Adam M. Smith
amsmith@ucsc.edu
UC Santa Cruz
Santa Cruz, California

Figure 1: The six different classes where a player can fail while playing videogames.

Failure is integral to playing videogames, we do not have a precise terminology for discussing different categories of failure. This is a problem because certain failures are critical to the design intent of the game, and, as such, are desirable, whereas other failures detract from the play experience and are meant to be avoided. In this paper, we taxonomize several classes of failure in the player's experience, offering a diagnostic tool that distinguishes *in-loop* failures from *out-of-loop* failures. By classifying both games and specific failure instances, the Taxonomy of Failure extends present vocabularies for: game designers making design choices, scholars critiquing the usability of games, educators teaching games, data analysts segmenting players, and game developers creating more adaptive games.

- **Applied computing** → **Computer Games**;
- **Human-centered computing** → *Interaction design process and methods*.

Failure, Game Design, Game Development

Batu Aytemiz and Adam M. Smith. 2020. A Diagnostic Taxonomy of Failure in Videogames. In *International Conference on the Foundations of Digital Games (FDG '20)*, September 15–18, 2020, Bugibba, Malta. ACM, New York, NY, USA, 11 pages. <https://doi.org/10.1145/3402942.3402979>

Failure in videogames is not desirable. A cursory look at several beloved videogames is enough to challenge the previous blanket statement. From popular cult classics such as the *Dark Souls* [12] franchise to the intentionally difficult and frustrating games such as *QWOP* [21] and *Getting Over It* [22], many videogames are known to explicitly invite failure. These games are enjoyed, even cherished, by many who play them [2, 8]. When players fail in these challenging games, the failures are seen not as a problem [52], but rather as a part of the game loop [50]. However, not every failure is met with such enthusiasm, nor is every type of failure healthy [4, 48]. If a player is failing because the on-screen text is too small for them to read, or because they do not know the controls then there is a problem—no one is excited to face these types of challenges. So, if some failures are welcome, whereas some are unwanted, and if some failures are opportunities to grow, whereas others make players feel helpless, how do we know, articulate, and communicate these differences? In this paper, we contribute a diagnostic taxonomy (visualized in Fig. 1) to aid in answering these questions.

Players can fail in a variety of different ways while playing games. Failure can be part of a game and might even be expected to occur naturally in the span of a gameplay session. Failure might also be undesirable and necessitate a response from either the game itself, in the form of a helping action, or from the game designer, in the form of a design change. For example, in an analytical game such as chess, we expect most of the exciting challenge of the game to revolve around the failure to form effective plans (and counter plans or counter-counter plans). If a player is failing because they made an ineffectual plan, this is par for the course and does not require a response from neither the game system itself nor from the game designer. However, if the players are constantly failing because the chess pieces are hard to grasp and keep slipping from their hands then this asks for a better-designed chess set, as moving the chess pieces around should not be where the challenge of chess lies. In the former failure case, the player failed because they didn't plan their moves correctly, which matches the presumed design intent of chess. However, if the player fails at handling the chess pieces this requires an intervention as this type of failure (dropping the pieces) does not match the presumed design intent of chess.

Let us look at another example: Jenga [49]. Jenga is a game of physical skill in which players take turns removing one block at a time from a tower constructed of 54 wooden blocks. Each block removed is then placed on top of the tower, creating a progressively taller and more unstable structure. In a game of Jenga, in addition to your strategy of deciding which blocks to move, your ability to physically manipulate the blocks also matters—maybe even more so! Contrary to chess, dropping a game piece while performing an action is acceptable because the failure in removing the block by dropping it - after having decided which block to move - matches what we imagine to be the design intent of Jenga.

In chess, after having made a plan (deciding which piece to move), the act of executing that plan should be seamless and should not pose any chance of failure, as it would detract from the play experience. Whereas in Jenga, after having made a plan (deciding which piece to remove), you can still fail at the act of executing that plan, and that failure would not be jarring to the overall experience at all, as it is part of the expected gameloop.

In our Jenga and chess examples, we outlined two similar types of failures (failures that occur when physically handling a game piece) and described how the consequences of those failures on the play experiences can be wildly different. This is why we need to have terms to clearly differentiate the types of failures we want from the types of failures we would rather avoid. In this paper, we designate the failures that support the design intent of a game as *in-loop* failures (referring to the ideas of a game loop). The existence of *in-loop* failures is not detrimental to the enjoyment of the game, and in many cases, this type of failure is the main component that makes the game challenging and engaging. On the other hand, *out-of-loop* failures represents the unintentional (from the designer's side) failures that detract from the vision of the game and should therefore be minimized. To use our earlier example, dropping the chess piece would be an *out-of-loop* failure whereas dropping the Jenga block (and the tower with it!) would be an *in-loop* failure, even though they are both execution failures. In Jenga, physical manipulation is a part of the intended challenge, whereas in chess

physical manipulation of the pieces is definitely not a part of the intended challenge.

In the context of videogames and this paper, we use the term “failure” to describe the inability to make progress towards the objective (or goal) suggested by the game. It is important to note that just because the game is suggesting an objective, it does not mean the players are going to play the game to accomplish that goal. However, having this terminology will still help designers hone in on what the game is trying to accomplish and if appropriate (such as when discussing creative sand-box games or transgressive play [5]) can be extended to include player-set goals.

Currently, the videogame industry lacks any specific vocabulary to describe the differences pertaining to failures that occur in videogames. This gap makes it difficult to communicate which types of failures are desirable and which should be avoided, especially because communication is one of the problems game development teams often face [45]. Our paper proposes a taxonomy of failures to facilitate this differentiation and increase specificity in communication. If we have a vocabulary that allows us to describe how we, as game designers, are making our players fail, we can investigate whether these failure instances match our design intent. We propose this Taxonomy of Failure as a tool to help designers, critics, analysts, educators, and scholars be more precise in their discussions of failures in videogames.

We envision this taxonomy being used to:

- (1) describe what categories of failures are *in-loop* and which are *out-of-loop* in the games we design, and
- (2) classify different instances of failures to better understand which are problematic and should be reduced, and which are essential and should be fostered.

2 RELATED WORK

Failure is an inherent component of games. In many cases, it is tailored to be a big part of the whole experience — to the extent that McGonigal [40] claims players spend a significant amount of their time failing. As such, the importance of failure has been a part of the discourse surrounding games from several different angles. Juul [33] investigates whether the presence of failure affects the enjoyment derived from playing videogames and concludes that players create elaborate theories of failure as a source of enjoyment. Further, in *The Art of Failure*, Juul [34] discusses the aesthetics of failure and argues that it is a crucial aspect of what makes a game a game. Building on this foundation by Juul, Roche describes the notion of “valid” failure [47], investigating the cases where the game recognizes, either from a gameplay or a narrative perspective, that the player has failed.

Another concept tightly related to failure is challenge. Adams, in *Fundamentals of Game Design* provides two broad categories of challenges: cognitive and physical [6]. Thinking about what the challenge facing the player should be is another way to conceptualize and differentiate *in-loop* from *out-of-loop* failures. However, it is important to note that these gameplay categorizations of challenge still struggles with measuring difficulty [7] and does not cover all the different ways a player can fail. Shifting our attention from challenge to failure allows us to engage our users on the users'

terms, by exploring how the challenges we design are received by our players.

In *Uncertainty in Games*, Costikyan [16] discusses another aspect crucial to failure: uncertainty. This discussion provides another perspective to describe *in-loop* and *out-of-loop* failures. If the player expects high uncertainty in their actions, blunders resulting from those specific moves are *in-loop* failures because the player acknowledges the likelihood of failing. *Out-of-loop* failures are those in which the player does not expect uncertainty. They do not foresee a potential of failure, and they are jarred if failure does occur. This lens can be another way to reason about how *in-loop* and *out-of-loop* failures affect the play experience.

Another angle to investigate failure is the aspect of learning and mastery. Gee [25] describes good videogames as learning machines and explains that failing and trying again makes failure central to learning and a part of the fun. Similarly, in *The Art of Computer Game Design* [17] Crawford, and in *A Theory of Fun for Game Design* [38], Koster, argues that fun, learning, and failing are intricately connected. In their work exploring failures and breakthroughs [31] Iacovides et al. discuss relationships between learning and in game involvement. Learning and failure have been connected to one another in other domains as well. Kapur introduces the concept of productive failures [36] and shows the effectiveness and how to design for them [37]. Productive failures are the notion of allowing people to fail while attempting to solve complex problems. This kind of failure facilitates the understanding of the problem at hand. It is easy to see the similarities between the good game design argued by Gee and Koster, and the notion of productive failures introduced by Kapur. Anderson digs into this connection by investigating how failure in games promotes learning [10], and Klievig investigates whether the game can play a role in changing a person's relationship to failure [35]. Desirable difficulty [14] is another theory that puts emphasis on understanding how people fail and—through the failure and difficulty—learn. Desirable difficulty is concerned with matching the learner with the correct difficulty and challenge. In all these cases having a more precise language to specify and identify different types of failures would help the clarity and communication of the ideas.

While the idea of learning is deemed to be important when it comes to enjoying games, the tutorial levels meant to teach players how to play usually leave some things to be desired—even though they have a significant effect on the rest of the game [9]. It is common for players to complain about the overprotective and cautious nature of some in-game tutorials, especially complaining about how the tutorial robs the player of a sense of discovery [46, 59]. There has been some work in creating context-aware tutorials [11] and adjusting the game difficulty to fit the player [29] but there is still room for improvement. A taxonomy of failure can allow designers to decide when to provide tutorials and where to let the player explore, which is essential in allowing games to keep the sense of discovery and learning.

Another important benefit of being able to specify different types of failure is related to accessibility in games. There have been improvements when it comes to game accessibility [24] and there are many resources that educate game designers on how to make games more accessible such as the extensive *Game Accessibility*

Guidelines [4]. There are still, however, many open research questions in this area. Being able to classify failures will surface cases where the players are failing due to accessibility issues. Classifying failures can also contribute to making games more accessible. Another use for the taxonomy is Universal Design [53], a design approach that prioritizes removing barriers to engagement with experiences. A taxonomy of failure can allow the designers to be more precise in deciding which failures act as barriers to enjoyment (and so should be removed) and which failures are essential for the game experiences (and so should be preserved).

3 TAXONOMY OF FAILURE

In order to classify where people can fail while playing videogames we propose a specific **model of how people play videogames** (shown in Fig. 2). Our model is inspired by and builds upon Swink's feedback loops [55], Sellers interactive loop [51] and Don Norman's conceptual models [44], and is adjusted to make delineating failure classes more straightforward.

Building upon the aforementioned work, we claim that players have a model of the game even before interacting with the game, created by their cultural knowledge, previous game experiences, and the aesthetic and representational aspects that the game is presenting [30]. The model consists of players understanding of the mechanics of the game (what they *can do* in the game) and the goals communicated by the game (what they *should do* in the game). Players then use this model to form a plan that can be arbitrarily short or long, in an attempt to achieve the goals they have set for themselves. Finally, the players attempt to execute this plan. Regardless of whether the plan is successful or not, the feedback from the game allows the player to update their mental model of the game and form a new plan.

However, in order to even engage in this model-plan-execute-update loop, the players need to be able to encode their inputs to the system and decode the outputs that are coming out of the game. Encoding input refers to the ability of the player to engage with the game on the most elementary level, whether it is through pressing the correct buttons, knowing which gestures to make on a touch screen, or being able to voice the correct verbal commands, etc. Decoding output in this context refers to the ability of the player to parse and understand the feedback of the game, whether it be visual, aural, tactile or otherwise.

The steps described in our model of how a player engages with a videogame also represent the different classes of failures—points in our model where a failure can occur.

3.1 Encoding Input → What physical action will allow me to act in the game?

Before one can truly start playing a game they need to be able to interact with the game system itself. This class refers to the players' knowledge of the control scheme and the physical capability of using it. Is the player aware of the correct buttons to press, the right gestures to make, or in general, the valid set of actions they can take to interact with the system? More importantly, is the player physically capable of taking those actions?

In-loop encoding input failure can arise in puzzle boxes or escape rooms, where the challenge lies in actually figuring out the specific

Figure 2: The proposed model of how people play games, specialized for delineating different types failure.

actions to get a response from the system or the correct physical components to push and pull.

It is much more common, however, for failures in the encoding input class to be *out-of-loop*. A simple example would be the player not knowing which buttons actually are to the game. A more serious encoding input failure is when a player can not use a functionality due to a physical limitation. A recent example is when a player with a physical disability loses the ability to play *Civilization* due to a change in the user interface [3].

3.2 Decoding Output → Can I parse the feedback of the game?

In addition to the ability of saying things to the game one must also understand what the game is saying in return. Decoding output refers to the player's ability to understand the information presented by the game, whether it be visual, aural, tactile or through any other medium that the game is using.

Games that use decoding output as an *in-loop* class would be spot-the-difference or hidden object games where the challenge lies in parsing a generally complicated scene, a game where the player is trying to find differences between two images/sounds or games where part of the challenge lies in overcoming the chaotic visuals[15]. Any game where the challenge lies purely in perception can be considered to have decoding output as an *in-loop* class.

Many *out-of-loop* examples in the decoding output class fall under accessibility issues: If the player can not differentiate enemies from allies due to their colorblindness [26] and the game's color scheme this is a decoding output failure. Further examples would include not being able to read text on screen because of it being too small [23], or not being able to understand the natural language that conveys crucial information about the game.

Encoding Input and Decoding Output groups cover the fundamental requirements for any player to give input to the system

and receive clear feedback as an output. Only after a baseline proficiency in these two classes are established can the player start engaging with the game productively. It is important to note that many failures in these two classes stem from accessibility issues and it is crucial for game designers to be aware of these cases.

3.3 Discovering Mechanics → Do I know what I can do in the game?

This class of failure refers to the player's understanding of the game's mechanics and the set of available verbs offered to the player. In other words, does the player know the capabilities of the tools that they are given?

If the game is about figuring out how different mechanics relate to one another and finding creative ways of solving problems through emergent methods, that game is using discovering mechanics as an *in-loop*-failure class. The joy in playing the game comes from learning what the tools are, making the discovery of what the mechanics are is part of the fun.

On the other hand if the players progression is halted because they could not figure out how a specific lock-and-key style mechanic functions—whether it be not realizing that bombs can blow open cracked walls, or the specifics of how levers and pressure plates work in many adventure games (and the challenge lies in figuring out where you should place the bombs and the order of pulling the levers, instead of figuring out whether you can do those things)—then it is an *out-of-loop* failure in the discovering mechanics class. Another example would be the player repeatedly getting defeated by a specific type of enemy because they didn't realize that a specific tool in their toolkit deals more damage to that type of target. If the joy in the game is purely about mastery over the mechanics, and not in uncovering what the mechanics are, then discovering mechanics is an *out-of-loop* failure.

To be more concrete, in rhythm based games such as *Guitar Hero* [27] or *Tap Tap Revenge* [56], the available set of verbs is simple and should be well communicated, and the joy in the game emerges from getting better at those verbs, not figuring out what the verbs are. Whereas in *Legend of Zelda, Breath of the Wild* [43] understanding how different systems of weather, conductivity, and temperature relate to another would make discovering mechanics an *in-loop* class for that game.

This topic also comes up in design discussions that happen in the game industry. In the NoClip Podcast the game director of *Doom Eternal*[32], Hugo Martin, describes why deciding whether to make game mechanics easy to discover or not is important[1]:

[*Doom Eternal gets really*] heavy-handed on the player education stuff, because we do not want to make learning [mechanics] part of the experience. As you know in *Sekiro*, a lot of the [mechanics] are there to be discovered and that's a huge part of the experience. For us, I want to make it clear as day. The fun of the game is not figuring [mechanics] out, the fun of the game is mastering it.

3.4 Setting Goals → Do I know what I should accomplish in the game?

Any moment a confused player asks “What am I trying to do here?” is a failure instance belonging to the setting goals class. The setting the goal class refers to the moments when the player does not know what to do to make effective progress in the game, or what progress even means. In other words, as long as the player is clear on what their goal is, the failure is not a setting goal failure.

In many open world games figuring out where to go next and who to talk to in order to move the story further is a part of the challenge and enjoyment. In these cases it is acceptable for the player to not know exactly what the proper next step is. The challenge and fun comes from discovering what comes next as well as exploring the space of possible goals. In other games, there are several ways to achieve victory, from conquering the world by force to becoming a diplomatic mastermind. In these games deciding which objective to pursue, and sometimes prioritizing, is fun and, as such, setting goals is an *in-loop* failure class as well.

In many games however the designers want their players to always know what their next objective is. If at any point the player is questioning what they should do next and if this questioning is not a part of the game's design, then failures stemming from this confusion are *out-of-loop* setting goal failures. This is a prominent issue with confusing quest texts and cases where a non-player character shares what the player should do next without recording that information anywhere permanent.

When the player does not know where to go and what to do next in open world survival games, such as *Subnautica* [58], it is not a problem, since the design of the game affords wandering around and setting goals through the experience of spending time in the space. Whereas if a player does not know that the goal of *Super Meat Boy* [57], an action-packed platformer, is to reach the end of the level, then this is an *out-of-loop* failure! In *Super Meat Boy* the goal should be abundantly clear and the challenge should not come

from the player understanding what the goal of the level is, but rather, the process to reach that goal.

The **Discovering Mechanics** and **Setting Goals** classes together represent a large portion of the mental model that the player uses to play the game. Players create this model before even playing the game, and through the experience of playing the game, this model is constantly updated. The player continually learns and improves their understanding of what is possible in the game and what they should do next, by forming plans, attempting to execute those plans, and updating their model based on the result of their actions.

3.5 Planning → What steps should I take to accomplish my goals?

Planning is the player using their mental model of the game to decide what they will do next, given the goals they have and their awareness of the game's mechanics. A plan might span just the next few actions if the game is a twitchy reflex game, or a plan might be an elaborate battle strategy in a complex turn-based grand strategy game. Failures that fall into this class include the failures that arise from making an ineffectual plan and the failures at simply making a plan.

In many slow-paced turn-based games most of the fun arises in the plans the player generates: Which piece to play in chess, which position to move a soldier next in a turn-based tactics games or which route to take to the next level in a strategy game. Any failure that arises from making the wrong plan would be an *in-loop* planning failure.

If a game is designed to be purely about fast reaction times, or repeated executions and the player finds themselves making longer term plans (and failing at them) this would be an example of an *out-of-loop* failure. To be more concrete, imagine a game where the designer's intent is to have levels that can be progressed with any loadout that the player chooses. If there exists a scenario that a player is left unable to progress due to picking the wrong loadout (a long term plan they made at the start of the level) then this does not match the outlined design intent of the game. This hypothetical scenario is an example of how the Taxonomy of Failure could be used to validate and maintain the design intentions of a game while designing its experience.

3.6 Execution → Was I successful in taking the steps in my plan?

Execution is the ability to implement the plan formed by the player. There is a failure in this class whenever a player flops and does not succeed in something they had planned to do.

Any game where dexterity is intended to play a role most likely has execution as an *in-loop* failure class. Whenever the player misses the next platform by a pixel because they jumped one moment too late, or they could not play the correct notes to the beat, or they parry one moment too soon and lose their last life, they are experiencing an *in-loop* execution failure.

If the game is not testing the players ability to execute those plans and instead tests the formation of plans, any failure in execution would be an *out-of-loop* failure. If a player miss-clicks and sends their armies to the wrong tile on a hex-based strategy game, if a

player clicks on the wrong verb in an interactive narrative or if a player is forced to play a chess piece touched by accident due to the touch-move rule and as a result loses the game, they suffered an *out-of-loop* execution failure.

It is important to discuss the difference between failures in **execution** versus failures in **encoding input** as they might seem similar. The encoding input class refers to the players knowledge about the means in which they can act in the game. Does the player know what actions to take to act in the game and are they generally physically capable of taking those actions? Whereas Execution refers to their moment to moment success in taking those actions in the correct time and order. The ability to press a button does not directly correlate to one's accuracy of pressing it at the right time - an example of execution failure. However if you have a physical limitation that prohibits you from comfortably pressing the button at all then that is an encoding input failure.

In this section we have described the six categories of failure that exist in the Taxonomy of Failure:

- **Encoding Input:** What physical action will allow me to act in the game?
- **Decoding Output:** Can I parse the feedback of the game?
- **Discovering Mechanics:** Do I know what I can do in the game?
- **Setting Goals:** Do I know what I should accomplish in the game?
- **Planning:** What steps should I take to accomplish my goals?
- **Execution:** Was I successful in taking the steps in my plan?

4 CLASSIFYING GAMES

Why is this classification useful? One way we can use the Taxonomy of Failure is to classify games by designating which of the classes of failures are *in-loop* and which classes of failures are *out-of-loop*. This classification allows us to look at games we are interested in through another perspective, complementing other ways of understanding games.

If we are in the process of designing a videogame then we can use the taxonomy to act as a map in describing where we want the challenge to exist. This description can be helpful in evaluating the design decisions we come across as the development process continues by ensuring failures happen in the *in-loop* classes, and failures in the *out-of-loop* classes are minimized.

Additionally, the Taxonomy of Failure might prove useful is when we are trying to understand games that already exist. After classifying the games that interest us, we can look at how these games actually guide their players away from *out-of-loop* failures and towards *in-loop* failures.

Although we believe the taxonomy to be useful, it is important to be aware of the limitations. An overemphasis on using the taxonomy during the initial phases of design might pigeon hole the game into a certain idea, and might cause interesting insights to slip by. It is important to be flexible and try to truly understand what makes the game fun! If you observe that your players are enjoying failing in categories you designated as *out-of-loop* failures then it might be valuable to reevaluate your initial classifications.

4.1 Super Mario Bros

Let us look at this process through a few examples. We start with *Super Mario Bros* [42] (referred to as *SMB* from here on out). We claim *SMB* is a game about mastering how to move in a world filled with spikes, piranha plants and hazardous landscapes. Given this observation, the *in-loop* classes are **planning** and **execution** and any failure outside of those two classes are *out-of-loop* failures.

Figure 3: The proposed in-loop (green) and out-of-loop (red) failure classes of Super Mario Bros.

We can start by digging into the *in-loop* failure classes of *SMB*. What is the main challenge in *SMB*? First, the player needs to decide which path to take through the obstacles. Is the player going to jump on the goombas, or will they try to dodge them by running on the platforms above? There are many paths in the game that are viable, but there are also many which lead to certain doom. If the player takes a leap of faith and, after falling into the pit, realizes that jumping is not possible, it is not a big problem. They will just take the other route the next time. They conducted a plan to make the jump and executed the plan but the plan was flawed. Thus the player failed, but this sequence was not problematic because **planning** in *SMB* is an *in-loop* failure class. (shown in Fig. 3)

After making a plan in *SMB*, the player actually has to be able to implement that plan. It is one thing to think, “I am going to jump over the goomba, dodge the fireball, and vault over the piranha plant” and another to actually be able to do it! Either way though, if the player is not successful in their execution, they can always try again and no player is expected to successfully execute their plans on their first try, or with have a 100% success rate. When Mario dies there is a relatively short respawn time, and this supports our analysis that players are indeed expected to die several times in each gameplay session. These factors make **execution** also an *in-loop* failure class.

Now, we can also look at the *out-of-loop* classes and see how games attempt to steer the players from failing at those classes. Let us start by investigating the **encoding input** class. When we actually look at the hardware design, we see that the Nintendo Entertainment System controller (Fig. 4) has very few buttons, so just by looking at the controller the player can gauge how to give input to the system. Consider playing *SMB* on an emulator on a modern computer. If the player does not know which of the myriad keyboard buttons actually map to the initial NES controller buttons, then they can spend several minutes trying to figure out what hardware buttons to press, that would be a good example of an **encoding input** class of failure.

Figure 4: The NES controller with no redundancy and a clear way to engage each button. Image from public domain Wikipedia, owned by Evan-Amos.

When it comes to **decoding output**, most *Mario* levels are not about trying to figure out where the obstacles are; rather, they are about avoiding them. Whether it be because of the color palette limitations of the day, or by intentional design (and realistically, a combination of both factors), obstacles in *Mario* are mostly easily discernable, and in many cases deaths, are not caused because the player does not see the enemies coming. Even if that ends up being the case, the non-random nature of levels indicates that the designers were comfortable with the players knowing where the hazards were before trying to navigate through them.

Figure 5: The starting area of the first level (1-1) of *Super Mario Bros*. Image taken from Video Game Level Corpus [54].

In updating the player’s mental model, the first level of *Mario* (Fig 5) does a lot of heavy lifting. The way *Mario* is positioned when the game initially starts, and how the camera moves in relation to *Mario* all support the idea that *Mario* should go towards the right. Furthermore, the first few minutes of the game, with the goomba appearing (and not letting the player continue onwards until they demonstrate they can jump), the specific placement of the question mark tiles with power-ups, and the varying height pipes, teach most if not all the mechanics the player has to know to be effective at the game. If needed, a more detailed breakdown of *Mario* 1-1 can be found elsewhere [19]. Just within minutes the player knows what they can do and what they should do in the game respectively ensuring that no one is failing in **discovering mechanics** or **setting goals**. Miyamoto himself says, “So even within that one section we knew that the player would understand the general concept of what *Mario* is supposed to be and what the game is about” [18]. This gradual learning curve that starts at *Mario* 1-1 and continues throughout the game because *Mario* is not a game

about the player exploring how complicated and varied systems interact with each other or figuring out where to go; rather it is about making a plan and executing it.

One might argue that when a speedrunner is trying to finish *Mario* in under five minutes, they are effectively exploring the ways of going faster and, as such, uncovering the hidden (and in this case potentially unintentional) mechanics becomes a part of the fun, making **discovering mechanics** an *in-loop* failure class. This argument goes to show that these types of generalizations using the framework run the risk of assuming the entire game audience engages with the game in an identical way. When using the taxonomy, care must be given to understanding different styles of games that any game can support, which might create several different *in-loop* and *out-of-loop* failure classes for each play style and player segment.

4.2 Sid Meier’s Civilization

Figure 6: The proposed *in-loop* (green) and *out-of-loop* (red) failure classes of Civilization games.

Another game we can look at is *Sid Meier’s Civilization* (shown in Fig 6) series [20]. Similar to *SMB* both **encoding input** and **decoding output** are *out-of-loop* failure classes. In *Civilization* figuring out which buttons to press and visually identifying what is going on the screen should not be challenging.

When it comes to **discovering mechanics**, most *Civilization*-like games, we claim, expect you to learn the rules of the system and then try to be successful within those bounds. The joy does not come from exploring whether hammers help you build armies faster, or whether apples allow you to grow your cities— these are elementary building blocks of the game that the player is expected to utilize. If the player has difficulty grasping these concepts there is a problem and the game has aggressive in game advisors to ensure that this rarely happens. Thus **discovering mechanics** for *Civilization* is an *out-of-loop* failure class.

The main bulk of the challenge comes from the player making both long and short term plans to decide how to move their units, what buildings to construct, how to conduct diplomacy and what policies to enact. If the player loses the game, it is expected that this is due to making a series of bad decision, which means that this style of loss is within the gameloop. Thus planning is an *in-loop* failure class for the *Civilization* series.

Following from this is the **execution** class. Unlike *SMB*, after having completed the plan, there should be no difficulty in actually implementing that plan. If the player has decided to build a granary, pressing the buttons to do so should be simple. Similarly if the player

has decided to invade the neighboring city giving the order should be trivial. If the player fails to finish the granary because the city falls to enemy hands, or the invading army is defeated by the city's defenders, these are not execution failures—the orders were given and executed properly. Rather, they were planning failures, since the plans were not formed effectively which, as discussed above is not problematic. Therefore, this goes to show that in *Civilization* **execution** is definitely an *out-of-loop* failure class.

Figure 7: The screenshot displaying a gameplay scene of the turn based strategy game *Civilization VI* [20]

Different from *SMB* however, is the **setting goals** class. In *SMB*, the goal is to reach the right side of the screen and that is closely attached to victory. However in *Civilization* there are several ways of achieving victory. You can conquer all the capital cities of your enemies, you can talk your way into becoming a world leader or you can win the space race. Deciding on the victory conditions of the player's goal is a part of the fun. If the player fails at a given playthrough because they did not align their goal with the strengths of their nation this is not a problem, because they can learn from their mistakes and try again. As such, setting goals is an *in-loop* failure class for *Civilization*, and not an *in-loop* class for *Super Mario* where the only is to go right (ignoring transgressive methods of engaging with the game).

4.3 Legend of Zelda: Breath of the Wild

Figure 8: The proposed *in-loop* (green) and *out-of-loop* (red) failure classes of *The Legend of Zelda: Breath of the Wild*.

Similar to our earlier examples **encoding input** and **decoding output** are out-of-loop failure classes in *Legend of Zelda: Breath of the Wild* [43] (referred to *BoTW* from here on out). This game

is not about figuring out which buttons to press or parsing the screen; rather, it is about, deciding what to do in an open-world, discovering how the game systems interact with one another and fighting different types of enemy characters.

Unlike the previously mentioned games however, a big portion of the fun when it comes to playing *BoTW* comes from gradually learning how the different systems of the game interact with one another. Initially, the player can figure out that wooden weapons catch on fire when held close to a firepit, which can lead to the realization that the shrubbery too is flammable. This understanding culminates in the player figuring out that the fire creates an air current they can hijack to gain some altitude using their paraglider (shown in Fig.9). Sequences like these in which the player discovers how primitive pieces interact and create emergent experiences contribute to what makes playing *BoTW* engaging. If the player fails because they perished in the forest fire they just created—showing that they failed because they didn't understand the mechanic previously—this is not a (design) problem. Nor it is a problem if the player has their progress halted while they play around discovering what tools they have to solve the puzzle they just encountered. Thus discovering mechanics is an *in-loop* class for *BoTW*.

Figure 9: The screenshot displaying Link using the updraft created by fire to glide upwards.

When it comes to **setting goals**, most open world games have an *in-loop* class and *BoTW* is no exception. The player is given the overarching victory condition of defeating Ganon in his castle. The player can set his sights on that goal directly, or the player can decide to have their goal be becoming more powerful by recruiting other characters or finishing different storylines—all these goals are supported by the base game. If the player fails to defeat the boss because they set their sights on defeating him too early before accomplishing other goals this is not game breaking failure. The player is allowed to change and readjust their goals and try again later.

Planning and **execution** are also *in-loop* failures as with many other action games. When the player is in combat or climbing a rocky mountain, they first need to plot the actions they want to take and then successfully execute those actions. If they fail because they made the wrong plan—miscalculating how much stamina they would need to climb the hill and slipping down—or because they didn't execute their plan correctly—missing their bow shot and failing to take down the enemy from afar, thereby becoming

overwhelmed—the play experience is not hampered and the player can respawn and try again, updating their model with their new-found experience.

4.4 Where’s Waldo?

Figure 10: The proposed *in-loop* (green) and *out-of-loop* (red) failure classes of *Where’s Waldo? The Fantastic Journey*.

We would like to conclude this section with an unorthodox game, *Where’s Waldo? The Fantastic Journey* [39]. Although, we argued that in most cases **encoding input** and **decoding output** is an *out-of-loop* failure class, this does not always have to be the case. The only manner in which a player should fail while playing *Where’s Waldo* should be in finding Waldo, and as such **decoding output** is an *in-loop* failure class in this case. The player should know what their goals are, what they could do and which buttons they should press. After spotting Waldo, the rest should pose no difficulty, and as such, all other classes for this game are *out-of-loop* failure classes.

Figure 11: A gameplay scene from *Where’s Waldo? The Fantastic Journey* [39]. Can you spot Waldo?

5 CLASSIFYING INSTANCES OF FAILURES

So far, we have been classifying games, but another use case of the Taxonomy of Failure is to classify instances of failure. We defined failure as “inability to make progress towards the objective

suggested by the game” earlier in the paper. For many games this might be an explicit “Game Over” screen with a call for the player to try again, but for others it might take the form of a temporary setback. What exactly is going to be classified as an “instance of failure” will once again depend on the specifics of the game and the context in which the taxonomy is being used.

We would go through the trouble of classifying how our players are failing if this process would help us identify whether a design change is necessary or not. If we keep observing that the players are failing in a specific *out-of-loop* failure category, we can use this to inform our decisions as to what changes to make.

Another capability that is of interest to both industry [13] and researchers is player modeling [60]. In most approaches to player modeling the player failures are not differentiated, if represented at all [28]. Given that player response to different types of failures vary, we expect discriminating different types of failure to be useful for downstream applications.

But is the granularity of the Taxonomy of Failure enough to be a useful tool in the design process? One might argue that there can be cases where two failures of the same class might have opposing impacts on the player; even though they are in the same class, one failure might be *in-loop* and another might be *out-of-loop*.

A simple example of this is repeated failures. The first time the player misses a difficult jump is fine. Perhaps the first five times it is fine, maybe the first fifty. But for many games there becomes a point in which even an *in-loop* failure can become frustrating and detrimental to the intended play experience. In those cases, the first few failures of the execution class are possibly *in-loop*, and the latter cases are *out-of-loop*.

Another example of this comes in the form of specific mechanical interactions. We argued that in many cases failures in decoding input are *out-of-loop* failures related to accessibility. However, consider the flashbangs in many first-person shooters or the screen obscuring attacks in *Super Smash Bros* [41] and other party games. Suddenly, if a player fails as they could not decode the screen because it was visually washed out, it is once again acceptable.

To account for these type of particular cases, in practice, it is valuable to extend the Taxonomy of Failure by considering instances of failures specific to the game one is working on. Going back to our earlier example, in *Super Smash Bros*, **decoding output** might be deemed an *out-of-loop* failure class, but underneath the decoding output class, the specific instance of “failure due to screen obstruction mechanic” would be designated as an *in-loop* failure instance. The provided Taxonomy of Failure does not come with this level of specific instances by default— we believe that for most games, many of the provided instances would be invalid, and it is better to construct this layer for each game individually. As such, we suggest expanding the taxonomy for the game one is interested in more deeply exploring, by describing the concrete ways how a player can fail in that game. This exercise allows the designers to be more specific when needed, and we also believe that the sole act of outlining the possible failure cases in a game one is designing to be a fruitful activity. Examples of specific failure instances can be seen in Fig. 12.

When it comes to using the Taxonomy of Failure in practice, one cannot simply halt the playtest and ask why the player failed after a loss of health points. While it is possible for the designer of the

Figure 12: The extended Taxonomy of Failure with a series of potential extensions shown.

game to guess what is happening, that would still be a guess. A possible way to alleviate this problem is to record the play session and if time permits, go over the failures with the player and work with them to annotate the failures.

6 CONCLUSION

In this paper we described why it is important to delineate between different types of failures and we offered the Taxonomy of Failure with the six classes of failures to facilitate doing so:

- **Encoding Input:** What physical action will allow me to act in the game?
- **Decoding Output:** Can I parse the feedback of the game?
- **Discovering Mechanics:** Do I know what I can do in the game?
- **Setting Goals:** Do I know what I should accomplish in the game?
- **Planning:** What steps should I take to accomplish my goals?
- **Execution:** Was I successful in taking the steps in my plan?

Further, we discussed how classifying a game using this terminology can be useful for both developing and analyzing games: How, for each given game, each class can be described as *in-loop* or *out-of-loop*. That is to say whether the described class of failure is expected in the game loop or is to be avoided.

We also described how analyzing a game using the Taxonomy of Failure can offer insights into the analysis and critique of the given game and also offer another metric in deciding which design changes are appropriate given the expected players and the state of the game.

Our hope is that after reading this paper the reader's relationship with failure in their favorite games has changed ever so slightly. In our future work we are looking forward to validating the Taxonomy of Failure both as a teaching and a design tool.

REFERENCES

- [1] [n.d.]. Designing DOOM Eternal with Hugo Martin - Noclip Podcast 22. <https://www.youtube.com/watch?v=qBH1al01ew&feature=youtu.be&t=1545>
- [2] [n.d.]. Dwarf Fortress: Losing is Fun. <https://dwarffortresswiki.org/index.php/DF2014:Losing>
- [3] [n.d.]. r/civ - Dear Firaxis, the change to the great works UI makes it nearly impossible to use for me as a disabled gamer.
- [4] [n.d.]. A straightforward reference for inclusive game design.
- [5] Espen Aarseth. 2014. I Fought the Law: Transgressive Play and the Implied Player. In *From Literature to Cultural Literacy*, Naomi Segal and Daniela Koleva (Eds.). Palgrave Macmillan UK, London, 180–188.
- [6] Ernest Adams. 2014. *Fundamentals of Game Design*. Pearson Education.
- [7] Alena Denisova Swansea University, Swansea, Wales, United Kingdom, Christian Guckelsberger Goldsmiths, University of London, London, United Kingdom, and David Zendle University of York, York, North Yorkshire, United Kingdom. [n.d.]. Challenge in Digital Games | Proceedings of the 2017 CHI Conference Extended Abstracts on Human Factors in Computing Systems. <https://dl.acm.org/doi/10.1145/3027063.3053209>. Accessed: 2020-1-22.
- [8] Fraser Allison, Marcus Carter, and Martin Gibbs. 2015. Good frustrations: the paradoxical pleasure of fearing death in DayZ. In *Proceedings of the Annual Meeting of the Australian Special Interest Group for Computer Human Interaction*. dl.acm.org, 119–123.
- [9] E Andersen, E O'Rourke, Y E Liu, R Snider, and others. 2012. The impact of tutorials on games of varying complexity. *Proceedings of the* (2012).
- [10] Craig G Anderson, Jen Dalsen, Vishesh Kumar, Matthew Berland, and Constance Steinkuehler. 2018. Failing up: How failure in a game environment promotes learning through discourse. *Thinking Skills and Creativity* 30 (Dec. 2018), 135–144.
- [11] B Aytemiz, I Karth, J Harder, A M Smith, and others. 2018. Talin: A Framework for Dynamic Tutorials Based on the Skill Atoms Theory. *Artif. Intell.* (2018).
- [12] BANDAI NAMCO. 2011. Darksouls. PC.
- [13] Paul Bertens, Anna Guitart, and África Periañez. 2017. Games and Big Data: A Scalable Multi-Dimensional Churn Prediction Model. (Oct. 2017). [arXiv:stat.ML/1710.02262](https://arxiv.org/abs/1710.02262)
- [14] E L Bjork and R A Bjork. [n.d.]. Making things hard on yourself, but in a good way: Creating desirable difficulties to enhance learning. teaching.yale-nus.edu.sg ([n.d.]).
- [15] Dennis Carr. 2018. Nerve Damage. <https://igf.com/nerve-damage>
- [16] Greg Costikyan. 2013. *Uncertainty in Games*. MIT Press.
- [17] Chris Crawford. 1984. The art of computer game design. (1984).
- [18] Eurogamer. [n.d.]. Miyamoto on World 1-1: How Nintendo made Mario's most iconic level.
- [19] Extra Credits. 2014. Design Club - Super Mario Bros: Level 1-1 - How Super Mario Mastered Level Design.
- [20] Firaxis Games. 2016. Civilization VI. PC.
- [21] Bennet Foddy. 2008. QWOP. PC.
- [22] Bennet Foddy. 2017. Getting Over It with Bennett Foddy. PC.
- [23] Ethan Gach. 2019. The Year In Tiny Video Game Text, 2019. <https://kotaku.com/the-year-in-tiny-video-game-text-2019-1840682509>
- [24] Lee Garber. 2013. Game accessibility: enabling everyone to play. *Computer* 6 (2013), 14–18.
- [25] James Paul Gee. 2005. Learning by Design: Good Video Games as Learning Machines. *E-Learning and Digital Media* 2, 1 (March 2005), 5–16.
- [26] Cameron Gidari. 2014. What It's Like To Play Games When You're Colorblind.
- [27] Harmonix Music Systems. 2005. Guitar Hero. Sony Playstation 2.
- [28] Danial Hooshyar, Moslem Yousefi, and Heuseok Lim. 2018. Data-Driven Approaches to Game Player Modeling: A Systematic Literature Review. *ACM Comput. Surv.* 50, 6 (Jan. 2018), 90:1–90:19.
- [29] Robin Hunnicke. 2005. The Case for Dynamic Difficulty Adjustment in Games. In *Proceedings of the 2005 ACM SIGCHI International Conference on Advances in Computer Entertainment Technology (ACE '05)*. ACM, New York, NY, USA, 429–433.
- [30] Robin Hunnicke, Marc LeBlanc, and Robert Zubeck. 2004. MDA: A formal approach to game design and game research. In *Proceedings of the AAAI Workshop on Challenges in Game AI*, Vol. 4. 1722.
- [31] Ioanna Iacovides, Anna I. Cox, Patrick McAndrew, James Aczel, and Eileen Scanlon. 2015. Game-Play Breakdowns and Breakthroughs: Exploring the Relationship Between Action, Understanding, and Involvement. *Human-Computer Interaction* 30, 3-4 (May 2015), 202–231.
- [32] id Software. 2020. Doom Eternal. PC.
- [33] J Juul. 2009. Fear of failing? the many meanings of difficulty in video games. *sfu.ca* (2009).
- [34] Jesper Juul. 2013. *The Art of Failure: An Essay on the Pain of Playing Video Games*. MIT Press.
- [35] Kathie Anne Kallevig. 2015. Perceptions of failure in education: Changing the fear of failure through gamification. (2015).
- [36] M Kapur. 2008. Productive failure. *Cogn. Instr.* (2008).
- [37] M Kapur and K Bielaczyc. 2012. Designing for productive failure. *Journal of the Learning Sciences* (2012).
- [38] Raph Koster. 2013. *Theory of Fun for Game Design*. "O'Reilly Media, Inc."
- [39] Ludia. 2009. Where's Waldo? The Fantastic Journey. NintendoDS.
- [40] Jane McGonigal. 2011. *Reality Is Broken: Why Games Make Us Better and How They Can Change the World*. Penguin.
- [41] Nintendo. 2019. Super Smash Bros. Nintendo 64.
- [42] Nintendo Entertainment. 1985. Super Mario Bros. Famicom.

- [43] Nintendo Entertainment. 2017. The Legend of Zelda: Breath of the Wild. Nintendo Switch.
- [44] Don Norman. 2013. *The Design of Everyday Things: Revised and Expanded Edition*. Basic Books.
- [45] F Petrillo, M Pimenta, F Trindade, and others. 2009. What went wrong? A survey of problems in game development. in *Entertainment (CIE)* (2009).
- [46] Jack Pooley. 2019. 10 Worst Video Game Tutorials Everyone Hated.
- [47] Eoin Roche. [n.d.]. “Valid” Failure in the Design of Digital Games: An Introspective View from a Player’s Perspective. *scss.tcd.ie* ([n. d.]).
- [48] Mike Schmierbach, Mun-Young Chung, Mu Wu, and Keunyeong Kim. 2014. No One Likes to Lose. *Journal of Media Psychology* 26, 3 (Jan. 2014), 105–110.
- [49] Leslie Scott. 1983. Jenga. Physical Game.
- [50] Michael Sellers. 2017. *Advanced Game Design: A Systems Approach*. Addison-Wesley Professional. 226–231 pages.
- [51] Michael Sellers. 2017. *Advanced Game Design: A Systems Approach*. Addison-Wesley Professional. 121–131 pages.
- [52] Serge Petralito University of Basel, Basel, Basel-Stadt, Switzerland, Florian Brühlmann University of Basel, Basel, Switzerland, Glenna Iten University of Basel, Basel, Basel-Stadt, Switzerland, Elisa D. Mekler University of Waterloo, Waterloo, ON, Canada, and Klaus Opwis University of Basel, Basel, BS, Switzerland. [n.d.]. A Good Reason to Die | Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems. <https://dl.acm.org/doi/10.1145/3025453.3026047>. Accessed: 2020-1-22.
- [53] Edward Steinfeld and Jordana Maisel. 2012. *Universal Design: Creating Inclusive Environments*. John Wiley & Sons.
- [54] Adam James Summerville, Sam Snodgrass, Michael Mateas, and Santiago Ontañón. 2016. The VGLC: The Video Game Level Corpus. (June 2016). [arXiv:cs.HC/1606.07487](https://arxiv.org/abs/1606.07487)
- [55] Steve Swink. 2008. *Game feel: a game designer’s guide to virtual sensation*. CRC Press.
- [56] Tapulous. 2008. Tap Tap Revenge. IOS.
- [57] Team Meat. 2010. Super Meat Boy. PC.
- [58] Unknown Worlds Entertainment. 2018. Subnautica. PC.
- [59] Alex Wiltshire and Mechanic Man. [n.d.]. Don’t hate on tutorials.
- [60] Georgios N Yannakakis, Pieter Spronck, Daniele Loiacono, and Elisabeth André. 2013. Player modeling.