Rebuilding games at runtime

Diego Castro
COPPE/Computer Systems Engineering Program
Federal University of Rio de Janeiro
Rio de Janeiro, Brazil
diegocbcastro@cos.ufrj.br

Cláudia Werner
COPPE/Computer Systems Engineering Program
Federal University of Rio de Janeiro
Rio de Janeiro, Brazil
werner@cos.ufrj.br

***Abstract*—The gaming industry is growing over time, having a considerable amount of fans. Creating a game is a complex process that can take a long time until its release, causing a lot of anxiety for the game fan population. Some of these fans can not wait for the release of a specific game, creating their own versions of games. This process of creating games in an ad-hoc manner from others that already exist is called mod and can be compared to the concept of Software Reuse, where software is created from others. This work aims to increase the systematic of creating mods based on reuse concepts such as Product Line. In this paper, an initial prototype of a game created by applying Product Line concepts to generate new mods in a more systematic way is described.**

***Index Terms*—Game, Mods, Product line**

I. Introduction

Games have become one of the most popular entertainment sources, having enthusiasts of all genres and ages, becoming one of the standouts in industry, raising billions of dollars over the years [1]. To get a sense of numbers, the gaming industry has raised over 400 billion in the last three years [1], [2].

Despite being a very profitable branch of the industry, the process of developing a game can be very time-consuming, complicated, taking a long time to be built. With so many game fans, waiting for a particular game to be released can create a lot of anxiety. With this, many fans of games end up creating their own versions. These modifications are called mods and involve the practice of developing modifications or adaptations of a game.

This idea of using an existing game to adapt and create another is very similar to applying Software Reuse (SR) opportunistically, which involves the use of pre-existing artifacts to create new ones [3] [4]. SR has several approaches, such as components, model-based development and product line. Based on the game modification scenario, where a game is adapted to generate others, one can suggest the use of a software product line (SPL). This approach is used when a similar set of functionality is reused and some new features are added to the original software, generating new products [5]. Therefore, it is possible to imagine a game where features are created and modified, generating new games.

Building a mod can also be time-consuming, having several problems along the way, such as: needing to understand the game's source code to modify it, copyright issues or even take longer than the original game to release the new version. Another problem worth mentioning is the rework that many programmers may have. The current tools provided for these developments do not demonstrate that a particular mod has ever been created. With so many game fans producing their games simultaneously, the possibility of creating similar games can be high.

Based on the facts described above, this work tries to modify the creation of mods context to bring a greater systematic in this process and to mitigate the problems listed before, reducing rework and generating more mods. Thus, the use of Product Line approach is proposed. An initial prototype was created where product line and mod concepts work together to provide automatic generation of new mods.

The remainder of this paper is presented as follows: Section II presents relevant concepts for understanding of the paper, III demonstrates a proposal to create a game for automatic generation of new mods, Section IV discusses what was observed with the creation of the game, and Section V concludes with the some remarks, limitations and future work.

II. Conceptualization

In literature, several definitions are given for games. However, it is important to note that there is no precise and consensual definition of a game. Games are fun activities, which have an unpredictable outcome within a fictional world [6]. Another definition is that games are systems in which users participate in a dispute, defined by rules, resulting in a consequence [7]. Therefore, a possible explanation would be that games represent activities that use an abstract world in which decisions, actions, and rules are performed, aiming at a recreational activity in the form of distraction or entertainment.

What makes a game be characterized as new? Understanding what constitutes a new game is not as simple as it may seem. By defining games, it is possible to observe some characteristics necessary to differentiate them, such as new rules, actions, decisions made by the player, among other characteristics. It is not necessary to modify all these elements of game definition to create a new game. There are stages of modification. When these modifications are made by the community they are called Mod and can be defined as the practice of developing modifications or adaptations of a game [8] [9]. The main types of mods are:

- **Interface customization:** Modifications to the visual elements, such as simply remodeling the elements that are shown on screen.

- **Mutators/tweaks:** Modify or add minor modifications that do not interfere with gameplay and mechanics.
- **Add-ons:** Game extensions such as new maps or new objects. The original mechanics of the game are little modified.
- **Mods:** They can be understood as the junction of the previous two.
- **Total conversion:** Some modifications go as far as creating entirely new games, such as CounterStrike, which was an adaptation of Half-Life.
- **Machinima:** Change the visual replay of game sessions, that is, modify the story or movie represented by an animation.

As mentioned before, SR is an extensive research area with several approaches, such as component-based development, asset repositories, and product lines. Software Product Line (SPL) is a set of methods, techniques, and tools for developing similar systems, where a common core and similar behaviors are reused and expanded. An SPL allows you to generate a family of applications that share a typical architecture. A set of components that provide essential functions is configured but allows modifications in components at certain points of variation. Through SPL, it is expected to reduce development time, ease of maintenance, higher code quality, and a more significant number of variability of products generated from an original one [5] [10]. This is the main advantageand demonstrate exploited by this work.

There are several ways to diagram and illustrate a product line. In general, there are three types of elements: mandatory (present in all applications, forming the reusable core), alternatives (restrictive characteristics, an application may have one feature or another), and optional (features that specific applications may or may not have). SPL differs from other reuse approaches when comparing predictive with opportunists approaches. Rather than creating generic software components in a library in the hope of future reuse, SPL only requires creating software artifacts when their reuse is expected in one or more products [5]. SPL, in general, has two main stages:

- **Domain engineering:** stage where common functionalities are verified and thought, in the case of games, mechanics, dynamics, and aesthetics.
- **Application engineering:** stage where common elements are reused, and specific functionality is added.

Looking at the gaming scenario, Domain Engineering is where all game features are thought, where the game design document is created. In the mod scenario where the game fan community is inserted, it is necessary to divide the application engineering into two parts, the first where the original game is created and the second where this game is expanded, giving rise to the mods.

SPL, in its essence, needs to be defined at design time. The characteristics must be described before running, so it is important to establish the domain of the application. Going back to the mod scenario, many end users may modify the original game and create features that the original game creators may not have thought, which is no longer a design-time issue.

A more recent view of product lines is Dynamic Product Lines (DPL). This approach can be understood as an SPL where modifications and adaptations to applications are made at runtime [10]. In the gaming scenario, mods are built after the game is released. With this in mind, the game product line possibly needs to be a hybrid line formulated initially at design time and later expanded by the community in a dynamic way.

III. Proposal

There are several ways to organize ideas for building a game, such as flow models, gameplay, features, among others. A well-known and widespread model in the gaming area is the MDA framework. All types of mods demonstrated in the previous section can be defined according to the dimensions of MDA. This framework aims to divide the main features of games into three parts which are [11]:

- **Mechanics** can be understood as the rules and actions of the game.
- **Dynamics** represent the behavior that occurs as a result of actions.
- **Aesthetics** are the emotions experienced by the player.

.

Bearing in mind that most games can be described in general by MDA, it can be said that if modifications occur in each of these parts, it is possible to generate a new mod. Thus, a prototype game based on MDA framework was created where runtime modifications are applied to the game, creating a new mod from time to time.

The prototype created makes use of the idea of dynamic product lines applied to mods. Because it is a prototype, this type of approach was used to create the game due to its greater ease in coordinating the changes that could occur in the game. As mentioned in the previous section, the features are created at design time in the classic product line. However, as it is a game, the dynamic product line was chosen to make it more fluid.

The game was created to resemble a dynamic product line, where each part of MDA varies throughout the game. A mod is created by applying one or n mutators to a game. Therefore, it is possible to think that there are mutators, one for each part of the MDA framework, and that these mutators are applied to games to generate new mods. Abstracting this idea a little bit more, it is possible to think that mutators from MDA applied in sequence can generate each one of the types of mods described in the previous section. For example, it would be enough to use an aesthetics mutator to create an interface customization mod. To make mutators or tweaks, it would be necessary to apply a dynamics mutator; finally, a total conversion would require to use n mutators of all three dimensions.

MDA uses aesthetics to refer to the emotions that are felt by the user when using the game, such as fun, challenge, creativity, among others. The original game is already thought to provide certain emotions. However, these sensations are provided to the player regarding cause and effect. The set of

Fig. 1. MDA Generation Game.

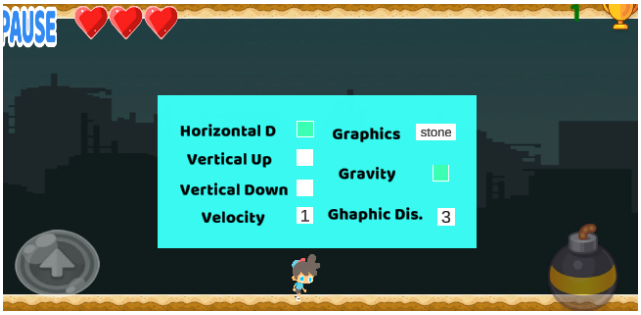

Fig. 2. Configuration panel.

TABLE I
GAME MECHANICS, DYNAMICS AND AESTHETICS.

Mechanics	Run and jump or fly, destroy objects with bombs, destroy objects by clicking on the object, lives, score, pick up coins to buy new levels, lose lives by hitting objects
Dynamics	Increase or decrease speed, increase or decrease gravity, change character directions
Aesthetics or Interface	Fire environment Ice environment Earth environment Air environment

mechanics and dynamics of the game is thought to provide a sensation for the player. While creating mods, it is challenging to develop new sensations due to this cause-and-effect relationship. When modifying rules and dynamics, sensations can change, but there is no specific modifier that generates a new aesthetic. Therefore, in this work, the aesthetic dimension of the game will be treated only as of visual part, that is, changing maps, objects on screen, etc.

The game works like an infinity runner. Every time the player reaches a checkpoint (represented by a flag), a new random set of mechanics, dynamics, and aesthetics is selected, thus creating a new version for the game at runtime. Figure 1 demonstrates 4 random phases created by the game. Every time the game character encounters a flag, a new kind of mod is generated randomly. To visualize the characteristics that were selected, the game presents a panel that demonstrates the chosen configuration. Figure 2 shows the dashboard.

Table 1 demonstrates the game characteristics divided according to the MDA framework. The mandatory mechanics are lives, scores, coins, and losing lives when hitting objects in the level. Note that any random mod will have these features. For all random features of the game, there is a boolean variable that controls whether it will be set in the game or not. For example, for each mechanic, the game will choose between walking and jumping or flying, destroying objects with bombs and destroying objects by clicking, serving the same rule for the dynamics. For game aesthetics, a terrain type will be selected in each phase.

Figure 3 shows the feature tree with all game functionality, where filled balls are mandatory feature and unfilled balls are optional. Filled triangles demonstrate that only one feature must be selected. The model presented in this figure was based on the FODA characteristics model [12]. The figure has been divided into the three parts of MDA for easy viewing.

IV. DISCUSSION

Several reasons lead a game fan to create a Mod. Among the main ones are: trying new things, solving bugs, creating new characters, increasing the difficulty of the game, gaining advantages about the game, increase the game life cycle, among others [1]. Modifiers, as well as games, are complex

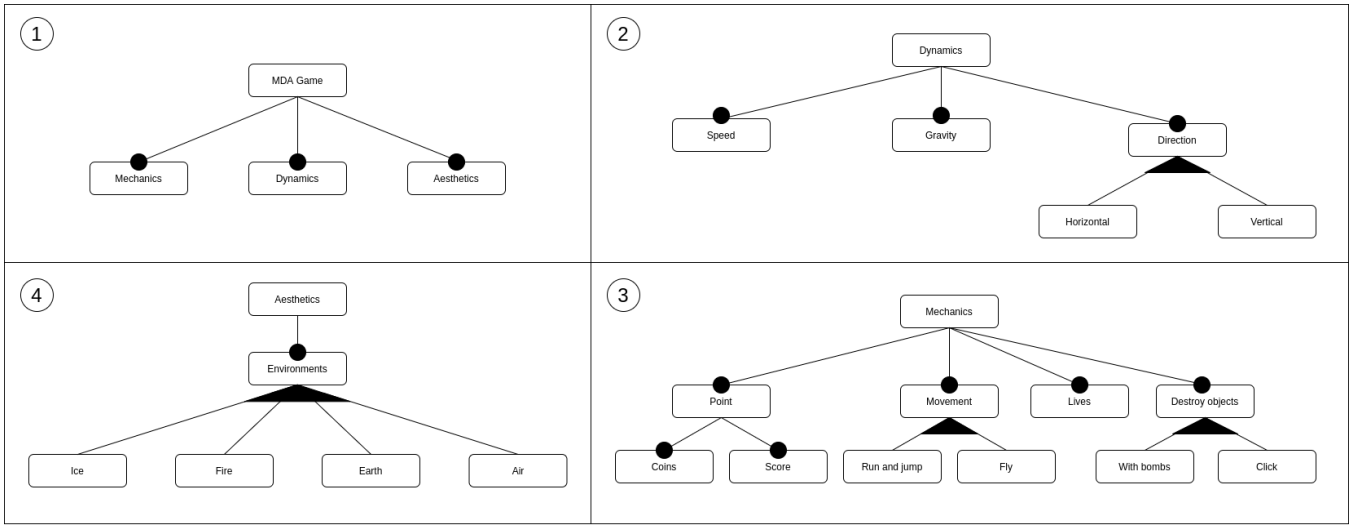

Fig. 3. MDA Feature tree.

and take time to develop. The time it takes to build a mod can vary a lot. Being built in a few days, or taking a lot of development time, having the advantage that some elements will be reused. Depending on the nature of a mod, it may require only one or more releases. For example, a mod that improves a texture in a game may need only one version. On the other hand, a mod that does a full review may require one or more releases [1] [13].

There are several ways to create a mod, among the most common ones are: having access to the game's original code, be it open source or provided by the game's producer, using reverse engineering, or even modify the game through a platform provided by the producer as, for example, World of Warcraft provides a UI customization tool that allows add-ons to reconfigure the user interface panel [14].

As mentioned so far, the process of creating a mod can be time-consuming, complex, and with many challenges, hence the need to automate the process with, for example, the use of SPL. Through SPL, it would be possible to solve most of the problems described above, increasing the number of mods and decreasing the launch time. It would be possible to have a product line where from several fans could start and build games together. The games created would be branches of the tree, where the new games could already serve as input for new ones. This idea could already help in minimizing the rework and increase the number of mods created, since all fans would know the new mods being created, being able to view the modifications made by the product line tree and use a mod that is being built by another fan. Understanding the source code would not be resolved directly but could be improved, as many fans were creating new mods. The most inexperienced programmers could rely on these mods to understand how they could create theirs. For an effective improvement on the understanding of code, it would be necessary to include other SR approaches, such as componentization. In this way, one could think of a programming strategy supported by components where there would be mutators or components that could be applied or added to the games to their extension or evolution.

The prototype created aims to demonstrate the possibility of automating the process, having a product line where the original game is the core of the game's functionalities. The user himself/herself could generate modifications to the game from the use of mutators according to the dimensions of MDA framework. From the initial prototype, it was possible to observe two distinct views in the game product line: a more classic one where each node is a feature, and to create the game, one must go through the tree adding the elements (view that was studied in this work), and another that still needs to be exploited where each edge of the tree can be a mutator that when applied to an original node generates a new, node being a new game with different characteristics.

V. Final Remarks

This work sought to increase the systematization of game mods construction process through the use of SPL concepts. A prototype was created to check the viability of the idea. It was created by applying the SPL approach to create new mods at runtime. From the prototype, it was possible to visualize the notion of systematizing mods by SPL and imagine new ideas for future work.

It is intended to evaluate the game with experts to validate the idea of creating mods from product lines. From this evaluation, new requirements will be collected to improve the process described throughout the paper.

References

[1] D. Lee, D. Lin, C.-P. Bezemer, and A. E. Hassan, "Building the perfect game–an empirical study of game modifications," *Empirical Software Engineering*, pp. 1–34, 2020.

[2] wepc, "Video Game Industry Statistics In 2020," https://www.wepc.com/news/video-game-statistics/, online; accessed 10 January 2021.

[3] J. L. Barros-Justo, F. Pinciroli, S. Matalonga, and N. Martínez-Araujo, "What software reuse benefits have been transferred to the industry? A systematic mapping study," *Information and Software Technology*, vol. 103, pp. 1–21, nov 2018.
[4] R. Keswani, S. Joshi, and A. Jatain, "Software reuse in practice," in *2014 Fourth International Conference on Advanced Computing & Communication Technologies*. IEEE, 2014, pp. 159–162.
[5] C. W. Krueger, "Software reuse," *ACM Computing Surveys (CSUR)*, vol. 24, no. 2, pp. 131–183, 1992.
[6] R. Caillois, "Les jeux et les hommes le masque et le vertige," 1967.
[7] K. Salen, K. S. Tekinbaş, and E. Zimmerman, *Rules of play: Game design fundamentals*. MIT press, 2004.
[8] W. Scacchi, "Computer game mods, modders, modding, and the mod scene," *First Monday*, 2010.
[9] ——, "Modding as a basis for developing game systems," *Proceedings of the 1st international workshop on Games and software engineering*, pp. 5–8, 2011.
[10] D. Sprovieri, "Dynamic re-configuration of software product lines towards an exploratory study on dspls," in *2016 IEEE Tenth International Conference on Research Challenges in Information Science (RCIS)*. IEEE, 2016, pp. 1–6.
[11] R. Hunicke, M. LeBlanc, and R. Zubek, "Mda: A formal approach to game design and game research," in *Proceedings of the AAAI Workshop on Challenges in Game AI*, vol. 4, no. 1, 2004.
[12] K. C. Kang, S. G. Cohen, J. A. Hess, W. E. Novak, and A. S. Peterson, "Feature-oriented domain analysis (foda) feasibility study," Carnegie-Mellon Univ Pittsburgh Pa Software Engineering Inst, Tech. Rep., 1990.
[13] M. M. Hofman-Kohlmeyer, "Players as content creators. the benefits of game modding according to polish users," *International Scientific Journal News*, vol. 2, pp. 8–26, 2019.
[14] W. Scacchi, "Computer game mods, modders, modding, and the mod scene," *First Monday*, vol. 15, no. 5, 1 1. [Online]. Available: https://journals.uic.edu/ojs/index.php/fm/article/view/2965