From “Mods” to “Gaming 2.0”: an overview of tools to ease the game design process

Damien Djaouti, Julian Alvarez, Jean-Pierre Jessel
IRIT, Univeristy of Toulouse III, France
djaouti@irit.fr, julian@ludoscience.com, jessel@irit.fr
Games: Design and Research Conference

This article focuses on tools that allow amateurs to create or modify videogames. In order to contribute to study the nature of game design, we will analyze games as crafted “artifacts”. We will first review five categories of tools that allow creating games, in order to highlight the different “parts” games are made of. We will then use this empirical review to introduce a simple model of the inner structure of games: the ISICO model.

Author Keywords
Game design, tools, mods, game creation toolkits, gaming 2.0, games, model.

INTRODUCTION

Among the directions game designers are currently exploring is the importance of player-generated content[1]. Following the wave of “Web 2.0” internet sites that allow users to create and share their own content (O'Reilly, 2005), several games are designed to let players create or modify in-game content. From the level editors featured in *Little Big Planet* (Media Molecule, 2008) and *Halo 3* (Bungie, 2007) to the object editors from *Spore* (Maxis, 2008) and *Drawn to Life* (5th Cell, 2007), some major console-based games are now incorporating player-generated content. In reference to the “Web 2.0” movement which emphasizes on user-generated content, such games involving player-generated content are labeled as “Gaming 2.0” (Le Roy, 2006). However, such approaches associated with player-generated content are far from being new practices. The computer games market shows a long history of in-game editors and player-created levels, mods and full games. The novelty seems to be the addition of sharing options to these tools.

The main objective of “Gaming 2.0” is to let players, who can generally be considered as “people without professional game designing skills”, create game-related content. In order to achieve this goal, game designers in the industry have invented several tools and methods to ease the game design process, so non-professional designers can create videogames too. **What are these tools and methods, and what can they teach us about game design?**

As an effort to answer this question, this article will first browse through history to analyze several empirical methods and tools aiming to ease the game design process. Through this overview, the objective of this article is to discuss the different “aspects” of a game that are created during game design. This analysis will lead us to propose a theoretical model of the different aspects that must be created in order to “design a game”.

1. THEORETICAL FRAMEWORK

In order to study tools and methods that simplify the game design process, we first need to define “game” and “game design.” Through this article, a “game” will be understood as the resulting artifact from a process called “game design,” as defined by Salen & Zimmerman (2003): “*Game design is the process by which a game designer creates a game, to be encountered by a player, from which meaningful play emerges” [p.80]*.

According to Juul (2005), a “game” can be defined as a variable state machine: *“In a literal sense, a game is a state machine: A game is a machine that can be in different states, it responds differently to the same input at different times, it contains input and output functions and definitions of what state and what input will lead to what following state. […] When you play a game, you are interacting with the state machine that is the game.” [p.60]*.

Thanks to these two definitions game design can be defined as the process that allows designers to create artifacts called “games”. For this article we will focus on a particular kind of games: videogames. We will also limit ourselves to “non-professional” designers.

So, we will now review tools that allow any player to engage in the game design process to create or modify videogames.

2. OVERVIEW OF EMPIRICAL METHODS TO EASE THE GAME DESIGN PROCESS

This section discusses empirical methods and tools used by players to engage in the game design process. They are divided in five categories, from light modification of existing games *(options menu)* to the creation of full games from scratch *(game creation toolkits)*.

2.1. Options Menu

"Options menus" are perhaps the most common tool to modify a videogame. So common, that today few players will regard it as a way to "modify" a game. Almost every videogame now features an "option menu". They allow players to "tweak" the game thanks to a list of predefined choices. Most of the time, these menus allow players to configure keys, buttons or choose between "input-styles". For example, in *Mario Kart Wii* (Nintendo, 2008), each players can choose between *"wiimote+nunchuk"*, *"wiimote+wheel"* or *"gamecube pad"* controllers. But they can also choose between *"manual"* or *"automatic"* styles of driving. In *"manual mode"* the players will gain turbo boosts by performing drifts. This "rule" is replaced by an improved steering ability in *"automatic mode"*.

Similar observations can be made for computer games, like *Lego Indiana Jones* (Traveller's Tale, 2008) where players can configure the keyboard keys used to play. Furthermore, in many games like *Left 4 Dead* (Turtle Rock Studios, 2008) players can select a *"difficulty level"* which affects both the rules and the levels of the game. Indeed, the difficulty is set higher by incrementing the power and the number of enemies. Some games even allow modifying their look'n feel through an option menu. For example, in *Quake 3* (id Software, 1999), players can change their focus of vision and customize the look of their avatars.

The obvious advantage of "options menus" is their ease of use: any players able to pick items in a predefined list can use them. However, this tool severely limits the "creativity" of players. They will not be able to perform modifications which aren't explicitly imagined by game designers. Facing this issue, some designers tried to make their options menu more interesting by adding a huge amount of "choices". A good example is the "editors" built in *Worms 2* (Team 17, 1997), where players can tweak almost any numerical parameter from the rules of the game (e.g., blast and damage radius for each weapon, amount of health, frequency of bonuses…).

From an historical point of view, the first example of this kind of tool seems to be the physical DIP switches[2] built in the arcade games during the early 80's. These switches were used by arcade owners to configure the difficulty (i.e. the profitability) of their games. With the advent of cheap battery-backed RAM components in the 90's, they were replaced by graphical menus, whose access was still restricted to arcade owners. When these games moved from arcade to home, these menus were finally accessible to players. Please also note that most games designed for the VCS 2600 (Atari, 1977) featured different "modes" that players could access using a switch labeled *"Game Select"* on the console. It can too be regarded as a very primitive way to let players "configure" their games, within limits defined by game designers.

To summarize, "options menu" is a common tool to ease the modification of the following aspects of an existing game:

- Input methods: configuration of input devices…
- Rules: modification of numerical values, selection of an abstract "difficulty" parameter…
- Levels: selection of the number of opponents through an abstract "difficulty" parameter…
- Look'n feel: avatar customization, display configuration…

"Options menus" are very easy to use, but their creative potential is limited to the "choices" imagined by game designers.

2.2. Level Editors

Alongside options menus, many videogames are also provided with a "level editor". As the name suggests, it's a tool that can create or modify levels. A current trend is the inclusion of such tools in console blockbusters, as demonstrated by the "Level Editor" from *Little Big Planet*, the "Forge" featured in *Halo 3* and the "Stage Builder" of *Smash Brosh Melee* (Nintendo, 2008). While creative freedom and ease-of-use vary greatly from one game to another, the common point between all level editors is the fact that they are solely designed to create levels. Editors for other aspects of videogames do exist, as discussed in

the next section, but they are less widespread than level editors.

A first distinction can be made between "official" and "unofficial" level editors. Official editors are released by the developers or publishers of the original game, often bundled within the retail game copy. Such editors are usually a toned down version of the professional tool used to design the game, like *"UnrealED"* for the *Unreal* series (Epic Games, 1998-2007) and *"The Elder Scrool Construction Set"* for *Morrowind* (Bethesda Softworks, 2002) and *Oblivion* (Bethesda Softworks, 2006). On the other hand, unofficial level editors may be created and released by skilled amateur developers. Such editors are less widespread than official ones, especially when developers are releasing an official editor. On the console market they are almost inexistent due to the difficulty of injecting newly created levels back in the game without using illegal methods. A notable exception is *New Super Mario Bros Wii* (Nintendo, 2009) who got two unofficial editors, *Reggie* (Trekkie, 2009) and *Tanooki* (Virus & Vash, 2009), less than a month after its retail release.

In association with a level editor, developers may also provide an online content sharing service, so players can exchange the "levels" they created. Such feature is the backbone of the *Trackmania* series (Nadeo, 2003-2009), which includes both a comprehensive "Track Editor" and a "TrackMania Exchange" service to give and get player-created tracks. However, this sharing service is not directly built in the game: players can create tracks inside the game, but must exit it to share the content they created. This service started out as an unofficial exchange site created by amateurs. The success of this site led the game designers to support it as an "official service", though it was never added inside the game. The 140936 player-generated tracks it hosts[3] are nevertheless often visited by players who run multiplayer-servers. If a player-generated track is installed on a multiplayer server, it'll be automatically shared to all players connecting to it. Online playing is here used as some kind of rough in-game sharing system that somehow prefigures what is now used by "Gaming 2.0" (see 2.5.).

In summary, if editors fail to launch an official sharing service for their games, chances are high that amateur online communities of players set up their own. Historically, the first exchange services were created by amateurs. The best known are *Doomworld* (1993-2010) and *Gamers.org-DoomGate* (1994-2010), which were hosting the player-created maps for *Wolfenstein 3D* (id Software, 1992) and *Doom* (id Software, 1993). *Doom* is a turning point in the history of player-generated content, as it's one the first examples of endorsement by the game developers themselves. Indeed, id Software was so positively surprised to discover amateur levels created for *Wolfenstein 3D*, that they decided to encourage and support them in their upcoming title, *Doom*. John Carmack created the "WAD"[4] format to separate the "content" from the "engine", so players can easily create and share content. However, the most-used level editor for Doom was not the official one, which ran only on NeXT platform, but the unofficial *Doom Editing Utility* (Brendon Wyber, 1994) released one month after the game (Kushner, 2003).

To continue with the historical perspective, the first game to be widely recognized for its built-in level editor was *Lode Runner* (Douglas Smith, 1983). Douglas Smith, who designed this game, developed a level editor in order to create 150 levels for the retail version of his game. At first, he didn't plan to include this level editor in the retail version of the game, as he only intended to use it as development tool. However, during the course of development, he asked kids in his neighborhood to design some levels for him. Noticing the way kids enjoyed to use this simple level editor, he decided to include it with the retail game. This feature greatly contributed to the success of this game (Gillet & Gorges, 2008).

While level editors are more common in computer-based videogames, a few console videogames also proposed this feature, such as the track editor from *Excite Bike* (Nintendo, 1984)[5]. Moreover, the earliest known example of commercial videogame to feature a level editor seems to be *K.C. Munchkin* (Ed Averett, 1981), a brilliant Pac-man clone packed with a maze-editor and released on Odyssey².

To summarize, a "level editor" is a common tool to ease the creation and modification of the following aspects of an existing game:

- Levels: placing of predefined "game objects" in a virtual space to create a "level" for players to explore.

Unlike "options menus", level editors allow for the creation of "emergent" content: player can create levels that were not anticipated by the designers of the game.

2.3. Modding

A "mod" is a set of redistributable modifications for a given videogame (Bogacs, 2008). These modifications can be applied to any aspect of a videogame. Popular examples of "mods" include *Counter-Strike* (Minh Le & Jess Clife, 1999), a modification of *Half-Life* (Valve Software, 1998) and *Defense Of The Ancients* (Eul, 2003), a modification of *Warcraft III* (Blizzard Entertainment, 2002).

These "mods" are created through the use of a collection of editors. Level editors were discussed in the previous section. Similar editors exist for the other aspects, although no real "name" applies to them. When these editors are official, they are usually gathered in a toolset called "Software Development Toolkit" (SDK). For example, the *"Source SDK"* can be used with any game running on Valve's Source engine, from *Half-Life 2* (Valve Software, 2004) to *Left 4 Dead 2* (Valve Software, 2009). This SDK is composed of about twenty tools including: *Hammer Editor*, a level editor, *Face Poser* and *Vtex*, which can modify the appearance of game objects, and a series of scripts that allow players to modify rules. As for the level editors, official SDKs are usually toned-down version of the professional development tools used by the studio. Please note that inputs methods are directly modifiable with an option menu embedded in the retail version of each game based on this Source engine (this is very common, as discussed in 2.1).

The "mods" created with these tools are not autonomous: players need to own the original game to enjoy them. Hence, like "levels", "mods" are now regarded as a key feature by game publishers who decide to provide an official online sharing service alongside the official SDK. An example of this trend is *Dawn of War II* (Relic Entertainment, 2009) and its sharing space on the official community website[6]. Last but not least, in the absence of satisfying officials SDKs, skilled amateurs may create and release unofficial toolkits. For example, several unofficial modding tools are available for the *Sims* series (2000-2009), in addition of the (limited) official tools and exchange platform. Unofficial exchange platforms exist too, one of the most popular being *Mod DB* (2002-2010) which hosts 6074 mods for a wide range of games[7].

Regarding "mods" themselves, an empirical distinction is usually made by players between "Total Conversion" ones (like *Counter-Strike*) and "Partial Conversion" ones (like *Defense Of The Ancients*). While a formal definition of these two kinds of "mods" is still lacking, "Partial Conversions" can be regarded as "mods" confined to a single aspect of a game while "Total Conversions" modify several aspects simultaneously. Please note that "levels" alone are not part of the modding category, though most "mods" also includes new levels. Indeed, due to the widespread existence of player-generated "levels", they are not usually regarded as real "mods" but simply as "levels", "maps", "tracks"...

While "modding" is clearly an approach to ease game design, compared to "options menu" and "level editors" it feels like a more powerful but more complex way to achieve it. The freedom of creativity is nearly endless with modding tools, as players can change any aspect of a videogame. Far from being limited by official tools and exchange platforms, some players will create their own modding tools if needed. The counterpart to this freedom is the expertise needed to enjoy it. Indeed, mastering these tools often requires a professional level of skills. Hence, some "mods" are created by team of players organized like small development studios (with coders, artists, level designers, project leaders…). The most talented modders are even offered jobs in the videogame industry. For example, *Tim Willits* joined id Software as a game designer thanks to the successful mods and levels for *Doom* he created in his spare time (Kushner, 2003). Overall, the main challenge faced by designers who try to make good game-content modification tools, is to find a balance between the level of skills it requires and the creative freedom it offers.

From an historical point of view, *Doom* was a major step forward regarding modding practices. This game got the first mod to coin the "Total Conversion" label, *Aliens TC* (Justin Fisher, 1994). However, as for its level editors, the tools used to edit *Doom* were unofficial, like the popular *DeHacked* (Greg Lewis, 1994) able to alter the rules of the game. A few years later, *Counter-Strike* seem to be the first "mod" to have reached an higher popularity than its original game, bringing attention from general public to what was then a quite "underground" community of creative players.

Regarding the "modding" spirit, we think it can be traced back to *Spacewar!* (Steve Russell & al., 1962). Steve Russell initial version of this pioneer game was modified by Peter Samson, who changed the background planetarium to reflect the real sky. Dan Edwards then added in a sun to modify the whole gameplay experience. These changes were emphasized when Martin Graetz included the

"hyperspace-jump" feature. Such modifications were possible due to the "open-source" nature of *Spacewar!*: anyone could access the source code of the game and modify it (Graetz, 1981). It was also crafted in a closed space (the TMRC student club at MIT), which means that modders could benefit from the help of the original designers of the game.

The first mod created by players without any help from the original developers seems to be *Dino Smurfs* (Andrew Johnson & Preston Nevins, 1983), a modification of *Dino Eggs* (David Schroeder, 1983). It was quickly followed *Castle Smurfenstein* (Andrew Johnson & Preston Nevins, 1983), a modification of *Castle Wolfenstein* (Silas Warner, 1981). Both mods were created by two teenagers who replaced the graphics and sounds of the original games by characters coming from the "Smurfs" comics. What started out as a joke required these two creative players to study how the art and sound assets where stored in the original games without any help (nor consent) from the game designers. *Dino Eggs*'s author didn't even get the chance to play the modded version of his game before 1998 (Johnson, 1999).

An earlier similar example is the arcade game *Crazy Otto* (General Computer Corporation, 1981), a bootlegged version of *Pac-Man* (Namco, 1980) that added legs to the avatar and tweaked the rules of the game. The technical skills of the GCC team allowed them to create "enhancement boards" that modded arcade games. Their first attempt was *Super Missile Attack (1981)*, whose existence led them to being sued by Atari, creator of the source game *Missile Command* (1980). As GCC won the trial, they were legally authorized to create and sell their mod-kits. When they showed the *Crazy Otto* mod to Midway, the American distributor of *Pac-Man* who was waiting for Namco to release a sequel, they surprisingly did not choose to sue GCC (Kent, 2001). Instead, they hired them to keep on modifying *Pac-man* in order to create what is known today as *Ms. Pac-Man* (Midway, 1981).

To summarize, "modding" is a way to allow players to modify the following aspects of an existing game:

- Input methods: configuration of input devices…
- Rules: modification of the rules through the use of a programming language…
- Levels: placing of predefined or newly created "game objects" in a virtual space to create a "level" for players to explore.
- Look'n feel: importation of new graphical and audio resource in the game…

Modding can be made thanks to a series of "editors" tools that allow players to create "emergent" content.

2.4. Game creation toolkits

The next step, after toolsets to create "mods" of existing games, is to use similar tools to create full games from scratch. Such "game creation toolkits" like *Game Maker* (Mark Overmars, 1999-2008) or *The Games Factory 2* (Clickteam, 2006) are the backbone of amateur game designers communities.

While close in appearance, game creation toolkits and modding tools are meant to serve different purposes. Modding tools are released for free (or created by amateurs) as a bonus to extend player's experience with retail games. Therefore, such tools, even when they are powerful enough to let players change everything in the game, cannot produce autonomous videogames. This is a major difference with "game creation toolkits" that feature similar tools but allow creating autonomous games. Some of them are freeware, but most of the popular "game creation toolkits" are commercial products. Here, tools are the product, and no longer a mere bonus for a retail game.

The most common strategies used by "game creation toolkits" to ease game design seem to be:

- **A technical limitation of the created games.** The most obvious examples is the "2D or 3D?" nature of game toolkits. 2D games are easier to create and thus require fewer skills. Hence, several toolkits focus on the creation of 2D Games. This is the case of *Game Maker* (although later versions can also create 3D games) and *Multimedia Fusion 2* (Clickteam, 2006). On the other hand, *3D Game Studio* (Conitec Datasystems, 1993-2010) is solely suited for the creation of 3D games.
- **A restriction of the creative freedom to certain "aspecs".** For example, *The 3D Game Maker* (The Game Creators, 2002) offers a level editor, but is restricted to an "option menu" for the look'n feel part. It doesn't let players get their hands on the rules and input methods: these aspects are fully pre-built for any game created with this toolkit.

- **An optional automation of some tasks.** Arguably one of the best efforts to democratize the creation of videogames, the series of tools designed by Clickteam (*Klik & Play*, *The Games Factory* and *Multimedia Fusion*) introduced two easy methods to create "rules":
    - The "step-by-step" tool starts by asking the player to apply some basic rule templates on elements (such as *race car*, *platform* or *top-down* behaviors). He can then test the game, and whenever a "probable event likely to become a rule" occurs (such as *collision*, *key pressed*, etc...), the game stops and asks the user if he do want to create a rule.
    - The second innovative method is the replacement of traditional programming languages by a "point & click" approach. The user can manipulate "actions" and "conditions" to create "rules". They are organized in a giant table, easier to read than a script for a beginner game designer. This feature is so interesting that several others toolkits like *Game Maker*, *Construct* (Scirra, 2008-2010) and *Game Develop* (CompilGames, 2009) include similar ones.

These general strategies are then used by two different kinds of "game creation toolkits":

- **General-purpose toolkits**, suited for the creation of any kind of videogames.
- **Genre-specific toolkit**s, designed to produce videogames from a particular genre, such as *First-Person Shooters*, *Role-Playing Game* or *Fighting games*. Here, the creative freedom is restricted to greatly improve the ease-of-use. Indeed, these tools feature a lot of "pre-built" elements in order to accelerate the design process for a particular game genre. For example, a new game project created with *RPG Maker* (Enterbrain, 1992-2007) comes with pre-built battle, exploration and character development systems. Such systems can be tweaked but are not intended to be removed or replaced, as they are core components of most RPG games. The same goes for Fighting games with *M.U.G.E.N*. (Elecbyte, 1999-2009) and for First-Person Shooter with *FPS Creator* (The Game Creators, 2005-2009).

From an historical perspective, the earliest general-purpose games creation toolkit that showed the way to its numerous successors was *GameMaker* (Garry Kitchen, 1985). This one is not to be confused with *Game-Maker* (Recreational Software Design, 1991) and the aforementioned *Game Maker*. In consideration of its release date, Garry Kitchen's toolkit for game creation was a very impressive application, with embedded editors for music, sound, graphic and levels. The code of the game was created through the use of a simple programming language. In 1994, Clickteam's *Klik & Play* introduced the concept of designing "rules" without the use of a programming language. As for mods and levels, unofficial sharing platforms accompanied the release of these game creation toolkits. For example, *The Daily Click* (2002-2010) hosts 3937 games[8] created with *The Games Factory* and *Multimedia Fusion*, while *Game Maker Games* (2004-2010) gathers 3576 games[9] created with *Game Maker*. Editors of such applications tend to follow the path opened by their users. For example, *Game Maker*'s official site now features a "share" section gathering 312 games[10]. However, this sharing service is still external to the game creation application. It's the main difference between "game creation toolkits" and "Gaming 2.0", as discussed in the next section.

Regarding genre-specific toolkits, the earliest example appears to be *Pinball Construction Set* (Bill Budge, 1983). It allows players to create a great variety of pinball games. This tool is centered on an easy-to-use level editor which let players create their own tables by drag-n-dropping pre-built elements. They can also freely draw the background. The resulting pinball tables can be exported to autonomous games and distributed like any other videogames of the era. Similar tools for others game genres quickly followed, such as *Adventure Construction Set* (Stuart Smith, 1985), *Racing Destruction Set* (Rick Koenig, 1985) and *Shoot'Em-Up Construction Kit* (Sensible Software, 1987).

On a more general note, we argue that "game creation toolkits" and "code libraries"[11] share the same philosophy: to speed up and ease the creation process of a computer application. While they target a programmer audience instead of a player one, code libraries are a way to ease programming. Indeed, if each computer program had to be written from scratch, few coders would be able to create them. Game toolkits are bringing this concept one step further: they introduce tools that are automatically

performing the most technical tasks in order to give players the ability to create games without professional-level skills. As for the others category of tools discussed in this article, the balance between creative freedom and ease-of-use remains central for designers of such toolkits.

To summarize, "game creation toolkits" are a way for players to create new games from scratch. They can do so by designing the following aspects of games:

- Input methods: configuration of input devices…
- Rules: creation of rules with either "point & click" methods or programming languages…
- Levels: placing of pre-built or newly created "game objects" in a virtual space to create a "level" for players to explore.
- Look'n feel: creation of new graphical and audio resource for the game…

Like modding, game creation toolkits rely on a series of "editors" tools that allow players to create "emergent" content, from scratch.

2.5. Gaming 2.0

The "Gaming 2.0" category is different from the other ones. As discussed in (Djaouti, Alvarez, & Jessel, 2010), "Gaming 2.0" doesn't introduce new ways to create games, but add sharing abilities to existing tools. More precisely, all of the four approaches we discussed previously can be found in "Gaming 2.0":

For example, *The Sims Carnival Wizard* (Electronic Arts, 2008) allows players to create full games using only predefined lists; it presents a series of choices to players in order to create a game from scratch. Players first select a game genre (e.g., *Racing*), then a sub-genre (e.g., *Top-Down Racing*) and a visual theme. Another series of questions will allow players to "fine-tune" the game (e.g., pick a goal between *win the race* or *last man standing*; set the *number of laps* and the value of *physics variables*...) [12]. Players can even modify the look of each car, and select a *race track* (i.e., a "level") from a predefined list. Overall, this method is very easy to use, but its creative potential is limited to the "choices" imagined by designers of the application. Indeed, it can even automatically generate games, which are created by letting the computer pick random choices from each list.

As discussed when reviewing level editors, many "Gaming 2.0" titles like *Little Big Planet*, *Halo 3*, and *Super Smash Bros. Melee* relies on such tools.

Another example shows how mods can be found in Gaming 2.0. *The Sims Carnival Swapper* (Electronic Arts, 2008) allows players to create "Partial Conversions" by modifying the look and feel of existing games. Players can pick any game available on the site and create a "mod" for it by replacing the art assets. *Ugengames* (MobiTween, 2007) offers a similar concept but doesn't offer any additional tools to work on the other aspects. While *The Sims Carnival Swapper* and *Ugengames* rely on external images resources, some games embed a full-featured art editor. For example, *Drawn to Life* allows players to literally "draw" the visual appearances of game elements while playing. A carbon copy of *Drawn to Life*'s editor can be found in *WhoseGame* (Orange, 2007). Its 2D drawing program allows players to draw within predefined zones, which will then be animated as parts of characters (e.g., arms, head, body).

Spore also features an art editor, in which players can create 3D models for any object in the game (e.g., creatures, buildings, vehicles). The *Spore* editor is a very interesting example of how to democratize game content creation without limiting it too much. Professional 3D creation software tools are usually very complex to use, and were obviously not designed to appeal for a large player audience. The actual editor built in this game works like some kind of puzzle: players can create quite complex 3D models by assembling pre-built forms and tuning their size, orientation, and color. As an indicator of its efficiency, *Spore*'s sharing platform hosts nearly 130 million objects designed by players.

Besides editors able to modify existing games, "Gaming 2.0" also gathers tools that allow players to create new games from scratch. The legacy of "game creation toolkits" in "Gaming 2.0" can be seen in the numerous websites where players can create and share games. Such examples are *Sims Carnival*, *PlayCrafter* (ZipZap Play, 2008), *Cartoon Network Game Creator* (Cartoon Network, 2008-2009), *Sploder* (Geoff Gaudreault, 2007) and *WhoseGame*. Far from being limited to entertainment, *Gamestar Mechanic* (GameLab, 2009) brings "Gaming 2.0" to the field of Serious Games (Alvarez & Djaouti, 2010). Indeed, this application is used by Institute of Play as a tool to teach digital media literacy (Salen, 2009). Last but not least, consoles also feature some "game creation toolkits meet Gaming 2.0" examples, such as *Wario Ware: Do It Yourself* (Nintendo, 2009) for Nintendo DS and *Kodu Game Lab* (Microsoft, 2009) for Xbox360.

While we could go further in the exploration of "Gaming 2.0", it would lead us beyond the scope of our current study. Indeed, as "Gaming 2.0" doesn't introduce new methods or tools to craft games, it won't allow us to identify new "parts" of games as artifacts. Nevertheless, it can be argued that the "sharing" spirit lying at the core of "Gaming 2.0" will encourage players to handle game design as a collaborative process. So, instead amateur game designers working on their own, maybe we will soon see games crafted by several unrelated persons who shared their creation on a "Gaming 2.0" platform?

So, at this point of our study, "Gaming 2.0" seems more relevant to study the game design process than the game artifacts themselves.

3. THE ISICO MODEL: UNDERSTANDING THE INNER STRUCTURE OF VIDEOGAMES ARTIFACTS

During the overview of tools used by players to design videogames, we noticed that these tools allow modifying or creating four distinct aspects of a videogame:

- **Input methods:** to design the ways players will use devices to send "information" to the game.
- **Rules:** to design how the game will interact with players.
- **Look'n feel:** to design how the game will display its content to players.
- **Levels:** we also observed that many tools allow creating "levels", "maps", "tracks"… i.e. to design a virtual space that players can explore during the game.

Besides our empirical observations, several theoretical models detail the different "parts" games are made of. Järvinen (2008) splits a game in nine "elements" belonging to three categories: "systemic elements", "compound elements" and "behavioral elements". The aspects we observed in our analysis seem to be close to some items of the "compound" and "systemic" categories, but Jarvinen's theory of game elements is more detailed than our observations. At first, the "ruleset", "game mechanics" and "information" categories feels like simple subcategories of what we observed as "rules". However, a closer study of this theory shows that it tries to understand the "meaning" of the different game parts, while our empirical observation analyses games through a more "technical" point-of-view.

A similar analysis can be made with the six "layers" outlined by Tajè (2007) and with the sixteen "dimensions" introduced by Elverdam & Aarseth (2007). These models are suited to provide an in-depth analysis of the structure of the game, and detail how each element relates to each other in order to create "meaningful play" (Salen & Zimmerman, 2003).

A different approach is the "token" model detailed by Adams & Morris (2003), but this one only focuses on understanding how to design the "rules" of games. Most of the "patterns" introduced by Björk & Holopainen (2004) serves a similar purpose: to help professional designing how a game will interact with players. The most "formal" model dedicated to "rules" seems to be the taxonomy of rules introduced by Frasca (2003), who defines three distinct categories: "manipulation rules", "goal rules", and "meta rules".

While all these detailed theoretical models are useful for deep analysis of games and their meaning, our empirical observation shows that "games", as artifacts, could be analyzed through a simpler model. Such a model would solely focuses on the "physical" parts of game artifacts. To define this model, we can begin with the definition of "interactivity" coined by Chris Crawford (2003): *"A cyclic process in which two active agents alternately (and metaphorically) listen, think, and speak."*

If we consider that an artifact called videogame is one of these *"active agents"*, we can do the following associations:

- *"Listen":* in order to listen to players, videogames will rely on their input devices, obeying to the way its designer configured them.
- *"Think":* the videogame will use the rules crafted by its designer to react to player's inputs. As a game is not a living being, we would rather say that it will use the rules to "compute" the data coming from its input devices.
- *"Speak":* The results of the computing phase will then be displayed to the players through a collection of output devices. By creating art assets, sounds, or force-feedback patterns, a designer can decide how the game will use its output devices.

So, we can see a connection between this definition and three of the "parts" we observed: "Input methods" with "Listen", "Rules" with "Think", and "Look'n

feel" with "Speak". What about "levels"? As discussed in the beginning of this article, Juul (2005) defines a game as a "state machine". It means that a game can be understood as a collection of "states", and that "interaction" is due to the system shifting from one state to another. According to this definition, "levels", "maps" and "tracks" are simply the initial state of the game. Indeed, when a designer uses a level editor, he sets the initial configuration of all the game objects. However, the words "levels", "maps" and "tracks" are usually tied to specific game genre. "Tracks" evokes racing, while "maps" seems linked to strategy games. Therefore, we propose to use the "Initial State" term to refer to what designers can create when using "level editor" or similar tools.

To summarize, we now have identified four "parts" that compose a game, as long as "game" means "a variable state system crafted through a game design process":

- **Initial State**: the starting state of the system.
- **Input:** the means that allow players to provide information to the system.
- **Compute**: the inner mechanics that allow the system to change states.
- **Output**: the way the system displays its current state to players.

These four "parts" can be created by separate persons, but in the end they will always be packed together to create a single artifact called "game." For convenience, we propose to refer to this theoretical framework as the **ISICO model** *(short for: **I**nitial **S**tate, **I**nput, **C**ompute, **O**uput)*.

Unlike previously mentioned frameworks, the ISICO model solely focuses on the different parts that need to be designed to create a "game" artifact. It doesn't allow understanding the meaning of videogames, but can be used to analyze tools that allow creating or modifying videogames. For example, the ISICO model is used in (Djaouti et al., 2010) as an analysis grid to compare how different "Gaming 2.0" examples let players create videogames.

CONCLUSION

Through an overview of methods and tools used by amateurs to create games, this article intends to stress out the difference between "game" and "game design".

A "game" is an artifact, created by one or several designers who craft its inner parts thanks to a variety of tools. The ISICO model, introduced in this article, identifies four "parts" that compose a "game": Initial State, Input, Compute and Output.

To create such games, designers must engage in a process called "game design". Many questions still need to be answered about its nature. A common definition of "process" is *"A series of events to produce a result."* [13] According to this definition, games are the "result" of game design. But what is the "series of events" that composes the game design process itself?

The scope of this question is very large. In order to try to find some answers, the next step of our research will focus on studies directly related to game design. More specifically, we intend to analyze a large corpus of "Game Design Manuals" in search of clues to the nature of game design. For example, Tracy Fullerton (2008) details the several stages of this process. But in another example, Jesse Schell (2008) proposes a different set of stages for this same process. Hence, we propose the following hypothesis: Game design is a not a single universal process, but a set of distinct processes defined by different stages. However, they all share a common goal: to create a game artifact, whose inner structure doesn't vary with the game design process used. In other words, we argue that, while many definitions of the game design process exist, all the games they can create will always be defined by four parts: Initial State, Input, Compute and Output. Our future works will try to confirm this hypothesis.

REFERENCES

1. Alvarez, J., & Djaouti, D. (2010). *Introduction au serious game*. Questions Théoriques.

2. Bjork, S., & Holopainen, J. (2004). *Patterns in Game Design* (1er ed.). Charles River Media.

3. Bogacs, H. (2008). *Game Mods - a survey of modifications, appropriation and videogame art* (Bachelor Thesis). Austria: Vienna University of Technology.

4. Crawford, C. (2003). *Chris Crawford on Game Design*. New Riders Games.

5. Djaouti, D., Alvarez, J., & Jessel, J. (2010). Can "Gaming 2.0" Help Design "Serious Games"? A Comparative Study. Presented at the SIGGRAPH 2010, Los Angeles.

6. Elverdam, C., & Aarseth, E. (2007). Game Classification and Game Design: Construction

Through Critical Analysis. *Games and Culture*, *2*(1), 3-22. doi:10.1177/1555412006286892

7. Frasca, G. (2003). Simulation versus Narrative: Introduction to Ludology. In M. J. P. Wolf & B. Perron (Eds.), *The Video Game Theory Reader* (1er ed.). Routledge.

8. Fullerton, T. (2008). *Game Design Workshop, Second Edition: A Playcentric Approach to Creating Innovative Games* (2 ed.). Morgan Kaufmann.

9. Gillet, S., & Gorges, F. (2008). La naissance de Lode Runner. In *Pix'N Love #4*. Editions Pix'N Love.

10.Graetz, M. (1981). The Origin of Spacewar! *Creative Computing*, 56-67.

11.Järvinen, A. (2008, March 8). *Games without Frontiers: Theories and Methods for Game Studies and Design* (PhD Thesis). Finland: University of Tampere. Retrieved from http://acta.uta.fi/english/teos.php?id=11046

12.Johnson, A. (1999). The first 'Official' Castle Smurfenstein Home Page. Retrieved May 21, 2010, from http://www.evl.uic.edu/aej/smurf.html

13.Juul, J. (2005). *Half-real : video games between real rules and fictional worlds*. Cambridge Mass.: MIT Press.

14.Kent, S. L. (2001). *The Ultimate History of Video Games: From Pong to Pokemon--The Story Behind the Craze That Touched Our Lives and Changed the World* (1er ed.). Three Rivers Press.

15.Kushner, D. (2003). *Masters of Doom: How Two Guys Created an Empire and Transformed Pop Culture* (First Edition.). Random House.

16.Le Roy, B. (2006, April 1). Gaming 2.0. Retrieved May 21, 2010, from http://weblogs.asp.net/bleroy/archive/2006/04/01/Gaming-2.0.aspx

17.O'Reilly, T. (2005). What is Web 2.0 - Design Patterns and Business Models for the Next Generation of Software. Presented at the Web 2.0 Conference, San Fransisco. Retrieved from http://oreilly.com/web2/archive/what-is-web-20.html

18.Rollings, A., & Morris, D. (2003). *Game Architecture and Design: A New Edition*. New Riders Games.

19.Salen, K. (2009). *Gamestar Mechanic Learning Guide*. Institute of Play. Retrieved from http://www.gamestarmechanic.com/

20.Salen, K., & Zimmerman, E. (2003). *Rules of play*. MIT Press.

21.Schell, J. (2008). *The Art of Game Design: A book of lenses*. Morgan Kaufmann.

22.Tajè, P. (2007, July 27). Gameplay Desconstruction: Elements and Layers. Retrieved from http://www.gamecareerguide.com/features/355/gameplay_deconstruction_elements_.php

NOTES

[1] http://en.wikipedia.org/wiki/User-generated_content#Player_generated_content
[2] A Dual In-line Package switch is a set of tiny switches gathered in a single component for circuit boards
[3] Retrieved 06-12-09 from http://www.tm-exchange.com/
[4] Which stands for « Where's All the Data? »
[5] This feature was actually included to promote the « famicom disk system », a new device that allowed the famicom to store data of floppy disks, thus providing a way for player to share generated content.
[6] Retrieved 01-12-09 from http://community.dawnofwar2.com/forums/world-builder-and-modding
[7] Retrieved 06-12-09 from http://www.moddb.com/mods
[8] Retrieved 12-06-09 from http://www.create-games.com/
[9] Retrieved 12-06-09 from http://www.gamemakergames.com/
[10] Retrieved 06-12-09 from http://www.yoyogames.com/
[11] A code library is a collection of reusable programming code commonly used by professional programmers
[12] Observations made on 12-01-09 from the tool available at : http://www.simscarnival.com/wizardtool
[13] Retrieved on May 20, 2010, from http://en.wiktionary.org/wiki/process