DEPARTMENT OF INFORMATICS

TECHNICAL UNIVERSITY OF MUNICH

Bachelor's Thesis in Informatics: Games Engineering

Game Design Principles for Multiplicative Gameplay

Katharina Maria Seitz

DEPARTMENT OF INFORMATICS

TECHNICAL UNIVERSITY OF MUNICH

Bachelor's Thesis in Informatics: Games Engineering

Game Design Principles for Multiplicative Gameplay

Game Design Prinzipien für multiplikatives Gameplay

Author:	Katharina Maria Seitz
Supervisor:	Prof. Gudrun Klinker, Ph.D.
Advisor:	Daniel Dyrda, M.Sc.
Submission Date:	16.08.2021

I confirm that this bachelor's thesis in informatics: games engineering is my own work and I have documented all sources and material used.

Munich, 16.08.2021 Katharina Maria Seitz

Abstract

In recent years the trend in video game development has shifted notably from a focus on more linear and story-based games to those featuring big open worlds where players have more freedom in deciding what to do and how to do something. In this thesis I examine the concept of Multiplicative Design for video games, which provides practical principles for the design process of games that focus on player freedom. Specifically, I define Multiplicative Gameplay as the gameplay that rises from the interactions of relatively few well-connected game elements which provide players with a vast possibility space; it takes place in a coherent game world that the players perceive as believable and logical in itself and which evokes an experience of freedom of agency. In the thesis I first outline the concept, the advantages, and the challenges of the design concept, and draw a comparison to games of emergence. Then I propose the method of using the natural language labels of subject, predicate, and object to semiformally fill out scenario-based design matrices in order to support designers in developing such games. The method also serves as a contribution to exploring more formal and guided game design techniques supporting game development. Lastly, I present guidelines and possible evaluation methods that give feedback on the design to further support the Multiplicative Design process. The analysis of Multiplicative Gameplay can provide more insight into a wider concept that is increasingly aspired in contemporary game development.

Contents

1 Introduction

In recent years, most major video game companies and smaller development studios alike predominantly released titles that let players have a high level of self-determination. Especially game series in the action-adventure genre that formerly were known for a more linear structure to progress through a story now offer additional freedom in their newer titles by giving players more choice in deciding what to do next, when to do something, and if they want to do certain things at all. Games of the quite new genre of survival are among the most popular ones. The majority of games nowadays offer an open world to explore in order to achieve this level of non-linearity. While the concept of more independence is not inherently new, the norm of how to build video games in the past was to feature only one main path to follow in order to finish the game, or to progress through levels that had to be mastered one after another, or to provide only one way of achieving the game's goal. This was until game developers started incorporating more freedom of choice and more ways of playing, which can be seen as the next logical extension of what games could be. Additionally, reason for this shift could also be the higher replay value of such games, as prices rose and players seem to get more value from games that feature diverse playing possibilities than what a linear game provides.

Considering this current trend, it is relevant to research the principles of these games, ways to master the concept of player freedom in video games, and methods to support the development of such desired game concepts, because more player freedom and massive game worlds bring about many challenges for developers. In this thesis I will present Multiplicative Design and the resulting Multiplicative Gameplay, a concept focusing on gameplay rising from the possibility space that the interactions between game elements provide. The concept is meant to keep development resources manageable despite the demand for a lot of gameplay possibilities, and focusses on providing vast player freedom while conveying a believable game world.

Along with the concept I will then propose a semiformal method supposed to help game designers achieve Multiplicative Gameplay. The process of designing video games is mostly unique to every company and every individual game. While there are standards that are widely utilized in the field, they often consist of informal approaches like writing down the concept in textual form which programmers must then interpret. Therefore, I introduce a method using the natural language labels of subject, predicate,

and object to semiformally fill out scenario-based design matrices in order to support designers in developing such games. This serves as a contribution to exploring more formal and guided game design techniques supporting game development.

2 Related Work

Concerning the study of emergence in games a very early and influential work has been Jesper Juul's *The Open and the Closed: Games of Emergence and Games of Progression* [1], the concept of which I will pull up frequently during the thesis as the theory is highly relevant and applicable to Multiplicative Gameplay. Other contributions to the research of emergence in the context of video games includes *The Open, the Closed and the Emergent: Theorizing Emergence for Videogame Studies* by Joan Soler-Adillon [2] and *Engineering Emergence* by Joris Dormans [3], which give in-depth elaborations on the topic and which I will refer to in order to explain the concept. Lastly, Jesse Schell's "The Art of Game Design" [4], a very popular work about practical knowledge for designing games, features an excellent section about emergence and proposes the usage of the sentence parts subject, predicate, and object, which I will further develop and use in the matrix method for Multiplicative Design. It further describes many of the principles for game design I discuss in this thesis in detail and shows the big picture behind them.

Aki Järvinen's work *Games without frontiers: Theories and methods for game studies and design* [5] provides detailed principles for categorizing game elements into different classes and shows the interplay between them. He also proposes to formulate game mechanics as verbs, and illustrates the relationship between players, the game components they manipulate, and the consequences of the interaction on the game environment, and draws distinctions between rules, goals, and mechanics. He therefore provides vast research on how to formally describe and utilize different game elements for designing games. Besides, for the description of mechanics I will refer to Miguel Sicart's *Defining Game Mechanics* [6] to lay the basis for the theory on game mechanics.

Lastly, I want to mention *Game Design Patterns* [7], the work of Staffan Björk, Sus Lundgren and Jussi Holopainen, where a model for supporting the design, analysis and comparison of games using game design patterns is proposed. Similar to my approach the elements of a game are first analyzed and then integrated into a framework in order to support the further design process of a video game.

3 Multiplicative Gameplay

In the beginning, I will introduce the concept of *Multiplicative Gameplay*. In this thesis the terms *Multiplicative Design* and *Multiplicative Gameplay* will be different in meaning: Multiplicative Design will describe the concept used in the design process of video games, while Multiplicative Gameplay will refer to the way players interact with a game that has been developed according to the principles of Multiplicative Design; thus the former enables the latter.

3.1 What Multiplicative Gameplay is

3.1.1 Origin

The term *Multiplicative Gameplay* has been made known by Nintendo during the Game Developers Conference 2017 when the creators presented their new title *The Legend of Zelda: Breath of the Wild* (Nintendo, 2017) [8] in a talk called *Change and Constant: Breaking Conventions with 'The Legend of Zelda: Breath of the Wild'* [9]. The developers explain that in its core, Multiplicative Gameplay is achieved by connecting elements in the game world with each other so that they are able to interact in certain ways. From the designer's point of view this means that every element that is added to the game should have as many links to the other elements in the world as possible, so that it results in "a game that uses multiplication to expand gameplay" [9]. Nintendo describes the calculation of this multiplication of elements in the following way: player actions * play field * objects. When approached from the player's perspective, as explained by the technical director Takuhiro Dohta, this multiplication creates a possibility space where anything can occur. Dohta argues that this Multiplicative Gameplay is found in the space between situations and goals, where the player can freely experiment and think of their own solution to reach a goal using the possibilities the situation provides. *The Legend of Zelda: Breath of the Wild*, which will be abbreviated with BotW from now on, is a non-linear open world game, where the player will encounter many of these sets of situations and goals. For each of these sets, the developers have envisioned a 'correct' way of solving them, while still leaving room for other approaches. An example for such a set is pictured in Figure 3.1. A possible solution to this problem could be to choose the *player action* of swinging a sword to cut down a tree in the *play*

Figure 3.1: BotW: A set consisting of a situation that provides a possibility space and a goal to cross the gap in the environment.[1]

field which then becomes a rigid body *object* that acts as a bridge to the other side.

To achieve what they call an *active game* as opposed to a *passive* one they used this principle of multiplication of elements to enable a vivid game world, where objects react to the player's actions and objects themselves also influence each other [9]. An example of such an influence between objects in the play field would be a lit fireplace that gets extinguished when it starts to rain. This kind of connectedness of game elements supports a believable and consistent game world; I will further elaborate on this matter in section 3.2.2.

In the talk they described how they broke conventions of their earlier established methods of game design. One of those was the way earlier titles of the Zelda series have been made in a what could be called *additive design* as the counterpart to *Multiplicative Design*. Additive design would then mean that every time a new feature is introduced to the game, it has its own set of items, objects, play fields and player actions that can almost exclusively interact with each other in a strictly regulated way, but not with the other game elements that have already been part of the game world. This concept has been prominent in almost every previous Zelda title: a new area was entered, and new

[1] Picture taken from Nintendo's GDC talk, see [9]

items were introduced with which to overcome its own set of puzzles, which could often only be solved in one fixed way. Outside of these areas the items were often not used again, thus they stay in their own context. This also prevents players from feeling the kind of freedom of choice that Multiplicative Gameplay enables, which I will further elaborate on in section 3.2.3. So, instead of adding a huge amount of elements, Multiplicative Design aims at keeping their number rather small, and instead builds the possibility space out of the many different variations resulting from the elements' interaction with each other, which ideally results in interesting new behavior.

3.1.2 Emergence in games

Multiplicative Design is not completely new, rather it is a combination of already known game design principles into one concept that optimally and elegantly accumulates them into a coherent whole. A concept that is closely related and intertwined with Multiplicative Gameplay is the one of *emergence*. It has been elaborated on quite a lot and has been recognized in the community; therefore it is highly relevant to address it here.

The term *emergence* was originally unassociated with video games. The concept has been researched under the name emergence since the mid-19th century in the context of philosophy and chemistry, later in the 20th century in combination with cybernetics and complexity theory [2]. While the concept is still not normed and different authors have different definitions [10], [11], some aspects are commonly agreed on and are useful when it comes to game studies.

The first one describes emergence as what appears when the whole is greater than the sum of its parts, usually in the context of a complex system [11], [12]. In other words: the whole has different properties than their parts taken separately or together. This proves helpful when looking at games since they can also be considered complex systems [5], [13]; and while they are made out of many parts, the game is more than just the sum of them. Emergence can therefore be understood as rising out of the game code, but not being specifically programmed itself. This compares to the multiplicative concept that gameplay is found in the interaction and thus multiplication of several elements that leads to a variety of possible behaviors, whereas in additive design gameplay arguably converges to the sum of the parts.

The second is related to the first aspect and describes emergence as complex behavior rising from a non-complex space [11]. Holland uses the common example of chess: "a small number of rules or laws can generate systems of surprising complexity." [11]. This already relates to games: many agree that a game is fundamentally made out of rules [4], [5], and although they can be quite simple there can emerge complex game systems. In the context of video games this is often referred to as *elegance* [3], [4], [14]

and directly relates to one of the core aspects of Multiplicative Design: relatively few well-connected game elements provide players with a vast possibility space.

A third aspect that often appears is the rule that emergent behavior cannot be predicted. Applied to games this can mean that behavior is only then called emergent when the game designers did not intend or foresee it. A well-known example for this is the so-called *rocket jumping*, where players of the game *Quake* (id Software, 1996) [15] discovered that when jumping and shooting rockets to the floor underneath them will catapult them across the play field [1], [16]. The developers unconsciously enabled this behavior by the way they implemented the two mechanisms and did not foresee that players would exploit the functionality and turn it into a frequently used feature. Examples like this can arguably be seen as emergence in games, but using predictability of behavior as a premise for emergence - as some designers do, see for example [17] - is critiqued by others. Juul argues that it does not proof to be analytically useful, since it would dismiss the real potential of emergence research in games [1].

Juul instead proposes three different kinds of emergence in games: The most basic form he calls *rule interaction*, which refers to the simple combination of rules where the resulting behavior is easily derivable from the rule set; he names rocket jumping as one example. Juul argues that these kinds of behaviors are too simple to be called real emergence. The second one is *combination* and is defined as the "variety of possible states and game sessions that a game's rules allow" [1]. Lastly, *emergent strategies* is the kind he attributes real emergence to. It encompasses all game strategies, like teamplay in Counter-strike [1]. Comparing these to the concept of Multiplicative Gameplay, I argue that the arising of all three kinds of Juul's emergence is supported by the design, but it does not guarantee it. In general, it is agreed upon that emergent behavior in games is, due to their complexity, non-obvious and thus cannot yet be engineered straight forward. But since it arises from the rules of a game there are design principles that will likely enable it. However, playtesting seems to be the only way to proof its appearance [1], [4].

Apart from the categorization of emergence, Juul proposed another differentiation in his influential paper about the topic: the distinction between *games of emergence* and *games of progression*. The former refers to games that provide "a small number of rules that combine and yield large numbers of game variations, which the players then design strategies for dealing with" [1]; the latter describes a structure where "the player has to perform a predefined set of actions in order to complete the game." [1]. As I have stated above, the counterpart to Multiplicative Design can be seen as additive design. These two again have apparent commonalities to Juul's categorization: like games of progression, additive design leads to more linear gameplay with less or no variety of possible game sessions and allows for more control by the game designer.

By the comparisons I have made now it became apparent that Multiplicative Design

is closely related to the concept of emergence in games. However, there is a difference between them that I will lay focus on in this thesis, namely the importance of connectedness of game elements, which yields the possibility space by multiplication and lays focus on a believable game world through consistency. In section 5.1.3 I will discuss possibilities of cultivating emergence for Multiplicative Gameplay using the design concept I will propose in the next chapter 4.

3.1.3 Definition: Multiplicative Gameplay

To differentiate and delineate the concept I will propose the following definition of *Multiplicative Gameplay*: Multiplicative Gameplay rises from the interactions of relatively few well-connected game elements which provide players with a vast possibility space; it takes place in a coherent game world that the players perceive as believable and logical in itself and which evokes an experience of freedom of agency.

3.1.4 Use case and application in existing games

Now that Multiplicative Design has been introduced, the following question must be asked: For what kind of game is Multiplicative Design applicable? I believe that in principle, every game that in some way focusses on player freedom, choice or connectedness of elements can profit from the concept of Multiplicative Design. It can give valuable reference points to guide the design process, even if designers only incorporate part of the concept or use it for only one fragment of the game. Juul and others argue that most games are a combination of *games of emergence* and *games of progression* [18], [19], e.g. using the former within a given game level and the latter in the firmly set sequence in which levels are presented [19]. Sweetser argues that a combination might even be desirable to achieve a desired sandbox-style play within the boundaries of predefined narrative [20]. Following this proposition, Multiplicative Design too can help designers to fulfill the potential of the whole game or only certain subsystems. An example for a modern game that makes use of such a combination is *Red Dead Redemption 2* (Rockstar Games, 2018) [21]. While it relies on story-based progression, it also features a large open world that can be freely explored and interacted with in between directed missions in arguably multiplicative ways. To ease argumentation in this thesis, I will mostly only differentiate between the two extremes - pure Multiplicative Gameplay or 'games of emergence' on one end of the spectrum, pure additive gameplay or 'games of progression' on the other - rather than take notice of the intermediate forms. Referring to the individual multiplicative parts or features of a game instead of the whole game is hence the most reasonable approach.

Arguing what game genre best fits the design has been proven to be inappropriate,

since on one hand most modern games do not fit in only one of the classic genres [22], and on the other hand many genres are fit to incorporate Multiplicative Design elements.

Characteristics of games of emergence or Multiplicative Gameplay have been present in video game design for some time now. Strategy games like *SimCity* (Maxis, 1989) [23] or *Shogun: Total War* (Creative Assembly, 2000) [24], stealth game *Metal Gear Solid* (Konami, 1998) [25] or action role-playing game *Deus Ex* (Ion Storm, 2000) [26] already included a lot of Multiplicative Gameplay rising from player freedom and an interacting and consistent system that enabled different game sessions. Nevertheless, in recent years the trend has shifted notably from a focus on more linear and progressive story-based games to those featuring big open worlds which most often incorporate Multiplicative Gameplay regarding player freedom, choice, and interactivity. However, most of these games, especially Triple-A titles, rely on massive content creation to fill their game worlds and do not explicitly focus on the creation of a relatively small set of objects to interact with. I will highlight these topics in the next section 3.2.

For now, the popular survival game *Minecraft* (Mojang Studios, 2011) [27] shall be presented as a good example for a contemporary game that features Multiplicative Gameplay. It fulfills all premises of the definition: Firstly, the world is highly interactive. The player is able to manipulate almost all elements existing in the game, and the elements themselves have many behavioral connections to other elements, e.g., when using the shovel item to remove a dirt block, or when merging elements in the crafting system to create new ones. Additionally, the game does not require many elements to provide a huge possibility space: only a few blocks and items enable many different play sessions. Further, the game makes extensive use of the concept of consistency. All wooden elements can be inflamed, water will extinguish all sources of flames, all blocks and items can only be placed into the grid of squares, every block of some kind of stone needs a pickaxe to be destroyed, etc. Therefore, the player perceives Minecraft's world as believable and logical. Lastly, the players possess vast freedom of agency.

In the following sections I will highlight these desirable principles of Multiplicative Gameplay and will name more examples of games with these features.

3.2 Why Multiplicative Gameplay is desirable

3.2.1 Cutting down on development resources

While this section mainly focusses on the positive effects on the player experience, this first argument sheds light on advantages from the developers' standpoint. Especially games that offer players a huge game world to be explored usually demand a lot of development resources. They rely on massive amounts of assets and must continuously provide interesting interactions and exploration. If a game like this fails to give enough

points of interest, players perceive its world as dead. Multiplicative Design creates gameplay from the interaction of a rather small number of different elements, which directly helps with this problem: developers can focus on creating a relatively small number of assets that are repeatedly placed into the world that provide a variety of gameplay through interaction; the principle quality over quantity is key. Instead of creating features that the player will only come across once or twice, they are encountered frequently over the course of the game. Harvey Smith, Lead Designer of *Deus Ex* (Ion Storm, 2000) [26], affirms this by saying that it is not possible to create rich interactions and emergent behavior by using additive design, because the creation of this level of detail is not manageable for the developer team [17]. An example for this advantage of Multiplicative Design from the game BotW is the design of the 120 shrines, which provide puzzles to be solved by the player. The designers created a relatively small set of repeating objects and mechanisms that have been arranged differently to create 120 unique riddles to be solved [9]. Other games have realized it in similar ways: *Assassin's Creed Odyssey* (Ubisoft, 2018) [28] provides its players with labyrinth-like underground tombs to be traversed, which, while providing noticeably different game experiences and riddles, consist of mostly the same objects that must be interacted with in order to reach the goal.

The limited number of connected objects and the coherent rules of the design simultaneously serve as another advantage: it does not only reduce complexity for developers, but also for the players. Developers are relieved of creating more and more elements to design and add to the game, and players are relieved from memorizing the behavior of a huge number of unrelated objects, which supports intuitive understanding and learning of the game [3], [29]. While in this way complexity is arguably reduced, Multiplicative Design might add complexity in other parts. I will elaborate on this in section 3.3.4.

3.2.2 Believability of the game world through consistency

One of the key concepts of Multiplicative Gameplay is the sense of a believable game world. In this context it is important to differentiate between believability and realism. In the mind of the player, a game that focuses on resembling reality in its graphics, mechanics, voice over, sound design etc. can feel just as believable as a 2-dimensional pixelated side-scroller. I describe believability as the unconscious conception of the game world in the mind of the player as in itself logic; as a world that can 'realistically' exist in the mind of the player within the boundaries of its own set of rules. Like reality has its consistent laws and rules to which everything must bow, the game world too must have consistency in its ruleset to be seen as a believable entity. If a game fails to convey believability to the player, the world feels staged, dead and 'unreal',

and the players do not build up what is often called a *suspension of disbelief* when playing the game [4], [5], [13], [30], which lets players intentionally suppress their awareness that the game is not real and instead makes them believe it for the sake of enjoyment. Salen and Zimmermann critique the often accepted view that players might forget that the game is artificial and actually believe that they are part of the imaginary world [13]. Instead, they argue that play is a process of metacommunication where the players take part in a "double-consciousness" and thus are aware of the artificiality of this reality. This is where I want to place the concept of believability. I argue that Multiplicative Design supports the rise of this player conception and does so by focusing on a consistent ruleset.

Consistency has often been named as the crucial factor for such believability of game worlds (see for example [4], [31]) and in turn inconsistency or contradiction can break believability. If the player encounters behavior which differs from what they experienced as true, it can be frustrating and break their understanding of how the world works. A recent example for the breakage of believability is found in the game *Cyberpunk 2077* (CD Projekt Red, 2020) [32]. Tall cactuses that can be encountered in the game world are easily destructible when driving over them with a car. Even at the lowest speed they will disassemble into little pieces while the car is barely slowed down in the process. This creates a certain picture in the mind of the player of how the game world works and along with it expectations for related behavior. Players get a notion that these cactuses are not robust and that they can easily be destroyed. The breakage of believability happens when the player tries to destroy them using weapons: they barely show damage to them, and even when using explosives, they cannot be destroyed.

The key to believability of a game world is, as mentioned, not to simulate cactuses and their behavior from the real world, but to keep a consistent logic in the virtual world. Players' expectations should be met in cases like these, even if it was unrealistic. The players are well aware of the artificiality and are willing to submit themselves to the artificial rules if they convey a consistent overall picture. Only then players know how the world reacts to their inputs, so that they are able to feel in control of their actions and develop strategies. It can even be beneficial for gameplay when the design actively differs from real world behavior and implement minor tweaks, or as Nintendo called it, clever lies, to meet the players' expectations and help them understand the world [9]. For example, in reality not everything that is made out of wood will easily catch fire, but incorporating it in the game might better meet players' expectations and will result in a more vivid play environment the player will understand as logic and interesting.

Consistency simultaneously acts as a help for players to learn and master the game, feel a sense of control and encourage to explore and experiment [3], [13]. Sellers argues

that "learning such a game is subjectively almost effortless, and each aspect learned adds to the player's progression and feeling of mastery, and the player does not have to mentally step out of playing the game to consider how to remember an exception or a seemingly contradictory rule" [29]. Consistency will cue the player what is possible or not, e.g. what looks the same has to act the same [4], [18]. An example that many games show in this regard is the different look of doors to buildings that cannot be entered in contrast to those that can be, e.g., by giving them knobs or not, or by showing buildings that can be entered as icons on the map. If the player could not tell the difference, it would be frustrating to try it every time; instead of control there might rise a feeling of powerlessness.

3.2.3 Focus on player freedom

The previous section has already touched on the concept of consistency and that it can encourage players to explore and experiment in the world. Multiplicative Design should support the feeling of curiosity in the player that is enabled one by the consistency of the game world ("If this is possible, is that possible too?"), and two by the freedom of choice it offers to the players.

Unlike movies or books, games are capable of giving players some degree of self-determination instead of having to fully submit to the prewritten sequence of events. It can be argued that for this reason games should make use of this possibility to create experiences other media forms cannot offer [4]. Although many games where the designers oriented themselves toward a movie format and thus feature a very directed and pre-designed experience are praised and loved by the audiences (see for example *Heavy Rain* (Quantic Dream, 2010) [33]), games that focus on freedom of choice can have many advantages.

First, I shortly want to add to the concept of consistency that can evoke curiosity. Because objects feature high interactivity and have coherent behavior, players are animated to experiment with their possibilities in the game world. It must be noted though that for some players the sheer possibilities might not be a motivation to do so. Bartle's taxonomy of player types for example tries to explain different preferences players have; curiosity as a motivation will, in his terms, mostly appeal to the type of the explorer. Järvinen claims that players always require clear goals in order to pursue actions [5], but Sicart opposes this and instead argues that some games encourage action without strict goals set by the game. He names the strategy game *SimCity* (Maxis, 1989) [23] as an example [6], which features multiplicative elements like freedom of choice and consistency. While some guidance by the system is advised for Multiplicative Gameplay (for this see section 5.1.5), I want to side with Sicart's stand: goals set by the system are not a requirement for play, and players can set their own goals to pursue or

take action based on motivations like curiosity. In these terms, I argue that consistency in the design will promote the motivation of curiosity by rewarding it: a feeling of satisfaction is evoked when players curiously try out an action in the game that the so far perceived logic suggested to them and they get a feedback from the system in the way hoped for (or in another interesting way). If instead nothing happened, the player freedom would have been limited and player's expectations would have been disappointed. Nintendo brought up a few examples from BotW: "So for example, this metal slab is placed here to allow you to cross this river. But once you pick it up, you might find yourself suddenly wanting to try dropping it on top of an enemy. [...] Once you see that log floating on that river, it's natural to want to try jumping on it. [...] Just riding a log down a river seems like it would be fun [...]. And if you can climb moving objects, what sort of battle would unfold if you were fighting a giant enemy that you could climb? [...] What would happen if you climb an object, that you charged up with energy using stasis?" [9]

Secondly, curiosity can also be evoked by freedom of choice. Play and curiosity are closely connected. Schell describes one possible definition of play as "manipulation that indulges curiosity" [4]; with other words: when you manipulate something out of curiosity you are playing. This definition implies that real play has the premise of willful agency. Games of progression then would be less play than games of emergence because they do not allow as much free and willful manipulation out of curiosity, but the reason for manipulation is rather to achieve a goal and to progress in the game. While Schell himself admits that this definition has flaws [4, p. 30], it hints to the important human need of autonomy. Autonomy has been named as one of three essential mental needs that humans must meet to remain mentally healthy [34], and is described as "I need freedom to do things my own way" by Schell [35]. Multiplicative Design aims to give players a vast possibility space through interactions of game elements, which on one hand makes it possible to solve problems in a variety of ways, and on the other hand to freely choose if a problem should even be solved or not. BotW left players the freedom to choose what to do next to the degree that the end-goal in the form of the final boss could be approached straight after the beginning sequence, leaving out all other main quests and side quests. This gives a game high variety in different gaming sessions: unlike strictly directed and linear games, Multiplicative Gameplay allows players to play the game in a different way every time and thus supports the replay value of a game. In addition, freedom of choice can be implemented in a way that allows players to develop and execute the strategy of their choice. Some players for example prefer a stealth tactic over an open confrontation; this of course depends heavily on the kind of game. If the strategy a player chose feels like it was their own idea and leads to a success, a feeling of accomplishment and autonomy is evoked in the player [9]. In many cases the designers have actively incorporated the possibility of

these strategies; it must be emphasized that the focus is not on the actual novelty of the player's ideas, but on the players' experience of autonomy within the boundaries of the rules.

3.3 Challenges of Multiplicative Design

3.3.1 Difficulty fine-tuning player experience

To a degree, giving players great freedom of action hands over designers' control to the players, i.e., designers do not have full creative control over the players' experience in Multiplicative Design. In contrast to directed linear games, the experience of Multiplicative Gameplay is a lot more difficult to fine-tune and foresee for designers [4], [9]. Getting insights through playtesting becomes trickier and takes more time, as the player's experience is unique to every play session, whereas directed games will act out almost the same regardless of the player. Adams and Dormans however claim that although player freedom enables variation, the gameplay will still often follow fairly regular patterns [19]. In contrast, Sicart argues that while formal understanding of player mechanics and thus players' possibilities to interact with the world allows developers to design and predict courses of interaction, it does neither determine how the game will always be played nor how the players' experience will turn out [6]. This is especially relevant for Multiplicative Gameplay.

Furthermore, there usually is a fundamental difference regarding the story construction between games with Multiplicative Design and games of progression (provided that the game focusses on story in the first place). Directed games often tell a scripted pre-written story to the players that will unfold as they progress in the game. Multiplicative Gameplay on the other hand lets players create their own story through gameplay and is often framed by a looser general narrative. In a GDC speech from 2004, Warren Spector, one of the lead developers behind *Deus Ex* (Ion Storm, 2000) [26] which is considered a milestone in video game history for the degree of freedom of the player, argued that such emergent narratives end up with a relative lack of direction and emotional resonance and provide fewer exciting and surprising moments [36]. While he does not deprive open-ended games of the ability to tell successful stories, he does not see their freedom of agency as an inherent advantage to storytelling. Developers of directed games are able to use well-proven dramaturgic methods that are common in movies, literature, theatre etc. to systematically achieve the emotional response that they are aiming for. Multiplicative Gameplay must therefore concentrate more on other means for emotional stimuli and on telling stories through mechanics, environments, aesthetics etc. to satisfy players' desire for emotional involvement. This is arguably harder to do than scripted storytelling and will result in an inherently different ex-

perience. For example, players of the more linear *The Legend of Zelda: Ocarina of Time* (Nintendo, 1998) [37] praise the game primarily for the emotional story it tells, whereas fans of *The Legend of Zelda: Breath of the Wild* (Nintendo, 2017) [8] rather emphasize the freedom and exploration it offers.

3.3.2 Consistency vs. unwanted outcomes

I have elaborated on the advantages of consistency in previous sections, but it is important to address that it also brings on challenges. Forcing consistency regardless of player experience can lead to a frustrating or outright broken game. An example is the game *Noita* (Nolla Games, 2020) [38]. It is a good example for thorough and sophisticated Multiplicative Design. It incorporates many mechanics through which the player can interact with the environment in consistent and interesting ways; the world is highly connected and provides open and emergent gameplay. While the vast majority of players praise the game as innovative, surprising and intelligent, it left some players frustrated [39]. This is caused by 'too much consistency': because the game world is highly manipulable and interconnected, situations emerge where effects and materials react to different inputs, sometimes in chain reactions, that the control is taken from the player and instead leaves him powerless in the hands of a chaotic system. Many players complain about a staggering difficulty because of surprising events they mostly have no chance to escape from. Some examples for such situations are: getting knocked into water that has an unforeseen electric item in it and getting electrified to death, getting killed by a piece of environment structure falling on top of their head or crafting a new magic wand with unforeseeable combinations of effects and accidentally killing themselves with it [40]. So, while well-made Multiplicative Design does bring forward novel, interesting and engaging gameplay, developers should always keep in mind how consistency can result in unwanted behavior of the system. In the case of Noita the developers were likely well aware of the challenging complexity of the game and embraced it to meet a target audience that is particularly interested in such challenges (again see for example Bartle's taxonomy of player types: achievers).

Sometimes however developers have reason to compromise in thorough consistency: for example, in a lot of games there are either no children at all or they are unable to be killed, in contrast to adult characters that can be. In this case developers had to compromise because of moral reasons. In other cases, developers decide that NPC main characters are immortal to prevent missions or the entire narrative to collapse. In contrast, the game *Westerado: Double Barreled* (Ostrich Banditos, 2016) [41] allows the killing of everyone without exception. Instead, the developers included extra behavior for some of these cases so that the game does not break. This is an example for a game where designers rejected creating additional rules or exceptions to rules to control the

player's experience - this is sometimes called *artificial balancing* [4, p. 197] - and instead achieved consistency in an elegant way.

In the context of consistency the term *ludonarrative dissonance* is often brought up: it describes the issue in games when the gameplay contradicts the story that is told. A popular example is from *Final Fantasy VII* (Square, 1997) [42]: using an item called "Phoenix Down" the character Aeris was able to always be brought back from the dead during the whole game, until the story progressed and the character dies permanently in a cutscene. This made players question why they were not able to bring her back this time. This caused the ludonarrative dissonance and consequently the loss of believability of the game world. The developers could have solved this problem by explaining the permanent death in the narrative part, e.g. by declaring her wound as too deep or the sword that pierced her as poisoned [43]. For Multiplicative Gameplay it must thus be remembered to not only keep consistency in the interactions, but also in the narrative that goes along with it. Sometimes, narrative can also be used to explain inconsistencies in order to keep up the believability of the game world.

Concluding, consistency and desired behavior stand in constant tradeoff during the design process of games, especially those featuring Multiplicative Gameplay. Sometimes it is desirable to make minor adjustments and compromise in total consistency to improve the player experience, and sometimes it can be solved in a way that keeps consistency and player's self-expression without breaking the game, or even embraces the behavior that results from it.

3.3.3 Implementing features the player might never discover

Trying to satisfy the concept of Multiplicative Gameplay will require developers to implement rules for the connections between game elements. Sometimes, it is obvious that a connection is important to make, while other times developers will question the importance of implementing it. Like with every game, development resources should mostly be concentrated on core concepts, main areas, and features that the players will most likely come across.

The opposite of this issue might be what is often called *negative possibility space*. It describes the issue that comes up when the game system does not give feedback when players expect it to; a possibility that was expected to be implemented but has not been realized. This could for example be in an open world game where players made their way up to the top of a mountain with the expectation to be rewarded in some way but are disappointed to find nothing. Consistency is relevant in this context too: if players are made to believe that the top of the mountain will hold something to discover, it is probably because similar locations did and thus trained players to expect the mountain to hold the same.

While this is an important issue to keep in mind for Multiplicative Gameplay, the counterpart is also not to be neglected: implementing something that the player will likely not come across. For Multiplicative Design this will mostly affect the 'multiplication' of elements and the following behavior of the interactions. When players do not encounter those anyway, it will neither result in more freedom of choice nor will it help their understanding of the world [4, p. 286]. One way to optimize the distribution of development resources is to flag features early on in the design process with levels of importance. A way to actualize this is proposed in section 5.1.1.

Multiplicative Design should help direct players away from negative space and instead to interesting interactions and discoveries (see section 5.1.5).

A vision for the future is to ideally implement Multiplicative Gameplay in a way that will only rarely open up the question of where to cut interactions, but to create systems that will automatically combine in a consistent way and thus will not require a lot of manual work.

3.3.4 Complexity

When talking about complexity, it is important to separate the perspectives of player and developer.

The one the player encounters is - fundamentally spoken - the mental burden that rises out of the game's complexness; it is the amount of "data the player has to store, the rules they have to process, and calculations they have to make to make a meaningful choice" [44]. In section 3.2.1 I argued that Multiplicative Design reduces players' learning curve, because rather than introducing a lot of separate functionalities (as in additive design), the behavior of the game is intuitively learned and understood in the process of interacting with it. But - as I pointed out in section 3.3.2 on the example of the game Noita - the connected game world can become highly unpredictable and complex due to the many inputs and outputs in the game world that can lead to chain reactions. While it is true that the learning of the game itself is eased, a rule of thumb could be that the higher the interactivity of the world and the more probable those interactions are, the more difficult the game becomes. BotW does not become as difficult and frustrating as Noita, because the emergent behavior does not result in unmanageable complexity. Both games feature precise implementation of Multiplicative Gameplay, but Noita takes it a level further by featuring not only high interactivity, but the appearance of those reactions have a much higher frequency, so that the player is not always able to properly react to them on time. BotW on the other hand features a more 'try and watch' kind of scenario, where the multiplicity is not the reason for difficulty but rather acts as a source for exploration and experimentation with the world. Therefore, designers must be aware of the kind of experience they want to create. The two examples show

that Multiplicative Gameplay can be used to create very different levels of complexity for the player which result in distinct experiences of difficulty and speed of a game.

Additionally, the freedom of choice offered during Multiplicative Gameplay can put players into a state of indecisiveness or even paralysis. When a player does not receive enough guidance by the system, they might be overwhelmed by the amount of different choices they could make and strategies they might try. Warren Spector called this phenomenon "tyranny of choice" [36]. Understanding of the game helps with this issue, since uncertainty of the effect of their actions and of their possibilities puts players into this state in the first place. A good implementation of consistency will support players' understanding of the game world, but developers must make sure that players are made familiar with the systems to avoid disorientation. Section 5.1.5 will feature methods designers can use to help players find their way through the game.

Apart from the players, developers too must face complexity during Multiplicative Design. For them it concerns the complexness of interactions between the game elements from which the gameplay is supposed to emerge. I explained in section 3.2.1 how Multiplicative Design helps to reduce development resources, since the gameplay rises not out of the sheer amount of elements that feature unique behavior, but from the level of their connectedness. However, realizing these connections can become quite complex as well. The focus regarding complexity hence shifts to the implementation of the interactions of elements and the behavior that results from the multiplicity, but also requires designing a solid concept for the game elements themselves that must be suitable for rich interactivity. Additionally, the possibility space that rises from a high variety of interactions must carefully be observed during the development process. With every new behavior that is implemented the space of possible gameplay is enlarged, so that predicting how the game and the player will behave becomes more complex for the developers. Besides the desired emergent gameplay, there could also arise behavior that will lead away from the direction developers are aiming towards. It can become quite complex to decide which elements to add, which to remove, and in which ways elements should interact to obtain satisfying gameplay. It can therefore take more time and be more laborious to achieve a desirable outcome with Multiplicative Design. Additionally, managing this system can become confusing and obscure. With the concept that I will introduce in chapter 4 I aim to support this process by enabling designers to keep an overview over their elements and the multiplicity they provide. However, even with this support the design can quickly become very large and complex. Complexity should therefore always be kept manageable by reviewing the current scale of the Multiplicative Design regularly during the process and fine-tuning it to match the available development resources.

4 Method: The matrix concept and the SPO framework

4.1 The matrix concept

In this section I will introduce the concept of matrices that use the terminology of natural language to act as a practical help for designers during the process of Multiplicative Design. The abbreviation SPO stands for subject-predicate-object. While I will already use the terms *SPO*, *scenario* and *game mechanic* here, I will explain them in more detail in the subsequent section 4.2.

4.1.1 Purpose and applicability

The matrices I will introduce here aim to serve three major purposes. Firstly, they support designers in creating games with Multiplicative Gameplay by assisting them with the connectedness of elements, the consistency of their behavior and creation of new elements. Secondly, they provide a graphical overview over the game elements and their interactivity. Lastly, they enable feedback about the current design based on the data written into the matrices.

The matrix and the SPO concept can additionally support the communication between designers and programmers. I will elaborate on this in section 4.2.3.

It must be noted that the outcome of the process described in the next section and therefore the look of the matrices depends heavily on the game and the way designers choose to execute the steps. E.g., in the example I purposely disregarded interactions that consist of sole collisions out of which results no special behavior, as this is something a physics engine could handle. It would be valid to also include these, or to do a second set of matrices that only target physics between game objects. This shows that designers have scope in deciding how to execute the following steps. The matrices by one designer will therefore likely look different from those of another. This is owed to the use of natural language, which is not strictly determined but rather intuitive.

4.1.2 The concept presented on an example

I will show the process of working with the matrix concept with the help of an imaginary design process of a game that should feature Multiplicative Gameplay.

The designers might have a first vision of the game: a single-player game where the player controls the hero Darius, which is a humanoid character that can move through an open world. The goal of the game should be to collect and return ten figurines stolen from the sanctum of a peaceful village that are now hidden all over the map. To reach them, the player must traverse the world and overcome its obstacles. These obstacles can be enemies that are in your way or hindrances in the environment. There are items that can be found in the world and which can be picked up by either the player or the enemy NPCs. Some first elements that can be encountered in the world the designers agreed upon are: rocks, trees and rivers, and items are sword, bow, arrows, bombs. Darius cannot swim but is able to lift up smaller rocks.

Even with a rudimentary first concept like this the matrix can already be applied to support Multiplicative Design early on. The steps on how designers could proceed are the following:

1. Create a list of all physical elements that are currently present in the game.

This encompasses everything that can interact or be interacted with in some way. They do not need to be categorized or in a certain order.

In the made-up example game this would be the player character, enemies, items, objects, and environment features like the river. The list could look like this:

Darius, enemy, figurine, smaller rock, tree, river, sword, bow, arrow, bomb

2. Think of a scenario that could appear in the game.

A scenario is a situation that could appear in the game. To make up a scenario, mainly think of what actions are available to the player and what they can manipulate in the world. Take the list of step 1 for help. Most scenarios rise from the use of a game mechanic by the player. I will define the term *game mechanic* and explain the concept in more detail in section 4.2. For this step the scenario can be formulated as an informal sentence, as bullet points or as it would be said in spoken language. This is where the advantage of the concept of natural language already takes action.

The designers of our example game could think of an instance where the player makes Darius throw a bomb into a tree which causes it to get destroyed.

3. Determine which parts of the scenario make up sets of subject, predicate, and object.

Now the scenario "Darius throws a bomb into a tree which causes it to get destroyed" is rephrased so that it holds the basic form subject-predicate-object. If the scenario comes from a player action, i.e. a mechanic, it is likely that it produces two or more sets of SPOs, since oftentimes the first set relates to the player using the mechanic, and the other sets to the consequences of this action in the game world. To differentiate the parts of the sentence correctly, ask these questions:

- For subjects and objects: Who or what does something (subject) with what (object)?
- For predicates: What does the subject do?

Applied to the example, we receive two SPO sets:

- Darius throws bomb.
- Bomb destroys tree.

Now it is advisable to repeat steps 2 and 3 until a good number of scenarios is found and formalized into sets of SPOs. By doing so, designers can use the list to rethink and refactor predicates to describe and concentrate sets. Other than subjects and objects, predicates are highly flexible and informal; they depend heavily on the designers' choices. Different predicates can lead to very different matrices and subsequently to different design. To help with this issue, a general master-matrix (as explained later in step 5) will provide additional help to optimize the definition of predicates.

4. Create a matrix for every predicate and find more subjects and objects.

Every predicate will now act as a pivotal point for behavior. For every predicate, a new matrix is constructed where the corresponding subjects are laid out horizontally and objects vertically.

Table 4.1 shows the two matrices that must be created from the two SPO sets from step 3. I will refer to these as scenario-based matrices from now on. The entries where subjects and objects 'cross' provide space to be filled with the behavior taking place in case of interaction. These behaviors represent the rules that need to be implemented by the programmers. For "Bomb destroys tree" the designers might decide to not destroy it completely but turn the tree into a log that can have further interactions. This means that *log* must be added to the list of step 1:

O \ S	Darius
bomb	activates bomb; only if Darius has ≥ 1 bomb in the inventory; subtract 1 from number of bombs in inventory

O \ S	bomb
tree	tree turns into rigid body log

Table 4.1: Matrices for predicates *throws* (left) and *destroys* (right)

Darius, enemy, figurine, smaller rock, tree, river, sword, bow, arrow, bomb, log

Next, designers need to ask themselves what other elements can act as subject or object in combination with the predicate of a matrix. To do this, traverse the list from step 1 and insert every entry as subject into the SPO set and check if this is a behavior that must either be added to keep consistency or could be interesting behavior. Then do the same for objects. Add those that suit either as subjects or objects to the corresponding predicate matrix. Apart from the element list, also think of other elements that are meaningful in the context of a predicate and add them to the list.

Let's look at the *throws* matrix and see which subjects can be added. *Darius*, the first element of the list, is already the subject. Next is *enemy*: "Enemy throws bomb". This can be added to the matrix since developers wanted enemies to be able to use items the same way a player does. The rest of the entries do not fit the predicate as a subject; however, during this step designers might get another idea: the *river* does not throw bombs, but what if it could transport floating bombs? This would make up a new predicate matrix: "River transports bomb". Getting back to our *throws* matrix, we want to add objects too: *Darius* is first on the list: "Darius throws Darius" is pointless, so it is disregarded. Next is "Darius throws enemy". This might not have crossed the mind of the developers yet, but opens up new interesting gameplay possibilities, so they might decide to add it to the table. Because we added enemy as a subject too, possible behavior for "Enemy throws enemy" also emerges, and "Enemy throws Darius" is considered to be relevant too. Next is *figurine*; this too is well possible. After completing the process of adding objects, the same steps are repeated for the *destroys* matrix and the new *transports* matrix.

When it comes to checking "Bomb destroys figurine" designers might be unsure and the question arises out of what material it is made. If it is out of wood, consistency demands destructibility, but in this case the player would be unable to collect it, which would corrupt the game's goal. Then they might decide to try and make the figurine

respawn at its original position when destroyed and explain it by making the figurines magical. This adds another dimension to the game. The same question arises for "Bomb destroys sword/bow", so designers might decide to introduce the differentiation of materials for weapons, which would further enhance multiplicity: everything that is made of wood will float on water and get destroyed by bombs and metal swords. Maybe they also burn when inflamed; maybe introduce fire? Maybe also feature rivers of lava? This shows that the process will not only specify elements further, but also fuel creativity. However, I will not realize further ideas like fire here to keep it simple. With the new differentiation of materials, the new list might look like this:

Darius, enemy, figurine, smaller rock, tree, river, wooden sword, metal sword, wooden bow, metal bow, arrow, bomb, log

A possbile layout for the extended matrices is shown in table 4.2. As seen in the new matrices, not every interaction must be annotated with a rule, since some predicates already describe the wanted behavior. Other interactions might not be desired and can be flagged with a keyword or symbol like '/' by the designers. E.g., "Metal sword destroys smaller rock" is actively disregarded. In this case, the exception occurred because subjects with slightly different preferred functionality like *bomb* and *metal sword* are thrown into the same predicate matrix. Because of this reason, designers might want to seperate such matrices by using different predicates. In this case though it is reasonable to keep them together in the *destroys* matrix, because it shows all destroyable objects together. However, the choice is up to the designers.

It must be noted here that traversing the list one element after another in order to fill the scenario-matrices can become quite laborious, especially for big game projects. In most cases designers will already think of their game elements as part of certain categories, e.g., items, agents, environment elements, weapons, etc. Respectively assigning properties like "destroyable" to elements of a certain category of e.g. environment elements can be of enormous help in these cases. Nevertheless, the method of traversing the list can have advantages. If designers do not yet have ordered categories, they can be built from the predicate-matrices. For example, the *destroys* matrix of table 4.1 defines that element *tree* is in the category "destroyable". Further, traversing all elements will ensure that no element is missed, and that consistency is not broken. Designers must therefore decide if a bottom-up design as explained here or the top-down method is most suitable for their project. For further thoughts on categories like these see section 5.1.2.

5. Create master-matrix for overview.

After designers created a first set of scenario-based matrices, it is very helpful to create a 'master-matrix' that shows the whole current concept. It helps to keep track of

O \ S	Darius	enemy
bomb	activates bomb; only if Darius has ≥ 1 bomb in the inventory; subtract 1 from number of bombs in inventory	activates bomb; enemies can only throw bombs they picked up in the environment
Darius	/	enemies can load Darius on their backs and throw him
enemy	Darius can load them on his back and throw them	enemies can load each other on their backs and throw them in Darius' direction
figurine		
smaller rock		
wooden sword		
metal sword		
wooden bow		
metal bow		

O \ S	bomb	metal sword	wooden sword	smaller rock	arrow	log
tree	tree turns into rigid body log; respectively: instant, 5 hits, 8 hits			/	/	/
figurine	figurine respawns at original point; respectively: instant, 5 hits		/	/	/	/
smaller rock		/	/	/	/	/
wooden sword		5 hits	8 hits	/	/	/
wooden bow		5 hits	8 hits	/	/	/
arrow		2 hits	3 hits	when thrown	when shot	when fallen on arrow
bomb	activates other bomb; chain reaction!	activates bomb				
log		5 hits	8 hits	/	/	/

Table 4.2: New matrices for predicates *throws* (upper) and *destroys* (lower)

S / O	Darius	enemy	figurine	smaller rock	tree	river	wooden sword	metal sword	wooden bow	metal bow	arrow	bomb	log
Darius		throws											
enemy	throws	throws											
figurine	throws	throws						de-stroys				de-stroys	
smaller rock	throws	throws										de-stroys	
tree							de-stroys	de-stroys				de-stroys	
river													
wooden sword	throws	throws					de-stroys	de-stroys				de-stroys	
metal sword	throws	throws											
wooden bow	throws	throws					de-stroys	de-stroys				de-stroys	
metal bow	throws	throws											
arrow				de-stroys			de-stroys	de-stroys			de-stroys	de-stroys	de-stroys
bomb	throws	throws		de-stroys			de-stroys	de-stroys			de-stroys	de-stroys	de-stroys
log							de-stroys	de-stroys				de-stroys	

Table 4.3: Master-matrix for example game at current state (with the two scenario matrices *throws* and *destroys*)

connections between elements and thus supports fine-tuning and correcting the concept, while getting an idea of the current amount of multiplicity the concept provides. This master-matrix is created by listing all elements that are currently in the list as subjects as well as as objects. Instead of creating matrices for every predicate, this matrix holds the predicates that connect every subject and every object as entries. It is helpful to fill it out parallel to the process described in steps 1 to 4 to always keep an overview over the current design. Both matrix concepts are compatible, which means that scenario-based predicate-matrices can directly be refactored into the master-matrix and vice versa. However, because the list of elements can get quite long during the design process, it can be valuable to not only create a matrix for the whole game, but for only parts of it. Here, categories of game elements are again crucial.

A possible representation of such a master-matrix of the current state of our example game is shown in table 4.3.

Table 4.4 shows the master-matrix with more possible filled entries. Here I marked some entries with '/', which again can be used by designers to signal that no connection is intended, whereas empty fields can still be filled out. In section 3.3.2 I already described that designers must sometimes compromise in consistency for the sake of

O \ S	Darius	enemy	figurine	smaller rock	tree	river	wooden sword	metal sword	wooden bow	metal bow	arrow	bomb	log
Darius	/	throws, punches, carries, picks up, puts down	hits	hits		drowns	injures	injures	hits	hits	injures	injures, kicks	hits, kicks
enemy	throws, punches, carries, picks up, puts down	throws, punches, carries, picks up, puts down	hits	hits		drowns	injures	injures	hits	hits	injures	injures, kicks	hits, kicks
figurine	throws, carries, picks up, puts down	throws, carries, picks up, puts down	/			trans-ports	/	de-stroys			sticks in	de-stroys	kicks
smaller rock	throws, carries, picks up, puts down	throws, carries, picks up, puts down				drowns	kicks	kicks				de-stroys	kicks
tree	climbs on	climbs on				/	de-stroys	de-stroys			sticks in	de-stroys	/
river	falls into	falls into	falls into	falls into		/	falls into	falls into	falls into	falls into	falls into	falls into	falls into
wooden sword	throws, carries, picks up, puts down, uses	throws, carries, picks up, puts down, uses				trans-ports	de-stroys	de-stroys				de-stroys	kicks
metal sword	throws, carries, picks up, puts down, uses	throws, carries, picks up, puts down, uses				drowns		kicks				kicks	kicks
wooden bow	throws, carries, picks up, puts down, uses	throws, carries, picks up, puts down, uses				trans-ports	de-stroys	de-stroys				de-stroys	kicks
metal bow	throws, carries, picks up, puts down, uses	throws, carries, picks up, puts down, uses				drowns		kicks				kicks	kicks
arrow	collects, climbs on	collects		de-stroys		trans-ports	de-stroys	de-stroys	shoots	shoots	de-stroys	de-stroys	de-stroys
bomb	throws, carries, picks up, puts down, collects	throws, carries, picks up, puts down		de-stroys		trans-ports	de-stroys	de-stroys			de-stroys	de-stroys	de-stroys
log	rolls	rolls			turns into	trans-ports	de-stroys	de-stroys			sticks in	de-stroys	kicks

Table 4.4: A possible way to further fill the master-matrix

the game's progression or of moral reasons. These could then result in '/'-entries in the master-matrix. Additionally, such omitted fields can result from interactions that are simply not logical, e.g. the interaction "Darius throws Darius" in table 4.2 could be marked by designers for this reason. The '/'-symbol also shows up in the master-matrix 4.4 for this S*O connection, as designers could have decided that there is also no other predicate that makes sense here. Another reason could be that the intended behavior is to have no interaction. In table 4.4 this is the case for the subject *wooden sword* and the object *figurine*, while the interaction "metal sword destroys figurine" exists. This adds to the distinction of the two weapons in order to give more meaningful choice to the player.

All these reasons are examples for different motivations for intentional omissions of behavior. This method helps to achieve a controlled game design: even when fewer entries are filled, the design can still be clearly understood later, even better when the reasons for each '/'-entry are kept. These reasons could for example be saved in a database. This helps to prevent extra work that results from not knowing if behavior in this area is still desired or not, and changes are easier to be made when the reason for the omission is known. This way it is thinkable to systematically check all matrix cells, so that in the end all entries are either filled with predicates or marked with '/' to which the different reasons are attached. This could prevent accidentally forgotten behavior that could lead to inconsistencies.

4.2 Game mechanics, scenarios and the SPO framework

Now that the matrix concept has been presented, I want to further elaborate on the theory behind it. First, I will discuss the definition of game mechanics. This serves as a preface for the concept of scenarios, which are the basis for the matrix design of section 4.1. Next, the terms subject, predicate, and object are discussed (sometimes called SPO in the following), and how this usage of natural language can help with game design.

4.2.1 Game mechanics

The term *game mechanic* is commonly used by researchers, designers and laymen when describing the functionality of a game. Oftentimes though, its precise meaning is either not clear or has been defined differently. In 2008, Miguel Sicart published a discussion and comparison between different definitions and also proposed his own [6]. For this thesis I want to shortly touch on different understandings of the term and then give reason why I will use Sicart's definition as it proves most fitting for my approach of Multiplicative Design and the theory of the design matrices in section 4.1.

Literature about the topic is mostly congruent in the statement that mechanics are - or at least encompass - either actions, operations or interactions initiated by the players or by the game system that lead to a change in the game (see for example [4], [5], [45]–[47]). A fundamental difference in the definition of mechanics, as Sicart points out [6], is the separation of the concepts of *rules* and *mechanics*. While some authors do not draw a line between the two (see for example [19], [45]), others acknowledge a difference and rather see them as two important parts of a game that influence each other. Adams and Rollings describe rules as basic instructions that the game must follow, e.g. "Caterpillars move faster than snails", whereas mechanics specify them accurately in a way that can be turned into code by the programmers, thus they define exactly how fast caterpillars and snails move in the game [22]. Järvinen approaches the difference of rules and mechanics differently: he argues that mechanics are individual play performances that are defined and regulated by rules in order for the game system to function [5]. In similar terms, Sicart proposes the following definition: "game mechanics are methods invoked by agents, designed for interaction with the game state" [6]. This definition suits my approach because of two reasons: First, it states that mechanics are actions taken by agents, with agents being both human and artificial. This harmonizes with what I call a *scenario*: an occurring of a sequence of events triggered by a mechanic. This fits the general idea of a game that 'waits' for players' agency: most games generally react to players' actions in an event-and-feedback manner. Thus, the definition of a mechanic as the essential event helps to name and describe this mechanism. Secondly, Sicart's definition implies a difference between rules and mechanics: while mechanics are the actions, the rules "provide the possibility space where that interaction is possible, regulating as well the transition between states. In this sense, rules are modeled after agency, while mechanics are modeled for agency" [6]. This illustrates the differentiation that is made in terms of the scenario-based SPO matrices: game mechanics are formulated into predicates that make up a matrix, while rules are the entries of the matrix that result from the triggered mechanic and describe the behavior of the S*P*O interactions. Following Sicart, these rules either define under which conditions a predicate takes action (see for example entry "only if Darius has ≥ 1 bomb in the inventory" for "Darius throws bomb" in table 4.1) or they regulate the transitions between states (see for example entry "tree turns into rigid body log" for "bomb destroys tree" in table 4.1). In other cases rules can specify the interactions and provide more detailed information for the behavior (see for example entry "enemies can load each other on their backs and throw them in Darius' direction" for "enemy throws enemy" in table 4.2).

4.2.2 Subjects, predicates and objects

I already introduced the SPO concept during section 4.1 and showed why it is useful for Multiplicative Design. On the example it was demonstrated how game mechanics get translated into predicates in the process of designing. This formulation of game mechanics as verbs has often been proposed in literature: Järvinen suggested to name each individual game mechanic according to the action it represents, e.g. "'manipulating', 'choosing', 'trading', etc." [5, p. 39]. He also states that mechanics usually act on something "as, e.g., moving pieces and building houses in Monopoly demonstrates" [5, p. 66]. This hints to a categorization of game components into subjects and objects in order to complement predicates and to form a coherent natural language concept. Sicart agrees with Järvinen's suggestion and further describes how rules and other syntactical elements have influence on how those verbs act in the game [6]. Schell uses the term game mechanics in a more general and inclusive way, and instead refers to the subject matter as *actions* [4]. These actions are similarly defined as Sicart's mechanics, and Schell also proposes to put them as verbs. Additionally, he uses the terms subject and object in combination with verbs: subjects are those that can be controlled by players, and thus those that can execute mechanics or actions; objects are those that the verbs act on [4, p. 143]. Nintendo's description of Multiplicative Gameplay in BotW involves similar terms. According to them, the interactions take place in the 'multiplication' space of player actions * play field * objects [9]. Player actions are thus the mechanics, and there is a distinction between play field and objects for what these mechanics act on.

This shows that the approach of using this kind of natural language in game design is not novel. However, integrating this approach into the actual design process and using it to develop a practical design method has not yet been anticipated to my knowledge.

In order to do so, and specifically to support the design process of Multiplicative Gameplay, I extend these existing approaches by widening the definitions of predicates and subjects: while all mechanics are predicates, not all predicates are mechanics; and while all agents are subjects, not all subjects are agents. This new definition allows to really capture a game's course of events: mechanics are, as described above, the trigger for events, but the actual behavior of a game appears as feedback to these. To be able to design Multiplicative Gameplay it proved to be useful to not only 'multiply' mechanics with the game's subjects and objects, but to also fragment all possible consequences of them into own sets of SPOs. This way coherency of behavior can be ensured in a much more thorough way. Let's take the example scenario "the player makes Darius throw a bomb into a tree which causes it to get destroyed" from section 4.1.2. The mechanic here is *throw* because this is the player's action. But when only mechanics can be predicates, where are the effects and the consequences of the action recorded? They

O \ S	Darius

bomb	only if Darius has ≥ 1 bomb in the inventory;
	subtract 1 from number of bombs in inventory;
	if (bomb hits tree), then turn tree into rigid body log;
	if (bomb hits figurine), then make it spawn again at original point;
	if (bomb hits bomb), then chain reaction!;
	if (bomb hits smaller rock || wooden sword || wooden bow || arrow || log), then destroy it

Table 4.5: Matrix for predicate *throws* if only mechanics were turned into predicates

could be turned into a rule and thus be written into the entry of the matrix, but by doing so all possible behaviors that could result from the SPO "Darius throws bomb" must be listed, which can result in long and rather unformalized descriptions (see table 4.5).

The example shows that instead of creating another whole predicate matrix for the verb *destroys*, it had to be integrated into the *throws* matrix. Additionally, behaviors like "metal sword destroys tree" would either be neglected or would have to be somehow included into a mechanic-SPO like "Darius uses metal sword", which again requires to list all possible interactions of the predicate *uses* in the matrix entry space. This approach leads to confusing arrangements and fails to reflect the multiplicative character that is to be achieved during game design. In other words, instead of only considering actions from the player agent (or an artificial one/NPC), the 'actions' executed by the game system are treated as equals (e.g. predicate *destroys*). This also delineates the expanded notion of the term *subject*: by declaring game system procedures as actions too, other game components apart from agents can also act as the subject (e.g. bomb, metal sword), which supports to reach the described aim of encompassing all game situations in order to ensure Multiplicative Gameplay.

Further, it is immensely useful to keep a link between the mechanic that made up a scenario and the SPO sets that stem from it. Let's again look at the example scenario from section 4.1.2: "the player makes Darius throw a bomb into a tree which causes it to get destroyed". This makes up two sets of SPOs: *Darius throws bomb* and *Bomb destroys tree*. The first one includes the mechanic predicate that triggers the second one. Keeping a connection between the two can have several advantages: first, it enables the development team to keep track of what a single mechanic can provoke in the game. This is important considering that the level of control the designers have over a non-directed game is relatively low. It can also be attempted to read these links out of

the matrices by matching up subjects and objects into a sequence of SPOs, e.g., "Darius throws bomb, bomb destroys tree, tree turns into log, log floats on river, river transports log, log is rigid body and thus Darius can stand on it". However, this method does not guarantee logical sequences per se, but needs the designer to manually read them out in an order that makes sense. Therefore, it is advisable to construct these links during the first steps of the procedure. Apart from this, creating the links helps to keep track of which implementations must be fixed in case a mechanic is deleted or altered. Another use case is to see how extensive a mechanic is. During the design steps described earlier it can occur that a single mechanic becomes very voluminous in terms of the amount of SPOs it generates; with a link designers can consider if this is something they want to keep. These links could be kept in some kind of database. A database is also useful for keeping all tuples of (S, P, O) in addition to their storage in the matrices.

4.2.3 SPO and matrix concept as interface between designers and programmers

The matrices and the classification of elements into subjects, predicates, and objects can support communication between designers and programmers during the development process. The design process results in already formalized SPO constructs that are both easily understandable and structured to allow easier translation into code. Thereby it can act as an interface by relieving programmers of interpreting the designers' vision: instead of reading game elements out of a text, elements are already categorized into subjects including agents, predicates including game mechanics, other distinct game elements and rules. The design process of the matrices could in this sense be seen as programming on a high level of abstraction. Sicart argues that his definition of game mechanics serves as an easy way for programmers to transcribe their behavior into an object-oriented programming framework: mechanics defined as "methods invoked by agents" can often directly be implemented as methods of classes, and rules as properties [6]. Following his proposal, it is thinkable to implement all subjects and objects as classes, and all predicates as methods of their according subject-classes that act on other classes. A huge advantage could be achieved when predicates are programmed in a universal way that can be applied to every subject- and object-class. The methods could then be reused instead of having to be implemented for each individual class that features this predicate. Multiplicative Design has therefore potential to support the implementation process of Multiplicative Gameplay in a powerful way. This way, a concept for multiplicative programming could be derived. Nevertheless, if this method is applicable needs to be tried and proved.

Although the proposed matrix and SPO concept holds potential for future design

processes regarding the communication in the development team, it is possible that there are other more effective methods suitable for a design as anticipated here that are already better explored and from which my approach can learn. For example, a well-known and established method for software development is *agile software development* in combination with *user stories*. Both have parallels to my design approach and could help to further enhance it. Agile development proposes the division of tasks into stand-alone workloads and thus features that can work independently from each other. Applied to the design matrix this could mean to build sets of SPOs in a way that enables their implementation without requiring the implementation of another. Furthermore, agile development welcomes changing requirements, even in late development [48]. With an independence of SPO sets, changes made in the design matrix can ideally be adjusted in code in the same way. This would further enhance the communication between designers and programmers, even after implementation has been started and progressed. The concept of user stories has similarities to the SPO and scenario construct as well. User stories too are coming from a scenario-like standpoint and are then rephrased into fairly strict formulations of natural language like "*As a* registered user - *I want to* change my password - *So I can* keep my account secure". This is comparable to the forming of structured sentences of the form subject-predicate-object resulting from an informal scenario. This helps relieving programmers of the interpretation of the designer's vision.

5 Guidelines and evaluation methods

5.1 Guidelines

In the following I want to propose some guidelines to orientate oneself in order to achieve Multiplicative Gameplay. First, I will give some more advice for the creation of the design matrices. Then I will close with some additional notes that are relevant for designers when attempting this kind of design.

5.1.1 Filling out many entries and setting different priorities

Generally, designers should work towards a master-matrix that has as many entries filled with predicates as possible. This will support high interactivity and a vivid game world. But, as I have mentioned in section 3.3.3, there might be interactions that are not as prevalent in gameplay as others; these are the ones the designers could decide on being of second order priority. To signal this to the programmers, they could flag these entries in some way, e.g. by coloring the entries in the graphical representation of the master-matrix. Likewise, very important interactions that are often encountered in the game or are part of the core behavior of the game can be flagged as important. Consequently, designers are able to introduce as much or as little gradation of importance as appropriate. Further, it is advisable to differentiate between agent and non-agent subjects. Entries in the columns of agent subjects, i.e., those that feature game mechanics as predicates, should generally have a higher priority to be filled in than those of non-agent subjects. This will ensure that the game mechanics the player can choose from can be executed on most objects in the game world, which will then potentially trigger further behavior. Moreover, entries that are intentionally left empty (see section 4.1.2) can be flagged as such. These have been marked with '/' in table 4.4.

In table 5.1 I show a possible weighting of entries of the example master-matrix. Here I use three different weights: Light grey entries are considered to be important, whereas dark grey entries are considered to be rather unimportant. Those that are left white are of average priority.

This weighting of features has potential to support several areas of the development process: the team knows on which features to work first, and if development resources

O \ S	Darius	enemy	figurine	smaller rock	tree	river	wooden sword	metal sword	wooden bow	metal bow	arrow	bomb	log
Darius	/	throws, punches, carries, picks up, puts down	hits	hits		drowns	injures	injures	hits	hits	injures	injures, kicks	hits, kicks
enemy	throws, punches, carries, picks up, puts down	throws, punches, carries, picks up, puts down	hits	hits		drowns	injures	injures	hits	hits	injures	injures, kicks	hits, kicks
figurine	throws, carries, picks up, puts down	throws, carries, picks up, puts down	/			trans-ports	/	de-stroys			sticks in	de-stroys	kicks
smaller rock	throws, carries, picks up, puts down	throws, carries, picks up, puts down				drowns	kicks	kicks				de-stroys	kicks
tree	climbs on	climbs on				/	de-stroys	de-stroys			sticks in	de-stroys	/
river	falls into	falls into	falls into	falls into		/	falls into	falls into	falls into	falls into	falls into	falls into	falls into
wooden sword	throws, carries, picks up, puts down, uses	throws, carries, picks up, puts down, uses				trans-ports	de-stroys	de-stroys				de-stroys	kicks
metal sword	throws, carries, picks up, puts down, uses	throws, carries, picks up, puts down, uses				drowns		kicks				kicks	kicks
wooden bow	throws, carries, picks up, puts down, uses	throws, carries, picks up, puts down, uses				trans-ports	de-stroys	de-stroys				de-stroys	kicks
metal bow	throws, carries, picks up, puts down, uses	throws, carries, picks up, puts down, uses				drowns		kicks				kicks	kicks
arrow	collects, climbs on	collects		de-stroys		trans-ports	de-stroys	de-stroys	shoots	shoots	de-stroys	de-stroys	de-stroys
bomb	throws, carries, picks up, puts down, collects	throws, carries, picks up, puts down		de-stroys		trans-ports	de-stroys	de-stroys			de-stroys	de-stroys	de-stroys
log	rolls	rolls			turns into	trans-ports	de-stroys	de-stroys			sticks in	de-stroys	kicks

Table 5.1: A possible weighting of the master-matrix

run low, it is easier to decide which features to neglect. If there is highly defective behavior it could be decided on if it should either be removed completely or fixed based on the priority of the interaction. Furthermore, it could help to decide on which areas to focus on during playtesting sessions.

5.1.2 Keeping consistency

Besides, during the design process designers should always have the player's perception of the game world in mind to keep track of the consistency in the design. For Multiplicative Gameplay consistency is the key to achieve a believable game world, the arousal of curiosity and the intuitive understanding of the world by the player (see section 3.2). To keep consistency during the process described above, designers must be aware of the commonalities of their game elements. The example game concept for instance features wooden swords and wooden bows. If wooden swords can be destroyed by bombs, then wooden bows should be too to ensure the consistent conception of the world in the player's mind. While the design steps are already modeled in a way that will generate consistency to a large extent, it is advisable to create categories of game elements that share the same properties. Even though in most cases designers will already think of their game components as part of categories (items, agents, environment features etc.), the master-matrix can easily bring forth those categories and help to further enhance already made assumptions: from table 4.3 it can easily be deduced that every subject that has the predicate *throws* is an agent; every subject that has the predicate *destroys* is an offensive item/weapon; every object that has the predicate *destroys* is either a wooden item or a destroyable environment object. By constantly checking if a predicate is missing from an entry for the sake of consistency, holes that accidentally occurred during the design steps can be filled. This helps to sort elements into categories and assign to them properties that must be met for all items in this category. This again acts as a bridge between designers and programmers, as programmers can, for example, use inheritance or interfaces - depending on the programming environment - to facilitate the management of consistency.

Another important notice to make for keeping consistency is to know which sets of SPOs are dependent on another, so that when deleting a subject, object or predicate, the developers know what else must be deleted to keep the game free of bugs. As touched on in section 4.2.2, it is therefore helpful to keep a database that keeps the connections between scenarios and resulting sets of SPOs. This way, the link between game mechanics and SPO sets that rise from them is also saved. In case of a deletion, developers can check which sets of SPOs are concerned with it and can easily determine which features have to be removed as well.

5.1.3 Cultivating emergence

Further, the SPO concept in combination with the matrices can support the arising of emergence in the game. I have already described the concept of emergence in games and how Multiplicative Gameplay enables its arising in section 3.1.2. Schell proposed several tips based on his verb-subject-object approach that are meant to higher the probability of emergent behavior in a game [4, p. 140-144]. As I mentioned in section 4.2.2, Schell uses the literal term *actions* instead of *game mechanics* in order to describe what players can do in the game world. He further differentiates between *operative actions* and *resultant actions*. The former refers to the 'classic' ways a player can act on the world; they come near to depicting what I defined as game mechanics and are expressed as verbs. The latter is described as the emergent behavior that rises from the way players use the operative actions. Therefore, they are situated in a metaphysical level above the operative actions, and Schell describes them as often being strategic moves. He uses the board game Checkers as an example: According to his theory, there are only three operative actions: "Move a checker forward", "Jump an opponent's checker" and "Move a checker backwards (kings only)", while there are a lot more resultant actions that rise out them, e.g. "Protect a checker from being captured by moving another checker behind it" or "Sacrifice a checker to trick his opponent" [4, p. 140-141]. Schell proposes that "the ratio of meaningful resultant actions to operation-oriented actions is a good measure of how much emergent behavior your game features" [4, p. 141], i.e. the more resultant actions a few operative ones put forth, the better.

Tying Schell's theory into my concept, it is to be noted that these resultant actions are not depicted in the matrices, and it is neither primitive to derive them from those. It is, as I have explained earlier in section 3.1.2, owed to the complexity of games and the nature of emergence that makes it - for now - impossible to be engineered straight forward. However, the probability of the occurrence of emergence can be increased. Schell's terminology fits my framework well and his tips for emergence are highly relevant here. Resultant actions, as Schell describes, "appear when operative actions [= verbs] interact with each other, with objects, and with the game space" [4, p. 141]. In my terms, emergence appears when predicates are combined with each other and with subjects and objects. Even though Schell restricts predicates to players' actions, his tips are still applicable.

His first tip is to add more verbs, i.e., more operative actions. For my concept, this means to add more game mechanics and thus more possible actions for players to execute in the game world. If the players have more variety of action, it is more likely that emergent behavior will appear, as they can combine these actions, and, in the optimal case, can execute these predicates on many objects in the game world.

Following this logic, I argue that not only more game mechanics are desirable, but in general more predicates, to operate on their own or as a causal chain to penetrate emergent behavior. Applied to the matrix concept this means that more predicates make up more scenario-based matrices and therefore more behavior, which leads to more filled entries in the master-matrix.

However, as Schell emphasizes, simply adding more verbs is not everything. This leads over to his second tip: Verbs should act on many objects. Schell is referring to more appearances of the same verb, but because my matrix will in most cases lead to heavily broken-down instances of actions, verbs are likely to differ even though they refer to the same chain of actions, e.g., "Darius uses wooden bow, wooden bow shoots arrow, arrow injures enemy". Schell would only use one verb for this, *shoot* [4, p. 142], and this verb would then act on different objects. In my matrix tip two would generally mean to achieve as many filled entries per subject as possible. In the example, this means to create as many interactions as possible for the subject *arrow*, mostly independent of the predicates itself. Regardless, it should still be avoided to create a lot of predicates that can only act on one or two objects; this would lead into the direction of additive design. Using the master-matrix is a good way to check if predicates could be merged or swapped to decrease their number. Tip two therefore helps with emergence by again increasing the general number of possible interactions and therefore the probability of resultant actions.

Tip three concerns the goals of the game and the way players approach them. I will discuss this in section 5.1.5.

Tip four states to create many subjects. For Schell this means entities that the player controls. For most games, this encompasses only one player character, others feature different characters to choose from, and some do not include a character at all, but let players control various non-character entities (e.g. strategy games like the SimCity or Total War series). Schell argues that by providing more subjects that can execute different actions under different constraints and that can interact with one another, more resultant actions are likely to appear. My design enables basically all game elements to become subjects. Likewise, the more subjects, i.e., the more elements can execute behavior, the more interactions are possible and thus more emergent behavior is likely to appear. Therefore, it should be aimed at giving objects behavior to turn them into subjects.

5.1.4 Depth vs. complexity

A concept that is closely related to the previous section 5.1.3 about emergence is the one of depth. While others have talked about similar concepts (see for example [5], [29], [35]) I would like to discuss it in terms of how it was explained in a YouTube video

on the channel *Extra Credits* [44], as the definition made there is well applicable in the context of Multiplicative Design and the proposed matrix system, and thus brings forth practical help for the design process. In the video, it is also attempted to explain a relationship between depth and complexity.

Depth is described as "the number of emergent experientially different possibilities or meaningful choices that come out of one rule set". They give two prerequisites that must be met by designers in order to achieve depth in a game: "the player's ability to make meaningful choices with the options the rules present to them and their ability to learn from the outcomes of those choices". This definition fits Schell's *resultant actions*: while *operative actions* are the basic predicates, resultant actions are what a player does when combining predicates, subjects and objects in a strategic way in order to reach a goal [4]. These resultant actions are the different meaningful choices a player can make in a game which rise out of the rule set. Since they often are experiential, players learn from their choices if the outcome is consistent. As I argued before, emergence and thus resultant actions cannot be primitively estimated, but must be proven by playtesting. However, with the help of the master-matrix and the scenarios that it rose from, several resultant actions can already be derived. This is another advantage of scenario-based design: it creates scenarios imagined by humans to create behavior, which makes it already more likely to find strategies that could take place in the game and therefore likely to help with the finding of resultant actions. Regardless, the practical use of found resultant actions and in which ways players discover and execute them must always be proven by playing the actual game.

Similar to Schell's examples of resultant actions in the game Checkers given in the previous chapter, it is possible to read such resultant actions from the master-matrix of the example game (see table 5.1). For example, to reach the goal of overcoming enemies in the environment (maybe in order to reach a figurine), there are a "number of emergent experientially different possibilities or meaningful choices that come out of one rule set" [44] to complete this task:

- pick up and throw enemy into river to drown
- pick up and throw enemy onto another enemy
- punch enemy to death
- pick up and carry enemy away (throw into cliff?)
- throw figurine/smaller rock/wooden bow/metal bow to hit enemy and make him unconscious for a while
- kill enemy with swords or bow and arrow

- throw smaller stone from hideout to lead enemies out of the way
- chop tree and let log fall onto enemy
- kill enemy by throwing bomb
- throw bomb beside enemies to kick them into river
- put down bombs near the enemies and shoot them with arrows to make them explode
- roll log from hill to kill several enemies at once
- bomb a group of trees to create many logs to roll down the hill and kill enemies
- roll log into river to stand on and move past the enemy group
- roll log into river to stand on and shoot arrows from the safety of the river
- climb on tree (taller trees might need arrows shot into it to climb) and shoot enemies from up there
- lay a path of bombs and activate one to kill enemies in chain reaction of bombs
- and more

Most of these can be estimated by looking at the master-matrix in combination with the scenarios: the first one for example consists of the following SPO sets: "Darius picks up enemy", "Darius carries enemy", "Darius throws enemy", "river drowns enemy".

ExtraCredits also relates the term of complexity to depth: as already described in section 3.3.4, complexity to the player corresponds to the mental burden the game puts on the player, i.e. "data the player has to store, the rules they have to process, and calculations they have to make to make a meaningful choice" [44]. In relation to depth this means to "get the maximum amount of depth out of the minimum amount of complexity". This again is closely related to the concept of elegance which I touched on in section 3.1.2 [44]. ExtraCredits describes that depth can be "bought" with complexity, as more subjects, predicates and objects that interact with each other and thus create more complexity are likely to also increase the depth of the gameplay. At the same time, complexity is said to be able to limit depth: when a game is so complex that a player does not understand it and thus does not discover the resultant actions, depth is actively decreased from the player [44]. To achieve a high depth to complexity ratio with the help of the matrix concept, designers can get a reference point by calculating a ratio by dividing the number of found resultant actions for a specific goal by the number

of mechanics available to the player (operative actions). This correlates to Schell's proposal of a high ratio of resultant actions to operative actions [4]. This demands to first list as many resultant actions for a goal as they can by looking at their matrices and confirming them by playtesting. This ratio, again, is not formal as the design of the predicates is rather intuitive, but the number can serve as a good reference point for depth. In these terms, this ratio could arguably also serve as an indicator for emergence in regard to a specific goal. In the example, there are sixteen resultant actions listed above, and there are ten mechanics available to the player: throws, punches, carries, picks up, puts down, climbs on, falls into, uses, collects, and rolls. So the ratio for this goal would be $17/10 = 1.7$. This means that this set of mechanics produces 1.7 times as much resultant actions in the context of the proposed goal of overcoming enemies. The more resultant actions are found, the higher the number for the depth becomes and the better the depth turns out to be.

The concept of depth is of particular importance for Multiplicative Design, as it focuses on the gameplay that takes place in the space rising from the interaction of game elements. Players should be able to freely choose between many different meaningful options in this possibility space to reach a goal. Thus, the higher the depth, the better, as it supports player freedom and emergence in the game. To cultivate more depth, which means more resultant actions, designers should keep a goal in mind and think about which interactions of elements and which mechanics can add more meaningful choice for the player to reach it.

Thinking of strategies in relation to a goal was also proposed by Nintendo [9] (see again section 3.1.1). Following their advise, designers should make sure to intentionally design situations to solve in interesting ways, and to always plan on at least one 'sample solution'.

5.1.5 Guiding the player and preventing dominant strategies

Tying to the previous section, when talking about meaningful choice it is not only important to enable different possibilities of reaching a goal, but also to make sure that players are actually taking advantage of the possibility space created for them. An issue that is frequently occurring in this context are *dominant strategies*, as Schell calls them [4]. Those are resultant actions that are more efficient or easier than others, or they are reliant to always lead to the desired outcome without needing to take risks. This results in players ignoring other options of reaching a goal once they discovered the dominant strategy and pursuing only this one strategy for the rest of the game [4]. In this case meaningful choice and depth is actively eliminated from the game by the players. This is often called an exploit. Juul also acknowledged this issue in his paper about the difference between games of emergence and game of progression: "If the game allows

for a good strategy that leads to interesting interaction, it is a good game. If the optimal strategy for playing the game leads to dull game sessions, the game will be considered dull" [1]. Hence, the existence of a dominant strategy does not always mean the game is poorly designed. As long as the play sessions stay interesting, players are free to play the game the way they like, which is a goal of Multiplicative Design. Nevertheless, the forming of dominant strategies that block out all other possible resultant actions can degrade the play experience. This issue is especially challenging for designers that work on Multiplicative Gameplay, as great player freedom heavily contributes to the issue.

Every game that lets players freedom of choice and development of own strategies needs some guidance by the system to ensure that players will engage with the possibilities the game provides. Schell describes that it is important to balance between freedom and a controlled experience in order to create a better play experience: he argues that in some cases more control can result in a better game than constant player freedom [4]. Schell proposes different methods of *indirect control*, which are ways of guiding the player to do the things the designers intend them to do in order to provide to them interesting play sessions. Instead of taking the freedom of choice from them, the player is encouraged to explore what the game has to offer. The first method is described as constraining the player without them feeling a loss in freedom by giving them only a limited number of different options to choose from [4, p. 285-286]. For Multiplicative Gameplay and our example game this can mean to only place a certain set of game elements that can be interacted with into one area at once to make players try out other strategies without taking the choice from them. His second method advises to set clear goals to the player so that they feel motivated to perform a certain task [4, p. 286]. Other methods include the help of the interface, the visual design, the game characters and the music to achieve player guidance [4]. Schell's methods help to both avoid dominant strategies and to guide players to points of interest. Another, more challenging approach focusing on the avoidance of a dominant strategy is to balance all possible strategies so that they are equally attractive to the player. If they are in balance or not can again only be identified through playtesting. This method of balancing however is arguably very difficult to do. Another idea is to let the game give rewards to players for performing different strategies. E.g., in *Far Cry 5* (Ubisoft, 2018) [49] players get more points for conquering a camp if they are not being discovered by enemies, which leads players to approach their goal carefully and strategically instead of the easier variant of simply using the grenade launcher in open confrontation.

Generally, players should not be forced to use a certain way of solving a problem, achieving a goal, or exploring the world, as this would go against Multiplicative Gameplay (see section 3.2.3). Instead, the game should loosely guide the player to provide them with a more engaging play experience than what it would have been if

the player had not been guided.

5.2 Evaluation methods

Arranging a game's design semiformally into matrices like these provides the advantage of enabling either humans or computers to make statements about it. In the following I present various approaches I attempted which are supposed to show ideas for how my matrices could be evaluated. These evaluations are the basis for giving feedback to the designers to support them in the realization of Multiplicative Design. The approaches are not meant to state if a design is good or bad, but rather to call the designers' attention to different aspects of their current design. Ideally, these are realized in code to enable the computer to perform the calculations and to simultaneously give designers computer-assisted feedback about the matrix during the design process. However, for now they are described informally so that designers from all backgrounds are able to manually apply them themselves.

5.2.1 Basic statements

Following section 5.1.1, the most intuitive approach is to give feedback on how many entries in the master-matrix are filled, and of which priority those filled and empty are. There can be made several statements that although very simple can give practical hints to the designers.

The most basic one is telling which percentage of the matrix is filled out. For this there can be made a difference between empty entries that are just not filled and those that are intentionally left blank. In the example table 5.1 I have marked those with the symbol '/'. Thus, the percentages are as follows:

- Filled entries in relation to all cells:
  $\frac{109}{13 \cdot 13} = \frac{109}{169} \approx 64.5\%$
- Filled entries plus those intentionally left blank in relation to all cells:
  $\frac{109+6}{13 \cdot 13} = \frac{115}{169} \approx 68.0\%$

Secondly, it is useful to know which subjects and objects have the most entries to get an overview of which ones are good and which ones need more work. Here I treated entries marked with '/' as filled entries, because those are already handled by the designers. The application to the example is shown in table 5.2.

Now it can be easily seen that e.g. *tree* has almost no behavior as a subject, whereas turned into a log it does have all entries filled. Similarly, the two bows are residing at

Subjects descending:

- bomb: $13/13 = 100\%$
- Darius: $13/13 = 100\%$
- enemy: $13/13 = 100\%$
- log: $13/13 = 100\%$
- metal sword: $13/13 = 100\%$
- river: $13/13 = 100\%$
- wooden sword: $11/13 \approx 84.6\%$
- arrow: $8/13 \approx 61.5\%$
- smaller rock: $5/13 \approx 38.5\%$
- figurine: $4/13 \approx 30.8\%$
- metal bow: $4/13 \approx 30.8\%$
- wooden bow: $4/13 \approx 30.8\%$
- tree: $1/13 \approx 7.7\%$

Objects descending:

- Darius: $12/13 \approx 92.3\%$
- enemy: $12/13 \approx 92.3\%$
- river: $12/13 \approx 92.3\%$
- arrow: $11/13 \approx 84.6\%$
- bomb: $9/13 \approx 69.2\%$
- figurine: $9/13 \approx 69.2\%$
- log: $9/13 \approx 69.2\%$
- tree: $8/13 \approx 61.5\%$
- smaller rock: $7/13 \approx 53.8\%$
- wooden bow: $7/13 \approx 53.8\%$
- wooden sword: $7/13 \approx 53.8\%$
- metal bow: $6/13 \approx 46.2\%$
- metal sword: $6/13 \approx 46.2\%$

Table 5.2: Percentages of filled entries of each subject and object

Subjects descending:

- bomb: $13/13 = 100\%$
- Darius: $13/13 = 100\%$
- enemy: $13/13 = 100\%$
- metal sword: $13/13 = 100\%$
- river: $13/13 = 100\%$
- tree + log: $13/13 = 100\%$
- wooden sword: $11/13 \approx 84.6\%$
- metal bow + arrow: $8/13 \approx 61.5\%$
- wooden bow + arrow: $8/13 \approx 61.5\%$
- smaller rock: $5/13 \approx 38.5\%$
- figurine: $4/13 \approx 30.8\%$

Objects descending:

- Darius: $12/13 \approx 92.3\%$
- enemy: $12/13 \approx 92.3\%$
- river: $12/13 \approx 92.3\%$
- wooden bow + arrow: $11/13 \approx 84.6\%$
- metal bow + arrow: $11/13 \approx 84.6\%$
- bomb: $9/13 \approx 69.2\%$
- figurine: $9/13 \approx 69.2\%$
- tree + log: $9/13 \approx 69.2\%$
- smaller rock: $7/13 \approx 53.8\%$
- wooden sword: $7/13 \approx 53.8\%$
- metal sword: $6/13 \approx 46.2\%$

Table 5.3: Percentages of filled entries of each subject and object with merged elements

the bottom of the lists, but the arrow has quite good interactivity. This shows that it can be useful to merge them for these statistics, as they belong together in behavior. This could remodel the list as shown in table 5.3.

Another way to handle this issue is to mark all cells that are empty for one but filled for another element that is connected to the first with '/', so that both would get the same percentage; e.g. for subject tree there is no entry for object Darius, but there is one for subject log; this cell would then have to be marked for subject tree.

Moreover, it can be practical to incorporate the priority weightings into these numbers. One way to do this is to set up different weights relating to the different priorities and to again create a ratio. The weighting of the master-matrix of table 5.1 has a gradation of three levels: important entries are marked by a light grey tone, unimportant entries by a dark grey tone. The rest of the fields should remain in a neutral weight. Thus, the following weights could be proposed:

- important/strong entries (light grey): 1.5
- medium/average entries, '/'-entries and empty ones (white): 1

- unimportant/weak entries (dark grey): 0.5

For the calculation I will neglect the merging of associated game elements as I did in table 5.3 and examine them individually. I will take the subject *wooden bow* to demonstrate the weighting calculation: the subject has two unimportant, one neutral, and one important entry, as well as nine empty ones. This results in the following ratio of weighted entries to the sum of all weights of the column:
$\frac{2 \cdot 0.5 + 1 \cdot 1 + 1 \cdot 1.5}{2 \cdot 0.5 + 1 \cdot 1 + 1 \cdot 1.5 + 9 \cdot 1} = \frac{3.5}{3.5+9} = \frac{3.5}{12.5} = 28\%$
This percentage can directly be compared to the unweighted percentage of table 5.2 which has been 30.8%. Because there are more unimportant entries than important ones, the percentage became smaller. This shows the designers where to work on stronger predicates. The application of this calculation to the whole example is shown in table 5.4. Now it can be seen for example that the subject figurine fell two places in the ranking as its predicates are rather weak.

The weighting and the merging can also be combined to see relevant statistics. Generally, low percentages are not meant to signal that it is compelling for the designer to work more in this section, but to show that there is potentially more room for rich interaction. If the designers decide that there should intentionally not be any behavior in a cell, they can add a '/' to higher the percentage. This way, it is possible to fill entries until all subjects and objects have 100%. Designers are free to use this method in the way they prefer. For this again see the last two paragraphs of section 4.1.2.

Another interesting take on looking at the master-matrix is to evaluate the difference between game elements regarding predicates. This can help to decide if a subject/object brings additional value to the design as a whole, and if it could possibly add meaningful choice. In the example it can easily be seen that the two subjects *wooden bow* and *metal bow* have the same entries and the same priorities, but as objects the two have slightly different behavior. To ease the discovery of subjects/objects that are much alike it is again useful to describe these deviations as percentages. The difference between the two objects *wooden bow* and *metal bow* is thus $\frac{4}{13} \approx 30.8\%$, as the entries only differentiate in their interactions with the four subjects *river*, *wooden sword*, *metal sword* and *bomb*. Their resemblance is therefore $100\% - 30.8\% = 69.2\%$ as objects, and 100% as subjects, which gives them an average total of 84.6% resemblance. This is relatively high, so designers could, based on this feedback, investigate these components to check if it is reasonable to keep both of them. Here, incorporating priorities can again help with the decision.

In case of a very large master-matrix, isolating certain categories of elements can again be very helpful for evaluations like these. For example, only giving feedback on all weapons can shed more light onto the design in this specific area. Further, removing parts from the matrix, e.g., disregarding interactions between environment elements,

Subjects descending:

- bomb: $16.5/16.5 = 100\%$
- Darius: $19/19 = 100\%$
- enemy: $19.5/19.5 = 100\%$
- log: $13/13 = 100\%$
- metal sword: $15.5/15.5 = 100\%$
- river: $19/19 = 100\%$
- wooden sword: $12.5/14.5 \approx 86.2\%$
- arrow: $8.5/13.5 \approx 63\%$
- smaller rock: $5/13 \approx 38.5\%$
- metal bow: $3.5/12.5 = 28\%$
- wooden bow: $3.5/12.5 = 28\%$
- figurine: $3/12 = 25\%$
- tree: $1.5/13.5 \approx 11.1\%$

Objects descending:

- enemy: $14.5/15.5 \approx 93.5\%$
- Darius: $14/15 \approx 93.3\%$
- river: $14/15 \approx 93.3\%$
- arrow: $14.5/16.5 \approx 87.9\%$
- bomb: $13/17 \approx 76.5\%$
- log: $11/15 \approx 73.3\%$
- figurine: $10/14 \approx 71.4\%$
- tree: $9.5/14.5 \approx 65.5\%$
- wooden bow: $8.5/14.5 \approx 58.6\%$
- wooden sword: $8.5/14.5 \approx 58.6\%$
- smaller rock: $8/14 \approx 57.1\%$
- metal bow: $6/13 \approx 46.2\%$
- metal sword: $6/13 \approx 46.2\%$

Table 5.4: Percentages of filled entries of each subject and object with incorporated weightings

can also give additional insights on the remaining parts when evaluating.

5.2.2 Information content and entropy

Another idea to make a quantitative statement about the design is to use the concepts of *information content* and *entropy* from information theory and transfer them to the matrix. In its original context, the information content I_x basically states how probable the occurrence of a particular event x is, and how much information an event provides. When the scalar value I_x of an event is zero, the probability of the occurrence is 100% and it provides no information. The greater the value becomes, the less probable the occurrence is, and the more information it provides upon occurrence. In the context of the matrix however, we do not deal with probabilities, but with certainties. Still, the application is possible and arguably useful. In the matrix, each predicate represents a different event for which a value for the information content can be calculated. Thus, the lower the information content of a predicate, the more often it occurs in the matrix, and vice versa. The general formula for the information content is $I_x = -\log_a(p_x)$. For the base a of the logarithm I insert the number of different predicates, as it refers to the cardinality of the alphabet. p_x is the probability of the occurrence of an event x. In this context, an event is a predicate. Taking again the example matrix from table 5.1, the information contents of the predicates, the '/'-entries and the empty fields are to be calculated. For the example, base a is set as 21, as there are 19 different predicates, the symbol '/' and empty fields. For the probabilities, the number of occurrences of a predicate (or of '/' and empty fields) is counted and divided by the sum of all predicate occurrences plus those empty and marked with '/'. This is necessary as there is more than one predicate in some cells; otherwise the denominator would be the number of all cells, 169. The calculation then looks like this:

- $I_{empty} = -\log_{21}(\frac{54}{237}) \approx 0.49$
- $I_{destroys} = -\log_{21}(\frac{27}{237}) \approx 0.71$
- $I_{throws} = I_{carries} = I_{picks\ up} = I_{puts\ down} = I_{kicks} = -\log_{21}(\frac{17}{237}) \approx 0.87$
- $I_{falls\ into} = -\log_{21}(\frac{11}{237}) \approx 1.01$
- $I_{hits} = -\log_{21}(\frac{10}{237}) \approx 1.04$
- $I_{uses} = I_{injures} = -\log_{21}(\frac{8}{237}) \approx 1.11$
- $I_{transports} = I_{/} = -\log_{21}(\frac{6}{237}) \approx 1.21$
- $I_{drowns} = -\log_{21}(\frac{5}{237}) \approx 1.27$

- $I_{punches} = I_{climbs\ on} = I_{collects} = I_{sticks\ in} = -\log_{21}(\frac{3}{237}) \approx 1.44$
- $I_{rolls} = I_{shoots} = -\log_{21}(\frac{2}{237}) \approx 1.57$
- $I_{turns\ into} = -\log_{21}(\frac{1}{237}) \approx 1.80$

As it can be seen here, the higher the number of occurrences of an instance is, the lower the information content becomes. These statistics can help to give feedback about the use of predicates to the designers. By showing how many times a predicate has been used, it can tell them which ones are strong and which ones are likely to be very particular to a subject or an object.

With the use of the information content the entropy $H(X)$ can be calculated. In information theory, entropy describes the average information content. In a wider sense, entropy is often described as the measure of disorder. The value of entropy resides between zero and one. When entropy equals zero, the information is completely ordered as there is only one kind of information, i.e., the next event can always be predicted. When it is one, the entropy is at maximum, which means that the source has the highest level of disorder and the probabilities of all possible events are equally high. The general formula for entropy is $H(X) = \sum_{x \in X} p_x I_x$. Applied to the example matrix, the entropy calculation looks like this:

$$\begin{aligned} H(X) \approx & \frac{54}{237} \cdot 0.49 + \frac{27}{237} \cdot 0.71 + 5 \cdot \frac{17}{237} \cdot 0.87 + \frac{11}{237} \cdot 1.01 + \frac{10}{237} \cdot 1.04 + 2 \cdot \frac{8}{237} \cdot 1.11 + \\ & 2 \cdot \frac{6}{237} \cdot 1.21 + \frac{5}{237} \cdot 1.27 + 4 \cdot \frac{3}{237} \cdot 1.44 + 2 \cdot \frac{2}{237} \cdot 1.57 + \frac{1}{237} \cdot 1.80 \\ \approx & \ 0.858 \end{aligned}$$

The entropy of the example matrix of 0.858 is thus quite high. This indicates a relatively even distribution of different predicates and empty fields in the matrix. However, calculating entropy for a smaller set of SPOs can have more informative value than for a large matrix. Looking at a part of the matrix, e.g., at a category of certain subjects or objects that possess a common attribute like being an item, a weapon, an agent etc., can give more insight.

The calculation of the information content and the entropy for only the two subjects *wooden bow* and *metal bow* does therefore look like this:

- $I_{empty} = -\log_4(\frac{18}{26}) \approx 0.27$
- $I_{hits} = -\log_4(\frac{4}{26}) \approx 1.35$
- $I_{falls\ into} = I_{shoots} = -\log_4(\frac{2}{26}) \approx 1.85$

- $H(X) \approx \frac{18}{26} \cdot 0.27 + \frac{4}{26} \cdot 1.35 + \frac{2}{26} \cdot 1.85 \cdot 2 \approx 0.679$

The value is quite low, which shows that there probably is one dominating entry, in this case it's the empty ones. This can already be seen by the information contents, where I_{empty} is much lower than the others. This can tell designers that for the two bow subjects there is no even distribution and quite a lot of empty space.

Again, this kind of calculation makes most sense if the computer calculated these values. Instead of showing the numbers to the designer, the system would draw conclusions like "In this area there is a lot of white space/there is one dominating predicate/there is an even distribution" and show them to the designers.

5.2.3 Other considerations

An observation that has been made during research was that the master-matrices feature certain similarities in their entries. They can be found when comparing e.g. subjects to each other, objects to each other, categories of subjects or objects to each other or matrices to each other. In the previous sections 5.2.1 and 5.2.2 I have shown some rather primitive attempts at determining these properties. It is thinkable however to apply more advanced methods in order to characterize these similarity patterns and to quantify their complexity. An approach has been made using different descriptions of dimension to determine how much similarity a matrix possesses and thus how complex a matrix is. However, these approaches have not been continued.

6 Discussion and conclusion

In this thesis I first described the concept of Multiplicative Gameplay. For this, I first discussed its publicity through Nintendo's *The Legend of Zelda: Breath of the Wild* and how they described the concept. Then I outlined the better-known theory of emergence in games and its proximity to Multiplicative Gameplay. Lastly, I defined the term in the following way: "Multiplicative gameplay rises out of the interactions of relatively few well-connected game elements which provide players with a vast possibility space; it takes place in a coherent game world that the players perceive as believable and logical in itself and which evokes an experience of freedom of agency". Next, I discussed the advantages and the challenges of the design concept. After this first theoretical part, I proposed the method of natural language using the sentence parts subject, predicate, and object, which in combination with a presented scenario-based matrix creation provides designers with an intuitive framework for creating Multiplicative Gameplay. In the end, I additionally provided further practical guidelines for designers and proposed different methods of evaluating the design matrices in order to give feedback about them.

Regarding Multiplicative Gameplay, I believe that the concept is valuable for video games of every genre. Its principles have the potential to enhance certain parts of games or to give a framework for the whole concept. Games of additive design are not inherently worse and can have strengths that pure Multiplicative Design can only hardly keep up with, like engineering a fine-tuned emotion-evoking story. Therefore, Multiplicative Design should only be applied where it fits. Combining multiplicative elements with additive ones is in my opinion the ideal choice for a lot of games, especially for those of genres like adventure, action, or RPG. Further, Multiplicative Design is not a new idea; rather it is a combination of already known game design principles into one coherent concept. Nonetheless, I hope that research like this gives direction for future game concepts, as current titles are increasingly featuring vast player freedom in often massive open worlds, which is still a challenge to master.

I believe that my matrix-based design approach using natural language can act as valuable initiative to further explore more formal and guided game design techniques and to push more standardized concepts supporting game development. However, the matrix and natural language concept has potential to be further developed. The momentary stage still depends heavily on human consideration. Formalizing the

procedure more and providing models for orientation can help designers with the process. Furthermore, setting up the system as a computer program will bring major improvement to the applicability of the concept, as manual handling can become highly laborious: changes made to the scenario-based matrices, the master-matrix, the list of elements, or the databases must be propagated to all other instances. Additionally, the method is prone to produce a large matrix quite quickly, which makes a computer handling the system crucial to keep everything synchronized, to draw the matrix, and to calculate feedback. Moreover, the methods of possible evaluations of the matrices are also in need of further development, so that they can produce better feedback on the design.

To prove the usefulness of the design process presented here, the concept should be reviewed by experienced game designers and used to develop an experimental game. This way, the benefit of the system could be confirmed, its weaknesses decreased, and the concept further formalized and developed to ensure that it matches the requirements of a real game development process.

List of Figures

List of Tables

Bibliography

[1] J. Juul, "The Open and the Closed: Games of Emergence and Games of Progression," in *Computer Games and Digital Cultures Conference Proceedings*, F. Mäyrä, Ed., Tampere: Tampere University Press, 2002, pp. 323–329. [Online]. Available: https://www.jesperjuul.net/text/openandtheclosed.html (visited on 03/30/2021).

[2] J. Soler-Adillon, "The Open, the Closed and the Emergent: Theorizing Emergence for Videogame Studies," *Game Studies*, vol. 19, no. 2, 2019. [Online]. Available: http://gamestudies.org/1902/articles/soleradillon (visited on 05/05/2021).

[3] J. Dormans, *Engineering Emergence: Applied Theory for Game Design*. Amsterdam: Universiteit van Amsterdam, 2012, ISBN: 9789461907523. [Online]. Available: https://research.hva.nl/en/publications/engineering-emergence-applied-theory-for-game-design.

[4] J. Schell, *The Art of Game Design: A Book of Lenses*, First Edition. San Francisco: CRC Press, 2008, ISBN: 978-0123694966.

[5] A. Järvinen, *Games without Frontiers: Theories and Methods for Game Studies and Design*. Tampere University Press, 2008, ISBN: 9514472527.

[6] M. Sicart, "Defining Game Mechanics," *Game Studies*, vol. 8, no. 2, 2008. [Online]. Available: http://gamestudies.org/0802/articles/sicart?viewType=Print&viewClass=Print (visited on 03/02/2021).

[7] S. Björk, S. Lundgren, and J. Holopainen, "Game Design Patterns," in *Level Up: Digital Games Research Conference*, M. Copier and J. Raessens, Eds., 2003.

[9] Nintendo, *Breaking Conventions with The Legend of Zelda: Breath of the Wild*, 2017. [Online]. Available: https://www.gdcvault.com/play/1024562/Change-and-Constant-Breaking-Conventions.

[10] P. A. Corning, "The Re–emergence of "Emergence": A Venerable Concept in Search of a Theory," *Complexity*, vol. 7, no. 6, pp. 18–30, 2002, ISSN: 1076-2787. [Online]. Available: https://onlinelibrary.wiley.com/doi/epdf/10.1002/cplx.10043.

[11] J. H. Holland, *Emergence: From Chaos to Order*. Oxford University Press, 2000, ISBN: 0192862111.

[12] S. Kauffman, *At Home in the Universe: The Search for the Laws of Self-Organization and Complexity*. Oxford University Press, 1996, ISBN: 0195111303.

[13] K. Salen and E. Zimmerman, *Rules of Play: Game Design Fundamentals*. MIT Press, 2003, ISBN: 0262240459.

[14] Z. Hiwiller, *The Game Designer's Playlist: Innovative Games Every Game Designer Needs to Play*. Boston: Addison-Wesley, 2019, ISBN: 9780134873329.

[16] M. Salmond, *Video Game Design: Principles and Practices from the Ground Up*. Bloomsbury Publishing, 2016, ISBN: 1472567498.

[17] H. Smith, *The Future of Game Design: Moving Beyond Deus Ex and Other Dated Paradigms*, 2001. [Online]. Available: http://www.witchboy.net/articles/the-future-of-game-design-moving-beyond-deus-ex-and-other-dated-paradigms/ (visited on 08/06/2021).

[18] J. Juul, *Half-Real: Video Games between Real Rules and Fictional Worlds*. MIT Press, 2005, ISBN: 9780262516518.

[19] E. Adams and J. Dormans, *Game Mechanics: Advanced Game Design*. New Riders, 2012, ISBN: 0321820274.

[20] P. Sweetser, *An Emergent Approach to Game Design: Development and Play*. University of Queensland, 2005.

[22] E. Adams, *Fundamentals Of Game Design*, Third Edition, ser. Voices That Matter. New Riders, 2013, ISBN: 0321929675.

[29] M. Sellers, *Advanced Game Design: A Systems Approach*. Addison-Wesley Professional, 2017, ISBN: 0134669452.

[30] A. Rollings and E. Adams, *Andrew Rollings and Ernest Adams on Game Design*. New Riders, 2003, ISBN: 1592730019.

[31] S. Poole, *Trigger Happy: Videogames and the Entertainment Revolution*. New York: Arcade Publishing, 2004, ISBN: 1559705981.

[34] E. L. Deci and R. M. Ryan, "Self-Determination Theory," in *Handbook of Theories of Social Psychology: Volume 1*, P. A. M. van Lange, A. W. Kruglanski, and E. T. Higgins, Eds., London: SAGE Publications Ltd, 2011, ISBN: 9781847875143. DOI: 10.4135/9781446249215.

[35] J. Schell, *The Art of Game Design: A Book of Lenses*, 3rd edition. Boca Raton: CRC Press, 2019, ISBN: 1138632090.

[36] T. McNamara, *GDC 2004: Warren Spector Talks Games Narrative*, 2004. [Online]. Available: https://www.ign.com/articles/2004/03/27/gdc-2004-warren-spector-talks-games-narrative (visited on 04/12/2021).

[39] *Noita Steam Reviews*. [Online]. Available: https://store.steampowered.com/app/881100/Noita/#app_reviews_hash (visited on 08/08/2021).

[40] McQueeb, *Noita: Unbelievable deaths and funny moments // McQueeb*, YouTube, 2020. [Online]. Available: https://www.youtube.com/watch?v=lcnXVw7Phs0.

[43] E. Adams, "Bad Game Designer, No Twinkie! XI," *Gamasutra: The Art and Business of Making Games*, 2010. [Online]. Available: https://web.archive.org/web/20190801022928/http://designersnotebook.com/Columns/108_BGDNT_XI/inconsis108_bgdnt_xi.htm (visited on 04/02/2021).

[44] Extra Credits, *Depth vs Complexity - Why More Features Don't Make a Better Game - Extra Credits*, YouTube, 2013. [Online]. Available: https://www.youtube.com/watch?v=jVL4st0blGU (visited on 07/12/2021).

[45] S. Lundgren and S. Björk, "Game Mechanics: Describing Computer-Augmented Games in Terms of Interaction," in *Proceedings of the 1st International Conference on Technologies for Interactive Digital Storytelling and Entertainment*, Darmstadt, Germany, 2003.

[46] R. Rouse III, *Game Design: Theory and Practice*, 2nd. Jones & Bartlett Publishers, 2004, ISBN: 1556229127.

[47] R. Hunicke, M. LeBlanc, and R. Zubek, "MDA: A Formal Approach to Game Design and Game Research," in *Proceedings of the AAAI Workshop on Challenges in Game AI*, vol. 4, 2004.

[48] K. Beck, J. Grenning, R. C. Martin, M. Beedle, J. Highsmith, S. Mellor, A. van Bennekum, A. Hunt, K. Schwaber, A. Cockburn, R. Jeffries, J. Sutherland, W. Cunningham, J. Kern, D. Thomas, M. Fowler, and B. Marick, *Manifesto for Agile Software Development: Principles behind the Agile Manifesto*, 2001. [Online]. Available: https://web.archive.org/web/20210701024401/https://agilemanifesto.org/principles.html (visited on 08/05/2021).

Ludography

[8] Nintendo, *The Legend of Zelda: Breath of the Wild*, Nintendo, Mar. 3, 2017.

[15] id Software, *Quake*, GT Interactive, Jun. 22, 1996.

[21] Rockstar Games, *Red Dead Redemption 2*, Rockstar Games, Oct. 26, 2018.

[23] Maxis, *SimCity*, Maxis, Feb. 2, 1989.

[24] Creative Assembly, *Shogun: Total War*, Electronic Arts, Jun. 13, 2000.

[25] Konami, *Metal Gear Solid*, Konami, Sep. 3, 1998.

[26] Ion Storm, *Deus Ex*, Eidos Interactive, Jun. 23, 2000.

[27] Mojang Studios, *Minecraft*, Mojang Studios, Nov. 18, 2011.

[28] Ubisoft, *Assassin's Creed Odyssey*, Ubisoft, Oct. 5, 2018.

[32] CD Projekt Red, *Cyberpunk 2077*, CD Projekt Red, Dec. 10, 2020.

[33] Quantic Dream, *Heavy Rain*, Sony Computer Entertainment, Feb. 18, 2010.

[37] Nintendo, *The Legend of Zelda: Ocarina of Time*, Nintendo, Nov. 21, 1998.

[38] Nolla Games, *Noita*, Nolla Games, Oct. 15, 2020.

[41] Ostrich Banditos, *Westerado: Double Barreled*, Adult Swim Games, Dec. 9, 2016.

[42] Square, *Final Fantasy VII*, Square, Jan. 31, 1997.

[49] Ubisoft, *Far Cry 5*, Ubisoft, Mar. 27, 2018.