A Feature-Based Approach to Develop Digital Board Games

Filipe M. B. Boaventura[1 [0000−0001−7949−7483]] and Victor T. Sarinho[1 [0000−0002−5653−8390]]

Laboratório de Entretenimento Digital Aplicado (LEnDA)
State University of Feira de Santana, Av. Transnordestina, s/n, Novo Horizonte
Feira de Santana, Bahia, Brazil
fmbboaventura@gmail.com, vsarinho@uefs.br

Abstract. Several types of development strategies are available to provide digital games in a reusable way. However, the idea of a "one-size-fits-all" architecture for digital games can be problematic, being preferable to build dedicated architectures for specific game genres. This paper proposes the development of *feature-based* artifacts for the production of digital board games. It presents a subdomain game architecture that represents configurable features of core concepts related to board games (the game *model* and *controller*), and implements feature artifacts capable of being executed in distinct game clients (the game *view*). For validation purposes, two types of classic board games, together with a proposed web client for board games, were developed, consolidating as a result a software product line approach to develop classic board games.

Keywords: Feature modeling · Software product line · Board games.

1 Introduction

Game development is a complex task that requires technical mastery and integration skills from several areas of computer science [26]. In fact, many of the major traditional challenges on software engineering arise during the development of digital games as complex software systems, such as large-scale software engineering, game requirements engineering, game software design, and so on [26]. However, develop digital games is not the same as develop traditional software systems [12], which causes some specific problems during their production.

Regarding digital game development strategies, the reuse of specific components for game development, which can be provided or not by *SDKs*, *frameworks* and *game engines* [14], can reduce the development time and cost [9] to provide desired games. However, reused game components may restrict or predetermine the types of digital games that can be developed [26], changing as a consequence the initial game design idea for adapting purposes, and providing as a result an "unwanted" game project for the game designer.

In this sense, one possible solution would be to separate the core of the game logic (the *G-factor*) from the implementation resources, allowing a game

portability between development environments to be reused in distinct game productions [3]. However, the idea of a *"one-size-fits-all"* architecture for digital games can be problematic for many reasons, being preferable to build dedicated architectures for specific game genres [12].

This paper proposes the development of *feature-based* artifacts for the production of digital board games. The objective is to represent configurable features of core concepts related to board games (the game *model* and *controller*), and implement feature artifacts able to execute configured features in distinct game clients (the game *view*). As a result, a simple and scalable digital board game development approach is provided, which can be classified as a Software Product Line [20] able to generates new games through configured board game features.

2 Related Work

2.1 Feature Modeling and Development

Features are aspects or characteristics of a domain that are visible to the user [16], which are used to identify similarities or differences between products in a product line [1], and can be defined as a unit of functionality of a software system that meets a requirement [2].

Kang et al. [16] introduced the concept of *feature*, originally presented as part of *Feature-Oriented Domain Analysis* (FODA). According to them [16], *Feature Modeling* is used to identify system properties during domain analysis, and the *feature model* represents standard features of a family of systems and the relationships between them [16].

Feature-Oriented Software Development (FOSD) [2] is a paradigm for the construction, customization and synthesis of large-scale software systems in terms of *features*. According to Apel and Kästner [2], the FOSD is not a single development method or technique but a combination of different ideas, methods, tools, languages, formalisms and theories, connected by the concept of *feature*.

2.2 FOSD and Digital Game Development

Regarding the use of feature based approaches in the development of digital games, Zhang and Jarzabek [27] proposed the *RPG Product Line Architecture* (RPG-PLA), a set of common and variable features where any of the four original RPGs, as well as other similar games, could be derived from the RPG-PLA.

Furtado et al. presented the *ArcadEx Software Product Line*, a SPL created for 2D *arcade* games [11]. It is an improvement over the *SharpLudus* project [13], and it uses feature models to describe the commonality and variability of the 2D game domain.

Zualkernan proposed a Feature Modelling Framework for *Ubiquitous Embodied Learning Games* (UELG) [28], which represents educational games where the players interact with an augmented physical environment. It is an approach

designed with the intent of reducing the cost and shorten the development life-cycle of such ubiquitous systems, while including features regarding pedagogy and learning concepts [28].

Finally, Sarinho et al. have been obtained some interesting results in the development of digital games by FODA and FOSD paradigms, such as reusable features focused on digital game concepts and implementation aspects (NESI [22] and GDS [25] models), SPL structures to provide digital games by feature configurations (FEnDiGa [23] and MEnDiGa [6]), and feature development for specific game genres and categories (AsKME [24]).

2.3 Board Games

Board games are a specific group of games where figures are manipulated in a competitive game mode played over a surface according to predefined rules [7]. The expression *board game* is used to refer to games played on a table [8], even those without a board such as classic card games. Board games usually have a delimited surface divided into sectors where a set of pieces, associated with players by shape or color, are moved according to the events of the game [18].

Objectives, rules, strategies and win conditions can be quite diverse on board games, such as conquering the most areas of the board, eliminating pieces of the opponents, achieving the highest score, or gathering the most of some form of currency [18]. Regarding the various mechanics present in board games, the *Board Game Geek* (BGG) is one of the largest and most used forums dedicated to board games on the internet [17]. This site is dedicated to cataloging information about physical board games in order to maintain a database for posterity and historical research [4]. Containing a listing of 99953 board games, 83 categories and 51 game mechanics [4], the BGG database became a source of information widely used by game designers and researchers alike [17].

2.4 Board Game Models

Based on available information of the BGG database, Kritz, Mangeli, and Xexéo [17] proposed a board game mechanics ontology, following the concept of mechanics presented in the *MDA (Mechanics, Dynamics e Aesthetics) Framework* [15]. MDA is a formal approach to analyzing games, built with the goal of understanding the concepts that help designers, researchers and scholars to perform the decomposition of games into coherent and understandable parts [15].

Thus, the proposed ontology organizes several board game mechanics [4] into two main concepts: *Algorithm mechanics* and *Data Representation mechanics*. *Algorithm* is the general mechanic for the processes that take place in the game and is subdivided into: *Action* — a set of mechanics that allow the user to interact with the game; *Ruleset* — mechanics that define, among other aspects, the behavior of the components of the game; and *Goal* - mechanics that define the various goals to be achieved during the game, such as win conditions or transitory goals. *Data Representation* is the general mechanic for storing and conveying information in games and is divided into: *Component* — representing

the elements of the game that the players can own and manipulate directly; and *Resource* — representing the elements of the game that the player must manage to fulfill goals.

Finally, the *Quiz Board Game Model* proposed a formal model to represent *quizzes* as board games with multimedia appeal [7]. It describes board games as a set containing: objects, positions, functions, actions, rules and effects. The game is presented on a background image, where objects of various types are arranged in positions according to a matrix that describes their initial layout. Quiz questions are distributed according to board game positions, and are presented to the player through multimedia assets as objects reach these positions.

3 Metodology

As three main stages of this work: 1) a *feature model* was proposed to represent *features* for the board game domain; 2) a development approach was defined to describe, configure and interpret the proposed board game features; and 3) a board game client was implemented in order to execute configured features.

3.1 Features for Board Games

Feature Models [16] are used to describe the relationships and dependencies of a set of features belonging to a specific domain [2]. For this work, a feature model was proposed to describe generic software components for the production of digital board games. These features were designed to represent families of board games according to board game concepts found in literature [4,7,17].

The proposed model presents 88 features responsible for describing a *Board Game*, such as the game *Id*, the game logic as a whole (*Game Flow*) and the information that is handled during a gameplay (*Game Data*). *Game Flow* is represented in terms of *Actions*, *Rules* and *Game Events*, while *Game Data* describes the *Component Options*, the *Board Options* and the *Player Options* of the game. A complete diagram of all features proposed by this paper can be found at `https://github.com/lenda-uefs/BoardGameEngine`.

Regarding the *Game Data* subfeatures, *Component Options* represents the configuration of the objects that can be manipulated by the players during matches [17], such as *Dice* or *Tokens* of different types (*Token Type*) (Fig. 1). A *Token Type* is identified by a *Type Id*, which is referenced by a *Token*, and it configures the visual representation of the *Token* by mapping a *Player Id* to a *Image File*. *Tokens* are positioned on the board according to their *Position Id*. *Dice* appear in two distinct types: the *N-sided Die* represents a common dice, containing a sequence of numbers from 1 to N; and the *Special Die* can be used by the *Game Designer* to represent custom dice, containing a list of custom values (*Value Set*). *Components* can belong to a specific player, identified by feature *Owner Id*.

Still on *Game Data* (Fig. 2), the *Board Options* feature represents the game board, which consists of a *Background Image* and a set of *Positions* which can

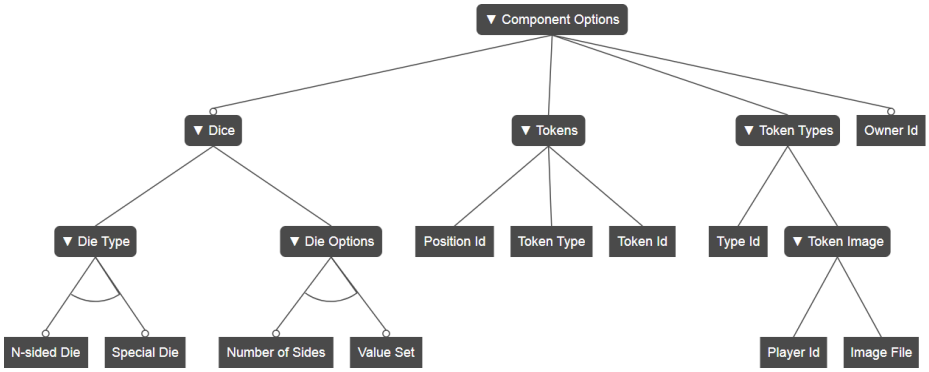


Fig. 1. *Component Options subfeatures.*

take on various types of (*Position Type*). *Positions* are identified by a *Position Id*. They have limited bounds defined by the *Area* feature and are located by the *Location* feature, which represent the Cartesian coordinates of the *Position* on the *Background Image*. *Positions* can hold a certain number of *Tokens* according to its *Capacity* feature. The remaining *Position* features are determined by the *Board Type*. On *Grid* boards, *Positions* are organized in a two-dimensional array, with the feature *Grid Index* holding its two-dimensional index. On *Point to Point* boards, *Positions* may point to adjacent positions (*Previous Position* and *Next Position*), forming the path to be traversed by the *Tokens*. Both *Previous Position* and *Next Position* can refer to more than one *Position*, allowing multiple paths to be formed on the board.

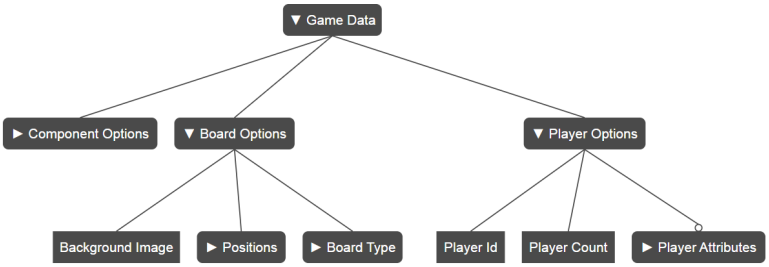


Fig. 2. Partial feature diagram for the *Game Data* feature.

Finally, *Player Options* represent the game settings regarding the number of players (*Player Count*), their identifiers (*Player Id*) and their attributes (*Player Attribute*). *Player Attribute* represents a list containing the various types of resources that players can accumulate during the game, such as score points and values to be used as currency. These resources are represented by *Name*, *Value*, a textual description (*Description*) and an illustrative image (*Image*).

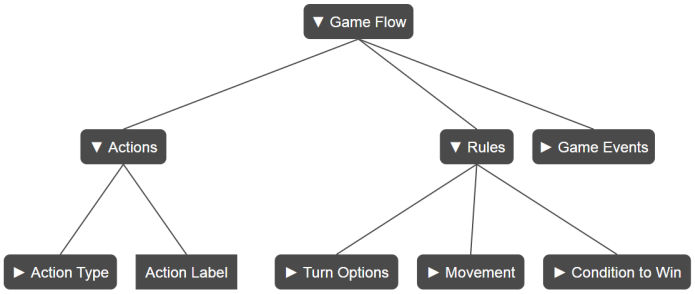


Fig. 3. Partial feature diagram for the *Game Flow* feature.

Regarding the *Game Flow* subfeatures (Fig. 3), the *Action* feature represents the set of actions that can be performed during the game. Each *Action Type* is presented to the player by an *Action Label*, which contains a textual description of the *Action* to be performed. By these actions, the player is able to change the state of the game, manipulating *Components* and/or *Player Attributes*. For example, through the feature *Roll Dice*, the player is able to interact with the *Component Dice* to generate random numbers. *Select Token* and *Select Position* allow the player to mark a *Token* or a *Position* respectively. *End Turn* allows the player to end his movement, giving the turn to the next player. Finally, the feature *Move Token* is used to move *Tokens* between *Positions*, according to the rules that will be discussed ahead.

Performing actions and evaluating rules can trigger several *Game Events*. Currently, these events include: *Dice Event* — triggered when a value is retrieved from a dice roll; *Passing Event* and *Stopping Event* — which are triggered when a *Token*, respectively, passes through or stops on a *Position*; *End Turn* and *End Game* — respectively triggered when a turn or game is completed; *Token Eliminated* — triggered when the player loses a *Token*; and *Token Select* — triggered when the player selects a *Token*.

Rule features are responsible for evaluating the execution possibility of *Action* features according to the current state of the game. It also defines the configuration of other aspects of the game, such as *Token Movement*, the player succession policies throughout the turns (*Turn Options*), and the *Conditions to Win* the game.

Regarding turn related rules, *Turn Options* describes the game settings according to the maximum number of turns (*Max Turn Count*), the *Player Order* and the *Action Queue* to be performed during a turn. The succession of players can happen in a *Static Order*, which means *player 3* will always play after *player 2*, *player 2* after *player 1* and so on, or it may vary according to some event (*Dynamic Order*). For example, a game may give the turn to a player who has just finished his turn if he or she has fulfilled any special conditions.

For the *Movement* rules, they are directly affected by the *Board Type* and the organization of the *Positions* on the board. In *Point to Point* boards, where the *Positions* are organized as a trail of interconnected points, the *Tokens* can

only move to the *Next Position* adjacent to their current position. This type of movement is known as *Point to Point Movement* [4,17]. If the current position of the *Token* refers to more than one adjacent position, it is necessary to decide which position the *Token* should be moved to. This decision is configured by the feature *Path Selection Rule*, which can be: *Manual*, delegating to the player the task of selecting the desired position; or *Automatic*, where a provided function decides the *Token* location that must be sent.

Regarding the *Position Selection Rule*, it represents an *evaluator* to be provided by the *game designer* to validate a *Position* selected by the player. This method is used in games that use the *Grid Board Type*, since the *Positions* will not hold information on which adjacent *Positions* the *Tokens* can move to.

Finally, *Condition to Win* represents a collection of the winning conditions of the game, as well as their evaluation settings. From the currently available conditions, *Reached Position* refers to the act of moving a *Token* to a position with a specific *Position Type*. *Player Attribute Value* and *Number of Remaining Tokens* are conditions related to numerical values and they refer respectively to: a *Player Attribute*, identified by an *Attribute Name*; or the remaining amount of *Tokens* of a certain *Token Type*. These conditions can be evaluated for the *Highest* value, *Lowest* value or compared with a *Exact* value, as described by the feature *Evaluation Option*. Finally, the feature *Evaluation Event* defines the moment when the evaluation of the conditions is performed. The conditions can be evaluated for every update of the game state (*On Game Update*) or only once per turn (*On Turn End*). In both cases, the game ends if the winner is found. If the game ends without a winner, such as the maximum number of turns was reached, the end game victory conditions (*On Game End*) are evaluated to decide the winner.

3.2 Developing Feature Based Board Games

Once the proposed feature model is defined, it is necessary to define a development approach for the implementation of configured board games. Several FOSD [2] approaches have been used successfully for the development of digital games, such as generative programming [25] and SPL instance [12]. For this work, the game development approach consist of the definition of a *Domain-Specific Language* (*DSL*) for the description and configuration of the proposed board game features, as well as a generic *game loop* able to interpret and run the board game configurations.

In this sense, by the proposed feature model, a DSL was defined using *JavaScript Object Notation* (*JSON*), where each feature was used to represent the numeric, textual, and procedural values involved in the configuration of a digital board game. Fig. 4 illustrates an example of a partial configuration of the *Game Flow* feature and its subfeatures *Actions*, *Rules* and *Game Events*.

In order to provide a *game loop* capable of running the proposed features, an interpreter has been developed using Javascript for the purpose of reading and executing game actions and events, and to control the game flow defined in a given JSON configuration file. The interpreter follows the AsKME strategy

```
gameFlow: {
  actions: [
    {actionType: "rollDice", actionLabel: "Roll Dice"},
    {actionType: "selectToken", actionLabel: "Select Token to move"},
    {actionType: "moveToken", actionLabel: ""}
  ],
  rules: {
    movement: {
      pathSelectionRule: function (GameStatus) {...} // "manual" or a function
    },
    turnOptions: {
      maxTurnCount: null,
      playerOrder: function (GameStatus){...}, // "staticOrder" or a function
      actionQueue:["rollDice", "selectToken", "moveToken"]
    },
    conditionToWin: {
      numRemainingTokens: {tokenType:null, evalOption:"exact", value:0, evalEvent:"update"}
    }
  },
  gameEvents: {
    tokenSelected: function(GameStatus, selectedToken) {
      if (
        selectedToken.position.positionType.includes("Base") &&
        GameStatus.currentPlayer.diceValue != 6 &&
        GameStatus.currentPlayer.diceValue != 1 &&
        GameStatus.currentPlayer.attributes["Active Tokens"] > 0
      ) {
        GameStatus.repeatAction(
          "You got a " + GameStatus.currentPlayer.diceValue}.
          + " You need a 1 or a 6 to move a token into the game."
          + " Please select another token.");
      }
    }, ...
  }
```

Fig. 4. JSON representation of a partial configuration of a board game.

for initializing and updating configured games, using the *startGameStatus* and *updateGameStatus* functions [24]. The *updateGameStatus* function performs the actions of the game, following the order queued in *Action Queue* (Fig. 4) and triggering the appropriate *Game Events* when necessary. Each performed action is removed from the queue, and, when there are no more actions to take, the turn ends, the action queue is restored and the game identifies the next player.

3.3 The Board Game Client

As a presentation layer of configured board game features, a web client (Fig. 5) was initially implemented using *Phaser*, an open source framework for browser games based on *HTML5 Canvas* and *WebGL*. The user interface is divided into three main areas. The left *sidebar* displays the values of the current *Player Attributes*, as well as its illustrative icon and textual description (displayed on a *mouse over* event). The central area displays, among other information, the game board, drawn by *Phaser* on a *Canvas* with a resolution of 800x600 pixels. Finally, the right *sidebar* displays the available *Actions* and feedback messages for the player. User inputs are obtained through click events on *hyperlinks* on the right *sidebar*, on the *Tokens*, or on the *Positions* of the board. These events execute the *updateGameStatus* function of the *interpreter* which in turn executes the appropriate *Action* according to the current state of the game. After running

the *updateGameStatus*, the *interpreter* notifies the client to update the current *View* with the new values from the current game state.

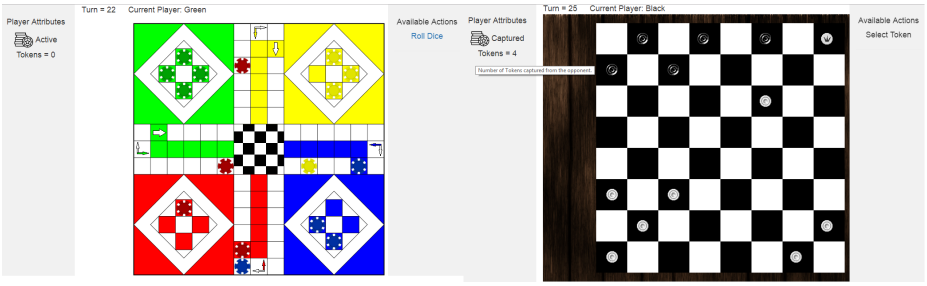


Fig. 5. Two screenshots of the *Phaser* client running *Ludo* and *Checkers*.

4 Results and Discussion

For the validation of the proposed board game features, two digital versions of classic board games were implemented (Fig. 5). For the *Grid* board, a digital version of the popular game *Checkers* was developed. This version is played by two players on an 8x8 grid, each starting with 12 *Tokens*. Regular *Tokens* (called *men*) can only move and capture enemy *Tokens* forward. Special *Tokens* (called *Kings*) can both capture and move backwards. Captures are mandatory and can be done in sequence, if possible. The winner is the player who captures all enemy *Tokens*.

For the *Point to Point* board, a digital version of the *Ludo* [5] was implemented. In this game, the players must guide their *Tokens* from their bases to the finish line on the center of the board. The movement is controlled by a simple *six-sided die* that determines how many positions a selected *Token* should move. However, *Tokens* can only move from their starting positions if the *Roll Dice Action* returns a one or six value. Also, if a player roll a six, it gets another dice roll. And if the player roll a six for the third time in a row, the turn ends. If a token stops at a position that contains an opposing token, the latter is "captured" and sent back to its base. If the token occupying the position is not from the opponent, the movement is undone and the turn ends.

Considering the reuse level achieved by FOSD usage in the developed games, some code metrics [10] were collected using the *Plato* code analyzer [21] for the *amount of reused code* and the *cyclomatic complexity*. The *Amount of Reuse* metric is used to evaluate and monitor the reuse improvement effort by tracking percentages of reuse of lifecycle objects over time [10], which can be defined as the number of reused *SLOC* (*Source Lines of Code*) of a given module divided by the total *SLOC* of the system. The *Cyclomatic Complexity* (*CC*) is used to

Table 1. Obtained SLOC and CC metrics for Ludo and Checkers games.

Game Name	**Reused SLOC / Total SLOC**	**Amount of CC / Total CC**
Ludo	948/(315+948) = 75,06%	120/(23+120) = 83,91%
Checkers	948/(274+948) = 77,58%	120/(38+120) = 75,94%

measure the complexity of the code based on the number of execution flows in the source code [19].

Table 1 shows the obtained metrics for the developed games, where the Reused *SLOC* and the total *CC* were calculated by the sum of the respective SLOC and *CC* values for the *Phaser* client, the JSON interpreter and the JSON configuration file. As a result, 76,32% of SLOC reuse and 79,93% of CC reuse, on average, were obtained from the developed artifacts, confirming the game core reusability and mantainability for the developed board games.

5 Conclusions and Future Work

This paper presented a feature-based approach for the development of digital board games. It is based on a previous development effort to provide a cross-platform product line architecture for the quiz game domain [24], but in this case providing board game features and artifacts able to manage the game logic represented as a game core of simple board games.

To achieve this objective, a feature model has been defined to represent relevant concepts of the board game domain, along with software artifacts capable of configuring and executing the board game dynamics in a web client. In fact, once the components of a desired board game are identified and configured through the proposed feature model, these can be executed by the implemented artifacts in a client application developed for a specific platform, providing as a result new board games for each new configured JSON file.

Regarding the Javascript interpreter for board game features, it provides an interface to retrieve and modify the state of the board game described by the JSON file. The client application, then, is responsible for querying the interpreter about the information needed and to use the obtained information to present the game using its own components. As a result, the Javascript interpreter works as a game controller facade based on features that communicates with developed cross-platform client applications, such as web, mobile platforms, game engines (Unity3D, Godot) and Arduino (Johnny-Five framework), in order to provide configured board games [24].

Digital versions of the games *Ludo* and *Checkers* were also presented in this work. It is an important verification/validation step of this project as an attempt to show the feasibility of the reusable artifacts to produce digital versions of classic board games. Furthermore, the obtained metrics for the developed software artifacts (SLOC reuse and cyclomatic complexity) presented satisfactory results, with over 76% SLOC reuse and 79% complexity reuse. However, the amount of Javascript functions that need to be programmed in order to configure board

game dynamics using the proposed model can compromise the maintainability and scalability of the developed games. Thus, it is necessary to further identify common behaviors between board games in order to build a more detailed board game feature hierarchy. It is also need to develop more games using the proposed artifacts to perform an improved benchmarking of metrics, as well as a better evaluation of their reuse and expansion capabilities.

As future work, it is necessary to expand the proposed feature model to cover more board game mechanics such as trading and collecting cards, for example. The development of new board game clients, such as mobile, Unity3D and embedded systems, along with the support for *multiplayer* matches, will also be performed in the future, in order to provide a *multiplayer* and *multiplatform* SPL for the development of digital board games.

References

1. Antkiewicz, M., Czarnecki, K.: Featureplugin: Feature modeling plug-in for eclipse. In: Proceedings of the 2004 OOPSLA Workshop on Eclipse Technology eXchange. pp. 67–72. eclipse '04, ACM, New York, NY, USA (2004)
2. Apel, S., Kästner, C.: An overview of feature-oriented software development. Journal of Object Technology **8**(5), 49–84 (2009)
3. BinSubaih, A., Maddock, S.: Game portability using a service-oriented approach. Int. J. Comput. Games Technol. **2008**, 3:1–3:7 (Jan 2008)
4. BoardGameGeek: Board game mechanics (2018), `https://boardgamegeek.com/browse/boardgamemechanic`, *Accessed 28 jul. 2018.*
5. BoardGameGeek: Pachisi — board game (2018), `https://www.boardgamegeek.com/boardgame/2136/pachisi`, *Accessed 20 jan. 2019.*
6. Boaventura, F., Sarinho, V.T.: Mendiga: A minimal engine for digital games. International Journal of Computer Games Technology **2017** (2017)
7. Bontchev, B., Vassileva, D.: Educational quiz board games for adaptive e-learning. In: Proc. of Int. Conf. ICTE. pp. 63–70 (2010)
8. Duarte, L.C.S., Federal, S.: Jogos de tabuleiro no design de jogos digitais. In: Anais do XI Simpósio Brasileiro de Jogos e Entretenimento Digital, Brasília, DF. pp. 132–137 (2012)
9. Folmer, E.: Component based game development: A solution to escalating costs and expanding deadlines? In: Proceedings of the 10th International Conference on Component-based Software Engineering. pp. 66–73. CBSE'07, Springer-Verlag, Berlin, Heidelberg (2007)
10. Frakes, W., Terry, C.: Software reuse: metrics and models. ACM Computing Surveys (CSUR) **28**(2), 415–435 (1996)
11. Furtado, A.W., Santos, A.L., Ramalho, G.L.: Sharpludus revisited: From ad hoc and monolithic digital game dsls to effectively customized dsm approaches. In: Proceedings of the Compilation of the Co-located Workshops on DSM'11, TMC'11, AGERE! 2011, AOOPES'11, NEAT'11, & VMIL'11. pp. 57–62. SPLASH '11 Workshops, ACM, New York, NY, USA (2011). https://doi.org/10.1145/2095050.2095061, `http://doi.acm.org/10.1145/2095050.2095061`
12. Furtado, A.W., Santos, A.L., Ramalho, G.L., de Almeida, E.S.: Improving digital game development with software product lines. IEEE software **28**(5), 30–37 (2011)

13. Furtado, A.W.B., Santos, A.L.M.: Using domain-specific modeling towards computer games development industrialization. In: In Domain-Specific Modeling workshop at OOPSLA (2006)
14. Gregory, J.: Game engine architecture. AK Peters/CRC Press (2014)
15. Hunicke, R., LeBlanc, M., Zubek, R.: Mda: A formal approach to game design and game research. In: Proceedings of the Challenges in Games AI Workshop, Nineteenth National Conference of Artificial Intelligence. pp. 1–5 (2004)
16. Kang, K., Cohen, S., Hess, J., Novak, W., Peterson, A.: Feature-oriented domain analysis (foda) feasibility study. Tech. Rep. CMU/SEI-90-TR-021, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, USA (1990)
17. Kritz, J., Mangeli, E., Xexéo, G.: Building an ontology of boardgame mechanics based on the boardgamegeek database and the mda framework. In: XVI Brazilian Symposium on Computer Games and Digital Entertainment, Curitiba. pp. 182–191 (2017)
18. Lucchese, F., Ribeiro, B.: Conceituação de jogos digitais (2009), `http://www.dca.fee.unicamp.br/~martino/disciplinas/ia369/trabalhos/t1g3.pdf`, *Accessed 12 nov. 2018.*
19. McCabe, T.J.: A complexity measure. In: Proceedings of the 2Nd International Conference on Software Engineering. pp. 407–. ICSE '76, IEEE Computer Society Press, Los Alamitos, CA, USA (1976), `http://dl.acm.org/citation.cfm?id=800253.807712`
20. Northrop, L.M.: Software product lines: reuse that makes business sense. In: Australian Software Engineering Conference (ASWEC'06). pp. 1 pp.–3 (April 2006)
21. Plato: Javascript source code visualization, static analysis, and complexity tool. https://github.com/es-analysis/plato (2012)
22. Sarinho, V., Apolinário, A.: A feature model proposal for computer games design. In: VII Brazilian Symposium on Computer games and Digital entertainment, Belo horizonte. pp. 54–63 (2008)
23. Sarinho, V.T., Apolinário, A.L., Almeida, E.S.: A feature-based environment for digital games. In: Entertainment Computing - ICEC 2012. pp. 518–523. Springer Berlin Heidelberg, Berlin, Heidelberg (2012)
24. Sarinho, V.T., de Azevedo, G.S., Boaventura, F.M.: Askme: A feature-based approach to develop multiplatform quiz games. In: 2018 17th Brazilian Symposium on Computer Games and Digital Entertainment (SBGames). pp. 38–3809. IEEE (2018)
25. Sarinho, V.T., Apolinário, A.L.: A generative programming approach for game development. In: Games and Digital Entertainment (SBGAMES), 2009 VIII Brazilian Symposium on. pp. 83–92. IEEE (2009)
26. Scacchi, W., Cooper, K.M.: Research challenges at the intersection of computer games and software engineering. In: Conference on Foundations of Digital Games (FDG 2015). Pacific Grove, CA (June 2015)
27. Zhang, W., Jarzabek, S.: Reuse without compromising performance: Industrial experience from rpg software product line for mobile devices. In: Proceedings of the 9th International Conference on Software Product Lines. pp. 57–69. SPLC'05, Springer-Verlag, Berlin, Heidelberg (2005)
28. Zualkernan, I.: A feature modelling framework for ubiquitous embodied learning games. In: New Trends in Software Methodologies, Tools and Techniques. vol. 231, pp. 198–216 (01 2011). https://doi.org/10.3233/978-1-60750-831-1-198