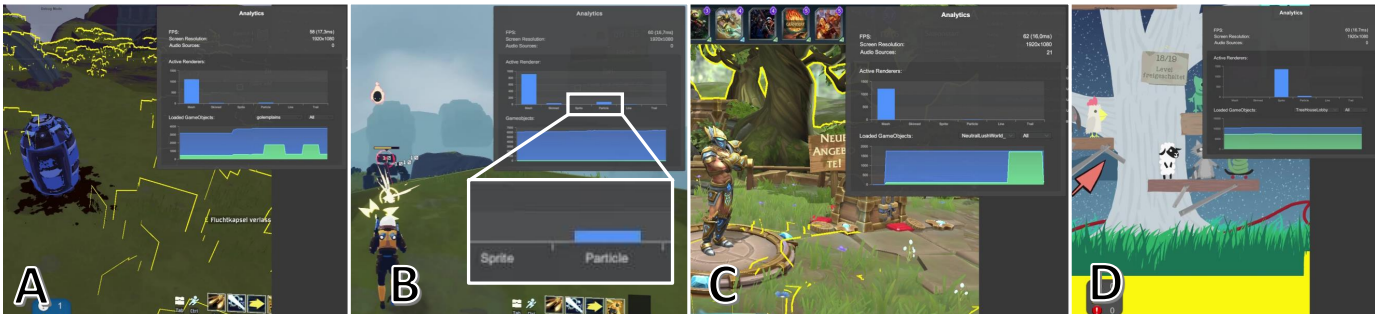

Fig. 6. Game analytics package adding the analysis overlay in dark gray: (A) shows Risk of Rain 2 in a very peaceful moment with some renders being active and approx. 6000 game objects are being loaded. The yellow outlines show the selected *grass* game object. (B) continues in Risk of Rain 2, but with an enemy shooting at the player causing more mesh renderers and particle renders to spawn. (C) Minon Masters with active mesh shaders, and (D) Ultimate Chicken Horse, a 2D Jump'n'Run platform mainly using sprites for its visuals.

affect the game performance. While we did not observe any performance problems with the modifications described in this paper, introducing heavy computations could potentially slow down the target game or affect its internal timing.

C. User interface of the code modification process

To support the process described above, we have automated the process and added code modification tools accessible through the Unity IDE. Repurposing the IDE for visual modification creation primarily provides casual developers with a familiar environment; of course, it does not make the source code of the target game available. However, even without source code, the tools added to the IDE make the weaving procedure rather straightforward: (1) Check the compatibility of the target game with CECILIA. (2) Import CECILIA packages into an empty Unity project. (3) Set up target asset bundle and assembly definition. (4) Select "build and deploy" and pick the target game executable. (5) Test the modified game. Please see the accompanying video for a live demonstration of the process.

V. Use cases

To test our work, we initially crawled Steam for popular games made with Unity and identified 235 for which we owned licenses. About half of these games could not be used due to versioning conflicts. Minor version conflicts can be resolved with a Unity update. Major version changes ($\geq$2017) should also work, though they might require more adaptation. We lacked the time to resolve these conflicts and instead concentrated on the over 100 games which were instantly compatible with CECILIA. From these games, we selected 17 games, covering a wide selection of genres, for use with CECILIA (Table II). In the remainder of this section, we discuss a variety of use cases and visual styles.

A. Debugging menus in games

In-game visualizations are particularly interesting to illustrate and investigate potential flaws in game design and game mechanics. As discussed in section III-A, CECILIA provides a runtime debugger to inspect Unity game objects in a manner similar to the Unity IDE. The scene graph is presented in a hierarchy view. With this option, one can select in-game objects, inspect their properties and change them (*e.g.,* size of an object, visibility, physics). Furthermore, game objects can be manipulated. Figure 5B shows the use of the debugging menu in *Magic the Gathering Arena* to change the color of the card border to pink, while Figure 5C shows a hidden test menu that we enabled. In both views, the default position of the camera has also been changed.

TABLE II
List of games tested that were with CECILIA. More information about a game can be found on the Steam website via https://store.steampowered.com/app/⟨STEAMID⟩.

Game Title	Steam ID	Year
Dinkum	1062520	2022
Enter the Gungeon	31169	2016
Getting Over It with Bennett Foddy	240720	2017
Human: Fall Flat	477160	2016
Magic the Gathering Arena	2141910	2022
Minion Masters	489520	2019
Pummel Party	880940	2018
Rain World	312520	2017
Risk of Rain 2	632360	2020
Shellshock Live	326460	2020
Seven Days to Die	251570	2009
Sixty Seconds! Reatomized	012880	2019
Slime Rancher	433340	2020
Subnautica	264710	2018
Unturned	304930	2017
Ultimate Chicken Horse	386940	2016
Valheim	892970	2021

B. In-game analytics

The built-in profiler and debugger built into the Unity IDE assume access to the application's source code. Therefore, profiling and debugging a compiled game (the "release version") is not possible with the Unity IDE.

The weaving approach of CECILIA enables profiling and debugging even when the source code is not available. We can retrofit profiling functionality after deployment. We relied on XCharts to create complex 2D charts shown as floating overlays. For example, we can use standard game engine features to inject missing functions, such as a frame-per-second meter, into every game.

We use C# reflection to filter game objects based on their components and compute their statistics for visual modifi-

Fig. 7. Visual modifications added to the Kart game. (A) Navigation visualization showing the route on the road and a directional arrow. (B) The bumper visualization depicts the distance to the surroundings, similar to the parking sensor visualizations in real cars. (C) Trajectories as trail behind karts, where the acceleration is color-coded and (D) the trajectory and acceleration data of the kart are visualized as 2D screen overlay.

cations. In this example, we implemented a trivial graphics performance analyzer that reports the variety of rendering objects used (mesh, sprite, particle, etc.) and the number of currently active game objects. We took care to only use basic Unity functions, so the performance meter is compatible with any game supported by CECILIA. Figure 6 shows a number of games with the game analytics package enabled and additionally shows the current resolution and frames per second (FPS).

C. Prebuilt visualizations

To demonstrate the use of prebuilt data visualizations, we modified a driving game, which allows the user to steer a taxi and transport passengers across a nightly city. The game itself is a total conversion of the "Kart" sample game shipped with Unity, where a procedurally generated city as well as ambient traffic in the form of randomly navigating non-player cars has been added.

a) Distance visualization: The distance visualization provides a collision warning (Figure 7B). It casts rays outward from a circle around a game object and intersects them with the surrounding terrain. The polyline resulting from connecting the ray intersections with the terrain is shown as colored lines emitted radially from the game object. The color of the lines is determined using a color gradient with the distance of the terrain hit point as an argument. Driving too close to the terrain turns the line from green to red.

b) Trajectory visualization: The trajectory visualization shows a visual representation of the path that a game object has taken through the environment, in the form of a polyline. The trajectory is created at a given frequency (using a redraw interval) or by adding a new data point whenever the object moves further than a threshold. Multiple three-dimensional trajectory visualizations can be created at once, as shown in Figure 7C, where every kart has its own trajectory.

c) Radar visualization: The radar visualization attaches to a game object and draws the objects it tracks as spheres. For specifying the tracked objects, we use the same filters as for the trajectory visualization. Multiple distinct sets of objects can have different colors and sizes.

d) Navigation visualization: We use the data on the procedurally generated city to provide "Google maps"-style navigation assistant. The calculated route is directly overlaid on the streets of the game world to mimic an "augmented reality" cockpit. When a passenger is picked up or an address is entered in the customization interface, a route from the current location in the city to the destination is computed using standard path finding (A*) algorithm. The route is drawn using line primitives, as shown in Figure 7A. The path is updated when a new route is requested or when the player drives too far from the path.

D. Gameplay mods

We used Subnautica, an underwater adventure game on a remote planet with alien marine life, to demonstrate the creation of a mod that extends the gameplay. In Subnautica, many different fish-like creatures live underwater and have different behaviors, swimming routines, interactions, and personalities. The main target of the game is to unwind the story, restore the space ship and escape the planet. What the original game does not offer is the ability to analyze the creatures' behaviors and their species diversity to get a better understanding of the planet from the point of view of a behavioral scientist. The aim of our extension is to see all species and to track their behavior. We contribute the following enhancements, which seamlessly integrate with the game mechanics.

a) In-game radar: The radar extends the player's HUD to easily visualize the creatures nearby (Figure 8, f). By tracking the position of creatures (Figure 9), we can filter them by distance and feed them to IATK. With IATK, we render a bird-eye view with the player at the center. The north direction of the radar aligns with the viewing direction of the player.

b) Observation lab: The lab consists of four screens and a positional tracker. Screen 1 (Figure 8, b) shows the selected species, where unseen species are blacked out, and seen species are rendered with their natural appearance. Screen 2 (Figure 8, c) shows all individuals of the selected species and allows the player to scroll through and select one. Screen 3 (Figure 8, d) shows a third-person surveillance view of the selected creature. Technically, we add a virtual camera that follows the selected creatures. Screen 4 (Figure 8, e) depicts the depth of the selected creature in real time using IATK for graph visualization. The position tracker is a 3D visualization of the trajectories of creates of a species, showing the player as a red dot and the selected creature as a yellow dot (Figure 8C). We retrieve species by reflection and add a position tracker to report positions to IATK.

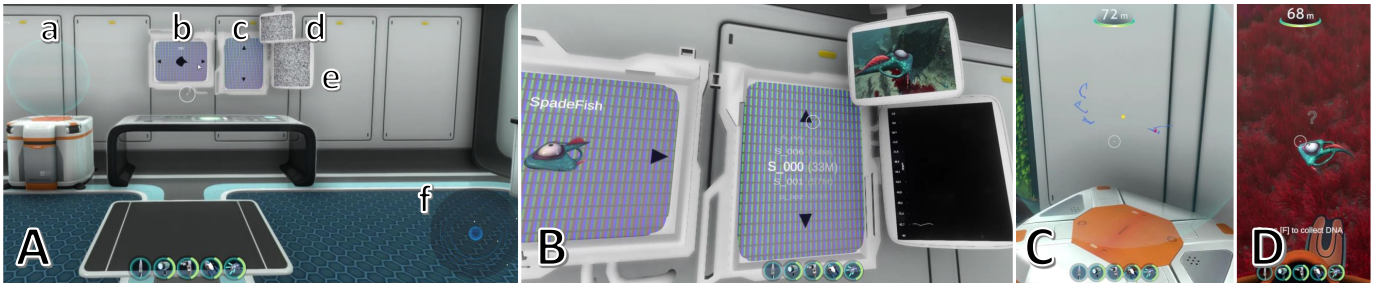

Fig. 8. *Enhanced* Subnautica: (A) New exploration area with in-game 3D assets: (a) positional tracker showing the trajectories of selected creatures, (b) species selection screen, (c) creature selection within a species, (d) surveillance view of a chosen creature, (e) visualization of the current depth as a line chart using IATK, (f) bird-eye radar aligned with the player to find creatures also using IATK. (B) Screens (b-e) visualize data of a tracked creature. (C) Positional tracker in action. (D) Question mark glyph above an unseen species .

```
var entity_tags = RuntimeUtility.FindComponentsOfType(RuntimeTypes.EntityTag);

foreach (var entity_tag in entity_tags)
{
  object slotType = entity_tag.GetMemberValue("slotType");

  if (slotType.ToString() == "Creature")
  {
    _creatures.Add(entity_tag);
  }
}
```

Fig. 9. Runtime evaluation of an type search added to the game Subnautica to retrieve all instantiated creatures. The resulting list `_creatures` can further be used to access and track the found creatures.

c) Enhanced creatures: A question mark is attached above unseen species (Figure 8, D). Once observed, the question mark is replaced by the species name.

E. Accessibility mods

Pummel Party is a tabletop game with embedded minigames. In this fast-paced game, it is often difficult to keep track of the player characters. Furthermore, such games usually do not come with any accessibility feature. By implementing a high-contrast mode and following color-palette suggestions for easy readability, we add a fully integrated accessibility mode. Technically, we use Unity's layer system, where the player characters are rendered onto a specific layer. We can produce a stencil texture for a screen-space overlay to isolate the players by adding a virtual camera to render only the players. Figure 10 shows the resulting gray background overlay and green player overlay. The implementation can be reused in any other game using the layer system.

Risk of Rain 2 is a third-person sci-fi shooter which lets the player battle against increasing amounts of monsters on a remote planet. The game is very fast paced, with a lot of jumping, shooting and running, naturally creating a very busy environment. The main objective was to apply the package created with Pummel Party to Risk of Rain 2 and to test how well it works. Alas, Risk of Rain 2 does not use Unity's layer system for rendering.

Therefore, we implemented a slightly slower alternative to the layer approach. It uses C# reflection to filter for game objects that contain certain components. To make the filtering independent of in-game behavior, we add an additional pass which renders the meshes of filtered objects into a stencil buffer; the latter is used as before in Pummel Party to create the screen space overlays. We adopt a colorblindness-friendly palette with gray background, the player in green, enemies in purple and interactive objects in yellow (Figure 10B, C). The only game-specific information required is the list of game objects that participate in the stencil creation.

Slime Rancher is a first-person adventure game in which the player needs to catch slimy creatures inhabiting the planet, collect their slime and make money to expand the slime ranch. Since the game is already vibrantly colored and uses a lot of small text [53], we integrate a text-to-speech processor, UAP [54]. We apply our filtering to find text widgets and attach the text-to-speech processor to render the text using the host device's standard synthetic voice. The only specific information required to apply this form of accessibility support is the game object that holds the text to be rendered.

VI. Developer Evaluation

The purpose of the study presented in this section was to obtain an informal first assessment of the usability of our toolkit in terms of workflow and ease of use. The participating experts (game developers) had to perform a set of tasks using CECILIA, with the help of a how-to video that demonstrated the required workflow. A questionnaire was presented before (Table III) and after (Table IV) the task (Table V).

A. Procedure

We investigated the suitability of CECILIA for modifying games with little insight into their workings. For that purpose, we recruited four software developers (one female, three male) from the authors' personal network to complete two tasks using CECILIA. All developers had experience with game development in Unity. We started by taking a demographic questionnaire and asking questions about the experience of the participants and the background of the developer as stated in Table III. Then we showed the participants a five-minute how-to video showcasing working with CECILIA. Participants used a dual-screen setup, with the video playing on one screen, while they worked along starting from an empty Unity project. The participants could stop the video at any time and were allowed to ask additional questions to the experimenter.

Fig. 10. High contrast accessibility modes using different techniques: (A) Pummel Party, depiction of the original game on the left and our accessibility version with the players highlighted in green on the right, (B) a high-contrast mode similar to (A) for *Risk of Rain 2*, (C) an action scene in *Risk of Rain 2* with color projectiles (purple). Both implementations utilize Unity's layer system for (A) and a secondary render pass for (B) and (C). In both cases, we render stencils on the overlays.

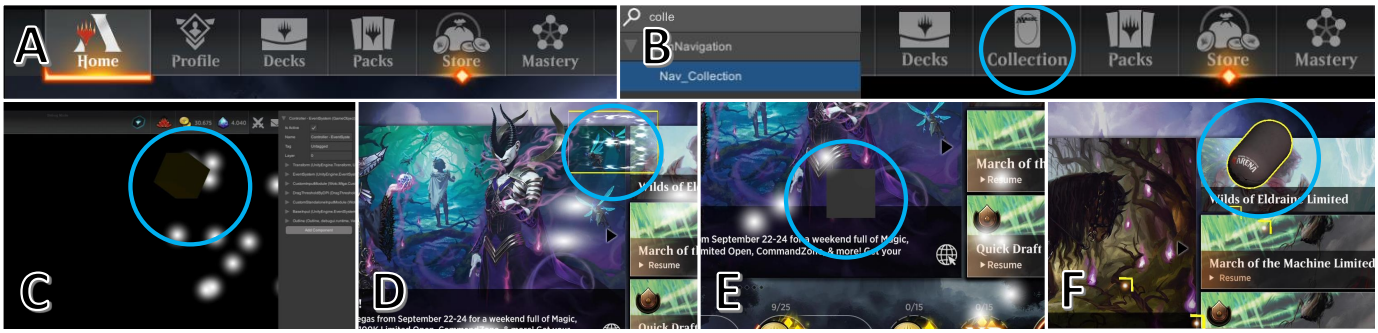

Fig. 11. Evaluation on Magic the Gathering: Arena: (A, B) before/after the comparison of A1, where the *Collection* element on the nav-bar has been enabled (circle in B), (C-F) outcome of A2 for participants D1 to D4, (D) animated in-game texture on the cube, (F) in-game texture on the capsule.

B. Tasks

After viewing the how-to video, participants were asked to perform two tasks, A1 and A2 (Table V). After looking into a number of games, we selected the main menu of *Magic the Gathering: Arena* (Figure 11). We found the menu to be a good target for novice users, since it is visually rich, but without confusing the developer with too much action. Any changes made by the developer could be immediately observed without the need to load a specific level or game situation.

Task A1 allowed participants to experience the debugging UI and code modification. The key operation was the use of the search function in the hierarchy view to find the element `Nav_Collection` (a hidden menu) and activate it. This task was intended to familiarize the developer with the UI and its search and scene-hierarchy capabilities, reminiscent of an exploration step aimed at finding a specific game component.

Task A2 was concerned with introducing new visual elements into the game. For that purpose, a prefab containing a cube had to be created and set up correctly with a particle system, and a material had to be created and applied to the cube. Both elements had to be added to the asset bundle and configured for automatic loading.

Finally, we administered another questionnaire about the workflow and conducted an open discussion about the experiences of the participants. At the end of the study, participants had to answer a final questionnaire about the workflow.

TABLE III
DEVELOPER STUDY INITIAL QUESTIONNAIRE

Q01	Are you familiar with the Unity Game Engine?
Q02	Are you familiar with prefabs in Unity?
Q03	Are you familiar with scripts in Unity?
Q04	Are you familiar with Unity's AssetBundles?
Q05	Are you familiar with how to create AssetBundles?
Q06	Are you familiar with how to load AssetBundles at runtime?
Q07	Are you familiar with scenes in Unity?
Q08	Are you familiar with scene loading in Unity?
Q09	How many years of experience do you have in developing or coding?
Q10	How many years of experience do you have using a game engine?
Q11	How many years of experience do you have in Unity?
Q12	How many years of experience do you have in C#?
Q13	How often do you use unity?
Q14	Have you compiled your own library?
Q15	Are you aware of managed and native libraries/dll?
Q16	Have you heard of code modification / weaving?
Q17	Have you heard of reflection in C#?
Q18	Are you familiar with mono/.net?
Q19	Are you familiar with assembly definition (asmdef) files in Unity?
Q20	Do you use AssetBundles in your projects?

C. Results

(Q1-8) All participants were developers and researchers (aged 26-33) and had intermediate or expert knowledge of Unity, covering all essential functions for handling scenes, game objects, asset bundles, deployment and release, and debugging. **(Q9-13)** Participant D1 has four years of experience in C# and Unity, 15 years of general development expertise, and uses Unity once every two months. Participant D2 has five years of experience in C# and Unity, overall 10 years of development expertise, and uses Unity on a monthly basis. Participant D3 has seven years of experience in 3D engines, one year in Unity, and one year in C#. Furthermore, D3 has

TABLE IV
DEVELOPER STUDY FOLLOW-UP QUESTIONNAIRE

F01	Was the workflow video helpful?
F02	Did you find the workflow video complicated?
F03	Did you find the workflow simple?
F04	Do you consider regular use of Cecilia with the presented workflow?
F05	How or for what would you use Cecilia?
F06	What would be a specific application?
F07	What did you like about the workflow?
F08	What did you dislike about the workflow?

TABLE V
DEVELOPER STUDY TASKS

A1	Enable the inactive *collection* menu within the navigation bar of the game *Magic the Gathering: Arena*.
A2	Add a colored cube or other 3D object emitting particles to the game *Magic the Gathering: Arena*.

14 years of development experience, uses Unity daily, and is selling a Unity-based game on Steam with more than 30K downloads. Participant D4 has six years of Unity and C# experience, ten years of general development expertise, and uses Unity weekly for virtual reality applications. **(Q14-15)** D1 and D2 work on mobile applications and make extensive use of Unity's cross-platform capabilities. **(Q16-18)** All participants had heard of code modification and are familiar with C# reflection. Only D3 and D4 have a deeper understanding of Unity's rendering pipelines. **(Q19-20)** None of the participants used assembly definition files, but D3 and D4 used asset bundles sporadically.

All participants conducted the study in less than an hour and completed both tasks successfully (Figure 11). D1 and D2 had some questions about asset bundles and materials, while D3 and D4 were very confident, drawing on their extensive knowledge of Unity. One pitfall was a mismatch between the standard rendering pipeline used in Magic, which conflicted with the universal rendering pipeline of the Unity 2020.3 project we provided. This conflict caused a failure to load the default shader, which was easily fixed by D3 and D4, while D1 and D2 needed additional hints.

All participants successfully built and deployed CECILIA into the game. **(F1-3)** The how-to video was found to be more than sufficient for the first use; several participants called it "easy" or "straightforward". All participants said that they were impressed by how easy it was to add content to a game or alter its internals. Participant D1 was noticeably excited when he first saw the sparkling cube in the game (Figure 11D). **(F4-6)** All participants would consider using CECILIA in their current project, *i.e.,* to post-patch features or fixes (D1-D2) or to investigate the attack surface of the commercial application of D3. **(F7-8)** After finishing, D1 explicitly mentioned the ease of use of CECILIA and the workflow presented.

One participant mentioned that it was a fast process to add something to a game. Another participant suggested testing CECILIA with a virtual reality game as potential future work. Participant D4 suggested a feature to automatically add created content to the asset bundle after forgetting to do so on the first try. We noted that the usability of CECILIA has some room for improvement, but no major problems were observed during the study.

Despite the fact that our study has to be taken with a grain of salt due to its very limited size, we did not uncover any pitfalls, and our expert participants were instantly capable of executing a basic code modification workflow with CECILIA. We are aware that novice users of Unity might face a harder time, but this must be expected as a general result of Unity's learning curve. Additionally, our how-to video was deemed helpful as a reference while performing the tasks.

VII. DISCUSSION AND CONCLUSIONS

We contribute the design and implementation of a toolkit to enhance Unity's game builds with in-game visual modifications. It allows users with a wide variety of backgrounds, such as quality assurance, modders, or game moderators, to better understand the game content and in-game behaviors. This paper demonstrates the workflow by modifying the compiled source code and how to add different forms of visual modifications. We present different use cases, including in-game debugging and analytics, visual and gameplay mods, and retrofitted accessibility modes.

Although we only had a small number of participants in our developer-centric evaluation, we could observe promising responses. Developers with Unity knowledge were surprised how easy and fast it was to get custom content in a game. The developer study took around 45 minutes from the first keystroke to successfully adding the content. In comparison, the use cases presented in Section V took us between 20 and 40 work hours to build. The time invested was primarily spent exploring the structure and mechanics of the game, while the actual process of adding the visual modifications takes little time once the right elements of the game are identified.

While the current version is designed for the game engine Unity, we plan to extend this in future work to other leading game engines, like Unreal [31], CryEngine[6] or Lumberyard[7]. Both Unreal and CryEngine have extensions[8] that support the .NET framework; Lumberyard supports the Python language, which uses an intermediate language that could be used for an approach similar to the one taken by CECILIA with .NET.

The current implementation of CECILIA is tailored to Unity's control flow. However, supporting the control flow of other engines is possible, although it would require some amount of software engineering: The commonly used game engines employ similar concepts, such as hierarchical scene graphs and entity-component patterns. On a conceptual level, the workflow of CECILIA – data exploration, then visual encoding and view rendering – remains largely the same across engines; only the implementation details are engine-specific.

While CECILIA is currently designed for experienced developers, we plan to further enhance the usability of the toolkit to support wider audiences. Especially retrofitting desirable

[6]https://www.crytek.com/cryengine
[7]https://aws.amazon.com/de/lumberyard
[8]https://github.com/nxrighthere/UnrealCLR,
https://devblogs.microsoft.com/dotnet/choose-a-net-game-engine

standard features, such as performance monitoring or accessibility modes, to existing games with minimal effort (or even in a fully automated manner) would be a worthwhile endeavor.

Like many advanced digital tools, the potential of CECILIA for dual use raises ethical questions. Manipulating the work of others without permission is an ethically gray area, where the interpretation strongly depends on the use case and the status of the user. It must be assumed that adding mods can interfere with the author's intention.

Clearly, publishing modified games violates the terms of service in most cases and can have legal consequences. In contrast, applying in-game changes for personal use (*e.g.,* enhancing the playability of the game) is ethically less objectionable and permissible in many legislations, provided one owns a valid license. Some use cases may be covered by the "fair use doctrine" covering, among others, scientific, educational or preservationist goals. The move to online games makes the topic of game modifications even more complex: While certain use cases, such as e-sports, crucially depend on added features such as game streaming, other game mods (*e.g.,* the famous "wall hack" letting players in shooter games make walls transparent to see opponents early) destroy fairness in games. We assume that most competitive online games have built-in defenses against unfair mods. CECILIA uses a rather straightforward weaving approach and does not provide any special features that could be used to bypass anti-modding saveguards built into games.

We see the main area of application of CECILIA in personal uses (such as accessibility) and in games research, the latter being the original motivation for developing CECILIA. Consequently, we will make CECILIA available to the community upon paper acceptance.

ACKNOWLEDGMENTS

The authors thank Alexander Plopksi for looking at countless games. Furthermore, we thank our study participants for taking the time to participate in the developer evaluation and the Alexander von Humboldt Foundation funded by the German Federal Ministry of Education and Research. This research was funded in whole, or in part, by the Austrian Science Fund (FWF) [10.55776/I5912]. For the purpose of open access, the author has applied a CC BY public copyright licence to any Author Accepted Manuscript version arising from this submission.

REFERENCES

[1] Emerging-India-Analytics, "10 ways: How is data analysis used in video games?" 2024. [Online]. Available: https://medium.com/@analyticsemergingindia/10-ways-how-is-data-analysis-used-in-video-games-b5db42337b1d

[2] J. Babb and N. Terry, "Comparing video game sales by gaming platform," *Southwestern Economic Review*, vol. 40, pp. 25–46, 2013.

[3] N. Männikkö, H. Ruotsalainen, J. Miettunen, H. M. Pontes, and M. Kääriäinen, "Problematic gaming behaviour and health-related outcomes: A systematic review and meta-analysis," *Journal of Health Psychology*, vol. 25, no. 1, pp. 67–81, 2020.

[4] L. A. Kort-Butler, "Gamers on gaming: a research note comparing behaviors and beliefs of gamers, video game players, and non-players," *Sociological Inquiry*, vol. 91, no. 4, pp. 962–982, 2021.

[5] V. C. Gazis, *Video gaming: The sociology of a lifeworld*. University of Exeter (United Kingdom), 2012.

[6] R. Dörner, S. Göbel, W. Effelsberg, and J. Wiemeyer, *Serious Games – Foundations, Concepts and Practice*. Springer, 2016.

[7] B. Bowman, N. Elmqvist, and T. Jankun-Kelly, "Toward visualization for games: Theory, design space, and patterns," *IEEE Trans. Vis. Comp. Graph.*, vol. 18, no. 11, pp. 1956–1968, 2012.

[8] J.-L. Hsieh and C.-T. Sun, "Building a player strategy model by analyzing replays of real-time strategy games," in *2008 IEEE International Joint Conference on Neural Networks (IEEE World Congress on Computational Intelligence)*. IEEE, 2008, pp. 3106–3111.

[9] K. Marriott, F. Schreiber, T. Dwyer, K. Klein, N. Henry Riche, T. Itoh, W. Stuerzlinger, and B. H. Thomas, *Immersive Analytics*, ser. Lecture Notes in Computer Science. Springer International Publishing, 2018.

[10] NexusMods, "Landing page," 2024. [Online]. Available: https://www.nexusmods.com/games

[11] A. E. Soupé, "Game engines market shares on steam (all time, 2022, 2021)," https://www.reddit.com/r/dataisbeautiful/comments/10vcj5e/oc_game_engines_market_shares_on_steam_all_time/, 2022, accessed: 2024-03-26.

[12] D. Lars and P. Anthony, "Game engines on steam: The definitive breakdown," https://www.gamedeveloper.com/business/game-engines-on-steam-the-definitive-breakdown, 2022, accessed: 2024-03-26.

[13] "Steam instant search," https://steamdb.info/instantsearch/, 2022, accessed: 2024-03-26.

[14] S. Team, "Did you know that 60% of game developers use game engines?" https://www.slashdata.co/post/did-you-know-that-60-of-game-developers-use-game-engines, 2022, accessed: 2024-11-26.

[15] R. Wallace, "Gaming poised to continue accelerated growth according to unity gaming report 2022," https://unity.com/our-company/newsroom/gaming-poised-continue-accelerated-growth-according-unity-gaming-report-2022, 2022, accessed: 2024-03-26.

[16] M. Menard and B. Wagstaff, "Game development with unity®, second edition," 2015. [Online]. Available: https://searcher.com.br/download.html/Game%20Development%20with%20Unity%202nd%20Edition%20Book%20of%202015%20Year.pdf

[17] K. Hornbæk, A. Mottelson, J. Knibbe, and D. Vogel, "What do we mean by "interaction"? an analysis of 35 years of chi," *ACM Transactions on Computer-Human Interaction (TOCHI)*, vol. 26, no. 4, pp. 1–30, 2019.

[18] P. J. Bank, M. A. Cidota, P. Ouwehand, and S. G. Lukosch, "Patient-tailored augmented reality games for assessing upper extremity motor impairments in parkinson's disease and stroke," *Journal of medical systems*, vol. 42, pp. 1–11, 2018.

[19] M. Carter, J. Downs, B. Nansen, M. Harrop, and M. Gibbs, "Paradigms of games research in hci: a review of 10 years of research at chi," in *Proceedings of the first ACM SIGCHI annual symposium on Computer-human interaction in play*, 2014, pp. 27–36.

[20] S. Kriglstein, G. Wallner, S. Charleer, K. Gerling, P. Mirza-Babaei, S. Schirra, and M. Tscheligi, "Be part of it: Spectator experience in gaming and esports," in *Extended Abstracts of the 2020 CHI Conference on Human Factors in Computing Systems*, 2020, pp. 1–7.

[21] H. Desurvire and C. Wiberg, "Game usability heuristics (play) for evaluating and designing better games: The next iteration," in *Online Communities and Social Computing: Third International Conference, OCSC 2009, Held as Part of HCI International 2009, San Diego, CA, USA, July 19-24, 2009. Proceedings 3*. Springer, 2009, pp. 557–566.

[22] M. S. El-Nasr, A. Drachen, and A. Canossa, "Telemetry collection and tools," in *Game Analytics: Maximizing the Value of Player Data*. Springer, 2013, pp. 83–201.

[23] A. de Jongh, "Playtesting: Avoiding evil data," 2017. [Online]. Available: https://www.gdcvault.com/play/1024132/Playtesting-Avoiding-Evil

[24] G. Wallner and S. Kriglstein, "Visualization-based analysis of gameplay data – a review of literature," *Entertainment Computing*, vol. 4, 8 2013.

[25] C. Bauckhage, K. Kersting, R. Sifa, C. Thurau, A. Drachen, and A. Canossa, "How players lose interest in playing a game: An empirical study based on distributions of total playing times," in *IEEE conference on computational intelligence and games*. IEEE, 2012, pp. 139–146.

[26] A. Drachen, A. Canossa, and J. R. M. Sørensen, "Gameplay metrics in game user research: Examples from the trenches," in *Game Analytics: Maximizing the Value of Player Data*. Springer, 2013, pp. 285–319.

[27] J. Mertens, "Broken games and the perpetual update culture: Revising failure with ubisoft's assassin's creed unity," *Games and Culture*, vol. 17, no. 1, pp. 70–88, 2022.

[28] R. Ramadan and Y. Widyani, "Game development life cycle guidelines," in *2013 International Conference on Advanced Computer Science and Information Systems (ICACSIS)*. IEEE, 2013, pp. 95–100.

[29] L. Poretski and O. Arazy, "Placing value on community co-creations: A study of a video game'modding'community," in *Proceedings of the 2017 ACM Conference on Computer Supported Cooperative Work and Social Computing*, 2017, pp. 480–491.

[30] H. Postigo, "Modding to the big leagues: Exploring the space between modders and the game industry," *First Monday*, 2010.
[31] E. Games, "Unreal engine," https://www.unrealengine.com/, 2022, accessed: 2022-04-28.
[32] V. Software, "Source engine," https://developer.valvesoftware.com/wiki/Source, 2022, accessed: 2022-04-28.
[33] U. Technologies, "Unity engine," https://unity.com/, 2022.
[34] L. Schreiner and S. von Mammen, "Modding support of game engines," in *The 16th International Conference on the Foundations of Digital Games (FDG) 2021*, 2021, pp. 1–9.
[35] W. Scacchi, "Modding as a basis for developing game systems," in *Proceedings of the 1st international workshop on Games and software engineering*, 2011, pp. 5–8.
[36] D. Spinellis, "Debuggers and logging frameworks," *IEEE software*, vol. 23, no. 3, pp. 98–99, 2006.
[37] "Building the world of assassin's creed origins," https://80.lv/articles/building-the-world-of-assassins-creed-origins/, 2018, accessed: 2022-05-08.
[38] S. Xu and V. Rajlich, "Cognitive process during program debugging," in *Proceedings of the Third IEEE International Conference on Cognitive Informatics, 2004.*, 2004, pp. 176–182.
[39] Family-Gaming-Database, "Landing page," 2024. [Online]. Available: https://www.familygamingdatabase.com/
[40] B. Yuan, E. Folmer, and F. C. Harris, "Game accessibility: a survey," *Universal Access in the information Society*, vol. 10, pp. 81–100, 2011.
[41] R. P. M. Fortes, A. de Lima Salgado, F. de Souza Santos, L. Agostini do Amaral, and E. A. Nogueira da Silva, "Game accessibility evaluation methods: a literature survey," in *Universal Access in Human–Computer Interaction. Design and Development Approaches and Methods: 11th International Conference*. Springer, 2017, pp. 182–192.
[42] K. Miesenberger, R. Ossmann, D. Archambault, G. Searle, and A. Holzinger, "More than just a game: accessibility in computer games," in *4th Symposium of the Workgroup Human-Computer Interaction and Usability Engineering of the Austrian Computer Society*. Springer, 2008, pp. 247–260.
[43] C. Mangiron and X. Zhang, "Game accessibility for the blind: Current overview and the potential application of audio description as the way forward," *Researching audio description*, pp. 75–95, 2016.
[44] L. Garber, "Game accessibility: enabling everyone to play," *Computer*, vol. 46, no. 06, pp. 14–18, 2013.
[45] K. R. Dillman, T. T. H. Mok, A. Tang, L. Oehlberg, and A. Mitchell, "A visual interaction cue framework from video game environments for augmented reality," in *ACM CHI*, April 2018.
[46] R. Sicat, J. Li, J. Choi, M. Cordeil, W. K. Jeong, B. Bach, and H. Pfister, "Dxr: A toolkit for building immersive data visualizations," *IEEE Transactions on Visualization and Computer Graphics*, vol. 25, pp. 715–725, 1 2019.
[47] M. Cordeil, A. Cunningham, B. Bach, C. Hurter, B. H. Thomas, K. Marriott, and T. Dwyer, "Iatk: An immersive analytics toolkit," 2019, pp. 200–209.
[48] P. Reipschlager, T. Flemisch, and R. Dachselt, "Personal augmented reality for information visualization on large interactive displays," vol. 27, no. 2, pp. 1182–1192, 2021.
[49] "Xcharts," https://github.com/XCharts-Team/XCharts, 2022, accessed: 2023-11-08.
[50] Y. Younan, W. Joosen, and F. Piessens, "Code injection in c and c++," *A Survey of Vulnerabilities and Countermeasures, Katholieke Universiteit Leuven, Belgium*, 2004.
[51] C. Stoy, "Game object component system," *Game Programming Gems*, vol. 6, no. 393-403, p. 44, 2006.
[52] E. Gamma, R. Helm, R. Johnson, J. Vlissides, and D. Patterns, "Elements of reusable object-oriented software," *Design Patterns*, 1995.
[53] "Slime rancher accessibility report," https://www.taminggaming.com/en-au/accessibility/Slime+Rancher, 2016, accessed: 2023-11-08.
[54] "Ui accessibility plugin (UAP)," https://assetstore.unity.com/packages/tools/gui/ui-accessibility-plugin-uap-87935, 2021, accessed: 2023-11-08.

Philipp Fleck is a PostDoc researcher at Graz University of Technology. His work focuses on Augmented Reality, Visualization and distributed real-time systems. He received his Master's degree in 2015 and his PhD in 2024 from TU Graz.

Michael Hochörtler is a Master's student at Graz University of Technology. Besides his academic endeavors he runs an Indie Game Development Studio to showcase his visions.

Georg Gotschier received his Master's degree in 2020 in Computer Science from Graz University of Technology. He continues his research on his games at GameLab in Graz.

David Kastl received his master's degree in 2022 from Graz University of Technology. His strong interest in games led him to computer science and his focus on game-related research topics.

Johanna Pirker is an assistant professor at TU Graz. She focuses on game research, human-computer interaction, virtual reality, and educational technologies. She is an active and strong voice of the local indie dev community.

Dieter Schmalstieg (Fellow, IEEE) is Alexander von Humboldt Professor of Visual Computing at the University of Stuttgart. His research interests span augmented reality, virtual reality, and visualization.