Games by End-Users: Analyzing Development Environments

Joana Gabriela Ribeiro de Souza
Department of Computer Science
Universidade Federal de Minas Gerais
Belo Horizonte, Brazil
joana.souza@dcc.ufmg.br

Raquel Oliveira Prates
Department of Computer Science
Universidade Federal de Minas Gerais
Belo Horizonte, Brazil
rprates@dcc.ufmg.br

***Abstract*—Digital games are increasingly present in people's lives, especially in younger people's lives. At times, users want to do more than just play games produced by specialized companies, but would also like to customize them or create and play their own games and share them with their friends. Enabling users with no programming knowledge to develop their games raises several challenges, that range from game design to technical aspects. In this paper, our goal was to investigate the following research question *"What strategies and constructs do game engines use to allow users with no previous knowledge of programming to create games?"*. To do so, we used the Semiotic Inspection Method to analyze two game engines aimed mainly at this audience - GDevelop and Stencyl. As a result, we have identified the main strategies and constructs used in the tools to support game programming by end-users.**

***Index Terms*—end-user programming, games, scientific SIM**

I. Introduction

Currently, there is a movement to introduce more and more computing into people's lives, making it possible for more people to have access and use its methods and techniques. Besides the use of computing being more present in professional spaces, its use for entertainment and leisure purposes is also growing [1]. Movements such as do-it-yourself are changing the users' profile from just content consumers to producers as well, thus empowering them [1].

Among the various uses of digital devices, games are present in the lives of many people, especially young people [2], games can be used in different platforms such as consoles, computers, and mobile devices and have several different genres such as shooting games, strategy, and others. Besides their use for entertainment and leisure, they are also used in different contexts such as serious games [3], which are games that have goals beyond entertainment, such as games aimed at educational purposes, health, or training simulations [4]. Thus, empowering people who are not game developers to act as creators and not just consumers is something that has emerged and is fostering the development of several game engines (software that brings together common tools and utilities for game creation) aimed specifically at this audience [5].

There are several challenges in end-user game development, ranging from game design to technical challenges such as implementing logic, physics concepts, etc. In this paper, we use "end-user" to refer to users interested in game development but who have no knowledge of programming languages. It is important to note that game development itself is challenging, even for professionals [6]. Therefore, understanding the key concepts and abstractions that end-users need to comprehend when designing a game is essential to building game engines that will assist them in this task. Therefore, defining how these concepts and abstractions are presented is a critical factor in improving the tools.

Analyzing game engines by looking at what types of games, strategies, and constructs are used by their developers is one way to understand what means are provided to end-users to develop their game and how. This understanding is important not only to support the evaluation of these tools but also to reflect on whether and how they can be improved and made easier to use by end-users. In the literature, most evaluations of game engines focus on their use by developers, with exceptions of some works that evaluate tools for non-programmers, but focus mainly on their usability and performance [7] [8] [9], or even characterizing game engines based on guidelines [5], without an analysis of what constructs and strategies are used to support the end-user.

In this paper, our goal is to answer the following question: *"What strategies and constructs do game engines use to allow users with no previous knowledge of programming to create games?"*. To achieve this goal, we conducted a systematic analysis using the Semiotic Inspection Method (SIM) [10] of two game engines - GDevelop and Stencyl. As a result, we have identified and listed the key strategies and constructs used in the tools to support game creation by end-users. Our results contribute to the research and the design and development of environments for end-user game creation. They can also be helpful to those interested in using an environment to help them compare or choose between different systems.

The organization of this paper is as follows. Section II presents the related work. Next, we introduce the Semiotic Inspection Method. In section IV, we show the methodology used in this work detailing the choice of the tools evaluated and the preparatory steps and the application of SIM. In section V we describe and present the results of our analysis for each system. Then we discuss the key constructs and strategies used to support the end-user in building games and finally, the conclusions and future works.

II. Related Work

In this research, our focus was on identifying constructs (or primary elements) necessary in a design environment to allow an end-user to create a game and strategies that could support them in their activity. Thus, in our literature review, we focused on papers discussing the design and development of end-user game environments or that presented an analysis or evaluation of such systems.

Several works have performed an analysis or evaluation of game engines, including those that support the development of serious games [11] [9]. In their study [9] points out that the most used engines are the ones aimed at creating games of pure entertainment. In the context of serious games, [12] presents a framework to help users choose the best game engine according to their needs. On the other hand, [13] uses [12] as a basis to develop another framework to help users in the same task but for the creation of games of any nature.

Regarding papers that evaluate games engines, some of them focus on the evaluation by end-users and others by developers. In [14], authors conducted a user evaluation of four game-engines analyzing aspects related to graphic quality, lights, and shadows, among other features for the use of these tools by architects. According to the literature, Unity 3D is one of the most complete game engines used by developers, so some evaluations were performed analyzing different aspects of the tool, such as usability [15] and performance [8].

In respect to the development of game engines, in [16] a mobile game engine was developed and tested. In preparation for the project, the authors compared the main mobile features in four popular game engines. As part of their study, they analyzed the most popular features available in these games engines and considered them as representative of the users' needs. Based on this analysis, they discussed the challenges in including these features in a mobile game engine. The tool developed was not aimed at end-users. Nonetheless, it is related to our work, as they used the analysis of existing tools to identify what designers of these tools considered relevant to users.

In [5], the author's paper proposes an analysis of different features of visual Integrated Development Environments (IDEs) aimed at non-programmers. They selected 25 visual IDEs from various domains and classified them according to a set of features described by a feature model to determine their usability and suitability. In the model, each IDE was described in terms of the features: Language, Workspace, Integration, Human Interface, and Audience, each of these features represented by a number of other (structured) features. As a result, they presented a summarized analysis of how each IDE was characterized through the feature model and an overall description of the IDE. Although the authors have characterized the IDEs according to a detailed feature model, their analysis has not included the main aspects that are the focus of our work - primary game elements offered as part of the visual language to users, and strategies regarding how to create the game.

There are several guidelines with suggestions of aspects that designers should consider when creating development environments for end-users and guidelines for end-users to get started in development environments designed for them. In [17], the authors propose four guidelines aimed at making it easier for end-users to understand code in different contexts. On the other hand, [1] and [18] propose guidelines for designers of end-user developing environments. In [1], the seven guidelines proposed should be taken into consideration for the development of novice programming environments using natural language and new languages. In [18], authors present thirteen design guidelines to support the design of generic (i.e. not focused on a specific domain) end-user development engines. These guidelines describe general considerations that should be taken into account but do not focus on the specific constructs offered to the end-user. Some examples of proposed guidelines are: provide interactions to fully navigate the code, help users use poorly constructed code [17], clearly distinguish 'code' from free-text, make the underlying structure visible to avoid errors of omission or commission [1], make syntactic errors hard and facilitate decomposable test units [18]. In a different direction, [19] proposes a model for a visual programming space that allows for the development of systems using visual programming. Although the model could be used to create other visual programming environments, it is not specific to games.

Besides guidelines and models, other suggestions for improving game engines follow different directions. In [20], authors did a study, and defined requirements to enhance game engines by focusing on affective aspects such as facial expressions and the way the character expresses feelings. They determine that the four main functionalities for an affective game engine are: recognition of user emotion, expression of emotions, representation of player's emotion, and modeling of emotions within the game characters. These recommendations are targeted at professional game developers.

In summary, although we have identified works that evaluate game-engines or discuss their development, we have not found any work that analyze, propose or discuss constructs or strategies for these tools even in the context of serious game development by end-users. In the articles focused on evaluation, their analysis is mainly based on the tool's usability or performance. The ones that discussed development, focused on general guidelines to support end-users' understanding of code. Thus, by identifying the main constructs and strategies used for end-user programming of games, this work contributes to the current state-of-the art of the field.

III. Semiotic Inspection Method (SIM)

Semiotic Engineering theory [22] is a Human-Computer Interaction theory, which perceives the interface of a system as an indirect, one-way communication from the system's designer to its users. The interface conveys to users the designers' understanding of whom the system is intended for, what goals the users want or need to achieve, and how to interact with the system to achieve them. The message

conveyed can be paraphrased through the metacommunication template:

> *"Here is my understanding of who you are, what I've learned you want or need to do, in which preferred ways, and why. This is the system that I have therefore designed for you, and this is the way you can or should use it to fulfill a range of purposes that fall within this vision."*

In this context, the quality of the system is associated with its communicability [23], i.e. how well the system can convey to its users its underlying design intent and interactive principles.

The Semiotic Inspection Method (SIM) [21] [10] is a qualitative, interpretive method for evaluating the communicability of interactive systems, based on Semiotic Engineering. This method has five distinct steps. Initially, there is a preparation step, in which the purpose of the inspection is defined, an informal inspection of the system is performed, and the focus of the evaluation and the evaluation scenario are defined. The next three steps of the method consist of a deconstructed analysis of the design message by separately examining how it is transmitted by the three classes of signs: metalinguistic, static, and dynamic. The metalinguistic signs explain other signs in the systems interface (e.g. tooltips, help). Static signs represent the state of the system (e.g., button, menu option). Dynamic signs represent system behavior triggered by user interaction (opening a modal window after a button click). The last two steps of SIM consist of reconstructing the design message by contrasting, integrating, and interpreting the data collected in the analysis of each class of sign.

Differently from the technical application that seeks to evaluate and improve a system's communicability, the scientific application of SIM aims to answer a research question [10]. The difference in the method applied in the scientific context is mainly in the analysis focused on the why and how of the use of signs within the evaluated system. In addition, endogenous and or exogenous validation are indicated as a way of triangulation of the results. Endogenous triangulation is performed concerning different aspects of the same system or systems in the same domain. When there is a triangulation between systems from different domains, it is considered exogenous.

IV. Methodology

In this work, we have applied the scientific version of SIM, as it is a method that allows for a systematic analysis of the system examined. In our case, we selected two different game engines that would allow us to analyze and perform an endogenous triangulation of the results to answer our research question regarding the constructs needed for end-users to create games and the strategies to support them. Next, we describe our process to select the two game engines and describe how we conducted the application of SIM.

A. Selecting the Game Engines

The first step we took to select the game engines that could be useful to our proposed analysis was to conduct an internet search for game engines that allow end-users to create games. As a result, an initial list of systems indicated in different rankings of game creation environment was generated. Next, we classified each of the game engines found, using 14 criteria that were defined based on aspects expected in modern game engines [9]: (i) no code needed; (ii) free; (iii) 2D game creation; (iv) quality assurance; (v) allows game designing; (vi) allows physics simulation; (vii) player management; (viii) characters and animation creation; (ix) 3D game creation; (x) multi-player network; (xi) exportation for mobile; (xii) exportation for Web; and (xiii) exportation for Desktop; (xiv) allows script-code. If the tools were not free (criteria ii), we checked if they had a free version available and only considered those that did.

As a result, we identified nine game engines that satisfied at least half of the selection criteria, namely: Coppercube[1], Game maker[2], Game Salad[3], GDevelop[4], Godot docs[5], NeoAxis Engine[6], Stencyl[7], Unity[8], and Unreal Engine[9]. As none of the tools met all the selection criteria, we selected the tools that had the main features that we considered more useful for beginners. For example, for a beginner having a 2D game creation feature (criterion iii) was considered more important than having a 3D game creation (criterion ix), as it would be easier to start with 2D development since the insertion of 3D elements can be a very challenging task that may require several tools for its creation and animation [24]. Including a multiplayer network (criterion x) was also not considered essential for beginners.

We then selected Gdevelop and Stencyl, as they met almost all the criteria, except for the possibility to create 3D games (criterion iii) and, in Stencyl, create multiplayer network games (criterion x). Although Unity and Unreal are the most popular for both serious and pure entertainment game development, they require users to have programming knowledge, so we opted for the other two. Furthermore, both Gdevelop and Stencyl have good documentation, tutorials for beginners, an active community, a large number of users, and an updated platform. Both tools have several games on itch.io[10] (an open marketplace for independent digital creators with a focus on independent video games) and in other game stores. For instance, the "Vai Juliette!" game developed at GDevelop, available in App Store and Google Play, has over 1 million downloads.

[1] https://www.ambiera.com/coppercube/index.html
[2] https://www.yoyogames.com/pt-BR/gamemaker
[3] https://gamesalad.com
[4] https://gdevelop-app.com
[5] https://docs.godotengine.org/en/stable/index.html
[6] https://www.neoaxis.com
[7] http://www.stencyl.com
[8] https://unity.com/products
[9] https://www.unrealengine.com/en-US/
[10] https://itch.io

B. Semiotic Inspection Method Application

The scientific version of SIM was applied to analyze the game engines Gdevelop and Stencyl. The inspection was conducted between May and June 2021 by a researcher with experience in applying SIM and supervised by a senior researcher expert in HCI and SIM. According to [10], one inspector is enough to generate an analysis of possible interpretative paths and encoded meanings of an interactive system. In our work, an inspector performed a detailed analysis, and it was consolidated through a discussion with the same senior researcher, by analyzing the inspected signs (or evidences) that supported the interpretations. In addition, each system was inspected separately, and only then the contrast between them and the triangulation of the results generated in each analysis conducted.

As preparation for applying the method, we defined the purpose of the inspection, performed the informal inspection, defined the scope and focus of the evaluation, and defined the application scenario. The goal of the inspection was to answer the following research question: *What strategies and constructs do game engines use to allow users with no previous knowledge of programming to create games?* Thus, the inspection aimed to identify the encoded meanings regarding the available constructs, what interpretations they trigger, and how these interpretations relate to the semantics of the system.

During the informal inspection of the tools' sites, we observed that support materials are available for users, such as documentation, forums, tutorials, galleries with games developed on the game engines, and extensions that can be added. The game engine Gdevelop on its website and platform has multilingual environments (for the interface and the code of the generated game). We also did an informal inspection of the game engines focusing on menus, tooltips, instructions, visuals aspects, and how the tools work.

To define the scope of the analysis, we observed that both game engines provide several features and allow the creation of several types of games. We chose to inspect the game engines using the steps for creating games described in the tutorials for beginners. The tutorials selected for the evaluation of each game engine have common aspects such as a user-controlled character that will move horizontally left and right according to the player's command, actions after collisions, and the concept of life that runs out after collisions. In addition, we evaluated the menus and options available in the engines and their websites.

In general, games are more used by children, teenagers, and young adults [2] [25]. Thus, as game engines facilitate the creation of games, without programming languages, this public can become even more interested in this subject. On the game engines' websites, they offer support for their use in education with the use of the platform for students from schools to universities. So the stakeholder chosen for the creation of the scenario is a teenager who wants to create her first game using one of the tutorials made available by the platforms.

For this evaluation, we used the same scenario for both game engines: *"Maria is 16 years old and likes computer games. She wants to learn how to create games because she is interested in this area. She does not have any programming knowledge, but a friend told her that with the Gdevelop/Stencyl tool, she would be able to create different kinds of games without the need to know any programming languages. She then downloads the tool and decides to follow a tutorial that exists on the platform's website to try to create her own games, play them and even share them with friends."*

V. Results of the SIM Analysis

In this section, we present the results of applying SIM to Gdevelop 5 and Stencyl 4.0.4 systems. We downloaded the programs in the desktop version for Windows 10. For each program, we present a brief overall description and consolidated metamessage (i.e. the message conveyed by designers to the users through the interface) generated from the analysis.

A. Gdevelop

Gdevelop presents itself as a free, fast, open-source, and easy to use the tool. With this tool, it is possible to create different kinds of games (e.g. platform, puzzle, strategy, among others) intuitively, without the necessity to use any programming languages. It allows publishing games in multiple platforms; exporting can be done in one click. Gdevelop provides visual editors, Javascript language, and tutorials created by its team. Figure 1 shows the system's interface.

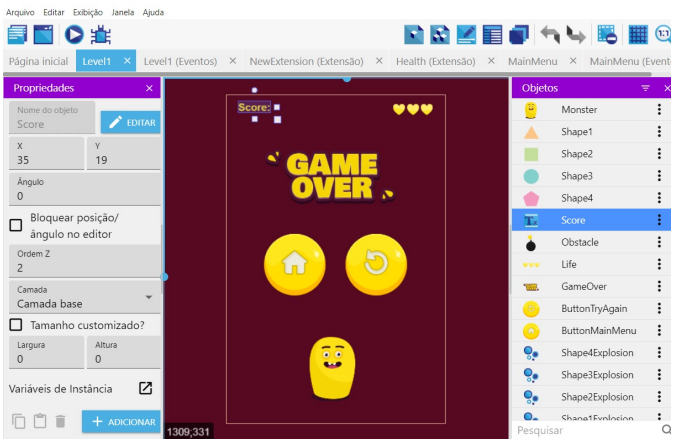

Fig. 1. GDevelop's interface

For this analysis, we followed the "Geometry Monster" tutorial on the platform's website, in which we created a monster that can move left and right (horizontally) according to the player's keypress or mouse touch. The monster must capture geometric shapes that fall from the top of the screen, and when it collides with one of them, the player scores a point. But if the monster collides with a bomb, the player loses a life and does not score points, if it collides with a total of 3 bombs the game is over.

B. Gdevelop's consolidated metamessage

In this subsection, we present the reconstructed metamessage from the GDevelop inspection, following the metacommunication template used in Semiotic Engineering (pg 3).

Who you are: You are fluent in English, or in one of the more than 40 languages in which the platform is translated (even if only partially). You are interested in creating games for professional purposes or as a hobby, with no prior programming knowledge required. You use one of the operational systems: Windows, Mac OS, or Linux, and you have access to the internet to download the tool or use its online version. You are interested in using tutorials and examples to learn how to create and publish your games. You want to play your games and export them to various platforms, and you are interested in playing games created by others.

What you want or wish to do: You want to use a tool for creating games without knowing or having to learn a programming language, and preferably accessible in your language. You want to have access to the platform's social networks through which you will have access to the community and access to games produced on the platform. You want to publish your games in several formats such as web, mobile, and desktop, having the opportunity to use one of the operating systems: Windows, Linux, and Mac OS. You want to start building a game either from scratch by following the steps of a tutorial or opening a pre-existing project. You want to set up the environment in which the game takes place (e.g. dimensions of the scene), as well as define different elements within the game (e.g. characters and objects) their behaviors and events that take place. Once you have created your game, you want to test it. In addition, you want to manage your games by viewing, for example, the analytical data of game access, game details, and its monetization.

Ways in which you can or should use the system: To start a new game, you can either create a new (empty) project or click on the "start with tutorials" option to start a game based on one of the available tutorials. You can also open an existing project you have been working on. If you have any questions, you can click on the "help and documentation" or on the "community forums" buttons. Also, in several features of the system, there is a contextual "help" button that explains the possibilities at that point. You can use the platform's social networks to access more information about games and the community or share knowledge. Via the fixed menu you can select one of the following options: project management, export, run or debug the game.

By opening the project management option, you access several elements to manage, such as scenes (which can be considered the levels or the environment in which the game takes place), "external events", "external layouts" and "functions/behaviors". When you click on an existing scene, the system will open two tabs "scene name" and "scene name (events)". In the first tab, you can add objects that can be, for example, characters, text, visual effects, among others. In this tab, you can edit properties, add or remove objects or groups objects from the scene, edit layers and use tools that make it easier to build the scene. Adding objects to a scene is done via drag and drop by placing the object in the desired position, and you can make adjustments using the mouse or by updating the x and y coordinates in the side configuration window. In a second tab, you can manage the events of that scene.

The system allows you to create objects by clicking on "Add a new object" and selecting the desired type. You can assign properties and behaviors to objects using the point-and-click strategy. Objects created in the scene belong to that scene only. It is possible to assign behaviors and functions to each object. To do so, you must have created the behaviors and functions or have acquired them from the platform's library of extensions. Extensions allow new behaviors for your game's objects, actions, conditions, or expressions ready to be used in events. For example, by assigning the "health" behavior to a monster, you can define the number of lives available to that character through the function "IsDead". Groups of objects can be created through the contextual menu option of a scene, allowing you to create a group from a set of selected objects within that scene. It is then possible to assign events and behaviors to the group, which will be applied to all the objects in that group at the same time.

You can also create and configure events by going to the "*scene name* events" tab. Events and behaviors are the main elements that define the logic of how the game works. Events are created as a list of actions and interpreted by the platform in cascade (from top to bottom). So in this space, you can create and edit events and sub-events (dependent on the parent events) through a few clicks. Creating an event, you can insert conditions to define when this event takes place. You must select the action(s) to be performed when an event occurs and its effects on objects or the environment. You can also include comments with information to help document your game.

To test the game, you should access options on the fixed menu. You should click on the "play" icon to run your game. You can also click on the "bug" icon and select the option to use the debugger and the performance profiler to find possible problems and optimize your game. The menu option to test the game over a network provides a link that can be used for a device on the same network to access and run the prototype.

In case you have created a free account, GDevelop also allows you to access your profile via the "file" menu option. In this case, you will have access to your games panel, in which you can see and access the games you have created, as well as analytic data from players, registration, and monetization details for a given game.

C. Stencyl

Stencyl presents itself as a tool for creating games quickly, easily, and without the necessity to use code. The platform's website says that the user can create a game design for different platforms with an intuitive toolset that accelerates the workflow and then gets out of the way. It also informs that there is no need to use code because it provides a drag and drop interface, but that it is possible, if the user wishes to.

Stencyl allows the creation of worlds and actors and also the possibility to monetize games. The system is free to download, but the game can only be exported for free to the web format; other formats (e.g. desktop or mobile) require the purchase of a subscription. Figure 2 depicts the system's interface.

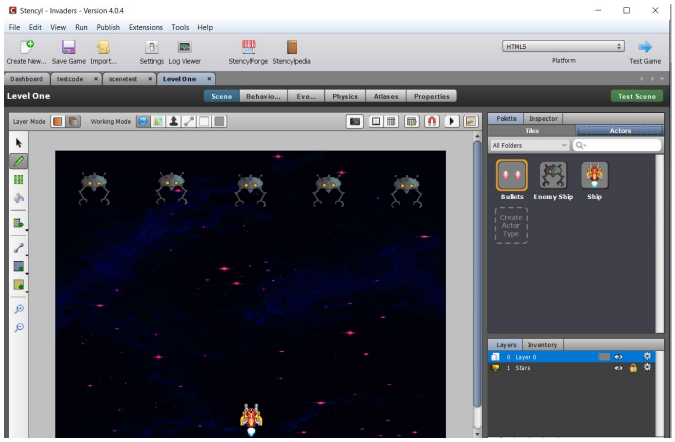

Fig. 2. Stencyl's interface

For this analysis, we followed the tutorial "Crash Course 2" that teaches how to create a "Space Invaders"-type game available on Stencyl's website. Thus, we created a game, in which a spaceship (i.e. the player's character) moves horizontally, fires bullets upwards as the key "q" is pressed. At the top of the screen, there are static enemy ships that are destroyed when they collide with a total of 3 bullets. Users win the game when they destroy all the enemy ships.

D. Stencyl's consolidated metamessage

In this subsection, we present the reconstructed metamessage from the Stencyl inspection, following the metacommunication template used in Semiotic Engineering.

Who you are: You are fluent in English, you have internet access to download the tool, and are interested in creating games with or without the use of a programming language. You are interested in creating an account so that you can create and publish your games. You are also interested in tutorials, examples, and documentation to learn how to create games, their elements and to test them. You are interested in publishing and playing your games, and exporting them to various platforms. You use the Windows, Mac OS, or Linux operational systems.

What you want or need to do: You want to use a tool in English to create games without the necessity to know programming languages, using one of the operational systems: Windows, Mac OS, or Linux. You want to create an account to be able to create and publish your games.

You want to create a new game from scratch using tutorials, or work on an existing project. In addition to the tutorials, you may want to get help in the "getting started" or "about" sections, or learn more from "Stencylpedia" (a Stencyl help center). To create a game, you want to create characters/actors and define aspects as appearance, how they behave, how they move within the scenario, and events that happen in the game and impact them, among other things. As for the scenario, you want to define the background scenes to use, sounds, texts, how physics forces act, among other aspects. You want to save games, import content, access settings, debug and test games in one of the possible formats. In addition, you want to play your games that can be exported to different platforms and you want to play games created by other people.

Ways in which you can or should use the system: You must download the tool and create an account to be able to create and publish your games. When you open the program, you must select one of the following options to start: sample games, choose one of the existing projects, or create a blank one. You can use tutorials and examples to learn how to develop games by clicking on the option in the bottom bar.

After you start creating a game, you can create different types of resources. When an actor is created through the "actor types" resource it has five distinct aspects associated with it. These aspects are present in different clickable tabs: appearance (image that represents it, dimensions, the visual part of the actor), behavior (behaviors applied to it), events (events that relate to the actor), collision (configuration of what happens in case of a collision with others game elements), physics (can define aspects such if it is affected by gravity) and properties (such as name and description). You can select images for the game background by clicking on its settings and their properties. The same behavior works for resources such as fonts and tilesets (sets of small regular-shaped images used to construct the game world or level map). You should click on scenes to create and manage scenes, that are the game's scenarios. You should navigate through the scene's context menu options to define visual aspects of the scene, such as element placement and background, manage behaviors, manage events, define physics aspects (gravity), atlases (resource categories), and properties. You can click on "sounds", set their properties, and import sound files into the game.

To add logic elements such as actor and scene behaviors, you must click on actor/scene behavior and create new behavior. You can use block programming or insert logic directly via a scripting language by clicking on code and using the scripting language Haxe. You can add resource packs (ready-made resource packages, such as actors and behaviors) via "StencylForce" and "create new", and extensions via the "settings" menu option. You can download new extensions from the Stencyl community (e.g features such as behaviors, functions, and others). After being downloaded, the elements added to the project are listed under the appropriate category in the dashboard tab.

To add logic to the game environment the system uses events and behaviors. You must use block language to create both. It can be done by dragging the blocks to the area where the code blocks are placed, snapping the pieces together to form functions. You can also use menu options to help create the conditions. You can use menu options to use existing behaviors, but not for events, which are exclusive to

a particular actor or scene.

You can interact with the main menu that facilitates access to frequently used functionalities. It permits the creation of any resource through the "create new" option, save the game with one click, import resources, access documentation (Stencylpedia), among others. To test the game, you must select the platform you want to simulate it in and then click "test game". In the test context, you can click on "log viewer" to access the execution log of your game.

In addition, you should access the publish option on the top menu to export your game to the desired platform. It is worth noting that with the free license, it is only possible to export a game to the Stencyl community and web. To export it to HTML5, it is necessary to download the Java JDK that requires the installation of export features. Exporting to Flash is not possible because the technology is currently unavailable[11]. With the Chrome Web Store option (Google's online store for web applications), Google requires you to pay a fee to add your game.

"StencylForge" is a game repository that contains resources and media for building games in Stencyl. You can use it to get material to build your games, providing content that is already ready and available. To play other players' games, you should access the Stencyl's site and access the menu option "#madeinstencyl" or through "StencylForge" in the game engine, and then choose an available game.

VI. Constructs and Strategies

The game engines GDevelop and Stencyl have the same general goal, allow people with no programming knowledge to develop games. Nonetheless, as can be seen in the metamessages presented, they offer different solutions to support users in achieving this goal. Despite the differences, based on our analysis, we have considered both systems to be end-user development (EUD) tools, as they both include steps related to the whole life cycle of software development [26].

To respond to our research question, after analyzing the solution proposed by each engine, we performed an endogenous (as both systems share the same domain) triangulation focusing on what are the primary elements included in the systems to support end-users in creating games. As a result, we have identified features in common that are present in both systems to support end-users activities in developing a game. We have organized the features in (i) constructs - primary game elements available to end-users to create a game; (ii) strategies - support offered by the EUD environment to support general development activities. Next, we present these constructs and strategies identified.

A. Constructs for game creation

In this subsection, we present the basic constructs identified in the environments necessary for creating a game. For each construct, we present its definition and explain how it is represented in each system - GDevelop and Stencyl.

[11] https://www.adobe.com/br/products/flashplayer/end-of-life.html

Scenes: Scenes can be defined as the environment in which the game takes place, for example, a world or a level. Thus, a game can have only one scene or several scenes. In both systems, this construct is called **scene**, and a game may contain one or more scenes. In GDevelop, users create a scene and then describe the objects that compose it (their dimensions on the screen, width, height, insertion of characters). In Stencyl, scenes are considered a resource that can be inserted in the game, so a game can have several scenes. In the scene editing space, it is possible to include several different elements and use editing tools. In this case, Stencyl is more robust in the scene construction, providing more resources to the user.

Game elements: Basic elements that can be added to a game scene, such as characters and texts within the scene. Although both tools include a set of elements, they are presented differently in each system. GDevelop presents the abstract concept of "object" that represents the different game elements available. Whereas, in Stencyl, game elements are considered resources. In GDevelop, every element added to a scene is an object, and it can be a character, text, effect, video, others. Each type of object has its peculiarities and can have different behaviors associated with them. For example, in the tutorial, the monster (a sprite object) had the "health" behavior applied to it, allowing us to add to it a maximum number of health/lives at the start of the game, and the number of lives is a function present in this behavior or the "shape explosion" a particles emitter object that adds a visual effect to the shapes. In Stencyl, we can understand game elements as resources that can be added to the game. So you can insert "actor types", "fonts", "sounds", and other resources that compose the game. The use of this feature can help users to think of the game as a collection of small parts with different behaviors and functionalities, which can be considered positive.

Groups: A group can be seen as an element that groups several other elements that will be treated as a single object under some user-defined aspect. In GDevelop, this construct is used to group objects so that all objects in the group can assume the same behavior or be impacted by the same events. There is some emphasis on this construct because there is an option in the scene menu that lists all the object groups, and it is possible to configure them directly. In Stencyl, groups are collections of actors and are specially used to deal with collisions. In this case, collisions are configured between groups of actors.

Events: Events can be seen as occurrences that take place in the game that can trigger any action or feedback in elements or scenes. So they are used to introduce logic into games. Events convey the idea of cause and effect, something usually clearly understood by end-users, which facilitates the use of this metaphor, e. g. when a bullet collides with a character it takes damage. In both platforms, an event is associated with a specific scene (or actor, in the case of Stencyl). In GDevelop, there is a separation in two columns in the event tab. In the first column, the user can add a condition (it is not mandatory), and in the second column, the user can choose the action to be performed. To choose conditions and actions,

the user navigates through the menus and selects among the available options. It is possible to define several settings and create events using pre-determined functions or direct value definitions. In GDevelop, it is also possible to create sub-events, these events that exist within other events, only run when the conditions for the parent event are true or the actions of the parent event have been processed. In Stencyl, event creation uses block programming. The tool provides a menu with 15 block categories, such as flow, actors, scenes, images, among others. Each category contains a set of commands (blocks) available. One of the categories is *custom*, which allows the user to create a new block for specific functionality. Comparing the systems, we can see that Stencyl gives the user more freedom to create events. On the other hand, GDevelop is easier to use when users do not know how to use block programming.

Behaviors: Behaviors add meaningful functionality to objects, scenes, and actors. Features such as allowing the player to move a character with the arrow keys. Once the user defines a *behavior*, it is available for reuse by other elements (e.g. actors) of the game (which is not the case of events). In GDevelop, besides being able to create new behaviors, users can access the behavior search option and choose from a series of options to fulfill the most varied needs. The creation of behaviors is somewhat similar to events, but it involves more steps and elements and tends to be a task for more advanced users because it requires more knowledge about logic (e.g. creating functions), but it is not necessary to use code. In Stencyl, it is similar to creating events in the platform as it also requires using block programming, with the same structure of categories of blocks. Differently from events, for behaviors there is a tab to view the generated code in the scripting language. The addition of this tab facilitating the viewing of the generated code can be seen as a way for the designer to encourage the user to try to learn the script language, or at least a way to allow for an association between block and script languages, in case the user is interested.

Collision: The concept of collision represents the definition of an action that should take place when two game elements (objects/actors/characters) collide. This concept is helpful for many games. For instance, in a shooting game, if the character shoots at an enemy, if the bullet hits (collides) with the enemy, it will trigger an action in which it will cause damage to the enemy (e.g. lose a life). This construct is present in both systems. In GDevelop, when navigating through the menus for creating events, you have to find the collision option, add this parameter, and set it up. In Stencyl, the functionality has greater prominence because it has a specific tab. All actors in the game have a collision tab associated to them. However, the default is that nothing happens when there is a collision. Thus, when users want to define the effects of a collision, they have to define groups of elements and then configure what happens to each of the groups in case of a collision between them. For that, they must use the constructs events or behaviors.

Physics: The use of elements related to physics, like force and gravity, is something commonly used in games, and the tools also provide this support to users, allowing them to define the presence (or not) of gravity, the direction of forces, speed, and others. In GDevelop, this can be done when creating an event, so when determining the action, you must select an object and assign a force to it, this is achieved by navigating the menu system. Stencyl has a specific tab associated with actors and scenes to define whether physical forces will be applied (or not) to these resources that can be configured through option selection and value assignment.

B. Development environment support strategies

In this subsection, we present the strategies that we have identified in game engines as support to users' overall development activities. For each strategy, we present its definition, and explain how it is represented in each system - GDevelop and Stencyl.

Adopting a familiar interacting style: Both engines adopt a drag-and-drop interacting style (i.e. technique that consists of dragging and positioning elements on the screen using the mouse). It is a well-known interaction style that users are usually familiar with[12], as it is used in operational systems, games, and other systems. Users can use the drag-and-drop technique in both the logic and design parts of the game. In GDevelop, this technique is used mainly in scene construction in which the user can drag objects to compose them, and events can be dragged up and down in the programming frame to determine their order of execution. In Stencyl, the technique is used in scene construction by positioning characters and tiles (small regular-shaped images used to construct the game world or level map) within scenes and in block programming, where the user must drag the blocks to the frame where the code is.

Defining inherent modularity: The game engines organize the games in constructs that compose a whole. Thus, they build in modularity, which is considered a good programming practice [27]. In GDevelop, the game is composed of scenes. In each scene, users insert objects (i.e animations, characters, elements of scene composition) and the events and behaviors associated with the scenes or its objects. In Stencyl, the modularization is even greater. Firstly, it separates resources and logic in the dashboard. Then resources are also organized into categories that facilitate the users' understanding of how to deal with each type of resource. Scenes and actors are the most complex elements, and in them, there is a clear division via tabs of visual aspects, events, behaviors, physics, and others. The modularization as done in Stencyl can be more flexible than in GDevelop because users can configure an actor and use it in several scenes, whereas in GDevelop, this is not possible because a character is an object associated with a specific scene.

Providing extensive help: The tools provide easy access to documentation, forums, tutorials, and social media. Help elements are always clearly available in the interfaces, either via

[12]Drag–and–Drop: How to Design for Ease of Use, available in https://www.nngroup.com/articles/drag-drop/

tooltips or help systems. It shows an expectation that the user is supported by the tool and the community in case of doubts and problems, which is a desired characteristic for this type of system. GDevelop offers help in many parts of the system and translation of the platform and portions of the site into other languages. For some languages (e.g. Portuguese, French and Spanish), GDevelop is almost completely translated. Stencyl focuses more on help in menus like "help" and "Stencylpedia", and Stencyl only uses English as a language.

Facilitating testing and debugging: Both systems allow games to be tested at any time using a single click. It makes it easier for the users to see if they have achieved the desired effects. Additionally, in both cases, there are tools to support the debugging process. GDevelop presents a more complete tool in this sense, offering several debugging options such as debugger and performance profiler, while Stencyl presents the execution log of the game in execution. Information about debugging options can be found in the documentation of both tools.

Exporting games: The tools allow games to be exported easily by selecting the format and then exporting, which takes one or at most a few clicks. GDevelop allows exporting to mobile, web, Windows, Mac OS, Linux, and Facebook, using only mouse interaction. The free version of Stencyl allows exporting only to the web.

Supporting advanced users: Both systems include the possibility of their use by more advanced users (or even professionals) by offering users the possibility to use a Script Language in the process of creating a game. In GDevelop, users can create an event or behavior and insert JavaScript code into it. It is interesting that in GDevelop's, tutorials there is an encouragement to use comments in the programming, which is considered good programming practice [27], this helps users to remember what they have done at a future time when they review their events and behaviors. In Stencyl, a scripting language called Haxe is used. As they say, it is similar to ActionScript and JavaScript. Stencyl encourages the use of the scripting language, as it is available in the platform's dashboard, and it is possible to see the code generated when building events and behaviors. In both systems, users can create their games through the visual interface or by using script language. Thus, the script language can also be perceived as an "invitation" to an interested end-user to take a look and even learn the script language and eventually advance their programming knowledge.

VII. Discussion

In our analysis, we have inspected two games engines that support end-users creating games without programming skills. According to Semiotic Engineering theory [22], the solution proposed by designers of these systems is conveyed to users through the system itself. By comparing the resulting metamessage for each of the systems, it is noticeable that they present different solutions to support end-users in their goal to develop a game.

GDevelop creates a more abstract level over the programming concepts and presents to users most of the decisions on game logic as a cause-consequence definition, with the possibility to include conditions. The system makes extensive use of metalinguistic and static signs making the options for creating and configuring events and behaviors clearer by having explicative elements and intuitive names throughout the interface. Thus, beginners who want to create their games can do so through configurable menu options. Nonetheless, users with experience in scripting languages can also use JavaScript in their game creation process if they would like to. The idea of using objects can be seen as a good metaphor for adding elements to games. However, the impossibility to reuse the same object in different scenes can generate re-work in a game with several levels.

Stencyl addresses more directly the concepts of programming in their interface to users. For instance, it organizes the game creation in resources and logic, and game design is through block programming. Block programming gives users greater freedom to use creativity using blocks of various categories and possibilities to create new customized blocks. The scripting language Haxe, specific to Stencyl, appears more noticeably on the dashboard, beyond the possibility to look at the code after generating some function using the blocks, which may encourage users to create familiarity with it or even learn it. Stencyl inspires the reuse of game elements, allowing the reuse of actors in several scenes, and accessing game elements in the repository in Stencylpedia (free access). The scene editor allows defining scenarios in more detail than in GDevelop.

Comparing the overall solution offered by Stencyl and GDevelop, we can see that Stencyl allows users a broader design space to create their games, but at the cost of learning more complex structures such as block programming and the concept of reuse. On the other hand, GDevelop can be considered an easier tool to use in the first contact with game development.

The application of SIM analyzing the designer's metamessage allows us not only to notice the difference between the proposed solutions, but also to identify what the designers considered relevant in supporting end-users in the development of their games. As a result, we have identified through our analysis the constructs offered to users to create their games, and the strategies to support users in the overall activity of developing a game in each of the systems. Triangulating the results of the two systems, we could identify the constructs and strategies that are common to both systems, despite the different solutions proposed by each of them. Thus, these constructs and strategies, which are common to both solutions, are regarded as relevant to support the end-user development of games in general.

VIII. Conclusions and Future Works

This paper analyzed two different game engines, GDevelop and Stencyl, through the scientific application of SIM. As a result, we identified a set of helpful (and probably necessary)

constructs to support end-user game creation and a set of strategies to support users in their overall development process.

The set of constructs and strategies identified are an original contribution of this work that is relevant to people interested in the research, development, and even adoption of game engines aimed at supporting end-user development of games. Researchers can use this set of constructs and strategies to investigate requirements for end-user game-oriented visual languages and end-user development environments; as well as a guide to evaluate or contrast them. Whereas the constructs are specific to the game domain, the strategies are focused on supporting end-user development in general and can be applied to other domains.

For professionals interested in developing new end-user programming engines, the constructs and strategies can help them to make decisions regarding the system they intend to propose and develop. Furthermore, the analysis of GDevelop and Stencyl can also be helpful as they represent distinct solutions, or competing systems, to the same goal. The reconstruction of their metamessage and the discussion comparing them can be useful in reflecting upon the costs and benefits of different solutions and decisions. The results can also be helpful for people who are interested in adopting such systems (e.g. gamers or educators who would like to create serious games for their students), as it helps them choose between GDevelop or Stencyl if they are systems being considered or even guide their overall analysis of other systems of interest.

The main limitations of our work are the threats to its validity mainly due to the number of researchers involved in the inspection and the number of systems examined. Only one researcher conducted the complete analysis of the systems, thus the researcher might have overseen aspects that could also be interesting to our research. This limitation was minimized by presenting and discussing the results with a second researcher. The inspection of only two systems resulted in an initial, but relevant set of common constructs and strategies for our discussion. Nonetheless, it would be interesting, as a future step to analyze other end-user development game engines to further consolidate our results, as well as to investigate if other constructs and strategies emerge.

ACKNOWLEDGMENT

Authors thank CAPES for its partial support to this research.

REFERENCES

[1] J. Good, abd K. Howland, “Programming language, natural language? Supporting the diverse computational activities of novice programmers”. Journal of Visual Languages & Computing, 39, pp. 78–92, 2017.

[2] V. J. Rideout, U. G. Foehr, and D. F. Roberts, “Generation m 2: Media in the lives of 8-to 18-year-olds”. H. J. Kaiser Family Foundation, 2010.

[3] H. Mouaheb, A. Fahli, M. Moussetad, and S. Eljamali, “The serious game: what educational benefits?”. Procedia-Social and Behavioral Sciences, 46, pp. 5502–5508, 2012.

[4] J. Alvarez, and D. Djaouti, “An introduction to Serious game Definitions and concepts”. Serious Games & Simulation for Risks Management, 11(1), pp. 11–15, 2011.

[5] J. M. Rouly, J. D. Orbeck, and E. Syriani, “Usability and suitability survey of features in visual ides for non-programmers”. In Proceedings of the 5th Workshop on Evaluation and Usability of Programming Languages and Tools, pp. 31–42, October 2014.

[6] C. M. Kanode,H. M. Haddad, “Software engineering challenges in game development”. In 2009 Sixth International Conference on Information Technology: New Generations, pp. 260–265, April 2009.

[7] P. Mishra, & U. Shrawankar, “Comparison between Famous Game Engines and Eminent Games”. International Journal of Interactive Multimedia & Artificial Intelligence, 4(1), 2016.

[8] F. Messaoudi, A. Ksentini, G. Simon, and P. Bertin, “Performance analysis of game engines on mobile and fixed devices”. ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM), 13(4), pp. 1–28, 2017.

[9] B. Cowan and B Kapralos, “An overview of serious game engines and frameworks”. Recent Advances in Technologies for Inclusive Well-Being, pp. 15–38, 2017.

[10] C. S. de Souza, C. F. Leitão, R. O. Prates, S. A. Bim and E. J. da Silva, “Can inspection methods generate valid new knowledge in HCI? The case of semiotic inspection”. International Journal of Human-Computer Studies, 68.1-2: pp. 22–40, 2010.

[11] B. Cowan, and B. Kapralos, “A survey of frameworks and game engines for serious game development”. In 2014 IEEE 14th International Conference on Advanced Learning Technologies, 662-664. July 2014.

[12] P. Petridis, I. Dunwell, D. Panzoli,S. Arnab, A. Protopsaltis, M. Hendrix and S. de Freitas, “Game engines selection framework for high-fidelity serious applications”. International Journal of Interactive Worlds, p. Article ID 418638, 2012.

[13] E. Christopoulou and S. Xinogalos, “Overview and Comparative Analysis of Game Engines for Desktop and Mobile Devices,” International Journal of Serious Games. Vol. 4, Issue 4, pp. 21–36, Dec. 2017.

[14] T. Koehler, A. Dieckmann, and P. Russell, “An evaluation of contemporary game engines”. In 26th eCAADe Conference Proceedings, Antwerpen (Belgium), pp. 743–750, September 2008.

[15] Ş. Mercan, and P. o. Durdu, “Evaluating the Usability of Unity Game Engine from Developers’ Perspective”. In 2017 IEEE 11th International Conference on Application of Information and Communication Technologies (AICT), pp. 1–5, September 2017.

[16] A. G. Peker and T. Can, “A design goal and design pattern based approach for development of game engines for mobile platforms”. In 2011 16th International Conference on Computer Games (CGAMES), pp. 114–120, July 2011.

[17] P. Gross, & C. Kelleher, “Non-programmers identifying functionality in unfamiliar code: strategies and barriers”. Journal of Visual Languages & Computing, 21(5), pp. 263–276, 2010.

[18] A. Repenning, & A. Ioannidou, “What makes end-user development tick? 13 design guidelines”. In End user development. pp. 51–85, Springer, Dordrecht, 2006.

[19] M. Hirakawa, M. Yoshimi, M. Tanaka, & T. Ichikawa, “A generic model for constructing visual programming systems”. In 1989 IEEE Workshop on Visual Languages. pp. 124–125, IEEE Computer Society, Jan. 1989.

[20] E. Hudlicka, “Affective game engines: motivation and requirements”. In Proceedings of the 4th international conference on foundations of digital games, pp. 299–306, April 2009.

[21] C. S. de Souza, C. F. Leitão, R. O. Prates and E. J. da Silva, “The semiotic inspection method”. In: Proceedings of VII Brazilian symposium on Human factors in computing systems, pp. 148–157, 2006.

[22] C. S. de Souza, “The semiotic engineering of human-computer interaction”. MIT press, 2005.

[23] R. O. Prates, C. S. De Souza & S. D. Barbosa, “Methods and tools: a method for evaluating the communicability of user interfaces”. Interactions, 7(1), pp. 31–38, 2000.

[24] A. Ioannidou, A. Repenning and D. C. Webb, “AgentCubes: Incremental 3D end-user development”. Journal of Visual Languages & Computing, 20(4), pp. 236–251, 2009.

[25] A. Y. Irmak, and S. Erdogan, “Digital game addiction among adolescents and young adults: A current overview”. Turkish Journal of Psychiatry, 27(2), 2015.

[26] B. R. Barricelli, F. Cassano, D. Fogli, and A. Piccinno,“ End-user development, end-user programming and end-user software engineering: A systematic mapping study”. Journal of Systems and Software, 149, pp. 101–137, 2019.

[27] I. Sommerville, “Software engineering 9th Edition”. ISBN-10, 137035152, 18, 2011.

[28] C. Kelleher, & R. Pausch, “Lowering the barriers to programming: A taxonomy of programming environments and languages for novice programmers”. ACM Computing Surveys (CSUR), 37(2), pp. 83–137, 2005.