Towards a Library of Game Components

A Game Design Framework Proposal

Marcos S. O. Almeida
Computer Science Coordination
Federal University of Technology – Paraná (UTFPR)
Campo Mourão, PR, Brazil
marcossilvano@utfpr.edu.br

Flávio S. C. da Silva
Computer Science Department
University of São Paulo (USP)
São Paulo, SP, Brazil
fcs@ime.usp.br

***Abstract*—"Game Elements" is a commonly referred term in games related publications. The notion that games are constituted by parts is widely accepted but no formal definition has been set. Designers commonly play a large amount of games in order to build a knowledge base over games concepts. At the same time, researchers and designers have an agreement upon the need for more formal design tools, highlighting approaches based on collections of reusable game concepts. While some implementations have been created, none have succeeded as practical tools, focusing too much on high level design abstractions. We believe that games can be dismembered into their forming components and these can be structured, analyzed and combined in order aid the build of new games, allowing an engineering based approach through a framework of building blocks-driven game design. This paper discusses the steps towards a conceptualization of this framework.**

***Keywords*— Game design tool; engineering-aided game design; game components; game elements.**

I. Introduction

It is common to find references in game design texts to the idea that games are built upon a specific set of parts. Usually, if we desire to create a platform game, we need to play the major titles of the genre, find out what they have in common, what makes each one unique and what doesn't work for them. As a result of this task, we get a conceptual database of the forming elements of games from this genre. From then on, we start having some tools to aid on the creation of a standard platform game. If we want to go a little further, making some innovations, we may also play games from other genres to find interesting elements that could be mixed with those commonly found on platform games, creating some cross-genre characteristics. Generally speaking, what we have done here is mining the forming elements of existing games in order to use them as a toolbox to the creation of new games. Although this may not represent the entire process of game creation, it does define an important part of it.

While not formally defined, this practice is widely applied by designers, whether they are part of big studios or independent creators. We believe that this practice can be formalized and concretized as a conceptual tool, a game design framework that will produce a library of game components, which could allow component-focused analysis and experimental design through the trials of different combinations of components. Throughout this paper, we will discuss and present the concepts and the structure towards the conceptualization of this tool.

II. A Common but not Formalized Term

Game publications and specialized media have been implicitly or explicitly referred to the idea that games have forming elements. When reviewing a game and pointing its main features, authors usually describe them as having elements that define or contribute to its gameplay and user experience. These elements can be described as the unities that constitute the key characteristics of games, varying from gameplay to narrative aspects of the storyline. The idea that one game borrows elements from others is often referred. Furthermore, there is a common understanding that game genres and themes, such as adventure, action, RPG or horror, define characteristic elements for its games. Thereby, cross-genres titles have elements from the genres that classify them. Overall, those "game elements" are a common part of the designers and analysts vocabulary. In that sense, the concept that games are constituted by parts is largely applied but no formal definition has been set. In this context, "formal" does not refer to mathematical models, but to organized, standardized and structured models and tools to aid the game design process.

In any product creation craft, it is vital to creators to keep up to date with the achievements of other professionals. Two facts can be highlighted about that need: we want to know what the competition plans are and it is vital to have an overall picture of the ideas already implemented, to decide if they can benefit our work. This is no different in games production. Like any other usual player, game designers play lots of games, though with a different purpose. As a very informal practice and an important part of their craft, designers constantly experience a vast amount of games through an analytical perspective in order to increase their knowledge over the elements that define them and lead to their uniqueness. The existing games typically act as a foundation to new ideas and designers want to find what works for them.

Researchers and designers have an agreement upon the need for more formal and standard design tools [1][2]. Game studies publications have mostly discussed this need and presented approaches that vary from the building of a shared design vocabulary to the creation of visual modeling languages for game design. One of its highlights is an approach based on a collection of reusable design concepts. While some implementations based on this approach have been made, none have succeeded as a design tool of practical use. Although some authors have urged for a tool that allows designers dissect games in order to understand how their parts balance and fit together, the implementations have focused on defining a high level set of abstract concepts rather than game forming components.

As observed, in these three presented scenarios – publications, industry and academia – there have been common references to the notion that games can be analyzed and created by looking to their forming components, but no previous work has specifically addressed this as a possible design tool. As a convention and a formal definition, in the framework described throughout the present work, we have chosen to use the term "game components" instead of "game elements" because we understand a component as the structured part of a whole. By this structure, we mean a very strict form of description, relation and modeling of game components.

III. Related Work

Game designers and researchers have developed a discourse around the need for more formal and standard design tools [1]. They consider the main current design tool – the design document – as too restrictive, creating a barrier to the evolution of the game design area. Costikyan [3], one of the first authors to address the issue, suggested the search for a tool based on a common vocabulary of design concepts. He believed that designers should have a way to analyze games, understand them and identify the elements that make them good or bad. Others authors that, directly or indirectly echoed Costikyan's speech, made references do the same idea. Church [4] suggested the lack of a common design vocabulary as the main inhibitor of the game design methods evolution. Fullerton (2008) followed the same direction.

The first attempt to establish a design tool based on a collection of design concepts was Church's FADT (Formal Abstract Design Tools). Church [4] indented to be able to dissect a game, identify and separate their forming components in order to understand how they fit and balance together, thus analyzing which ones benefit or harm certain games or game genres. Although his work hasn't been concretized as a tool, it certainly inspired other authors. Falstein and Barwood [5] initiated "The 400 Rules Project" aiming to identify, to record and to share practical experiences of designers as guidelines. In a more structured way, Björk, Lundgren and Holopainen [6] were based on the design patterns model of software engineering [7] to define design concepts as Game Design Patterns. Although more structured, the tool fails to deliver a practical tool by lacking adequate software support and documentation, and by not having enough correspondence between patterns and the games or genres that use them.

Some years later, LeBlanc, Hunicke and Zubek [8] theorized the MDA, a framework to organize and describe elements of games under three layers – mechanics, dynamics and aesthetics – but no collection of elements was provided. More recently, Järvinen [9] presented an academic study about theories and methods of design and discussed the creation of a library of game mechanics at a highly theoretical level. Currently, the GiantBomb[1] website is the only attempt to maintain a practical source of design concepts, but remained fairly informal and aimed at end users. Although at an academic level the proposed approaches brought contributions to the area in the form of improved discussions over the issue early addressed by [3], as practical tools they show an evident lack of maturity and computational support for adoption and experimentation in real world scenarios.

Some authors have analyzed the use of visual languages to aid the game design process. They have found in the visual modeling a strong ally in communicating the designer's vision of the game to the development team. Librande [10] emphasized that visual models are more synthetic, naturally communicative and scale better. He presented the One-Page Design, a comprehensive game design map freely created with textual and visual artifacts. Other authors have suggested the use of more formal approaches to build game design diagrams. Demachy [11] and Blumenthal [12] have shown examples of application of the software engineering UML [13] for this purpose. Sicart [14] presented an approach for game design based on the object-oriented model, suggesting the use of UML.

A key element that has not been explored in the previous works discussed is the integration between the collection of design concepts and the visual language for design modeling. Diagrams of the design of a game could be made through the gathering of the concepts that the game implements. It would make easer to see the interrelations between these concepts and, the games and genres that employ them. Beyond that, it would take the visual documentation to a more concrete level, once the concepts of a game wouldn't be more just a collection of scattered game parts.

The Game Components Framework (GCF), subject of this paper, will be discussed in the following sections. It is a doctoral project aligned with the earlier speeches around the need for game design tools published by [3] and [4], having as main objective allowing designers to analyze and to create games through their forming parts. Furthermore, it is fundamentally a fusion of the two game design tools approaches presented in this section: the constitution of a common design vocabulary, through a collection of design concepts, and the standardization of visual design maps built with a formalized language, to express the concept interrelations and their use in games and genres.

IV. The Game Componentes Framework

In opposition to some heavily theoretical attempts to bring game design tools, this paper approach is strongly based on real world observations. By this observation, we mean the experimentation and analysis of games. We do not intend to perform deep studies about the theories behind games, engagement, "gamification" or player interaction. The GCF was idealized and will be built upon the existing and

[1] http://www.giantbomb.com/concepts/ (visited on 2013/03/13)

consolidated results of the designer's work, which are the games they have created. We believe that there is an extensive, though not formally addressed, design knowledge base embedded into the vast library of the existing games. In this sense, we also believe that new games can be planned as a composition of interconnectable smaller parts, whether they are designed for the game or reused from existing titles. Also, by documenting the forming components of games, we expect to understand the relations between games, their forming components and genres, in order to discover which lines can be traced between games and their commercial and critical success or failure, thus aiding in the creation of new games. Considering that all of the existing games reuse or combine concepts found on past games somehow, this work of practical observation may bring a significant contribution to the area.

Generally speaking, the GCF establishes a formal structure that helps designers to see games through a composition of small forming parts and their relations. In this sense, the framework serves both as a design and analysis tool. As a collateral effect, the resulting components will be stored into a library, allowing querying and reusing of these, thus encouraging design experimentations through combinations of game components.

A. What are game components?

Some authors refer to term "game mechanics" as the components of game or gameplay [9] [8] [14]. Others describe games as systems of rules [15]. However, a game uniqueness doesn't rely solely on its mechanics or rules. Aside from the obviousness that part of a game identity is built upon its sound and artistic style, there are many other characteristics that lead to its uniqueness. The game controls, narrative, point of view, level design and even the HUD (head-up display), have a strong influence on how the player experiences the game. In that sense, a game component is any recognizable part of a game that can be identified, separated and presents some aesthetical influence. It can be a rule, a mechanic, a narrative aspect, a type of camera or any other part of game that makes it work. It can be very specific to a game implementation or generically applicable to most games of a genre or a platform. It can be a very small part of a game, at the lowest level of structural granularity, or comprise a more abstract concept. Overall, by looking to a visual model of game components a designer must be able to understand the game main characteristics.

Based on the definitions from the MDA framework [8], we want to reach the designer's perspective by a reverse path, starting from the player point of view. Thus, by "reverse engineering" the game through the player's perspective we will identify components that directly or indirectly have significance over the gameplay experience. For this reason, game components cannot be restricted to mechanics or rules.

If we look to the contemporary games, we will see that many of their characteristics were borrowed or modified from past games, whether they belong to the same genre or not. To demonstrate this idea, let's take by example the platform games genre, also known as "platformers". Two earliest well known platformers are Super Mario Bros [16] and Super Mario Bros 3 [17]. By comparing these games with others of the same genre, like Kid Chameleon [18] or Donkey Kong Country [19], or more modern titles, such as Braid [20] or Super Mario Galaxy 2 [21], we clearly perceive that they all borrow components from the two earlier games. The main component of Mario's attack, the "head stomp" movement, reappears in all of those games (and in many others). Another component of Super Mario Bros 3, the "world map" – the main place by which all of the game stages can be accessed – it is often reused by many platformers.

Besides borrowed, components can be modified or combined to form new ones. As an example, the "head stomp" found in games like Ducktales [22] and Castle of Illusion [23] slightly differs from Mario's attack, needing a button press to be performed. Little Big Planet [24], by the other hand, allows the "head stomp" attack only on certain levels. Moreover, this process is not restricted inside one genre only. In the game Retro City Rampage [25], an open-world action-adventure, the player's character uses "head stomp" attacks, as well as Chun-Li, a character from Street Fighter [26] games.

Within the context of the examples cited, one may ask: why have designers opted to adopt the "head stomp" attack in their games? One probable explanation is that the games where this component was first seen, Super Mario Bros 1 and 3, made a huge commercial and critical success, being considered icons of their genre. Also, aesthetically speaking, the "head stomp" strongly contributes to the main game element of platformers: the "jump". There is an intrinsic relation between these two elements that may lead to some questions: it is possible to use the "head stomp" without the "jump" element? What attack elements are used in other platformers that don't have a "head stomp"? The practice of studying the constitution of success cases may improve our knowledge over the designer's craft and add a powerful tool for reusing and experimenting with past good ideas.

Through the discussion presented in the examples, we can observe the existence of a complex inheritance relationship between the games through the components they borrow, modify or combine, which can lead to the elaboration of inheritance maps. If we could trace a chronological line relating these maps and the commercial or critical outcome of the games, we would be able to cross and extract valuable data that could help us to identify what components can possibly contribute or harm new game projects. In this sense, a framework for describing games as grouping of components alone can't answer these questions, but if we store the designed components into a database, we can.

B. How will designers use the GCF?

The GCF represents a top-down approach in which complex game systems can be decomposed into smaller, simpler parts. It also defines a "components-driven" approach to the design in which new games are built by a set of parts. Furthermore, it is a tool based on software engineering object-oriented approach and as such, the Unified Modeling Language (UML) serves as the basis for a specific tailored modeling language to express games through its forming elements.

The GCF tool is comprised of three parts:

- A structured framework for describing components (**Components Framework - CF)**;
- A library of the designed or reverse engineered game components (**Components Library - CL**);
- A visual language for modeling games through its components, as well as, to represent components interrelations (**Components Design Language - CDL**).

In the CF, components are described through its definition, attributes and its relations with games, genres and other components. Every game component, whether designed for a new game or reverse engineered from an existing one, is stored in the library for future reuse (CL). Also, the library enables analytical views by different dimensions, such as genre, platform or time, based on data crossing with market and critic data. Finally, modeling of games and relations between components is done via visual diagrams with a language tailored from UML to the specificities of the game components approach (CDL).

In the GCF, the diagrams made via CDL are the key expression tools to design game through components, as well as, to the relations and specificities about their use. Although composed of more formal structured elements when compared to Librande's On-Page Designs, the CDL doesn't imply into more restrictions to the game designer. They are allowed to merge illustrations, concept arts, schemes, game or character pictures, within the diagram. Also, it is possible to embed small graphical elements into the game components to visually iconize them. The overall concept of the CDL game diagrams is to facilitate the comprehension of the designer's vision and to contribute to the communication across the development team.

As a data analytical tool, the library (CL) acts as a searchable design database that allows navigation along components, diagrams and modeled games, and visualization of statistics about components use, gathered from market and critics data. By combining the tree parts of the GCF, it allows designers to start a blank game project, search the desired components and select them for use. They can also start a project from a combination of existing games.

C. *What are the framework features?*

To summarize every feature described so far, we present a simple list of the key characteristics of the GCF and its approach:

- It is a **design framework** as, it forces designers to plan games through its small forming parts that are later related and grouped into bigger components, until the game is fully described.
- It is an **analysis tool**, as it forces designers to see and describe the features of the existing games as components. Also, it presents analytical data about relations between components and different dimensions, such as genres, sale numbers, platforms, users and chronology of game releases.
- It helps to define a **design vocabulary**, as it standardizes names for every piece of a game.
- It is a **knowledge library** of game design, as it stores every game component defined, whether designed for a new game or reverse engineered from an existing one, allowing the querying and reusing of these components.
- It is a **top-down approach**, as complex game systems can be decomposed into smaller, simpler parts.
- It is a **building blocks-driven approach**, as games are built by a set of components, which can be used with, compose, be composed of, specialize or generalize other components.
- It is an **experimental driven approach**, as it allows designers to try combinations of game components from previous created games.
- It is an **engineering approach**, as it forces the conception of games by very strict construction rules as oppose to focus on narrative and artistic methods.
- It adds **standardization** over the current game design tools, as it allows traditional documentation to be generated from game models and game components.
- It is a **computer-aided design**, as the library and the modeling are made through a software tool built to help designers to work with the approach.
- It is a **communication tool**, as it provides a standard components vocabulary and a way to create visual maps of game design that can be hanged on the wall to communicate the designer's vision to the development team.
- It is **built upon proven standards** as it's strongly influenced by the approaches of object-oriented software development and the Unified Modeling Language (UML), as described in the section V.

D. *What differentiates it from other approaches?*

The GCF works towards the same directions of the approaches discussed in the section III. Some of these approaches focus on defining a framework for collections of reusable design concepts. Others worked towards the establishment of a visual language aiming to express the design of games through diagrams. In this sense, the GCF combines these approaches in one solution, comprised by a framework for description of games via its forming parts (the components), a library to stores and manages the described components and a visual language to model the design of games by using these components. Apart from this, what differentiates the GCF from the other approaches? As discussed in section IV.B, the GCF is comprised of three main parts: the framework for describing the components, the library to manage them and the visual language to apply them in the design of games.

Approaches based on collections of concepts proposed so far are more abstract and high level compared to the GCF. These approaches are most based on collections of aesthetical elements of games. As an example, “Perceivable consequence” is a FADT [4] and a game design pattern [6]. Its definition is “a clear reaction from the game world to the action of the player”. Another example is the pattern “Exaggerated Perception of Influence: Players perceive that they can influence the outcome of the game, regardless of whether this is correct or not”. These concepts are clearly concerned in describing the outcome of some game parts regarding to the player experience, and not the parts itself. On the other hand, the GCF aims specifically to the parts of games, which consequentially will promote aesthetics. As an example, the “Third Person Over-the-Shoulder” is a kind of POV (point-of-view) that is a very characteristically component of modern shooter games. Originated in Resident Evil 4 [27], it generates more “immersiveness” than the traditional “Third Person Behind-the-Character POV” and still allows the player to view its avatar in very close details, valuing the graphical quality of the character. The employment of such component brings a lot of difference in the final player experience and that’s why so many newer shooters have done it.

Overall, the previous works are most concerned on game aesthetics, towards a more narrative perspective of the game. On the other hand, the engineering perspective, which fundaments the GCF, is concerned on how things work and on how they are built. In this sense, we aim at the forming parts of games, and consequently, their outcome as gameplay experiences. The engineering perspective also brings another concept: the elaboration of game designs through grouping of game parts. This approach is based on building blocks, which allows a top-down analysis of games by decomposing them into its forming parts and a bottom-up construction process, in which games are built upon a set of elements. In this sense, the components work as implementation objects, whether concepts work as guidelines. Although the later do have relations, it’s not possible to create a game design by just selecting a group of concepts.

In the GCF, designers will describe games as an interrelated group of components. By doing that, they are populating a library of components that allows reusing and analysis of them. In this scenario, the composition construction has as key role: components can be reused and blended in order to create "bigger" ones, which in turn, may compose entire games. Games can also be seen as components, which allows new games to be designed as a grouping of entire games or as a mixture of some of their components. Furthermore, with a comprehensive library of components, it’s possible to use data mining techniques and crossing of information, such as market and critics data, in order to answer questions like “What are de most successful applications of specific components?”. Also, it’s possible to apply techniques that will inform how much close are a group of games in terms of their forming parts, or even, what successful titles of a particular genre have in common. Similar approaches already discussed don’t allow this kind of tool as they don’t focus on their proposed frameworks and not in the operation of the collection of concepts itself.

The third and final part of the GCF is the design visual language (CDL). Other approaches based on collections of concepts don’t define a language that allows designers to build schematics of their designs based on these concepts. On the other hand, there were studies of application of proven visual languages for game design, but they were mostly based on application of raw UML into game projects and did not tailored the language for the designer’s specific needs. Also, the UML cases are totally disconnected from the approaches based on collections of concepts. The only approach that allows designer to express the vision of a game through a visual representation is Librande’s One Page Designs, but the elements used to draw these design maps don’t have standardization and thus, reusing is not a conceivable option.

Lastly, the GCF works towards the discourses of Costikyan (1994) and Church (1999), which highlighted the need for a tool for dissect a game, identify and separate its forming components, understand how they fit and balance together, and analyze which ones benefit or harm certain games or game genres.

V. THE ENGINEERING BUILDING BLOCKS APPROACH

The heterogenic nature of the people who have worked as game designers often leaded to different comprehensions of what defines the designer’s craft and in which it relies. In the earlier days, a game was basically built by one person who had software development skills. As programmers, they had a very strict engineering way of doing things. As the time passed by, games gradually became more story-oriented and narrative skills started to play an important role in its creation. However, as Costikyan [3] pointed, games do not have the unique function of telling a story as they don't follow a linear structure: the outcome is result of the player actions. LeBlanc, Hunicke, and Zubek [8] presented a similar perspective, emphasizing the dynamic nature of games as systems that exhibits emergent behaviors through gameplay experiences. In this sense, even that some authors have advocated a strongly narrative-driven development with minimal constraints [28], a structured approach is better suited. Thus, the approach presented through this paper is strongly influenced by methods and tools of software engineering.

A. Drawing inspiration from Object-oriented paradigm

The key concept of the OO development paradigm is the abstraction of real world things and concepts into autonomous components called objects, which process its own data and communicates with other objects. Programs are seen as a collection of interacting objects. Each object has attributes, which qualify it, and execute actions through methods. Objects are instances of classes. Each class defines the type of many objects, which are called instances of the class. Thus, an object can be described as the application of a class into a specific scenario. Each object only belongs to a single class. Through OO, software are planned, designed and built by their forming components.

A game component is a part of a game that exhibits influence over the gameplay experience. A game can be seen as a collection of interacting autonomous components. A component may relate to others: it may uses, depends on, generalizes, specializes or even composes other components. It has its own attributes that vary according to its use in games, helping to define the game uniqueness. In this sense, the GCF has close relations with the OO paradigm. More formally, a component is an OO class and its application in a game, an instance, like an OO object.

The structure of the game components is presented in the following text. We will analyze the relations of the present approach and the OO paradigm. Samples of the CDL, the components design language used for games diagramming, are provided throughout the paper for illustration purposes. However, these must be taken as sketches of an initial planning, once the GCF it's in early development stages as a doctoral project.

Although a game component definition explicitly draws inspiration from the OO paradigm, some fundamental distinctions must be made. As an abstraction from real-world objects, the OO would imply that all the objects from the game world should be mapped to game components, such as characters, enemies, vehicles, buildings, bullets, explosions or even environment parts, like trees and rocks. However, not all of these objects have explicit meaning over the gameplay experience. By the other hand, a jump movement, an auto-aim feature or a spin attack strongly influences and differentiates the player’s experience when playing games. In this sense, the objects that we are interested on are not those which necessarily abstract the game world objects.

Another fundamental differentiation of the GCF from the strict OO approach is regarded to what constitutes a game component. A class has its own attributes and actions. However, game components can be both abstractions of objects from the game world as actions. As an example, in the context of the OO approach, a “jump” may be an action of a “character” object, which means, it cannot be detached from it and has no meaning by itself. Actions belong to objects and must be attached to them. However, the “jump” is a part of a game that brings meaning to the player’s experience. As an example, the lack of the “jump” component in a horror-adventure game may lead to restriction and tension, which are desired player emotional responses in this genre. On the other hand, in platform or action game genres, the same lack would probably lead to frustration. Thus, a “jump” is an autonomous part of a game in the sense that it brings aesthetical significance when inserted or removed from a game design. Thereby, differently from a strict OO approach, in the context of GCF, a “jump” is a component by itself, which may be attached to the “character” component.

The Fig. 1 shows a graphical representation of the “jump” component and its instance in a game with the CDL. These representations are inspired in UML Class Diagram. The component (Fig. 1, left image) describes a more generic game part, containing attributes that define its characteristics. It is described by name, category (type of component) and short explanation. When used in a game composition, the component becomes an “instance” (Fig 1, right image), which is, an application of the component to a specific scenario. Then, the component attributes must have its values defined and optional game descriptions and illustrations may be attached.

Instances in games are made through games components. For every instance in a game, there must be a component. Any

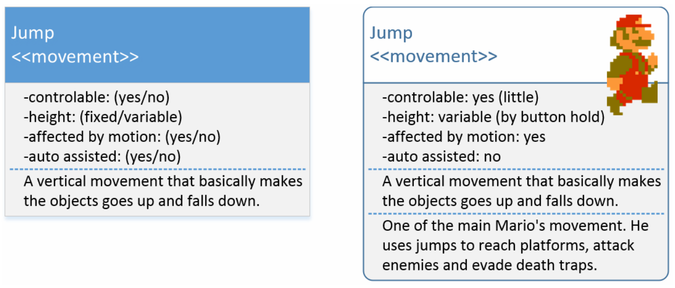
Jump
<<movement>>
-controlable: (yes/no)
-height: (fixed/variable)
-affected by motion: (yes/no)
-auto assisted: (yes/no)
A vertical movement that basically makes the objects goes up and falls down.
Jump
<<movement>>
-controlable: yes (little)
-height: variable (by button hold)
-affected by motion: yes
-auto assisted: no
A vertical movement that basically makes the objects goes up and falls down.
One of the main Mario's movement. He uses jumps to reach platforms, attack enemies and evade death traps.

Fig. 1. Visual representation of an component in CDL. The left image shows the component structure. The right image shows an instance, an application of the Jump component in the Super Mario Bros game.

modification in the structure of a component must generate a new component. In other words, if a game needs to apply an existing component in its design, without modifications on the component structure, it just has to instantiate it. However, if the designer needs a slightly different version of an already existing component, he needs to extend it (as explained in the next section) and generate a new component, which now can be instantiated into the design of the game.

B. Use, composition and inheritance relations

As another reference to the OO paradigm, components exhibit relations between them. The possible relations are: use, composition and inheritance (generalization or specialization). The “use” relation defines that two components or instances interact in some way. In the example provide by Fig. 2, the Mario’s action of throwing a turtle shell is modeled as follows: an instance of a “Throw Object” action component uses a instance of a “Bouncing Object” *throwable* component. The specific details of the two instances are shown in their attributes and descriptions.

The composition relation implies that a set of components can be grouped to form a new “bigger” one. These “bigger” components can also be grouped to form new ones. There are no restrictions to the “size” of a component, which is, the number of component it contains. Both “smaller” and “bigger” component can be related to other components, games or genres. By this simple mechanism, every game character can be expressed as the composition of two or more component, if desired by the game designer. Fig. 2 shows an example of how composition can be used in games diagrams to represent a character. The Piranha Plant character contains two instances of components with their attributes values defined. After designed, it can be included in the library as a new game component. Furthermore, a whole game can be represented as a component in the library. This allows a game to be easily expressed as the union between two or more games. The composition relation gives a clear perception of building blocks with a top-down approach: complex component can be dismembered into simpler ones.

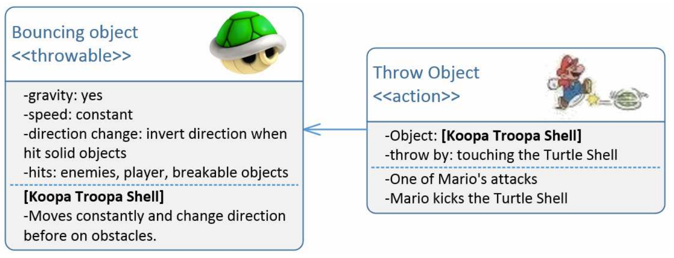
Bouncing object
<<throwable>>
-gravity: yes
-speed: constant
-direction change: invert direction when hit solid objects
-hits: enemies, player, breakable objects
[Koopa Troopa Shell]
-Moves constantly and change direction before on obstacles.
Throw Object
<<action>>
-Object: [Koopa Troopa Shell]
-throw by: touching the Turtle Shell
-One of Mario's attacks
-Mario kicks the Turtle Shell

Fig. 2. Example of "use" relation between instances of components for the Super Mario Bros game in order to represent the "throw" design concept.

The third type of relation allowed in game components is the inheritance, another fundamental construction of the OO paradigm. As an abstraction of the real world, the inheritance allows a class to include the characteristics of another class. In this relation, the inheriting class is known as sub-type or specialized class, and the inherited is known as super-type or generalized class. This mechanism is called an "is-a" relation to symbolize the fact that the sub-class's type becomes the same of the super-class. To the context of the GCF, components may specialize a more generic component to specific uses, maintaining the originator characteristics. However, a "type-of" relation is better suited to describe the kind of inheritance relation between components.

The Fig. 3 shows an example of inheritance relation between some different types of jump components. As illustrated, the "jump" component is the most generic type of jump. Other sub-types of jump elements, like "attack jump", "assisted jump", "triggered jump" and "multiple jump", inherit the "jump" component attributes. The relations presented in the example may continue expanding as long as new types of jump components are being identified. In this sense, a "head stomp jump", common in platformers, and an "aerial attack", found in beat'em up games, would be types of "attack jump". Other examples would be the "double jump", a common type of "multiple-jump" found in side-scrollers, and the "chain jump", found on Super Mario 64 [29] and a type of "enhanced jump".

Another example of the inheritance flexibility is the use of metaphors. A designer can describe a character of its own project as a modified version of a character from another game. Let's take an example. If we want to explain a character behavior as "a Piranha Plant that fires a glue ball" we can make this character inherit the Super Mario's Piranha Plant (see Fig. 3) and add the glue ball component. Just like in software design, the combination of composition and inheritance brings a powerful tool to game design modeling.

The three relations types presented in this section – use, composition and inheritance – can be freely used in a components diagram of a game by designers when meaningful to their vision of the game representation. As in software engineering, the game modeling activity has as inherent abstract interpretation, which means that different designers can create slightly different map versions of the same game. This is due to the fact that modeling is an activity of creation and by no means can be restricted to one absolute interpretation. It is the direct result of the modeler's vision.

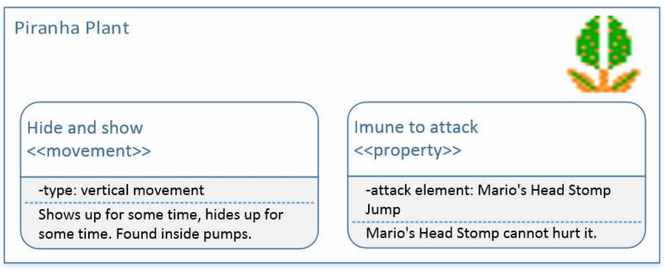
Piranha Plant
Hide and show
<<movement>>
-type: vertical movement
Shows up for some time, hides up for some time. Found inside pumps.
Imune to attack
<<property>>
-attack element: Mario's Head Stomp Jump
Mario's Head Stomp cannot hurt it.

Fig. 3. The Piranha Plant instance, a NPC character from Super Mario Bros. Its is composed by two other components that represent its behavior.

C. *The Game Components Library (CL)*

The documentation of the game components in the library follows a very strict structure, similar to those found on the Game Design Patterns [6] and the collection of concepts from the GiantBom website. Each game component must be uniquely identified and clearly defined in order to facilitate its use and recognition in games. The documentation structure is organized into two sections: component definition and relations.

Each component is documented by name, description, category, attributes and consequences of use. The "name" field must be unique and as short as possible. The "definition" shall include illustrations of the concept, which can be schematics or games screenshots. The "attributes" are the component key characteristics and have its values defined when the component is used in a game as an instance (Fig. 1). The "consequences" field describes the implications of using a component in a game. Some implications are usually related to aesthetics and can be defined when an instance of the component is made.

The second section of the documentation of an component in the library is composed by its relations with games, genres and other components. The relations of components with games and genres are the following: games that instantiate the component; game where it had first appeared; genres that it

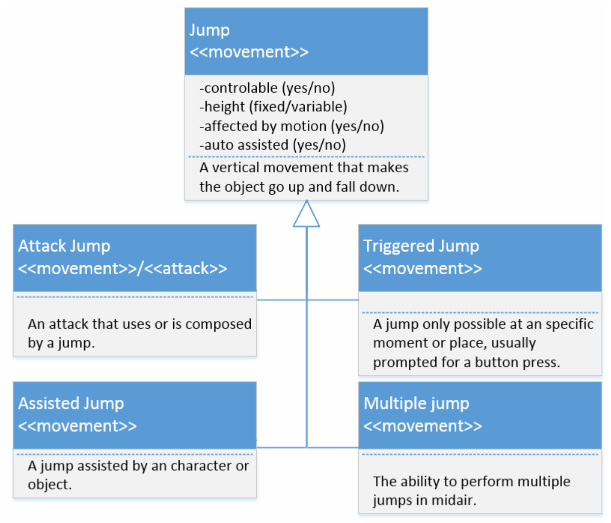
Jump
<<movement>>
-controlable (yes/no)
-height (fixed/variable)
-affected by motion (yes/no)
-auto assisted (yes/no)
A vertical movement that makes the object go up and fall down.
Attack Jump
<<movement>>/<<attack>>
An attack that uses or is composed by a jump.
Triggered Jump
<<movement>>
A jump only possible at an specific moment or place, usually prompted for a button press.
Assisted Jump
<<movement>>
A jump assisted by an character or object.
Multiple jump
<<movement>>
The ability to perform multiple jumps in midair.

Fig. 4. A example of inheritance relations between jump components.

defines; and, genres that use it. The relations between components are those previously presented in this paper: use, composition and inheritance. Thus, the possible relations are: inherits from; is inherited by; uses; is used by; composes; is composed by; and; frequently used with.

Together, all the documented relations are necessary to allow data crossing between components, games, genres and general reception data by specialized media. By associating the information documented in the library with data about market sales and critics scores of games, gathered from specialized websites, chronological lines can be traced to aid in the discover of what components may had helped a game to achieve its success or failure, thus contributing to the design of new games.

VI. EARLY EXPERIMENTS AND FURTHER EVALUATIONS

The framework is in early stages of development. The library software tool is also being built. Some first experiments with sketches of the visual language were done for Super Mario Bros., as already shown in the section V. The game was used as a case study for the development of the GCF. Through reverse engineering of the game, the components definitions, the framework structure and the visual language are being tailored to best fit the engineering approach for game design. This process will be iteratively performed with other games in order to evolve the GCF.

On Super Mario Bros., more than sixty components were documented over various categories: movement, property, NPC behavior, "throwable" object, status, action, reward, attack, enemy, level construction, item displacement, level progression, environment, hud, multiplayer, game progression, world structure, item, power-up and pov. These categories are still under study and will change. A large percentage of the documented components presents reusing cases on a many other games, including platformers and other genres.

When mature enough, evaluations with the GCF will be conducted by employing the tools into game development projects as cases studies, in order to discover how the GCF will influence the development project in both positive and negative ways.

VII. FINAL THOUGHTS AND NEXT STEPS

The concept that games are constituted by parts is widely applied but no formal definition has been set. Game publications and specialized media have been implicitly or explicitly referred to the idea that games have forming elements. On industry, designers commonly experience a vast amount of games to increase their knowledge over the parts that makes than fun and unique. At the same time, researchers and designers agree upon the need for more formal design tools. Attempts have been made to create collections of reusable game concepts, but none have succeeded as practical tools.

The concept that games can be thought as a assembling of smaller and structured components is fairly discussed. The problem is not the concept itself, but how to structure these components. Here lies the real challenge: how to conceive and bring to real world usage a database of design knowledge easily accessible that can be applied to the analysis and design of games in a productive way.

As pointed by Costikyan [3] and Church [4], designers would have a way to analyze games and dissect them, thus identifying, separating and understanding how their forming components work and recognizing which ones benefit or harm certain games and genres, thus helping to establish a shared vocabulary, an ontology of game design. The Game Components Framework (GCF) was planned towards this discourse. By reverse engineering the game through the player's perspective of the gameplay, we can not only recognize the game components, but also their aesthetics. From an engineering perspective, recognizing games as a composition of smaller components makes possible to design them as object-oriented (OO) systems, a proven approach long applied in software engineering. Thus, we can benefit from tools strongly consolidated in this area by tailoring them to the needs of the game design. By applying the OO concepts to game components we can understand them as objects that have attributes and can be used with, compose and inherit other objects. We can also modify UML diagrams to create visual maps of these relations and their applications in games and genres, leading to a very synthetic practice of game design diagramming. In this practice, we can use, mix and discover components in games and genres, allowing an experimental and building blocks-driven design approach.

The GCF will allow designers to analyze the forming components games and to create new games concepts from these components. More than that, when accomplished as a concrete software tool, it will enable designers to discover essential facts about games such as:

- The core, common and uncommon characteristics of specific games or genres;
- The usual characteristics of a user profile;
- The often characteristics found in better and worse rated games of a genre;
- The characteristics often borrowed from one genre to another.

Further enhancements may be made over the GCF. As an example, concepts from other computer science areas may be borrowed and specifically tailored as game design tools. If games can be seen as groupings of parts stored in a database, we may use a "Games Query Language" to cross data and discover interesting facts and tendencies about games, genres, market and their users. We may also apply techniques from Data Mining [30] to discover commonalities between games through its compositions, more specifically to define how far or how close two or more games are, thus allowing the generation of family trees of games. Furthermore, as a long-term planning, if components become concrete game software objects, designers will be able to play test the conceived games through a prototype generated from the component diagrams of these games.

As seen, many further tools may be envisioned starting from the concept presented in this paper. They range from a component-driven analysis and design to the construction of prototypes from components associations. Thus, the GCF will serve as the basis for an entire game design experimentation environment.

REFERENCES

[1] K. Neil, Game Design Tools: Time to Evaluate. Proceedings of DiGRA Nordic 2012 Conference: Local and Global – Games in Culture and Society, 2012.

[2] J. Dormans, Engineering Emergence: Applied Theory for Game Design. Teste de Dou-torado. Amsterdam University of Applied Sciences, 2012.

[3] G. Costikyan, I Have No Words & I Must Design. Interactive Fantasy, Eng., n2, 1994.

[4] D. Church, Formal Abstract Design Tools. Gamasutra, Jul. 1999. http://www.gamasutra.com/view/feature/3357/formal_abstract_design_tools.php

[5] N. Falstein, H. Barwood, More of the 400: Discovering Design Rules. Presentation at GDC 2002. http://www.gdconf.com/archives/2002/hal_barwood.ppt

[6] S. Björk, S. Lundgren, J. Holopainen, Game Design Patterns. Level Up - Proceedings of Digital Games Research Conference (DiGRA). Utrecht University, 2003.

[7] E. Gamma, R. Helm, R. Johnson and J. Vlissides, Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley, 1994.

[8] M. LeBlanc, R. Hunicke, R. Zubek, MDA: A formal approach to game design and game research. Proceedings of the AAAI-04 Workshop on Challenges, 2004.

[9] A. Järvinen, Games Without Frontiers: Theories and Methods for Game Studies and Design. Doctoral Dissertation. University of Tampere, 2008.

[10] S. Librande, One-Page Designs. Presentation at GDC 2010, San Francisco CA, Mar. 2010. http://stonetronix.com/gdc-2010/

[11] T. Demachy, Extreme Game Development: Right on Time, Every Time. Gamasutra, http://www.gamasutra.com/view/feature/2827/extreme_game_development_right_on_.php

[12] R. Blumenthal, Space Invaders: A UML Case Study. Regis University,class notes, 2005.

[13] M. Fowler, UML Distilled: A Brief Guide to the Standard Object Modeling Language . Boston, MA: Addison-Wesley, 3a ed, 2004.

[14] M. Sicart, Defining Game Mechanics. Game Studies: The International Journal of Com-puter Game Research, v. 8, n. 2, dez. 2008. http://gamestudies.org/0802/articles/sicart.

[15] K. Salen, E. Zimmerman, Rules of Play: Game Design Fundamentals. MIT Press. MIT Press, 2003.

[16] Nitendo, Super Mario Bros. Nintendo Enterteinment System: NES [Cartridge], Japan: Nintendo Co. Ltd, 1985.

[17] Nitendo, Super Mario Bros 3. Nintendo Enterteinment System: NES [Cartridge], Japan: Nintendo Co. Ltd, 1988.

[18] Sega, Kid Chameleon. Mega Drive [Cartridge], United States: Sega Technical Institute, 1992.

[19] Nintendo, Donkey Kong Country. Super Nintendo Enterteinment System: SNES [Cartridge], England: Rare Ltd., 1994.

[20] Number One, Braid. Games for Windows [Digital], United States: Number One Inc., 2009.

[21] Nintendo, Super Mario Galaxy 2. GameCube [Disc], Japan: Nintendo EAD Tokyo, 2010.

[22] Capcom, Ducktales. Nintendo Enterteinment System: NES [Cartridge], Capcom Co. Ltd., 1989.

[23] Sega, Castle of Illusion. Mega Drive [Cartridge], Japan: Sega AM7, 1990.

[24] Sony, LittleBigPlanet (LBP). PlayStation 3 [Disc], England: Media Molecule, 2008.

[25] Vblank, Retro City Rampage. PlayStation 3 [Digital], United States: Vblank Enterteinment, 2012.

[26] Capcom, Street Fighter Series. Japan: Capcom Co. Ltd., 1987-2012.

[27] Capcom, Resident Evil 4. GameCube [Disc]. Japan: Capcom Co. Ltd., 2005.

[28] B. Kreimeier, Game Design Methods: A 2003 Survey. Gamasutra, Mar. 2003. http://www.gamasutra.com/view/feature/2892/game_design_methods_ a_2003_survey.php.

[29] Nitendo, Super Mario 64. Nintendo 64 [Cartridge], Japan: Nintendo EAD, 1996.

[30] A. Rajaraman, J. Leskovec, J. D. Ullman, Mining of Massive Datasets. 2013, http://infolab.stanford.edu/~ullman/mmds.html.