Playing architecture:
The usability of a game engine as a real-time design tool
Siemon Boes

Supervisor: Prof. dr. Ronald De Meyer
Counsellors: Dr. ir.-arch. Pieter Pauwels, Dr. ir.-arch. Ruben Verstraeten, Ir.-arch. Tiemen Strobbe

Master's dissertation submitted in order to obtain the academic degree of
Master of Science in Engineering: Architecture

Department of Architecture and Urban Planning
Chairman: Prof. dr. ir. Arnold Janssens
Faculty of Engineering and Architecture
Academic year 2014-2015
Playing architecture:
The usability of a game engine as a real-time design tool
Siemon Boes

Supervisor:     Prof. dr. Ronald De Meyer
Counsellors:    Dr. ir.-arch. Pieter Pauwels, Dr. ir.-arch. Ruben Verstraeten, 
                Ir.-arch. Tiemen Strobbe

Master’s dissertation submitted in order to obtain the academic degree of
Master of Science in Engineering: Architecture

Department of Architecture and Urban Planning
Chairman: Prof. dr. ir. Arnold Janssens
Faculty of Engineering and Architecture
Academic year 2014-2015
ACKNOWLEDGEMENTS

I would like to express my gratitude to a number of people that have made it possible to realize this master thesis and explore a topic that is of great interest to me. To my supervisor and counsellors, Ronald De Meyer, Pieter Pauwels, Ruben Verstraeten, and Tiemen Strobbe, I am grateful for their assistance and insightful suggestions throughout the entire research and writing process. I would like to thank Pieter Pauwels in particular for his patience and words of inspiration, and for helping me to maintain a clear focus in my writing. Finally, I wish to acknowledge the support of my parents, whose encouragement and enthusiasm for my work pushed me further than I thought I could go.

The author gives permission to make this master dissertation available for consultation and to copy parts of this master dissertation for personal use. In the case of any other use, the copyright terms have to be respected, in particular with regard to the obligation to state expressly the source when quoting results from this master dissertation.

Siemon Boes  May 19, 2015
Abstract—The development of increasingly powerful and user-friendly game engines, combined with unprecedentedly low licensing fees, have made video game technology more accessible than ever for applications outside the game industry. While adaption in the professional field of architecture has been relatively rare thus far and mostly limited to mere visualization, we argue that modern game engines show potential for design-related purposes as well. This study examines the role game engines can play in the design process and, more specifically, takes a closer look at the CryEngine game engine as a tool for creating a regulated, real-time design environment.

Keywords—architectural design; game engine; medium; rules

I. THE SIMILARITY OF ARCHITECTURE AND GAMES

The relevance of the game engine for the discipline of architecture is related to the intrinsic similarities between games and architecture. Definitions of ‘game’ and ‘play’ by influential thinkers such as Johan Huizinga or Roger Caillois are of limited use because of their generality and the fact that they predate the advent of video games. Video games, as a specific kind of games, are particularly interesting for architectural designers because they rely heavily on spatiality and are typically more complex than other, traditional games. In addition, architecture possesses a number of game elements, such as objectives, rules, and narrative.

Spatiality is what defines almost every video game, whether its world is two- or three-dimensional. Gameplay is created by allowing the player to interact with certain elements in the game world and explore its virtual spaces. This holds true for a wide variety of video game genres (first-person shooter games, strategy games, simulation games, racing games…) and is also of fundamental importance for real-world architecture and the architectural design process. Architects create spaces for the user/player to experience and interact with. Video games are different from other games because, by definition, they are based on electronic systems; computers. While the level of complexity in traditional games is mainly limited to the human player’s capability of retaining and processing information – sometimes aided by physical objects such as cards and tokens – computer hardware has raised the bar considerably. Enormous amounts of information can be stored in the memory of the computer, while extensive simulations can run in the background, allowing the player to concentrate on the more important gameplay decisions. ‘Complex’ is also what describes architecture. It is the act of designing and realizing a building, while simultaneously keeping an eye on spatial qualities, urban context, aesthetics, material characteristics, stability, thermal performance, HVAC, acoustics, budget, and many more. As with video games, there is a focus on trying to free the player/designer from this burden of complexity by using computer technology (e.g. building performance analysis software). Ideally, computers would allow the architect to concentrate solely on what really matters and cannot be automated, or would not be beneficial to automate.

Other elements that (video) games share with architecture are objectives, rules and narrative. In the majority of video games, the player is presented with one or more goals, or objectives, to give direction and motivate his actions. The architectural designer also pursues objectives, but these are typically less clearly defined and often completely unknown. Delivering ‘good architecture’ is an obvious but ultimately vague objective. Rules constitute a second important game element. They mark the playing field by fixing parameters, limiting the amount of possibilities, and therefore imposing constraints. In both video games and architecture, a distinction can be made between natural rules and conventional rules. Natural rules are fundamental and do not rely on other rules, while conventional rules are built on top of these natural rules and are not nearly as fixed. An example of a natural rule present in the real world (and many game worlds) is gravity. The amount of possibilities in architecture is limited by gravity in the first place, but not every building that conforms to gravity will conform to the rules of architecture, which are purely conventional. Architectural rules are the means of which critics can dispose to value a design, taking into account how the architect handled the imposed restrictions or even turned them into his advantage. A third game element is the narrative. It is the story that links all pieces of the game or building coherently together and gives meaning, weight, and context to them, so that the whole becomes more than the sum of its parts. In both video games and architecture, narrative is often not explicited, but embedded in spaces or objects.

II. PLAYING THE ARCHITECTURAL DESIGN GAME

In order to understand the role a game engine can play in the architectural design process, it is crucial to first understand the unique circumstances under which the architect operates: the ‘wickedness’ of the design problem and the representational nature of design media.

One of the main reasons why the architectural design process is thought of as being difficult, is related to the mostly unknown nature of its objectives, as stated above. Designing ‘good’ architecture boils down to solving a wicked problem, a term formally described by Horst Rittel and Melvin Webber [1]. A wicked problem is a problem that is impossible to truly solve because the elements that would contribute to a solution are often interdependent and notoriously hard to identify. Rationalist methods of problem solving based on system analysis are of limited use with wicked problems due to the sheer scale of their systems and the partly unknown elements of which they are constituted. Although Rittel and Webber originally used the term to describe societal and planning problems,
it applies to architecture as well. An important implication of the wickedness of the architectural design problem is that it is impossible to know if the problem was sufficiently well understood and if the resulting solution was in fact the most optimal one (which is highly improbable). There is no formula for ‘good’ architecture.

A second important difficulty that the architect has to overcome is related to the design medium. Contrary to many other design professions, the architect is rarely able to directly work with his object. Instead he makes use of representations: sketches, drawings, physical scale models... Therefore, if the architect can be considered a player, he does not play just a game, he plays the metagame, which symbolizes the actual game. By using a design medium, he reformulates a complex design problem in the terms of this medium, simultaneously leaving its marks on the resulting design. It is of crucial importance to recognize the representational power of the medium as a tool that is not neutral, nor external to the design process.

But how does a designer work on a problem that is impossible to fully contain and does not allow for an unambiguous definition, while disposing only of tools that are inevitably flawed in their translation and communication of ideas? An answer to this question is provided by the concept of ‘reflective practice’, elaborately described by Donald Schön [2]. For the architect as reflective practitioner, the biased nature of the design medium is key to overcoming it, instead of just an undesirable side-effect. By using a simplified, abstract representation of reality, the architect imposes order on top of the design problem, effectively limiting the amount of possible outcomes. The resulting friction provides the architect with feedback, to which the architect can then respond by adjusting certain elements or entirely reframing the problem: this is reflection-in-action. Therefore, choosing the right medium is crucial for the architect, since it defines the nature of the feedback he will receive.

A critique on the metaphor of the architect playing the game of architecture, is provided by Chris Totten [3]. Totten suggests that the architect is not a player, but a designer of games, which others (the building’s users) play. He advocates a new way of designing that borrows from the video game industry to counter the sometimes dominant top-down approach in architectural design. The game engine as a medium could be transferred to the architectural design process to fulfill this purpose. Other design media that are explicitly aimed at real-time interaction (e.g. Navisworks) do not offer the same level of interactivity and visual quality.

III. CREATING A REAL-TIME DESIGN ENVIRONMENT IN A GAME ENGINE

The game engine market has been evolving at a very high rate in the last few years, which has been attributed mainly to the still recent transition to the eighth generation of video game consoles. This has led developers to make thorough updates to their existing game engines, as well as switch to more aggressive pricing strategies. As a result, the accessibility of game engine technology has increased significantly.

As a general trend it can be observed that game engines, besides getting more powerful and feature-rich over time, also grow in user-friendliness and cater to an increasingly broad array of purposes. While the earliest game engines were only suited for a few game genres, modern game engines can be used for nearly every genre, as well as for applications other than typical video games (e.g. serious games). Unreal Engine, Unity and CryEngine can be considered the most capable engines currently available that are both freely licensable and widely supported. For our case study, we use CryEngine.

An interesting feature of a modern game engine environment, besides the impressive capabilities of the real-time rendering engine, is the default integration of natural rules such as gravity, and real-time lighting. However, realism in game engines is still largely an illusion. Video game developers are often more concerned about appearances than about accuracy, which is reflected in the way game engines approach realism. They offer a kind of semi-realism that is inadequate to accurately mimic real-world circumstances. While these integrated rules are still lacking in realism, a solution is provided by the flexibility of the game engine. Contrary to most traditional design media, additional rules can be manually embedded in the game engine, making it possible to adjust the rules of the medium to the designer’s needs. In this investigation, we succeed in adding rules to our CryEngine environment by using its natively included Flowgraph visual scripting utility, which removes the need for knowledge of a programming language.

![Figure 1 – Design environment based on CryEngine](image)

IV. CONCLUSION

We situated the game engine’s potential use for architectural design applications by highlighting the similarities between video games and architecture, and the importance of the medium in the design process. The rules/constraints embedded in the medium have a defining impact on how the design progresses to its final state. When used as a design medium, the most important qualities of a game engine environment are found in its potential for a user-out approach and the flexibility of its rules. The latter represents an important advantage when making the comparison with traditional design media. Game engines can provide a foundation for a wide variety of design games and purposes, therefore representing a multiplicity of media, rather than just one medium. A recent increase in the accessibility of game engines opens up opportunities for adoption in the professional field.

REFERENCES

# TABLE OF CONTENTS

1 INTRODUCTION........................................................................................................1

2 ARCHITECTURE AND GAMES..............................................................................3
   2.1 Definition(s) of game .........................................................................................3
   2.2 Architecture as a video game ............................................................................5
      2.2.1 Spatiality and complexity ...........................................................................5
      2.2.2 Objectives, rules and narrative ..................................................................7
   2.3 A multiplicity of games? ....................................................................................12

3 PLAYING THE DESIGN GAME ...........................................................................14
   3.1 Introduction ......................................................................................................14
   3.2 The architectural design process ......................................................................15
      3.2.1 Solving a wicked problem .........................................................................15
      3.2.2 Playing a metagame ................................................................................18
      3.2.3 A reflective practice ..................................................................................21
   3.3 The architect as game designer ......................................................................24
      3.3.1 User-out vs. top-down .............................................................................24
      3.3.2 Video games in the design process? ..........................................................27

4 THE GAME ENGINE ..........................................................................................30
   4.1 What is a game engine? ..................................................................................30
   4.2 Current game engines ....................................................................................33
      4.2.1 Unreal Engine ............................................................................................33
      4.2.2 Unity .......................................................................................................34
      4.2.3 CryEngine ...............................................................................................35
   4.3 Architectural applications .............................................................................37
      4.3.1 Academic research ..................................................................................37
      4.3.2 Professional use ......................................................................................39

5 CASE STUDY: CRYENGINE ..........................................................................41
   5.1 Description ......................................................................................................41
   5.2 Workflow .......................................................................................................43
      5.2.1 Overview ..................................................................................................43
      5.2.2 Visual assets ............................................................................................44
      5.2.3 Flowgraph logic ......................................................................................48
   5.3 Result ..............................................................................................................56
   5.4 Integration in the design process ....................................................................57

6 DISCUSSION: TOWARDS A GAME? ..............................................................60
   6.1 Objectives and rules .....................................................................................60
   6.2 Interface .......................................................................................................61
   6.3 Simplification ...............................................................................................63

7 CONCLUSION ..................................................................................................64
1 INTRODUCTION

In the last few decades, the discipline of architecture has become an increasingly complex marriage of many subdisciplines. Today, architectural designers have to reflect not only on aesthetics, material mechanics, and economics, but also energy performance, acoustics, psychology, sociology, and more. At the same time, the physical world we inhabit has lost some of its weight through the superposition of the now ubiquitous networks of digital communication and virtual cloud-based worlds. This has left the discipline in a difficult position. If architecture can no longer be the same powerful achievement it used to be, then what can it still be? In search for an answer to this question, the paradigm of the game has been proposed. Architecture is a game. While this statement might seem bold, the two do have a lot in common, as American architect and professor Wes Jones observes.

Like so much else today, the discipline of architecture has come to be taken less seriously – by itself and by others. […] Architecture has been exposed, frankly, as a game. […] Both seem to demand a surprising amount of attention for (mere) diversions, perhaps because both are concentrated versions of their respective worlds: the game is a reduced slice of life, and architecture is an intensified version of building.¹

Jones considers architecture and games to be similar primarily because of their intrinsic quality of being able to capture and concentrate meaning. Because of the connotations of the term ‘game’, this comparison also serves as a provocation, encouraging architects and architecture students to approach their discipline from an unconventional point of view. The notion that architecture might in fact be a game offers a framework that can be used not as a ready-made answer, but rather as a means to fuel the ongoing discussion about architectural design and urbanism. It might be important to note that this is not a development of the 21st century. The moment in which architects realized the importance of games for their trade can be situated in the 1950s, “when the modernist functionalist approach to architecture revealed itself as no longer useful for addressing contemporary artistic crises. […] For these ‘new’ architects of the 50s, architecture had to relate to a kind of new humanism to which the physical and psychological presence of man in space was integral”.² This approach puts the architect in the position of a player and correspondingly sought to strengthen the sense of playfulness in the architectural design. As video games made their entrance in mainstream culture, game-oriented designers naturally became interested in the possibilities of this new technology. Despite the fundamental

'unreal' aspect of video game spaces, their dynamic interaction model and powerful ways of immersing players, could hardly be ignored by designers of real-world environments. The more wide-spread acceptance nowadays of the idea that video games and architecture are strongly related is perhaps best illustrated by the publication of *Space Time Play* (2007), a book that compiles over 180 articles from various authors (practitioners and theorists of architecture as well as game designers) on the topic of video games, architecture, and urbanism. The editors of the book included an introduction in which they explain their motivation, highlighting the perceived need for a clear reaction to the profound influence of the *virtual* on the way people experience space.

The digital spaces so often frequented by gamers have changed and are changing our notion of space and time, just as film and television did in the 20th century. [...] Today we again face the development of new typologies of space – spaces that are emerging from the superimposition of the physical and the virtual. The spaces of the digital games that constitute themselves through the convergence of space, time and play are only the beginning.

The articles following the introduction are meant as an attempt to answer the question what the parameters of these new spaces are and how they can influence our design strategies as well as the actual characteristics of the design. However, the core message of *Space Time Play* is clear: architects have been struggling to keep up with the developments of our digital era and would benefit from paying closer attention to the ultimate generators of space and spatial interaction in the digital realm – video games. Perhaps the essence of real architecture is most clearly manifested in the dynamics of polygon-built, virtual worlds. Architecture is not just a game, it is a video game.

This thesis will examine the usability of the video game engine as a medium in an architectural design context. The first two chapters serve as a theoretical background and will go into greater detail about the similarity between video games and architecture, as well as the concepts behind the architectural design process and the use of media in this process. Subsequently we will introduce the game engine and discuss the efforts that have already been made in using it for architectural purposes. The final chapter encompasses a practical case study of the CryEngine game engine.

---

2 ARCHITECTURE AND GAMES

2.1 Definition(s) of game

In order to better understand the similarity between architecture and games in general, and the relevance of this similarity, it is crucial to first define game. While it is not our intention to delve deep into the history of game theory and studies – an area generally referred to as ludology – it would be beneficial to give a brief overview of the most relevant studies on this topic. For the sake of clarity, we will consider the terms game and play as equivalents from this point on. Indeed, both words are commonly treated this way in the English language. However, not every author chooses to do so, with the notable example of game theorist Gonzalo Frasca who argues that “games have a result: they define a winner and a loser; plays do not”.

Although theories about play date back to as early as Plato5 (5th century BC) the modern study of play can be traced back to the publication of Johan Huizinga’s influential work Homo Ludens (1938), in which he identifies a number of characteristics that an activity must have in order to qualify as ‘play’. According to Huizinga all play is fundamentally voluntary. This quality of freedom indicates that none of the involved players feel obliged to take part in the game, they play because they simply want to. At no point is the activity considered a necessity. Furthermore, the activity of playing escapes from the enclosure of ordinary or real life. It is crucial that all involved players are aware of this collective escape from reality into the realm of the game. However, this does not prevent the game from being carried out in an utmost seriousness. The players can be fully engrossed in their activity but ultimately still know they are pretending, that it is only a game. Because play is essentially disconnected from the real world, it cannot serve a tangible real-world purpose (such as making profit). “It interpolates itself as a temporary activity satisfying in itself and ending there”. A third characteristic of Huizinga’s play is its secludedness in both time and place. Due to this restriction in time, play can be repeated, transmitted to new generations, and repeated again, almost indefinitely, resulting in the creation of traditions and

---


5 Plato considered play (Greek: paidia, etymologically related to the word for ‘child’, pais) irrational and morally questionable, while simultaneously acknowledging that intellectual play in some form could be used as a tool to improve understanding. Notwithstanding etymology, there is strong evidence that the ancient Greeks did not dismiss games as merely an activity for children or therefore took them less seriously.


culture. Therefore, games allow to capture and concentrate meaning – a notion Wes Jones referred to when he made the comparison with architecture. Huizinga’s play is also restricted in time, carried out inside a marked out play-ground, an intensified, idealized version of the outside world. Here the rules of everyday are suspended and make way for a new order that captivates the player and imposes its own rules on all participants. The interaction between this order and spontaneous play also lends a natural beauty to the activity. Individuals who do not respect the order of the game, spoil it, break the illusion. The game ceases to exist and the rules of everyday apply once again. Ludologist Hector Rodriguez summarizes Huizinga’s definition of play as “a free and meaningful activity, carried out for its own sake, spatially and temporally segregated from the requirements of practical life, and bound by self-contained system of rules that holds absolutely”.

He also stresses the importance of Huizinga’s work, arguing that his theories, along with the assumption of the central role of play in culture, have laid the very foundations for many subsequent writings on the topic of play by scholars such as French intellectual Roger Caillois (Man, Play and Games, 1962). As the academic studies of play and games outgrew their limitation to the fields of history and anthropology, Homo Ludens remained a reference work for scholars and game designers alike.

Still, Huizinga’s definition is not the only viable one. Rodriguez mentions an important point of disagreement about a particular aspect of play, namely its seriousness, or rather lack thereof. Indeed, Huizinga argues that play is essentially not a serious activity (despite the possibly serious behavior of the players involved), “but his writing is notoriously elusive on this point”.

The question whether play can or cannot be serious and what seriousness exactly means in this context has important implications for designers of ‘serious games’. This term was introduced in 1970 by Clark C. Abt in his eponymous book and has become generally accepted as referring to “games that aim at training, educating, persuading or communicating values and idea’s”. However, the adjective ‘serious’ remains extremely problematic because it implies that all other games are non-serious, and if Huizinga’s theories are to be followed, serious games are no games at all. Serious games available today are developed for a variety of purposes, ranging from military and corporate training to healthcare and education. In principle, the term can apply to a broader range of applications, including video games that are not typically regarded as

---


‘serious’. *SimCity*, a city-builder simulation game first released in 1989, is an example of a widely released commercial title that could be considered a serious game, according to ludologist Gonzalo Frasca. He argues that *SimCity* fulfills the rhetorical potential of its genre by communicating structures of interconnection between centralized decision-making and its effects, both short- and long-term, on the functioning of an urban environment. While *SimCity* already treads closely to the field of architecture and urbanism, it is still bound by its nature as a means of mass entertainment. An example of a serious game that approaches the same topic on a more abstract, fundamental level is *Spacefighter*, directed by MVRDV in 2008, which “seeks to model interactive urban development as an evolutionary process, […] reflecting on and comparing imaginable, forecasted, interactive urban processes, actions and reactions”. The game explores the dynamics of urban developments, and – similarly to *SimCity* – demonstrates the effects of the player’s decisions on these developments.

### 2.2 Architecture as a video game

#### 2.2.1 Spatiality and complexity

Video games or digital games are a relatively recent kind of games, obviously due to their reliance on computer hardware, and are therefore absent from the theories of thinkers like Huizinga that laid the foundations of our modern understanding of play in the second half of the 20th century. In our introduction we already suggested that video games, or digital games, might actually be the most interesting kind of games to compare architecture with, due to their exceptional focus on *spatiality*. In video game studies, a subdiscipline of ludology, this spatiality is considered to be the very defining element of their study object. Video game theorist Espen Aarseth argues that “more than time, more than actions, events and goals and unquestionably more than characterization, games celebrate and explore spatial representation as a central motif and raison d’être.” Indeed, video games, as an essentially visual medium, almost invariably require the player to navigate through two- or three-dimensional spaces and/or interact with objects that move through these spaces. Gameplay is thus created through space. This holds true for *Pong* (1972), one of the first arcade video games that simulates a table-tennis game in 2D, but...

---

also for the vast majority\textsuperscript{15} of video games that followed in its wake, ranging from strategy games that present a top-down view on the game world to first-person shooters that drop the player in the midst of it, offering a point of view similar to the one we experience in our non-virtual world. Interestingly, the reliance on space for providing an experience that is both interesting and enjoyable is also what defines architecture and the design process.

Another characteristic of digital games that sets the genre apart from most other games is their \textit{complexity}. Ludologist Brian Sutton-Smith famously noted that the video game is \textit{“the most complex toy ever built and vastly more responsive than any other toy ever invented. […] Chatty Cathy\textsuperscript{16} has a dozen responses; the computer has millions”}.\textsuperscript{17} Complex non-video games already existed in the pre-digital age, but – in the case of strategy board games – they required the players to complete elaborate and often tedious tasks such as moving small figures, representing soldiers, on a world map, managing different kinds of cards and game money, and keeping track of the overall, increasingly complicated state of the game. The arrival of computers allowed game designers to leverage the power of microprocessors and computer memory to take on the burden of background simulations and storing information about the player’s actions, as well as expanding upon the amount of interactions that were possible in the game. The player needed only to give input that made the game interesting and enjoyable, the computer did the rest. Despite the essential contribution of computers to this new medium of digital games, one has to realize that the game is more than computer software running on hardware; \textit{“digital games are systems, just like every other game […] The physical medium of the computer is one element that makes up the system of the game, but it does not represent the entire game”}.\textsuperscript{18} Here, the term ‘system’ is very broad and refers to all elements and their interrelations that in some way contribute to the experience of the game. Still, the capabilities and also restrictions of the underlying computer hardware can have repercussions for the actual design of a game and the way it is experienced, for example when the size of a game level needs to be reduced in order to achieve adequate performance, or when physics simulations were first introduced, allowing players to interact with their environment and create new kinds of gameplay.

\textsuperscript{15} Examples of video games that do not rely on this spatiality are the genre of text-based games that make use of text instead to provide gameplay and convey a story.

\textsuperscript{16} \textit{Chatty Cathy} is a “talking” doll that was very popular in the 1960s, being able to speak a limited amount of different sentences when the user pulled a string.

\textsuperscript{17} \textsc{Sale}n, K. \& \textsc{Zimmerman}, E. (2004) \textit{Rules of play: Game design fundamentals}. Cambridge (USA), MIT Press. p. 85.

\textsuperscript{18} \textsc{Sale}n, K. \& \textsc{Zimmerman}, E. (2004) \textit{Rules of play: Game design fundamentals}. Cambridge (USA), MIT Press. p. 86.
Just like video games, or perhaps even more so, architecture is incredibly complex. While the essence of building can be described in rather simple terms, architecture is a discipline that encompasses numerous subdisciplines. It is the act of designing and realizing a building, while simultaneously keeping an eye on spatial qualities, urban context, aesthetics, material characteristics, stability, thermal performance, HVAC, acoustics, budget, and many more. Computers have undoubtedly made the life of building designers easier, but the architectural game still demands a lot of ‘manual’ input and coordination in order to achieve a coherent result that is more than the sum of its parts. One could argue that both digital games and architecture seek to automate as many components as possible to relieve the player/designer and enable him to concentrate on what really matters and cannot be automated or would not be beneficial to automate (e.g. because it makes the game fun or gives a human touch, an elusive quality, to the design).

In fact, the comparison between video games and architecture boils down to comparing a player, interacting with an artificial, virtual world, to an architect, operating in and designing for a very real world. Both worlds, or rather systems, are highly complex, and while virtual systems rely entirely on computers for their creation, architectural design benefits a lot from computer technology as well because it offers the ability to make the complexity of the real world more manageable.

2.2.2 Objectives, rules and narrative

Now we have established the general similarity of digital games and architecture based on their inherent characteristics, let us look into three essential game elements that are present in both fields. We will distinguish objectives, rules and narrative. All three elements are often treated in writings on game theory and design, but rarely are they considered to be equally important. For example, game historian David Parlett defines game only as a twofold structure based on ends (objectives) and means (dictated by rules), completely ignoring the importance of the underlying story for the game experience. This omission should not be considered a plain fault, however, since not all authors choose to approach the topic of games from within the same theoretical framework. The debate about narrative and games can be described as the conflict between narratology and ludology, while in fact the two are very often interwoven. The reason why the study of video games has traditionally had little regard for narrative is thought to be a result of its perceived desire for self-emancipation.

The advent of electronic games as a new entertainment and art form is sometimes treated as an event divorced from cultural history. [...] According to this view, games in general and computer games in particular display a unique formalism which defines them as a discreet experience, a different genre from narrative, drama, poetry. [...] Attempts by other scholars to discuss games as part of a larger spectrum of cultural expression are denounced as ‘colonialist’ intrusions on a domain that belongs only to those who are studying games as abstract systems.20

This debate between ludology and narratology is ongoing in the world of video games, and the two are still treated as different approaches to the same topic. Coming from an architectural background, one is tempted to immediately draw the comparison with the emancipation of architecture as a discipline that is different and independent from other art forms and sciences. In reality, both video games and architecture are related to countless other fields, and it can be argued that their very strength lies in borrowing from them but ultimately creating a coherent whole that offers an added value.

OBJECTIVES

A first element that is practically indispensable to a video game is the objective. Video games that lack clear objectives are ambiguously called ‘non-games’. However, more often than not, these games (e.g. SimCity) stimulate the player to set his own objectives by providing him with certain tools or rewarding his actions. The player needs a goal to pursue in order to motivate his actions. In ludology, some authors use the Latin word for ‘game’, ludus, to indicate that there is an objective, as opposed to the Greek paidia, which is considered spontaneous play without objectives.21

The main purpose of an objective, besides motivating the player, is defining which state the player has to achieve in order to finish the game or some part of it. The player is presented with one or more main objectives, which he absolutely needs to meet in order to win, alongside a larger amount of smaller objectives that do not necessarily lead to victory or defeat, but help the player defining his priorities. Still, achieving these objectives can eventually contribute to the outcome of the game, simultaneously providing means to value the merits of the player, based on the amount of achieved objectives. The main objective of a game is often very easy to summarize, since it does not specify which series of actions the player should undertake in order to achieve his goal. An example of a video game’s main objective would be ‘capture your enemy’s base’, while smaller objectives could provide an additional challenge, like ‘complete the game in less than one

hour’ or suggest certain actions on the fly for which the player could be rewarded, such as ‘cut off the supplies to the enemy’s base’.

Likewise, in architecture there are main objectives and smaller objectives. The main objective is the assignment to design and/or realize a building that is fit for a certain specified purpose. Smaller objectives can be implicit (cost efficiency), or explicit (‘include at least eight conference rooms’), as well as imposed by the client, or by external parties, such as the government (building performance). Meeting the main objective, that is, delivering a finished building, is an absolute requirement for the architect. Still, architecture is almost never considered a ‘victory’ solely because of this – if at all. Compared to games in general, the objectives that the architect should aim to achieve are often ambiguously defined, revealing a fundamental difference between architecture and games, according to master thesis student Maarten Audenaert.\textsuperscript{22} However, we would argue that this observation does not necessarily hold true for video games, which are typically much more complex than traditional games. An example to illustrate this claim is the turn-based strategy game series \textit{Civilization}, in which the player is presented with an objective that is simply described as “build an empire that stands the test of time”.\textsuperscript{23} While victory can actually be achieved by meeting a certain set of conditions, these are never explicitly shown to the player\textsuperscript{24} and they partly depend on game mechanics that are handled in the background, relying on complicated algorithms. Therefore, although the main objective is arguably clear (building a great, lasting civilization), the specifics of winning the game are not.

The same can be said for the architectural game. Designing and realizing an outstanding building is an obvious goal for an architect, but the aspects of his specific take on the project that eventually contribute to a perceived success or failure are difficult to point out or simply unknown. To put it briefly, architecture is a game with mainly unknown objectives.

\textit{Rules}

The presence of rules constitute a second element that video games share with architecture. Game rules are closely related to Huizinga’s order (cfr. supra) and without them, there can be no game. Rules regulate, they essentially mark the playing field by fixing parameters and defin-

\textsuperscript{22} \textit{AUDENAERT}, M. (2011) \textit{Inzetbaarheid van virtual reality en serious games in het architecturaal ontwerpproces}. Ghent University. p. 70.
\textsuperscript{23} This phrase is also the actual subtitle of the original game, released in 1991, and a recurring theme in the ongoing \textit{Civilization} franchise.
\textsuperscript{24} In order to find out the different kinds of victories (\textit{Civilization V} distinguishes victories based on military domination, science, culture, diplomacy and final game score) and their specifics the player would need to start digging in the elaborate \textit{Civilopedia} game manual by himself.
ing specific means that the player can dispose of to meet the game’s objective, therefore imposing constraints. Beyond this main purpose, rules in video games are also considered “explicit and unambiguous, shared by all players, fixed, binding and repeatable”. Wes Jones distinguishes between natural rules and conventional rules in games, referring to a ball game as an example: “the natural rules are related to things like gravity and the size or geometry of the ball, and evolve directly from the principal activity of the game: smacking it, throwing it, rolling it, dodging it, and so on. The conventional rules then give that activity a goal, a point, and a means for judging the quality of the smacking or throwing”. Although Jones’ example concerns a non-digital game, his reasoning can be applied to video games as well. However, the term ‘natural’, as opposed to ‘conventional’, loses some of its intuitiveness in this context, since a virtual world is artificial, or unnatural by definition. The distinction between natural and conventional rules does establish a certain hierarchy in the system of rules, with the former at the base and the latter at the top. Natural rules are related to the basic properties of all elements in the game that interact with each other or with the player and the nature of these interactions. Therefore, these rules are closely related to what video game designers call ‘core mechanics’. Gravity could be a natural rule in video games, just like in reality, being decisive for common player interactions such as jumping or throwing things. If we take Civilization V (2012) as an example, a more abstract natural rule in this game concerns the division of the world map in hexagonal tiles, each of which represents a type of terrain, such as grassland, forests or mountains. The player’s units can move from tile to tile for a specified total amount of moves, and only one unit can occupy a tile at the same time. These rules are essential to the game and do not rely on other, more fundamental mechanics. Conventional rules, however, are built on top of the already established natural rules and create interesting gameplay, that is, challenges for the player to overcome. Each terrain type in Civilization V has certain characteristics, e.g. slowing units down that pass through them or giving units that end their turn on the tile an additional defensive bonus. Knowing these conventional rules, the player can then take advantage of them, outrunning enemy units or blocking them by using the terrain intelligently.

The same distinction can be used to describe the rules in the game of architecture:

This basic activity [of architecture] is linked to the idea of rules in general at a ‘deep’ level, where ‘architecture’ becomes a metaphor for ‘organization’ generally. Here the ‘natural’ rules of architecture spill over from the building’s concerns with gravity and geometry into the realm of logic. The two go together: value cannot exist, much less be

---

‘added’ or appreciated, unless it can be recognized and described – that is, unless there are some ‘rules’ by which it might be identified, measured judged.\(^{28}\)

In summary, rules limit the amount of possibilities that are at the architect’s disposal for manipulating his environment and creating architecture. *Natural* rules are fundamental, fixed laws that are specific to our world, and thus also apply to the environment outside the game. By themselves, these natural rules only describe what is physically possible, while *conventional* rules are not nearly as fixed and define which of these possibilities are considered ‘good’ or ‘bad’ in the discipline of architecture. Stacking bricks will always conform to the laws of physics, but not necessarily to the laws of architecture. Architectural rules are the means of which critics can dispose to value the achieved objectives, taking into account how the architect handled the imposed restrictions or even turned them into his advantage.

![Civilization V screenshot](image)

**FIGURE 2.1** — Screenshot of an early-game stage in *Civilization V*: the hexagonal grid encompasses the entire game world and is the foundation on which most of the gameplay rules and interactions in the game are built. Similarly, in architecture, the force of gravity is a fundamental rule that defines the playing field.

**NARRATIVE**

A third element that bears importance to both video games and architecture is *narrative*. Notwithstanding the ludology-vs-narratology-debate, many video games do rely heavily on storytelling to deliver a satisfying experience to the player. Narrative can be considered the thread that links all game actions together and gives meaning, weight, and context to them. It prevents the game from seemingly being carried out in a vacuum. The player does not just move units from tile to tile on a world map in *Civilization*, he commands armies and leads campaigns against enemy empires, or at least, this is what the narrative suggests. Without this narrative, the actions of the player would be a lot less satisfying, lacking the sense of empowerment that controlling a historic empire would provide. Likewise, without interesting gameplay mechanics, the story

would be just a story, and the player would have a harder time trying to really experience it. Still, the relationship between gameplay mechanics and narrative is not always evenly balanced in video games. In some games, narrative comes first, and game rules and objectives are then implemented in such a way that strengthens the story, while in other games the narrative is just there to flesh out the gameplay.

Again, we can make the comparison with architecture. Narrative is important for architects as well, since stories can provide design decisions with the needed weight. They allow architecture to have “contexts that reach far beyond the physical structure and site, and into the culture that surrounds the building”. The narrative can be embedded in various elements of the design, such as the shape of a room, the material in a facade or the general layout of the plan, evoking sensory experiences, and referring to traditions, ideas, etc. Rules and objectives often relate directly to the narrative. When an architect is commissioned to design a specific type of building (objective), he should take into account not only the natural and conventional rules, but also the context which surrounds the building. Once the architect has familiarized himself with this context he is able to construct a meaningful narrative, incorporating elements that refer to other buildings with the same function, reacting against or reaffirming typical design rules etc. Objectives, rules and narrative are therefore not just separate elements in either architecture or (video) games. They are strongly related and all three have their significance for the architectural design or the game experience.

2.3 A multiplicity of games?

So far we have compared video games to architecture in order to demonstrate their similarities. However, the monolithic image of architecture as one distinct game, does not seem to correspond to the reality of the modern discipline anymore. Wes Jones argues that “since contemporary construction capabilities have largely removed the limitations that historically dictated architecture’s natural rules, and because today’s hyperpluralistic global culture has spawned uncountable permutations of acceptability, it is probably more accurate to think of architecture now as a collection of games”. According to this idea, architecture would be a playing field where all these games, “the re-emerging greenness game, the affordability game, pop-culture/branding game, […] digital game, geometry game, and other technology games” are being carried out simultaneously, some of which having a stronger presence than others. Architecture games obey their own logic, have their own values and measures of judgment, and sometimes

---

even directly oppose each other. The division of the one game into a multiplicity of games serves as an answer to the ever increasing complexity of architecture’s subdisciplines as well as the apparent isolation in which the games are being played. Sometimes architecture seems to be dominated by one specific game, without much consideration for the others, for example when the shape of a concert hall is purely derived from acoustic simulations, or when the positioning of a corporate building and the design of its facade primarily serve the purpose of maximum visibility and branding. Jones criticizes this insularity of contemporary architectural games, which is, according to him, due to their self-interest and sense of entitlement: “Born at the near end of a long history of technical struggle, and subscribing to the still-hardy romance of the avant-garde, the games mistake the divorce of their own elective formal predispositions, within the already natural openness of architecture, as an achievement.”31 The notion that architecture is not just one game but a collection of games is thus as much an acknowledgement of its reliance on various, increasingly complex subdisciplines, as it is a critique on its divided character. Once again, we could draw a comparison with video games, in which certain mutually related gameplay elements can seem to be disconnected from the others to such an extent that they are considered to represent a mini-game within the larger enclosure of the main game. If one particular mini-game is too dominant, the coherence of the entire game, as well as its overall identity is at stake. Indeed, the essence of the entire experience is more than just the sum of all the mini-games, and the same can be said for architecture.

3  PLAYING THE DESIGN GAME

3.1  Introduction

In the previous chapter we explored the similarities between architecture and games by taking into account their shared characteristics and pointing out some essential game elements. However, all this time the word ‘architecture’ has been used ambiguously, sometimes referring to the product of a process, other times to the process itself, or even both at once. This is complicated even further, because the process of realizing this product consists of both building a design and conceiving a design, which are very different from each other. Only once the designer begins building, or even later, when the building is finished, does it start to become clear to him how the conceived architecture stands up to the rules of the real world, how it is valued by its users and observers, and how it influences its surroundings in the broadest way possible.

The enormous amount of information that is required to have a sound insight into the precise repercussions of a specific design introduces a first difficulty the architecture practitioner has to deal with. A second one is a result of the fact that, during the entire design process, the architect’s relationship with the final product is an indirect one, necessitating the usage of a medium that is merely a representation of the actual thing. Inevitably, this medium has its own shortcomings and biases towards certain kinds of architecture, which, in turn, influences the outcome of the design process. Therefore, the medium in itself dictates an entirely new game that only symbolizes the ‘real’ game of architecture.

These two difficulties, which we will explore first, are key to the design process and result in what is essentially a reflective practice – a concept elaborated upon by influential thinker Donald A. Schön and signifying an activity in which challenges are overcome through experience-driven improvising. Finally, we will return to the topic of video games by having a look at the comparison between the architectural design process and video game design, as a means of critique, and in order to introduce the game engine as a tool for the architectural designer.

3.2 The architectural design process

3.2.1 Solving a wicked problem

One of the main reasons why the architectural design process is thought of as being difficult, has a lot to do with the lack of clear objectives, as mentioned earlier. Identifying problems and addressing these correspondingly by means of a specific design, without losing track of the bigger picture, proves to be one of the biggest challenges a designer has to deal with. Therefore it seems that designing architecture boils down to solving a *wicked problem*. The term ‘wicked problem’ was formally described for the first time in a 1973 treatise by American professors Horst Rittel and Melvin Webber, in which they apply the term to social planning.\(^{33}\) The authors argue that the professional rationalism and efficiency on which modern scientific research and civil engineering are based, have failed to provide the right tools to deal with our societal problems. Because a rationalist method for problem solving relies on *system analysis*, it is naturally unsuitable for coping with problems that are deeply rooted into society, since the system of interconnected nodes would simply be too large to contain. Attempts to expand the boundaries of the system model, thereby internalizing externalities, were insufficient to amend the rationalist method. According to Rittel and Webber, this is due to the distinct nature of societal problems.

*The classical paradigm of science and engineering – the paradigm that has underlain modern professionalism – is not applicable to the problems of open societal systems. [...] The kinds of problems that planners deal with – societal problems – are inherently different from the problems that scientists and perhaps some classes of engineers deal with. Planning problems are inherently wicked.*\(^{34}\)

This is also true for architectural design, which deals with planning problems that concern our community as a whole. When an architect is commissioned to design a building, it is crucial for him to develop an opinion on the societal problems that the building will be addressing. For example, he cannot responsibly design a house without considering the way people live or could live, or design a school without looking into the difficulties and challenges that our educational system is facing. Because the architectural designer – typically – designs buildings with a considerable lifespan, he essentially helps shaping the future working of society. Being an architect is a *social* profession. Decisions that are made during the design process have far-reaching repercussions that are hard to predict beforehand and only become visible at a much later point in time. Due to the sheer scale of the system in which the architect operates, both the problem and the effects of the design decisions that attempt to solve it, are particularly difficult to define.


Rittel and Webber provide a series of statements, defining wicked problems, most of which are easily applicable to architecture. In order to have a better understanding of the apparent wickedness of the architectural design problem, we will look into the most important ones.

First, “there is no definitive formulation of a wicked problem”. If the designer is able to select all the information that is required for solving the problem – a definitive formulation would contain this information – he has already solved it. Since architecture is connected with such a large system or context, it is exceptionally hard to recognize the elements in the system that constitute the root of the design problem. Different designers often approach the same project from different angles, select different information to work with, and create different designs. And even if both design prove satisfactory, one can still wonder if the problem has been truly solved. The difficulties that naturally arise when trying to quantify the merits of architecture are testimony to its inherent wickedness. Surely, building performance, stability and durability are quantifiable in a way, but other properties, such as openness, lightness, serenity, etc. are much more ambiguous. Furthermore, these properties cannot always be considered qualities or bear the same weight in the imagined score equation. The existence of such a clearly defined equation would imply that it is possible to know beforehand what exact input will yield ‘good’ architecture. Obviously, this is not the case; there is no formula for ‘good’ architecture.

Secondly, “wicked problems have no stopping rule”. According to Rittel and Webber, one can never truly solve a wicked problem. Since understanding the nature of the problem directly leads to the solution, but there is no way to know if this understanding is sufficient or even legit, there is always room for improvement. Once again, this also applies to the architectural design process. Because the designer can never know for sure if he has found the best possible solution to the design problem, he can never give a definitive answer to it. Of course, in practice, he ceases his work at a certain point because of practical reasons; because he has ran out of time, money, or patience, but these considerations are essentially external to the problem. The wickedness of the design process is perhaps most apparent in the design studio, where most students continue working on their project until the deadline forces them to stop. Being finished with the project days or hours before the deadline is typically thought of as a waste of time that could have been used to improve the design.

36 One could easily draw an analogy to physics problems. Indeed, once the question has been reduced to a mathematical equation, finding the answer is often trivial. Physics problems are not wicked, though, because the amount of information and formulas that are needed to find the answer (not ‘an’ answer), are typically limited. The system can be contained.
Thirdly, “wicked problems do not have an enumerable (or an exhaustively describable) set of potential solutions”.38 This one is closely related to the first statement. There is no such thing as a definitive formulation of a wicked problem, and every formulation leads to a potential solution. Since the problem-solver can freely choose (from a very large pool) which pieces of information to include in his formulation of the problem, the possible amount of formulations and solutions is endless. This property of Rittel and Webber’s wicked problem, can be clearly observed in an architectural design studio as well. Although every student is presented with the same design problem, the end result will always consist of a very wide range of possible solutions. The studio professor can predict, to some extent, on which topics most students will concentrate, or which qualities or sentiments they will try to evoke, but it is impossible for him to envisage every idea that could sprawl from his students’ minds. Of course some ideas are less promising or valuable than others, some will never even be pursued, but it is only a matter of judgements which ones are considered to be solutions, and which ones are not. “In such fields of ill-defined problems and hence ill-definable solutions, the set of feasible plans of action relies on realistic judgement [and] the capability to appraise ‘exotic’ ideas […] that will lead to the conclusion, ‘OK, let’s try that’.”39

Finally, “every wicked problem is essentially unique” and related to this, “every solution to a wicked problem is essentially a ‘one-shot operation’; because there is no opportunity to learn by trial-and-error, every attempt counts significantly”.40 Rittel and Webber depart from the assumption that two wicked problems can never be identical to each other. There will always exist some minor differences, even though both problems might seem incredibly similar. The same goes for architectural design problems. Projects always differ from each other in some way, in their general function, location, societal context, etc. One specific design or set of design rules for a building can never be the most desirable solution for all conceivable circumstances, notwithstanding the ambitions of historical movements such as the International Style, which sought to provide a single universal answer to international design problems. Because every wicked problem is unique, each problem needs to be approached accordingly. An additional difficulty lies in the fact that buildings have a considerable lifespan, and the undesirable consequences of basic design decisions are mostly impossible to remove or amend once the project has progressed beyond the planning phase. Since the designer cannot truly test his design against reality before committing to actually building it, the architectural design process is naturally unforgiving towards experimenting. The irreversibility of design decisions becomes more

apparent and possibly harmful as the scale of the projects increases. A dysfunctional design for a single building cannot compromise the functionality of an entire community, but large-scale urban planning projects most definitely can.

3.2.2 Playing a metagame

Besides having to deal with the ‘wickedness’ of the architectural design process, the architect is confronted with another difficulty that is inherent to his discipline. The metaphor of architecture as game and the architect as player is compromised by the indirect nature of the medium that is used in the design process. Due to practical reasons, the architect is restricted to playing a game that merely symbolizes the real-world act of building and the natural rules (cfr. 2.2.2) to which it is subjected. Whether he makes use of drawings, digital or physical (scale) models to design a three-dimensional building, most of his mental labor involves working with representations. Therefore, the architect does not play just a game, he plays a metagame. The term ‘metagame’ contains the Greek word for ‘after’ or ‘beyond’, métà, implying that the conventional boundaries of the base game are transcended, and was first described in 1971 by Nigel Howard.41 Howard’s metagame is meant as an alternative approach to mathematical game theory that also takes into account the participants’ mutual awareness of the game-like nature of their activity, and the influence of external factors. Metagame analysis subsequently attempts to solve real-world conflicts by examining the metagame behind them.

Similarly, by using his traditional media, the architectural designer frames a three-dimensional problem of tectonics as a game that mimics, but also surpasses the boundaries of the original. He occupies a metalevel that allows him to strip the design process from some of the distracting particularities of ‘physical’ architecture, thus arriving at a higher level of abstractness that facilitates more efficient planning. Pencil lines in architectural drawings cease to be just lines and indicate real walls, or windows, or flights of stairs. This way, design problems are translated, or reformulated by the medium used by the architect. However, playing a metagame also implies that some of the rules that naturally apply to physical, three-dimensional structures get lost in translation, while other, new ones, are introduced. Decision-making in the design process is based on a select amount of information that only partly overlaps with the entirety of actual data. When brought to its logical conclusion, decision-making can never be completely rational this way. The term to describe this, ‘bounded rationality’, was famously

coined by Herbert A. Simon and does not only characterize the architectural design process, but is prevalent in all of society.\textsuperscript{42} Rationality is limited by the amount of information and the capability of the human mind to process this information. This insight is especially valuable when taking the circumstances of the architectural design process into consideration. The bounded rationality hidden in the medium is the subject of an essay by architectural historian Robin Evans, who primarily focuses on what is arguably the most important tool of the architect: the drawing.

\begin{quote}
I was soon struck by what seemed at the time the peculiar disadvantage under which architects labour, never working directly with the object of their thought, always working at it through some intervening medium, almost always the drawing [...]. The sketch and maquette are much closer to painting and sculpture than a drawing is to a building, and the process of development – the formulations – is rarely brought to a conclusion within these preliminary studies. Nearly always the most intense activity is the construction and manipulation of the final artefact, the purpose of preliminary studies being to give sufficient definition for final work to begin, not to provide a complete determination in advance, as in architectural drawing.\textsuperscript{43}
\end{quote}

Evans does not only highlight the unique circumstances of the architectural design process, but also recognizes the drawing’s power as a medium. Because of the artificial relationship between drawing and building, based on the entirely different logic of pencil and paper (or

\textsuperscript{43} EVANS, R. (1997) \textit{Translations from drawing to building}, Cambridge (USA), MIT Press. p. 156.
vectors and pixels for digital drawing), the medium possesses a relative autonomy. This autonomy is not without dangers. Indeed, architectural drawings can represent magnificent structures that could never be built in the three-dimensional, gravity-dependent world. They can also be beautiful in their own right, as a visual work of art, while depicting architecture that might inspire entirely different sentiments when built. “Drawings are therefore both creative and reflective”, they shape their own reality while representing another. Furthermore, Evans suggests that the inherent two-dimensionality of a paper plane, and the orthogonal projection that is used to display three-dimensional objects, encourages an emphasis on frontality in architecture. According to him, many theories on architectural proportions, such as the golden ratio, relied on this frontality to prove their legitimacy. To illustrate the dogmatic nature of these theories, Evans refers to a well-known analysis of Hellen Wills’ face, that sought to prove that the beauty of the American tennis player relied on the golden ratio: “the analysis is not of the rotund, undulating, folded, punctured surface we call a face, but of quite another surface, onto which the face was flattened by the process of photography”. In architecture, the order is reversed; proportions are first imposed on a two-dimensional projection (through the drawing as medium), which is then translated into a three-dimensional object. When looking at the real thing, perspective distortion subsequently hides the very proportions that are believed to lend a sense of beauty to the visual experience.

Even within the realm of the drawing, different styles and languages can be distinguished, all of them making different contributions at different stages in the design process. For example, a drawing made with a pen and a ruler promotes an exactness that might reveal issues with a building’s detailing, while rough, spontaneous sketches are often the tool of choice when exploring concepts at earlier stages. The enormous importance of the sketch in the design process is a recurring theme in the work of Gabriella Goldschmidt. She refers to sketching as “an extension of mental imagery”, which allows designers and non-designers alike to quickly validate new ideas and highlight their specific strengths. The unique qualities of the sketch can largely be explained by the openness and freedom of the medium. While the ‘reality’ of the sketch is still fundamentally artificial, its distinctive fluency and inaccuracy still leave space for different interpretations and evolutions of the initial idea. In other words, the rules of the sketch are less binding than those embedded in other types of drawings, and are often highly dependent on the personal style of the sketcher.

Evans and Goldschmidt prove that the drawing or sketch is not just an ‘innocent’ medium, because it clearly leaves its marks on the architecture derived from it. Similar things can be said for other design media that are used by architects in the design process, ranging from computer simulations to physical scale models. Modern CAD software may claim to bridge the gap between 2D and 3D, but ultimately, computer graphics are still bound to the two-dimensionality of computer screens. Furthermore, one of the most practical methods to create digital 3D geometry is directly derived from 2D, and involves the extrusion of simple 2D shapes in the direction perpendicular to the display plane. Still, most CAD software does allow for the creation of complex, doubly curved 3D shapes, optionally aided by parametric design to make a subdivision in smaller, more manageable building components. One could argue that digital technology frees the architect from the confines of traditional media, but – however true this might be – this freedom of shape does not automatically result in ‘better’ or ‘worse’ architecture. It inevitably does contribute to the way architecture is practiced today, aiding in the development of a “new language of architecture [that] seems to be based upon the adoption of a new generation of 3D modelling tools. […] Indeed, the choice of a representational/design medium has a huge impact on the character of the design results. The medium is never neutral and external to the work”.

Notwithstanding the benefits of free form and meticulous control over all details in the design, CAD software also bears disadvantages, such as cumbersome, often counter-intuitive input methods, and an inherent preciseness and ‘perfection’ that “does not encourage further discovery, exploration and generation of new ideas”.

The realization of the power of the medium, of the underlying metagame, is crucial for the architectural designer and obviously very relevant for this thesis, in which the capabilities of a new design medium are examined.

3.2.3 A reflective practice

Now we have looked into the so-called wickedness of the architectural design problem and the consequences of the indirect relationship between medium and building, it makes sense to project both insights onto the design process. But how does a designer work on a problem that is impossible to fully contain and does not allow for an unambiguous definition, while disposing only of tools that are inevitably flawed in their translation and communication of ideas? In order to tackle these obstacles, he can engage in the design process in the role of a ‘reflective practitioner’ – a concept that is elaborately described by Donald Schön in his 1983 book. Schön proposes a new

---

way of practice-based thinking that is particularly suitable for professions that are difficult to master without considerable experience, and to which a simple, rationalist approach does often not suffice for an answer (cfr 3.2.1). The author observes that there has been a crisis of confidence in professional knowledge that is related to the disconnected relationship between researcher and practitioner, stating that “researchers are supposed to provide the basic and applied science from which to derive techniques for diagnosing and solving the problems of practice. [...] The researcher’s role is distinct from, and usually considered superior to, the role of the practitioner”. A possible solution to this would involve a seamless integration of thinking and practicing, or rather reflection-in-action. In order to illustrate his point, Schön discusses various professional disciplines, of which architecture is obviously the most interesting to us.

Schön’s reflective practitioner recognizes the complexity and wickedness of the architectural design problem, but does not consider this to be an unsurmountable obstacle. Interestingly, the biased nature of the design medium is key to overcoming it, rather than just an undesirable side-effect. The author also expands this medium to include more than just drawing: “drawing and talking are parallel ways of designing, and together make up what I will call ‘the language of designing’. [...] The language of designing is a language for doing architecture, a language game [...] a language about designing, a meta-language”. Therefore, drawing and talking are the tools of which the architect disposes while playing the metagame. They impose an order (cfr. Huizinga) on top of the open-ended design problem to effectively limit the amount of possible outcomes. The designer can push his work in a certain direction by limiting himself to pursuing sketches and abstracted drawings that make sense on paper or to concepts that he can logically explain to someone in words. This way, the order of the medium assists in taking control of a problem that initially might seem to be too large to handle. “Although a problem-setting experiment cannot be judged in terms of its effectiveness, the practitioner tries nevertheless to set a problem he can solve. [...] Hence he steps into the situation with a framing of the problem for which he feels he can find a solution”. According to Schön, the course that guides the designer to a solution consists of many small steps. Some of these steps are sidesteps or lead to a dead end, perhaps necessitating a new reframing, but all of them should be reactions to the feedback that the previous steps provided. These reactions emerge from the friction between the design object and the order or frame imposed on top of it by the medium. Coincidence

can play a major role in this process as well. This is perfectly illustrated by the popular anecdote of the architecture student who thrashes his cardboard scale model in frustration, only to find out that the force of gravity succeeded in producing a design that is way more interesting than the ones he could come up with by himself. Subsequently, the student adjusts his design to this coincidental reframing and perhaps discovers new meanings that he never thought of before. Schön summarizes the architectural design process as follows.

A designer makes things. Sometimes he makes the final product; more often, he makes a representation – a plan, program, or image – of an artifact to be constructed by others. He works in particular situations, uses particular materials, and employs a distinctive medium and language. Typically his making process is complex. There are more variables – kinds of possible moves, norms, and interrelationships of these – than can be represented in a finite model. Because of this complexity, the designer’s moves tend, happily or unhappily, to produce consequences other than those intended. When this happens, the designer may take account of the unintended changes he has made in the situation by forming new appreciations and understandings and by making new moves. He shapes the situation, in accordance with his initial appreciation of it, the situation ‘talks back’, and he responds to the situation’s back-talk.54

The architect responding to his design’s back-talk can be understood as reflection-in-action. It is a sort of practical knowledge or intuition that only surfaces in the midst of the design process, when the designer is interacting with his medium. This explains why studio teachers often encourage their students to develop a hands-on mentality, to make countless sketches and use scale models as a design tool, rather than just a presentation tool. Schön illustrates the concept of reflection-in-action with the performance of a big-league baseball player, more specifically, a pitcher. In order to control the game, the pitcher needs to get a ‘feel for the ball’, automatically adjusting his pitching technique during the game based on how well it has been working against his opponents. This intuitiveness involves “a kind of reflection on their [the players’] patterns of action, on the situations in which they are performing, and on the know-how implicit in their performance”.55 Similarly, in order to become better, the architectural designer needs to develop a sensitivity for the feedback that he receives during the design process. The more experience he gains, the better he will be able to react on the feedback, discovering similarities to previous projects of which the outcomes are known to him.

Schön’s analysis of the design process is relevant to us, because it provides a better insight in the (meta)game that the architect plays. Its daunting complexity is constrained by the abstracting power of the designer’s tools. Indeed, in order to even arrive at a (not ‘the’) solution, the game needs additional rules and constraints, which are embedded in these tools. Otherwise the architect/player gets lost in an excessive amount of degrees of freedom, without a sense of direction and objective. By constantly evaluating the effects of his decisions and measuring the resistance he encounters, he then steers towards his goal, occasionally adjusting it to his means and vice versa.

3.3 The architect as game designer

3.3.1 User-out vs. top-down

The comparison that we made between architecture and games relied on the idea that making architecture is similar to playing a game. However, this approach puts the architect in a privileged position as the only one qualified to play the game that is architecture, leaving out the people that actually use the building and experience the architecture. American architect and professor Richard Bender noticed this conceptual shortcoming, stating that “what has become equally important to me is the understanding that, as architects, we not only play in this way [using the moves and the rules of the game], but we design games which others play”.56 This ‘correction’ preserves the role of the architect as an authority while also including in the game the people that are not actually involved in the design process but still ‘use’ the architecture – they are the players. Bender gives rise to the idea that the architect is a game designer, inventing scenarios, or rather frameworks, in which each person can play his or her own game. He also uses this notion to point out some of the dangers that, according to him, the discipline of architecture is facing:

*It is not easy to design a good game. Too often architects’ and planners’ ‘games’ are too constricting. It’s too tempting to fix the outcome. Too little is left to the players. We do not trust them to bring richness to the game. [...] If we approached chess the way we do most buildings or neighborhoods, we would market not only the board, the chessmen and a set of printed rules, but each step of the games themselves. [...] We would not want to approach a building design in this way.*57

In other words, Bender criticizes the lack of freedom that architecture provides. In his opinion, the people who use the building designed by the architect, who actually play the game, are not

given the means they require in order to shape their own experience and make it meaningful that way. This shortcoming results in architecture that is overall a lot less engaging than it could be. Perhaps, in order to learn to design better games, architects should turn their attention to ‘actual’ games that do succeed in capturing an audience, before they increase the scale and complexity to the level of real architecture. There have been attempts by educators to use the approach of designing games in architectural design studios, but the resulting games were still meant to be played by designers, not by the users. For example, the experiment conducted by Taiwanese PhD researcher Sheng-Fen Chien required students to “analyze designs of a particular style or by a particular architect and develop [board and web-based] games that when played could produce designs of the style or the architect”. While Chien does illustrate the potential of using the context of game design to teach architecture, observing that it “enables students to gain design knowledge as well as to be able to view design constraints constructively”, her approach to ‘architecture as game design’ does not address Bender’s concerns.

In 1979, when Bender published his critique, the video game industry was still in its infancy. However, since video games might have more in common with architecture than most other games (cfr. 2.2), it could be valuable to apply the insights of video game design to the discipline of architecture. This is exactly what game designer and former architecture student Chris Totten does in his 2008 master thesis, in which he proposes a design method for architecture that borrows heavily from video game design, in order to remedy some of the weaknesses of the traditional architectural design process. While game design and architecture differ fundamentally in their reliance on respectively a virtual and a real world, they are both fields of design, and the similarity between video games and architecture only affirms the relevance of this comparison. “Though approached from many different ways, design is thought by many to be universal, with successful designs always ending in a meaningful articulation of parts to form a functional and elegant whole.” In essence, both architecture and games seek to join elements in a coherent way that allows the people who use the design to have valuable and interesting experiences.

Interestingly, Totten observes that there is a fundamental difference in the way both fields of design are typically approached. Architecture, so he claims, is taught as a top-down method of

---

design, having the architect – quite literally – look down on the site plan and lay out the general geometry of the design, based on an analysis of the context and organizational concerns. He would then try to fit the required spatial program into this general shape, while also using section drawings to at least help him improve his understanding of the three-dimensionality of the actual object he is designing. Although this method is generally thought of as an efficient way to design a very organized building, “it distances them [the architects] from the actual experience of being inside a space and potentially alienates the occupants for whom the building will ultimately be created”. In the worst case, the architectural experience could be nothing more than the result of a two-dimensional question of geometry.

Game design, on the other hand, exemplifies a bottom-up, or user-out approach, as Totten calls it. This method of designing is almost the exact opposite of the previous one and departs from the basic experience of the user to develop the spaces of the game. Designers first consider the different actions that the player can perform, how he moves, interacts with his environment, and then try to facilitate these in their design in a way that is the most interesting to the player. These actions are core mechanics, the basic actions possible within the confines laid out by the natural rules of the game world (cfr. 2.2.2). The modernist adage ‘form follows function’ is thus rephrased by the game designer as ‘form follows core mechanics’. In order to assure an optimal integration in the game level, it is crucial for the designer to test the design after every change that is made to it. This testing while playing is commonly referred to as playtesting. Because playtesting puts the designer in the position of the user, the player of the game, he is naturally better able to make adjustments from which the user will benefit. The designer plays and designs the game simultaneously. Totten argues that this method of designing can be projected onto the architectural design process in order to counteract the disadvantages of the top-down approach.

*This eventually results in the final, more meaningful product. As a generator for architectural designs this is where the true potential of the core mechanic lies: in the ability of the designer to prototype a design for an audience and change the parti [concept] quickly to better fit the building’s intended function if it does not fit. As long as a building’s experiential qualities remain indicative of the core mechanic, the form of the building itself can be exchanged many times over the course of the design process without losing sight of the design’s overall goal.*

---

This suggestion, to alter the design process through the filter of game design, can be understood in terms of Schön’s reflective practice. Both the top-down and user-out approach involve reflection-in-action, but the vantage point from which the reflective practitioner interacts with his design is different in both cases. Therefore, the feedback that is returned to the designer differs as well. The abstracting power of the top-down approach is not necessarily beneficial to the design itself, even though it helps containing the complexity of the problem. In order to produce a design that better caters to its users, the designer needs to use the right tools. These tools, or media, provide feedback, and should steer the design in a direction that does not alienate the people that will actually experience the built design. Put shortly, the medium should not produce a design that is merely the result of its own abstract logic and alien to the real world. Perhaps it is possible then to apply the video game designer’s medium, which clearly allows for a user-out approach, to the architectural design process.

**Figure 3.2** – A schematic representation of the top-down approach (left) versus the user-out approach (right), demonstrating the fundamentally different vantage point of the designer in both design methods; left, the final object is a consequence of abstract planning, while right, it is derived from the user’s experience of building.

### 3.3.2 Video games in the design process?

Before examining the viability of a design approach based on video game design, it should be noted that tools for emulating the user’s perspective in the design process are not a new invention by any means. Totten’s criticism results from the perception that these tools are underused or lacking in some respect, stimulating a search for new media. The simplest method available is the perspective sketch. Throughout the design process, the architect can derive perspectives from his drawn floor plans and sections in order to provide him with additional feedback and a better understanding of spatiality from the future users’ point of view. However, constructing this kind of perspectives achieved by sketching requires the architect to manually convert 2D...
drawings to a projected 3D rendition, which makes the result very susceptible to inaccuracies and exaggerations. This is where computer software comes into play. Building information modeling (BIM) software, such as Autodesk Revit and Graphisoft ArchiCAD, allows for a concurrent development of 2D drawings and 3D models, so that perspective views are created almost effortlessly. Still, as with the manual sketch, there is no real user-out interaction model and the perspectives are mostly static. This shortcoming is alleviated by software solutions primarily aimed at real-time navigation and 4D simulation, for instance Autodesk Navisworks and Graphisoft BIMx. These tools are very well suited for exchanging information with popular CAD software and are commonly used for design reviews and construction planning in a more advanced design stage. Despite the possibility of real-time navigation from the user’s point of view, the level of interactivity and graphics quality fall short when compared to the vibrant worlds of many popular video games.

More than any other video game genre, the first-person shooter (FPS) manages to immerse the player in a virtual, three-dimensional world, which he experiences through the eyes of a virtual character, as if he were there himself. Furthermore, FPS games are suitable for emulating real-world circumstances (e.g. variable time-of-day lighting, gravity, material properties…) and typically put a heavy emphasis on graphical fidelity, which is why “many of the game industry’s big technological innovations arose out of the games in this genre”.64 Because these games try to mimic the real world as accurately as possible, they could offer an ideal framework for investigating experiential qualities in a design before it is built.

---

Similarly to the medium of the drawing, the category of video games is very broad and shelters many different genres (of which the FPS is only one), each with their own strengths and weaknesses. Some genres aspire to create an environment that is a seemingly close approximation to the real world (e.g. *Battlefield*), while other employ a more abstract, symbolic playing field (e.g. *Civilization*). Some offer a first-person perspective, resembling how we experience our world in real-life, others are entirely built around a fixed, top-down point of view. Different genres favor different objectives, rules, and narratives. As with the drawing, a large variety of approaches could be viable in the design process. It would then be up to the architect to choose or construct the game that is best suited as a framework for his architecture, whether this is a simulation game akin *SimCity* for urban development, an FPS game for creating architectural promenades (preferably without the guns), perhaps a racing game for examining the kinetic perception of car-oriented landscapes, or yet something entirely different. This is where the game engine comes into play.
4 THE GAME ENGINE

4.1 What is a game engine?

In the previous chapters we explored the game-like nature of architecture and delved into the architectural design process to better understand the game, or rather metagame, behind it. Alternatively, this design process can be understood as the design of a game, played by others, rather than the game itself. This critique suggests a new way of designing that is based on video game design, a so-called user-out approach better suitable for ensuring experiential qualities in a design. But what is this game designer’s tool – the game engine? And how can it be transferred from its original industry to the discipline of architecture?

The distinction between a game and its engine is not always clear. Sometimes the core software component, the actual engine, is so closely interwoven with the game world and its rules, that it is difficult to think of them as separate entities. It is suggested that the term ‘game engine’ should be reserved for “software that is extensible and can be used as the foundation for many different games without major modification”. The engine then serves as a framework in which developers can fit custom pieces of content, referred to as ‘assets’. Assets are a general term and can vary from 3D models to lines of code. This framework can be further subdivided into a number of core components or subsystems, for example the rendering engine, the collision and physics engine, the audio system, the animation system etc. Some components are deeply rooted in the engine and do not allow for extensive modification, while others are completely customizable by the user. One could easily draw a comparison to the distinction between natural rules and conventional rules in games (cfr. 2.2.2). Natural rules are embedded in the game engine and are customizable only to a limited extent, while the inclusion of conventional game rules is completely up to the game designer. The freedom offered by the engine to cater it to a wide variety of purposes has grown considerably since the rise of the first game engines in the 1990s. Focusing on one distinct purpose used to be the preferred way of designing game engines because it allowed for a higher level of fine-tuning and optimizing, but the advent of ever faster computer hardware has offset this disadvantage to a large extent.

Despite the larger versatility of modern game engines, they are still somewhat genre-specific. It is easy to understand that a turn-based strategy game such as Civilization puts different demands on the underlying game engine than, for instance, a racing game akin Need for Speed.

---

Both games provide the player with a different point of view and rely on fundamentally different game mechanics – in short, offer a different game experience – which is reflected in the frameworks on which they are built. A game engine that is exclusively developed for one genre can afford the luxury of specializing in the aspects that are important to this genre, putting less effort in the rest. Therefore it is important for developers to select an engine that is made for a similar goal as the one they are trying to achieve. Although “it is now possible to use a first-person shooter engine to build a real-time strategy game, for example, [...] the trade-off between generality and optimality still exists”. Indeed, many essential functionalities would need to be added and developed from scratch, which could have been avoided if the right engine was selected right from the start.

Another consideration that should be made is related to the appearance of modern video games. It is easy to be carried away with the seemingly effortless realism achieved in some games, where walls collapse after being hit, tall grass waves in the wind, rain drops leave splashes on the wet ground, and so on, especially when making the comparison with the typical quality of 3D visualizations achieved in the architecture office. In this context, it is crucial to understand that realism is approached very differently by game designers than it is by architects. Most ‘realistic’ video games embody a sort of pseudo-, or hyperrealism just accurate enough for the player to uphold a suspension of disbelief, which is required for successful immersion. In the end, realism is compromised due to technical limitations and for the purpose of accommodating game mechanics that are actually fun to the player, not just realistic. Another fundamental difference lies in the fact that, in game design, the medium’s abstraction of reality is completely transferred to the final product, while in architecture, it is not. The

---

game designer can allow himself a lot of freedom during the development phase, altering his design tool, the game engine, along the way to better suit his vision. After all, the resulting video game will be based on the same engine. The architect, on the other hand, will always need to have his design tested against a reality that is different from the artificial laws that applied during the design process.

[The video game] model can be worked and reworked into a product within the model world itself, in stark contrast to an architectural model. And so arises a paradox that game designers copy the limits under which real architects operate even though these have no significance for virtual space. […] What architects experience as undesirable limits are welcomed by game designers because they confer authenticity on the desired end-product.69

This results in game worlds that seem to be ruled by the laws of nature, but are still fundamentally artificial. To illustrate this, let us imagine the example of a stone wall in an FPS game that collapses after being hit by a missile. The game would have the player believe that the stone material could not handle the force of the impact and failed as a result. This, of course, is an illusion. A plausible description of what really happened is the following.70 The collision system of the engine first detects the missile hitting the wall’s collision geometry (cfr. 5.2.2). As a result, the missile disappears and smoke and fire particles are spawned to simulate the visuals of an explosion. The engine calculates the damage that should be dealt to the target, possibly based on the type of missile and the distance to its target (in this case, it is a direct hit) and decides that it exceeds the wall’s attributed amount of hit points. The wall – because it is labelled as a ‘destroyable object’ – then falls apart in a number of predefined parts, which were specifically modeled for this purpose. At the same time, the physics engine starts a simulation of the explosion’s blast force, providing each part of the wall with an impulse in a direction that directly faces away from the point of collision. As a result of simulated gravity, the parts will fall down on the terrain and come to rest eventually, after which the physics simulation is halted. This is only one example, but it demonstrates how game engines simulate reality on a level that is realistic only to the eye. It is also a critique on the all too tempting idea that game engines today could be easily used to create a design environment that perfectly duplicates real-world circumstances.

70 This reconstruction is loosely based on the way CryEngine handles this kind of events, though most FPS engines use similar methods. (http://docs.cryengine.com/display/SDKDOC2/Breakable+Objects#)
4.2 Current game engines

For the purpose of understanding the current situation of the game engine market, we will have a look at some engines that are both technically up-to-date and freely available or licensable.\(^{71}\)

As a general trend, it can be observed that game engines, besides offering increasingly advanced technology (better rendering, physics, audio, animation, developer tools…) are also becoming cheaper to license, resulting in an unprecedented competition in the industry.\(^{72}\)

Most modern game engines incorporate the WYSIWYG (‘what you see is what you get’) principle, meaning that the appearance and functionality inside the game engine editor closely correspond to the actual, finished product. This allows for playtesting within the editor without having to compile after every change, which is crucially important for an iterative design process. Despite the enormous availability of game engines, this overview only includes three engines that are used in many recently released commercial video games – thus representing up-to-date technology – and fully support small developers.

4.2.1 Unreal Engine

Unreal Engine (UE) is a widely-used game engine that was initially released by Epic Games, together with their first-person shooter Unreal (1998).\(^{73}\) The newest iteration of the engine is Unreal Engine 4 (2014), which serves as a successor to the UE3-based Unreal Development Kit (UDK). UE4 has been developed with an emphasis on the WYSIWYG process, as well as a complete scalability across a wide variety of platforms. UE4 games can be released on PC, Mac, iOS, Android, Xbox One and Playstation 4, ranging from simple 2D side-scrolling games to graphically demanding 3D games. Besides offering support for advanced DirectX 11 and 12 rendering features, the engines integrates many design tools that seek to empower the level designer.\(^{74}\)

One of these tools is Blueprint, which allows for visual scripting and does not require the knowledge of a programming language.

Until March 2, 2015, UE4 was available to everyone for $19 per month with a 5% royalty on games and other applications developed with the engine. This was already a large departure from UE3’s

---

\(^{71}\) Many proprietary game engines are used by a select number of game studios only, without the possibility of licensing to third parties, e.g. the Frostbite game engine by EA (Battlefield) or Anvil, developed by Ubisoft (Assassin’s Creed).


\(^{73}\) (https://www.unrealengine.com/what-is-unreal-engine-4)

$99 per month and 25% cut of revenue. However, Epic Games changed their pricing model even more drastically, removing the monthly subscription fee entirely, while maintaining the 5% royalty.\(^7\) This royalty is only due if a developer makes more than $3,000 per quarter per game. A subscription to UE4 grants developers complete access to the C++ source code of the game engine.

**FIGURE 4.2** — Unreal Engine 4 can be used to create visually impressive scenes, as showcased in one of Epic Games’ technology demos (left), but is also flexible enough for mobile 2D game development (right). *Tappy Chicken* (2014), a recreation of the mobile game *Flappy Bird* (2013), was made by a single artist using the Blueprint visual scripting system. ([https://docs.unrealengine.com/latest/INT/Resources/Showcases/RealisticRendering/index.html](https://docs.unrealengine.com/latest/INT/Resources/Showcases/RealisticRendering/index.html) [http://www.engadget.com/2014/05/22/unreal-engine-4-tappy-chicken/](http://www.engadget.com/2014/05/22/unreal-engine-4-tappy-chicken/))

4.2.2 Unity

Claiming a share of approximately 45% of the game engine market,\(^7\) adoption of Unity has grown rapidly since its initial release in 2005. Unity’s broad appeal has been attributed to its user-friendly interface and cross-platform integration, offering support for both the previous and current generation of consoles, as well as most (mobile) operating systems and virtual reality (VR) platforms.\(^7\) Another advantage is the game engine’s wide support for file format types and, consequently, for third-party content creation software. While the previous version of the game engine, Unity 4, did not offer many advanced rendering features, this caveat is thoroughly addressed in Unity 5, which was released only very recently, on March 3, 2015.\(^7\) Despite the relatively user-friendly nature of Unity, there is no visual scripting functionality, which makes knowledge of a programming language a requirement for adding game logic.

---

\(^7\) ([https://www.unrealengine.com/blog/ue4-is-free](https://www.unrealengine.com/blog/ue4-is-free))

\(^7\) According to an unreleased report by McKinsey; ([https://unity3d.com/public-relations](https://unity3d.com/public-relations))


\(^7\) ([http://unity3d.com/5](http://unity3d.com/5))
While past iterations of Unity have always offered a free edition of the game engine, these versions lacked some major features, such as dynamic shadows and lighting. The professional version, which did include these features, was available for a one-time payment of $1,500 or $75 per month. As of the arrival of Unity 5, a full-featured version is now completely free (without royalty), as long as the developer makes less than $100,000 with it. Full access to the source code is not included, however, and has to be bought separately.

4.2.3 CryEngine

CryEngine, developed by Crytek, was used for the first time in the first-person shooter Far Cry (2004). The game engine quickly gained a reputation for its technologically advanced graphics and correspondingly high system requirements after the release of Crysis (2007), which gave rise to the catchphrase ‘But can it play Crysis?’ widely used in review articles and forums about computer hardware. Although the current iteration of the engine, CryEngine 3 (recently rebranded to CRYENGINE or CryEngine without serial number) still puts an emphasis on graphical fidelity and real-time visual effects, attention has shifted somewhat to cross-platform develop-
opment for PlayStation 4, Xbox One, Wii U, Windows, Linux, and mobile platforms iOS and Android. CryEngine has many real-time features aimed at fast iteration and includes a visual scripting tool called Flowgraph, which is similar to UE4’s Blueprint.

Crytek offers two versions of its game engine; a free version for non-commercial use, and professional version that can be licensed for a monthly fee of $9.90 (without royalty) and provides full access to the C++ source code. This pricing model was revealed just one day after the launch of UE4 (initially $19 per month), which clearly illustrates the highly competitive nature of the current game engine market. A considerable disadvantage of the free version of CryEngine, besides its restricted use, is the fact that it is discontinued and will no longer receive updates with new features and bug fixes.

![CryEngine's graphical capabilities along with its powerful level design tools, as showcased in this screenshot from Crysis 3 (2013), are considered to be its greatest strengths. Despite relatively high system requirements, the latest iteration of the engine can be executed on mobile platforms as well.](http://www.cryengine.com/showcases/crysis-3)

---


4.3 Architectural applications

This game engine overview clearly demonstrates that the market is quickly evolving today, with competing game engines trying to offer the lowest price as possible, while adding new features and thoroughly revising their rendering technology. The reason for this intensified game engine competition can be attributed to the still recent transition to the eighth generation of video game consoles. New, more powerful hardware allows for graphics with a higher level of realism, and simulations that are computationally more demanding. Therefore, the companies that make game engines are updating their products in order to win over large console game developers, which represent an important flow of revenue.

Interestingly, this fierce competition and the resulting price reductions have increased the game engines’ appeal to designers outside the game industry as well. Game engine developers are now even marketing their software directly to architects, recognizing the medium’s potential as “a common language between all these common fields [game design, industrial design, architecture and filmmaking]”, to quote the words of Epic Games’ founder Tim Sweeney. Although the use of game engines in architectural education has been a popular research subject since the 1990s, the incredibly fast pace at which game engine technology evolves, necessitates continuous research and re-evaluations of its potential for the discipline. Modern game engines’ growing user-friendliness, affordability and capabilities to create realistic, real-time environments are likely to further increase adoption rates in both academic and professional environments.

4.3.1 Academic research

The usability of game engines for the purpose of architectural design and construction has been examined in a wide variety of publications. Researchers mainly highlight the immersive qualities of game engines and their ability to present “realistic virtual worlds featuring user friendly interaction and the simulation of real world phenomena”. Moreover, the user-centered nature of the first-person perspective in most game engines is considered to be a welcome departure from

the typical point of view in CAD software, presenting “an intuitive way of understanding and experiencing a digital building”. An experiment conducted in 2006 by Ralph Jones and Russell Lowe examines the potential of game engines when used by students as a central tool in a landscape architecture studio. The most important advantages compared to physical scale models are found to be a better comprehension of scale, a superior sensory richness of the architectural experience (visual and auditory), a more complete understanding of space and time (due to the sequential nature of moving through the landscape), and the ability to interact with other avatars and explore each other’s work in a more engaging fashion. A rather significant but sole disadvantage indicated by the students is the steep learning curve of the game engine software and sometimes devious workflow.

While the advantages mentioned by Jones and Lowe apply more than ever for the current generation of game engines, the medium’s user-friendliness has massively improved in recent years. Importing custom objects in a game engine and setting up a walkthrough-ready, first-person perspective presents few obstacles nowadays. However, it has been observed that creating realistic-looking visuals, while maintaining the quality of real-time rendering requires many optimizations and ‘tricks’ that are commonplace in the game industry, but mostly unknown to architects. Creating 3D models with most BIM and CAD software can result in geometry that is too complex and thus unsuitable for real-time rendering in a game engine. Therefore, game artists only model the geometry that can be seen by the player and rely heavily on textures to feign geometry with more detail than is actually the case, while also utilizing models with different levels of detail (referred to as ‘LODs’) that get swapped out depending on the viewing distance. This results in graphically pleasing 3D models that are suitable for display but lack the information needed in the construction industry. Game engines do support highly complex geometries, but the level of detail is proportional to the power (and price) of the computer hardware that is required to maintain playable framerates. Performance optimization is therefore a keyword in the context of real-time rendering. In summary, there exists friction between the typical workflow of a game artist and an architect in terms of 3D model preparation, which has to be taken into account when using game engines for architectural purposes.

---

90 Unreal Engine 1 still required the user to recompile the entire level before playtesting, thereby significantly hindering the iterative nature of the design process. This difficulty has since long been overcome by the prevalence of WYSIWYG game engines.
There has also been experimenting with game engines as an educational tool for simulating practical situations in the construction industry, rather than as a design tool in an architectural studio. A notable example of this is the so-called ‘Situation Engine’, developed at the University of New South Wales, which is defined by their creators as “an application that provides for specific and managed practical building and construction experience to be made available to students through advanced digital technologies”. The game engine’s (CryEngine) immersive qualities are used to embed students in realistic work environments in order to better prepare them for their transition into a professional practice. The semi-reality that is found in most game engines is obviously very valuable for these kinds of serious game applications, providing native support for physics and other environmental conditions (variable weather, time of day, etc.) that play an important role in real-life situations.

4.3.2 Professional use

Despite the considerable amount of academic research on the application of game engines for architectural applications, adoption in the professional field has been sparse and mainly focused on pure visualization. The narrowing gap in visual quality between real-time rendered and pre-rendered images has led an increasing amount of architectural firms to integrate real-time visualization into their workflow. Being able to render walkthrough videos in real-time as opposed to multiple hours when using traditional rendering engines, such as V-Ray or Octane Render, allows for quicker iterations and a higher level of flexibility when dealing with clients.

While real-time has long been a feature in some specialized software packages (e.g. Autodesk Navisworks or Graphisoft BIMx), the graphical fidelity of these visualizations is rather poor in comparison to the capabilities of modern game engines.

As a response to Epic Games’ decision to drop the monthly subscription fee for Unreal Engine 4, popular architecture website ArchDaily summarized the pros and cons of using game engines for professional visualization. The most important advantages are considered to be the flexibility of real-time visualization, the slow adoption rate in the industry (allowing pioneering firms to

---

gain an edge) and the fact that game engines can now be used completely free of charge. Disadvantages listed in the article include the steep learning curve, additional complication of the architect’s workflow, and simply the fact that game engines are not made for architecture, meaning that there is an important risk of creating imagery that is reminiscent of video games or that wanders off too far from reality. However, the strengths of real-time rendering have not gone unnoticed by developers of architectural visualization software. Programs such as Lumion and TwinMotion, both originally released in 2010, try to capitalize on the drawbacks mentioned above and are exclusively meant to be used by architects, combining real-time rendering technology with a more user-friendly interface (in comparison to game engines). A major disadvantage of these software solutions, especially in comparison with game engines, is the price, which ranges between approximately $2000 and $4000, depending on the desired amount of features.

The author also claims that “architects do not produce a commercial product by definition, [which] means that the royalty fee assessed to video game makers does not apply” (http://www.archdaily.com/607849/unreal-visualizations-3-pros-and-3-cons-of-rendering-with-a-video-game-engine/). The potentially dangerous discrepancy between reality and its representation is fundamental and lurks in every medium, not just the game engine. (http://lumion3d.com; http://www.twinmotion.com)
5  CASE STUDY: CRYENGINE

5.1  Description

For this thesis, we will examine the potential of a game engine for creating an architectural design environment. Rather than serving purely as a visualization tool, game engines also allow for scripting and imposing restrictions on the game world and the (inter)actions that take place in this world. When these rules are properly defined, the designer can play with them – literally and figuratively – and come up with a design that both obeys the rules and is likely to possess experiential qualities, being the product of a user-out approach (cfr. 3.3.1).

Ideally, the game engine-powered design environment is a flawless, digital duplication of the real world, subjected to the same laws of nature and looking exactly the same. Only then would we know for sure that the resulting design would perform equally well when built. However, since this is obviously not the case due to technical and other limitations, designers should keep in mind that the medium presents a distorted version of reality which will inevitably affect the design product. Perhaps this realization is even more important when working in a semi-real design environment as opposed to the heavily abstracted environment ruled by paper and pencil, and whose artificial relationship with reality is always obvious to the designer. In an ideal design environment the designer would also have absolute freedom in shaping his building components, as long as the laws of physics are obeyed. Providing this kind of freedom in-game\(^\text{98}\) would be very impractical and goes beyond the scope of this case study. Therefore, we will limit ourselves to a building system of which the components are already defined and modelled with third-party software. This way, the user does not have to leave the game engine environment in order to make changes to the 3D model(s) of his design. One could easily draw an analogy between this way of designing and playing with LEGO bricks. The bricks’ shape and size are fixed and there are rules that define how they can be joined in order to form new combinations. However, in our game engine environment the bricks are life-sized, detailed building components, allowing the designer to walk between, in and over them, and giving a plausible sense of what it would be like to experience the real thing. The top-down approach – characteristic to playing with LEGO – is thus abandoned in favor of a more user-centered point of view.

\(^{98}\) From this point on, we will simply use the term ‘in-game’ when referring to the (in-engine) state in which the user can assume the role of his player character, navigating the game world (or rather design environment) in first-person perspective. While many game engines offer an editor tool that can be used to easily manipulate the game world, navigation and point of view in editor mode lack the immersive qualities of their in-game counterparts.
The case study encompasses two projects: the VUB student residences (1973) by Belgian modernist architect Willy Van Der Meeren (1923 – 2002) and – focusing more specifically on the incorporation of rules – C2008, which is a set of guidelines for social housing design, issued by the Vlaamse Maatschappij voor Sociaal Wonen (VMSW) in 2008.

Willy Van Der Meeren’s student residences, located on the Etterbeek campus of the Vrije Universiteit Brussel (VUB) consist mainly of prefabricated modules, which allowed for a swifter and correspondingly cheaper building process. The modularity of the building system results in an almost unlimited amount of possible combinations, of which the built project represents only one. The game engine is meant as a tool that can be used to explore these other combinations while engaging the designer’s creativity in a playful manner. In order to achieve this, all rules inherent to Willy Van Der Meeren’s building system must be embedded in the game engine. Some of these rules are explicit and clear (e.g. a limitation of the modules’ stacking height due to structural reasons), others are more implicit or even ‘invisible’ because of their obviousness. The impossibility – or rather absurdity – of placing a roof on another roof might be evident to a designer but still needs to be implemented in the rule set managed by the game engine. While this case study does not aspire to present nor implement an exhaustive list of all rules, it does implement the most relevant ones and – more importantly – sheds light on the methods that can be used to achieve this. The inclusion of the C2008 guidelines serves as an additional challenge because of the higher level of abstraction that is involved and the lack of predefined objects that are to be interacted with.

FIGURE 5.1 – The VUB Student Residences during construction (left) and in their current state (right); the modular character of the residences and the rationality behind its construction is evident from the construction photograph.
(http://www.vub.ac.be/ARCH/ae-lab/projects/retrofit)
The game engine that was used for this case study is **CryEngine**. Although the three engines listed in the overview (cfr. 4.2) are all very much capable in their own right, CryEngine was found to be the most interesting one. Its graphical capabilities, powerful editor and intuitive visual scripting tool make CryEngine very compelling for the creation of a believable, regulated environment. Moreover, at the time of initial research, a CryEngine license was considerably cheaper than UE4, while Unity 5 was not even released yet. The free version of Unity 4 lacked features such as real-time shadows, which are crucial for creating believable environments.

### 5.2 Workflow

#### 5.2.1 Overview

In order to import custom visual assets for further use within the CryEngine Editor it is recommended to use third-party software. This software, commonly referred to as ‘digital content creation (DCC) tools’, represents the 3D artist’s primary tools for producing all the visual components encompassed by the game world or level. Since CryEngine uses file formats that are not natively supported by the DCC tools, Crytek offers free plugins that take care of the required file conversions. The plugin files are included in the game engine’s main folder and only need to be copied to the plugin folder of the corresponding DCC tool. It is worth pointing out that, while only Autodesk 3ds Max – our software of choice for this case study – and Autodesk Maya are officially supported for the export of 3D models, there is also a third-party plugin tool available for Google Sketchup. However, the official documentation does not offer support or guidelines for this software package. Preparing textures is only possible with Crytek’s CryTIF Plugin for Adobe Photoshop, which enables the 3D artist to store additional information, other than the actual image, in the .tif format. CryEngine uses a proprietary .cgf format (Crytek Geometry File) for 3D geometry, which is created by the 3ds Max plugin during export. For textures, the lossless .tif bitmaps are automatically converted to the more optimized .dds format (DirectDraw Surface) by the Resource Compiler, based on the additional information specified with the CryTif Plugin.

Inside the CryEngine Editor it is then possible to create *materials*, consisting of at least one texture, and configure the material shaders, which greatly influence their appearance. In order to add functionality and interaction possibilities to geometry objects inside the engine, it is re-

---

99 Development version 3.6.17 – Build 3217 [EaaS].

100 CryEngine Editor does have a so-called ‘Designer’ tool that allows for the creation and manipulation of geometry (up to the level of individual vertices) as well as texture mapping, but dedicated 3D software proves to be more suitable for this purpose, offering a larger degree of control and freedom.
quired to specify entities. Entities are linked to .cfg files and have additional, customizable properties that allow them to interact with the physics system and the player’s input. Most types of interaction, rules, if you will, can be specified with CryEngine’s visual scripting tool Flowgraph. A big advantage of this tool is that it does not require knowledge of a programming language, which makes it very accessible to game designers – or architects – without any programming experience. Drawbacks of Flowgraph are the somewhat limited amount of possibilities and slower performance compared to low-level C++ programming.¹⁰¹

5.2.2 Visual assets

We will briefly look into the process of creating visual assets (geometry, textures and materials) for this case study and the obstacles that had to be overcome in order to import them into CryEngine. Special attention to detail is required in order to truly make use of the game engine’s rendering technology and interaction capabilities. Still, as this case study will prove, one does not have to be a professional 3D artist to be able to create believable environments.

¹⁰¹ (http://docs.cryengine.com/display/SDKDOC2/Flow+Graph+Editor)
In order for 3D geometry to show up as a selectable object inside the CryEngine Editor, one has to respect the hierarchy that is used by the engine. Each object consists of three elements, which have to be set up correctly in 3ds Max before exporting with the CryExport Plugin: the dummy node (1), the render mesh (2) and the physics proxy (3). The dummy node is an invisible helper element that will define the object’s name inside CryEngine and its origin coordinates. It is important to choose a logical, descriptive name so that the desired object can be quickly found in the editor. The main geometry is the actual 3D model and only visible element of the object. Because of this, it is also referred to as ‘render mesh’. When modeling the geometry, one should...
try to utilize as few *polygons*\(^{102}\) as possible, without compromising visual fidelity too much. As a general rule, the higher the polygon count of all rendered geometry in the game engine, the slower performance will be, since the computer’s graphics card will need more time to process each frame. Especially for real-time purposes, it is crucial to find the right balance between performance and image quality. Polygons that do not contribute to the overall appearance of the object should therefore be omitted. The third element – the physics proxy – is required in order for the object to interact with the player and its environment in a physically ‘realistic’ way. The physics proxy consist of geometry that is invisible in the game engine and used for collision detection and physics simulations instead of the main geometry of the object. The simpler the physics proxy, the faster collision and physics calculations will be performed. Again, the aim should be to achieve a meaningful trade-off between physical accuracy and performance. Finally, all three elements (dummy node > render mesh > physics proxy) have to be hierarchically attached to each other in 3ds Max. This is crucial for a successful export process.

---

**FIGURE 5.4** – Example of a successful export setup for objects inside 3ds Max; dummy nodes (green cubes) are attached to corresponding render meshes (left), which in turn connect to physics proxies (right; red). Note the simplified geometry of the physics proxy in comparison to the render mesh.

### TEXTURES

Textures offer a great opportunity to add detail to the base geometry and to make use of CryEngine’s more advanced rendering features. In order to be used as textures, images must be square and the amount of pixels on each side must be a power of two (typical texture resolutions are 256 x 256, 512 x 512, 1024 x 1024 and 2048 x 2048 pixels). Choosing a suitable texture resolution once again boils down to considerations about performance and visual quality. Furthermore, it should be noted that very high texture resolutions will only be noticeable when the object to

---

\(^{102}\) In 3D graphics, every object is essentially rendered as a 3D wireframe, consisting of interconnected points in space, or *vertices*. In most cases, the term ‘polygon’ refers to the 2D shape defined by three vertices and is therefore the same as a *triangle*. 
which the texture is applied can be viewed from up close and the computer screen itself supports an adequately high resolution. For this case study, most textures are based on real photographs, which were then edited in Adobe Photoshop to achieve the desired look and tileability.\footnote{In this context, ‘tileable’ means ‘repeatable’ in both horizontal and vertical direction. Tileable textures are practical since they allow you to map very large objects without the need for excessively large textures. However, starting from an existing photograph, it is often challenging to eliminate visible seams.}

As with most other modern game engines, CryEngine supports a variety of texture map types, each storing different kinds of information about the surface to which they are applied. Some of the most important ones are the diffuse map, the normal map, the gloss map and the displacement map. Multiple texture maps can be combined in one material and rendered on top of each other to achieve a realistic looking result. The diffuse map contains only the color information of the surface and has to look as ‘flat’ as possible for the best results. The normal map plays an important role for defining the object’s appearance when illuminated by a light source, such as the sun. It describes the ‘depth’ of each texture pixel, so that lighting effects can be rendered independently from the diffuse map. The gloss map contains information about the glossiness of the object, defining the sharpness of reflections and giving a sense of roughness or smoothness on a micro-level. In the case of CryEngine, this map is stored inside the alpha channel of the normal map, thereby eliminating the need to for a separate texture file and saving memory in the process. Finally, the displacement map can be used to activate shading effects that fake a sense of 3D depth (not used for lighting) without actually altering the geometry of the surface. This practice is referred to as ‘parallax occlusion mapping’. In order for CryEngine to handle the created texture maps correctly it is important to add an additional extension in front of the actual .tif file extension before exporting with the CryTIF Plugin. The extensions for the map types mentioned above are ‘_diff’, ‘_ddna’ and ‘_displ’.

**Materials**

Just like in the real world, materials define the ‘look and feel’ of the objects to which they are applied, which makes them very important to both game designers and architects. CryEngine’s materials are based on the superposition of one or more textures and can be customized extensively by adjusting the material parameters with the CryEngine Editor’s material tool. However, it is also possible to create multi-materials, which in turn consist of two or more sub-materials. This comes in handy since every object can be linked to only one (multi-)material and most objects (e.g. building parts) combine a wide variety of different surface types (glass, concrete, steel, wood...). The physics proxy also needs its own sub-material in order for the collision de-
tection to function properly. Applying materials and mapping the corresponding textures to geometry objects takes place in 3ds Max, before exporting. When trying to create multi-materials in CryEngine it is required to use 3ds Max’ equivalent ‘multi/sub-object materials’ and assign their sub-materials to the corresponding parts of the 3D geometry. It is also possible to add a ‘UVW Map’\textsuperscript{104} or ‘Unwrap UVW’ modifier to take control of the way in which the textures are projected on the geometry. The CryExport Plugin then allows to sync the material to CryEngine. However, this ‘sync material’ option did not seem to work in our case and triggered the CryEngine Editor to crash. Manually creating a new material with the same name as in 3ds Max alleviated the problem and restored the connection between the 3ds Max material and the CryEngine material. Texture file paths and other material parameters are stored in an automatically created .mtl file, which can be edited with a basic text editor.

\textbf{5.2.3 Flowgraph logic}

Creating a real-time design environment demands the ability to add some in-game interaction possibilities and ‘game rules’. That way, the player can be immersed in a first-person perspective while simultaneously being able to adjust the design and test it against a semi-reality. The

\textsuperscript{104} UVW mapping or \textit{UV mapping} is the process of assigning 2D texture coordinates (described by the U and V axes) to 3D geometry (X, Y and Z axes), thereby defining the coordinate transformations that have to be executed by the rendering engine.
CryEngine Editor includes a visual scripting tool, Flowgraph, which is relatively user-friendly and suitable for the creation of basic game logic. Flowgraph schemes consist of nodes which can receive input, apply operations to it and send their output to other nodes. Functionalities that were implemented for this case study include:

**VUB RESIDENCES**
- selecting modules by scrolling through an inventory
- spawning modules in a grid and on top of each other (stacking height is limited)
- spawning roofs on top of modules
- deleting, rotating and mirroring modules
- returning modules to their position in the grid after physics-induced movements
- shifting the grid by a specified length
- adding vertical paneling to the modules’ sides (connecting them as one physics object)
- adding fully functional doors to modules
- changing the modules’ accent color
- displaying the number of entities per object type

**C2008**
- displaying min. surface area and min. depth per room based on the number of inhabitants
- creating rectangular building footprint and rooms by designating the start and end point
- displaying warnings that prevent the user from creating rooms that are too small/large or have an unacceptable aspect ratio (warnings are active during the process of designating the end point)

We will try to clarify how CryEngine’s visual scripting tool works by looking into some key Flowgraph constructions. Most of these examples are only small components of the rather extensive schemes that were put together for this case study, and help illustrate the weaknesses and strengths of the scripting tool. All of them involve the in-game manipulation of entities and the creation or *spawning* of new entities. Every entity that is spawned receives a unique *EntityID*, which has to be called in order to change any of its properties. One of these properties is the entity’s position, which is given by the coordinates of its point of origin (defined by 3ds Max’ dummy node) in the game world. The thought process behind most Flowgraph constructions follows the same pattern: identifying the entities that are involved in the action (1), optionally checking the requirements that are to be met (2), performing the action (3).

**Basic Actions**
Some of the solutions for implementing basic actions are provided below. The absence of seemingly essential nodes suggests that the in-game application of these actions is not fully supported. However, Flowgraph proved to be flexible enough to overcome this obstacle.
Every box-like element in FIGURE 5.6 - 5.9 represents a node. The leftmost node in each Flowgraph scheme (‘Game:Start’) outputs a signal when the game starts (InGame=1). As a result, the node next in line (Debug:InputKey) activates a specified key (e.g. Key=k in FIGURE 5.6) that enables the user to initiate the action in the in-game environment. Pressing the key enables a node (Input:MouseRayCast) that continuously checks for the position of the mouse cursor until the key is released. When the imaginary ray that runs from the player’s eyes to the mouse cursor intersects with the collision geometry of an entity it will forward its EntityID to the nodes that need this information to be able to perform their actions. From this point on, the schemes in FIGURE 5.6 - 5.9 start to diverge. In FIGURE 5.6 a detour is used to bypass the non-existence of a delete node. Instead, the EntityID signal is forwarded to a Math:SetNumber node which sends the value ‘0’ to the Opacity input port of an Entity:RenderParams node, making the entity geometry invisible to the player. Simultaneously, an Entity:BeamEntity node receives the EntityID and teleports (‘beams’) the entity to the origin of the game world (Position=0,0,0). This creates the illusion of completely removing the entity, while avoiding a negative impact on performance.105

When trying to implement a mirror functionality, similar problems are encountered. Even though Flowgraph does not include a mirror node, we can create a mirror effect by using additional nodes. Also, the mirrored version of every object that the player should be able to mirror has to be exported separately from 3ds Max, which makes the solution somewhat less elegant. As showcased in FIGURE 5.7, this scheme incorporates the delete functionality (bottom-right corner). This is because the mirror effect is achieved by deleting the entity and immediately replacing it with its mirrored version. Indeed, the EntityID is also forwarded to an Entity:EntityInfo node, which outputs the Archetype name of the entity. This name can be specified in the CryEngine Editor and is identical for all entities of the same ‘kind’. The Archetype name passes through a String:SetString node, which converts it to a series of characters (or

---

105 Removing the entity from sight is an additional measure to ensure that the entity will not be rendered by the engine, even though its opacity is already 0. This is required because even invisible geometry will sometimes cast shadows in CryEngine, which is obviously undesirable.
string) and arrives at the input port of a String:Concat\(^{106}\) node. Here the character ‘M’ is added to the existing string, thereby creating the Archetype name of the entity’s mirrored version. In order for this to function correctly, it is obviously required to be consistent when naming the Archetypes (e.g. ‘BasicModule’ and ‘BasicModuleM’ for its mirrored version). The name of the mirrored version is forwarded to an Entity:SpawnArchetype node, which spawns the new entity in the same position as the original one (Pos and Rotate values are received from an Entity:GetPos node utilizing the original EntityID).

\(^{106}\) This is short for ‘concatenation’ which is the operation of joining strings.

When compared to previous actions, implementing a rotate functionality is relatively straightforward (FIGURE 5.8). The entity is simply ‘beamed’ to its original position but in a rotated state (in this case 180°). An Entity:GetPos node receives the EntityID of the original object and outputs the Pos and Rotate values of the entity. The Pos value arrives unchanged at the Position input port of the Entity:BeamEntity node, while the Rotate value first passes through a Vec3:AddVec3 node. Here, a rotation of 180° around the z-axis (B=0, 0, 180) is added to the vector containing the entity’s rotational information (A). While the showcased scheme only allows for rotation of 180°, alternative functionalities could be implemented. For example, incremental rotation of 1° for every 0.1 seconds the input key is pressed would be possible as well, with the help of additional nodes.

\(^{106}\) This is short for ‘concatenation’ which is the operation of joining strings.
A fourth basic action we will discuss here allows the player to *paint* the accent color of an entity (FIGURE 5.9). Painting can be understood as changing the overlay color of the desired sub-material’s diffuse texture map. In order to achieve this effect, a Material:EntityMaterialParams node is introduced. This node receives the EntityID of the entity that is to be painted and specifies the sub-material (here: SubMtID=6) and texture map (here: ParamColor=diffuse) involved. A Math:SetColor node, which determines the RGB value of the paint color (here: 147, 32, 20; i.e. dark red) is triggered by receiving the EntityID and outputs the color to the ValueColor input port of the Material:EntityMaterialParams node. Obviously, the input port of the Math:SetColor node can be easily provided with other colors as well.

![FIGURE 5.9 – Paint functionality Flowgraph scheme.](image)

**COMPLEX ACTIONS**

Besides facilitating these basic actions, Flowgraph can be used to create larger, more advanced schemes as well. Two examples of complex actions are the (restricted) placement of modules (VUB Housing) and the creation of rectangular rooms with segmented walls (C2008). Rather than explaining the Flowgraph schemes entirely in great detail, we will sketch the general philosophy behind these functionalities and only zoom in on some parts.

One of the most important functionalities for the VUB Housing design environment enables the player to select modules and position them in the game world, while staying within the confines of the building system. This means that all modules should be placed in a grid (cell size corresponding with a module’s outline) and the stacking height is limited to two modules. An additional mechanism allows for a temporary shift of the grid by half the length of a module. The grid is implemented by manipulating the HitPos output of an Input:MouseRayCast node, entering from the left in FIGURE 5.10. In order for multiple modules to snap to the grid, the position indicated by the mouse cursor needs to be modified. The position vector is first split up into three separate values (x, y, and z) by passing through a Vec3:FromVec3 node. The coordinates are then rounded to a multiple of the grid’s cell dimensions (9.6 x 2.7 x 3.0 m). This is achieved by three sequential operations: dividing each coordinate by the corresponding cell dimension (Math:Div), rounding the result to the nearest integer (Math:Round), and multiplying this integer again with the cell
dimension (\texttt{Math:Mul}). Two \texttt{Math:Add} nodes (top of \textbf{FIGURE 5.10}) can shift the grid by adding \texttt{B} to the x-coordinate before and after the three operations. By default, \texttt{B}=0, while a shift of the grid by half the length of a module corresponds with \texttt{B}=-4.8 for one node and \texttt{B}=4.8 for the other one.

\textbf{FIGURE 5.10} – Fragment of the module placement functionality, showcasing the grid shift mechanism.

There are two different ways of designating the spawning position of a module. The first one is used when the mouse cursor hits a random spot on the terrain, as described above, and the second one when an already existing module is designated. Hitting a module instead of the terrain is an unambiguous way of telling the system you want to spawn another module on top of the designated module. The z-coordinate of this module’s point of origin (normally '0' when a maximum stacking height of only two is allowed) is requested based on its \texttt{EntityID} and added to the height of a module, which produces the spawning coordinates of the new module. These rules seem quite straightforward, but implementing them also requires additional \texttt{Logic} nodes (\texttt{OR}, \texttt{NOT}, \texttt{AND}, Any…) to avoid issues such as double spawning.\footnote{Double spawning can occur when there is still an \texttt{EntityID} in the ‘memory’ of the Flowgraph scheme when the player clicks on terrain, and the system spawns two modules. To overcome this issue, both spawning methods need to be made mutually exclusive by performing the right checks before initiating the spawning process.}

The main functionality of the C2008 case study allows for the creation of a simple apartment that conforms to social housing guidelines concerning the minimum surface area and aspect ratio per room. The player can first select the number of inhabitants, after which the minimum surface area for each room (and for the total apartment) is altered according to the C2008 guidelines. It is then possible to draw the outer walls and fill the apartment area in with different rooms, which can be selected by scrolling through a list. A rectangular room – the only shape allowed because of its compactness – is defined by two key presses. The implementation of this specific key press mechanism is showcased in \textbf{FIGURE 5.11} (the blue part). Every key press triggers a \texttt{MathSetNumber} node to output the value ‘1’ to a \texttt{Math:Calculate} node, which adds it to the value \texttt{B} in its input port (\texttt{Operation=Add}), forwarding the result to two \texttt{Math:Equal} nodes. These nodes check if the result equals \texttt{B=1} and \texttt{B=2}. If the latter is true, the value \texttt{B} in
the input port of the Math:Calculate node is reset to ‘0’, leading to an alternating output of ‘1’ and ‘2’ with every key press. This way, the system ‘knows’ if the key is pressed for the first or second time in a row, making it possible to display appropriate messages (Debug:DisplayMessage; e.g. ‘the second corner has been determined’) and manage the processing of information differently for the first and second key press.

After the first corner has been fixed by the first key press, the position of the mouse is tracked and the player is presented with a preview of the room’s measurements. If the resulting room would be too small or too narrow, this information is displayed in red and the player is unable to finish the creation process. Only when the surface area and aspect ratio are within an acceptable range is it possible for the second key press to be registered. The outer corners of the rooms appear immediately, giving a quick impression of the area size, and are quickly followed by the rest of the walls. The walls consist of smaller segments, which are gradually spawned, starting from the two positions indicated by the player. An additional difficulty for constructing the Flowgraph logic lies in the fact that the wall segments have an inner and an outer side with distinct textures which should face the correct direction when spawned (the Rot value differs depending on the wall segment’s spawning position).

FIGURE 5.11 – Fragment of the room creation functionality, showcasing the key press detection (in blue).
CHAPTER 5 – CASE STUDY: CRYENGINE

5.3 Result

While the result of this case study cannot be considered a finished product by any means (especially not the C2008 prototype), it is sufficiently advanced to illustrate CryEngine’s potential when used to create a design environment. In its current state, the VUB student residences prototype allows the designer to combine modules according to an embedded set of rules. The player finds himself on an empty terrain, can add modules, manipulate them, delete them, while constantly being able to playtest his creation. He can see it from a variety of angles and, perhaps most importantly, from the perspective of a (virtual) person, encouraging a more user-centered approach.

The presence of predefined rules prevents the designer from making a design that is not physically possible or supported by the building system. In an in-game scenario he does not need to actively remind himself of these rules and restrictions. When he initiates an action that should not be allowed, the game responds by displaying an error message or just by doing nothing. This way, the applying restrictions are recalled in a convenient fashion.
Limitations to the number of possibilities, due to the added rules, are both an advantage and a disadvantage in a typical design scenario. While they are mostly helpful in assisting the designer to abide to real-world limitations, they can also feel restricting by not allowing exceptions to the rule that might actually add value to the design. On the other hand, the strictness of the regulated design environment in our prototypes allows the player to intuitively explore the extends of the playing field dictated by the specific building system.

**FIGURE 5.15** – The VUB student residences prototype; trying to add a third module on top of a stack of two modules is made impossible. An error message saying ‘You aren’t allowed to stack more than 2 modules vertically’ is displayed in red.

**FIGURE 5.16** – The C2008 prototype; the room type to be created can be selected (in blue) by scrolling through the vertical list with the mouse wheel. The user can retrieve additional information (min. surface area and dimension per room) based on the indicated number of inhabitants. If the C2008 guidelines are not respected, appropriate warnings are displayed in red, preventing the user from creating the room.
Both prototypes were made for very specific design situations with specific rules. However, many mechanisms that add functionality and interaction, can be transferred to other, similar design environments. The Flowgraph scheme that enables the player to place objects on top of each other until a specified limit is a fairly general one and only requires a few modifications to function under different circumstances. Most of the basic actions, such as ‘delete’ and ‘rotate’, as well as the grid functionality can also be transferred with relative ease. Indeed, the elements that are specific to our prototypes are often merely values in the nodes’ input ports, for instance the grid dimensions in Figure 5.10. If used by a sufficiently large number of designers, libraries of general Flowgraph functionalities could be created to be shared between individuals. In this regard, it is convenient that Flowgraph schemes can be saved and exported as simple .xml files.

Another point that should be noted when trying to create similar games is related to the visuals in CryEngine. While a comparatively high level of visual quality is achieved in our VUB student residences prototype, creating decent textures and setting up the materials in 3ds Max and CryEngine demands a fair share of work, as demonstrated in the workflow description. It is easy to get lost in small details when striving towards a result that is as realistic-looking as possible. However, the availability of advanced graphics rendering technology does not obligate users to utilize them to their fullest extent, something that arguably could only be expected from professional game environment artists. Especially early in the design process, when general spatiality is more important, playtesting can be done in basic environments consisting of geometry with only one material and one simple diffuse texture map. The powerful lighting system in CryEngine proved to be able to avoid the ‘cartoony’ look that could be expected from this basic setup, as demonstrated in Figure 5.14 and 5.15, where the high building in the background is only equipped with a bland, light gray, diffuse texture map. Alternatively, and similarly to the Flowgraph situation, the architect’s workflow would be sped up significantly if he could choose his CryEngine material from a catalogue of preconfigured materials with corresponding texture maps. However, composing such a catalogue would only start to pay off if the game engine would be used extensively as an important medium in the design process.

5.4 Integration in the design process

Integrating CryEngine – or a similar game engine – in the architect’s design process can be done in different ways. While our case study covers the creation of an entire, regulated, in-game design environment, this is not the only viable approach. Embedding rules that define how different building components can be combined will probably only be useful in design situations that rely on a clearly articulated building system, as is the case with the VUB student residences.
Once the game engine is adapted to this system, the architect can literally play with it until a favorable design is achieved, without even having to leave his design environment. However, this method implies that the design of the system itself and the building components precedes the application of the actual game engine environment. Another, perhaps more frequently usable design scenario, would have the architect depart from a single, roughly defined shape that is gradually modified and refined over time. Here, the game engine could come into play right from the start, but would be combined with a third-party modeling tool that allows for greater control when modifying the geometry. This method of designing is very common in game level design, where it is referred to as ‘white boxing’. This name is derived from the typical white, box-like geometry objects that are used to design the general layout of the game level before adding details to the geometry and breathing life into the scenery through colors and textures. By playtesting a white boxed prototype, designers can quickly check if the design ‘works’ for the player and which parts need to be improved in order to arrive at a more interesting or engaging experience. This design method could easily be used for architectural purposes as well and is clearly well supported by CryEngine. A 3D model displayed in CryEngine will update instantly when the source geometry is modified and exported from 3ds Max without even requiring the editor to restart. The same is true for texture maps altered and saved in Photoshop. When using a multi-monitor setup with CryEngine opened on one screen and 3ds Max/Photoshop on the other, switching between modeling and playtesting could be done in a nearly seamless fashion.

Perhaps the greatest difficulty in integrating the game engine in the design process lies in the interoperability with other, more commonly used software tools. While 3ds Max can bridge the gap between most CAD software (especially from Autodesk) and CryEngine, few architects have access to this software. Furthermore, the 3D models made with other software packages are rarely ideal for display in a game engine environment because of the specific modeling approach supported by these applications (cfr. 4.3.1). Stefan Boeykens distinguishes four different modeling techniques that can be used in the architectural design field, each with their advantages and disadvantages (FIGURE 5.17): polygonal modeling (1), generic CAD (2), building information modeling (3), and digital content creation (4; used in game design). Transferring models between software packages that support different modeling techniques tends to produce ‘translation issues’, often requiring remodeling of critical parts or even the entire model. It should be noted that this phenomenon is therefore not only reserved to workflows that incorporate a game engine. The time spent adjusting models when switching between software can

---


become very significant when multiple modeling tools are used concurrently throughout the entire design process, since automatic synchronization is often not an option. This might be one of the main reasons why many firms that currently employ game engine technology only use it as a means of final visualization. This way, the translation process does only need to happen once. However, the difficulty of this translation should not be exaggerated either. Typically, the 3D models employed by architects are fairly simple, consisting of mostly rectangular shapes and lacking excessive details. Importing these models in 3ds Max and exporting them to CryEngine is likely to yield acceptable results. Furthermore, it is possible to bypass the need for a separate physics proxy (needed for navigation, cfr. 5.2.2) by ‘physicalizing’ the material of the main geometry with the CryExport Plugin. This way the render mesh is also used for physics and collision calculations, resulting in possibly lower performance as a trade-off for convenience.

<table>
<thead>
<tr>
<th>MODELING TECHNIQUE</th>
<th>ADVANTAGES</th>
<th>DISADVANTAGES</th>
</tr>
</thead>
<tbody>
<tr>
<td>Polygonal Modeling</td>
<td>Swift modeling process</td>
<td>No 2D drawings</td>
</tr>
<tr>
<td></td>
<td></td>
<td>No visualization</td>
</tr>
<tr>
<td>Generic CAD (e.g. AutoCAD)</td>
<td>2D drawings</td>
<td>Disconnection between 2D drawings and 3D models</td>
</tr>
<tr>
<td></td>
<td>3D models</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Accuracy</td>
<td></td>
</tr>
<tr>
<td>Building Information Modeling (e.g. Revit)</td>
<td>2D/3D developed concurrently</td>
<td>Limited for freeform geometry</td>
</tr>
<tr>
<td></td>
<td>Integrated listing and visualization</td>
<td></td>
</tr>
<tr>
<td>Digital Content Creation</td>
<td>Freeform models</td>
<td>No drawings</td>
</tr>
<tr>
<td></td>
<td>Visualization and animation</td>
<td>Difficult for accuracy and scale</td>
</tr>
</tbody>
</table>

**FIGURE 5.17** – Different design problems demand different modeling techniques and different modeling software. There is no superior option that is suitable for every situation.

(Figure based on: BOEKENS, S. et. al. (2008) p. 2.)
6 DISCUSSION: TOWARDS A GAME?

6.1 Objectives and rules

As we already discussed in the first chapter, architecture typically incorporates certain game elements, such as objectives, rules and narrative. While objectives are often ambiguously defined or even unknown to the architectural designer, the presence of quantitative information can serve a similar goal by providing the designer with a drive to search for an ‘optimal’ solution. In our case, the amount of information about the design that can be displayed is still very limited (e.g. the number of used entities per object type; VUB student residences) and clear objectives are entirely absent. Therefore the current prototypes can be considered to be non-games (cfr. 1.2.2). The VUB student residences prototype simply drops the player on the Etterbeek Campus and provides him with total freedom, as long as he stays within the boundaries of the building system. In theory, it is possible to perfectly recreate Willy Van Der Meeren’s design. However, the fact that he starts from a blank slate poses the implicit challenge to come up with something more efficient and/or attractive while disposing of the same basic vocabulary. The presence of more clearly defined objectives and quantifiable information could intensify this sense of a challenge. Perhaps the game could take into account the economy of space by keeping track of the module density. Additionally, a system could be implemented that determines the total cost of the design by looking at the number and diversity of used modules, with many instances of the same module resulting in a higher cost efficiency. Of course these are merely examples, but the fact that they both concern measurable efficiency is not coincidental. General appeal or spatial qualities are much harder to take into account because of their subjective nature. This is related to the ‘wickedness’ of the design problem. Incorporating an extensive score system that judges the inherent qualities of the design is fundamentally impossible.

This case study puts a heavy emphasis on the incorporation of rules. The advantage of a game engine as a medium lies in the malleability of these rules. While a more or less fixed logic is embedded in most other media, for example cardboard scale models, the virtual nature of game worlds allows for a total freedom in creating, or rather programming, additional rules or constraints. When working with cardboard scale models, it is possible to create structures of which the full-scale translation in concrete or steel could never support its own weight, let alone resist additional forces. Therefore, designing a structure with this medium requires the designer to constantly question the realism of his representation. In other words, the medium’s backtalk (cfr. 3.2.3) can be a distracting element in the design process if it contradicts real-world rules. While game engines do not allow for reliable physics simulations based on material
properties – which would offer the most convenient solution to these real-world considerations – constraints can be added manually. In our case study, an example of this is the modules’ restricted stacking height of two units (higher stacks would compromise the integrity of the concrete structure). This way, custom-made, conventional rules can be used to mimic the consequences of natural rules in the real world, highlighting the flexibility of the design medium. It should be noted that these rules are artificial and the process of defining them represents an additional task for the designer. In fact, this can be compared to the task of choosing between traditional design media based on their embedded rules and suitability to the specific design goals. While it would be wishful thinking to assume that a modern game engine could readily replace all other design media, it is certainly more flexible because of its customizable rules.

6.2 Interface

Another element that is crucial to any game and still lacking in our prototype is the interface. Generally speaking, the interface is the boundary across which the user interacts with the system of the medium. Human interaction with traditional design tools, such as sketches and physical scale models, tends to be a very natural and uncomplicated process, suitable for the translation of initial ideas into ‘sketchy’ spaces and objects. In the case of computer software, interaction has to bridge the gap between a physical system (i.e. the human body) and a digital system, which can hamper efficient utilization of the medium’s capabilities. This issue is one of the fundamental reasons why CAD software has been regarded as being not user-friendly.

Basic actions in sketching (e.g. drawing a line with pencil on paper) or scale model building (e.g. joining pieces of cardboard) are immediately evident to the user, which is often not the case with CAD software. While video games obviously face the same obstacles as all other computer software, developers typically pay a lot of attention to the design of their interface. This has been attributed to the fact that “the game is not a tool being used to fill an external, utilitarian need. [...] The explicit interaction of the game is not a means to an end; [...] rather, the play of the game represents an end in itself”. Awkward, or hard-to-use interfaces defeat

---

110 This makes the relative user-friendliness of scripting tools such as CryEngine’s Flowgraph or UE4’s Blueprint all the more valuable.
112 This realization has sparked a trend toward more accessible and natural interfaces, based on sketching instead of the traditional and still dominant paradigm of WIMP (window, icon, menu, pointer).
the entire purpose of most video games, which is providing a fun experience. With CAD software, on the other hand, the end result is crucial, and the tediousness or difficulty of the creation process is therefore more easily tolerated.

In its current state, our prototype still lacks an attractive and user-friendly user interface. Each functionality (e.g. delete, mirror, paint...) was simply mapped to a not yet occupied key on the keyboard, resulting in a suboptimal experience for the uninitiated user. Furthermore, the default navigation controls were left untouched and constitute of the WASD keys. Moving around with these keys is a second nature for seasoned FPS players, but it cannot and should not be assumed that most members of the target audience are familiar with this navigation interface. Ultimately, the success or failure of an in-game design environment will highly depend on the intuitiveness of its interface, of which the design could warrant an entire case study of its own.

Luckily, adding such an interface would not require radical changes to the Flowgraph schemes, since only the input nodes would need to be changed. CryEngine supports the Scaleform GFx third-party middleware for rendering Flash elements (e.g. functional buttons and other screen overlays) which can be used to build a graphical user interface (GUI). A less traditional approach could focus on bridging the gap between the physical system of the user’s body and the digital game world. In the context of architecture education, researchers have already successfully utilized Microsoft’s motion sensing technology Kinect as a means to couple real-world gestures to in-game interactions in a virtual construction environment. A logical addition to the use of motion sensing technology is Virtual Reality (VR), which recreates sensory experiences in order to evoke an illusion of physical presence in a virtual 3D world, thereby rendering the interface effectively invisible. Although the term was introduced back in 1989, development of VR technology was long hindered by “underpowered hardware, underdeveloped software, [...] interaction metaphors not based on a deep understanding of human capabilities, and disproportionately little funding for visualization”. It seems that these obstacles are finally being overcome with the development of consumer devices such as Facebook’s Oculus Rift and Sony’s Project Morpheus, and mainstream VR applications are now almost within reach. VR-powered architectural visualization is likely to grow into a more popular research subject as the availability of these and similar devices will increase.

---


6.3 Simplification

In order to arrive at a playable game, reality often needs to be abstracted or simplified. This simplification is evident in our case study of the VUB student residences and especially C2008. The VUB project is naturally ‘simple’ to contain in a virtual design environment because of its underlying grid, modularity, and limited number of different building components. Even so, some details were omitted in order to reduce this number even further and maintain a manageable building set. While we have argued in the first chapter that complexity is a characteristic shared by video games and architecture, complexity in video games does not necessarily apply to their player interaction models. Indeed, gameplay complexity can increase to a point that it becomes detrimental to the player’s enjoyment of the game, which game designers obviously try to avoid at all costs. An unenjoyable game experience, in turn, impedes out-of-the-box thinking and the player’s ability to make creative (design) decisions, rendering the ‘fun’ aspect of games an important concern for developers of game-based design applications. Of course, the perceived level of complexity and enjoyability of a game is also determined by its interface design, in addition to its actual gameplay content.

Another reason for the simplification in our prototypes is the fact that CryEngine does not allow for custom geometry modeling in an in-game environment, as opposed to the in-engine environment of the editor. The player can only spawn or interact with entities, which are fixedly connected to pre-modelled pieces of geometry. The resulting restrictions are most obvious in the C2008 prototype. Currently, the creation of rooms is based on a grid with a mesh size of 0.6 by 0.6 meters, dictating the axes on which correspondingly sized wall segments can be spawned. Here, simplification is thus achieved by concretizing an otherwise freeform building system and effectively shrinking the vocabulary of possibilities. However, it would be unfair to consider the grid merely as an undesirable side-effect of technical shortcomings. It is not coincidental that the grid has been extensively used throughout history in search for more efficient urban planning and the development of standardized construction techniques. Interestingly, the grid has also become a dominant element in video games that involve (city) building, such as SimCity or The Sims. German art professor Annet Zinsmeister has noted that the game environment of The Sims, first released in 2000, is “a virtual world that in its constructive protocols has conspicuous similarities [e.g. the grid] with the first concepts of utopian, ideal and modern cities. [...] Grid and standard (norm) provide the basis on which every player may construct his or her identity out of a range of prefabricated elements”.117 This observation illustrates how virtual spaces in video games can provide commentary on architecture-related issues, again highlighting their mutual similarity.

CONCLUSION

This study was set out to explore the usability of a game engine as an architectural design tool and has approached this subject from both a theoretical and a more specific, practice-based point of view. Although there have been many studies about game engines in the context of architecture and construction, the impressive rate at which game engine technology keeps developing warrants continuous research efforts in this field. Furthermore, it can be observed that the relatively scarce adoption of game engine technology by architecture firms today has been focusing mainly on mere visualization, leaving its potential for the architectural design process untapped. Although the real-time rendering engine is one of their core components, game engines can be used for more than visualization only, which is powerfully demonstrated in the highly interactive, rule-based worlds of video games.

The relevance of video games for the discipline of architecture lies in their fundamental similarity. Video games are typically more complex than their non-digital counterparts and, more importantly, put a heavy emphasis on spatiality. In essence, interaction with 3D spaces makes up the foundation of both architecture and video games, notwithstanding the virtual, ‘unreal’ nature of the latter’s world. Additionally, architecture incorporates many (video) game elements, such as objectives, rules and narrative. Based on these similarities, the application of the video game designer’s main tool – the game engine – for architectural design purposes seems logical.

The architectural design process, however, is highly complex. It is impossible for the designer to contain the entirety of factors that contribute to a ‘good’ or ‘bad’ design and the conception of architecture is seemingly plagued by the fact that architects have to work with media that provide mere representations of reality. The abstract nature of these representations is key to the solution because it reformulates the design problem in simpler terms, simultaneously leaving the medium’s mark on the resulting design. A possible side effect of abstracting reality is the risk that the design becomes purely the result of a system that is only loosely bound with, or even entirely divorced from real-world experiences. Especially the top-down method of designing, typically associated with 2D drafting, is more likely to produce this kind of designs, as opposed to a more user-out approach, in which the perspective of the future building’s user is the main concern. Still, both methods have their virtues, and despite its tendency to neglect the experience of spatiality, the top-down method is inherently better suitable for efficient planning and keeping track of the bigger picture. This insight is crucial when adopting the game engine as a design medium. Just like the architect’s more traditional media, video games (and their corresponding game engines) cater to different user scenario’s with different objectives, rules, and narratives. It is up to the architect to choose – or construct – the right game for the job. The greatest strengths of video game technology, when compared to CAD, BIM and even 3D design
review software such as Navisworks or BIMx, are its exceptionally powerful interaction model and potential to display virtual worlds in a visually more compelling fashion.

Currently, the most capable game engines are arguably Unreal Engine, Unity and CryEngine. All three of them are offering more features and increasingly advanced technology with every update and are more accessible than ever due to an unprecedentedly fierce price battle in the industry. Notwithstanding advances in technology, realism in video games is still mostly an illusion, representing an important obstacle when using a game environment as a surrogate for the real world in the design process. Due to the programmability of a game engine, custom rules and constraints can be added as a means of bypassing the shortcomings in natively embedded rules (e.g. gravity). In this respect, a game engine can be considered to represent a multiplicity of media, rather than just one. Our case study prototypes in CryEngine – while not representing a finished product, or game for that matter – have served as a proof of the game engine’s sufficient flexibility to create a custom, rule-based design environment with numerous in-game functionalities. Also, we were able to implement these rules and functionalities without the knowledge of a programming language, with the help of CryEngine’s visual scripting tool. If creating a design game is not achievable or preferable for a specific design situation, however, the engine can be used for simple playtesting purposes as well. The interoperability between CryEngine and its supported DCC tools allowed for a swift information transfer and iteration in the creation process of 3D models. Setting up convincingly looking materials in the game engine did not reveal any major issue either. Therefore, a lack of technical know-how should not necessarily represent an unsurmountable obstacle for architects trying to adopt a game engine in their workflow. Additional, similar case studies of the current iterations of Unreal Engine and Unity would be beneficial in this regard, especially since both engines have recently become entirely free to license and are even more accessible as a result.

Although their use in the professional field of architecture is still modest and largely confined to mere visualization, game engines are likely to permeate deeper into the design process as architects start to realize the flexibility and interactivity of these powerful software tools. Eight years ago, Dr. Andrew Hudson-Smith – Director of the Bartlett Centre for Advanced Spatial Analysis at University College London – pointed out an additional challenge that would first need to be overcome: “game engines are destined to play an ever-increasing role in the industry; people just need a little vision, and to get over the word ‘game’”.118 Perhaps we are almost there.


