A Feature-Based Approach to Build Quiz Games in a Multiplayer Multiplatform Perspective

Victor Travassos Sarinho
Universidade Estadual de Feira de Santana (UEFS)
Laboratório de Entretenimento Digital Aplicado (LEnDA)
Feira de Santana, Bahia, Brazil
vsarinho@uefs.br

***Resumo*—This paper presents the *Assessment of Knownledge Multiplayer Multiplatform Environment* (AsKMME), a feature-based approach to develop multiplayer multiplatform quiz games. It provides a domain game architecture, based on identified features of the quiz game dimension, that follows 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 multiplayer multiplatform quiz games was provided, together with the development of a quiz game for validation purpose.**

***Keywords*-software product lines; features; multiplayer; multiplatform; quiz game;**

I. Introduction

Digital games represent a great domain for conducting research related to Computer Science and Software Engineering (SE) [1]. However, it is important to emphasize that developing digital games is not the same as developing software systems [2], which causes some specific problems during their production.

In fact, understanding gaming engine architecture, as well as its programming quirks, is generally not a simple or intuitive task for its developers [2]. *Game designers* are also often directed at using either an SDK or a development *framework* to carry out their game designs as soon as possible [1], limiting as a result the creativity during production as well as possible resources able to be made available by them. Another major problem is due to the types of requirements applied (ex.: "the game should be fun to play"), which are difficult to fulfill as software engineering tasks [1].

Among the possible solutions, there is a focus on the development and incremental launching of minimal sets of game *features* capable of adaptively growing to meet the non-functional informal requirements (NFRs) raised for them [1]. Another important solution consists in separating the *G-fator* from the implementation game itself, in order to support game portability in distinct game development environments [3]. Finally, the production of simple and powerful tools capable of providing abstractions for digital game domains, coupled with the flexibility of gaming engines and other features reusable according to the projected game variants [2], is also something important to be applied in this context.

This paper presents the ***A**ssessment of **K**nownledge **M**ultiplayer **M**ultiplataform **E**nvironment* (**AsKMME**), a building environment of *multiplayer* multiplatform quiz [4] games based on *features*. It is a tool that provides domain abstractions based on the identification of features aimed at quiz games, capable of providing the adaptive evolution and rapid production of multiplayer quizzes, as well as the representation and execution of the G-factor in multiple software platforms, such as *web*, console, instant messaging, among others.

II. Domain Analysis

The AsKMME project aims to manage the variability of multiplayer quiz games. To this end, different features capable of representing families of quiz games, previously modeled on the AsKME project [5], were reused and complemented in the configuration of multiplayer quiz. These are organized features to: identify the proposed game (*Id* and *Locale*); represent initial menu data (*Initial Menu*) used in user interaction (*Start Menu*, *Highscore Menu*, *About Menu*, etc.); and set gameplay options according to the desired game settings (*Gameplay Menu*, *Game Flow*, *Game Score*, *Player Help*, *Prizes*, *Questions* and *Multiplayer*) (Figure 1).

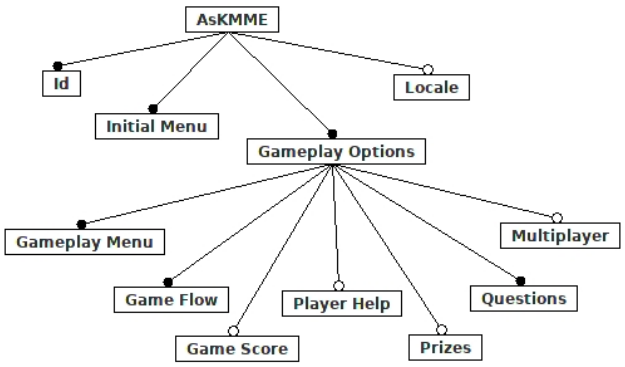

Figura 1. Partial illustration of the AsKMME feature model.

Focusing on the feature *Multiplayer* (Figure 2), it indicates the possible multiplayer game styles that can be applied in an available game mode. Depending on the game mode, this may be *single player*, multiplayer or both, according to the configuration applied to the feature *Multiplayer* and the boolean value applied to the *Is Single Player* feature. *Game Styles* describes the possible multiplayer approaches to the configured game mode, indicating: minimum and maximum number of players in a match; possible turn styles (*turn-based*, *round-based* and *real-time*); wait time for the player to lose the turn; and total time of the multiplayer match. Features *Messages* and *Performed By Event* follow the same logic defined in the *Game Flow* feature, however, with the ability to send messages to other players and with access to current game session information.

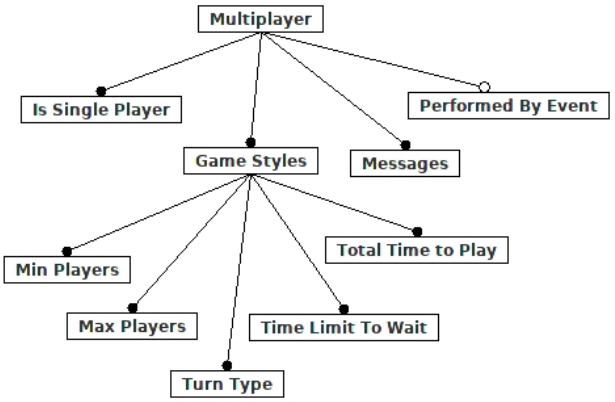

Figura 2. *Multiplayer* subfeatures of the AsKMME model.

III. Domain Implementation

For AsKMME games, the ***F**eature-**O**riented **S**oftware **D**evelopment* (**FOSD**) strategy was applied as a way of implementing the analysis domain. It is a paradigm for building, customizing, and synthesizing large-scale software systems in terms of features [6] that has been prominent in the context of building SPLs [7].

In this sense, following the described FOSD strategy, a ***D**omain **S**pecific **L**anguage* (**DSL**) was defined via ***J**ava**S**cript **O**bject **N**otation* (**JSON**) for AsKMME features, together with a multiplayer multiplatform game loop proposal able to execute such specifications successfully.

Regarding the multiplayer multiplatform game loop, a JavaScript state machine has been implemented (Figure 3). It is capable of interpreting the provided JSON configurations, according to the player inputs and game logic outputs, providing:

- the initialization of the game (*start-multiplayer-game* state) with the execution of the *startGameStatus* game routine;
- the verification that the player is active in the game and if it is your turn to play (*verify-multiplayer-status* and *get-turn-player* states);
- the player turn waiting to play when the game is turn-based (*wait-player-turn* state);
- the rendering of messages to be sent, as well as the execution of the *updateGameStatus* game routine until there is a message content available (*render-multiplayer-gameplay* state);
- checking and waiting for the end of the round when the chosen game is based on *rounds* (*verify-round-turn* and *wait-round-turn* states);
- the execution of the *updateGameStatus* game routine with the end-of-game checking (*update-multiplayer-gameplay* state);
- closing the match by confirming the current "dead" player status, which is then forwarded to a state to wait for the end-of-game of the other players in the current session (*wait-game-end* state); and
- the finalization of the game with the display of the final results of the players and the routing to the initial menu of the game (*end-multiplayer-gameplay* and *goto-initial-menu* states).

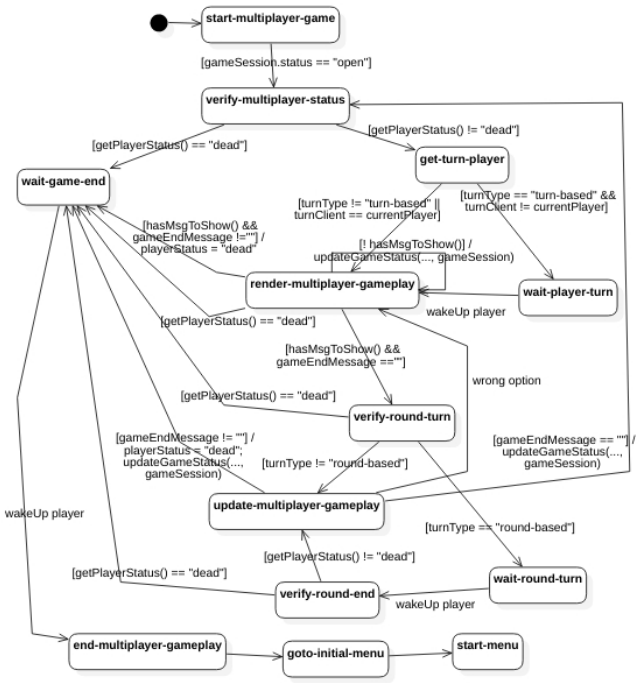

Figura 3. State machine details for multiplayer games.

Regarding to the management of the game sessions, it is necessary to carry out a temporal control of the sessions status for opening and closing purposes, as well as monitoring the current status of their respective players, to prevent the game from being caught due to the absence of

plays by the players. In this sense, whenever a new game session is opened, a session status check routine is activated (*refreshGameSessionStatus*), which is responsible to:

- open the session and wake the players if the maximum number of players has been reached, or if at least the minimum number of players has been reached after the end of the session opening wait time;
- terminate the attempt to open the session and wake up the waiting players if the session opening wait time has been exceeded and the minimum number of players has not been reached;
- close the game and wake up the waiting players if the time limit for the duration of the match has been reached;
- close the game and wake up the waiting players if all players have died in the game;
- put in standby players who lost the turn in a round;
- pass the turn to the next player still alive when the turn player loses the turn;
- wake up the live players after the end of a round;
- wake up the next live player after a player has made the move in turn; and
- delete session data after 3 seconds of session closure.

IV. Requirements Analysis and Product Configuration

To demonstrate the requirements analysis and configuration of a product based on provided AsKMME domain artifacts, a cross-platform version of the BodyZap game [8] was chosen to be produced. It is a single player multiplatform quiz game, previously produced on the platforms IMgine [9] and AsKME [5], which presents questions related to the digestive and respiratory systems studied in the teaching of human physiology.

To perform the requirements analysis and product configuration of the BodyZap multiplayer version, the single player configurations previously developed in features of the AsKME [5] platform were reused, adding the multiplayer features settings to each previously implemented game mode. Thus, for **Play by Time** mode, the following features were added: *gameStyles* settings for matches in *real-time*; multiplayer messages for new prize (*newPrizeMessage*), prize loss (*lostPrizeMessage*) and win (*winnerEndMessage*) events; and *performedByEvent* code putting all players to status "*dead*" after a player wins (Figure 4). For the **Play by Quantity** game mode, the following features were added: *gameStyles* configurations for round-based matches; multiplayer messages for new prize (*newPrizeMessage*), prize loss (*lostPrizeMessage*), win (*winnerEndMessage* and defeat (*loserEndMessage*) events; and *performedByEvent* code putting all players to status "*dead*" after a player wins. Finally, for **Best of 5** mode, the following features were added: *gameStyles* settings for *turn-based* matches; multiplayer messages for issue error (*errorMessage*) and

```
multiplayer: {
    isSinglePlayer: true,
    gameStyles: [
        {turnType:"real-time", menuKey:"T", menuText:"Tempo Real",
        minPlayers:2, maxPlayers:3, timeLimitToWait:60, totalTimeToPlay: 720}
    ],

    winnerEndMessage:["true","Jogador System._playerNumber venceu a partida!!"],
    newPrizeMessage: ["true","Jogador System._playerNumber conquistou o órgão:System._newPrize."],
    lostPrizeMessage:["true","Jogador System._playerNumber perdeu o órgão: System._lostPrize."],

    performEvent: function(gameStatus, gameSession, e){
        // possible events: start-game, show-question, evaluating-answer, game-win,
        //   game-lose, turn-end,   wrong-answer, correct-answer, new-prize, lost-prize
        if (e == "game-win")
            for (p in gameSession.players)
                    gameSession.players[p].status = "dead";
    }
}
```

Figura 4. Multiplayer JSON configuration of the *Play by Time* game mode.

```
multiplayer: {
    isSinglePlayer: true,

    gameStyles: [
        {turnType:"turn-based", menuKey:"T", menuText:"Turno",
         minPlayers:2, maxPlayers:3, timeLimitToWait:60, totalTimeToPlay: 180}
    ],

    loserEndMessage:["System._totalCorrectAnswers != 1","Jogador System._playerNumber finalizou sua
    errorMessage:["true","Jogador System._playerNumber errou uma questão!"],

    performEvent: function(gameStatus, gameSession, e){
        if (e == "wrong-answer" || e == "correct-answer"){
            console.log("### NEW TURN PLAYER ###");
            for (p in gameSession.players)
                if (gameSession.players[p].jidServer == gameStatus._currentPlayer.jidServer &&
                    gameSession.players[p].jidClient == gameStatus._currentPlayer.jidClient)

            gameSession.turnClient =
                gameSession.getNextPlayer(gameSession.players[p], gameSession.players).jidClient;
        }
        else if (e == "game-lose"){
            for (p in gameSession.players)
                if (gameSession.players[p].jidServer == gameStatus._currentPlayer.jidServer &&
                    gameSession.players[p].jidClient == gameStatus._currentPlayer.jidClient)
                    gameSession.players[p].status = "dead";
        }
    }
```

Figura 5. Multiplayer JSON configuration of the *Bether in 5* game mode.

defeat (*loserEndMessage*) events; and *performedByEvent* code indicating the next player of the turn (*turnClient*) after getting a correct or wrong answer from the current player, and putting the current player to status "*dead*" after completing all 5 proposed questions (Figure 5).

Multiplayer messages are intended to warn other session players about events that occur with the current player. In this way, for each configured BodyZap game mode, each opponent player of the session will be warned about correct/wrong answers, received/lost prizes, and victory/defeat of the current player in a match. As a result, each player known the opponents status in the context of the match, thus contributing to the engagement and immersion of the same in the configured game.

V. Results and Discussions

The AsKMME project follows the AsKME line of work, but with a simplification in the multiplatform client architecture and the centralization of the game loop on a single multiplayer server. The idea is that each AsKMME client must be concerned only with sending/receiving messages, which are worked by the multiplayer server that processes and contextualizes the status of each connected player based on them. As a result, the *perform* method of the multiplayer multiplatform game loop in the server centralizes client event processing, sending messages (the *sendUserInput* event) as responses from message requests (the *newMsg* event)

Tabela I
BODYZAP REUSE METRICS OBTAINED IN THE *IMgine*, *AsKME* AND *AsKMME* ARTIFACTS.

Project	Reused SLOC/ Total of SLOC	Reused Cyclomatic Complexity / Total Cyclomatic Complexity
IMgine	1513 / (889+1513) = 62,98%	284 / (19+284) = 93,73%
AsKME	1343 / (203+1343) = 86,87%	342 / (6+342) = 98,28%
AsKMME	3098 / (393+3098) = 88,75%	540 / (15 + 540) = 97,30%

according to the logic defined for each configured game.

Finally, considering the level of reuse achieved with BodyZap in the AsKMME project, Table I presents some metrics [10] obtained for complexity and reused code amount in produced versions for the BodyZap game. The *Total SLOC* metrics and *Total Complexity* metrics are calculated by summing the metrics of the developed games with the metrics of the supporting feature artifacts developed in each project (IMgine, AsKME and AsKMME). As a result, 88.75% SLOC reuse and 97.3% complexity reuse were obtained in the AsKMME project, surpassing the SLOC reuse data obtained in the previously developed IMgine and AsKME projects.

VI. CONCLUSIONS AND FUTURE WORK

This paper presented the AsKMME platform, a feature based approach for the construction of digital multiplayer multiplatform quiz games. For this, some SPL production steps (Domain Analysis, Domain Implementation, Requirement Analysis and Product Configuration) were performed based on the FOSD approach, resulting in feature models and artifacts capable of being configured in desired quiz games. Finally, some practical results obtained with the BodyZap game configuration were also presented, along with collected reuse metrics that are able to be compared with previous implementation approaches made for the chosen game.

AsKMME follows a modeling and implementation strategy applied in the AsKME platform, which ensures the production of static and dynamic quiz games, following predefined features that increase the abstract representation of desired quiz games, in conjunction with configured scripts that allow a level of flexibility similar to game engine artifacts. However, this flexibility generates a "learning curve" problem, especially with the development of competing multiplayer games, or in games that follow a different approach to the traditional quiz game loop (*Blackjack* and *TicTacToe* games developed in the AsKME platform, for example).

Regarding the level of obtained reuse, AsKMME presented excellent results in comparison to the previous work of IMgine and AsKME. However, it is still necessary to implement more AsKMME games derived from the AsKME platform, in order to: improve the comparison of metrics, complement the validation process, and confirm the possibilities of reuse and extension capacity.

As future work, it is intended to develop other multiplatform clients already obtained in previous IMgine and AsKME projects, as well as to start the implementation of a client with voice interpretation and associated natural language processing. Multiplayer versions of the single player games already developed in the IMgine and AsKME projects will also be delivered in the near future. The usability evaluation with AsKMME clients that will be developed, a comparison with other approaches of develop quiz games, and the application of unit and acceptance tests in developed feature artifacts will also be made as soon as possible. Finally, the production of dedicated feature-based approaches to other digital game domains, such as board games (BoardMME), textual adventures (AdverMME), mini-puzzles (PuzzleMME) and Role-Playing Games (RPGMME) will also be developed in the near future.

REFERÊNCIAS

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

[2] A. W. Furtado, A. L. Santos, G. L. Ramalho, and E. S. de Almeida, "Improving digital game development with software product lines," *IEEE software*, vol. 28, no. 5, pp. 30–37, 2011.

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

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

[5] 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.

[6] S. Apel and C. Kästner, "An overview of feature-oriented software development." *Journal of Object Technology*, vol. 8, no. 5, pp. 49–84, 2009.

[7] S. Apel, D. Batory, C. Kästner, and G. Saake, *Feature-oriented software product lines*. Springer, 2016.

[8] 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.

[9] V. T. Sarinho, G. S. de Azevedo, F. M. Boaventura, and F. de Santana, "Providing an im cross-platform game engine for text-messaging games," in *XVII Brazilian Symposium on Computer Games and Digital Entertainment (SBGames)*, 2018.

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