Providing an IM Cross-Platform Game Engine for Text-Messaging Games

Victor T. Sarinho, Gabriel S. de Azevedo, Filipe M. B. Boaventura
Lab. de Entretenimento Digital Aplicado (LEnDA)
Universidade Estadual de Feira de Santana (UEFS)
Feira de Santana, Bahia, Brazil
Email: vsarinho@uefs.br, gabrielsilvadeazevedo@gmail.com, fmbboaventura@gmail.com

***Abstract*—Several types of Text Messaging (TM) games are available today. They explored different types of games genres, categories and platforms, such as Short Messaging Service (SMS) competitions and text-based adventures on Instant Messaging (IM) bots. This paper presents IMgine, a game engine able to provide TM games across distinct IM platforms. It implements game engine routines in a configured game loop for multiple IM platforms, capable of representing TM game states and logic by the execution of predefined game functions. As validation process, IM games were developed to show the IMgine capability of providing game mechanics and dynamics aspects according to TM and IM restrictions. As a result, a foundation for IM games that organizes the definition of TM game states and logic is provided, allowing game developers to be concerned with the game core implementation instead of IM transmission process or TM interaction limits.**

***Keywords*-text-messaging games; instant messaging; cross-platform; game engine;**

I. Introduction

Instant Messaging (IM) and Textual User Interface (TUI) represent a historic and important combination for digital games. IM services have provided new opportunities for game development, offering a new vocabulary [1] and multimedia capabilities [2] for game designers. TUI has been applied as a common approach to define basic human-computer interactions, such as prompts and menus [3], for traditional text-based games. Together, they have been used to provide different types of single and multiplayer Text Messaging (TM) games over the years, such as trivia, combat and strategy games [4].

TM games are in many cases simple, small and directly programmed for native platforms, discouraging additional investment in reusable assets during game production. However, the continuous evolution of communication technologies and game design requirements increased the "internet time" problem [5] for TM games, where the rapid usage of IM innovations and the short time to launch it become top priorities.

In this sense, reuse strategies have been applied over the years to provide TM games for IM platforms, such as game engines for dedicated games [6], game engines focused on game genres [7] for dedicated platforms [8], and proprietary solutions on the cloud to design games on modern IM platforms [9]. As a consequence, there is a current demand for open source TM game engines able to use modern IM platforms resources in different types of game genres.

This paper presents **IMgine**, an open source IM cross-platform game engine for TM games. It defines IM communication resources, TUI models and a generic game loop able to perform an overall flow control according to developed TM games.

To this end, section 2 presents related work on TM games and reusable strategies applied to them. Section 3 describes the applied methodology to perform the generic game loop for TM games in distinct IM platforms, and presents the production of two TM games via proposed game engine. Section 4 presents the obtained TM game results together with a qualitative and code analysis of the developed game engine. Finally, section 5 presents the conclusions and future work of this project.

II. Related Work

Different types of concepts, examples and reusable strategies were applied for TM games. Some of them can be summarized in game genres, such as Text Adventure and Quiz games. Others can be described following the evolution of IM technologies, in this case from Short Messaging Service (SMS) to modern IM platforms supported by Chatbot games.

In this sense, for the reuse of Text Adventure games, Gabsdil, Koller and Striegnitz [10] described an engine for text adventures 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 text adventure games [11]. 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.

For Quiz games, whose main purpose is the successful answering of questions, Mininel et. al. [12] described the application of the Mogabal game engine to provide quiz games using permanent event-sprites, which are linked to a long list of random quizzes regarding various topics. Wolf et al. [13] proposed a Jeopardy! like quiz game with questions automatically generated from DBpedia, gathering ranking information about persons to provide a basis for the evaluation of semantic ranking heuristics. Cheong et al. [14]

presented the use of a multiple-choice quiz implemented by a software tool (Quick Quiz), which was applied in three undergraduate IT-related courses as a gamified learning activity. Finally, Klopfenstein and Bogliolo [15] proposed the Quiz-Master game, an application of the deep linking feature that allows a bot to ask contextual questions in a persistent quiz session.

Considering SMS games, Alien Revolt [16] was a Location-Based Mobile Game (LBMG) that follow a SMS strategy to locate and "shoot" other players within a specific radius in the city space. Bontchev et al. [17] provided a mobile chess game that could be played simultaneously in Web browsers and SMS mobile phones. Marcote et al. [18] introduced the AMUSE platform to support mobile gambling services (lotteries). Flintham et al. [19] presented "Day of the Figurines", a narrative-driven TM game for mobile phones that sends and receives SMS in order to support an episodic storytelling on mobile phones. Olla et al. [20] affirmed the usage of the MADIC SMS game engine for speed and cost reasons to create effective mobile applications. Finally, PhoneAdventureGame [21] provided an adventure game engine that used Twilio's response language (TwiML) to redirect game inputs and output via SMS Text process actions.

Regarding Chatbot, talkbot, chatterbot, bot or IM bot games, which represent a computer program that conducts a conversation via auditory or textual methods, "Hello, Stranger!" [22] is a text-based adventure available in multiple IM bots and mobile platforms whose game goal is to help the main character to escape the ocean depths alive in real time. LibrasZap [23] is a multi-IM game bot that evaluates the player knowledge in the Brazilian Language of Signals (LIBRAS). Klopfenstein et al. [24] presented a Telegram bot-based multiplayer game that was deployed to handle large-scale treasure hunts in different use-cases, including educational, cultural, or touristic applications. Script Creation Utility for Nodejs Maniacs (SCUNM) [8] is a text-adventure game engine that uses Telegram as standard output for text, images (even animated gifs) and interactive selections representing the game play. Telegram Gaming Platform [25] allows the distribution of HTML5 chatbot games with graphics and sound loaded on demand as needed, just like ordinary webpages. Finally, for IM cloud platforms, OnSequel [9] allows the production of interactive multi-IM games, building quiz, story-based games, or whatever else in an easy, model-based, and code-free approach.

III. Methodology

The IMgine development and validation process was divided in 3 main parts: 1) the definition of a multi-IM platform to be used; 2) the configuration of game engine routines to represent an IM game loop; and 3) the implementation of game functions called during the game loop to develop the core of IM games.

A. The Multi-IM Platform

Regarding available multi-IM platforms solutions [26], SPLIMBo is an open source Software Product Line (SPL) [27] that was chosen to configure and deploy cross-platform IM bots in a "write once, run anywhere" perspective. It is based on local hosting support and XML-based specifications (the ZapML language) to describes textual dynamics for TUI responses in configured IM bots.

ZapML defines *Menu*, *Prompt*, *Command*, *Sequence*, *Exec* and hypertexts tags as main XML elements to represent IM bots. Each ZapML response to IM clients is based on a tag evaluation, which defines the current state of a bot in an IM conversation. For example, when a *Menu* tag is evaluated, the IM bot shows the menu options and wait for a IM client response. If the IM client selects a sub *Menu* option, it executes a new ZapML tag that can be a *Command*, a *Sequence*, or even a new *Menu*. If a back option is selected, it returns to the previous tag content. If the IM client sends an invalid option, the current *Menu* is executed again showing the same menu options of the current node [26].

```
<zapapp path=".../DemoZap/">
  <menu includeBackOption="false">
    <text>Options: </text>
    <command description="Hello World" keycode="System.order">
      <text>Hello World!!!</text>
    </command>
    <prompt description="Echo" keycode="System.order"
          execAfterConfirmation="echo=System.currentMessage"
          confirmationMessage="Echo: System.echo">
        <text>Send a text (V- Back):</text>
    </prompt>
  </menu>
</zapapp>
```

Figure 1. *DemoZap* configuration.

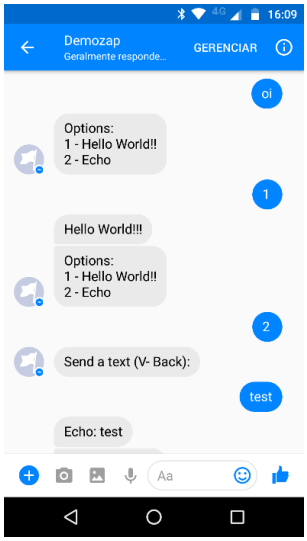

Figure 2. *DemoZap* configuration running on Facebook Messenger.

Figure 1 illustrates one ZapML configuration example called *DemoZap*. It is based on one *Menu* and two *Command*

tags (the *Menu* options) able to send *"Hello World!!!"* and a message *Echo* as IM response [26]. Figure 2 also demonstrates the *DemoZap* configuration running on the Facebook Messenger platform.

B. Game Engine, Game Routines and Game Loop

Game engine is defined as an "extensible software that can be used as the foundation for many different games without major modification" [28], and represents "the collection of modules of simulation code that do not directly specify the game's behavior (game logic) or game's environment (level data)" [29]. For game loops, they offer "an overall flow control for the entire game program", executing a sequence of game actions per frame repeatedly until the user quits [30]. Game loops also "decouple the progression of game time from user input and processor speed" [31], organizing the execution of game tasks to achieve consistent simulations in a game [32] [33].

Regarding the configuration of game engine routines to represent an IM game loop, Sicart [34] defines game loop as an "algorithm that relates the current state of the game and the properties of the objects with a number of conditions that consequently can modify the game state". In addition, Gregory [28] affirms that the "main game loop runs repeatedly, and during each interaction of the loop, various game systems such as artificial intelligence, game logic, physics simulation, and so on are given a chance to calculate or update their state for the next discrete time step". Gregory [28] also affirms for game outputs that they are "rendered by displaying graphics, emitting sound, and possibly producing other outputs such as force feedback on the joypad".

By the application of ZapML tags, game loop iterations can be controlled to interpret user inputs and provide game logic outputs, according to configured game engine routines. In this sense, *initialize game status*, *render game status*, *get player input* and *update game status* game routines were configured in a game cycle via ZapML to provide the IMgine game loop for TM games. *GameConfig* and *GameStatus* entities are also used to control current user inputs/outputs and the game end during the game loop.

The *initialize game status* routine is performed in two steps (Figure 3). The first, when the game is loaded after the player (*IMgineClient*) sends an initial message, *GameConfig* values are obtained to send a "splash" text or image about the game as a response. The second, before entering the game

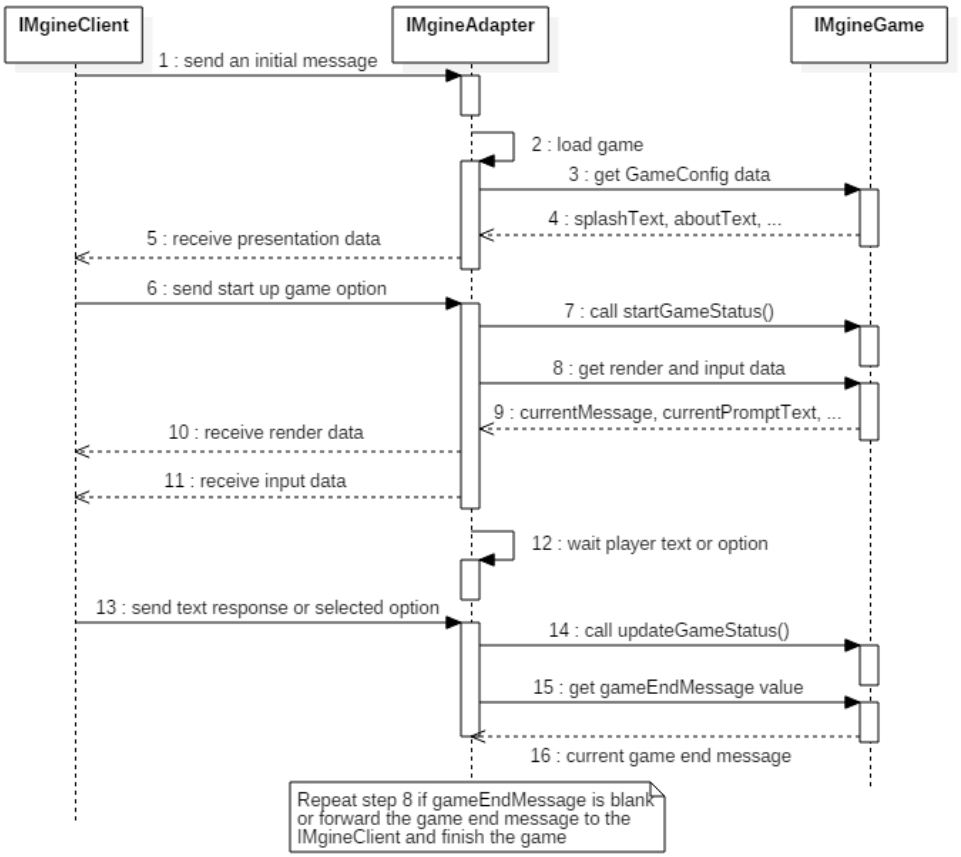

Figure 3. Sequence diagram of the IMgine game loop.

loop to start the game play, the *startGameStatus* function is executed to prepare initial *GameStatus* values.

Inside the game loop, the *render game status* routine verifies current *GameStatus* values and decides which IM content must be send to the player (Figure 3). By IM bot platforms, an IM game response can be performed via multimedia content types, such as *text*, *image*, *audio*, *video* and *document*. *GameStatus* fields, such as *message*, *imagePath*, *httpText*, and *httpImage*, indicate which content can be directly sent or loaded via *http* protocol to the player.

After rendering the current game status to the player, the *get player input* routine is performed (Figure 3). *GameStatus* fields, such as *promptText*, *menuText*, *currentOpt1*, *currentOpt2*, *currentOpt3*, and so on, indicate which prompt message or menu options must be sent to the player to start the interaction. *Prompt* and *Menu* tags verify if there is an initial content to be sent in the respective *GameStatus* field. If *true*, the prompt or menu content is sent to the player, asking a question or giving a current game option for the player interaction.

After player response, the *currentCommand* variable will receive the player input that could be a small text or a selected menu option to be processed by the *update game status* routine. This game routine executes the *updateGameStatus* function to perform game rules according to the developed IM game, and updates the *gameEndMessage* variable with the current game end message value (Figure 3). The *gameEndMessage* value is used to decide when the IMgine game loop must be finished or not. *Player score* and *highscore* data are also verified by this routine to include the current *player id* in the *Hall of Fame* after the game end.

C. Developing IMgine Games

GuessMyNumber and *BodyZap* [35] games were developed to validade the IMgine game loop. *GuessMyNumber* is a classic prompt game where the player must discover a random number in a range of possibilities, 1 to 1000 in this case. *BodyZap* follows the classic quiz genre with time limit, inviting the player to answer *Body* [36] questions to conquer a total of 10 organs of 2 organic systems in less than 10 minutes.

GuessMyNumber presents an initial menu where the player sends the "**S**" option to start the game play. The *startGameStatus* function defines an initial *promptText* mes-

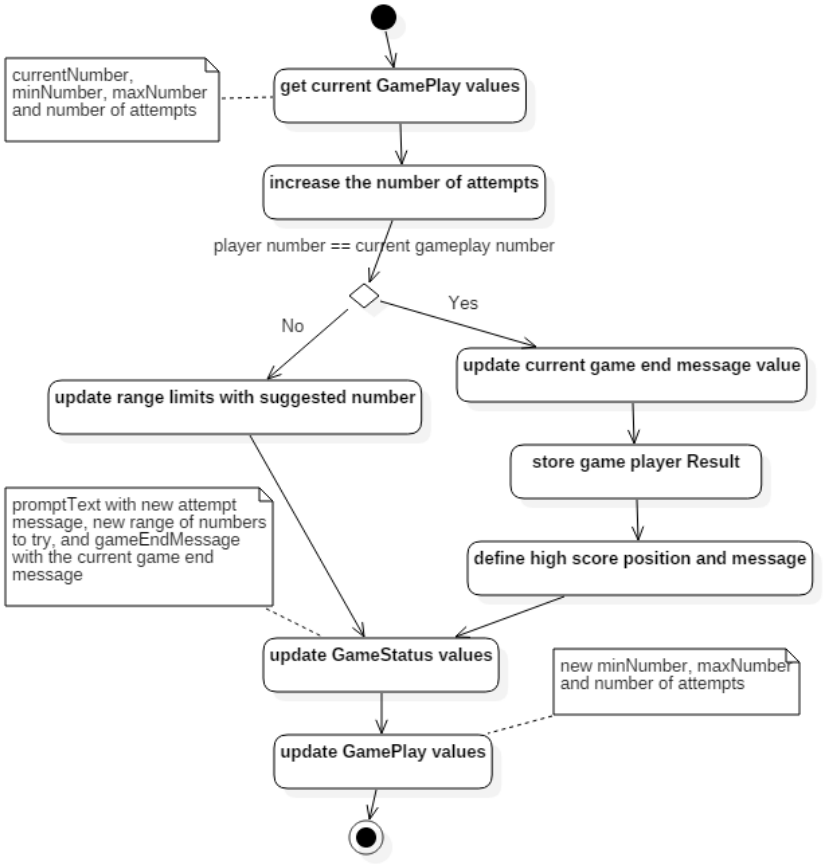

Figure 4. Activity diagram of the *GuessMyNumber updateGameStatus* function.

sage to the player, asking about a number between 1 and 1000, and set initial values to *currentNumber*, *minNumber* and *maxNumber* fields of the *GamePlay* entity.

For each player answer, *updateGameStatus* (Figure 4) gets the current *GamePlay* values and verifies if the player number is equals to the current gameplay number. If *true*, the current *gameEndMessage* will be updated, the game player result will be stored, and the high score position will be defined, together with an invitation message to put a name in the *Hall of Fame*. If *false*, the range limits (*minNumber* and *maxNumber*) will be redefined according to the player number. *GameStatus* and *GamePlay* entities will also be updated to show either a new player prompt with another range of numbers to guess, or the game end message congratulating on the discovery of the current gameplay number.

Regarding the *BodyZap* game, it presents an initial menu to the player with an initial option to start the game play. The *startGameStatus* function defines an initial message describing the time left to the player, and sets a "*timer*" value to the *GameStatus.statusId* field. This field is used to control the game actions that will be performed in each game cycle. Moreover, for *GamePlay* fields, the following values are initially applied: *startUp* is *now()*; zero *attempts* to the player; and *null* for the *lastCardId*. Previously *ConqueredOrgans* by the player are also removed before starting a new game.

After BodyZap initialization, for each player interaction, *updateGameStatus* (Figure 5) gets the current *GamePlay* and *ConqueredOrgans* values and verifies if the game time expired (10 minutes). If *true* for "time expired", a "*the time is up*" message with the total of *ConqueredOrgans* is prepared and sent back to the player, together with the *gameEndMessage*. If *false*, the current *GameStatus.statusId* value is verified ("*timer*", "*card*" or "*feedback*") which indicates the last performed operation in the game.

If the last performed operation is equals to "*timer*", a new question *Card* is obtained and shown to the player as a menu with the respective question text and answer options. *GameStatus.statusId* is also updated to "*card*", and

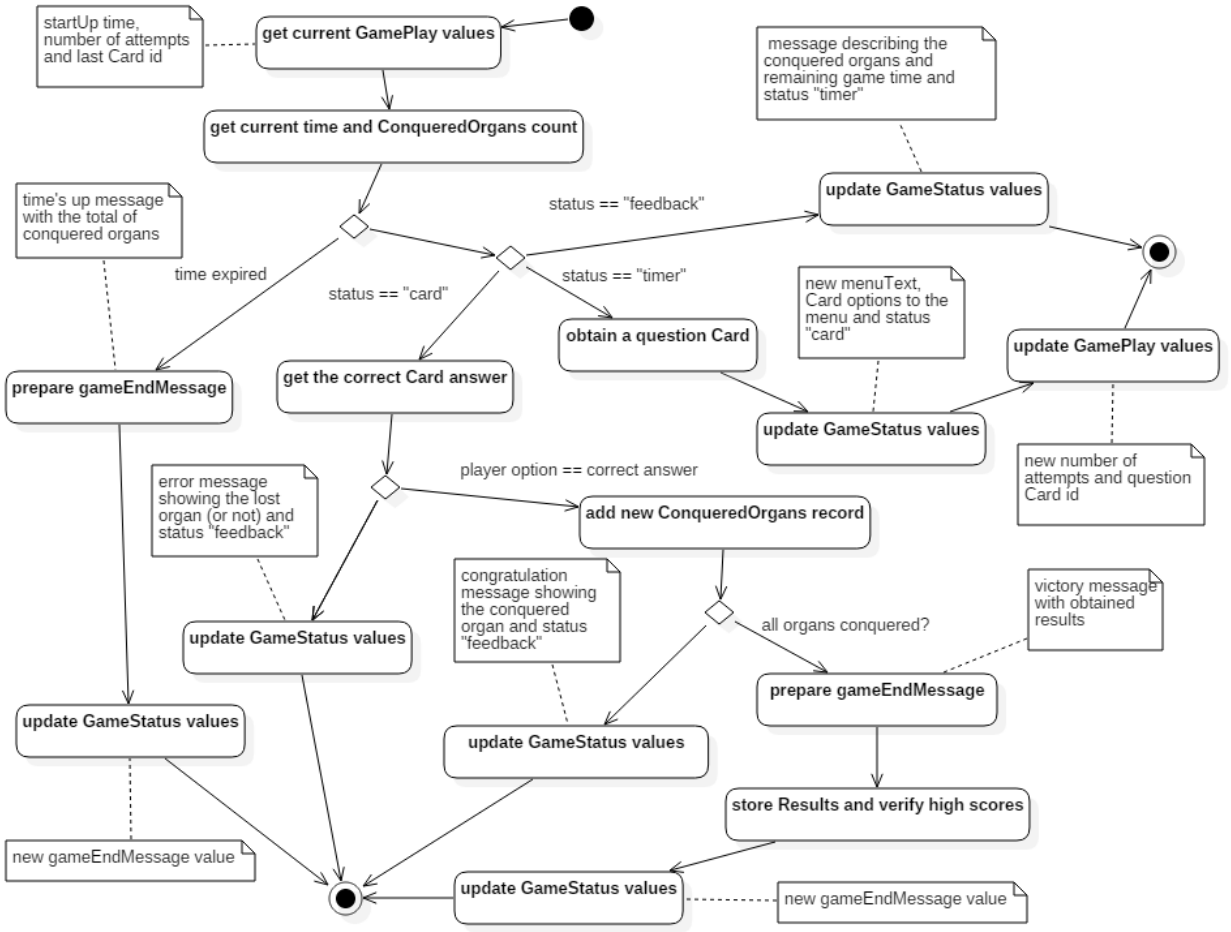

Figure 5. Activity diagram of the *BodyZap updateGameStatus* function.

the *idLastCard* field is referred to this new question *Card*.

If the last performed operation is equals to "*card*", the selected player option is compared with the correct answer of the *idLastCard* field. If the selected player option was a wrong answer, an error message showing the lost organ is sent to the player, together with the *GameStatus.statusId* update to "*feedback*". However, if the selected player option was a correct answer, a congratulation message showing a conquered organ is sent to the player, together with the *GameStatus.statusId* update to "*feedback*". Moreover, if all organs were conquered, a victory message with obtained results is sent to the player as a *gameEndMessage*.

Finally, if the last performed operation is equals to "*feedback*", a message describing the conquered organs and the time left to end the game is sent to the player, together with the *GameStatus.statusId* update to "*timer*" value. As a blank *prompt Text* and none menu option is defined in "*feedback*" state, the IMgine game loop: renders the time left message; skips the get player input operation; and immediately calls *updateGameStatus* again to continue the proposed game logic for the "*timer*" *GameStatus.statusId* value.

IV. Results and Discussion

Different approaches can be applied in TM games to provide TUI interactions for players, such as Mininel et al. [12] linking textual messages to on-screen sprites in developed games. For the IMgine game engine, it is based on SPLIMBo features that configure TUI templates, applying player interactions based on prompts and menus for user input, and textual and multimedia messages for user outputs.

Following this approach, IMgine games can be directly rendered by modern multi-IM platforms, such as Telegram and Facebook Messenger (Figure 6 and Figure 7), according to SPLIMBo resources able to interpret them. IMgine games can also be interpreted by alternative platforms according to developed *IMgineClients* that follow the proposed IM game loop and SPLIMBo input/output strategy, such as command-line and web interfaces for example (Figure 8 and Figure 9).

Regarding the development of TM games, they are often programmed to "dedicated" platforms without using a specific game engine or game reusable approach, such as the LibrasZap game loop directly configured via SPLIMBo resources [26]. By the IMgine game loop, there is a reusable and low-coupling way of developing the core of TM games, organizing the game logic modeling and deploying to distinct TUI platforms.

In this sense, Table I presents some obtained reuse metrics [37] for the amount of reused code and complexity of the *GuessMyNumber* and *BodyZap* games, according to Plato code analyzer [38] of the IMgine artifacts (Figure 10). As a result, more than 95% of SLOC reuse to *GuessMyNumber* and more than 90% of complexity reuse were obtained for both games, confirming the game core reusability and maintainability for IMgine games.

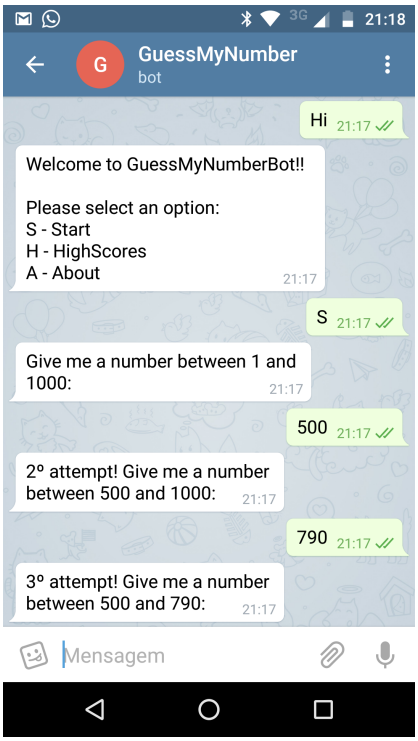

Figure 6. *GuessMyNumber* game running on Telegram platform.

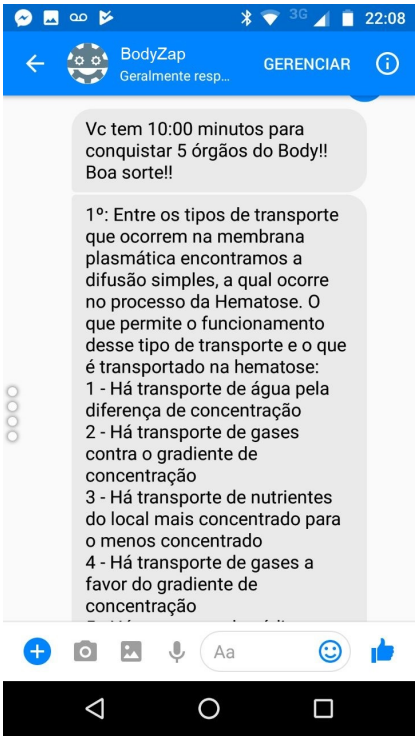

Figure 7. *BodyZap* startup with time left message and initial question running on Facebook Messenger platform.

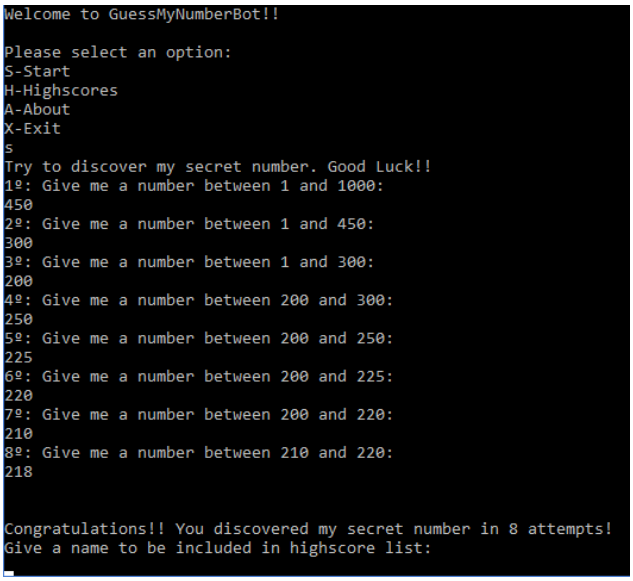

Figure 8. *GuessMyNumber* running on command-line interface.

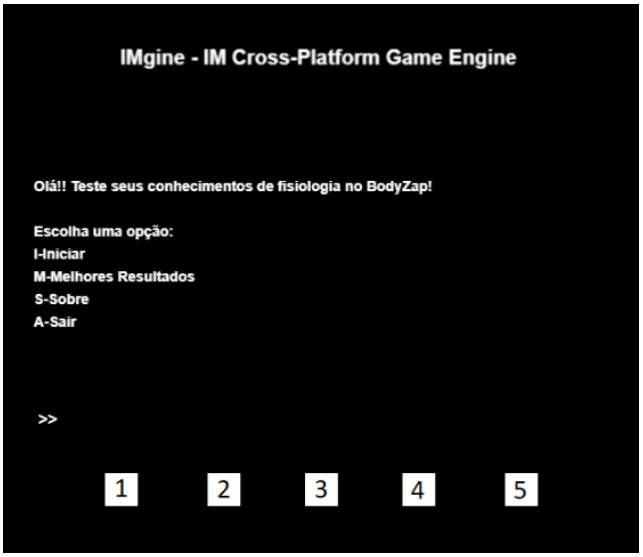

Figure 9. Initial menu of the *BodyZap* game running on Web interface.

Table I
REUSE METRICS OBTAINED BY *GuessMyNumber* AND *BodyZap* GAMES.

Game Name	Amount of SLOC / Total SLOC	Amount of Complexity / Total Complexity
GuessMyNumber	49/(49+1513) = 3,14%	6/(6+284) = 2,07%
BodyZap	889/(889+1513) = 37,01%	19/(19+284) = 6,27%

Finally, regarding available game engines in the IM context, some initiatives were developed in distinct game categories. They provided current TM game engines focused on dedicated platforms for HTML games [25] on legacy

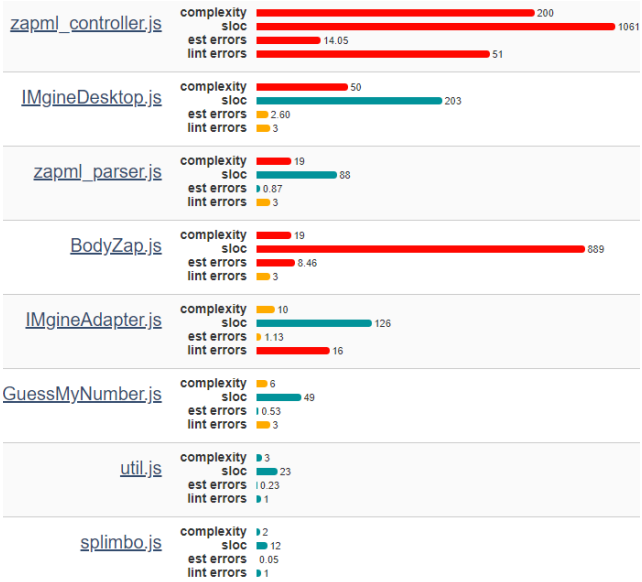

Figure 10. Complexity and SLOC results of the IMgine code analysis.

technologies such as SMS [21], and on cloud-paid services in proprietary code [9]. Table II summarizes a qualitative comparison among IMgine and these available TM game engines according to license, reusability, usage and multi-platform criteria. As a result, IMgine can be classified as an interesting solution to the production of TM games in a reusable and open source multi-IM perspective.

V. CONCLUSIONS AND FUTURE WORK

This paper presented IMgine, an open source cross-platform IM game engine solution for TM games. It defined open source resources to allow the IM communication for game user, TUI models for TM games, and a generic game loop able to perform an overall flow control according to multi-IM platforms. To this end, the SPLIMBo platform was chosen to cover the IM transmission process for multi-IM platforms, and to provide feature-based resources to design IM bots. Using SPLIMBo, game engine routines were defined via ZapML configurations to provide the IM game loop, and IM game states were represented by the realization of IMgine game core functions.

By the IMgine usage, it can be applied in the production of interactive text adventures with multimedia resources, following dynamic demands and interface restrictions of quiz games, providing the instantaneous communication of SMS messages, and reaching modern IM platforms currently used by the population. IMgine provides a foundation for IM games, simplifying the TM game design production to be concerned with the implementation of game functions that define the game core, instead of IM restrictions such as transmission process or gaming interaction limits. As a result, IMgine presents itself as an interesting solution to

Table II
QUALITATIVE COMPARISON AMONG IMGINE AND AVAILABLE GAME ENGINES IN THE IM CONTEXT.

Game Engine	License	Reusability	Usage	Multiplatform
PhoneAdventureGame [21]	Open Source	SMS wrapper	Local Hosting	No
Telegram Gaming Platform [25]	Proprietary	Telegram API for HTML games	Web Service	No
SCUNM [8]	Open Source	Adventure game routines for Telegram platform	Local Hosting	No
OnSequel [9]	Proprietary	Limited graphic flows	Cloud	Yes
IMgine	Open Source	*startGameStatus* and *updateGameStatus* functions	Local Hosting	Yes

provide classic TM games, as well as to produce new textual mechanics and dynamics on modern IM platforms.

As a proof of concept, two IM games were presented to validate the IMgine capability of producing IM games. These games explore the production of a gameplay using prompt-based and menu-based interactions, having success in representing their contexts and logic for quiz games over IM limitations. For verification and dissemination reasons, these game implementations are available to the open source community at https://github.com/vsarinho/IMgine.js, together with IMgine configurations and necessary SPLIMBo resources to reproduce and extend developed IM game services.

However, despite the IMgine benefits in the production of TM games, there are some possible problems with the IMgine adoption that can be described. Game engines, for example, often restrict the game design [39], and, although developed TM games are simple in most time, they may suffer at some point with the constraints imposed by the chosen engine. SPLIMBo adapters do not use dedicated resources from specific bot platforms to provide advanced IM interactions, such as in-line buttons and *webviews* [40]. The learning curve to use IMgine may not compensate it usage at some situations in comparison with the direct game development to a target IM platform via SPLIMBo or dedicated API. Only punctual games were implemented with IMgine, being necessary to improve the verification and validation process in a large-scale level at some moment. Finally, a quantitative comparison among IMgine and available open source engines for TM game development was not performed yet due to structural divergences among them, such as distinct programming languages and paradigms, developed for dedicated IM platforms, and the focus on a specific game genre/category.

As future work, a standard model for an initial TM game menu system [41], something important for specific game categories such as adventure games and RPG, will be provided. The support of multiplayer TM games, such as Telegram Game API and SMS competitions, is not available yet, being necessary to improve the proposed IMgine game loop to support this game style as soon as possible. Finally, the production of dedicated game engines for specific game domains, such as quiz (AsKIME), text-adventure (AdventIME) and board games (BoardIME), will be performed by the adaptation of *startGameStatus* and *updateGameStatus* functions to parse respective domain configuration. The idea is to provide a fast configuration of specific TM game categories and genres in a SPL strategy [27] extending the production possibilities of TM games in a write once, run anywhere perspective.

REFERENCES

[1] B. Danet, L. Ruedenberg-Wright, and Y. Rosenbaum-Tamari, "Hmmm where's that smoke coming from?" *Journal of Computer-Mediated Communication*, vol. 2, no. 4, p. JCMC246, 1997.

[2] C. Lebeuf, M.-A. Storey, and A. Zagalsky, "Software bots," *IEEE Software*, no. 1, pp. 18–23, 2018.

[3] I. Sommerville, *Software Engineering*. Addison-Wesley, 2007.

[4] C. Shchiglik, S. J. Barnes, and E. Scornavacca, "Mobile entertainment services: a study of consumer perceptions towards games delivered via the wireless application protocol," *International Journal of Services and Standards*, vol. 1, no. 2, pp. 155–171, 2004.

[5] R. Baskerville and J. Pries-Heje, "Short cycle time systems development," *Information Systems Journal*, vol. 14, no. 3, pp. 237–264, 2004.

[6] WereWolf, "A game engine for running werewolf in a chat client," https://github.com/hjwylde/werewolf, 2016.

[7] M. J. Wolf, *The medium of the video game*. University of Texas Press, 2001.

[8] SCUNM, "Script creation utility for nodejs maniacs," https://github.com/jlvaquero/SCUNM, 2017.

[9] OnSequel, "Create interactive games for messengers," https://www.onsequel.com/gamebot, 2016.

[10] 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.

[11] B. D. Ballentine, "Textual adventures: Writing and game development in the undergraduate classroom," *Computers and Composition*, vol. 37, pp. 31–43, 2015.

[12] S. Mininel, F. Vatta, S. Gaion, W. Ukovich, and M. P. Fanti, "A customizable game engine for mobile game-based learning," in *Systems, Man and Cybernetics, 2009. SMC 2009. IEEE International Conference on*. IEEE, 2009, pp. 2445–2450.

[13] L. Wolf, M. Knuth, J. Osterhoff, and H. Sack, "Risq! renowned individuals semantic quiz: a jeopardy like quiz game for ranking facts," in *Proceedings of the 7th International Conference on Semantic Systems*. ACM, 2011, pp. 71–78.

[14] C. Cheong, F. Cheong, and J. Filippou, "Quick quiz: A gamified approach for enhancing learning." in *PACIS*, 2013, p. 206.

[15] L. C. Klopfenstein and A. Bogliolo, "The quiz-master bot: a persistent augmented quiz delivered through online messaging," in *INTED2017 Proceedings (11th International Technology, Education and Development Conference). IATED*, 2017, pp. 9806–9811.

[16] A. d. S. e Silva, "Alien revolt (2005-2007): A case study of the first location-based mobile game in brazil," *IEEE Technology and Society Magazine*, vol. 27, no. 1, pp. 18–28, 2008.

[17] B. Bontchev, N. Gabarev, and H. Pavlov, "A mobile chess game," in *Proceedings of 16th SAER Conference*, 2002, pp. 138–143.

[18] E. Marcote, D. I. Iglesia, and C. J. Escudero, "An external short message entity for gambling services," in *Proceedings of the 2004 ACM SIGPLAN workshop on Erlang*. ACM, 2004, pp. 27–32.

[19] M. Flintham, G. Giannachi, S. Benford, and M. Adams, "Day of the figurines: Supporting episodic storytelling on mobile phones," in *International Conference on Virtual Storytelling*. Springer, 2007, pp. 167–175.

[20] P. Olla, N. Patel, and C. Atkinson, "A case study of mmo2s madic: a framework for creating mobile internet systems," *Internet Research*, vol. 13, no. 4, pp. 311–321, 2003.

[21] P. Collins, "Twilio powered adventure game," http://saveandexit.com/twilio-powered-adventure-game/, 2013.

[22] GameOn, "Hello, stranger!" https://itunes.apple.com/us/app/hello-stranger/id1048613928?mt=8, 2016.

[23] V. T. Sarinho, "Libraszap-an instant messaging game for knowledge assessment in brazilian sign language," *Brazilian Journal of Computers in Education*, vol. 25, no. 01, p. 44, 2017.

[24] L. C. Klopfenstein, S. Delpriori, B. D. Paolini, and A. Bogliolo, "Code hunting games: a mixed reality multiplayer treasure hunt through a conversational interface," in *International Conference on Internet Science*. Springer, 2017, pp. 189–200.

[25] Telegram, "Gaming platform 1.0," https://telegram.org/blog/games, 2016.

[26] V. T. Sarinho, "Splimbo–developing and evaluating a software product line for cross-platform im bots," *Journal on Advances in Theoretical and Applied Informatics*, vol. 3, no. 2, pp. 18–23, 2017.

[27] P. Clements and L. Northrop, *Software product lines: practices and patterns*. Addison-Wesley Reading, 2002, vol. 3.

[28] J. Gregory, *Game engine architecture*. AK Peters/CRC Press, 2014.

[29] M. Lewis and J. Jacobson, "Game engines in scientific research," in *Communications of the ACM*, vol. 45(1), 2002, p. 21.

[30] S. Madhav, *Game Programming Algorithms and Techniques: A Platform-Agnostic Approach*. Pearson Education, 2014.

[31] R. Nystrom, *Game programming patterns*. Genever Benning, 2014.

[32] D. Sanchez-Crespo and D. S.-C. Dalmau, *Core techniques and algorithms in game programming*. New Riders, 2004.

[33] A. LaMothe, *Tricks of the Windows game programming gurus*. Sams Publishing, 2002.

[34] M. Sicart, "Defining game mechanics," *Game Studies*, vol. 8, no. 2, 2008.

[35] V. T. Sarinho, E. M. Granjeiro, and C. O. Cerqueira, "Bodyzap: Um jogo de im para o ensino de fisiologia humana," in *II Workshop de Jogos e Saude, XVI Brazilian Symposium on Games and Digital Entertainment (SBGAMES)*. SBC, 2017.

[36] G. A. Borges, C. O. C. Lima, E. A. Granjeiro, V. T. Sarinho, and R. A. Bittencourt, "Body: Um jogo digital educacional de tabuleiro na area de fisiologia humana," in *XV Brazilian Symposium on Games and Digital Entertainment (SBGAMES)*. SBC, 2016, pp. 412–420.

[37] W. Frakes and C. Terry, "Software reuse: metrics and models," *ACM Computing Surveys (CSUR)*, vol. 28, no. 2, pp. 415–435, 1996.

[38] Plato, "Javascript source code visualization, static analysis, and complexity tool," https://github.com/es-analysis/plato, 2012.

[39] A. BinSubaih and S. Maddock, "Game portability using a service-oriented approach," *International Journal of Computer Games Technology*, vol. 2008, p. 3, 2008.

[40] A. Shevat, *Designing bots: Creating conversational experiences*. " O'Reilly Media, Inc.", 2017.

[41] B. Fox, *Game interface design*. Thomson course technology Boston, MA, 2005.