AsKME: A Feature-Based Approach to Develop Multiplatform Quiz 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 approaches have been proposed to manage the game domain variability in different instances and strategies. However, the idea of an one-size-fits-all game architecture can be misleading, being necessary to built reference game architectures for target (sub)domains. This paper presents the *Assessment of Knownledge Multiplatform Environment* (AsKME), a feature-based approach to develop multiplatform quiz games. It provides a subdomain game architecture, based on identified features of the quiz game dimension, in a Model-View-Controller strategy implemented by feature artifacts adapted to be executed in distinct software platforms. As a result, a reusable approach to develop multiplatform quiz games was provided, together with the development of educational and casual quiz games for validation purposes.**

***Keywords*-game domain variability; feature modeling; quiz games; multiplatform game environment;**

I. Introduction

Nowadays, computer games represent "the quintessential domain for Computer Science and Software Engineering (SE) research and development" [1]. It is the result of the current demand for technical mastery and integration skills by different software system application arenas during the game production [1]. As a consequence, many traditional grand challenges in SE arise during the development of computer games as complex software systems, such as large scale software engineering, game software requirements engineering, game software design, global game development, and so on [1].

Regarding requirements engineering, game development focuses primary attention on creating and satisfying non-functional requirements (NFRs), like "the game must be fun to play" by a target audience of users/players on a target platform at some retail price point or monetization scheme [2]. As a consequence, the SE challenge appears in determining what to do during game software development to address or satisfy such NFRs as engineering tasks [1].

A common solution is the production of games that can be incrementally developed and released with a minimum set of game play *features* that can adaptively be grown to meet informal NFRs [1]. This approach provides the game system as an emerging online interactive gameplay service, determining whether more *features* will be dynamically added or integrated to the game, rather than just as a traditional software product [1].

However, as game development is not the same as software development, traditional requirements engineering is also not applicable [3] to determine a suitable set of *features* in a game. In this sense, it is necessary to follow a game development approach able to prescribe game *features* as subject to functional or non-functional game requirements.

Regarding game design challenges, as game designers are directed to employ a game SDK or development framework to realize their game projects, the selection of a game development environment constrains or pre-determines what kind of computer game may be more readily developed [1]. As a result, the game development environment encroaches into the functional requirements or NFRs space [1], something that can be avoided with the separation of the *core* of game objects (*G-factor)* from the implementation itself in order to support the game portability among game development environments [4].

Considering game development environments, understand game engine architecture, interaction paradigm, and programming peculiarities usually is not a simple or intuitive task [3], being necessary the expertise of an architect or senior developer to map the requirements of each product *variant* onto the framework [5]. Language-based tools can automate this mapping step, capturing *variations* in requirements via language expression and encapsulating the abstractions that a game engine defines, but without the flexibility level provided by game engines for example [3]. Therefore, there is a hiatus on game development caused by a lack of simple yet powerful tools to provide abstraction for specific game domains along with the flexibility of game engines or other reusable asset according to designed game *variants* [3].

Features, *variants* and the *core communality* terms are managed by the *software variability* area [6]. Per *features*, they can be defined as logical units of behaviour that corresponds to a set of functional and non-functional requirements in a system [7], and are an interesting "way to abstract from requirements" [8]. *Core communality* and *variant* terms are worked by the *variability management*, which controls the points in the platform where the product version functionalities differ [6].

Several approaches have been proposed to manage the game domain variability in different instances and strategies

[9]. They provided interesting types of: game design and modelling languages; game software modelling languages; game models and meta-models; and game software models, frameworks, environments, and technologies [9]. However, the idea of an one-size-fits-all game architecture can be misleading, being necessary to built reference game architectures for target (sub)domains [3]. Moreover, the popular concept of "game genres" can be confuse in a variability process [3] because they are ambiguous and imprecise [10]. Therefore, it is necessary to describe expectations for predefined core game dimensions, such as player, graphics, flow, and other representative game features [3].

This paper presents the *Assessment of Knownledge Multiplatform Environment* (AsKME), a feature-based approach to develop multiplatform quiz games. The idea is provide a subdomain game architecture, based on identified features of the quiz game dimension, in a Model-View-Controller (MVC) strategy implemented by feature artifacts adapted to be executed in distinct software platforms, such as console, mobile, web, embedded systems, an so on.

To this end, section 2 presents related work on game domain variability and available quiz models. Section 3 describes the proposed feature model along with the applied methodology to perform configured features in different execution platforms. Section 4 presents the developed and obtained AsKME games, together with a qualitative and quantitative analysis of the developed assets. Finally, section 5 presents the conclusions and future work of this paper.

II. Related Work

A. Game Domain Variability

For computer games, the variability is a direct consequence of the game domain diversity, working with simulations (sports, adventure, fighting), hardware technologies (mobile games, web games), human interactions (immersion, multiplayer) and complex stories (games based on movies, Role-Playing Game (RPG) series) [11].

Considering the communalities and variabilities in the digital game domain represented by features, the "Narrative, Entertainment, Simulation and Interaction" (NESI) feature model is an attempt of representing the G-factor according to game concepts found in the literature [12]. The "GameSystem, DecisionSupport and SceneView" (GDS) feature model describes generic configurations and behavioral aspects found on game implementation resources identified in the literature [13]. The Feature-based Environment for Digital Games (FEnDiGa) is a game production environment based on a combined representation of NESI and GDS features [14] via Object Oriented Feature Modeling (OOFM) approach [15]. Finally, the Minimal Engine for Digital Games (MEnDiGa) is an extensible collection of representative classes, based on a simplified set of NESI and GDS features, that can be used as the foundation for small and casual games without major modifications [16].

Regarding game ontologies, the Game Ontology Project (GOP) is a framework that defines a hierarchy of ontology concepts for the game domain, such as interface, rules, goals, entities and entity manipulation [17]. SharpLudus proposed the use of a game ontology to set ad hoc aspects identified for a game 2D implementation [18]. The DGiovanni project defines an ontology to support the creation of different stories in an open source multiagent architecture for building interactive dramas [19]. Finally, the PerGO project proposed an ontology to structure and accelerate the domain analysis process on the emerging pervasive (computer) game genre [20].

Finally, by modeling engineering approaches, ArcadEx represents an improvement of the SharpLudus project, replacing the previous ad hoc approach with a Software Product Line (SPL) realized by a Domain Specific Language (DSL) in a Domain-Specific Game Development process [3]. The Serious Games Modelling Environment (SeGMEnt) is based on a model-driven approach to aid non-technical domain experts in serious games production for use in games-based learning [21]. Finally, the Model-Driven Game Development (MDGD) introduces the use of a selection of UML diagrams to gather required information to automate generation of code for 2D platform games [22].

B. Quiz Models

Quiz can be considered as one of the simplest types of game domain, which describes game dynamics that can be directly reused by other types of game subdomains. Quiz is a great way to make self assessment tests or final exams [23], which the main objective is the successful answering of questions [24]. In this sense, there are several frameworks and tools able to provide educational quizzes via automatic generation of questions, automatic assessment of answers and parameterized questions [23], but without a communality and variability representation of them.

Regarding quiz modelling, the "Generalized Platform for Creating of Testing Games" is a virtual platform where teachers can define different structures to follow the students progress [25]. This platform defined limited quizzes only with an *id*, a *question* and at least one *answer* that can be from different types (*boolean*, *string*, *integer* or any other complex type previously defined) [25].

For quiz feature modeling, Quiz Product Line (QPL) presented mandatory and optional features in a *System* perspective for quiz applications, such as: *Questions*, *Layout*, *License*, *Report Generator*, *Operation Mode*, *Question Editor*, *Quiz Question Generator*, *Quiz Utilities*, and *Publish* [26]. As a result, it defined an interesting representation of quizzes, but without a concrete structure to perform them.

Regarding Model-Driven Engineering (MDE), a collection of metaclasses were proposed to formally represents the possible variations among the elements in multi-platform

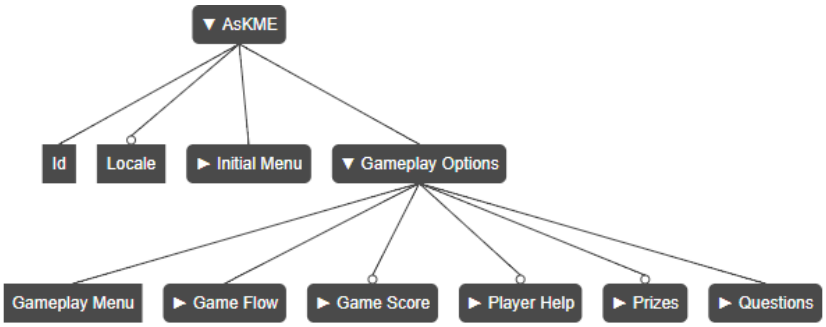

Figure 1. Partial illustration of the AsKME feature model.

videogames, such as trivia, platform, puzzle, touch and strategy [27]. For trivia games, they are generically represented by a collection of one or more *Questions* that contains zero or more game *Elements* each one [27].

Finally, considering the modern trend of presentation and control of questionnaires, the Quiz Board Game Model proposed a formal model of board games able to represent quiz dynamics with multimedia appealing [23].

III. Methodology

The AsKME development process was divided in 3 main parts: 1) the design of game features for the quiz game dimension; 2) the definition of a game development approach to represent and interpret AsKME features; and 3) the implementation of multiplatform game clients able to execute AsKME feature configurations.

A. Features for the Quiz Game Dimension

The concept of a feature is useful for the description of communalities and variabilities in the analysis, design, and implementation of software systems [28]. Feature modeling is a method and notation to elicit and represent common and variable features of the systems in a system family [29]. Together, features and feature models are able to represent communality and variability aspects of computer games and their subdomains [13].

The AsKME project has the objective of managing communalities and variabilities of the quiz game subdomain. In this sense, 98 features were modeled to represent system families of quiz games (Figure 1). These features were organized to identify the proposed game (*Id* and *Locale*), to represent *Initial Menu* data for user interactions (*Start Menu*, *Highscore Menu*, *About Menu*, etc.) and to define gameplay options for game configurations (*Game Menu*, *Game Flow*, *Game Score*, *Player Help*, *Prizes* and *Questions*) (Figure 1).

Six main features define the *Game Flow* of an AsKME game: *Conditions To Win*, *Conditions To Lose*, *Time Counter Scale*, *Turn Time Limit*, *Messages*, and *Performed By Event* (Figure 2). *Conditions To Win* and *Conditions To Lose*

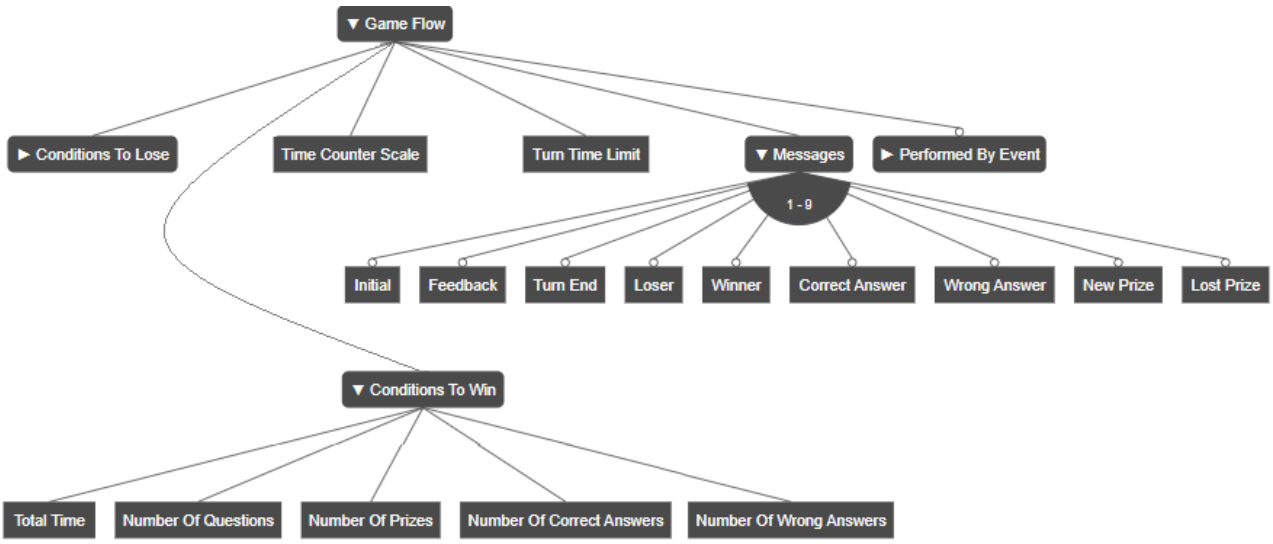

Figure 2. AsKME *Game Flow* subfeatures.

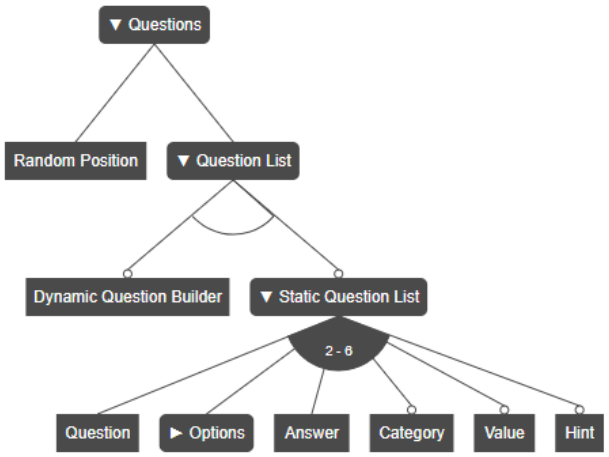

Figure 3. AsKME *Questions* subfeatures.

share a similar set of features, describing the total time and the number of questions along with correct answers and wrong answers obtained during a gameplay. If one of these conditions were achieved, the respective *Messages* are sent to the player according to configured game dynamics. *Time Counter Scale* describe the timer pattern to be shown during the gameplay, and *Turn Time Limit* defines the time the player has to answer the question before losing his turn. Finally, *Performed By Event* provides game events that can be evaluated via scripts in the construction of extra game dynamics, such as increase player score with an extra value due to a correct answer in the last game minute.

Questions feature defines a collection of questions that an AsKME game should present to the player (Figure 3). These questions can be statically defined in a *Static Question List* or dynamically provided by a *Dynamic Question Builder*. Regardless of the production approach, each question must present the text of the *Question* itself, *Options* to be chosen by the player and the indication of which of the provided

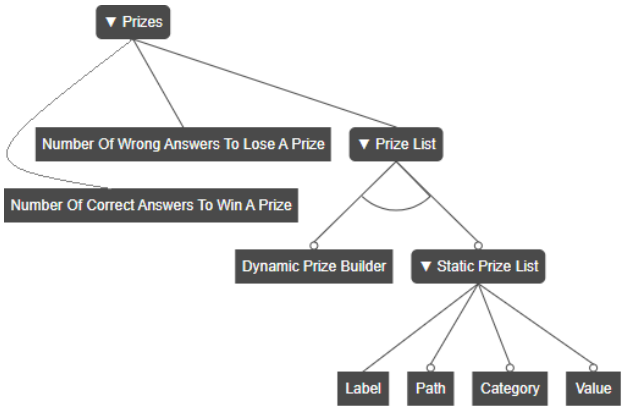

Figure 4. AsKME *Prizes* subfeatures.

options is correct (the *Answer* feature). *Category*, *Value* and *Hint* are extra features that can be added to each produced question in order to increase the game dynamics provided by the modeled AsKME game. Finally, the *Random Position* indicates whether the set of options should be scrambled or not before they are presented to the player.

Prizes feature represents possible trophies that a player can win during the gameplay. For this, it is necessary to indicate the *Number of Correct Answers to Win a Prize* and the *Number of Wrong Questions to Lose a Prize*. The *Prize List* is also provided by static and dynamic approaches, both indicating the *Label* of the prize, a *Path* for multimidia content that represents the prize, possible *Category* to indicate the prize group, and a current *Value* to increase the player score.

Player Help is a feature that describes possible aids that a player can have in the game (Figure 5). Three types of aid were initially represented: *Hint*, *Exclusion* and *Jump*. All three define a menu representation for user selection and the total number of attempts available for use. *Exclusion* also defines the *Max Options to Left After Exclusion* in a current *Question*, indicating the final number of *Options* available for a player after an *Exclusion* usage. Finally, *Menu Help* indicates player interaction messages to be used in a configured game, and *Number of Correct Answers to Win a Help* describes itself.

B. Game Development with AsKME Features

Feature is a unit of functionality of a software system that satisfies a requirement, represents a design decision, and provides a potential configuration option to be composed as a software system [28]. Feature-Oriented Software Development (FOSD) is a paradigm for the construction, customization, and synthesis of large-scale software systems in terms of features [28].

Different FOSD approaches have been applied with success to provide digital games, such as generative programming [13] and SPL instance [3]. For AsKME games, a FOSD strategy was applied by the definition of a DSL for AsKME features along with a generic and multiplatform game loop proposal able to perform them.

Regarding DSL, it provides pre-defined abstractions to directly represent concepts from the application domain [29]. It also offers a natural notation for a given domain and avoids syntactic clutter that often results when using a general purpose language [29].

By the AsKME feature model, a DSL was defined via JavaScript Object Notation (JSON), where each feature was used to directly structure and represent boolean (*Random Position*), numeric (*Turn Time Limit*), textual (*Start Menu*) or procedural (*Performed By Event*) values applied in an AsKME game configuration. As an example, Figure 6 illustrates the JSON representation of a partial configuration

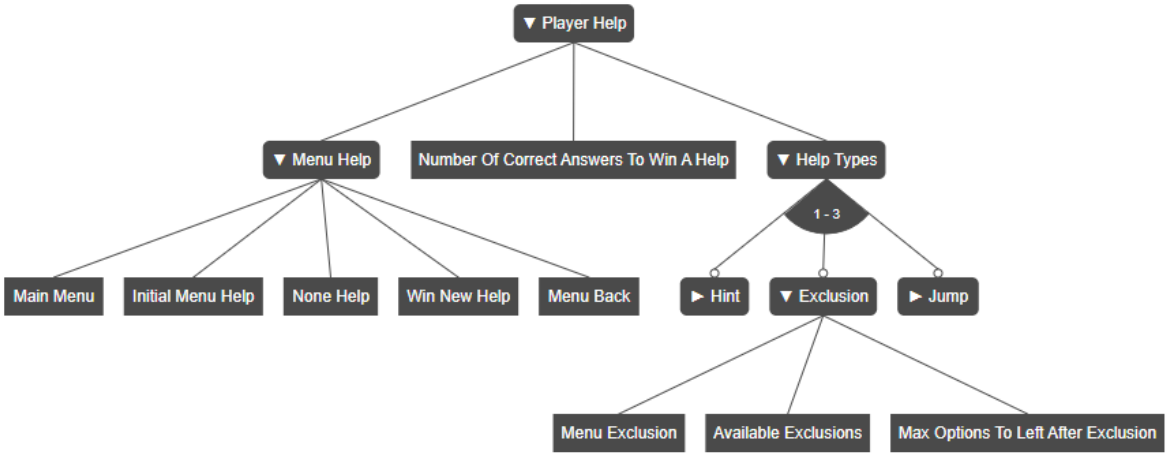

Figure 5. AsKME *Player Help* subfeatures.

of *Gameplay Menu* and *Game Flow* features of an AsKME game.

Regarding game loop, it is 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" [30]. Moreover, 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" [31].

For the proposed multiplatform game loop to perform AsKME features, a Javascript state machine was implemented, capable of interpreting the provided JSON configurations according to player inputs and game logic outputs (Figure 7). As a resumed explanation, the proposed state machine is able to:

- show the list of available AsKME games (*game-list* and *start-menu* states);
- provide a start menu for a selected game (*start-game*, *about-game*, *highscore-game* and *mode-list* states);
- perform game routines (*startGameState* and *up-*

```
gamePlayOptions:[{
  gamePlayMenuOption:"T", gamePlayMenuText:"Jogar por Tempo",
  gameFlow: {
    conditionsToWin: {totalTime:0, numberOfQuestions:0, numberOfPrizes:5, numberOfCorrectAnswers:0, numberOfWrong...
    conditionsToLose: {totalTime:600, numberOfQuestions:0, numberOfPrizes:0, numberOfCorrectAnswers:0, numberOfWr...
    timeCounterScale:"min" /* min, secs */, turnTimeLimit:60,
    initialMessage:["true","Vc tem 10:00 minutos para conquistar 5 órgãos do Body!! Boa sorte!!"],
    feedbackMessage:["System._prizeCounter != 1","Restam System._timeCounter minutos e System._prizeCounter órgão...
    turnEndMessage:["true","Desculpe, mas seu tempo limite para responder a questão acabou!"],
    loserEndMessage:["System._totalPrizes != 1","Que pena, o tempo acabou!! Vc conquistou System._totalPrizes órg...
    winnerEndMessage:["true","Parabéns!! Vc conquistou o Body faltando System._timeCounter minutos após System._q...
    correctMessage: ["true","Parabéns, vc acertou!!"],
    newPrizeMessage: ["true","Como prêmio vc conquistou o órgão:System._newPrize."],
    errorMessage:["true","Que pena, vc errou!! A resposta correta é: System._answer."],
    lostPrizeMessage:["true","Como punição vc perdeu o órgão: System._lostPrize."],

    performEvent: function(gs, e){
        // possible events: game-win, game-lose, turn-end, wrong-answer, correct-answer, new-prize, lost-prize
        if (e == "new-prize"){
            gs._score += 100;
        }
        else if (e == "lost-prize"){
            gs._score -= 100;
        }
        else if (e == "game-win"){
            gs._score += (600 - gs._totalTime) * 2;
        }
    }
}, ...
```

Figure 6. JSON representation of a partial configuration of an AsKME game.

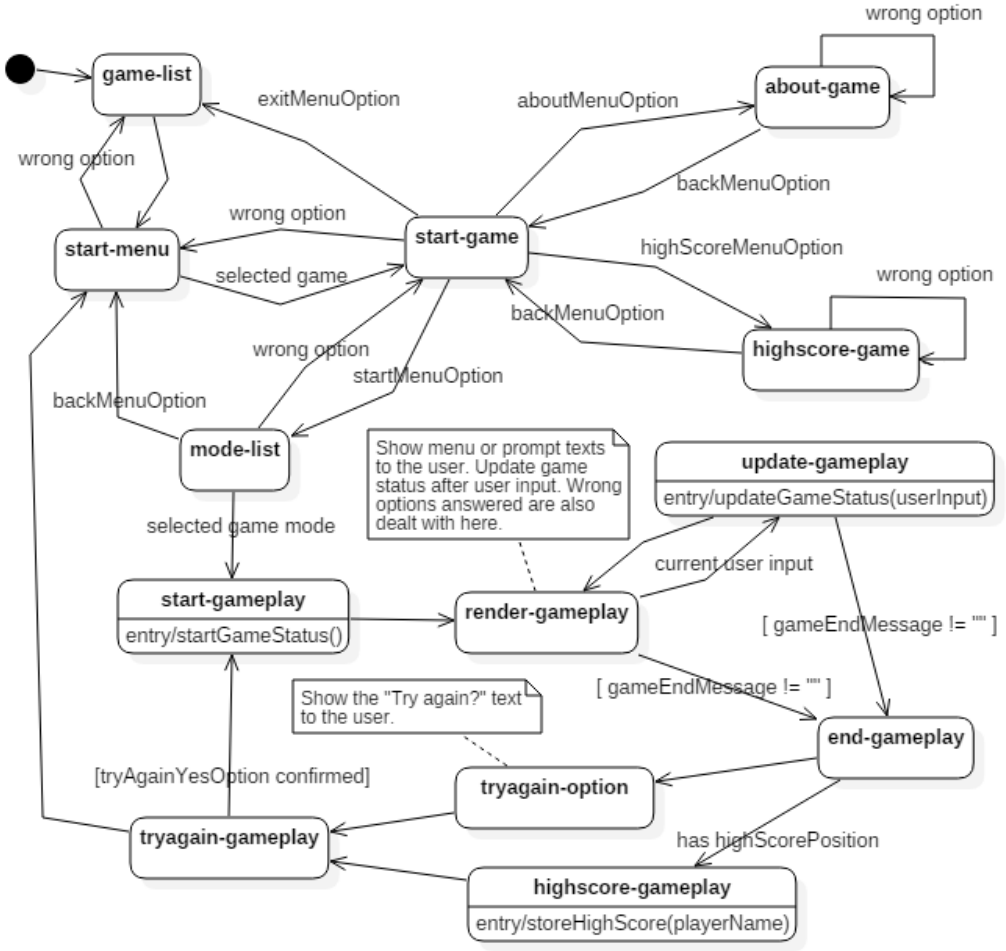

Figure 7. Proposed multiplatform game loop state machine to perform AsKME configurations.

dateGameState) according to a game loop execution (*start-gameplay*, *render-gameplay*, *update-gameplay* and *end-gameplay* states); and

- finalize the game with a high score record along with the verification of another attempt to play (*tryagain-option*, *highscore-gameplay* and *tryagain-gameplay* states).

The *updateGameState* routine also represents another substate machine that is performed by each game loop interaction among *render-gameplay* and *update-gameplay* states (Figure 8). For each interaction, it uses the last user input and decides for the next internal state to be evaluated (*show-question*, *evaluate-answer*, *perform-help* or *feedback*) according to *Gameplay Options* feature values.

For example, when the *show-question* state is activated, it prepares the current question to be shown to the user, defines the internal state to be *evaluate-answer*, and wait for another game loop interaction with a new user input to be performed. The *evaluate-answer* state verifies for turn end, help menu selection, correct answer and victory conditions according to current user input. If the help menu option is selected, a help menu is sent to the player via *showHelpMenu* procedure, and the *perform-help* state will be activated to wait for the selected help option in order to provide a new question to be evaluated by the player.

For an identified correct/wrong answer or a turn end situation, the *feedback* state is activated to show a message content for the player. *Feedback* also demands the processing of a loop interaction between *render-gameplay* and *update-gameplay* states (Figure 7), by the definition of an empty prompt or menu for the player that forces the *update-gameplay* state to start an interaction search for a menu or prompt information to be displayed to the player.

Finally, if conditions to win or to lose are identified in the *evaluate-answer* state, the *gameEndMessage* value is applied to the current gameplay, and the interaction among *render-gameplay* and *update-gameplay* states follows to the *game-end* status (Figure 7).

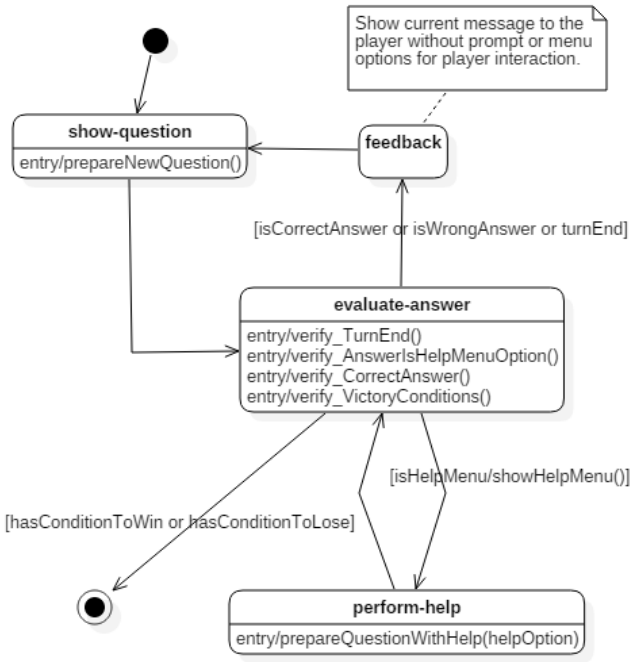

Figure 8. The *updateGameState* routine represented as a state machine.

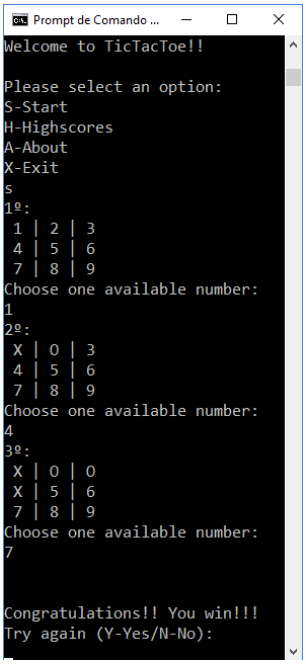

Figure 9. Console AsKME client running on the *TicTacToe* game.

C. Multiplatform AsKME Clients

Multiplatform development tools are gaining a worldwide popularity due to their characteristic to compile the application source code for various supported platforms, especially for mobile operating systems [32] [33]. Feature artifacts encapsulate design and implementation information of FOSD phases, allowing an almost automatically generation of applications due to the clean mapping with representative features of the domain knowledge [28]. AsKME feature artifacts were implemented with the goal of achieving multiplatform targets, such as console, mobile, web and Arduino. They follow the proposed JSON structure (Model) and the state machine game loop (Controller) in order to provide AsKME clients (View) for configured AsKME features.

Regarding the console AsKME client, it was implemented using *Node.js*, a JavaScript runtime built on Chrome V8 JavaScript engine. This client version gets user inputs using the *scanf* function, and provides player outputs via *console.log* function. Because it is a textual interface, no configured multimedia content could be processed. As an example, Figure 9 illustrates the console AsKME client running on the *TicTacToe* game.

The mobile AsKME client was implemented using the *Ionic Framework*, a platform that lets web developers to build, test, deploy and monitor cross-platform apps. This client version gets user inputs from various Ionic components such as text fields, *Alert Dialogs* and buttons. The outputs are displayed on three different sections of the screen, organized vertically: the top section displays text messages and questions; the middle section displays multimedia content (when available) through *HTML* tags, such as *video*; and the bottom section displays either a list of options, for multiple choice questions, or a text input field for questions with more open answers. Game end messages and Highscore messages are presented by Alert Dialogs, asking the player if the game should be restarted or a name to be included in the Highscores list. As an example, Figure 10 illustrates the mobile AsKME client running on the *BodyZap* [34] game.

The web AsKME client was developed using *Phaser*, an open source framework for *Canvas* and *WebGL* powered browser games. This client version gets user inputs from browser events from keyboard and mouse clicks on fixed buttons over the screen (Figure 11). Outputs are provided by two sections that divide the screen, the left for textual messages and the right for multimidia content such as image and video (Figure 11). Sound effects are also performed following game events, such as correct and wrong answers, the game start, and so on. As an example, Figure 11 illustrates the web AsKME client running on the *LibrasZap* [35] game.

Finally, for the Arduino AsKME client, it was implemented using Johnny-Five, a JavaScript platform for robotics and Internet of Things (IoT) projects. This client version gets user inputs using an infrared sensor, and provides player outputs using a LCD screen. The infrared code seeks to decode the analog signals in hexadecimal, and with these values inform which buttons are being pressed from an

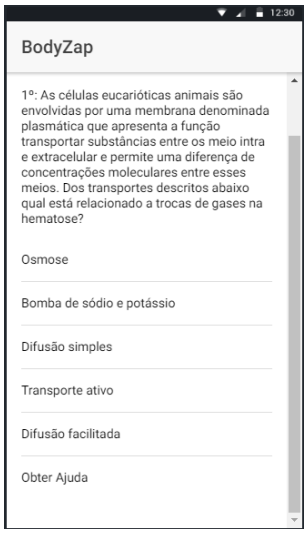

Figure 10. Mobile AsKME client running on the *BodyZap* game.

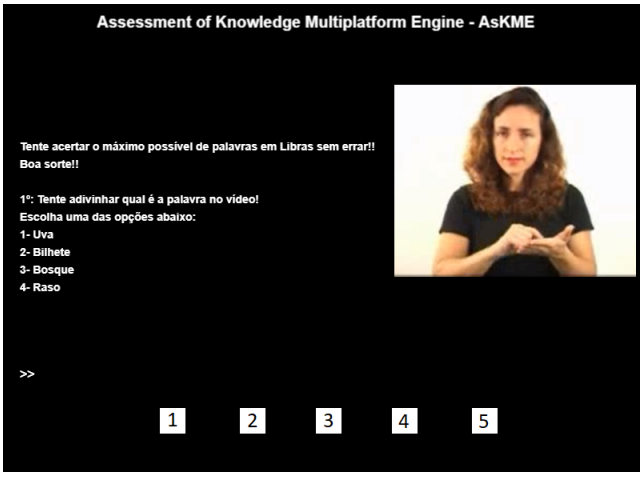

Figure 11. Web AsKME client running on the *LibrasZap* game.

infrared control. The LCD screen code directly render colors and letters in screen positions on a matrix form (columns and rows), using a buffered strategy to avoid screen flashes due to the main Arduino processing loop. As an example, Figure 12 illustrates the Arduino AsKME client running on the *GuessMyNumber* game.

IV. Results and Discussion

Some quiz games were developed for AsKME evaluation purposes (Table I). Each one presented specific characteristics and needs during the development, such as the interface style applied to the user (menu-based or prompt-based), static or dynamic approaches to the production of game questions, and the processing of particular events during the gameplay to guarantee designed game dynamics. As a result, basic games that follow the quiz style have

Figure 12. Arduino AsKME client running on the *GuessMyNumber* game.

been successfully developed, as well as the replication of configured game dynamic for distinct educational contexts, such as human physiology (BodyZap), sign-based languages (LibrasZap) and software engineering (ERQuiz). The production of board games and storytelling games with AsKME also demonstrates the capability of quiz games to provide game dynamics from other game styles, but in this case with minimal interaction resources and following a question & answer game mechanics approach.

Regarding the reuse level achieved with AsKME games, Table II presents some obtained metrics [36] with the Plato code analyzer [37] for the amount of reused code and complexity of each developed game. The Total of SLOC and Total of Complexity metrics are calculated by the sum of the respective AsKME game metrics with the console AsKME client metrics. As a result, more than 88% of SLOC reuse and more than 93% of complexity reuse in average were obtained for them, confirming the game core reusability and maintainability for developed AsKME games. However, it is possible to verify that the complexity of AsKME games increases when they use a dynamic approach to provide player questions, something that can be interpreted as a demand for more representative features for dynamic quizzes in the AsKME feature model.

V. Conclusions and Future Work

This paper presented AsKME, an assessment of knowledge solution to provide quiz games for multiplatform environments. It defined a feature model to represent the game quiz subdomain, along with feature artifacts able to perform quiz game dynamics by AsKME clients in multiplatform environments.

By the AsKME usage, the production of static and dynamic quiz games was guaranteed, following predefined

Table I
CURRENT LIST OF DEVELOPED ASKME GAMES.

Name	Description	User Interface	Question Builder	Perform By Event
GuessMyNumber	A game to find out the secret number in a range of 1 to 1000 in the fewest guesses	Prompt-based	Dynamic questions based on *min* and *max* range of values	Score calculation by player attempts
BodyZap [34]	Educational quiz for assessment of knowledge in human physiology	Menu-based	Static questions	Score calculation by consumed time, new prizes and lost prizes
LibrasZap [35]	Educational quiz for assessing knowledge in the Brazilian sign language (LIBRAS)	Menu-based	Dynamic production of questions based on a list of available videos	Score calculated by player attempts
ERQuiz	Educational quiz for assessing knowledge in requirements engineering	Menu-based	Static questions	Score calculated by the sum of correct question values
TicTacToe	Paper-and-pencil game for two players, X and O, who take turns marking the spaces in a 3x3 grid	Prompt-based	Dynamic questions showing the available grid options to choose	Definition of computer movement after the player choice
Blackjack	Comparing card game between the player and a dealer	Menu-based	Dynamic questions showing the player and the dealer hands, together with Hit or Stand options to choose	Perform dealer movement and defines the player victory or not
Sobreviva	Storytelling for player survival in a terrorist attack	Menu-based	Dynamic definition of questions to show by a static collection of storytelling cards	Defines the next storytelling card to show or ends the game
Cancer de Boca	A game that tells stories of patients who have had mouth cancer	Menu-based	Dynamic definition of questions to show by a static collection of storytelling cards	Defines the next storytelling card to show or ends the game

Table II
REUSE METRICS OBTAINED WITH ASKME GAMES.

Game Name	Amount of SLOC / Total SLOC	Amount of Complexity / Total Complexity
GuessMy Number	84/(84+1343) = 5,89%	9/(9+342) = 2,56%
BodyZap	203/(203+1343) = 13,13%	6/(6+342) = 1,72%
LibrasZap	254/(254+1343) = 15,9%	21/(21+342) = 5,78%
ERQuiz	154/(154+1343) = 10,29%	3/(3+342) = 0,87%
TicTacToe	158/(158+1343) = 10,53%	89/(89+342) = 20,65%
BlackJack	258/(258+1343) = 16,15%	22/(22+342) = 6,04%
Sobreviva	138/(138+1343) = 9,32%	27/(27+342) = 7,32%
Cancer de Boca	252/(252+1343) = 15,8%	53/(53+342) = 13,42%

features that increases the abstraction representation of quiz games, along with configured scripts that achieve the flexibility available in game engine artifacts. However, this flexibility can generate an AsKME learning curve problem for games with a different quiz game loop (*Blackjack* and *TicTacToe* games, for example), something that can be solved with the inclusion of new extra features in the proposed feature model.

For validation purposes, 8 quiz games were also developed with AsKME assets. They confirmed the reuse possibilities of quiz game dynamics with AsKME, something realized by the ERQuiz game that repeated the "modes of play" designed in BodyZap features. They also confirmed the AsKME capability of building game dynamics from other game styles, such as storytelling and board games, something that can be extended to provide dedicated feature based engines in the future for specific game (sub)domains.

As future work, it is necessary to perform a usability assessment with developed AsKME clients, as well as a comparison with other development approaches to provide quiz games. The support for multiplayer matches, along with the AsKME adaptation to Quiz Board Game features [23] and test-driven development approaches will also be done in the future.

REFERENCES

[1] W. Scacchi and K. Cooper, "Research challenges at the intersection of computer games and software engineering," Proc. 2015 Conf. Foundations of Digital Games, 2015.

[2] D. Callele, E. Neufeld and K. Schneider, "Requirements engineering and the creative process in the video game industry," 13th IEEE International Conference on Requirements Engineering, pp. 240–250. IEEE, 2005.

[3] A. Furtado, A. Santos, G. Ramalho and E. Almeida, "Improving digital game development with software product lines," IEEE software, 28(5):30–37, 2011.

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

[5] D. Roberts and R. Johnson, "Patterns for evolving frameworks," Pattern languages of program design 3, pp. 471–486. Addison-Wesley Longman Publishing Co., Inc., 1997.

[6] J. Bosch, R. Capilla and R. Hilliard, "Trends in systems and software variability," IEEE Software, (3):44–51, 2015.

[7] J. Bosch, "Design and use of software architectures: adopting and evolving a product-line approach," Pearson Education, 2000.

[8] J. Gurp, J. Bosch and M. Svahnberg, "On the notion of variability in software product lines," Working IEEE/IFIP Conference on Software Architecture, pp. 45–54. IEEE, 2001.

[9] S. Tang and M. Hanneghan, "State-of-the-art model driven game development: A survey of technological solutions for game-based learning," Journal of Interactive Learning Research, 22(4):551–605, 2011.

[10] C. Lindley, "Game taxonomies: A high level framework for game analysis and design," Gamasutra feature article, 3, 2003.

[11] S. Kent, "The ultimate history of video games: From pong to pokémon and beyond-the story that touched our lives and changed the world," roseville, 2001.

[12] V. Sarinho and A. Apolinário, "A feature model proposal for computer games design," VII Brazilian Symposium on Computer games and Digital Entertainment, Belo horizonte, pp. 54–63, 2008.

[13] V. Sarinho and A. Apolinário, "A generative programming approach for game development," VIII Brazilian Symposium on Games and Digital Entertainment, pp. 83–92. IEEE, 2009.

[14] V. Sarinho, A. Apolinário and E. Almeida, "A feature-based environment for digital games," International Conference on Entertainment Computing, pp. 518–523. Springer, 2012.

[15] V. Sarinho and A. Apolinário, "Combining feature modeling and object oriented concepts to manage the software variability," International Conference on Information Reuse and Integration (IRI), 2010 IEEE, pp. 344–349. IEEE, 2010.

[16] F. Boaventura and V. T. Sarinho, "Mendiga: A minimal engine for digital games," International Journal of Computer Games Technology, 2017, 2017.

[17] J. Zagal and A. Bruckman, "The game ontology project: Supporting learning while contributing authentically to game studies," Proceedings of the 8th international conference on International conference for the learning sciences-Volume 2, pp. 499–506. International Society of the Learning Sciences, 2008.

[18] A. Furtado and A. Santos, "Defining and using ontologies as input for game software factories," Proceedings of the 3rd Brazilian Symposium on Computer Games and Digital Entertainment, 2006.

[19] V. Muller, "An open source architecture for building interactive dramas," Brazilian Symposium on Games and Digital Entertainment, pp. 89–100. IEEE, 2011.

[20] H. Guo, H. Trætteberg, A. Wang and S. Gao, "Pergo: an ontology towards model driven pervasive game development," OTM Confederated International Conferences, pp. 651–654. Springer, 2014.

[21] S. Tang and M. Hanneghan, "A model driven serious games development approach for game-based learning," Proceedings of the International Conference on Software Engineering Research and Practice (SERP), pp. 1. The Steering Committee of The World Congress in Computer Science, Computer Engineering and Applied Computing (WorldComp), 2013.

[22] E. Reyno and J. Cubel, "Model driven game development: 2d platform game prototyping," GAMEON, pp. 5–7. Citeseer, 2008.

[23] B. Bontchev and D. Vassileva, "Educational quiz board games for adaptive e-learning," Proc. of Int. Conf. ICTE, pp. 63–70, 2010.

[24] M. Wolf, "The medium of the video game," University of Texas Press, 2001.

[25] R. D. Riesco, "Generalized platform for creating of testing games," Master thesis, https://upcommons.upc.edu/handle/2099.1/11733, 2011.

[26] L. Zaid, F. Kleinermann and O. Troyer, "Feature assembly: A new feature modeling technique," International Conference on Conceptual Modeling, pp. 233–246. Springer, 2010.

[27] N. Valdez, E. Rolando, V. Garca-Daz, J. Lovelle, Y. Achaerandio and R. Gonzlez-Crespo, "A model-driven approach to generate and deploy videogames on multiple platforms," Journal of Ambient Intelligence and Humanized Computing, 8(3):435–447, 2017.

[28] S. Apel and C. Kästner, "An overview of feature-oriented software development," Journal of Object Technology, 8(5):49–84, 2009.

[29] K. Czarnecki, "Overview of generative software development," Unconventional Programming Paradigms, pp. 326–341. Springer, 2005.

[30] M. Sicart, "Defining game mechanics," Game Studies, 8(2), 2008.

[31] J. Gregory, "Game engine architecture," AK Peters/CRC Press, 2014.

[32] H. Heitkötter, T. Majchrzak and H. Kuchen, "Cross-platform model-driven development of mobile applications with md2," Proceedings of the 28th Annual ACM Symposium on Applied Computing, pp. 526–533. ACM, 2013.

[33] M. Palmieri, I. Singh and A. Cicchetti, "Comparison of cross-platform mobile development tools," 16th International Conference on Intelligence in Next Generation Networks (ICIN), 2012 , pp. 179–186. IEEE, 2012.

[34] V. Sarinho, E. Granjeiro and C. Cerqueira, "Bodyzap: Um jogo de im para o ensino de fisiologia humana," II Workshop de Jogos e Sade, XVI Brazilian Symposium on Games and Digital Entertainment. SBC, 2017.

[35] V. Sarinho, "Libraszap-an instant messaging game for knowledge assessment in brazilian sign language," Brazilian Journal of Computers in Education, 25(01):44, 2017.

[36] W. Frakes and C. Terry, "Software reuse: metrics and models," ACM Computing Surveys (CSUR), vol. 28(2):415–435, 1996.

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