DEPARTMENT OF INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Master's Thesis in Informatics

Easy to Learn, Hard to Master: Tutorials for Challenging Games

Leo Traub

DEPARTMENT OF INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Master's Thesis in Informatics

Easy to Learn, Hard to Master: Tutorials for Challenging Games

Einfach zu lernen, schwer zu meistern: Tutorials für herausfordernde Spiele

Author:	Leo Traub
Supervisor:	Prof. Gudrun Klinker
Advisor:	Sven Liedtke, Daniel Dyrda
Submission Date:	15.10.2021

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

Munich, 15.10.2021 Leo Traub

Abstract

In this thesis we take a closer look at Tutorials in video games and Tutoring in general. A Tutorial can be seen as a closed environment, where we have total control over the challenge and the skill level of the player. We are therefore looking at the role of Tutorials in the Flow theory by Mihaly Csikszentmihalyi. We define an optimal amount of knowledge, that should be taught by Tutorials, which lies between an explicit minimum and an implicit maximum. The minimum can be taught with a Mandatory Tutorial in the beginning of a game.

Additionally we have implemented an adaptive system, which provides Hints to the player depending on the individual performance, such that experienced players still receive valid Tutoring. There are many ways to track the player based on ingame variables and the right visualization of the data has proven to be a reliable tool for self-driven improvements. However the resulting Hints, which can be seen as an additional source of Help, did not adapt to the player as much as it would be desirable, due to several reasons.

We have implemented all described Tutorial solutions in the highly challenging game "Afterthought" programmed by me and my Team at Studio Moondowner.

Kurzfassung

In dieser Masterarbeit werfen wir einen näheren Blick auf Tutorials in Videospielen und generell auf das Unterrichten. Ein Tutorial kann als geschlossenes System angesehen werden, in dem wir absolute Kontrolle über die Herausforderung und das Können des Spielers haben. Deshalb durchleuchten wir die Rolle von Tutorials in der Flow Theorie von Mihaly Csikszentmihalyi. Wie definieren eine optimalen Menge an Wissen, die durch Tutorials vermittelt werden sollte, welche zwischen einem expliziten Minimum und einem impliziten Maximum liegt. Das Minimum kann durch ein obligatorisches Tutorial am Anfang des Spieles beigebracht werden.

Zusätzlich haben wir ein adaptives System implementiert, welches dem Spieler je nach Leistung Tipps gibt, sodass auch erfahrene Spieler noch valide Anweisungen erhalten. Es gibt viele Wege, um den Spieler durch spielinterne Variablen zu messen, und es hat sich gezeigt, dass die richtige Visualisierung der Daten als verlässliches Werkzeug zur Selbstverbesserung dient. Allerdings haben sich die resultierenden Tipps, welche als zusätzliche Quelle für Hilfe anzusehen sind, aus verschiedenen Gründen nicht so sehr an den Spieler angepasst, wie es erwünschenswert wäre.

Wir haben alle beschriebenen Tutorial Lösungen in dem herausfordernden Spiel "Afterthought"implementiert, welches von mir und meinem Team bei Studio Moondowner programmiert wird.

Contents

1 Introduction

When we think about our favorite games, all of them have a set of interesting and interactable game mechanics. Most games live from the understanding of the game world and the choice of when to use a certain mechanic. But sometimes it can take rather long until one has understood the core gameplay and can intuitively make the right decision in each situation.

In order to teach the game, developers usually utilize Tutorials, that can vary in style a lot. There is still no optimal solution for the design of a Tutorial as it also depends a lot on the player. Everyone is a different type of learner and that is why we implement Tutorials on a wide range from theory to practice. Additionally we have to think about the availability of Tutorials, which can either be totally mandatory or super voluntary. No matter if they are written, spoken, graphical or interactable, all kinds of Tutorials are supposed to teach something to the player, meaning they want to increase the skill level of the player for upcoming challenges.

Therefore Tutorials are highly related to the classical Flow theory by Mihaly Csikszentmihalyi, which makes connections between the skill level and the challenge [1]. First of all we want to understand the relation of Tutorials to skill and challenge and we want to determine the status of Tutorials in the sense of Flow theory. It seems that Tutorials help to keep the player in Flow, when otherwise the skill level would be too low.

Research Question 1: *What is the role of Tutorials in the Flow theory?*

Hypothesis 1: *A Tutorial needs to increase the skill level of a player and present an appropriate challenge.*

As developers, we also need to think about the optimal amount of information for Tutorials. We assume that there is a minimum of information, as every game consists of rules and actions, which the game could not be played without. Intuitively a maximum is also supposed to exist. However that maximum can not clearly be defined, as it depends on the individual understanding and creativity of the player. The maximum is rather implied by decisions of the developers, who exclude hidden game mechanics from their Tutorials.

Research Question 2: *What is the optimal amount of knowledge, that should be taught inside Tutorials?*

Hypothesis 2: *The optimal amount of information lies between an explicit minimum and an implicit maximum of information.*

Particularly when thinking about challenging games, we need to consider the importance of Tutorials for experienced players. Most games tend to have some Tutorials in the beginning

of the game, but after some point the player is completely left alone. Anyway it seems effective to also give experienced players some sort of long-term guidance, when it comes to understanding the meta of a game.

Research Question 3: *How can we tutor an experienced player?*

Hypothesis 3: *With an increasing skill level of the player, explicit Tutorials become obsolete, however implicit Hints become valuable.*

Finally when we want to guide an experienced player, it is far more difficult to determine the strengths and weaknesses of an individual, in order to present the right Hint or Tutorial to that player. We need to analyze the gameplay and explain to the player, how it can be improved. This can be done by implementing an adaptive system, like we find them in the field of Intelligent Tutoring Systems.

Research Question 4: *How can we utilize an adaptive system in order to tutor the player?*

Hypothesis 4: *An adaptive system can measure the player's performance, in order to determine the right type of Tutoring.*

In this paper we will refer to the game "Afterthought", produced by me and my team at Studio Moondowner. The game is treated as an example, which utilizes several Tutorial implementations, that are also evaluated based on user feedback. Afterthought is a highly competitive game especially designed for the Speedrun community, which is known for analyzing the gameplay and all possible interactions on a deep mechanical level. The question arises, whether players of a competitive target demographic have to be treated differently, when it comes to Tutorial solutions. As the game is entirely oriented around its target demographic, we also want to keep the learning process as enjoyable and effective as possible for the users.

At first we will take a look at all different types of Tutorials in general and how they can be categorized. Afterwards we will learn about Flow theory and how it can be linked to different Tutorial types, before we will dive deep into the complex platformer mechanics of Afterthought. Different Tutorial implementations in the game are evaluated and discussed. Especially the adaptive system designed on the basis of Intelligent Tutoring Systems will be described in detail. What can we learn about Tutorials?

2 Related Work: Tutorials

While there are various ways of teaching game mechanics to players, one can not generally say, which way is the best. The best teaching strategy also depends on the learner, meaning that there should always be multiple implementations, which present the game mechanics.

Tutorials in video games can differ in a lot of properties. While the presence of Tutorials is the first property to think of, it is also important to consider the context-sensitivity of Tutorials. Are they embedded inside the game at points, where the player actually needs the information? Do they provide help voluntarily or are the Tutorials mandatory? As well as thinking about the availability, one also needs to think about the freedom, that the player is given during the Tutorial. Depending on the domain, it can either be beneficial to constrain players inside the learning environment, as well as giving them as much freedom as possible. [2]

2.1 Availability on a Range from Mandatory to Voluntary

While most games tend to have a mandatory Tutorial in the beginning, there are also many games, where the player is not forced to go through any kind of lesson. The desired help can still be integrated inside the game, but will only be used by the players on a rather voluntary basis. Few games do not offer any kind of Tutorial at all. But for those games the help might as well be available outside the game. Similarly, most games have hidden mechanics, which are not explained inside the game, however they might be explained in detail somewhere on the internet.

2.1.1 Mandatory

Nowadays it is almost mandatory for Game Developers to have a Mandatory Tutorial at the beginning of a game. The player is forced to learn at least one game mechanic, which is enough to beat the first obstacle inside the game, therefore describing a minimum set of information, which is necessary to play the game. The Mandatory Tutorial can already set the skill level high enough for the player to overcome the entry barrier of the game and reach a state of Flow as described in section 3.1.2.

In order to introduce the player to the game, a Mandatory Tutorial should not only increase the skill level of a player, but also present the challenge, for which the skill level is appropriate. Both parameters have to be balanced in order to achieve a Flow experience. In Afterthought we needed a lot of time to find an appropriate level of challenge for new players, as described in section 4.4.3.

Apart from their effectiveness, Mandatory Tutorials also tend to force the player into certain patterns and thereby restrict the freedom a lot. They often feel like a lesson in school and sometimes bore or even annoy the players. One Mandatory Tutorial at the beginning is acceptable for new players, but being forced to learn more and more game mechanics throughout the game with restricted freedom while doing so, can also have an opposite effect on players, such that they do not get motivated to play the game, but rather want to quit it. [2]

2.1.2 Voluntary but available

Not all players want to learn everything about the game. However there are always some players who want to learn more than others. For these players it seems reasonable to give them voluntarily accessible on-demand help inside the game.

In his study Andersen suggested, that the availability of on-demand help in a game would improve the player retention. Unfortunately during the statistical analysis at the end of his study he found out, that on-demand help could even harm the player experience, if and only if there were no other types of Tutorials available in the game. For the testing cases, where players had other types of Tutorials as well as on-demand help, there were no significant improvements of the overall play time or the number of cleared levels. [2]

Once the Developer has decided to give the player additional, voluntary information about the game, the question arises, how much information should actually be shown. If we have already given the player a minimum set of information in order to play the game during the Mandatory Tutorial, we need to ask ourselves, which is the maximum set of information, that we want to teach to the player. The answer to this question is discussed later in 3.2.4.

As it does not seem to matter, how much additional information we put into a game, many games offer an ingame glossary, which explains basically everything important. Usually it is only accessible through a rather boring menu (sometimes with a search bar) and will only be used when someone has very specific questions.

For action games like "Far Cry 5" the glossary contains 27 entries [3]. In an open world game like "No Man's Sky", the glossary contains 36 entries [4]. In games with highly challenging mechanics like "Monster Hunter: World" the glossary contains more than 120 entries [5]. The enormous amount of Tutorials in top-selling games emphasizes the hypothesis, that it does not really matter, how many Tutorials you have in a game. It matters far more, that this type of help is voluntary and does not force the player to learn everything.

2.1.3 Play Station 5: Game Help

The Play Station 5 established a new feature in the industry called "Game Help". With this feature the player can access an overlay, which gives hints on how to progress at any point of the game. Technically these information are presented outside the game, but as they have to be prepared by the Developers and can be accessed from inside the game, they are still considered as voluntary ingame Tutorials. Also the overlay can be customized, such that it appears as a little window on the side of the screen. [6]

The interesting part about this feature is, that it is not just under total control of the developers, but that it also tries to avoid spoilers. Depending on the player type, one can choose between different stages of hints, where the first one is only a vague phrase or small image. The hints are split into portions, such that the player does not get the entire solution at once. Thereby the engagement with puzzle games can be improved and makes the entire game accessible for more players, when otherwise many players might get stuck at some point. [6]

2.1.4 Super Voluntary: Outside the Game

Nowadays it seems that even if we wanted to, we could not teach everything about our game inside the game. With the internet and dedicated online communities, there is always a chance, that someone will find out something about a game, that nobody has ever found out before. It can either be bugs or hidden mechanics. As soon as the community of a game grows large enough, there might even arise a fan-made dedicated wiki, which explains the mechanics in far more detail than the Game Developer ever intended.

Especially for Speedrunners, which are the core demographic of Afterthought, this type of devotion is characteristic (see section 4.1.1 for the definition of a Speedrunner). For the Game "The Legend of Zelda: Ocarina of Time" [7], there is not only a wiki, which provides an entire walkthrough as intended by the developers [8], but even another dedicated wiki which explains all bugs and glitches of the game, that can be used during a Speedrun [9].

In contrary to the Game Help feature of the Play Station 5, these wikis contain a lot of spoilers for new players and can therefore ruin the self-driven discovery of ingame mechanics. We should always make a difference between guiding the player towards a solution or providing the solution all at once.

2.2 Immersion on a Range from Theory to Practice

It can have a big impact on the learning outcome as well as on the engagement, how Tutorials are presented inside games, but there is no clear tendency of which way works best. Even nowadays video games still contain written instructions like in a classical manual, however they are often accompanied by graphical content. In some cases the instructions might even be shown as a gameplay video, where the player can watch the performance of another player in order to get an idea, of what needs to be done.

While this seems to be the easiest way for the Developer, one can also design a context-sensitive Tutorial, where the player gets instructions during gameplay. Especially at the beginning of a game, this way of introduction is widely used, as it does not only feel like a lesson, but also engages the player with the game.

Even if having a guided Tutorial seems reasonable, there are also games, which don't seem to need Tutorials at all, as the players are asked to discover everything on their own. But this is actually not true. The level design itself might already teach interactions to the player, even if there are no instructions given. In a well embedded Tutorial the players do not even realize, that they are currently learning something.

2.2.1 Game Manual

Historically the first Tutorials for video games existed outside the game, where the player had to read a printed manual in order to understand the game mechanics, before even playing the game. Anyway that does not mean, that there was no immersion. In contrary to immersive Tutorials, that are directly embedded inside the game, a manual can utilize immersion in a totally different way. In figure 2.1 we will take a closer look at the manual of Pokémon Yellow Version [10], which was packed with any version of the game, that could have been purchased back in the past. As a printed form of Tutorial it still has some remarkable attributes. First of all on the title page it says "Trainer's guide" instead of "Manual", which already acts as a tool of immersion. The player is not treated as a "Player" who needs to take a lesson, but as a "Trainer" in the world of Pokémon.

Apart from introducing the player to the world, the characters and the story, the manual also explains the controls with a basic layout and then explains, how the gameplay system and the fighting system in the game works. Every page is filled with graphics and there are a lot of headlines, which serve the purpose to structure the page. The written instructions are kept as short as possible, so that nobody is overwhelmed with text blocks.

Additionally we can see prominent Hints in several places which are labeled as "Oak's Memo". We can therefore see Professor Oak as the Tutoring persona of the manual as well as the Tutor of the entire game. It shows, that it is sometimes important to give our Tutorials a face, as it can increase immersion and build a sense of trust towards a character.

Towards the end of the manual, we can see that the information level is getting more and more detailed. Some of the knowledge does not seem to be necessary any more, while other pages describe the meta in detail, however this is the glossary type of Tutorial, which has been discussed in section 2.1.2 and it may be valuable to some players on a voluntary basis.

Figure 2.1: Manual Excerpts of Pokémon Yellow Version from source [11]

Lastly we can see that the manual is not entirely designed as a lesson. On the last pages of the manual, we can see the Pokémon list, which does not only have an informational purpose, but also acts as a challenge. On the first page of the list, we see that almost all entries of the list are filled. However on the second page there are already a lot of gaps. The player is challenged to fill the gaps, which represents the core challenge of the game in the best possible way. Thereby we see, that Tutorials should not only teach the skills, but also define the challenge, even we are just designing a manual. [11]

2.2.2 Written Instructions

Nowadays written instructions are still popular and used in commercial games, but they usually serve as an additional source of help, when an information has already been learned. As mentioned in section 2.1.2 written instructions are often sorted in some kind of glossary, where they are always accessible to the player.

Written instructions are often accompanied by graphical depictions or screenshots with notes. Depending on the player type, either type of information might be beneficial, but it has been shown, that the learning process is usually enhanced by graphics, as too many and too badly written instructions will rather frustrate the player than help. [2, 12]

When defining textual instructions as a Developer, one needs to think in detail about the terminology. In Tutorials you define, what a game mechanic is called and you have to be consistent throughout the entire game, so that the player does not get confused, when you use different words for one and the same mechanic. In Afterthought we discussed about the right terminology for ingame mechanics extensively. Even if the Tutorials are embedded in the game and not purely textual, there are still written instructions on the screen. In Afterthought we are following three guidelines for the definition of textual instructions.

- Be precise and consistent with the terminology.
- Make them as short as possible.
- For conditional clauses, never have more than one condition and never have more than one result.

Additionally we were writing the names of game mechanics with animated fonts individually, such that the player does not only recognize the word but also the according design.

2.2.3 Watching others perform

Ingame interactions usually demand the player to perform multiple, mechanical actions in a row. Because a player can not see the dynamics of these interactions in a still frame, it is much more beneficial to show the player a recorded performance. Additionally the necessary inputs can be displayed, such that the player does not only see what the interaction looks like, but also how it is performed.

Usually these kinds of performances are recorded gameplay videos, that have been edited in order to contain information. Sometimes the video is not played in real-time but slower, so that certain parts of the interaction are emphasized and shown in detail.

In the case of Afterthought the Ghost feature explained in 4.1.2 fulfills the same purpose. It is not a recorded video and also does not look exactly like the player, but it shows the interactions together with the inputs on a graphical level and in contrary to a video, it is embedded in the gameplay, such that one can compare the own interactions to the Ghost interactions. It also provides some kind of video player, where one can watch the Ghost perform in different time speeds and pause at certain moments.

2.2.4 Performing with Instructions

As some players might lose track when switching between a manual and gameplay, there are also context-sensitive Tutorials, where the instructions are embedded into gameplay. These kinds of Tutorials are very valuable as they help the player to focus on a single task [2]. Nowadays they are the most popular type of Tutorials, because they also engage the player with the game. You already get a feeling for the game, while still learning it, which makes a huge difference for newcomers. As mentioned in 4.4.2 a well designed Tutorial can also be designed as a challenge and thereby put the player into a Flow state.

When thinking about the freedom in context-sensitive Tutorials, the player is usually limited to less possible interactions. The player is often forced to solve a task in a very strict and intended way. However limiting the interactions can also reduce the challenge.

For a challenging game like Afterthought these kinds of Tutorials tend to be the best, as they also teach the execution of mechanical interactions and not just knowledge. In section 4.4 we see in detail how the player can be challenged to learn something with the right amount of freedom.

2.2.5 Learning by Doing without Instructions

Philosophically one could say, that an entire game can be a Tutorial. In complex games the player can always learn something new throughout the game. Sometimes level design can even challenge the player to find something out. This type of Tutoring can especially be used in order to introduce new game elements, that have never been shown before. There are various examples from different games, that teach us, how the design of a level itself can already tutor our players or at least guide them towards a solution.

However we will take a look at an example from Afterthought: The first time a blue wall is shown, it is not explained how this kind of level geometry can be utilized by the player in any way. The fist blue wall in the game is aligned in a row with two blue balls as shown in figure 2.2. At that point the player already knows, that blue balls can be attacked in order to get a Reset, so the geometry of the level is now challenging the player to draw a conclusion. There are a lot of indicators, which imply that a blue wall can be attacked just like a blue ball in order to get a Reset. It has an equal type of animation and as described in section 4.2.1 the blue wall has an overlay of blue circles. All these indicators together will eventually make the player realize, that the wall can be attacked. In the end it is up to the player to simply try it out.

Figure 2.2: Tutorial without instructions

One could argue that this kind of Learning by Doing approach is not a real Tutorial, but when the learning process is intended and mediated by the Developers, it is also hard to say, that it is not a Tutorial. Therefore an entire game can be designed in a way, that keeps the player focused and learning. In order to achieve this, all game mechanics should always be clear and accompanied by indicators on multiple levels, such that a player always draws the right conclusions. If everything is done right, a curious player can even master the execution of hidden game mechanics, that have never been explained anywhere, by simply interpreting unambiguous ingame feedback.

3 Flow in Tutorials

If we ask ourselves, what we want to achieve with Tutorials in Games, we can draw simple conclusions: Above all we want to explain the game mechanics to the players, so that they have the minimum amount of information in order to start playing the game. In other words we could say, that the players' skill levels should be increased to prepare them for upcoming challenges in the game. This terminology highly corresponds to the Theory of Flow introduced by Mihaly Csikszentmihalyi [1].

3.1 Related Work: Flow

The state of Flow has been named by the psychologist Mihaly, who describes it as a mental state of full involvement and enjoyment. It is a universally uniform state of being, that everybody recognizes, if described to them, as it is usually linked to good memories. According to Mihaly it can only be reached under certain conditions and most commonly appears in various forms of play. Therefore it is obvious, that the theoretical concepts of Flow can also be applied to video games. [1]

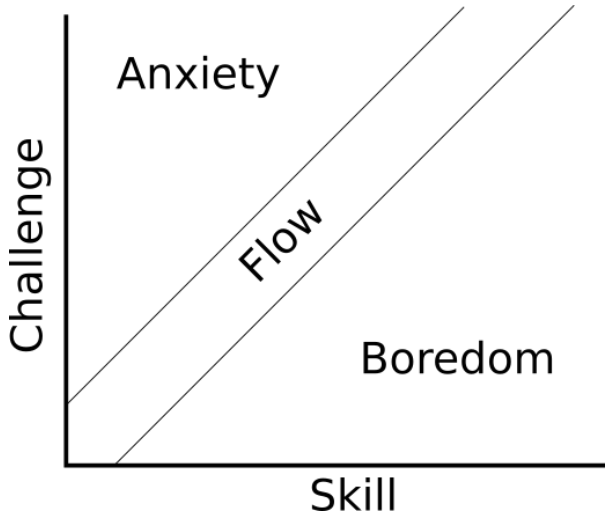

Figure 3.1: Flow Channel diagram by Mihaly Csikszentmihalyi [1]

3.1.1 Flow in Video Games

Especially for video games the Flow Channel diagram plays a big role. As shown in figure 3.1 it depicts the relationship between challenge and skill for any kind of activity. In theory the player is in a flow state, whenever the challenge of the current situation is in balance with the skill level of the player. According to this theory most games linearly increase the challenge throughout the entire game, while the player's skill level (hopefully) increases with every mastered challenge. When the challenge is too high for the current skill level, the player feels stressed or anxious. When the challenge is too low for the current skill level, the player feels bored. [1, 13]

Mihaly also stresses that the Flow state is subjective and depends on the personality of the player as well as on the perception of the current challenge and the perception of the own skill level. It is a psychological concept and therefore also linked to personality traits like confidence, persistence and curiosity. Mihaly states, that people with this kind of personality have an Autotelic Personality and are more accessible to the Flow experience than others. [1]

As described in 4.1.1 the average person in the target demographic of Afterthought has a lot of attributes of an Autotelic Personality, which makes it easier for these players to reach a Flow state, while playing the game.

3.1.2 The Experience Fluctuation Model

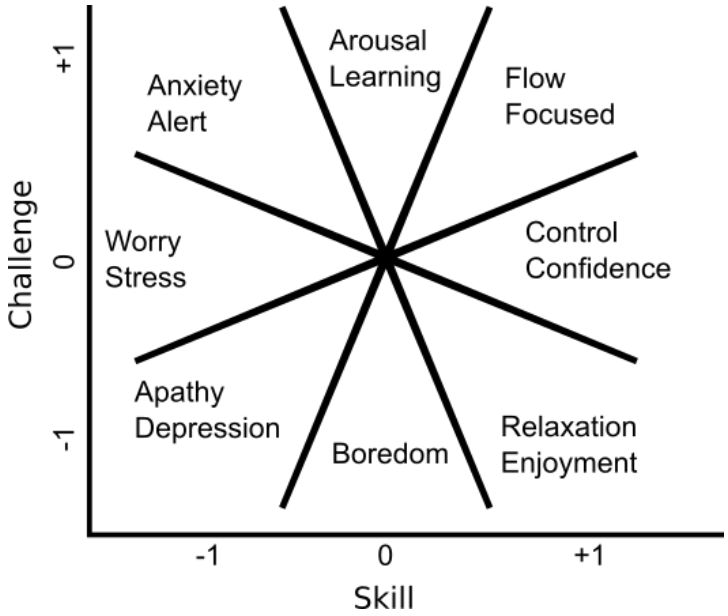

Figure 3.2: Experience Fluctuation Model by Mihaly Csikszentmihalyi [1]

A deeper insight on the subject by Mihaly suggests, that the desired Flow state can only be reached if a certain level of expertise has been reached, so that a person below that skill level, will not be able to enter the Flow state [1]. The Experience Fluctuation Model as shown in

figure 3.2 depicts some more mental states depending on the current challenge and skill. In this model it also clear, that Flow can only be achieved if the challenge as well as the skill level are above a certain threshold. At that point Tutorials already play an important role. As stated by Cowley [13] the minimum level of expertise to enter the Flow state is rather low in video games. It can also be called the entry barrier and a Mandatory Tutorial at the beginning of a game fulfills exactly the purpose to help the player above that barrier. This is the minimum set of information, that has to be taught, which is discussed later in 3.2.3.

Additionally the Experience Fluctuation Model makes a difference between Flow and happiness. Mihaly says, that people are not only happy, when they are having a Flow experience, but they can also be happy, when they are relaxing and collecting new energy for challenging activities [1]. When we think about video games, we do not always want to keep the players inside a Flow state, but sometimes also give them time and opportunities to relax. Above all in game design we do not only want to keep the players in the Flow state, but rather make them happy.

When we look at the model in figure 3.2 and think about the role of Tutorials inside this model, we can assume that Tutorials always ensure, that the player's skill level is not negative. Depending on the application and the user, Tutorials can even enforce the skill level to be positive. If we look at the diagram, the player is always somehow happy on the right side, when the skill level is positive, even if the Flow state is not reached. The dependency of happiness on the skill level emphasizes the importance of Tutorials in video games, because they are the only way to forcefully increase the player's skill level.

On the other side, if the challenge increases for a player with a neutral skill, we can find the player in a state of arousal, that is accompanied by learning. Curious players will leave that state faster and slip into the Flow zone according to Mihaly [1], for example when they discover new game mechanics and thereby increase their skill level. This sort of self-guided learning is much more satisfying to the player than learning everything inside Tutorials, but it can not be enforced and totally depends on the player's motivation and curiosity. However the learning process can always be supported by immediate and unambiguous feedback inside the game, such that the discovery of ingame dependencies gets easier.

3.1.3 Flow Measurements

In order to measure Flow, we could use the Experience Sampling Method as described by Mihaly. The purpose of this method is to simply ask the person about their current feelings at random times of the day. From a small set of different questions the participant's emotional state is defined [1]. Transferred to video games we could ask the players at certain points in the game, how pleasing their experience was in the last level, but this method seems to awkwardly break immersion far too often. We prefer a more elegant way.

If we want to keep the player in Flow using an adaptive system, we somehow need to measure, whether the player is currently enjoying the game. Flow as well as enjoyment can not trivially be measured, but we can draw certain conclusions about them. Especially when looking at the dependency of Flow from challenge and skill, we can say two things: The skill level can be measured in a rather reliant way and the challenge can be adjusted by the game

designer accordingly.

In his works Mihaly states, that the skill level can be derived from the highest challenge, one has already mastered. In video games we follow the same strategy, as we conclude, that a player has understood an obstacle once it has been overcome (filter theory). On the other hand Mihaly says, that this is not always a reliable metric. As Flow is a psychological construct, we also have to consider the perception of the player, which can differ a lot from the actual challenge and skill level. [1]

The User-System-Experience Model (USE Model) [13] tries to model the player's skill level, but relies on many different variables, which can be derived from gameplay. For example the accuracy of a player, when performing actions, can tell us a lot about the skill level. It seems to be a rather promising method of measuring skill, but we have to measure different parameters for every video game genre independently.

In theory, once we have measured the skill level, we would need to adjust the challenge, in order to keep the player in Flow. From Mihaly we know, that players (especially players with an Autotelic Personality) should choose the challenge on their own. In video games this is often done pre-game, such that a player can choose from different degrees of difficulty. However, As it would interrupt the experience of immersion, the player is never again asked to adjust the challenge later on. [1, 13]

The purpose of an adaptive system would be to adjust the challenge to the skill level throughout the entire game, without the player even noticing it, and therefore without breaking immersion. Unfortunately, we can not change the level geometry or game mechanics in competitive games like Afterthought, but we rather have to think about the challenge in a different way. In Afterthought the most challenging part is not completing a level or overcoming some sort of obstacle, but the challenge is to be fast, while doing so. Therefore it should be easy for a player to choose goals like "I want to beat that level in 20 seconds" individually. For less motivated players the next appropriate challenge according to the skill level could also be written on the screen, such that the challenge is clear and nobody overestimates themselves.

3.2 The Relation of Flow and Tutorials

In figure 3.3 we see the role of Tutorials in the context of Flow theory. The figure is a suggestive research model of how Tutorials may fit into the theory. It has been created based on the Tutorial implementations in Afterthought, which are discussed in this thesis. Also the figure was built as a universal draft, which can be applied to other video games.

The underlying hypothesis of the theories discussed in this section is that a Tutorial can be interpreted as a closed system. When we are talking about Tutorials in this section, we mainly mean immersive gameplay Tutorials, where the player's freedom is at least reduced a bit, in order to enforce learning. Also we are taking a look at the optimal amount of knowledge, that should be taught to a player by the game.

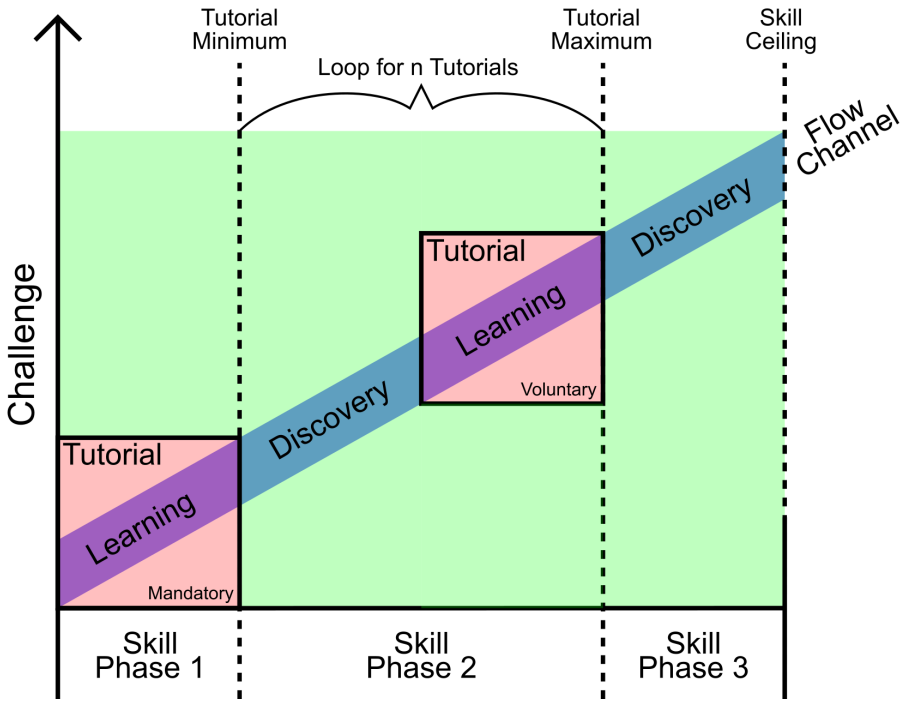

Figure 3.3: Research Model: Role of Tutorials in Flow theory

3.2.1 Tutorial as a closed system

If we look at a Tutorial as a closed, separated environment, we can interpret it as a game itself. Inside this closed system, we can apply the Flow theory and the Flow Channel diagram as described in chapter 3. The interesting part is, that inside the Tutorial, we always know the player's current skill level, as the purpose of any Tutorial is to increase the skill level. If we now want to keep the player in the Flow channel while playing a Tutorial, we also need to have some sort of challenge in the Tutorial and increase it according to the skill level. Inside a Tutorial the Flow state is characterized by direct Learning, whereas, when the player is inside a Level and does not have any kind of guidance, the Flow state is characterized by self-driven Discovery.

In his paper Andersen has stated, that the restriction of freedom in Tutorials to very limited interactions would keep the player focused and enhance learning. But at the end of the study Andersen evaluated, that the restriction of freedom did not have a clear impact on the learning outcome [2]. When comparing this hypothesis to Flow theory, limiting the interactions decreases the current challenge, as it makes it easier to overcome an obstacle, when you have less options to choose from. However keeping the challenge too low in a Tutorial can indeed have a negative effect, not on the learning outcome but on the chance of the player to enter a Flow state inside the Tutorial. When the player is already in a Flow state before entering a restrictive Tutorial, one might even get pulled out of that state, which has a psychologically negative effect on the player.

When the Tutorials in Afterthought did not include any challenge at all, we learned that players were just performing the described actions without thinking about the application of these actions (see chapter 4.4.2). This sort of brain-dead repetition of demonstrated interactions had a negative effect on the learning outcome, as it didn't leave any space for comprehension and self-reflection. Accordingly we limited the freedom as few as possible.

3.2.2 Skill Phases

In the research model the skill level of the player is divided into three phases, where Tutorials and other types of help are presented in different ways. The phases are divided by the "Tutorial Minimum" and the "Tutorial Maximum", which are explained in detail later.

In the first phase the skill level of the player is rather low. In this phase the player needs guidance the most. Therefore the Tutorials in this phase are labeled as "Mandatory" in figure 3.3. Essentially the player is inside this phase until one has learned enough to play the game without guidance. The phase, where Tutorials are Mandatory is short for most games, as players are usually encouraged to find out things on their own very soon. Also the phase can be shorter when there exist similar games with game mechanics, that follow certain conventions such that new players are already familiar with these actions.

Once a player is left alone inside the level for the first time, we consider the player to be in skill phase two. At this point there might still be Tutorials, which explain new game mechanics, but they are usually available on a more voluntary base, such that players will only play them if they really want to increase their skill and knowledge.

Phase three starts, when the player has learned and mastered everything, which has been explicitly explained inside the game. The skill level can still be increased inside this phase, but only through discovery. This phase is guided by unambiguous ingame feedback and statistical analysis, but there are no more explicit Tutorials, however there may be Hints available inside the game. Additionally the player can seek for Tutorials on a super voluntary basis on the internet.

These three phases highly correspond to the range of Tutorial implementations described in chapter 2.1. Mandatory Tutorials are important in the beginning, while voluntary Tutorials should accompany the player throughout the entire game for a very long phase. In the end everything depends on the player, who might even be willing to search for more help from third parties.

3.2.3 Minimum Set of Information

Most games have a Mandatory Tutorial in the beginning, where the player is forced to learn at least one game mechanic. The knowledge acquired in the Mandatory Tutorial should be enough to fulfill the first objective inside the game, therefore describing a minimum set of information, which is necessary to play the game.

When looking at the Experience Fluctuation Model as described in 3.1.2, Mihaly has stated, that a state of Flow can only be entered after a certain level of expertise has been reached [1]. It is hard to say, whether a player can already enter the Flow state during the first Tutorial of

a video game, as it seems possible, that the minimum set of information is exactly the level of expertise, which is necessary to enjoy a game.

We already stated, that we are increasing the challenge throughout our Tutorial resulting in a final challenge at the end of the Tutorial. So in a perfect scenario the player enters the Flow state during this last challenge and is then thrown into the first level of the game with a positive and motivated state of mind.

3.2.4 Maximum Set of Information

In the research model the maximum set of information is the theoretical point from which on the player does not need guidance any more. For any game there exists a point from which on the players have learned enough, in order to gather all following information on their own.

It is hard to generally define this point, as it lies somewhere else for any type of player. Creative types will start to experiment on their own as soon as the game gives them enough freedom, while casual Gamers usually need a lot more guidance.

From the developer's point of view the maximum set of information contains every explicit game mechanic, while hidden mechanics are excluded by design. A decision has to be made at some point, however the developers sometimes do not realize, that they have already made that decision. Thereby the maximum set of information is usually much lower than the actual Skill Ceiling of the game, which can only be reached by using the hidden mechanics.

For Afterthought there are a lot of references throughout this thesis, where information is considered to lie inside or outside the maximum set of information. As the developers, we made the decision to show the players every Skill in dedicated Tutorials, but by design we did not want to provide any information about possible combinations of Skills. Once the players use two Skills at once, they will hopefully find out, that new types of interactions arise, but it is totally up to them, to explore the rules behind these interactions. The discovery of connections is still guided by a lot of ingame feedback, but there are no explicit explanations inside the game. This is where we draw the line.

3.2.5 Skill Ceiling

The Skill Ceiling describes the necessary skill for the maximum of what can be reached inside a game. For Afterthought this is the theoretically most efficient Speedrun, which can be performed to beat the entire game. Especially in competitive games, where the player can choose from different playstyles, it is sometimes hard to clearly define the Skill Ceiling.

In the Speedrun community, players have created a software called the Tool-Assisted Speedrun Robot (TASBot). TASBot can send predefined inputs to the console with frame-perfect timing, such that it can play any game in the "best" way. Of course it does not count as a new record, when a bot beats a human player, but the performance of the bot tells us a lot about the Skill Ceiling. In order to perform the perfect run, someone has to feed all the inputs to TASBot, meaning that someone at least in theory had to figure out the most efficient way of playing a game. The Speedrun is then programmed on a frame-by-frame basis and naturally it would be impossible for any human to perform the inputs, as they are either too

fast or too exact. But the best possible human performance, which would be considered the Skill Ceiling, lies somewhere very close to the inputs, which could be performed by a bot. [14]

In Afterthought we offer another interesting feature, which allows the player to explore the Skill Ceiling: In some kind of training mode, the player can adjust the time scale of the entire game. When it usually runs at 100% time scale, the player can also play a level at 50% or even 20%. This leaves far more time for difficult inputs to enter and gives more time to navigate through the geometry, when the player is moving with a high velocity. This way any player can figure out a desirable way of making it through the level. Again playing with 20% time scale can not define the exact Skill Ceiling, as it might be impossible to execute some inputs precisely, when the time is at 100%, but it can still deliver some valid evidence.

3.2.6 Using Tutorials to keep the Player in Flow

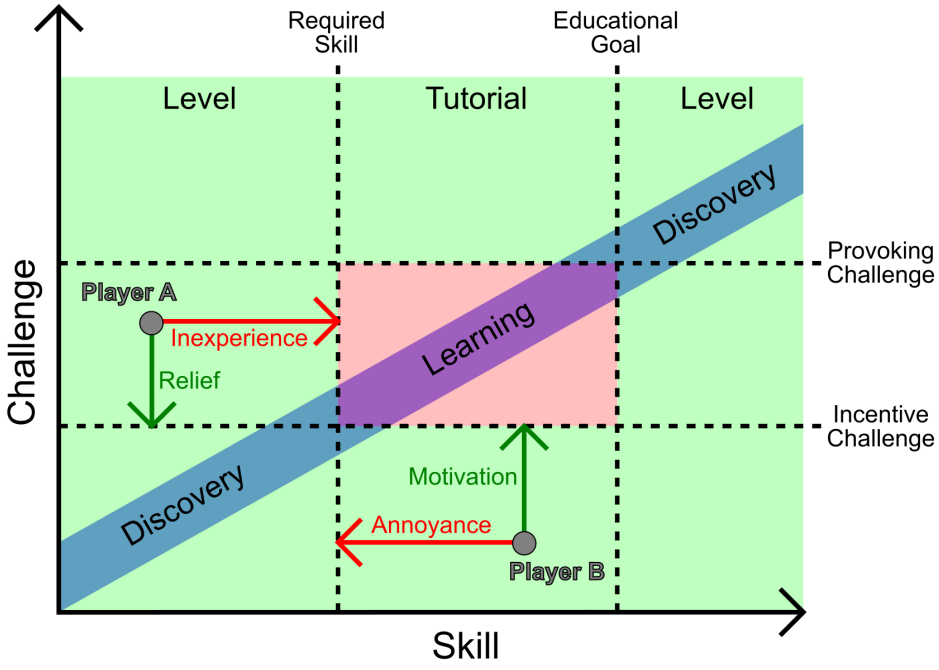

Figure 3.4: Effects of Tutorial timing on the player

In figure 3.4 we can see the importance of timing, when we introduce the player to new mechanics, meaning the moment, when a new Tutorial is presented to the player. As a Tutorial is understood as a closed environment, it starts at a certain skill level, which is necessary in order to understand the lesson. It then teaches the player how to reach a higher skill level, which is considered to be the educational goal of the Tutorial. As the challenge inside the Tutorial needs to be increased according to the projected skill level, it ranges between two levels of challenge. In the beginning of the Tutorial, there is an Incentive Challenge, which

introduces the new game mechanic and should somehow excite the player. Whereas the final challenge of the Tutorial is rather provoking and dares players to explore their limits.

In the figure we can see two examples, of how Tutorials can pull players back into the Flow channel. Tutorials can have rather positive effects on the general enjoyment of the game, however the timing is important, as there are also negative effects, which might dominate. We are using the examples in order to evaluate the research model of this section.

Example A

Before entering the Tutorial, Player A is confronted with a challenge, that is beyond the acquired skill level. In the original Flow diagram as described in chapter 3, Player A is in a state of Anxiety.

When the Tutorial starts, Player A is thrown into a new environment, where the required skill level is beyond the actual skill level of Player A. We call this gap of knowledge Inexperience and it is recommendable to keep this gap as small as possible, when introducing new Tutorials. For instance, imagine a newcomer, who has not acquired any skill inside the game yet, and a Tutorial tries to teach the most advanced mechanics to the player, without even explaining how to perform a basic jump.

Aside from the negative effect of Inexperience, the Tutorial also has a positive effect of Relief on the player. Player A perceives the current challenge inside the game as higher than the own skill level. The Incentive Challenge of the Tutorial makes clear, what the game is actually demanding of Player A right now. When Player A sees the difference between the perceived challenge and the actual challenge, Player A feels relieved.

When we look at the effects on Player A, we can pose the theory, that a Tutorial only has a positive effect, if the feeling of Relief outweighs the Inexperience. The gap of knowledge can usually be compensated, when it is not too large and when the player is willing to make assumptions inside the Tutorial. In that case the player would be pulled back into the Flow state.

However if the gap is too large, the entire lesson might be beyond the understanding of Player A. The initial feeling of Anxiety would only be strengthened. The player would be even further away from the Flow state than before.

Example B

Player B is in a state of Boredom before the Tutorial starts, as the perceived challenge is smaller than the skill level.

Upon entering the Tutorial, Player B feels annoyed, as the Tutorial requires a lower skill level than the player has already achieved. Resetting to a lower skill level inside a Tutorial is still rather acceptable for most players, as long as the difference is not too big.

In the case of Player B the positive effect of the Tutorial comes right on call. The Incentive Challenge of the Tutorial is higher than expected and motivates Player B, who was already looking out for new challenges.

Again when we look at both of the effects on Player B, we can assume, that the positive effect of Motivation should always dominate the feeling of Annoyance, so that the player can enter the Flow experience again. Otherwise the initial feeling of Boredom might only be increased, if the skill level of Player B was already far beyond the required skill level of the Tutorial, or even far beyond the educational goal.

Summary

As we can see in figure 3.4 the introduction of a new Tutorial can have positive as well as negative effects on the player. Under the assumption, that the player is not too far away from the Flow channel, we can use Tutorials as tools in order to pull the player back into the Flow state. As described in this section there is always a margin for which the positive effects outweigh the negative effects. The margin might be different for different player types. We could even imagine a Player C who is already in the Flow state before entering the Tutorial, on whom the Tutorial might have a neutral effect.

In order to determine the best timing for a new lesson, we have to measure the skill level of our players and create adaptive Tutorials, which pose the right type of challenge for every player individually. But before we can implement an adaptive system, we have to look at the game mechanics of Afterthought and at the already existing Tutorial solutions.

4 Afterthought

Afterthought is the name of a game developed by me and my friend Andy Tran. It is an athmospheric 2D platformer with nearly infinite movement options. We have been working on the project for five years now. In early 2020 we received financial support from the Bavarian funding organisation "Film Fernseh Fonds Bayern" (FFF). Since then we have produced the full game, which contains 40 levels and 8 bosses. Graphics from the original design document of Afterthought were used in this section.

Figure 4.1: Afterthought Title Artwork

4.1 A Game for Speedrunners

The main production goal always was to create a game for Speedrunners. We analyzed our target demographic to have very certain attributes. Their main goal is to be as fast as possible, meaning they want to find the most efficient way of making it through the game by analyzing the mechanics, in order to save time with every interaction.

4.1.1 Definition of a Speedrunner

The word "Speedrunner" refers to a person, that tries to run through a game with the highest possible speed. For some games, which are not actually built for racing, this means, that the person is trying to beat the game in a way, that was not intended by the developers. If you look at the way, a Speedrunner is playing any kind of game, you can observe a certain set of characteristics, which are highly important to us.

In the history of Speedrunning, it all has started as some kind of challenge. On the example of "The Legend of Zelda: Ocarina of Time" [7, 15] (short OoT) one can see, that a Speedrunner does not only use the provided game mechanics, but also bugs and errors in order to get through the game. OoT was never a game, where completion time was the most important

objective. However a whole community has formed around the game, trying to beat it as fast as possible, because people just loved the game so much, and wanted to find out everything about it [16].

The way that Speedrunners have analyzed OoT over the years is almost scientific. The acquisition of knowledge is mandatory for any Speedrun. It ranges from simple tables with all kinds of movement speeds, that can be achieved by the character, to advanced knowledge of the game engine. Every part of the game is deconstructed into little passages, for which one has to find out, whether "rolling" or "backwalking" is the faster game mechanic in this certain situation. Speedrunners know everything, that is possible in the game. They can utilize glitches in order to skip major parts of the story or literally write ones and zeros to the inventory. [16]

When everything is known about the game, one can start to plan the actual Speedrun. At this point the player needs to decide, which parts of the game actually have to be played and which can be skipped. Using a glitch, where the player gets teleported to the end fight after beating the first dungeon, almost every part of OoT can be skipped, except some parts, that are actually mandatory, just like the end fight. Now it is important to determine crucial parts of the planned Speedrun, for example points, where the player needs to perform an action with an exact angle. These situations can be practiced beforehand. [15]

When it comes to actually performing the Speedrun, it is not only important to get all the inputs right, but randomness also plays a big role. For some parts of a Speedrun, the player can run into sequences, where objects are randomly positioned or behaving randomly, such that a passage can be completed faster (or slower), than with another random distribution. In that case even if everything is done right, luck still plays an important role. [15]

All these observations lead to gameplay decisions in Afterthought.

4.1.2 Mechanics for Speedrunners

As described in 4.2 the Game Mechanics offer near infinite different options of combining actions in order to be fast inside a level. Identifying the most efficient way of making it through the level is the core challenge of a Speedrunner. In Afterthought you get a rank for the time of your score. The ranks range from A to F, similar to the american grading system in schools, with an additional S rank for an extraordinary good time.

Apart from the game mechanics rewarding fast playthroughs, there are some other features, that were integrated especially for Speedrunners. Of course, as the developers, we are trying not to give the players any opportunities for unwanted glitches, however there are some combinations of Skills, Resets and Level geometries, which allow complex interactions, that almost feel like glitches.

Aiming with Skills

All of the Skills described in 4.2.2 can be performed in different directions. The performed direction usually results in a speed boost in said direction, so precise aiming is key while maneuvering through the level. In order to aim, the player can keep the Skill button pressed,

which slows the ingame time down for as long as the button is pressed. In slow motion players can take as much time as they want for aiming and sometimes wait for the right moment to perform a difficult interaction. On the other hand the score timer is not affected by the slowed ingame time, thus ruining the score for Speedrunners, if they spend too much time on aiming. The better a player gets, the faster one will be able to press a Skill button, adjust the direction angle and release the button, so that in the end, the entire interaction can be performed in a single moment.

Instant Reload Button

Because Speedrunners generally spend a lot of time in the game, everything has to be replay-friendly. One of the core features is an instant Reload button, that resets you to the start point of the level, whenever you fail somewhere in the level. This feature has proven to be extremely powerful and addictive. Especially in the first ten seconds of a level players tend to reset the game a lot of times, until they finally hit the desired inputs right to get an exact movement trajectory.

Time Damage

As well as giving the Speedrunner opportunities, punishing mechanics inside the game are also built for Speedrunners: There is no classical kind of damage in Afterthought, which would let the players die in order to keep them from making it through the level. If one receives damage, the ingame timescale is affected in a negative way, such the gameplay gets slower every time the player receives damage. This makes the game easier for unexperienced players, as it offers more time for reactions. But on the other hand it is a crucial punishment for any Speedrunner, as the score timer is not affected by the slowed ingame timescale, thus ruining the score. At this point the instant Reload button comes into play, because usually a Speedrunner will reload the level upon receiving damage.

Speedrun Mode

Gameplay changes dramatically, if the player is not only trying to speedrun a single level, but the whole game. Usually players have to count the time, they spend on a Speedrun, externally as most games are not built for Speerdrunners. For that reason we are offering a Speedrun Mode inside Afterthought, that lets the player play through all 40 levels without any interruption. The instant Reload button is also available in Speedrun Mode. It still reloads the current level, but it does not reset the overall master timer. If the player now receives damage inside a level, it can be a time-saving decision to either reload the entire level or to keep playing from the current point.

Ghost

Optimization plays a big role upon planning the route through the game. In order to optimize the player has to compare the current run to the previous one. For that reason the game is

recording all the inputs of the player, so that they can be viewed again later. This feature is called the Ghost and it can display the exact trajectory and inputs of any run in realtime, such that players can compete with their own scores. Every score in the game is saved with the corresponding Ghost, so that one can even compete with online scores by other players in order to beat them.

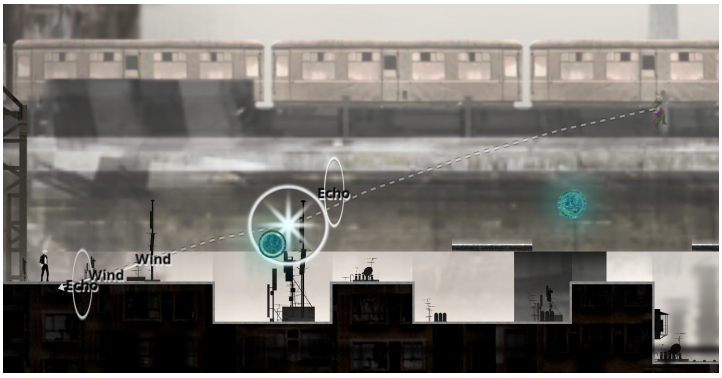

Figure 4.2: Speedrun Ghost

Ghosts are extensively used in Tutorials in order to show the player new mechanics. While a realtime animated Ghost showing the player how to execute a combination of actions is very valuable for new players, it can also contain high level gameplay information for experienced players. We are using the Ghost feature to establish developer Ghosts, that show advanced Speedruns through any level, equally to Mario Kart Wii, where competing with the developer Ghosts is a key feature in solo racing mode [17].

4.1.3 A story for Speedrunners

The story of Afterthought is fundamentally about repetition and the struggle to find meaning within it. The game is about a woman called "Black", who wakes up on a bench without memory. The whole world around her keeps changing, ever evolving, but she always wakes up on that same bench. Eventually she realizes she is stuck in a time loop and tries to seek meaning in the chaos. A fundamental question in the game is, whether the act of doing anything itself should be the reward in the end.

As the story reflects the players' states as Speedrunners, who are caught in their own repetitive gameplay loops, it will resonate with the players' feelings and keep them playing, where a linear story would probably disrupt the flow.

4.2 Game Mechanics

Before talking about all the basic game mechanics, one needs to understand, what we wanted to achieve. To be attractive to Speedrunners we wanted to build a platformer game with the mechanical complexity of Super Smash Bros. Melee [18] (short Melee) or similar fighting

games. What is so special about Melee, is not only the complexity but also the fact, that the community of the game is still active after so many years, even after there have already appeared several new titles in the franchise.

We wanted to understand, why the game is so special, and have found out that Melee players treat the characters inside the game like instruments, that need to be practiced and can be performed on stage (e.g. in tournaments). They talk about rhythm when it comes to frame perfect interactions and they talk about style, when one and the same character is played in two totally different ways. Player expression is just as important as the leverage of a player, especially when it comes to live performances. [19]

We wanted to utilize these attributes in Afterthought and simultaneously give players the chance to express themselves through gameplay. While Afterthought is in principle a single player game, it can still lead to the same hype as Melee, when two players play the same level concurrently on different machines, which can be interpreted as a live performance or a race. When we tested Afterthought on the Melee tournament Awakening 5 in 2019 near Munich, we realized we were definitely on the right way. As expected the game was very attractive to core gamers because of its mechanical complexity.

4.2.1 Basic Movement System

All moves contained in the Basic Movement System are always available to the player, while the Skills can be customized, such that you can only equip three out of the five Skills concurrently, before any level starts.

Since all moves of the Basic Movement system are always available, they can be seen as a metric, that describes how hard it is to surpass an obstacle in the game. While the Skills offer so many possibilities and interactions, that nearly any obstacle can be surpassed, the options given with the Basic Movement System are quite limited. The actions include:

- A basic Jump from the ground
- An Airjump, that can be performed once in the air
- A Walljump, while touching a wall in the air
- A Hit interaction, that can be performed on blue objects
- A Block interaction, that can be performed on red objects
- Fastfalling, which increases the players downwards momentum
- The Stomp, which is basically a downwards directed Hit interaction

2D platformer players should already be familiar with most of these actions. Jumping is the main interaction of any platformer game like one of the classics: Super Mario Bros. [20]. The mechanic is still used in New Super Mario Bros. Wii [21] 24 years later, just like the Walljump and the Stomp, which have been added to the Mario franchise to give the player other intuitive options. With the hitting, the blocking and the fastfalling mechanic we are moving away from

traditional platformer games, as we would rather find these moves in fighting games like Super Smash Bros. Melee [18]. As well as the Airjump, which can be found in the same game, these moves add some mechanical complexity to the Basic Movement System. However there are no enemies in Afterthought, so that the Hit and the Block can only be performed on static level elements, which requires less tactical decisions.

All of the basic moves are easy to learn and especially in the first half of the game they are already enough to make it through the levels, however the player will not be very fast and probably not enjoy the game as much. Naturally they can not be used to build momentum, but they can help to maintain the speed, which has been obtained differently (see 4.2.3).

By design any Skill as well as the Airjump can only be used once as long as the player is in the air, but that is where the blue and red objects come into play. If players successfully hit a blue object, they get a Reset, which means they regain the ability to perform any of the mentioned actions. The same holds for a successful block, which can be performed on red objects, with the risk of receiving damage by the object, if the block is timed wrong. For color blind people all blue objects are designed with some kind of circle overlay, while all red objects have spikes and a cross overlay as shown in figure 4.3.

These interactable objects already define a gameplay loop:

1. Use all possible actions (Airjump, Skills) to move into a desired direction
2. Reach a blue or red object
3. Perform a successful Hit or Block to gain a Reset
4. Start again with 1.

If everything is done right, the player can basically make it through the level without ever touching the ground. By design the normal walking speed of the player on the ground is the slowest way to move in the game, because we want to encourage players to stay in the air for as long as possible.

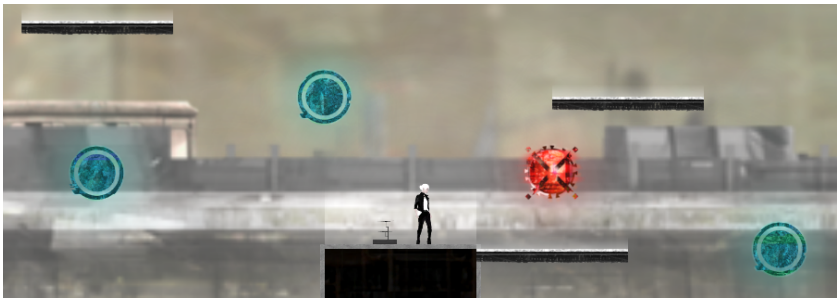

Figure 4.3: Blue and Red objects

4.2.2 Exponentially complex Skill System

Great care was put in to give every Skill multiple functions. They all have one very basic and obvious function, followed by a lot of small, very specific and sometimes hidden functions.

Players can choose three from a set of five Skills before the level starts. They are equipped with these three Skills for as long as they need to clear the level, however they can reset the level at any point and rethink their choice.

While it seems natural, that using all three Skills in combination will lead to the best outcome, it is in fact not. Even if three Skills are equipped, a player can still choose to only use one or two of them. If one Skill is used excessively, that Skill gets stronger over the duration of a level, allowing a player who uses only one Skill to clear a level as fast as a player who uses all three Skills. This way anyone can find their personal playstyle and master the Skill or combination of Skills, they are mostly interested in.

These possibilities derive from our observations of Super Smash Bros. Melee [18], where players somehow treat their characters like instruments [19]. In contrast to Melee, we do not offer different characters, but different ways of playing one and the same character, which is technically the same. All the combinations of Skills (e.g. Solo Dash or Dash combined with Bolt etc) can be seen as different sets of possibilities, which can be compared to the different characters in Melee. One can practice all these techniques or specialize in one of them.

While there is one main scoreboard for each level, there are 25 separated scoreboards for every combination of Skills, so that players can still be the best in certain categories, even if they can not beat the overall record of a level.

Dash

Dash is the most intuitive Skill of the five. The player gains a speed boost for a certain period of time, which can be aimed for precise and quick maneuvering. The reach while airborne is low, however if aimed at the floor, Dash can be converted to a slide, which adds additional functionality to the Skill. By doing this it is possible to keep up a high ground movement speed. If Dash's angle is tangential to the floor the range of the slide is increased.

When sliding off an edge in the level, the player speed before the Dash can be converted into another direction. For example if the player stomps into the ground and performs a Slide-Off, the downwards momentum of the Stomp is converted into a horizontal momentum, launching the player off the edge. Furthermore Dash offers combinations with other Skills. For example dash can pull a Bolt along with the player and cause a detonation towards the end.

Figure 4.4: Dash from air into Slide

Wind

Wind lets the player cast a wind blast, which can either push away objects or propel the player. The recoil of the player increases the closer one stands to a wall or floor. This makes Wind a good Skill for tight hallways or passages, but less powerful on an open field.

Therefore Wind has a secret feature, allowing the player to switch into a windy state, where one has less gravity for some seconds. It is activated, if the player draws a full circle with the control stick, while aiming with the Skill.

A recoil blast to the wall can be combined with a wall jump to create an even more powerful version of it. Similar to other Skills Wind can be combined. For instance along with Bolt, an aimed projectile can be fired.

Figure 4.5: Wind Propel

Bolt

Bolt places an object that detonates after a set amount of time. Resulting explosions can be used to launch the player into the air horizontally. The player can place up to two Bolts which if detonated at the same time launch the player further. If the player stands in the middle of two Bolts one will be launched vertically into the air.

Bolt's theme is raw energy and raw potential. It is less powerful when used alone but very powerful in combination with other Skills, that can lead the raw energy into certain directions. Detonations can be timed by hitting or stomping Bolts. Also contact with blue objects causes Bolts to explode immediately. These functions add even more combo options.

Figure 4.6: Bolt Launch

Echo

Echo casts a two-sided glowing object looking like a magnet. One side repels other objects, while the other boosts objects passing through it. On either side objects are always accelerated, meaning that if the player passes through or bounces off an Echo with a certain speed, one will be even faster afterwards. With these attributes Echo offers a lot of potential as a complementary support Skill.

The player is able to cast two Echoes at once which offers an additional level of complexity for setups and combinations. However Echo can as well be used as a solo Skill, if the player stomps it like a trampoline

Figure 4.7: Stomp Echo

Displace

Displace allows the player to instantaneously move from one point to another similar to a teleportation. Thin walls or corners can be passed through, enabling shortcuts. This Skill is in many ways an exception towards the rules, that other Skills follow. Amongst other things it is the only Skill that breaks the laws of physics. The player can not really build speed with Displace which dampens combo potential.

By design this Skill is the best Skill for solo usage. It has a lot of additional features, so that it can compete with the combinations of other Skills. If performed with the right timing, a player can perform a Displace Chain with up to four continuous teleportations.

After some progress in the story, the Skill gets corrupted by a wolf-like creature, which attacks the player every time, one finishes a Displace chain. This attack can be blocked in order to get a Reset and some momentum, basically allowing the player to stay in the air forever with only one Skill and without any objects around.

Figure 4.8: Teleportation chain

4.2.3 The game meta

If the gameplay loop of interacting with blue and red objects as described in 4.2.1 is performed right, the player can basically stay in the air forever. That is indeed an important part of the meta, but does not give the player a high speed.

We have split the meta of our gameplay into three important phases.

Phase 1: Launch

The first phase describes the Launch of the player at the beginning of a level. There are several Skill combinations, which can be used to launch the player off the ground similar to a rocket. These combinations are easy to handle and easy to learn. Sometimes the Launch has to be performed several times in one level, if the player fails to maintain the speed at some point.

Phase 2: Maintain

In order to maintain the speed from the launch at the beginning until the end of the level, it is important to stay in the air. As soon as the player touches the ground again, all of one's speed is lost and a new launch has to be performed. Blue and red objects play an important role during this phase of gameplay, as they let the player perform actions like Airjumps or Skills again and again. At some points the level geometry will try to interrupt the gameplay flow of the player, meaning that players do not only need to stay in the air, but at the same time they need to navigate through some kind of narrow geometry, while maintaining their speed. It

depends a lot on the level, which Skill is the best to maintain the speed at bottlenecks of the geometry.

Phase 3: Accelerate

After being able to perform the first two phases an experienced player will even want to accelerate, while keeping up a already high speed. In this phase the player tries to get Resets by blue objects and use the own Skills again and again in order to get faster with every usage. This part is the hardest part, as it requires a lot of concentration and well timed interactions, while also keeping an eye on maintaining the built up speed. For this part a very simple rule holds: The more you accelerate, the faster you will hit the next wall, if you can not evade it by some kind of maintaining technique.

Non-linear Acceleration

Acceleration has to be handled carefully codewise, as it would be overpowered, if the player was trice as fast after using a Skill for the third time. Therefore acceleration is handled in a way, that the value, which is added on your current velocity, depends on your current velocity. The essential rule here is: The faster you are, the harder it is to get even faster.

The calculation can be described by the following formula:

$$v_{new} = v_{before} + f(v_{before}) * v_{add}$$

The factor $f(v_{before})$ depends on the velocity of the player before the acceleration event. It is described by a different curve for every possible acceleration event and dampens the velocity, which is added on top of the current velocity, with a factor between 100% and 10%. This way there are never less than 10% of the additional speed added on top of the old speed, meaning that there is no top cap of the possible velocities, that can be reached. A typical curve can be seen in 4.9.

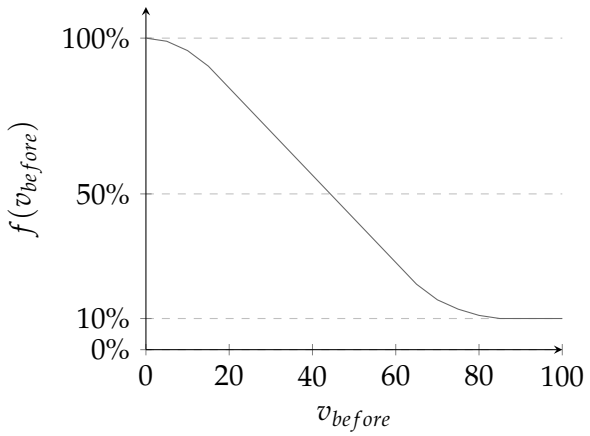

Figure 4.9: Curve of $f(v_{before})$

For example if a standing player with $v_{before} = 0$ uses Bolt once, one gets a velocity of 25 meters per second, as $v_{add} = 25$ for Bolt. If the player with now $v_{before} = 25$ uses Bolt a second time, one gets a velocity $f(v_{before}) * v_{add} = 19.25$ added on top of the speed, so that $v_{new} = 44.25$. If the Skill is used a third time, $f(v_{before}) = 49\%$ so that $v_{new} = 56.5$. When we now compare the speed of the player to the speed, if we would simply have taken the Bolt velocity times three, the impact of the non-linear acceleration becomes clear. Especially when we think of the case, that a player might be using a Skill over twenty times in one and the same level. This whole calculation makes gameplay more balanced and less exploitable. At the same time it ensures, that every combination of acceleration events is still balanced in a dynamical way. Otherwise especially Speedrunners might be able to find an overpowered combination of Skills.

4.3 Level Design based on Mechanics

Of course the game mechanics already tell us a lot about how levels are designed in Afterthought. The players need a lot of free space in the air, so that they don't get interrupted and lose all their speed upon hitting a wall. Naturally they need objects in the air, while it would not make any sense to place blue objects on the ground, as nobody needs a Reset there. The proportion of blue and red objects can already define the difficulty of a level, as well as the mean direction of a level, which can be described as a vector pointing from the start point to the end point of a level. It is much harder fighting the gravity in a vertical level, than simply moving to the right in a horizontal level.

The game is split in eight worlds with five levels each and a boss at the end of each world. New features are introduced with every new world:

1. World introduces all the basic interactions
2. World is full of fog, especially at the ground, so that the player wants to stay in the air, 45° ceilings are introduced
3. World contains blue and red arrows, that don't only give you a Reset, but also high speed in the arrow's direction
4. World contains Skill Cancel Zones, where you can not use your Skills, and Gravity Zones, that push you towards the ground
5. World contains moving objects and arrows
6. World contains Wind Zones, where the player can gain high speeds
7. World contains new objects, that are both blue and red, mean direction is top-right with few platforms on the way.
8. World contains Time Zones, where the ingame time is faster or slower, mean direction is upwards with basically no platforms

While some worlds try to hinder the player (World 2/4/8) other worlds give the player a lot of new advantages to accelerate (World 3/5/6), resulting in totally different types of gameplay.

All levels, even the frustrating ones, already have some efficient route planned by us, the developers. However it is not possible to say, that this route is really the fastest way of making it through the level. The objective of a Speedrunner is not only to find the fastest way, but sometimes to surprise the developers with new ways of playing the game.

While all of the usual levels in the game are static and forced back into their initial state upon reloading the level, the boss levels follow another scheme. The bosses try to challenge the player in a dynamic way, such that one needs to react to random actions or boss reactions to the players movements, which are also less predictable. The factor of randomness makes it hard for a Speedrunner to determine the fastest way of making it through the boss fight. Speedrunners typically talk about the "RNG" [22] (Random Number Generator) in that case and simply reload the boss fight, if they are having bad luck. It is hard to say whether the boss fights will actually excite the Speedrun community. On the other hand all of them are rewarding fast movement speed and they are an important part of the story told by the game, so they deserve to exist.

4.4 Tutorial Solutions in Afterthought

There have already been a handful of approaches on Tutorials in Afterthought, as it is hard to teach all the interacting mechanics to new players. It has been difficult to differentiate between mechanics, that are mandatory and mechanics, that should be explored by the players.

4.4.1 Sandbox with no Tutorials at all

During early development of Afterthought, we have discussed a lot about the question whether the game actually needed Tutorials or if all game mechanics could simply be explored by the players on a playful basis. While the player is already familiar with most of the basic mechanics as described in 4.2.1, it can be hard to understand unintuitive mechanics like the Skills.

At some point it was clear, that there had to be at least one short description of every Skill somewhere in the game. However the big question still remains: What is the minimum and what is the maximum set of information, that has to be taught?

At least the basic purpose of a Skill has to be shown, so that the player can start using the Skill. For example when we first showed players the Echo Skill, they were just thinking it was something like a floating platform, and they could not do anything interesting with it. When we told them, they could use it as a trampoline, they started to show interest and when we told them, it accelerates the player with every use, some started to fall in love with the Skill. The question is: Where do you stop? Should we also tell players about the possibility to use two Echoes at once? Should we show them possible setups with two Echoes? What about the limitless combinations with other Skills?

There is almost no end to what can be explained in a complex game like Afterthought. On the other hand for some features it might be better, if players discovered them on their own. It is far more rewarding, if you find something out on your own, instead of getting everything explained like in school. The game might lose some of its magic, if everything was explained in depth.

Also for Speedrunners the aspect of exploring the game has a totally different meaning. No matter how much we explain in Tutorials, the community will anyways find a new mechanic, that we would never have thought off. Therefore the sandbox aspect of the game should be kept and even promoted at some point, in order to encourage the players to find out about new ways of playing the game.

4.4.2 Many small Tutorials for every new Skill or Combo

During the early production, we started making small Tutorials on a handful of different themes:

- The Basic Movement System (without Blocking)
- Blocking and the Damage System
- General usage of Skills on the example of Dash
- Five Tutorials for the Skills (one for each)

We have split all the mechanics into many small Tutorials, because most new players were not interested in learning everything at once before using it in the game. Usually a new player would play the Basic Movement Tutorial and the Skill usage Tutorial. Some were still interested in one or two Skills or the Blocking mechanic afterwards, but most of the testers were already overwhelmed by the information overload of the first two Tutorials.

All the different Tutorials were designed pretty straight-forward, meaning they rather told the player, what inputs to execute, than the reason why. The Ghost feature as described in 4.1.2 turned out to be very handy in Tutorials. It showed the players, what buttons to press as well as the resulting player trajectory. It is very intuitive to simply press the buttons at the exact same position as the Ghost proposes. Unfortunately that approach also decreased the learning outcome a bit, as some testers were simply pressing the buttons without thinking about the logic behind these actions. They could not perform the same interaction, if there was no Ghost showing them the inputs.

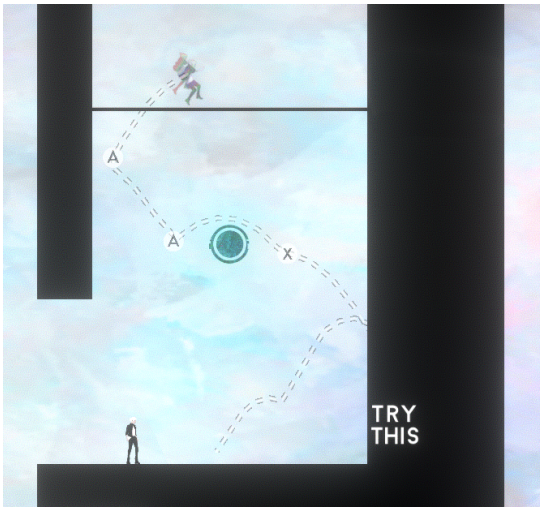

Figure 4.10: Tutorial Ghost

For that reason all of the small Tutorials
had some kind of final obstacle, where the player is not guided by the game and had to find an own way to use the newly learned techniques.

With all the mechanics split into different Tutorials we were planning on making an additional Tutorial for every combination of Skills, resulting in an enormous amount of Tutorials in the end. These Tutorials should then be unlocked at some point in the game. Some Tutorials might get available if the player reaches a certain point in the story, other Tutorials might be unlocked by beating a Developer Ghost. We also thought about making a Combination Tutorial available as soon as the players discover the combo on their own, so that one can learn more about the logic behind the combo afterwards.

However we never implemented these Combo Tutorials, as it is controversial, whether the player actually needs all these Tutorials. Aside from that we decided, that all possible combos are hidden mechanics by design. For some levels there might be Developer Ghosts, that perform combos, but the information is shown rather implicit and not explained anywhere in detail.

4.4.3 One Mandatory Tutorial

After we have built all the separate Tutorials, we have realized that the Tutorials have not been as effective. The key indicator for that problem has appeared during testing. When we watched players inside the Tutorials, they still had a lot of questions. It turned out, that we had to explain a lot of details in addition. However a video game as well as a commercial product should explain everything on its own without needing any additional information.

We came up with an ultimate test for ourselves, that every developer should probably consider. When we are now testing new Tutorials on players, we are always observing ourselves, whether we are able to stay completely silent during the entire process. Sometimes it is very hard for a developer to watch a player make wrong assumptions or perform actions in an unintended manner, especially when it happens over and over again on the same points of the Tutorial. After all designing something is always a process, and we just wanted to design the best possible Tutorial, in order to welcome new players inside the game.

In summary, we wanted to compress all the necessary mechanics in one Mandatory Tutorial, instead of many small ones. We have already been able to explain the game mechanics, but we needed to build a Tutorial, that explained the value of these mechanics. In our first Tutorials we showed to the players how to extend the time, one can stay in the air by hitting blue objects, but we didn't explain to them, that it was important to stay in the air for as long as possible. We showed the players how to use all the Skills, but we never mentioned, that they needed to be fast with these Skills. Many players intuitively just used the Skills to overcome obstacles, but they never tried to build up speed, when there were no obstacles.

At the same time it is simply not handy to have all the important mechanics split into so many different Tutorials, even when they are logically distinct.

Therefore we made a Mandatory Tutorial, which is rather designed like a challenge than like a lesson. In our opinion it now shows, what the game really is, and does not just teach the mechanics. The game is not a simple platformer, where you need to overcome obstacles,

but is rather about challenges like staying in the air or using Skill combos, in order to be fast. As well as showing what the game really is, the Tutorial should also filter out players, that are definitely not made for highly mechanical games like Afterthought.

Tutorial as a filter

One of the biggest threats to the product are bad reviews from players, who are not part of the target demographic. For example someone might see the artworks and want to play the game in order to enjoy the artstyle. Unfortunately this person might not have so much fun with the mechanics, so this person might give Afterthought a bad review. After downloading a free demo and playing the Mandatory Tutorial, most of the people in this category will hopefully realize, that they are not enjoying the highly challenging gameplay in Afterthought, and therefore not even buy the game.

Teaching the player to leave the ground

Figure 4.11: Air challenge: Together with the time counter a circle fills, while the player needs to stay in the air for ten seconds.

Many new players in Afterthought never actually tried to leave the ground. Some people were just walking through the first level and complaining about the slow walk speed of the character. They only left the ground, when they were forced to by an obstacle and they only used the Reset mechanic by blue objects, when they needed to.

The Mandatory Tutorial now challenges the player to stay in the air for ten seconds, while providing a handful of blue objects. The challenge exists in two variations. The first time the player can not use any Skills. At this point the only possible way to stay in the air, is using the Airjump again and again upon every Reset. When the same challenge occurs again, the player

can additionally use Dash, but the blue objects have a much higher distance to each other, so that if a player hits an object and gets a Reset now, one can not only use Airjump, but has to overcome the distance by using Dash (see figure 4.11). Within ten seconds the player needs to get about five Resets. Naturally the player remembers from the first air challenge, that the Airjump can also be used in order to extend the airtime. So the actions Dash, Airjump and Reset already form the basic gameplay loop as described in section 4.2.1.

We can see this challenge as the incentive challenge of the Tutorial, which has been explained in section 3.2.6. It tries to engage the player with simple mechanics, so that one can fulfill the first objective in the game.

Teaching the player to Speedrun

Although the key challenge of Afterthought is the Speedrun, it was never explained in any kind of Tutorial. After teaching new players to stay in the air, the Mandatory Tutorial challenges the player to surpass a short distance without any obstacles in under five seconds like shown in figure 4.12. A player, who is simply walking from the left to the right, has no chance to make it. If one fails, there instantly appears a tip, telling the player to "use Dash". There are no further instructions given, as the usage of the Skill itself is already the solution for a player who has not thought about using it.

Afterwards, the same challenge appears again, but now the player has one second less. This already represents the mindset of a Speedrunner: The basic goal has already been fulfilled, but can it be done a bit faster? Now the player already needs to understand, that Dash can not only be used in an arbitrary way, but efficiently. Again upon failing the challenge, there appears a tip, telling the player how to use Dash in order to perform a Slide. The player has to perform three Slides continuously with a small temporal margin, that leaves space for error.

If one has managed to perform a Slide on the flat ground, the Mandatory Tutorial teaches the player, that a Slide-Off can be performed, when there are edges provided by the geometry of a level. The player has to perform two continuous Slide-Offs in order to beat the third timer.

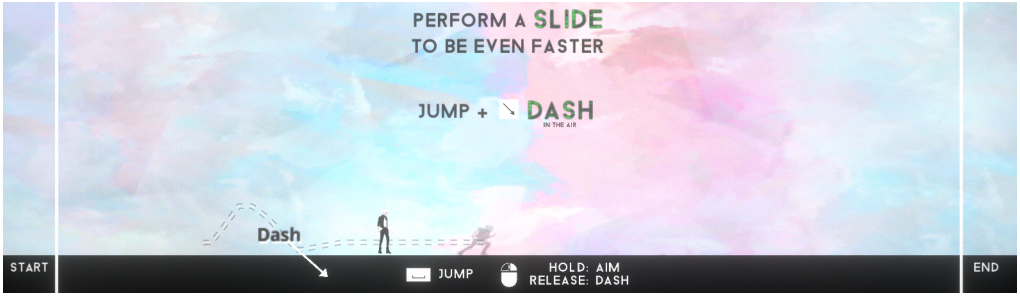

Figure 4.12: Slide challenge: If the player fails, the game provides instructions and a Ghost.

The final challenge

The final challenge is the hardest part of the Mandatory Tutorial. As well as filtering out most of the casual gamers, it also serves as a promotional tagline:

"Ninety percent of all players can't make it through our Tutorial."

Studio Moondowner, marketing tagline of Afterthought

While most people would not pay much attention to a game with this tagline, there is a certain kind of player base, that gets attracted by this, which is exactly the target demographic we want to reach.

Figure 4.13: Final challenge of the Mandatory Tutorial

During the final challenge the player needs to use all, that has been learned during the Mandatory Tutorial. At the same time it is introducing the Ghost feature as a competitive mechanic, as the player is challenged to beat a developer Ghost. The Ghost is demonstrating the path and inputs, that need to be performed. Additionally all the necessary inputs are displayed on the screen, so that the player has multiple sources of the same information.

Upon failing, the instant Reload button is introduced. All the previous challenges are taking place on rather small stages, so that one could simply walk back upon failing. On the final stage it is much more efficient and convenient to use the Reload button, so this is the perfect time and place to get players engaged with this feature, as it teaches them to save time.

As described in sections 3.2.6 we see the final challenge of the Tutorial as the provoking challenge, which dares the players to explore their limits. In the best case, a player is entering the Flow state while fulfilling the objective here, and will afterwards dive into the first level with motivation.

Tutorial conclusion

At the end of the Mandatory Tutorial there is a big text on the screen, that can be seen in figure 4.14. It serves as the conclusion of the Tutorial, telling the players, that we do not only

want them to reach the end of a level, but also to experiment and have fun while doing so, in order to explore new, creative ways of beating the game.

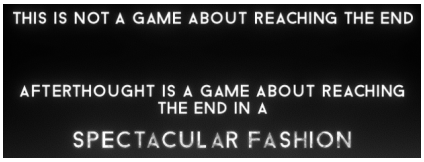

Figure 4.14: Conclusion of the Mandatory Tutorial

4.4.4 Tutorial Discussion

The Mandatory Tutorial in the beginning of the game has been tested on about hundred people and has gone through various iterations to make it as clean as possible. It is representative for all the other Tutorials in the game, which have been reworked following the same guidelines. By now all Tutorials in the game contain increasing challenges with one final challenge in the end. There are never more than two textual instructions on the screen. At some points there are Ghosts performing complex mechanics.

When we are talking about the freedom inside the Tutorials, we can say, that the player has enough freedom to perform actions wrong, but will always get noticed by the game upon doing so. One can not progress in the Tutorial without performing actions in the intended way.

The learning outcome in the end of the Tutorials is the same for every player, however some players just need much longer to clear the objectives. Especially the last challenge of the Mandatory Tutorial has proven to be extremely motivating and engaging, no matter how long a player needs to clear it.

According to the relation of Tutorials and Flow described in section 3.2 we have distributed the Tutorials across the entire game, such that they get unlocked, whenever the player reaches a certain skill level. In between the player has a lot of phases, where self-driven discovery is necessary to get further. In order to evaluate the player's skill level, we are using experience points, which are granted based on the performance in the last level. The player can get exponentially more experience points, when a level is cleared faster, so that players either have to be fast to unlock new mechanics or they have to play the same levels several times, until they reach the next experience level. This is indeed not an adaptive system, but it is at least oriented around Speedrunners.

When looking at the skill phases of the player in section 3.2.2 we can say, that an adaptive system is important during the second phase. While explicit Tutorials do also exist in this phase, an adaptive system with additional Hints can guide the phases of self-driven discovery in between the explicit Tutorials. In a challenging game like Afterthought it seems important to never leave players alone, who are overwhelmed by all the possibilities. On the other hand, there will be players, who do not need any help at all an what to discover everything on their own. Therefore the adaptive system described in section 5 is absolutely voluntary to use by anyone, who needs help.

5 Adaptive Tutoring

After we have built explicit Tutorials based on the concept of Flow, we also need to take a look at adaptive systems and how they can be used in order to improve the learning experience using positive psychology. In this context Intelligent Tutoring Systems are highly relevant applications.

In order to implement any kind of Intelligent Tutoring System, we need to keep an eye on the player, so that the individual needs of every player are fulfilled sensitively. Therefore we need to track the game mechanics and measure the performance of the player. From these values we can derive a Player Model, so that our Tutor Model can choose the right type of Tutoring, meaning the information, which the player currently needs. When everything has been measured and evaluated, we also need to present the result to the player over the User Interface.

5.1 Related Work: Intelligent Tutoring Systems

In order to follow a pedagogical approach, we also have to take a look at Intelligent Tutoring Systems (short ITS). The history of these Systems dates back decades ago, as computers have always been seen capable of improving the learning process.

> "Computer systems for intelligent tutoring are being developed to provide the same instructional advantage that a sophisticated human tutor can provide. A good private tutor understands the student and responds to the student's needs."
>
> *Anderson, Boyle and Reiser on Intelligent Tutoring Systems [23]*

The basic structure of an ITS consists of a Domain Model, a Student Model, a Tutor Model and a User Interface [24]. It can be seen in figure 5.1.

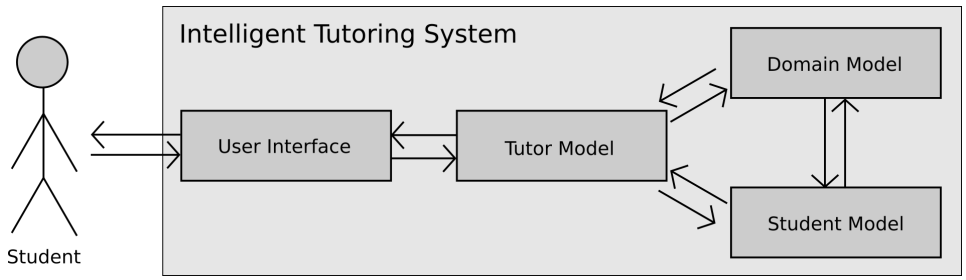

Figure 5.1: Architecture of an Intelligent Tutoring System [24]

As we are developing a skill-based video game and not always teaching knowledge to our players, we can not directly apply all of the theorems from ITSs to our game, but these theorems still serve as a different view on the topic of teaching and thereby deliver highly important information.

5.1.1 Domain Model

The Domain Model contains all constraints and all possible solutions of a problem, that has to be taught to the student. It represents the overall knowledge, that can be achieved. For a specific problem, the goal needs to be stated as well as the underlying constraints. For some goals the constraints have to be sorted by relevance, such that the student does not get feedback on constraints, that are currently irrelevant. [25]

In a video game, the game itself is the Domain Model. All the game mechanics and rules behind these mechanics together form the achievable knowledge. An individual level in the game defines the goal and the constraints, from which the challenge derives. Whereas in real world problems, there is sometimes only one possible solution to a problem, in video games, you often have infinitely many ways of making it through the game. All the knowledge from chapter 4 combined forms an outline of the Domain Model in Afterthought.

5.1.2 The Student Model

The only purpose of the student model is to define the current state of the student. The student can be analyzed on several topics, which are the knowledge, the skill level, the strategy, the motivation and other emotions. [24, 25]

In a video game we can measure some of these values, while others seem to be impossible to derive. In section 5.2 we will take a deeper look into the question how we can exactly model the player.

5.1.3 The Tutor Model

For the Tutor Model we can use techniques from the fields of Artificial Intelligence, Psychology and Education. The Tutor Model decides, how the knowledge from the Domain Model is taught based on the current state of the student derived from the Student Model. Just like a human tutor would do, the Tutor Model decides about every step of the learning process.

The possible tutoring techniques range from simple presentation through lessons to guided self-driven discovery. Every time new knowledge has been presented, the tutor somehow tests, whether the student has actually understood the knowledge, in order to evaluate the learning outcome and adjust the Tutor Model accordingly. [25]

In section 5.3 we will take a deeper look at the 5-Step Tutoring Frame, which is mostly compatible with a challenging game like Afterthought and was implemented in the context of this thesis.

5.1.4 User Interface

Like for any computer program, the information somehow has to be delivered to the student. For an ITS the User Interface is usually visible on the screen. Often the Interface makes use of textual lines, sometimes based on conversational techniques known from the fields of education.

As human tutors are so effective when accompanying a student, it is thought, that conversational interaction may be key to learning. No matter whether it is written text or a speaking agent, Artificial Intelligence can be used in the context of natural language processing.

The learning process can than be modeled as a turn-based dialogue. The artificial tutor asks a well formulated question based on pedagogical concepts, which should motivate the student to take over the conversation. Once the student has finished answering, the system is always giving feedback on the last answer and asks a followup question in a natural way, such that a conversation develops [25].

In a video game an AI talking with natural language would be a great tutor as it would not take up any space on the screen and let the player focus on gameplay. Unfortunately the field of AI is not at the point where such an AI could be integrated easily by developers with limited budget.

5.2 Tracking Game Mechanics

To define the "Student Model" known from Intelligent Tutoring Systems, we can derive player values from five categories: The knowledge, the skill level, the strategy, the motivation and other emotions. In this section we are only defining, how these values can be measured. In section 5.4 we will talk about how these values can be used to make a decision. If we want to adapt to the player and drop a precise Hint, we need to take a lot of factors into consideration, which act as constraints for the tutoring problem.

5.2.1 Knowledge

In order to define the knowledge of the player, we can simply look at the number of Tutorials, that have been completed. In Afterthought there are about ten Tutorials for all the basic game mechanics. The player unlocks new mechanics and the according Tutorials, when certain levels have been cleared. Only the first Tutorial is mandatory, while for the other ones there is a popup message, that a new mechanic has been unlocked. In the main menu the player sees an exclamation mark at the Tutorial section, in order to lead the player's attention towards a new Tutorial.

We can only conclude, that a player has achieved the knowledge, when a Tutorial has been cleared. As described in chapter 4.4.2 every Tutorial in Afterthought has some kind of final challenge at the end of the Tutorial. Only if this challenge has been mastered, the Tutorial counts as "cleared". If a player starts to play the Tutorial, but can not beat the final challenge, it only counts as "played".

Later when we display hints for the player, we could consider the difference between these two states of knowledge.

5.2.2 Skill Level

When we want to help the player, we need to measure the performance of the player. Most importantly we need to evaluate the skill level, which is probably the easiest part in competitive games. Especially in a game like Afterthought there is always a theoretically best performance. When we know how this performance can be mastered, we simply have to identify the mistakes, which have been made by a player. Depending on all the other factors we can then confront the player with the mistakes.

Speed Loss

Speed is the most important factor in a game, where you need to get from A to B as fast as possible. We therefore measure the speed consecutively throughout the entire level. This way we instantly see, when the player loses speed.

As shown in table 5.1, there are basically three ways of losing speed, while there are also several techniques, how a speed loss can be avoided.

5-Step Tutoring Frame		
	Speed Loss	Avoidance
1	Player hits Red Object	Red Objects can always be blocked
2	Player touches the ground	Stay in the air using Skills and Resets (see section 4.2.3)
3	Player bumps into wall	Use Skills, Airjump, Fastfall to adjust the trajectory

Table 5.1: Reasons for Speed Loss in Afterthought

For all of these incidents the exact time, position and amount of lost speed is saved, such that they can be displayed to the player after the level. The amount of speed loss can be evaluated in a way, that only for the type of incident with the highest speed loss amount there is a Hint displayed on the screen. For example a beginner will have a lot of problems with staying in the air and not really face the problem of bumping into walls, while an experienced player will probably only have problems with bumping into walls, because it is hard to evade them, when moving too fast.

Lost Opportunities

When a player gets a Reset in the game, every Skill can be used again, as well as the Airjump, leading to a total of four interactions, that could be performed after every Reset. Of course it is not always useful to directly use all of these abilities, but especially beginners will probably

not even think about using more then one Skill. The player loses opportunities and thereby also the potential of getting a better Score.

For the implementation one has to be careful with these values. Using 100% of all opportunities can be beneficial but using only 50% can still be enough for an optimum performance. When a player only uses below 20% of all opportunities, the information gets valuable for the Tutoring System. In section 5.5.4 we see that the visualization of lost opportunities does not work in an intuitive way.

Causal Efficiency

For many interactions in video games, we have some sort of causality, which should be satisfied. Usually when we execute an action inside a game, we want the action to result in a specific event. For these cases we can simply count the execution of the action and count how often it leads to the desired event. Upon comparing the two values, we can then determine a proportion of how efficient the player is executing the action. Sometimes there are even multiple consecutive events, which should be triggered, if the action has been performed well.

We know the most famous causal efficiency rate from shooter games. Whenever the player shoots a bullet, the bullet should hit something. In this case the rate is called accuracy.

As an example from Afterthought, we will take a closer look at the Dash Skill. The player can perform a Slide, when the Dash is pointing towards the ground. If the Slide is performed across an edge, it gets converted to a Slide-Off, which is the most desirable outcome of any Dash, as it increases the player's momentum (see figure 5.2). This means we have one action (Dash) and two resulting events (Slide and Slide-Off). In order to evaluate the efficiency we can now compare the number of Dashs to the number of Slides. If less than 80% of all Dashs are converted to a Slide, the player needs a specific Hint. If less than 80% of all Slides are converted to Slide-Offs, the Hint should be different.

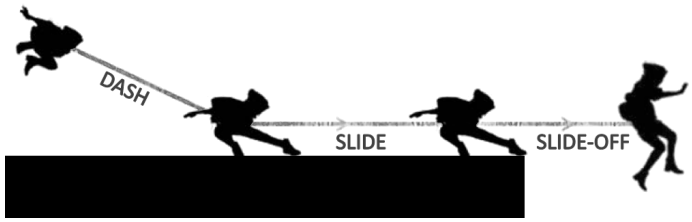

Figure 5.2: Dash converted to Slide, Slide converted to Slide-Off

Similarly we can find correlations for all Skills as well as other mechanics in the game. When we are counting every action and all resulting events in the game, our artificial Tutor can apply the rules of causality in order to formulate a Hint. In simple words: If the usage of an action does not lead to a desirable event, the efficiency can be improved.

The Launch

As derived from the game meta described in chapter 4.2.3 the player should always perform a Launch at the beginning of a level. In order to measure the execution of a Launch, we define the minimum speed value, which a player should have after any kind of Launch. Because the Launch should be executed at the beginning, we measure the time after the start of the level, until the moment, when the player is above the threshold. If a player takes too long in order to reach that threshold, we can assume, that one has not fully understood the meta yet. We can then drop a Hint, which tells the player, how to perform the Launch.

Time Rank

The most intuitive way of rating the player's performance is the Time Rank. This value might easily be forgotten, when looking at all the other values, which can be obtained. But the point is, that even if a player makes mistakes, a tremendously good Time Rank can still be reached. We don't want to bore a player, who reaches an S Rank with Hints, that an Airjump opportunity has been lost.

There might even be intended "mistakes" contained in the run, when they lead to a better outcome in the end. For example a player might bump into a wall on purpose, because a different trajectory can be obtained that way. So for each kind of Hint, we always have to define a set of Time Ranks, for which the Hint is appropriate. We do not want to bore or even insult good players and we do not want to overwhelm bad players with Hints, that ask too much of them. This corresponds to the theory, that a good Tutor or Tutorial should always adapt to the skill level, which is discussed in chapter 3.2.

5.2.3 Strategy

With strategy we mean everything, that has to be planned by the player, before a level has started. Therefore, we have to take a look at the playstyle, as well as the route, that has been chosen through the level. Unfortunately we can only make assumptions here, as we never know, that the player actually intended to play a certain strategy.

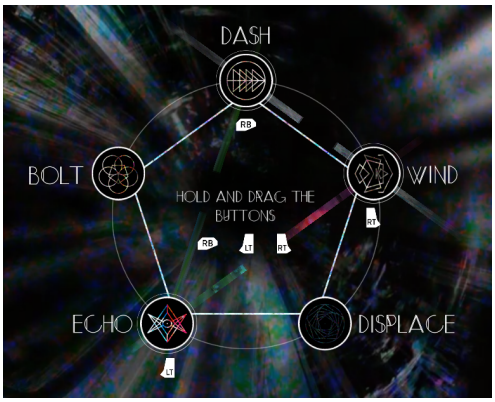

Figure 5.3: Skill Selection

Playstyle

When we look at the playstyle in Afterthought, we mainly have to look at the combination of Skills, that has been chosen. The player can equip three Skills before the level starts as shown in figure 5.3, but a Skill only counts, when it has also been used at least once. By design, the players should be free to express themselves through gameplay, so every Skill combo is a valid one in the eyes of the developers.

The game encourages the player to master challenges like "Solo Skill Runs", where only one Skill is used throughout the entire level. Also there are separate online scoreboards for every kind of Skill combination, so we have to consider the chosen playstyle, when displaying Hints to the player.

We do not want to show the player Hints on how to use other Skills, when the player has only used one Skill, as we assume that the Run was supposed to be a Solo Skill Run. For example, if one chooses to perform a Solo Dash Run, we must not tell the player, that the score could be improved, if Wind was used additionally. We should rather tell the player how to use Dash in a more advanced way.

The most specialized challenge in the game is a "No Skill Run", where the player does not use any Skills at all. Displaying Hints gets hard, if one chooses to perform this kind of run, as the player will never be quite fast without using Skills. Especially Hints, which teach how to build up momentum, will be quite useless in this case.

Route

For a Speedrunner it is always important to find the most efficient route through a game or level. Usually it is the shortest and most direct route, that can be taken through a level, whereas it is sometimes beneficial to take a longer way, when the player can achieve a higher speed there.

In Afterthought it is hard to say, which route would be the most direct one, as the Displace Skill lets you teleport through walls and breaks most of the intended routes. Using advanced techniques like shooting Bolt with a Wind, allows you to destroy objects far away from the player, which also allows to skip certain parts of the level.

When we want to measure the route efficiency we simply measure the distance, that a player is passing from the start point to the end point of the level. We compare the distance against the shortest distance, we could think of as the developers. The resulting score is rather inaccurate so we can only rate the route efficiency with values like "good", "medium" or "bad".

In order to define the shortest distance, we use a tool, that lets you paint an intended path right inside the level. This tool is also available to the players, so that they can plan their routes. As the developers we can use it, when we want to define an optimal route. After defining the route, we get the summed length of the path segments. Furthermore we have to increase the length by about 10% for evaluation, as a player will never move along the route linearly, but rather in parabolic curves.

We have to consider multiple factors, when drawing the route. If the level has choke points, where the player definitely needs to pass, the route has to go through these choke points. But when there are multiple paths, we also have to consider interactable objects or zones inside the level, that could influence the player's speed. Another important topic are shortcuts. In most levels, one will find a certain formation as depicted in figure 5.4, which is basically a wall, that can not be passed by the player at first sight. But there are two Skills, that let you pass through the shortcut. The first Skill is the Displace Skill, which would let you teleport through even thicker walls. The second Skill is the Dash, which lets you Slide through the

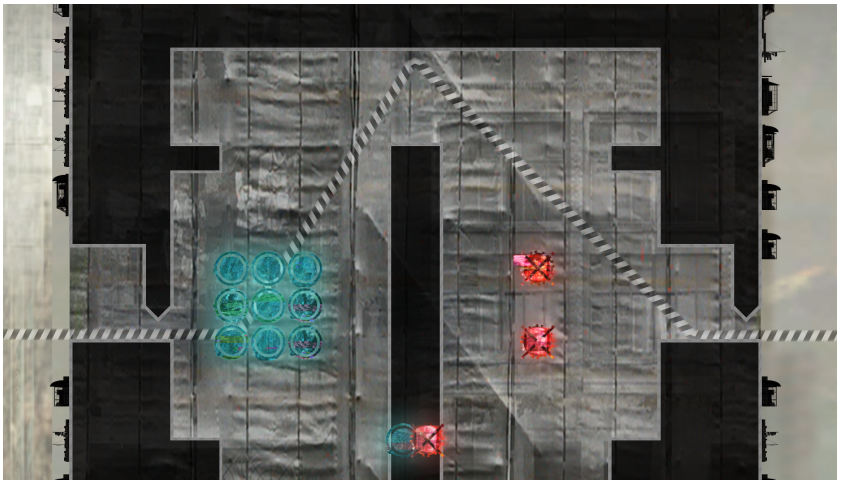

Figure 5.4: Planning an optimal route through a level segment including shortcuts

tiny hole at the bottom of the wall. When we are planning the optimal route, we always build in the shortcuts, as we assume that the player usually has at least one of the two necessary Skills equipped.

For a more advanced rating of the player's route, we could not just compare the length to our intended route, but also the segments of the route. We could even show the player our intended route. But we decided against this method, as it would restrict the creativity of the player too much. This information lies outside the the maximum set of information, which we want to present to the player.

5.2.4 Motivation

When we want to measure the motivation of the player, we can take into consideration values like the consecutive play time or the input rate inside menus. Andersen proposes to measure the return rate of players. In his studies, Andersen uses the return rate as a measure of engagement, in order to compare several types of Tutorial solutions.

> "Finally, we measured the return rate, defined as the number of times players loaded the game page in Refraction and Hello Worlds, and as the number of times the game was restarted in Foldit. While these metrics are not precisely the same as "engagement" or "learnability," we argue that they are closely related, given that players are free to leave at any time and cannot advance in a game without learning its associated rules and strategies."
>
> *Andersen et al. [2]*

When we want to evaluate the quality of a game, the overall motivation of players is a rather important metric, but it is difficult to react to the individual motivation value of a single player inside the application.

For a conversational agent it turns out useful to know the motivation, as it can embed motivational phrases into the conversation in order to encourage a discouraged player. For an agent without any kind of identity like in Afterthought (see chapter 5.5), we can not really use the information in an easy and practical way.

However we can generate motivation by declaring side goals and bonus objectives. Particularly achievements tend to occupy players for no specific reason aside from the motivation to unlock something. One could think about generic achievements with generic rewards, which adapt to the current motivation level of the player, but in the context of this thesis we are only using static and universal achievements in order to keep the player motivated. Especially in a competitive game, generic rewards might become very unfair and exploitable if the developers do not put enough time and effort into the system.

5.2.5 Emotions

Apart from motivation, we could think about measuring other emotions like happiness, sadness and anger. Equally to motivation we can not consider these values when implementing the chosen type of interface in Afterthought.

Also it gets much harder to measure emotions. We can only vaguely assume that a player is happy, when a good Score has been reached. In the future there might be AIs, which can read emotions from the user's mimics by using the webcam, but implementing this type of AI would exceed the scope of this thesis.

5.3 5-Step Tutoring Frame

The 5-Step Tutoring Frame as described by Graesser [25] fits the design of most video games pretty well. It is actually a script for a turn-based dialogue between a student and the tutor. In the case of video games the dialogue is not a conversation but rather expressed by Level Design and Gameplay, as shown in table 5.2.

Even if we do not practically have a virtual tutor with some kind of avatar, the game itself can take the role of the tutor if it gives enough feedback on good and bad performance. Gameplay can be interpreted as a dialogue between the player and the game.

In the case of Afterthought, the feedback is collected and presented at the end of the level. As it is a very fast game, it would not make sense to tell the player during gameplay, when a mistake has been made. The player would not have the time to react on that feedback and in worse cases it might even distract the player from the game. Therefore the positive as well as the negative feedback is saved until the end of the level, when the player has time to process the information. Because of this order, each turn of the dialogue takes as long as the player needs to make it through the level.

5-Step Tutoring Frame		
Step	Tutoring Theory	Afterthought
1	Tutor presents problem	Game presents new Level
2	Student gives an initial answer on the problem	First playthrough of the Level
3	Tutor gives feedback	The game gives feedback (Time Rank, Error Analysis)
4	Tutor and Student collaboratively improve answer in turn-by-turn dialogue	Repetition of the level with feedback upon every run
5	Tutor evaluates whether the Student understands	The game gives positive feedback (Time Rank, Special Effects, Unlocks) to reward the Student

Table 5.2: 5-Step Tutoring Frame from ITS theory compared to Afterthought

5.4 Implementation

When we have gathered all necessary information about the player, we can start to sort the data. In order to serve as a Tutor, we need a dynamic system, that decides about the next Hint to be displayed to the player. The system is called the "Hint Decider" and it decides about the relevance and priority of all possible Hints, which could be displayed. The decision is based on several thresholds and ratios, which have been defined through gameplay analysis.

5.4.1 Analysis Basis

For the evaluation of the gathered data, we are mainly using universal values, that can be applied to every level, instead of absolute values, that differ a lot from level to level. For instance an absolute value would be the number of Dashs, that have to be performed inside a level. Unfortunately, the value is not very useful to us, when we do not have quantitative data of many different players, who have completed the level with Dash. Note that there might exist a value x of Dashs, which is optimal for a certain level and which we could base our evaluation upon. However we would need to modify the online score system in a non-trivial way in order to seek these values efficiently with a growing player base.

One could even think about programming a Neural Network, which analyzes gameplay based on many interconnected variables, in order to evaluate the quality of the gameplay. Anyway many Neural Networks suffer from the problem, that the creators do not really understand the reasoning of the AI. It would be hard to intuitively describe to the player, how the gameplay can be improved with such an AI.

Therefore we are mostly using universal ratios, which should range around certain values independent from the level structure. In order to find out, which ratios are relevant, we have been analyzing professional gameplay, how it was intended by the developers. Many of these ratios describe a percentage on a range from 0% to 100% and can be described intuitively in

just a few words. For this reason we are even displaying some of these percentages to the player with a simple label like "Successful Block Attempts" or "Slides per Dash".

5.4.2 Hint Decider

As we can not implement an Artificial Intelligence, which uses natural language processing, in the context of this thesis, we are using predefined textual instructions named "Hints". This decision already limits the possibilities, but also determines a clear structure, which can be implemented in a convenient way.

Each Hint is part of a category, which defines the overall context of the Hint. The category also serves as a first tool for sorting, as we never want to display more than one Hint from one category at once. The Hints in one category are sorted by relevance and can be seen as sequential lessons by the Tutor. In simple words we could say: If the first Hint inside one category is triggered, the player is not ready for the second Hint inside the category.

Naturally each Hint has a condition, for which the Hint is triggered. For these conditions we obtain all kinds of values from the player and compare these values or ratios of these values with defined thresholds. These thresholds are either derived from professional gameplay or set up by logical conclusions, under the assumption that a professional player would never or only rarely fulfill one of the conditions.

After some Hints are triggered by their conditions, we need to sort the Hints by their priority. Therefore we define a priority score for each Hint, which is usually linked to the conditional values of the Hint. Usually the priority scores lie in a range between 0 and 100 as most of the scores are derived from ratios, which range from 0% to 100%. The scores are always weighted by how far away the player's value is from the desired threshold. This way the priority of crucial mistakes gets higher than the priority of small outliers.

Also we add a bias to the priority score, which is supposed to sort out the wrong audience for the Hint. The usage of biases is known from the fields of AI, where Neural Networks usually use some kind of bias, that acts as a secondary value to control the activation of Neurons. Choosing the right biases can dramatically improve the performance of a Neural Network. However, if we compare the Hint Decider in Afterthought to a Neural Network, it is only a rather simple network with a single layer of neurons and therefore no interactions between the neurons. Also the Hint Decider has never been trained with actual player data, but only adjusted by hand.

5.4.3 Audience Bias

When thinking about Hints we can easily see, that some Hints are not meant for every player. There are several Hints, which should only be viewed by a certain audience. For simplicity, we have split the audience into two groups, of which one are the "Noobs" and the other one are the "Pros". The term Noob refers to a Newbie or Newcomer, while the term Pro refers to a professional player. A Hint can therefore either be useful to Noobs or Pros.

In order to find out where the player has performed on the range from Noob to Pro, we simply use the obtained Time Rank. A player with a bad Time Rank is considered a Noob

compared to a player with a better Time Rank. Depending on the projected audience of the Hint, we add a bias to the priority score, which has been calculated, such that the Hints get weighted differently. In figure 5.5 we can see the distribution of biases. When a Hint is supposed to be shown to the player, the bias increases the priority of that Hint, while it can also decrease the priority, if a player should not see the Hint.

Note that there is a region around the C and the D Rank, where the priority of a score is not influenced much by the bias. In order to reach these Ranks, you have to do at least some things right, while there are obviously also some things, that you are still doing wrong. Especially on that skill level, every Hint can be useful to the player.

If the player reaches an S Rank, every priority score gets pushed to zero, as we assume, that a player who has performed that well, does not need any guidance. That is how we define the maximum set of information for the Tutoring of the Hint Decider.

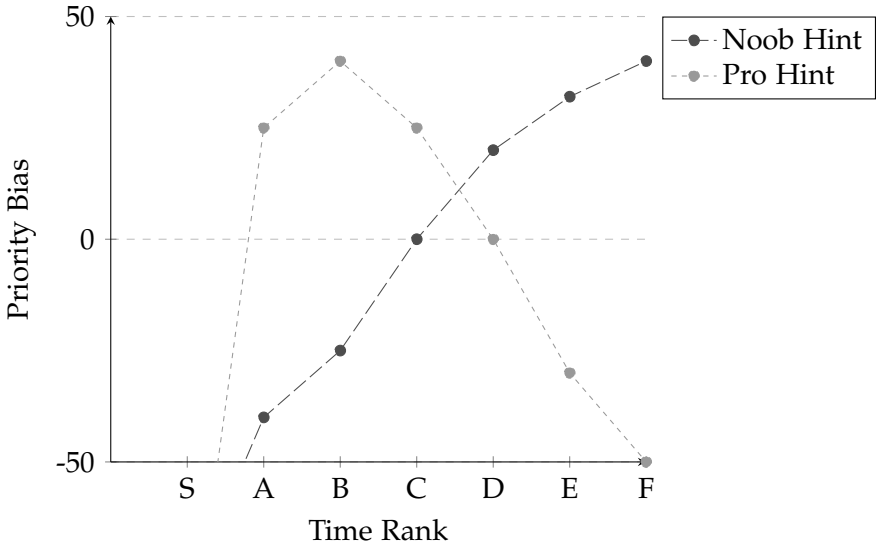

Figure 5.5: Priority Bias based on the Time Rank and the projected Hint audience

5.4.4 Hint Examples

In table 5.3 we see examples of three Hints with different properties. The Hint Decider entirely contains over 30 Hints from 14 different categories, which all follow structures similar to the examples.

The first example represents the meta of the game, which has been discussed in 4.2.3. It simply tells the player to perform the first part of the meta, which is the Launch, as fast possible. When we want to measure, whether the player is performing a Launch at the beginning of the level, we measure the time, when the player first exceeds a certain speed threshold. We call this time the Launch Time and for an experienced player it is approximately one second. Therefore the condition is triggered, when the player needs more than two seconds. The priority score of the Hint gets higher, the longer it takes the player

Hints				
Category	Condition	Priority Score	Audience	Phrase
Launch	$LaunchTime > 2s$	$LaunchTime * 10$	Noob	Try to reach a high speed as fast as possible. There are several Skills and Combos, which can Launch you at the beginning of the level.
Dash 1	$\frac{Slides}{Dashs} < 80\%$	$120 - \frac{Slides}{Dashs}$	Noob	Dash into the ground, in order to Slide. You Slide further, when your Dash starts in the air.
Dash 2	$\frac{SlideOffs}{Slides} < 80\%$	$100 - \frac{SlideOffs}{Slides}$	Pro	Try to convert all of your Slides into Slide-Offs. Always look out for an edge before you Dash into the ground.

Table 5.3: Hint Decider in Afterthought with exemplary hints

to exceed the threshold speed. If the player never exceeds the speed, the priority might get infinitely high, as the player has definitely not understood the first part of the meta.

For the second and the third example in table 5.3 we take a look at the Dash category. Most importantly both of these Hints are strategy sensitive, meaning that they will never get triggered, if the player does not use Dash during the playthrough. Both conditions are triggered by a Causal Efficiency ratio, which is described in detail in section 5.2.2.

Additionally, when we look at the two Dash examples, we can see, that the Hints are meant for different audiences. In the case that the second Dash Hint would have been triggered with a priority score of 25, when the player has obtained an E Rank, we see that the priority score is now decreased by 30 due to the Pro bias of the Hint. The result is a priority score of -5, which means that the Hint will not show up, as Hints with a score below 0 are sorted out.

Note that the player will never see both of the two Dash Hints at once. If the condition of the first Hint gets triggered, the player has not learned how to Slide properly. Therefore the information of how to perform better Slide-Offs is irrelevant to the player at that point.

5.5 User Interface to transport Information

As described in chapter 5.3 every kind of feedback is presented to the player after a level has been completed. There is a priority hierarchy for the feedback, of which some can not be skipped, while other feedback is highly voluntary. Once we have decided at what point in the game we want to give what kind of feedback to the player, we can design the User Interface.

5.5.1 Score Screen

The most obvious and also the most important type of feedback is the Time Rank, which the player gets every time after clearing a level. The Time Rank as well as the clearing time is basically everything, which counts at the end of the day, so this data is displayed on a dedicated screen, which we will call the Score Screen (see figure 5.6). It also contains a number and a rank for the Style of the playthrough, which is only a secondary unit to measure the player's performance. The Style value increases if the player performs combos and fulfills side goals (e.g. "No Damage"), but it is not nearly enough to serve as a valid source of feedback, which could be considered a Tutor.

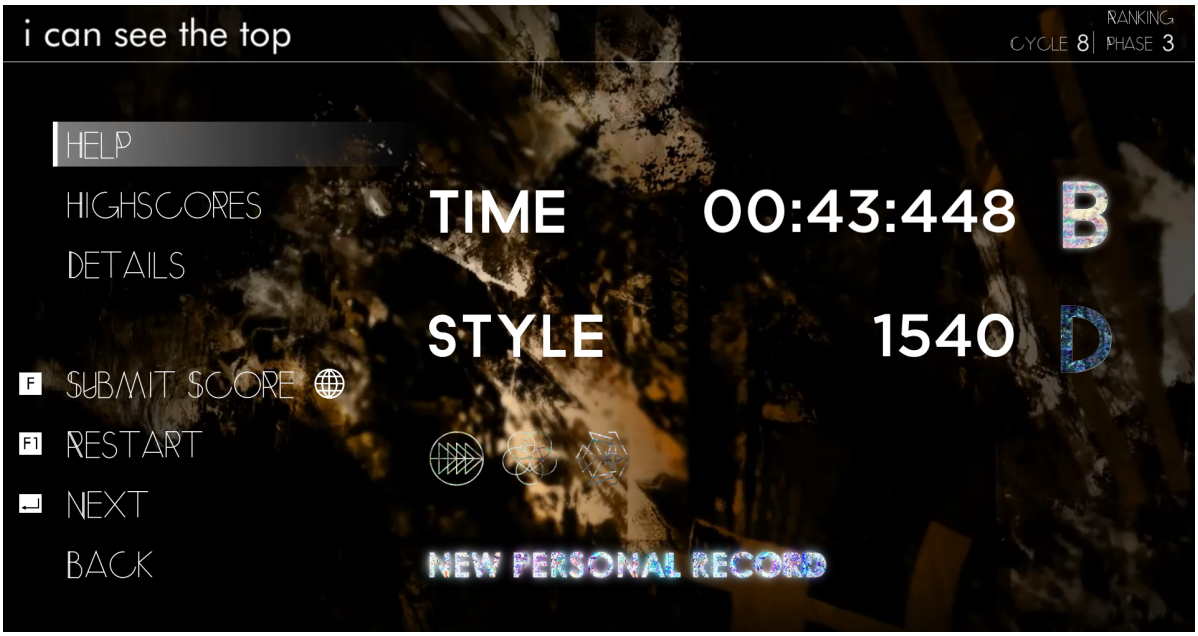

Figure 5.6: Score Screen in Afterthought

The Score Screen also contains a header for the level, which has been played. It displays the used Skill Combo and shows an indicator, when a new personal record has been achieved. As well as providing navigational options, it allows the player to directly upload the Score to the online scoreboard, in order to encourage competition. When the player selects the Time Rank as well as the Style Rank, there is an individual screen explaining, how these Ranks are calculated. The Help Screen is only accessible from the Score Screen, as it refers to the last run. When the player leaves the Score Screen, all the momentary feedback gets lost.

5.5.2 Time Rank Screen

The Time Rank screen is also very important as one can observe the own performance mapped on a time axis there, as shown in figure 5.7. It can be directly compared to the last pages of the Pokémon Manual described in section 2.2.1. It is the clearest visualization of challenge inside the game, as the player can directly see which clearing time is required in order to get

a certain Time Rank. There are several vague challenges in Afterthought like "Reach an A time in level X", which are directly translated into numbers on this screen. Even if reaching an S time is not really the skill ceiling inside the game, as one can still get a better time than that, it is indeed some kind of challenge ceiling. The players can directly see what is possible inside a level and also get a feeling of where they currently are on the range.

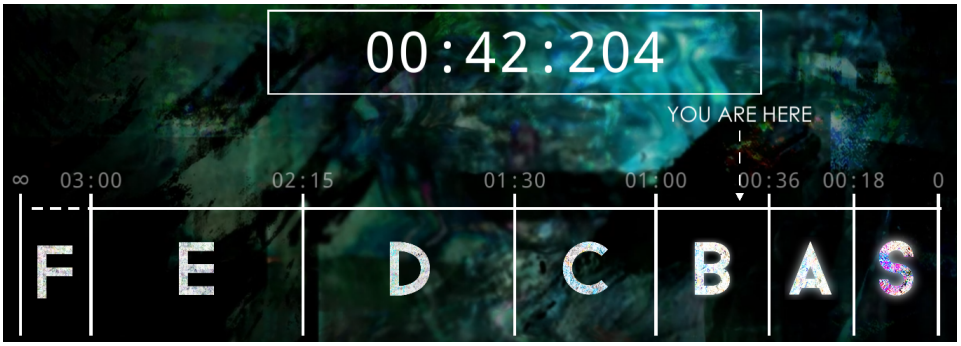

Figure 5.7: Time Rank Screen in Afterthought

5.5.3 Stat Screen

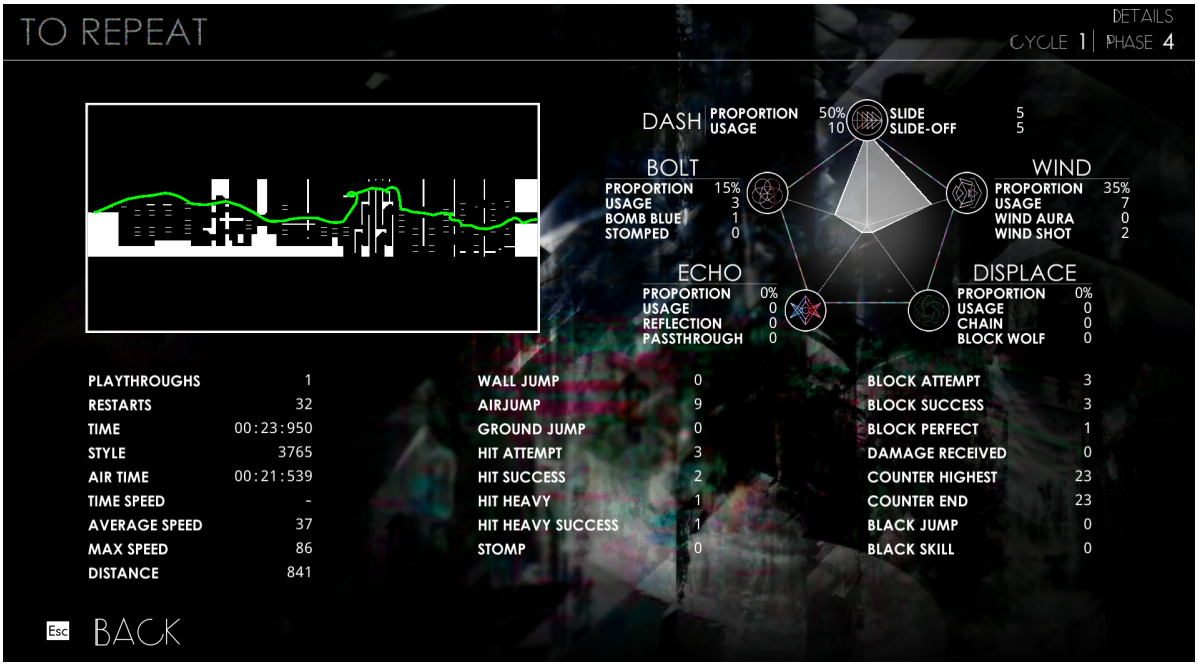

Figure 5.8: Stat Screen in Afterthought

The Stat Screen serves the purpose to show the player statistics of the previous run as shown in figure 5.8. Apart from the route, which has been taken through the level, it shows

the proportion of Skills, that have been used, and lists all the values like the number of Airjumps or the distance, which has been travelled during the run. The screen exclusively contains informational details, in order to be viewed by curious players and a similar screen exists in the main menu, listing details of all runs, which have ever been performed inside the game. This way the player can for example view the number of all Block interactions, that have ever been performed, or the average number of Attacks inside a certain level.

The difference between this screen and the Help Screen is, that the information is not viewed in context. When looking at all the numbers, one can not simply determine the mistakes, that have been made and it is rather hard to find out, what could be done better during the next run. But we are indeed using a lot of the information from the Stat Screen, when we analyze the player on the Help Screen, as well as the player can use the Stat Screen to voluntarily improve the own gameplay.

5.5.4 Help Screen

Upon implementing any kind of Tutoring System, many developers choose some kind of avatar, which is presenting the information to the user. One of the most famous tutoring avatars is "Clippy" from Microsoft Office [26], which is a friendly paperclip, that helps first-time users to interact with the program. While one can spend a lot of time and thoughts on the question, what the best design and personality for such an avatar is, in Afterthought we decided against having a tutoring avatar at all. There are about five actual characters in the entire game and one of the themes is a general lack of identity. So it might be rather disturbing to have an artificial avatar, which tells you how to play the game.

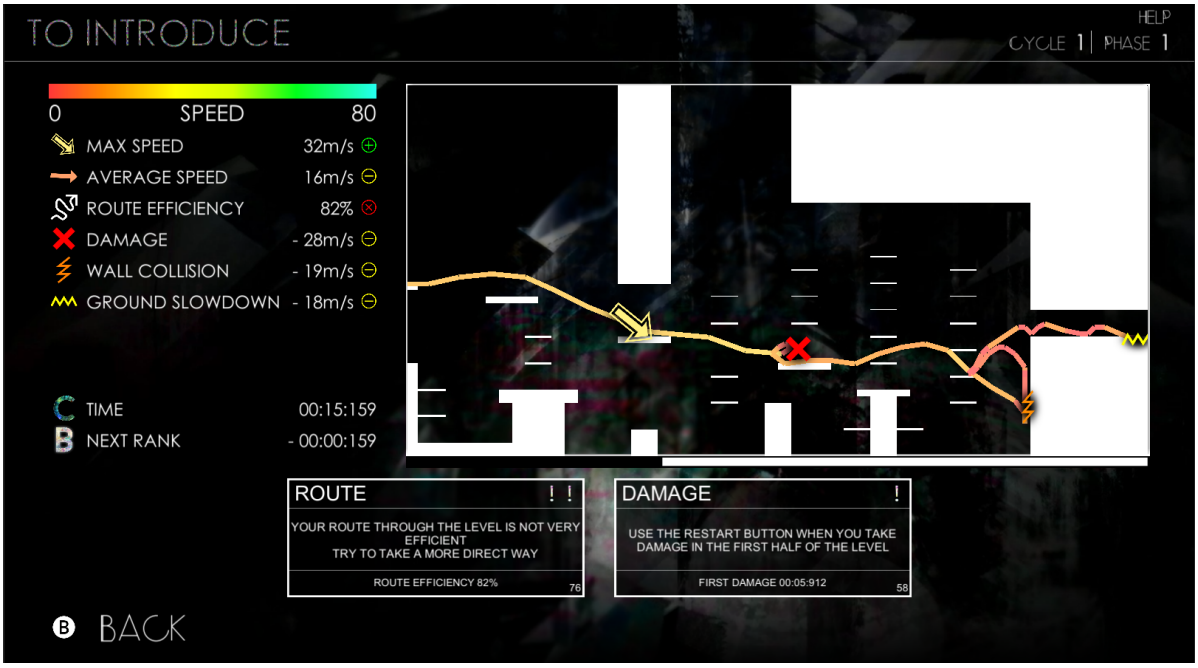

Figure 5.9: Help Screen in Afterthought

That's why we decided to transport the information through an informational screen, which presents a lot of helpful information to the user. There are textual lines to make clear, what can be improved, but the interface is not designed to hold an actual conversation with the player, like other Intelligent Tutoring Systems would do. The conversation rather takes place on a gameplay level like described in chapter 5.3.

In figure 5.9 we can see a screenshot of the help screen after a player has played level 1.1 "To Introduce" and completed the level with a C Rank. The screen is split into three regions. On the left we see general information about the run. On the right we see a scrollable view of the level with a visualization of the playthrough. On the bottom we see the Hints, that have been chosen by the Hint Decider. These three sections of information stand in strong correlation to each other. We assume that different types of players will benefit differently from the sections.

Data visualization

We could collect much more data about the player, than we could ever display, so we need to decide, which data represents the gameplay in the most intuitive and recognizable way. In Afterthought the speed of the player is the variable, which describes the performance best, which means we are displaying the overall speed performance indicated by the Maximum Speed and the Average Speed. Additionally we are displaying events, that act negatively upon the Speed, meaning events, that slow the player down, like taking damage, bumping into a wall or touching the ground. For each of these events, we can measure how much speed the player has actually lost. The performance is then rated by how crucial one of these events was.

We are also rating the stats on the Help Screen, which is a big difference to the Stat Screen, where the player could obtain the same information without a rating. The rating is indicated by a small symbol right of the value. Intuitively the symbol can be a green plus for a good performance, a yellow minus for a medium performance or a red X for a bad performance.

In order to generate motivation, we are displaying a value of how much the player would need to improve the time in order to get the next Rank.

Playthrough Visualization

When analyzing a level playthrough we always need to look at the Route, which has been taken, in order to find out at what time and place during the run the player has made a mistake. We can combine two variables, if we sample the speed of the player with a certain frequency throughout the entire level, and thereby map the speed values onto the according positions of the player. Additionally we can sample the time and position of certain other events during the playthrough, which are either beneficial or crucial.

All these variables can be plotted onto the same map and deliver a rather recognizable representation of the playthrough, which we can build upon, when tutoring the player. The location of the player throughout the run is represented as a line, that is drawn into a simplified sketch of the level. The color of the line is determined by the speed of the player

at that position and ranges from red for a slow speed and green for a fast speed. Gameplay events are drawn into the figure as symbols.

The interesting part about all this data on the same map is, that we sometimes see a drastic speed change on the line of the player. The line might be green on the left side of a gameplay event and suddenly turn red at the right side of the event. This way one can intuitively determine problematic obstacles in the level.

Also we often see loops or other kind of zigzag moves. Together with the red color of the line in these areas and possibly some negative gameplay events, we see that the player has obviously had a problem at this particular spot.

Crucial Gameplay Events

As already mentioned, there are several beneficial or crucial events, which can influence the run dramatically. Under crucial events we understand every type of event, that decreases the player's current speed. This can either happen, if the player takes damage, if the player touches the ground or if the player crashes into a wall. Whereas taking the damage is the most obvious mistake one can make, it is far harder for players to understand, that touching the ground also takes up all of their speed. Crashing into walls is the hardest event for players to prevent, especially if one is moving at a very high speed. The only things that helps here are a creative combination of Skills or Fastfalling, which can only implicitly be explained to the player, because we treat Skill combinations as hidden mechanics.

Resets

We see Resets as beneficial events, that give the player new options, as they reset all Skills of the player as well as the Airjump. For Resets it is rather interesting to look at the rate of how efficiently the resetted abilities are used. That way we could directly see points, where the player would have been able to use a Skill or Airjump before a crucial event has happened during the run. The Help Screen could directly show players, where they could have stayed in the air by making suggestions of which abilities could have been used at those positions.

Unfortunately after some testing it became rather clear that showing the position of Resets and the rate of the Reset Usage, only confused the players. In figure 5.10 we can see the Screen as it was intended to be shown. The first test players could not focus any more as the Help Screen was filled with too much information. Also the rates of the Reset usage are not very intuitive, as a rate of 50% is still very good due to several reasons, but the testers usually wanted to reach 100% for every rate. In the playthrough visualization the blue circles for the Resets might look nice at first, but they totally distract the player from the crucial events, which are far more important. Therefore the additional information has been removed from the Screen during the first iteration.

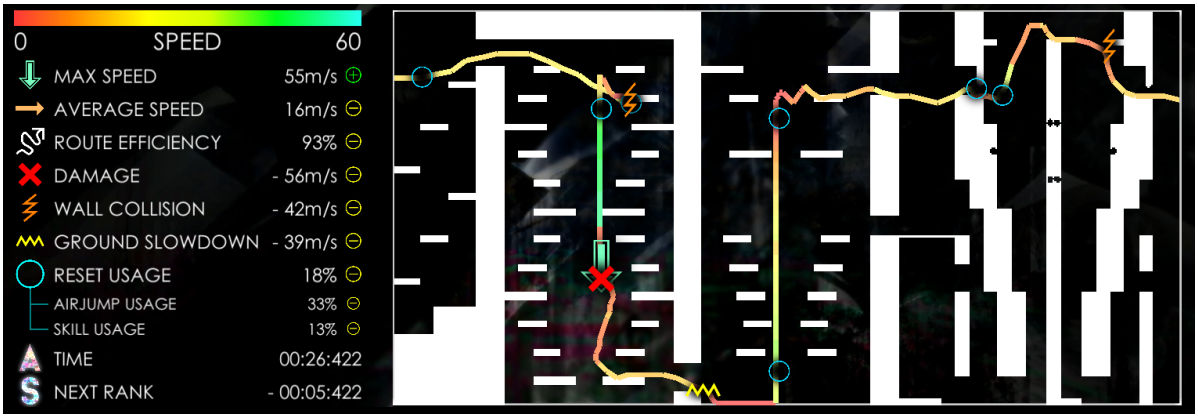

Figure 5.10: Help Screen before Reset information was removed

Hints

The Hints as described in chapter 5.4 are written down as a list sorted by their priority. For every Hint we display the category as a headline and the Hint itself as a textual instruction. Also there are some Hints, which get triggered by rather simple ratios, that can be intuitively put into words. Therefore we write down the underlying ratio for the player to understand the own performance on a mathematical level.

When the Hints have been sorted by priority, the Hint Decider displays the two most important Hints to the player, so that the player is never overwhelmed by too much negative feedback. Thereby irrelevant Hints with a low priority are eventually sorted out.

Figure 5.11: Hint visualization

In figure 5.11 we can see that the Hints have some exclamation marks in the top right corner. The number of the exclamation marks depends on the priority of the Hint, meaning a Hint of high priority will have up to three exclamation marks, while a less relevant Hint will only have one. Also the symbols are animated in order to draw the player's attention towards important Hints. In the bottom right of the Hints we see the number of the priority score, which is only displayed for debug reasons and will be removed in a future release.

There are also two cases when we do not want to show any Hints to the player. If the player performs an S Rank, we simply congratulate the player, as the player has gained the best Rank in the level. The other case is, that a player has performed a playthrough without using any Skills. A No Skill run is a special type of challenge in the game. However the player will

never be able to get a good Time Rank or gain a high speed without using any Skills. Most of the Hints tell the player to use the Skills more frequently or in a different way. Therefore we do not display any Hints to a player who is not using any Skills.

5.6 Evaluation

Before we can answer the question, whether the adaptive form of Tutoring explained in this thesis will actually be part of the final game, which is a commercial product, we need to look at the positive as well as the negative aspects of the system.

Due to the limitations of Coronavirus and a limited time for this thesis, the Help Screen as well as the Hint Decider could not be tested properly. However, it has been tested by some representatives from the target demographic, in order to gather the necessary feedback.

The testing scenario made it hard to gather valuable information about the Help Screen. As we have stated in section 3.2.2, there are three phases of skill level for the player. By design an adaptive system like the Hint Decider is not at all useful during the first phase of the skill level and it should become more useful during the second phase. However all testers of the Hint Decider have started at the bottom of the second skill phase, when testing the game, and they have never been close to the third skill phase, where the Hint Decider might have unleashed its full potential in theory. In simple words, the Help Screen can only help a player who has at least some experience with the mechanics and who has played a level for several times already. It is designed to help the player optimize the strategy and it makes good suggestions when it comes to planning the next run. Anyway for newcomers the Help Screen is very cryptic and can not suggest any valid improvements, because they are not thinking about improvements yet and only want to beat the level at first. Also the testers have not been able to play any level enough times to get a deep understanding of the route planning process.

5.6.1 Positive Feedback

For all testers, we have observed, that the Help Screen was really good at visualizing the last playthrough. Especially the scrollable level view with the player path describes the playthrough in a very intuitive and visible way. Bottlenecks and hard obstacles of the path become clear quite fast. Even if the path is not evaluated in depth, it still serves as a valid tool for self-analysis, which might especially prove useful to players in the third skill phase, when there are no Tutorials given by the game any more.

The Hints provided by the Hint Decider were not as precise as intended, however they made the testers think about their gameplay. All testers had to acknowledge, that they were having problems with the specific topics, where the Hint Decider made suggestions.

5.6.2 Negative Feedback

As we are using universal average ratios for the evaluation of the gameplay, the system works better when a lot of data has been gathered. Especially in long levels, where the player needs

to perform a lot of different interactions, the average ratios are representative. On the other hand the Hint Decider works worse in short levels, where the player sometimes only performs one of each interactions. Especially in that case the average ratio becomes useless as it is solely determined by the quality of a single action. If a player makes a single mistake in a short level, it is weighted much more than the same mistake in a long level.

Although the Hints have confronted the testers with their problems, they could not make much use of the textual instructions provided by the Hints. The problems of the testers were only rarely appearing due to a lack of information. Usually it was just too challenging to perform the inputs in the right way. Unfortunately the Help Screen is not designed to provide an immersive Tutorial experience, where the players would be able to train the mechanics. However there is a training mode for every level in Afterthought, where the player can train a lot and some of the Hints have been formulated in a way, that they point the player towards the training mechanics like the "slow mode" or the custom spawn point. It might be beneficial to set up some more Hints related to training.

Lastly there seemed to appear an overall problem with the skill estimation of the players. Many of the Hints which have been set up for the "Noob" audience as described in section 5.4.3 were actually beyond the understanding of a newcomer. By design these Hints should appear more frequently for players with an "E" and "F" time Rank, but the testing has shown that they suited players with a "C" or "D" rank far better. It is not too surprising, that splitting the audience of our Hints in to two groups has been very simplified. We now see, that at least three groups would have been far better. The third audience for Hints would be actual newcomers, who are not only uneducated about the game mechanics, but also struggle to perform the actions fast and precisely. It would either be necessary to come up with completely new Hints for this audience, or to simply not provide any Hints to these players at all, as they will clearly not benefit from suggestions as much as from actual Tutorials, which can be replayed at any time.

Although there are a lot of negative points about the Hint Decider, the testing has definitely shown, that different types of Tutoring need to be applied when the players are having different skill levels. Suggestive Hints are valuable for professional players, while newcomers need a lot more information about explicit improvements and sometimes even immersive Tutorials, where mechanics can be practiced.

5.6.3 Discussion

When we look at the suggestions by the testers we get some ideas, that might instantly improve the Hint Decider. It has turned out, that seeing the same Hints again and again is not very useful to players, that need some other advice at that point. Although the Hints which are repeatedly shown are probably the most accurate ones for the specific player, one does not feel good upon reading the same feedback every time. We would need to give a broader spectrum of feedback as we can speculate, that transformative knowledge from another Hint might also solve the problem, that the player is having.

The most simple idea is to randomize the choice of Hints a bit, so that it is less likely to get the same suggestion many times in a row. This was also linked to the idea of a tester

to additionally put all Hints into the loading screen on a completely randomized basis, like many other games do. However that could not be considered an adaptive system in any manner.

Lastly someone pointed out it would be very interesting to implement a long-term memory for the Help Screen. Currently the Help Screen does only focus on the last playthrough. But what if it could remember other playthroughs of the player and compare the results? Thereby the problem of showing the same Hint too often could also be eradicated. On top of that we could specifically mark areas inside the level, where the player is repeatedly having problems. We could give the player positive feedback on values, that have improved, and we could measure in which areas the player has not improved at all. We could use a much bigger dataset in order to analyze the player and the problem of having too few data in short levels might also be solved.

A second iteration for the Help Screen might cost some time, but the results could probably be improved, so that players would be tutored in a far more accurate way.

5.6.4 Conclusion: Commercial Value

Let us ultimately look at the question, whether the Help Screen together with the Hint Decider will be part of the commercial product. Of course after putting so much time and effort into an adaptive system like this, it would be uneconomical to discard all the features from the game. On the other hand we need to be careful, when we present a feature to the broad masses, which is obviously not fully functioning yet. While the scrollable level view only shows actual information that are always useful to the player, the Hint Decider still has some errors, and should be treated with caution. Misinformation can be even worse than no information at all, and we never want players to be confused or misinformed by the system, which happened sometimes during the testing because of the mentioned downsides.

In conclusion the Help Screen together with the level view will make it into the final game, as we have learned that the right visualization of the player data can already serve as a valdi source of self-driven improvements. But the Hint Decider will still be tested in depth and probably be modified or completely removed from the Help Screen. We are on the safe side, as long as the Hints are very general, but with the current state and without the suggested features the Hints do not adapt to the individual player as much, as it would be desirable.

6 Recapitulation

Research Question 1: *What is the role of Tutorials in the Flow theory?*

We have stated, that Tutorials needed to increase the player's skill level while confronting the player with appropriate challenges. Therefore we have modeled Tutorials as closed environments, where we have total control over the challenge and the skill level of the player. On the example of the Mandatory Tutorial in Afterthought we have explained in depth how we can teach game mechanics to players, while engaging them with increasing challenges. Especially during the final challenge of the Tutorial the player is likely to enter a Flow state, as the skill level, which is needed to clear the challenge, is equal to the minimum skill level needed for a Flow experience.

Research Question 2: *What is the optimal amount of knowledge, that should be taught inside Tutorials?*

We have proposed, that an optimal amount of information lies between an explicit minimum and an implicit maximum. The research model in section 3.2 has mapped the minimum and the maximum onto the skill axis of the player. According to our theories the minimum is closely related to the first, mandatory Tutorial, and corresponds to the minimal level of skill which is necessary in order to enter a Flow experience. The maximum depends mostly on the developers, who need to make a decision. Everything which is not taught in Tutorials is considered to be a hidden mechanic. However the existence and the interactions of hidden mechanics can be shown implicitly through unambiguous ingame feedback. Beyond the maximum set of information a skill ceiling is supposed to exist, which represents the highest possible skill level, that can be reached inside a game.

Research Question 3: *How can we tutor an experienced player?*

We expected that experienced players would need less guidance by explicit Tutorials and more guidance from implicit Hints. Anyway the Hints have turned out to be less powerful than the simple visualization of the last playthrough. We have built the Help Screen as a Tutoring tool for experienced players, which confronts players with their mistakes and improvable ratios. From testing we have seen, that inexperienced players did not benefit from the information as much as players with more practical knowledge did. The visualization of the acquired data during the last playthrough has proven to be a reliable tool for self-driven improvements. We see, that experienced players are more thankful for the raw data than for the Hints, when they are seeking for information on how to perform better. However due to organizational problems the feature could not be properly tested on professional players, for whom the system could have unleashed its full potential.

Research Question 4: *How can we utilize an adaptive system in order to tutor the player?*

We have stated, that an adaptive system could measure the player's performance in order to determine the right type of Tutoring. The Hint Decider on the Help Screen has been built as an adaptive system in the way, that it selectively chooses from a set of Hints based on the player's performance. Testing has shown, that the measurements could model the player very accurately. On the other hand in a game with so many possible interactions, we could not distinguish between the states indicated by the data very well. In some cases (especially in short levels) small outliers in the player's performance have led the Tutor in a wrong direction, such that confusion and misinformation have been spread by the Hints. In other cases the Hints were simply too general and did not adapt to the player that much. Due to several reasons the adaptive system discussed in this thesis has partially failed, however a second iteration based on the research might be able to adapt to the player in a far more reliant way. Further investigation is needed in order to optimize the system for the commercial product.

Afterthought will be released in Early-Access on 30th November 2021. Therefore there is still some time to improve the existing Tutorial solutions. Especially the Hint Decider might benefit a lot from the critical audience during the Early-Access phase. In the end a game should always be able to answer all the questions of the player on its own. Explicit Tutorials follow the purpose to explain most of the game, while other feedback inside a game lets the player draw implicit conclusions.

As developers we can utilize all kinds of techniques in order to present our game mechanics. Some types of Tutorials engage the player with our game, while other forms of Tutoring just deliver raw and precise information. So far it is not known, which way works the best. When it comes to teaching game mechanics, it seems, that we can still learn a lot.

List of Figures

List of Tables

Bibliography

[1] M. Csikszentmihalyi and R. Larson. *Flow and the foundations of positive psychology*. Vol. 10. Springer, 2014.

[2] E. Andersen, E. O'Rourke, Y.-E. Liu, R. Snider, J. Lowdermilk, D. Truong, S. Cooper, and Z. Popovic. "The impact of tutorials on games of varying complexity". In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. 2012, pp. 59–68.

[6] *A closer look at PS5's Game Help feature*. Sony. URL: https://blog.playstation.com/2021/06/11/a-closer-look-at-ps5s-game-help-feature/.

[8] *"The Legend of Zelda: Ocarina of Time Speedrun" fan-made Wiki*. URL: https://zelda.fandom.com/wiki/The_Legend_of_Zelda:_Ocarina_of_Time.

[9] *"The Legend of Zelda: Ocarina of Time Speedrun" fan-made Wiki dedicated to Speedrunning the game*. URL: https://oot-speedruns.fandom.com/wiki/OoT_Speedruns_Wikia.

[11] *Pokémon Yellow Version printed manual*. URL: https://manuall.co.uk/nintendo-gameboy-pokemon-yellow/.

[12] S. M. Harrison. "A comparison of still, animated, or nonillustrated on-line help with written or spoken instructions in a graphical user interface". In: *Proceedings of the SIGCHI conference on Human factors in computing systems*. 1995, pp. 82–89.

[13] B. Cowley, D. Charles, M. Black, and R. Hickey. "Toward an understanding of flow in video games". In: *Computers in Entertainment (CIE)* 6.2 (2008), pp. 1–27.

[14] N. A. Hanford. *Moments of Play: Uncovering the Performances of Videogame Play*. Rensselaer Polytechnic Institute, 2017.

[15] CosmoSpeedruns. *Zelda: Ocarina of Time Speedrun in 18:10 by Cosmo [commentated]*. Youtube. Aug. 1, 2014. URL: https://www.youtube.com/watch?v=aq6pGJbd6Iw.

[16] J. Newman. "Wrong Warping, Sequence Breaking, and Running through Code Systemic Contiguity and Narrative Architecture in The Legend of Zelda: Ocarina of Time Any% Speedrun". In: *Journal of the Japanese Association for Digital Humanities* 4.1 (2019), pp. 7–36.

[19] C. M. Ford. "Virtuosos on the Screen: Playing Virtual Characters Like Instruments in Competitive Super Smash Bros. Melee". In: *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*. 2017, pp. 1935–1948.

[22] M. Schmidt. *Gotta go fast: Measured rationalities and rational measurements in the context of speedrunning*. 2018.

[23] J. R. Anderson, C. F. Boyle, and B. J. Reiser. "Intelligent tutoring systems". In: *Science* 228.4698 (1985), pp. 456–462.

[24] H. S. Nwana. “Intelligent tutoring systems: an overview”. In: *Artificial Intelligence Review* 4.4 (1990), pp. 251–277.

[25] A. C. Graesser, M. W. Conley, and A. Olney. “Intelligent tutoring systems.” In: *APA educational psychology handbook, Vol 3: Application to learning and teaching.* (2012), pp. 451–473.

Ludography

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

[4] Hello Games. *No Man's Sky*. Aug. 9, 2016.

[5] Capcom. *Monster Hunter: World*. Jan. 26, 2018.

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

[10] Nintendo. *Pokémon Yellow Version*. Sept. 12, 1998.

[17] Nintendo. *Mario Kart Wii*. Apr. 10, 2008.

[18] Nintendo. *Super Smash Bros. Melee*. Nov. 21, 2001.

[20] Nintendo. *Super Mario Bros.* Sept. 13, 1985.

[21] Nintendo. *New Super Mario Bros. Wii*. Nov. 11, 2009.

[26] Microsoft. *Microsoft Office*. Nov. 19, 1990.