GEnEbook: A Game Engine to Provide Electronic Gamebooks for Adventure Games

Victor Travassos Sarinho
Lab. de Entr. Digital Aplicado (LEnDA)
Univ. Estadual de Feira de Santana (UEFS)
Feira de Santana, Bahia, Brasil
vsarinho@uefs.br

***Abstract*—Gamebooks can be defined as a printed fiction that allows the reader to participate in the story by making choices. They were famous in the 80s or 90s and are reappearing today due to the combination of readers' interest with the ease of use and the presentation novelty provided by ebooks. This paper presents GEnEbook, a game engine proposal able to provide electronic gamebooks for adventure games. It uses a game data model and a multimedia game panel capable of dynamically represent gamebook adventures, together with a game builder structure able to provide valid ebook files according to security limitations defined by ebook readers. As a result, a development approach able to create interactive and gamified ebooks (the *g-books*) was provided, together with a reusable perspective for the fast generation of electronic gamebooks (*g-books*) without worrying about interpretation and execution details of compatible ebook readers.**

***Index Terms*—game engine, ebook, gamebook, g-book, interactive story, adventure game**

I. Introduction

Books can reveal how great people made things possible, spreading words of wisdom and transforming the readers lives. They are also able to *connect* ordinary people to the best minds that have accomplished something in life, and *connection* is the correct word to define our current moment in the world.

In contrast to the pleasure of touching and feeling the reading of a physical book, digital books (ebooks) have been gaining a notorious space in today's internet time. They offer many benefits and advantages, which can be summarized as: delivered almost instantaneously, no trees required to manufacture them, sold nowadays with bonuses, no space to store them, portable, searchable, can be interactive, no packing and shipping expenses, and so on[1]. As a result, ebooks allow people to really use the phrase "there are so many books and so little time to read them" that crosses the minds of many who are passionate about books and reading.

Per gamebooks, it is "*a book with a story that can be read sequentially or not*" [1], where the player keeps reading until he reaches a fork in the story, going to a different page every time the book asks him to make a choice, until he reaches the end of the book[2]. Since most gamebooks have more than one ending, when the player finishes reading the first time, he can go back and start again, but this time making a totally different set of choices, resulting in a completely different story for your enjoyment.

Gamebooks became famous in the 80s or 90s, and a possible resurgence of gamebooks is coming, due to the combination of the readers interest with ease of use and presentation novelty currently provided by the technological support of ebooks. However, despite the multimedia resources offered by ebooks, they require, for security reasons, a static structure for navigation and representation of their elements. As a consequence, it makes difficult to dynamically represent in a systematic reusable way all the necessary elements to create gamebooks, especially when observing the current production strategies of text adventure games, where the developer only sets the desired characteristics for the game without worrying about interpretation and execution details.

In this sense, this paper presents the design, production and validation of the **GEnEbook**, a **G**ame **En**gine proposal able to provide **E**lectronic game**books** for adventure games. It is based on a game data model and a multimedia game panel capable of dynamically represent gamebook adventures, together with a gamebook builder application able to construct ebook packages for a *g-book* in line with security limitations defined by ebook readers.

II. Related Work

A. Adventure Games & Development Approaches

Text adventures represent a subset of the adventure genre, where the player takes the role of the protagonist in an interactive story driven by exploration and puzzle solving [2]. It represents one of the oldest types of computer games genre, and contains different types of concepts, examples and reusable strategies able to develop them.

Focusing on the reuse of text adventure games, Gabsdil *et al.* [3] described a game engine for text adventure games that uses computational linguistics and theorem proving techniques based on description logic.

Following a data-driven approach, Quest is a free, open source software for creating interactive stories and text adventure games [4]. It is a point and click editor that configures main elements and commands necessary to represent a textual adventure, such as rooms and exits, verbs to read and respective responses, objects to interact, player inventory, and so on.

[1]https://www.successconsciousness.com/ebooks_benefits.htm
[2]http://gamesvsplay.com/a-brief-history-of-gamebooks/

As an open-source tool for telling interactive, nonlinear stories, Twine allows the creation of interactive fiction in the form of web pages[3]. A Twine game is made from “nodes” (or “passages”) and links between them. The application’s interface represents each node as a box, and links between nodes as arrows from one box to another [4]. Twine also allows the creation of simple stories without writing any code, but it is possible to extend the stories with variables, conditional logic, images, CSS and JavaScript via defined macro system.

Finally, regarding text messaging platforms, Script Creation Utility for Nodejs Maniacs (SCUNM)[5] is a text-adventure game engine that uses Telegram as standard output for text, images (even animated gifs) and interactive selections to represent the gameplay.

B. *Javascript Game Engines for Text Adventure Games*

TextAdventure[6] is a text adventure game engine based on four main components: a simple server, a retro command line that represents the *Terminal*, the text adventure engine itself colloquially called the *Console* and finally the *cartridges* (game configs) which are defined by two JavaScript objects.

Cartridges are loaded into the Console to be playable by the user. They consists of two important objects, *gameData* and *gameActions*, together with a collection of helper functions. The gameData object has four required components; *commandCounter*, *gameOver*, *introText*, *outroText*, along with a *player* object and a *map* object. It can also contain any number of other fields, objects and functions as needed by the cartridge. The *gameActions* object holds *functions* that the user can access as available actions that will be performed during the gameplay.

Text-engine[7] is a small (approximately 200 lines) and easy to use text-based adventure game engine. It uses a disk metaphor (JSON data) to represents the configured game, which describes three top-level properties: *roomId* (String), a reference to the room that the player currently occupies; *inventory* (Array), the list of items in the player’s inventory; and *rooms* (Array), the list of rooms in the game.

Each room has: a *name* (String), that will be displayed each time it is entered; an *id* (String), that represents an unique identifier for this room; the *img* (String), that will be displayed each time the room is entered; a *desc* (String), with the description of the room that is displayed when it is first entered and when the player select the *look* command; the *items* (Array), with a list of items in this room that can be interacted; and the *exits* (Array), with the list of paths from this room. Each *exit* indicates the direction the player must go to leave (the *dir* property) and the next room that the player will go (the exit *id* property). Each item has a *name* (String) to identify it, a *desc* (String) to be displayed when the player looks at the item, the *isTakeable* (Boolean) indication whether the player can pick up this item (if it’s in a room), and the *use* (Function) that will be called when the player uses the item.

Advenjure[8] is a text adventure game engine that allows the player to move around *rooms* and interact with *items* through verb *commands* such as Go, Look, Take, etc. Per items, they are represented by maps, having a *name*, a *description* and a set of key-value pairs to customize *behavior*. Rooms can have only one name, and an optional attribute to display an initial room description the first time the player visits it. Rooms are also maps, and once the developer have created a lot of items, it needs to put them in a room or directly into the player’s inventory. Each room map is built by connected rooms in a plain clojure hash map.

As advanced features available in the engine, it is possible to: *Overriding messages*, using custom messages for a given action on a room or item; define *Pre conditions*, a function hook to define if an action can be performed; define *Post conditions*, a function hook to customize how the game state is modified after an action is performed; use *Dialogs*, in this case interactive dialogs with “character” items; apply *Text customization and internationalization*; use *Custom verbs/commands*; and define *Plugin hooks*, in order to customize a game behavior without modifying the library.

C. *Providing Gamified Ebooks*

Regarding the production of games to play in different types of game platforms, Okuda and Emi [5] showed how to make a game using the EPUB3[9] format with some interactive functions based on HTML5 related technologies.

For the production of gamified ebooks in a game engine perspective, Figueiredo and Bidarra [1] presented an evaluation of the possibility of creating gamebooks that are effective in teaching and learning. After analyzing the features available in free and open tools for making ebooks, they decided to develop an Unity based tool to provide a novel interactive book (the *g-book*).

An extension of Figueiredo and Bidarra work was proposed by Figueiredo, Bidarra and Natálio [6], were EPUB3 and iBooks Author[10] versions were provided to build a model of a dynamic book that may function as an educational game. Preliminary tests with the developed prototypes revealed a good usability and a promising pedagogical potential for the proposed models.

Droutsas, Patsilinakos and Symvonis [7] also described a personalized system for learning, based on EPUB3 format, that allows the creation and use of interactive and personalized eBooks for educational purposes.

Miller and Ranum [8] developed an open source electronic textbook that incorporates a number of active components such as video, code editing and execution, and code visualization as a way to enhance the typical static electronic book format, the learning experience for students and the teaching experience for instructors.

[3]http://twinery.org/
[4]http://catn.decontextualize.com/twine/
[5]https://github.com/jlvaquero/SCUNM
[6]https://github.com/TheBroox/TextAdventure.js/
[7]https://github.com/okaybenji/text-engine
[8]https://github.com/facundoolano/advenjure
[9]http://idpf.org/epub/30/
[10]https://www.apple.com/ibooks-author/

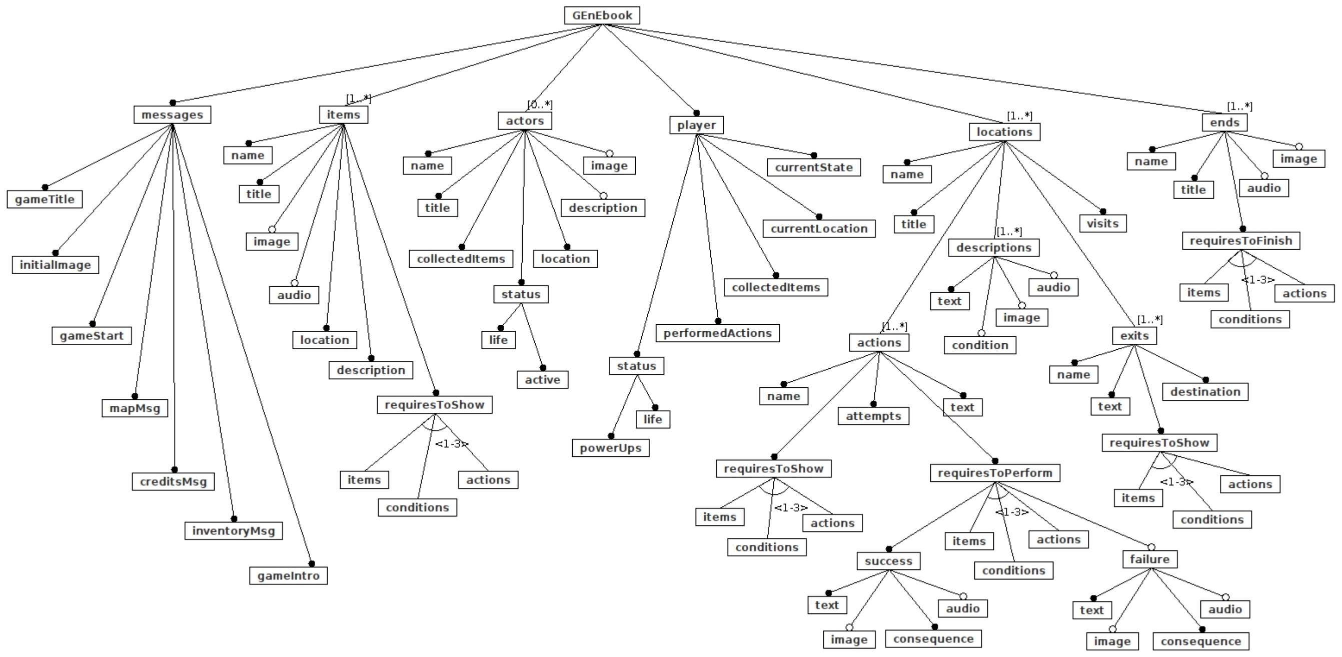


Fig. 1. Feature diagram of the proposed JSON game data.

Möslein-Tröppner and Bernhard [9] presented the production of collaborative gamebooks[11] as an alternative and contemporary method for transmitting knowledge in education. To this end, it presented a manual for the creation of digital gamebooks with collaborative elements, capable of transmitting the learning content in an interactive, fun and collaborative way through modern media.

Finally, Sigarchian *et al.* [10] proposed a novel concept of an EPUB 3-based Hybrid e-TextBook to allow the interaction between the digital and the physical world. For this, an EPUB 3-based Hybrid e-TextBooks prototype was developed to connect the EPUB learning content to smart devices in classrooms, leveraging both digital publishing and Semantic Web tools.

III. Methodology

Game engines for text adventure games present common structures and behaviors able to be configured in new and distinct adventure games, such as rooms, items, actions, conditions to perform, and so on. Regarding the production of gamified ebooks, it is limited to the development of specific and dedicated gamebooks, or to the use of game engines to build casual and mini gamebooks in dedicated platforms. As an attempt to integrate these two perspectives in a public format for ebook readers, GEnEbook provides a game engine for text adventure gamebooks able to be executed in compatible EPUB3 readers.

In this sense, the GEnEbook construction was performed in three main steps. The first one was the creation of a JSON model able to represent text adventure games, as stated in defined structures described in related work. The second step was the production of a JavaScript+HTML5 game engine able to execute the modeled JSON in line with security limitations imposed by ebook readers. The third step was the implementation of a gamebook builder able to: 1) apply a default EPUB3 structure in the desired ebook; 2) insert the required media according to the modeled JSON data in a static way; and 3) put the developed game engine together with the JSON data in a valid EPUB3 file for ebook readers.

[11]https://www.gamebook.ch/dgb/eisenhower/

A. The JSON Model

Domain analysis is a requirements engineering approach for a Software Product Line (SPL) with a result that can be documented in a feature model [11], or in a JSON data model as a Domain Specific Language (DSL) representation [12]. It is an important step in the SPL Engineering process (SPLE), whose main objective is: 1) the explicit manipulation of system variability, and 2) the systematic reuse of implementation artifacts [11].

Looking for the systematic reuse of text adventure game artifacts, five main features were defined in a proposed JSON game data able to represent a DSL for text adventure games: *player*, *actors*, *items*, *locations* and *ends* (Fig. 1). They are based on common structures represented by related text adventure game engines, such as: *gameActions*, *player* and *map* objects of the *TextAdventure*; *rooms*, *inventory* and *exits* of the *Text-engine*; and *items*, *pre-conditions* and *pos-conditions* of the *Advenjure*. An extra structure was also defined to represent game messages that are used during the gameplay, such as *gameTitle*, *gameIntroduction*, *goButton*, *creditsText*, among others.

The *player* feature presents information about: the current game state (*currentState*), the current room of the player (*currentLocation*), a list of collected items due to performed actions (*collectedItems*), a list of performed actions (*performedActions*) to be interpreted by game rules, and the current player status (*status*) with information about number of *lifes*, *powerUps*, etc.

The *actors* feature presents a list of possible Non-Player Characteres (NPCs) and enemies with desired information for the game story. Each actor feature in the *actors* list has a *name*, a *title*, a *description* and an *image* to be shown to the player, together with the current room in the game (*location*), the list of collected items (*collectedItems*), and the *status* properties, such as number of *lifes* or if the actor is yet *active* or not.

The *items* feature presents a list of items available in the game. Each item in the *items* list has a *name*, a *title*, a *description*, an *image* and an *audio* to be shown to the player. The *location* property describes the room where the item is available, and the *requiresToShow* property describes the conditions to decide when the item can be shown or not to the player. If the player has some previous items (*requiresToShow.items*), or performed previous actions (*requiresToShow.actions*), or has conditions (*requiresToShow.conditions*) whose evaluations results were *true*, then the player can see the respective item when entering the room where the item is located,

The *locations* feature presents a list of available rooms in the game. Each location has a *name*, a *title* and possible *descriptions* with *text*, *image* and *audio* to show, according to the evaluation of the associated *condition*. Possible *actions* to be performed in the location are also available, with *name*, *text*, *requiresToShow*, *requiresToPerform* and number of *attempts* remaining for each configured action to describe it.

The *requiresToPerform* property is similar to *requiresToShow*, but including *success* and *failure* behaviors to be performed after the *requiresToPerform.items*, *requiresToPerform.actions*, and *requiresToPerform.conditions* evaluation. Both *success* and *failure* present *text*, *image* and *audio* content to be shown to the player, together with the *consequence* execution, becoming possible to execute JavaScript expression and commands, game engine routines or execute another *action* if necessary.

As possible *exits* in a location to visit another location, each exit defines the *name* identification, the *text* description for the exit button, the *destination* (another *location name* to go) and the *requiresToShow* property. For each player visit in a configured location, the *visit* property is incremented by one.

Finally, the *ends* feature applies verification conditions to determine if the gamebook was finished or not. For this, collected *items*, performed *actions*, player lifes, time limit and other *condition* values can be verified by the *requiresToFinish* property after each player interaction with the ebook. Together with the verified conditions, the final content to be presented to the player is also defined by the *title*, *text*, *image* and *audio* properties, according to the specified death situation for the player.

B. The Game Engine

There are several limitations to use JavaScript+HTML in an ebook for the EPUB3 format. The security mode, for example, avoid the inclusion of dynamic content in an ebook due to the *sandbox* security for EPUB3 files with embedded scripts. As a persistence problem, the values of script variables in an ebook content can be erased after loading a new ebook chapter or ebook topic due to the reader navigation over the ebook. In addition, the loading of external JSON files presents some synchronization problems during an ebook initialization or chapter loading, becoming possible the duplication or the restart of declared values in script variables. Moreover, a previous declaration and inclusion of all external media content that will be used in an ebook is required for security reasons, becoming necessary a dynamic building of the ebook structure in line with the applied ebook media.

In order to solve these described limitations, the GEnEbook was defined as a game panel that is loaded in the ebook initialization together with the modeled JSON data (game data and game engine in only one file). It works as a media state control able to show and hide media resources that should or should not be presented according to the ebook story.

The proposed game panel includes a pool of static buttons for possible *items*, *actions* and *exits* able to be shown according to the gameplay, whose labels are based on *local storage* values that are updated according to the gamebook reader interaction. For the gamebook required media, it is previously included in the game panel as HTML tags for media content (*audio* and *img*), but with the *hidden* property activated.

Predefined areas are also configured in the proposed game panel to organize the gamebook presentation (text, buttons and media content), such as: *introduction*, *inventory*, *map*, *credits*, *info*, *end*, *story*, *items*, *actions*, *exits* and *options*. They are represented as HTML tags whose content are defined by the manipulation of *innerHTML* and *style* properties according to the player context during the gameplay.

To control the presentation of this game areas, 7 main rendering states will indicate when each one will be shown: *game-start*, *game-status*, *game-info*, *game-inventory*, *game-map*, *game-credits* and *game-end* (Fig. 2). *Game-start* is activated when the gamebook is initialized, presenting the introductory information about the gamebook. *Game-status* shows the current status of the game, working as the main panel of the gamebook. *Game-info* presents intermediary information about the game, such as the result of a game action, or more details about an available item in a room. *Game-inventory* shows the current items collected by the player that are available to be used. *Game-map* presents all the rooms that the player visit during the current gamebook reading. *Game-credits* present some information about the gamebook creators. Finally, *game-end* shows the game end that the player achieved during the gameplay, which can be represented by different types of game wins or losses.

Some game routines are also provided by the game engine to execute player interactions according to the proposed

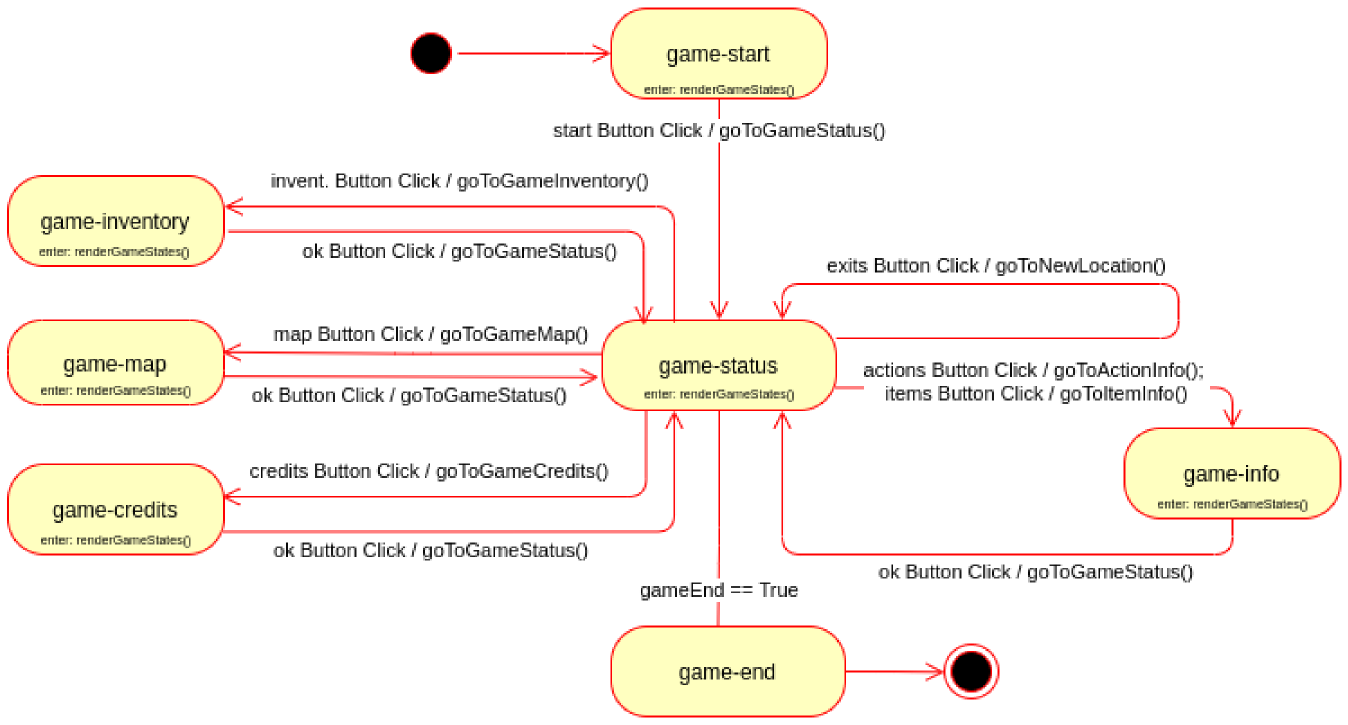

game-start
enter: renderGameStates()
start Button Click / goToGameStatus()
invent. Button Click / goToGameInventory()
game-inventory
enter: renderGameStates()
ok Button Click / goToGameStatus()
exits Button Click / goToNewLocation()
map Button Click / goToGameMap()
game-map
enter: renderGameStates()
game-status
enter: renderGameStates()
ok Button Click / goToGameStatus()
actions Button Click / goToActionInfo();
items Button Click / goToItemInfo()
game-info
enter: renderGameStates()
credits Button Click / goToGameCredits()
game-credits
enter: renderGameStates()
ok Button Click / goToGameStatus()
ok Button Click / goToGameStatus()
gameEnd == True
game-end

Fig. 2. GEnEbook state machine for rendering states life cycle.

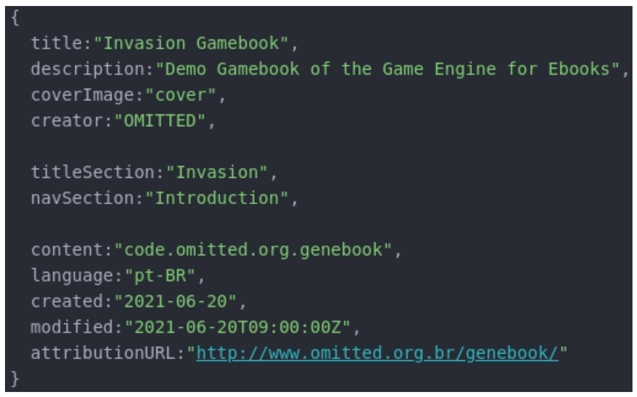
rendering states. They are used to: get specific elements in the current player context (*getPlayer*, *getActor*, *getItem*, *getLocation* and *getAction*); execute player routines to change the game context (*putInTheBag*, *takeOutOfTheBag*, *insertAction*, *removeAction* and *execAction*); and perform the "player navigation" among rendering game states (*goToGameStatus*, *goToNewLocation*, *goToActionInfo*, *goToItemInfo*, *goToGameCredits*, *goToGameMap* and *goToGameInventory*).

As an example, the *execAction* routine, when called by the *goToActionInfo* routine, is responsible to execute the indicated action in a button after the player click. To complement an action execution, the *execAction* routine can also be called by a *success* or *failure consequence*, which is performed by the evaluation of the respective *requiredToPerfom* property.

C. The Gamebook Builder

An EPUB3 ebook is a packaged file that follows a predefined directory structure populated with specific file types according to the EPUB3 format. The "*opf*" file type, for example, is responsible to define the external/extra files that will be used by the ebook. Navigation files, such as "*nav.ncx*" and "*nav.html*" are responsible to indicate the book content (summary, chapters, book sections) that the player can select to navigate over the book. Furthermore, "*mymetype*", "*container.xml*" and "*EPUB.css*" files are necessary to complete the EPUB3 compatibility with ebook readers according to the proposed format.

Each EPUB3 ebook has some particular data to identify it, such as title, description, cover image, creator, content, language, creation data, modified data, attribution URL, and so on. In addition, due to security reasons, each EPUB3 ebook demands the static definition of the necessary content (xhtml, script, figures, audio) to be evaluated by the ebook reader. Furthermore, the EPUB3 directory content must be validated and correctly packaged to become a valid EPUB3 file for ebook readers.

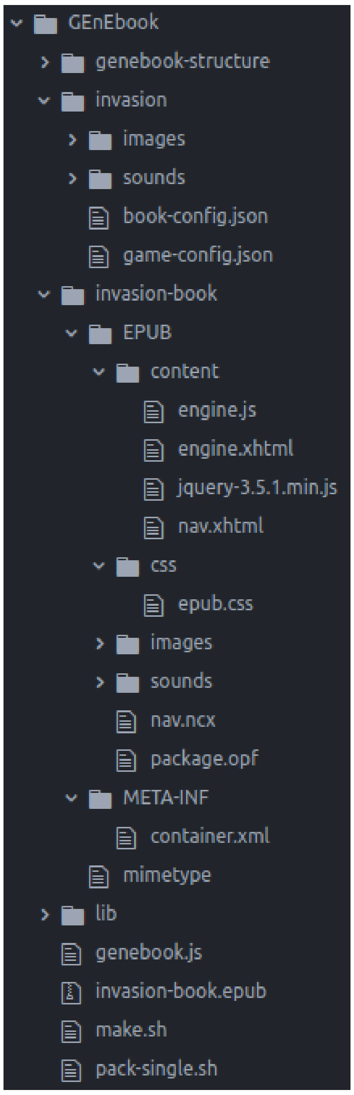

```
{
  title:"Invasion Gamebook",
  description:"Demo Gamebook of the Game Engine for Ebooks",
  coverImage:"cover",
  creator:"OMITTED",

  titleSection:"Invasion",
  navSection:"Introduction",

  content:"code.omitted.org.genebook",
  language:"pt-BR",
  created:"2021-06-20",
  modified:"2021-06-20T09:00:00Z",
  attributionURL:"http://www.omitted.org.br/genebook/"
}
```

Fig. 3. JSON model for book configuration.

Two JSON files are used by the GEnEbook gamebook builder to prepare and build a final and valid EPUB3 file: the *game-config* and the *book-config*. *Game-config* is responsible to define the game logic according to the proposed DSL based on features for text adventure games. *Book-config* defines the necessary information to configure predefined structure according to EPUB3 format (Fig. 3). Two directories (*images*



```
v GEnEbook
  > genebook-structure
  v invasion
    > images
    > sounds
      book-config.json
      game-config.json
  v invasion-book
    v EPUB
      v content
          engine.js
          engine.xhtml
          jquery-3.5.1.min.js
          nav.xhtml
      v css
          epub.css
      > images
      > sounds
        nav.ncx
        package.opf
    v META-INF
        container.xml
      mimetype
  > lib
    genebook.js
    invasion-book.epub
    make.sh
    pack-single.sh
```

Fig. 4. EPUB3 directory and files of the Invasion gamebook.

and *sounds*) are also necessary to allocate JPG images and MP3 audios that will be used as static media resources for the gamebook.

Both JSON files and media directories are used by a *NodeJS*[12] app (*genebook.js* called by *make.sh*) that performs the following steps to provide a valid EPUB3 directory and content: 1) open and read the JSON files; 2) make a new ebook directory based on the new ebook name; 3) copy a baseline EPUB3 content to the created ebook directory; 4) copy multimedia content (*images* and *sounds* directories) to the new book directory; 5) prepare the "nav.xhtml" and "nav.ncx" files using *book-config* data; 6) integrate the game engine and the *game-config* data in the "engine.js" file; 7) prepare the "engine.xhtml" file to be the game panel with static and hidden media content; and 8) prepare the "package.opf" file with the references for the static media resources. In the end, the *pack-single.sh* (also called by *make.sh*) executes the *EPUBcheck.jar*[13] file to validate the EPUB directory content and generate the desired EPUB3 file. Fig. 4 illustrates a gamebook directory example and the provided EPUB directory after the gamebook builder execution steps.

[12]https://nodejs.org/

D. The Invasion Gamebook

GEnEbook developers can define dynamic rules with multimedia content to represent locations, items, actions, exits and ends in a desired game. The freedom is limited by the use of *requiresToShow*, *requiresToPerform* and *requiresToFinish* elements to decide when and what information must be shown or not to the player according to the gameplay context.

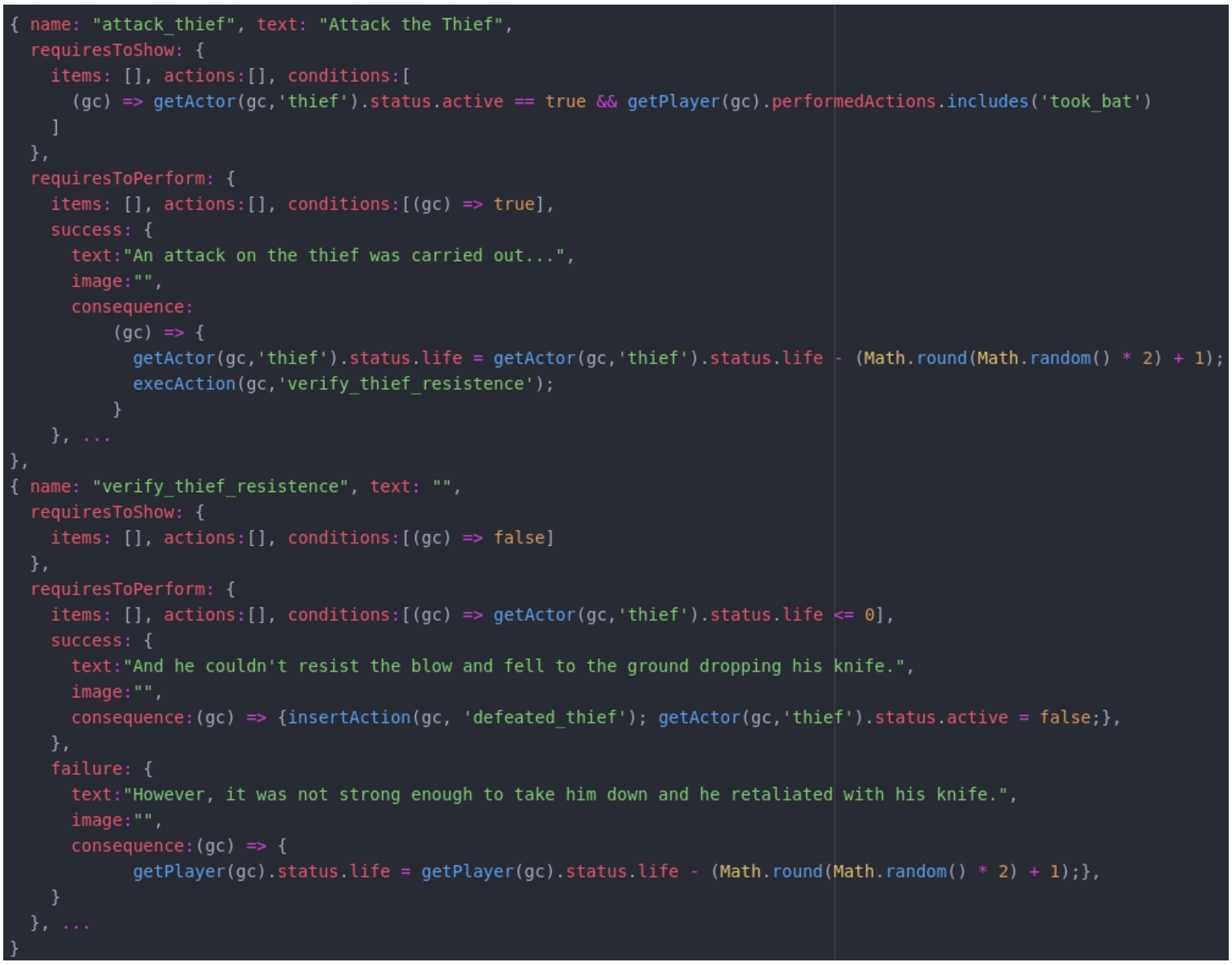
In this sense, an evaluation gamebook called "Invasion" was developed using the proposed GEnEbook assets. The game objective is to escape from a thief that invaded the kitchen of the house, after the player wakes up. A battle against the thief is also performed during the gamebook reading, and the player can win or not according to the player luck during the fight. If the player wins the fight it can get the thief knife, which is the necessary tool to escape the house and call the police in the end.

Fig. 5 shows the use of *requiresToShow* and *requiresToPerform* elements in a *location* to provide a fight with a dynamic result in the game, where: 1) the "**attack-thief**" action appears to the player, if the *Thief* is alive and the player has the *Bat*; 2) when the "**attack-thief**" action is performed, it reduces the Thief live with a random value and executes the "**verify-thief-resistence**" action; and 3) the "**verify-thief-resistence**" evaluates if the *Thief.life* $<= 0$, inserting the "*defeated-thief*" action in the *performedActions* list of the player and killing the Thief (*status.active* = *false*) as a result. If the *Thief.life* > 0, the player live will be reduced by a random value, whose result will be verified by the *ends.requiresToFinish* element to decide if the game was finished or not after the action execution.

Regarding the gamebook execution, Fig. 6 presents the first rendered page of the gamebook when it is initialized by the ebook reader[14]. At this moment, the *game-start* status is activated, and the *introduction* area presents the *game-title* text, *game-intro* text, and *game-start* button. If an initial *game-image* or *game-audio* content is configured in the JSON data, the respective *img* and *audio* tags are shown by setting the HTML *display* property.

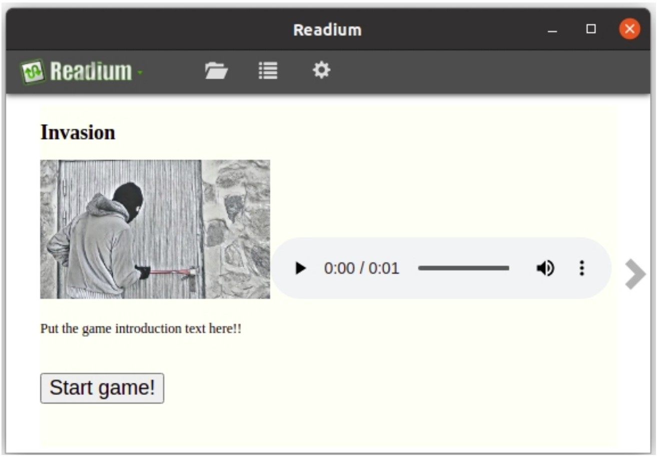
When the player clicks on the *game-start* button, the *game-status* is activated. Then, the main panel of the game is shown by the ebook reader[15], hiding the introductory game info and showing the *story*, *items*, *actions*, *exits* and *options* areas with the respective player location content (Fig. 7). The *story* area presents the location *title* and the location *description* that can change according to the gamebook story, together with the configured *story-audio* and *story-image* content. *Items*, *actions* and *exits* are represented as a list of available buttons for the player selection, whose *innerHTML* values are changed

[13]https://github.com/w3c/EPUBcheck
[14]https://readium.org/
[15]https://github.com/futurepress/EPUB.js/



```
{ name: "attack_thief", text: "Attack the Thief",
  requiresToShow: {
    items: [], actions:[], conditions:[
      (gc) => getActor(gc,'thief').status.active == true && getPlayer(gc).performedActions.includes('took_bat')
    ]
  },
  requiresToPerform: {
    items: [], actions:[], conditions:[(gc) => true],
    success: {
      text:"An attack on the thief was carried out...",
      image:"",
      consequence:
          (gc) => {
            getActor(gc,'thief').status.life = getActor(gc,'thief').status.life - (Math.round(Math.random() * 2) + 1);
            execAction(gc,'verify_thief_resistence');
          }
    }, ...
},
{ name: "verify_thief_resistence", text: "",
  requiresToShow: {
    items: [], actions:[], conditions:[(gc) => false]
  },
  requiresToPerform: {
    items: [], actions:[], conditions:[(gc) => getActor(gc,'thief').status.life <= 0],
    success: {
      text:"And he couldn't resist the blow and fell to the ground dropping his knife.",
      image:"",
      consequence:(gc) => {insertAction(gc, 'defeated_thief'); getActor(gc,'thief').status.active = false;},
    },
    failure: {
      text:"However, it was not strong enough to take him down and he retaliated with his knife.",
      image:"",
      consequence:(gc) => {
            getPlayer(gc).status.life = getPlayer(gc).status.life - (Math.round(Math.random() * 2) + 1);},
    }
  }, ...
}
```

Fig. 5. JSON actions for the player fight in the Invasion gamebook.



Invasion

Put the game introduction text here!!

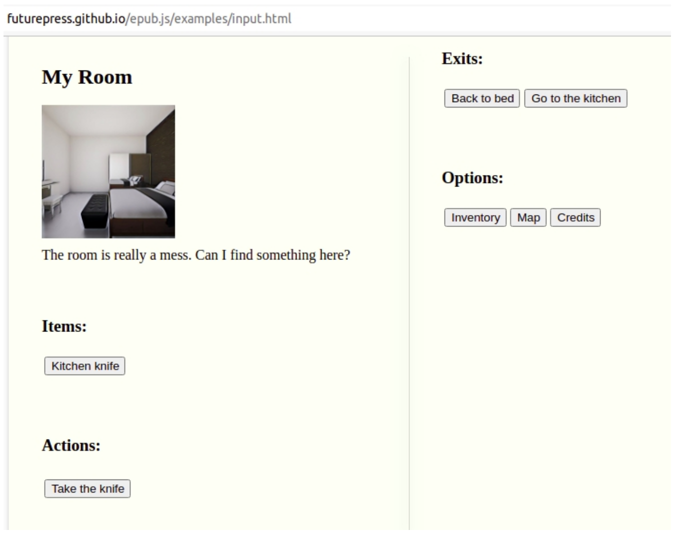
Fig. 6. Initial screen of the Invasion gamebook at Readium[14] ebook reader.

dynamically according to the gamebook story. The *options* area presents buttons that allow the player to navigate to the *game-inventory* (Fig. 8), *game-map* (Fig. 9) and *game-credits* states, showing the respective content as a result according to the ebook reader[16] [17].

When the player selects an *exit* button, a new location is defined (the *goToNewLocation* routine), and the *game-status* refresh the game panel content to show the new game context information (Fig. 7). When the player selects an *item* button, the *game-info* status is activated to show the respective item information (Fig. 9). The same status is also activated when an *action* button is selected, indicating the action name to be executed (the *execAction* routine), and showing the success or failure results of the action execution. In both cases, an update of the *info-title*, *info-image*, *info-text*, *info-audio* and

[16]https://www.edrlab.org/software/thorium-reader/

[17]https://play.google.com/store/apps/details?id=com.gmail.jxlab.app.reasily &hl=engl=US


futurepress.github.io/epub.js/examples/input.html

My Room

The room is really a mess. Can I find something here?

Items:

Kitchen knife

Actions:

Take the knife

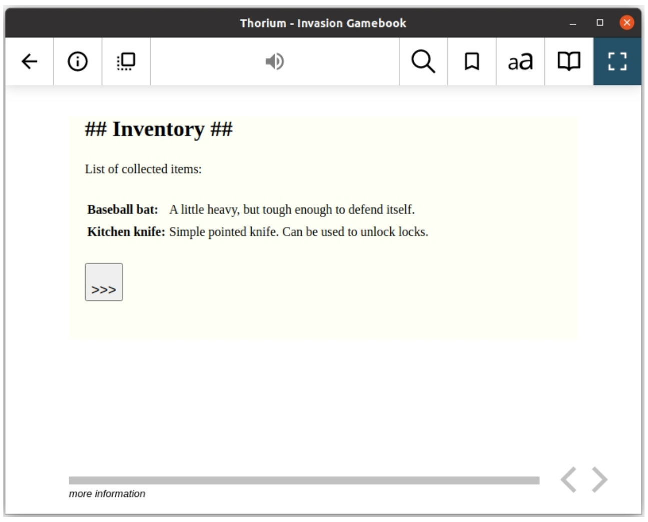
Exits:

Back to bed | Go to the kitchen

Options:

Inventory | Map | Credits

Fig. 7. Main screen (game panel) of the Invasion gamebook at EPUBjs[15] ebook reader.


Thorium - Invasion Gamebook

Inventory

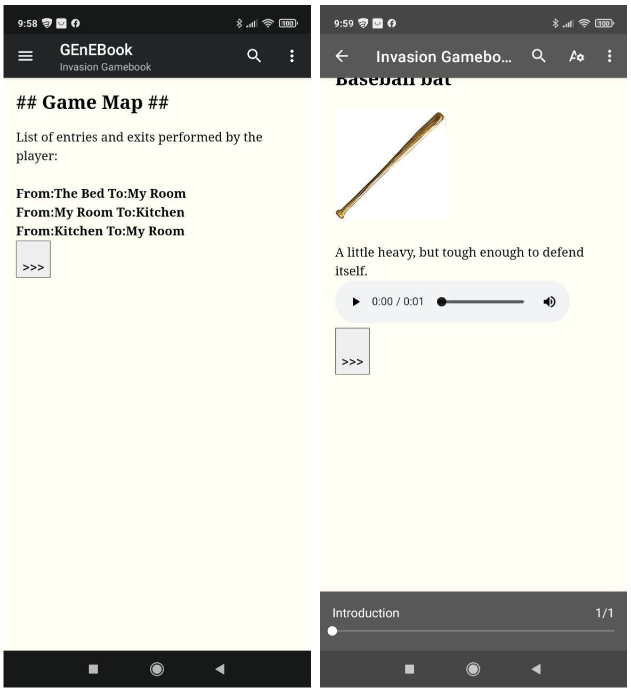
List of collected items:

Baseball bat: A little heavy, but tough enough to defend itself.
Kitchen knife: Simple pointed knife. Can be used to unlock locks.

>>>

more information

Fig. 8. Inventory screen of the Invasion gamebook at Thorium[16] ebook reader.


GEnEBook
Invasion Gamebook

Game Map

List of entries and exits performed by the player:

From:The Bed To:My Room
From:My Room To:Kitchen
From:Kitchen To:My Room

>>>

Invasion Gamebo...

Baseball bat

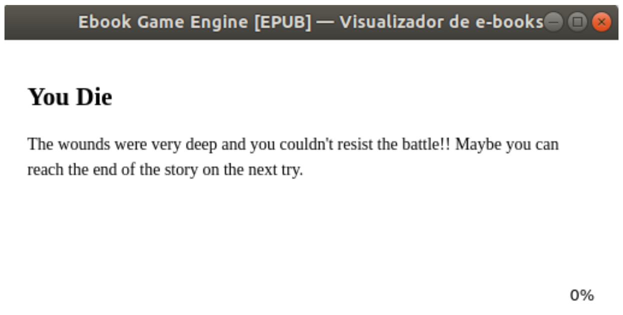
A little heavy, but tough enough to defend itself.

0:00 / 0:01

>>>

Introduction 1/1

Fig. 9. Map screen at Reasily[17] ebook reader, and Information screen for items and actions at Lithium[18] ebook reader.


Ebook Game Engine [EPUB] — Visualizador de e-books

You Die

The wounds were very deep and you couldn't resist the battle!! Maybe you can reach the end of the story on the next try.

0%

Fig. 10. End screen of the Invasion gamebook at Calibre[19] ebook reader.

info-go values is performed, as illustrated in Fig. 9 by the ebook reader[18]. To exit this state, the *info-go* player's click (at the button with ">>>" label) sets the *currentStatus* of the player to *game-status*, refreshing the main game panel with the current game context values.

Finally, considering the gamebook end, by each verification of the JSON *ends* conditions, when one of the ends condition is satisfied, the *game-end* state is activated and the *end-title*, *end-image*, *end-audio* and *end-text* values are shown to the player by the ebook reader[19] (Fig. 10).

[18]https://play.google.com/store/apps/details?id=com.faultexception.reader&hl=engl=US
[19]https://calibre-ebook.com/

IV. Obtained Results

As different types of ebook readers are available today, the generated gamebook Invasion was evaluated in distinct versions of ebook readers for desktop, mobile and web platforms. Table I presents the used ebook readers and their respective platforms, together with the identified inconsistencies and possible reasons and solutions to solve them.

As obtained results, desktop versions presented audio and image dimension problems, web versions presented image dimension and security restriction problems, and no problems were found in mobile versions during the gamebook execution. However, despite the identified execution problems, it is important to state that the gameplay of the Invasion gamebook in general was not affected by them. The only exception was the

TABLE I
EVALUATION RESULTS OF THE INVASION GAMEBOOK.

Ebook Reader	**Platform**	**Problems, Reasons and Solutions**
Calibre (v5.22.1)	Desktop	Audio controls are shown but sounds are not played. Unfortunately, the sound support is not available for this ebook reader.
Thorium (v1.7.1)	Desktop	Audio controls are not shown. Images with wrong dimensions. Possible workaround for audio play is available to be applied. Necessary use extra style configurations for image components.
EPUBjs (v0.3)	Web	—
Readium (v2.31.1)	Web	Images with wrong dimensions. Necessary use extra style configurations for image components.
EPUB Reader (v2.0.12)	Web	Security problem with audio and image loading. Necessary to investigate whether it is a Javascript issue, an EPUB configuration issue, or an ebook reader incompatibility.
Lithium (v0.24.1)	Mobile	—
Reasily (v2021.08)	Mobile	—
Adobe Digital Editions (v4.5.11)	Mobile	—.

TABLE II
REUSE METRICS OBTAINED WITH THE INVASION GAMEBOOK.

Gamebook SLOC / Total SLOC	**Gamebook CC / Total CC**
15+395/(463+1286+15) = 23,24%	1+13/(78+142+1) = 6,33%

EPUB Reader[20] use, which was not able to run the gamebook correctly.

Regarding the reuse level achieved with GEnEBook, Fig. 11 presents some collected metrics [13] using the Plato code analyzer [14] for the Javascript files of the Invasion gamebook. These metrics are used to calculate the Gamebook SLOC and Gamebook Cyclomatic Complexity (CC) by the sum of *game-config.js* and *book-config.js* data (Table II), and to calculate the Total SLOC and Total CC by the sum of *engine.js*, *gene-book.js* and *book-config.js* data (Table II). As *game-config.js* is combined with *engine.js* to provide a valid Javascript code for a final version of the *engine.js* file, *game-config.js* data is not included in the sum of Total SLOC and Total CC. As obtained result, without considering the reused HTML and extra EPUB assets necessary to provide a valid ebook file, approximately 77% of SLOC reuse and more than 93% of CC reuse were obtained, showing the game core reusability and maintainability for developed GEnEBook games.

V. CONCLUSIONS AND FUTURE WORK

This paper presented the GEnEbook, a game engine proposal able to provide electronic gamebooks (*g-books*) for adventure games. It provided a JSON model in a DSL perspective capable of representing text adventure games according to related text adventure game engine models. It also provided a JavaScript+HTML5 game engine able to interpret JSON data according to security limitations imposed by ebook readers, together with a gamebook builder application that can integrate the modeled game and the developed game engine in an initial gamebook structure able to provide a valid EPUB3 file for ebook readers.

For the initial GEnEbook prototype, *locations*, *items*, *enemies*, *inventory*, *player status*, *exits*, *requirements to perform* actions, *requirements to show* outputs and *requirements to finish* the game can be modeled by the JSON game data. For the JavaScript+HTML5 game engine, a pool of buttons, whose values are adapted according to *local storage* values, were used as input interfaces for player interactions (*items*, *actions*, *exits* and *options*) in a main game panel that shows the current game status and content (text, audio and image) according to the player context in the adventure game. Finally, for the ebook production, a NodeJS builder was responsible to provide the final EPUB3 file to represent the modeled gamebook, by integrating a previous directory structure according to EPUB3 format with: the modeled JSON, the implemented game engine, and the static media content that will be used by the gamebook.

Regarding the related work, these have either developed their own readers to represent the modeled gamebooks, or their static solutions and manuals/support rules for programming a specific and desired ebook. GEnEbook can provide valid gamebooks according to the EPUB3 format in a systematic reusable perspective (a new gamebook for each new JSON data) without worrying about interpretation and execution details of ebook readers. As a result, GEnEbook proposed a possible solution that covers these two related production strategies, since it uses only JSON configurations to represent the desired adventure game model. Furthermore, it is able to build gamified ebooks (*g-books*) with interesting reuse metrics results for different ebook readers, becoming interesting and possible to be extended by new game builders for different game environments and platforms as dedicated gamebook readers.

However, as GEnEbook limitations, not all ebook readers presented the generated EPUB3 file in a correct way. As identified errors, they are associated with security limits imposed by the browser when using media content, and with HTML5+Javascript compatibility in the configuration of *style* properties. Moreover, it is also necessary to improve the interface layout for the provided gamebooks, as well as their respective web components, something that will be achieved through necessary improvements in the CSS settings together with an advanced use of the *Canvas* component resources. Finally, additional tests are necessary to determine the performance results with the proposed gamebook structure, in special to evaluate the media content limit that can be loaded by evaluated ebook readers, in order to avoid long delays capable of compromising the player's interaction with modeled gamebooks.

As final considerations for this research, despite the limitations found in ebook readers [15], which were mostly

[20]https://chrome.google.com/webstore/detail/EPUB-reader/mbcgbbpomkkndfbpiepjimakkbocjgkh?hl=en

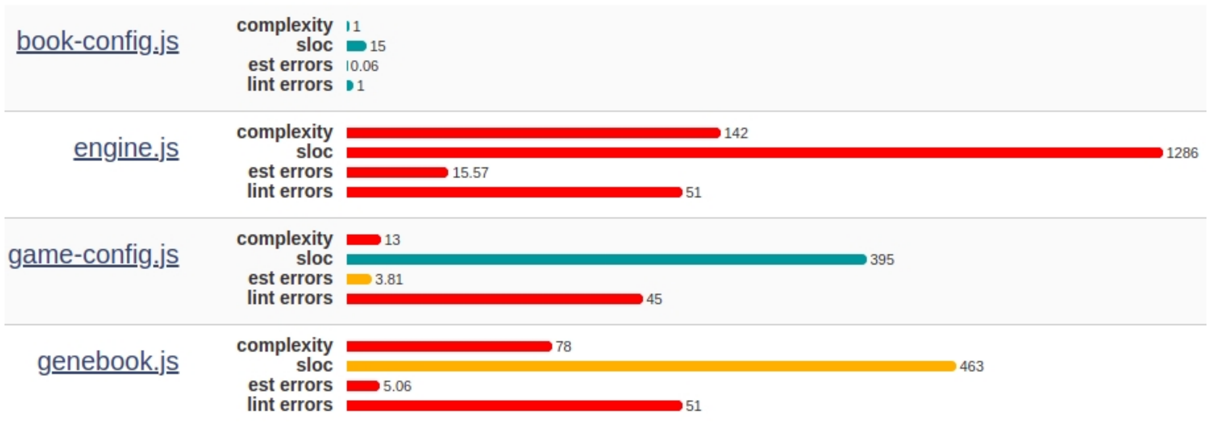


Fig. 11. Collected metrics using the Plato code analyzer.

"circumvented" in the case of GEnEBook, EPUB3 files can be considered as a "safer" and standardized platform for compressing and distributing games and dynamic multimedia resources in comparison with other web publishing strategies. In this sense, it is important to emphasize that, despite the GEnEbook ability to generate games that are able to be packaged and distributed as a web game for different types of browsers, the challenge of this research was to provide gamebooks using EPUB files, which is a known, stable and limited standard for packaging multimedia content for a dedicated audience. Moreover, regarding the current GEnEBook development and audience, a GEnEBook version with fps generation and graphic manipulation (drag'n'drop, point-and-click) is already in production. In this sense, the GEnEBook target audience for this moment will still be game programmers. However, there is the interest to allow the GEnEbook use by a less specialized audience, which opens the possibilities for the creation of support tools (such as Twine) to simplify its use in the development of games for specific game domains, such as quizzes, puzzles, adventures, cards, among others.

As extra work for this research, make the GEnEBook available for the community, conduct courses to publicize the tool and get feedback from external developers are important activities that will be carried out in the near future. In addition, as only an initial trial version was produced with GEnEBook to show the potential use of JavaScript features in games for ebooks, it is also necessary to select some physical gamebooks to be converted to the EPUB3 format by GEnEBook assets for validation purposes. Moreover, as extra work for this research, it is necessary to carry out: the creation of a graphic editor for composing game adventures according to the proposed JSON; the creation of a web repository to make the produced adventure games openly available; the creation of game builders for different target platforms (messaging applications, embedded environments, customized reader); the creation of games of different genres (quiz, storytelling) based on extensions of the proposed JSON model; the implementation and improvement of new interaction components capable of being used as game elements (puzzles, animations, etc.); and the extension of the compatibility of the produced EPUB3 file with other ebook readers currently in use.

References

[1] M. Figueiredoa and J. Bidarrab, "The development of a gamebook for education," *Procedia Computer Science*, vol. 67, pp. 322–331, 2015.
[2] A. Rollings and E. Adams, *Andrew Rollings and Ernest Adams on game design*. New Riders, 2003.
[3] M. Gabsdil, A. Koller, and K. Striegnitz, "Building a text adventure on description logic," in *International Workshop on Applications of Description Logics, Vienna, September*, vol. 18, 2001.
[4] B. D. Ballentine, "Textual adventures: Writing and game development in the undergraduate classroom," *Computers and Composition*, vol. 37, pp. 31–43, 2015.
[5] S. Okuda and K. Emi, "Make once, play anywhere!: Epub 3 interactive function enables us to make and play game software anywhere!" in *2013 IEEE 2nd Global Conference on Consumer Electronics (GCCE)*. IEEE, 2013, pp. 381–384.
[6] J. Bidarra, M. Figueiredo, and C. Natálio, "Interactive design and gamification of ebooks for mobile and contextual learning," *International Journal of Interactive Mobile Technologies (iJIM)*, vol. 9, no. 3, pp. 24–32, 2015.
[7] S. Droutsas, P. Patsilinakos, and A. Symvonis, "Interactive personalized ebooks for education," in *2018 9th International Conference on Information, Intelligence, Systems and Applications (IISA)*. IEEE, 2018, pp. 1–8.
[8] B. N. Miller and D. L. Ranum, "Beyond pdf and epub: toward an interactive textbook," in *Proceedings of the 17th ACM annual conference on Innovation and technology in computer science education*, 2012, pp. 150–155.
[9] B. Möslein-Tröppner and W. Bernhard, *Digitale Gamebooks in der Bildung*. Springer, 2018.
[10] H. Ghaem Sigarchian, S. Logghe, R. Verborgh, W. De Neve, F. Salliau, E. Mannens, R. Van de Walle, and D. Schuurman, "Hybrid e-textbooks as comprehensive interactive learning environments," *Interactive Learning Environments*, vol. 26, no. 4, pp. 486–505, 2018.
[11] S. Apel, D. Batory, C. Kästner, and G. Saake, *Feature-oriented software product lines*. Springer, 2016.
[12] V. T. Sarinho, G. S. de Azevedo, F. M. Boaventura, and F. de Santana, "Askme: A feature-based approach to develop multiplatform quiz games," in *XVII Brazilian Symposium on Computer Games and Digital Entertainment (SBGames)*, 2018.
[13] W. Frakes and C. Terry, "Software reuse: metrics and models," *ACM Computing Surveys (CSUR)*, vol. 28, no. 2, pp. 415–435, 1996.
[14] Plato, "Javascript source code visualization, static analysis, and complexity tool," https://github.com/es-analysis/plato, 2012.
[15] B. Leporini, L. Minardi, and G. Pellegrino, "Interactive epub3 vs. web publication for screen reading users: the case of'pinocchio'book," in *Proceedings of the 5th EAI International Conference on Smart Objects and Technologies for Social Good*, 2019, pp. 235–238.