Tampere University

Topi Nieminen

GAMIFICATION OF A TRAINING SIMULATOR

Faculty of Information Technology and Communication Sciences
M. Sc. Thesis
June 2023

ABSTRACT

Topi Nieminen: Gamification of a training simulator
M. Sc. Thesis
Tampere University
Master’s Programme in Software Development
June 2023

Gamification has increased in popularity from 2010s onward, and can be defined as the addition of game design patterns, elements from games such as from video games, to non-game related contexts. Examples of such contexts include education and training. This thesis is a case-study that aims to offer improvement suggestions to a training simulator's educational capabilities with the use of game design patterns defined in literature. This is done by identifying game design patterns in existing simulation adjacent video games, and assess their usefulness in improving the training simulator from an educational point of view.

The training simulator in question is the tram simulator created by Creanex Oy for the Tampere Tramway Ltd.. The simulation takes place in Tampere and allows tram drivers to practice driving the tram in a virtual environment depicting the tramway routes in the city of Tampere. The virtual environment is made to be a realistic replica of the tram network, including the view of the city alongside the lines. The simulator includes traffic simulation with cars and pedestrians, as well as weather conditions that affect the driver's perception.

The thesis shows that the games analyzed do contain useful patterns, some of which are already implemented in the training simulator, but which can be improved with the ideas from the games. Patterns have been sorted into three categories: essential, recommended, and supplementary. Essential patterns, such as Clues and Postponed feedback, should be included in all training simulators. Meanwhile, recommended patterns, such as Real-time instructive feedback and Goal Indicators, are not required but do add value. Supplementary patterns can increase trainee engagement, but require more careful evaluation of return on investment. Additionally, some implementation considerations are discussed.

Key words and terms: simulation, serious games, training, education, gamification, game design patterns, game elements

Contents

1 Introduction

Gamification as a topic has gained popularity starting in 2010s as seen in internet search interest on the topic shown in Figure 1. Gamification as a term has many definitions with some nuances due to its use in many areas, such as business and education. However, it is generally accepted that the term means the act of adding elements found in games to a non-game context. The main purpose of gamification is usually to make an activity more motivating and/or enjoyable. For a business this might mean better productivity in a gamified task, while in education this usually means making learning more fun and engaging. In the realm of education, Duolingo is an example of a gamified app [Duolingo, 2023].

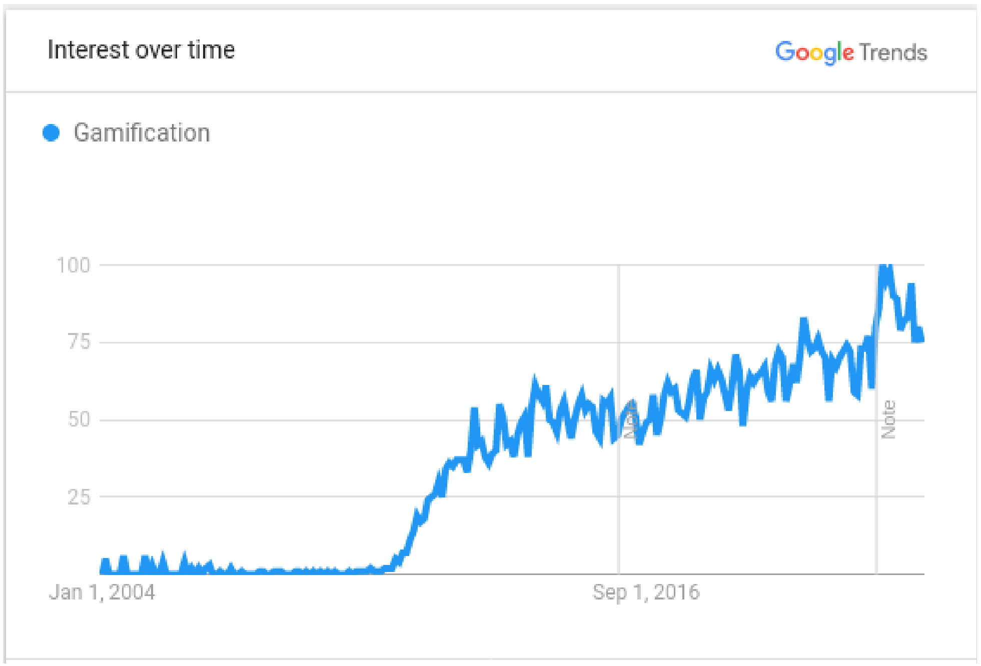


Figure 1: Worldwide search interest in the topic of gamification from 2004 [Google Trends, 2023]

A simulator is a machine made to simulate environmental and other conditions [Collins, 2023]. One use case is training people, such as training a pilot to fly a plane with the help of a flight simulator. These type of simulators can be called training simulators. The main advantage of a training simulator is that it is a safe way to practice the operation of a potentially dangerous machine. In the case of a pilot trainee, even a critical mistake in a flight simulator won’t lead into casualties caused by falling from great heights. The case study of this thesis is a tram simulator designed for tram driver trainees. For this and many other simulators, the simulation happens in a virtual world simulated by software. This includes the simulated version of the machine being trained, which the trainee can control using controls attached to the computer. By containing a virtual world, these types of simulators have common elements with video games.

The case-study training simulator in this thesis is the Tampere tram training simulator. It was released for use in 2020 and was developed in collaboration between Creanex Oy, Škoda Transtech Oy, and 3D Talo Finland Oy, with Creanex Oy being in charge of the simulation, training exercises, and hardware [Creanex Oy, 2021]. The length of the tram driver training is about six months, and includes theory lessons, teaching with the simulator, and teaching in a real tram [Creanex Oy, 2021].

This thesis aims to find ways to improve the tram training simulator with the help of gamification. More precisely, this thesis tries to answer three questions. First, does gamification offer useful elements that help improve the training simulator from the trainee’s perspective? Second, what gamification elements, if any, are already present in the training simulator? And finally, could simulation video games offer useful elements or ideas for the training simulator? The main goal of the improvements is to enhance the learning gained from the use of the simulator as a part of training. More specifically the goals are to make the trainee aware of what is expected of them, help the trainee gain mastery through repetition, and make the trainee aware of their own skill level so that the trainee doesn't have a false understanding of their skill level, be that because of too much or too little confidence. This is hoped to add to the value of the simulator from the point of view of the trainee, the training provider as a customer, as well as the simulator provider in terms of product value. The research is carried out with a qualitative analysis of three video games guided by a literature review.

The thesis is structured into seven chapters. Following the Introduction, Chapter 2 discusses the definitions of gamification, as well as some of its potential benefits and challenges in educational contexts. Chapter 3 contains an introduction and a literature review of game design patterns, which are used to analyze video games in Chapter 4. Chapter 5 introduces the case study training simulator, an overview of its software architecture, and an analysis of its game design patterns. Recommendations on how to improve the case study training simulator are presented in Chapter 6. The recommendations are based on the game design patterns found in the video game and training simulator analyses. Included are some implementation considerations, as well as discussion about the applicability of the recommendations to training simulators in general. Chapter 7 concludes the thesis.

2 Gamification

2.1 Definitions

Gamification can be defined in many ways [see e.g.: Bogost, 2011; Deterding and others, 2011; Kapp, 2012]. Here we will discuss three of them and choose one to be used for the rest of the thesis. Before trying to define gamification, defining the concept of a game can be helpful. An important distinction that should be made is that the concept of "play" is separate from "game" [Salen and Zimmerman, 2004; Deterding and others, 2011; Kapp, 2012]. Salen and Zimmerman [2004] stress that "play" as a concept should be separate from "game", but that they have a strong relationship with each other, and can be understood as subsets of each other depending on the context. Deterding and others [2011] propose that in game studies this should be understood in the way that "play" is a broader and looser category, which contains, but is different to, "game". Kapp [2012] offers the following definition for games: *"A game is a system in which players engage in an abstract challenge, defined by rules, interactivity, and feedback, that results in a quantifiable outcome often eliciting an emotional reaction."*. Kapp [2012] further defines the elements of this definition as listed below:

- A *system* is a set of interconnected elements that interact and are integrated with each other. The elements and interactions occur within the space of the game. For example, actions are limited by the rules and can affect score.
- A *player* is a person playing the game thus interacting with the game content, game elements and other players. This applies to all games ranging from board games to video games. Kapp argues that playing games often leads to learning, and this possibility can be useful in educational settings.
- *Abstractness* in games typically means the abstraction of reality which causes the game to take place in a specially defined "game space". Thus a game has elements of realistic situations or their essence, but does not copy them completely. And example for this is the game of Monopoly, which is a simplified imitation of real estate and business transactions and dealings.
- *Challenge* comes from players trying to achieve goals and outcomes. These should not be simple or straightforward, not necessarily even in games with simple rules and mechanics. Lack of, or too little, challenge makes the game boring.
- *Rules* define and provide structure to the game. They can, for example, define sequence of play, winning states, what is allowed and not allowed, fair and not fair in the bounds of the game environment.
- *Interactivity* is a large part of games. Games involve and are built around interactions of players with one another, the game system, content and other elements of game.

- *Feedback* provided to players is major and one the most important feature of games. Feedback within the game is usually instantaneous, direct and clear. Players can react to feedback by attempting corrections, changes or maintaining current behavior based on received feedback, which can be positive or negative.
- *Quantifiable outcome* means that games are designed so that the winning state is concrete. In well-designed games, a player should clearly know whether the player has won or lost the game. No ambiguity exists. A clear outcome can be defined by, for example, a score, level, or winning state such as a checkmate in chess. "Play" does not have a defined end state or quantifiable outcome, therefore making games distinct from a state of "play". Kapp suggests that this element of games make them ideal for instructional settings.
- *Emotional reactions* are typically involved in games. This can be a feeling of satisfaction from winning or completing the game, or excitement during the actual playing of the game. Or, they can be feelings of frustration, anger and sadness. Games can evoke a wide range of emotions, some of which can be very strong.

Arguably, improper level of challenge is harmful, because this may lead to the player becoming frustrated and even giving up on the challenge altogether. Games should contain enough challenge to keep the game interesting. Brathwaite and Schreiber [2008] agree with Kapp [2012] in that feedback is an important part of games. I agree with Kapp's [2012] suggestion that quantifiable outcome is ideal for instructional settings. For example, a student can clearly tell if a grade received from a course or an exam is "winning" as in passing the exam. In my view, receiving and seeing the grade is also a form of feedback, although not instantaneous. This is because the student can "react" and use it to assess their study methods and performance, and decide on whether changes are needed, just like with regular feedback defined by Kapp [2012]. This applies to games as well, when feedback is given after a game-play section or session.

Deterding and others [2011] offer the following definition: "*the use of game design elements in non-game contexts*". Same definition is used by other sources as well [Basten 2017]. Deterding and others [2011] note that although a "full-fledged" game made for non-entertainment purposes is considered a "serious game", and an application that just incorporates elements of games is called a "gamified application", the distinction between them is not always clear. This, combined with the variety of game genres, and the fact that games can be digital or not, cause challenges when defining what a game element is. As a solution, Deterding and others [2011] suggest that elements of games in the context of gamification should describe elements that are "characteristic" to games. These are elements that can be found in most, though not all, games, and are easily associated with games. They also play a role in game-play. Deterding and others [2011] recommend that the term gamification is reserved for design, and not game-based technologies or practices, such as game controllers or game authoring tools. Deterding and others [2011] identify that elements that are compatible with this definition, the game design elements, exist on multiple levels of abstraction, and argue that all of them should be included in the definition. These levels and the example elements that belong to them will be discussed later. Deterding and others [2011] believe that “gamifying” a game should instead be considered game design, and not gamification. This is because “non-gaming context” in the definition is intended to exclude the use of game design elements when designing a game.

Bogost [2011] advocates replacing the term “gamification” with “exploitationware”. This is meant to disassociate it from games, and to connect it to software fraud, such as malware, and eventually remove the practice and concept altogether. This is motivated by a concern that the implementation of low effort and nonessential game elements could exploit the user, or in a business context, the customer or employee. An example given is a loyal employee who advances the company’s goals being awarded a worthless virtual badge or points instead of real rewards, like a promotion or pay raise. Bogost [2011] asserts that games should be used for meaningful and complex things, instead of something “easy, cheap, and replicable”. This criticism is evidently directed towards gamification in a business context. Since our topic is about training and education, we will not discuss this use case in this thesis. However, I would argue that this definition and concerns do not fully apply to educational contexts. This is because even if the students receive “cheap” things such as badges, points or a high place on the leader board, they will always gain something “real”: learning. This doesn’t mean that Bogost’s [2011] concerns should be discarded in this context, and should be kept in mind. After all, gamification and even the educational aspect itself can be done poorly and therefore offer subpar value to the student.

Kapp [2012] defines gamification using the concept of a “game”: *“Gamification is using game-based mechanics, aesthetics and game thinking to engage people, motivate action, promote learning, and solve problems.”* Aesthetics here refers to elements such as the user interface, look and feel, design of the experience and engaging graphics of the gamified experience [Kapp, 2012]. This definition of aesthetics seems to be focused more on software-based solutions. Kapp [2012] argues that aesthetics is an essential part of gamification, and that gamification may not even be successful without it, because it affects how a person perceives the experience thus influencing the person’s willingness to accept gamification. Mechanics in this definition includes things like levels, earning of badges, point systems, scores, time constraints and other elements used in many games [Kapp, 2012]. The term “(game) mechanic” can refer to other things in the game design space too [see e.g.: Brathwaite and Schreiber, 2008].

Kapp’s definition of mechanics are included as a sub-category of Deterding and others’ [2011] definition of game design elements. This makes the definition of gamification offered by Deterding and others [2011] a subset of Kapp’s [2012] definition in a way, limited to mainly game design elements and not covering aesthetics and other things. However, Kapp [2012] argues that game design elements alone are insufficient to turn a boring experience in to an engaging, game-like experience, but agrees that they are vital building blocks to be used during the gamification process. We will keep this in mind to address concerns identified by Bogost [2011].

This thesis uses the definition of gamification offered by Deterding and others [2011]. This is done to limit the scope of the thesis and focus the research. Choosing this definition ignores, for example, the aesthetics aspect of gamification as mentioned by Kapp [2012]. However, the case study software we are discussing in this thesis is trying to be a realistic simulation and current aesthetics have already been found acceptable. If the training simulator would indeed be considered a full-fledged (serious) game, and therefore a “gaming context”, it would hurt the applicability of the definition by Deterding and others [2011] for our needs. However, the training simulator experience consists of many parts, as will be explained later in the thesis, so it is not exclusively a game in my opinion. Further, Kapp [2012] argues that serious games are influenced by gamification, and could be considered a sub-set of gamification.

2.2 Benefits

Feedback is a core part of games, and therefore gamification, as discussed before. Gamified learning tries to maintain positive relationship with failure by providing a rapid feedback cycle and keeping the stakes low for individual learning events [Lee & Hammer, 2011]. Kapp [2012] believes that gamification offers a safe environment that adds the opportunity of gaining experience via trial and error to the learning process, especially when learning safety topics. Arguably, virtual simulations seem to offer this benefit the best. Training to use powerful and potentially dangerous devices or machinery in a simulated virtual environment mitigates any damage that could be caused by using the “real thing” in the real world.

Gamification may be beneficial for repetitive and monotonous tasks by making them more motivating and engaging [Basten, 2017] Training, especially in the context of the usage of complex machines or vehicles, can be repetitive. This repetition can arise, for example, from the need to repeat certain maneuvers to build muscle memory, thus making the maneuvers more efficient and make it easier for the trainee to study more advanced topics like combinations of different maneuvers.

2.3 Potential challenges

Improper implementations of gamification can induce unwanted behavior. Basten [2017] suggests some ways to mitigate unwanted behavior in organizations and businesses and to encourage desired behavior. These have been summarized in Table 1.

Behavior	**Mitigation**
Gamification distracts users from the task's main purpose leading to decreased work quality and productivity.	Level of gamification provided in the software must be appropriate.
Users feel a disadvantage due to other users cheating the system. May cause decreased productivity or even rejection of gamification.	Users must not be able to easily gain rewards by cheating. Cheating may be caused, for example, by unclear rules.
Non-meaningful design such as simply awarding points only, or treating gamification as the only element that increases user acceptance and makes the system more enjoyable to use.	Meaningful design choices such as meaningful and more complex rewards, and treating gamification as only one of many elements of satisfying design.
Feeling of strict organizational control, lack of trust and privacy concerns caused by data collection and monitoring accompanied by gamification. Both data about the activity being performed and the user performing it can be collected.	Clear differentiation between public and private data, and giving the user more control over what data should be published. Data should preferably be used in an aggregated form.
Positive effects of gamification decreasing because of novelty wearing off, or tasks being perceived as too simple as the user's skills improve. Removal of game elements may cause user performance to decline dramatically.	Updates such as new challenges in the long term. Not removing game elements abruptly.

Table 1: Mitigation of unwanted behaviors caused by gamification. Summarized from: Basten [2017].

While these suggestions focus on business and organizational contexts, most of them apply to training and educational contexts as well. Distraction from the task's purpose, which in our case is learning and improving skills caused by inappropriate amount of gamification should be taken into account. Too much gamification might make the activity an actual game rather than a learning tool. Being rewarded for cheating is an unwanted behavior in any context. In training, this might lead to skills not being acquired or, even worse, learning to do things the wrong way. If the training has a competitive element such as a leader board, easy rewards by cheating further encourages the behavior from others and likely frustrates those who don't cheat. Before adding gamification to a training, the goals of the training should be decided first. User data collection from software training solutions that contain gamification is a useful tool for educators. But, both users and their data must be respected even in training contexts. The need for long term motivational solutions such as adding new challenges may be less important in the training context we are discussing in this thesis. Having appropriate challenges for the trainee for all levels of skill they may reach during the training process is most likely good enough.

Kapp [2012] asserts that implementations that only use "badges, points and rewards" and other less useful or “exciting” elements of games should not be considered gamification at all. Kapp [2012] argues that gamification should instead be built on more complex elements of games such as engagement, storytelling, interactivity and problem solving. Kapp [2012] recommends that entire experience of the learner is considered, and that all elements work together. Bogost’s [2011] comments indicate an agreement with Kapp [2012]. This would imply that Bogost’s [2011] concerns can be addressed by following the suggestions proposed by Kapp [2012].

GDP Wiki [2023] identifies a separate category for game elements, also known as game design patterns, called called “negative patterns”, which game designers should in general avoid, or at the very least limit, when designing a game. Often referred to as ‘anti-patterns’ in other pattern collections, these patterns generally cause the player to subjectively experience negative emotions related to the game [GDP Wiki, 2023]. However, GDP Wiki [2023] argues that some of the patterns of this type may be useful in certain situations in carefully measured ‘doses’. Examples of negative patterns include unwinnable game states, repetitive game-play, non-consistent narration, and such. Since we are focusing on beneficial patterns, this category will not be the focus of the research and thus will not be focused on further, though the patterns are good to keep in mind in general.

3 Game design patterns

This chapter introduces relevant game design patterns. Björk and Holopainen [2005] define game design patterns as parts of game-play, the "core aspect" of games. Other names include atoms [Brathwaite and Schreiber, 2008], game design elements [Deterding and others, 2011], and mechanics or elements [Kapp, 2012]. Due to the number of game design patterns that have been defined in literature, choosing a way to categorize them for analysis is reasonable. One categorization, or "levels of abstraction", is provided by Deterding and others [2011]. The five levels, ordered from concrete to more abstract, are: "interface design patterns"; "game design patterns" or "game mechanics"; "design principles, heuristics or 'lenses'"; "conceptual models of game design units", and finally, "game design methods and design processes" [Deterding and others, 2011]. The more concrete first and second levels contain patterns such as badges, levels, and time constraints [Deterding and others, 2011]. These two categories are more interesting for our purposes because they are something that can be more readily added to an existing application.

The patterns in this chapter use the categories described by Björk and Holopainen [2005]. We use this categorization system because the number of patterns discussed here is quite small, and this system is adequate for our needs. Other categorization systems for game design patterns exist. One such example is proposed by the GDP Wiki [2023], which expands the original categorization system by Björk and Holopainen [2005] with new categories, and identifies that the categories are overlapping such that a pattern can indeed belong to more than one category. Thus category membership, according to GDP wiki [2023], could be considered as a form of tagging. This categorization could arguably be more useful when there are a large amount of patterns. Another categorization system is the Mechanics-Dynamics-Aesthetics (MDA) framework proposed by Hunicke and others [2004], which sorts game design patterns into three main categories. This system seems to have very few example patterns for the categories and would thus increase the categorization effort in this chapter.

Kapp [2012] does not define categories per say, but does however identify patterns that exist in all games, such as goals, rules, and feedback. Since they exist in all games, and are therefore essential, there is little reason to individually state that they exist in every game we are analyzing, especially if the patterns are abstract like "rules". Another example that applies to our case could be the pattern "game world". Björk and Holopainen [2005] define the pattern as "The environment in which the gameplay or parts of the gameplay takes place is determined by the spatial relationships of the game elements". GDP Wiki [2023] renames the pattern "game worlds", and gives an "updated version" of the definition: "Fictional worlds in which gameplay takes place". GDP Wiki

[2023] points out that with this definition, not all games have or need game worlds. One of the examples given is Rock-paper-scissors. Evidently, simulations, which includes the games and the training simulator we are going to analyze and are concerned with, require a game world in which the simulation takes place. Some patterns, such as “feed-back”, we will split into more than one pattern to help the analyses.

Category	**Patterns**
Game elements	Avatars; Clues; Enemies; Ghosts; High Score Lists; Levels; Obstacles; Save points; Score
Resource and resource management	
Information, communication, and presentation	Feedback (real-time instructive and post-poned)**[1]**; Goal Indicators; HUD inter-faces**[1]**; Loading hints**[2]**; Mini-maps; Progress Indicators
Actions and events	Reward structures; Save-load-cycles
Narrative structures, Predictability, and Immersion Patterns	Character development; Storytelling
Social interaction	
Goals	Goal
Goal structures	
Game sessions	Game Pauses; Time**[1]**
Game mastery and balancing	Difficulty levels**[1]**; Replay**[1]**
Meta Games, Replayability, and Learning Curves	Achievements**[2]**

Table 2: Placement of game design patterns in the categories proposed by Björk and Holopainen [2005]. Patterns categorized without further explanation are categorized by Björk and Holopainen [2005]. Other categorization reasoning: **[1]** Similar to other patterns in this category. **[2]** Compatible or similar category found in GDP Wiki [2023].

Table 2 shows at a glance which patterns are included, and which pattern is placed into which category. The categories form the sections for this chapter, and a definition is provided for each category and pattern. The patterns are introduced and defined in the section titled after their category.

Since the GDP Wiki [2023] alone lists over 600 patterns, some curating had to be done. In general, we focus on introducing patterns that appear in games that we will analyze in Chapter 4. Another consideration to try to limit the number of patterns discussed here is to focus on patterns most detectable by the player, or patterns that, in some other way, affect the game-play experience the most.

Something that should be mentioned is that the GDP Wiki is maintained by Björk and Holopainen, the authors of the book Patterns in Game Design (2005) [GDP Wiki, 2023]. It seems that most, if not all, content in the book is included in the wiki. Some pattern entries in the wiki have a note of being a “rewrite” or an “updated version” of the pattern in the book. Because the content is mostly the same, citations to the wiki and the book are used somewhat interchangeably.

3.1 Game elements

Björk and Holopainen [2005] define this category as patterns that represent in-game objects that the player can manipulate and that have characteristics that are used to define the area of the game reality. Included in this category are “abstract objects”, which may or may not have a concrete manifestations in the game world, but represent abstract abstract values associated with game states. One example of these is the “score” pattern.

Avatars are game elements that allow the player and computer controlled entities to affect the game world [Björk & Holopainen, 2005]. When present, the avatar is usually the only way the player can affect the game world. Avatars is a common game design pattern. Usually the players control their own avatar and make actions through them such as jumping on a button to open a door in a platform game. Avatars represent the player and those that exist in non-digital games may have different terminology such as “tokens” or “pawns” [Brathwaite and Schreiber, 2008].

Clues are game elements or information that provide help and instructions for players on how to complete goals [GDP wiki, 2023]. Clues may be categorized as explicit in that they describe the exact way a goal is reached, or they can be implicit and therefore describing facts and events in the game world and require interpretation by the players. The vagueness of clues can vary therefore making this categorization more of a spectrum. Some games are designed so that finding out what the goals are is part of the gameplay and thus contain little to no clues. Examples of clues include Non-Player characters instructing the player via dialogue, as well as simple signs in the environment that point out where certain locations are in the game world. Clues can also communicate advice, encouragements, and warnings [GDP Wiki, 2023]. Encouragements include near miss indicators, which are shown when the player started, or tried, doing correct actions but failed to execute them correctly. We will call these types of clues as “real-time instructive feedback” defined in Section 3.3. Clues can also be in the game world itself as objects or in the UI as static elements. Arguably, clues that are communicated in response to player actions could be considered a form of *feedback*.

Enemies are game elements, such as avatars and other units, that actively hinder the player while they are trying to achieve and complete goals in the game [GDP Wiki, 2023]. When designing an enemy, how to overcome or evade them should also be defined. Overcoming them can mean eliminating them. It should also be decided where in the level they appear, for example blocking a path.

Ghosts is a pattern that is used to show actions from previous game sessions and help the player compare their current progress with other people, or their previous attempts [GDP Wiki, 2023]. Ghosts are often found in racing games where the player can race against their previous attempt. Ghosts are often shown as a translucent and immaterial avatar that the player avatar can pass through. Some games allow the player to practice, and compare, against the optimal path [GDP Wiki, 2023]. Some Ghosts implementations could be seen as an instance of the High Score Lists pattern, because it records players attempts and allows comparison [GDP Wiki, 2023].

High Score Lists [Björk and Holopainen, 2005; GDP Wiki, 2023], or leaderboards [Kapp, 2012], are a list of top scores, or results, players have achieved in the game. The score pattern is explained later in this section. The High Score Lists pattern allows the player to compare their performance in relation to others [Kapp, 2012; GDP Wiki, 2023], or their previous attempts [GDP Wiki, 2023]. Kapp [2012] sees that the pattern works as a motivator to play the game repeatedly, and gives bragging rights and social capital to the individuals who achieved the highest scores. This fact qualifies the pattern as a form of Reward Structure found in Section 3.4.

Levels are part of the game world where all player actions happen [Björk & Holopainen, 2005]. Levels are completed by reaching a certain goal or when an end condition is fulfilled [Kapp, 2012]. Kapp [2012] writes that sequential levels build and enforce skills. This is done by introducing new techniques or abilities level by level starting with the basics at early levels. Some levels can also simply offer a chance to practice skills acquired earlier without introducing new skills or information. Basten [2017] suggests that sequential game levels of increasing difficulty or complexity overall improve usability by introducing new features gradually. Kapp [2012] suggests that well-designed levels also aid the game's storytelling by revealing new information each level compelling the player to continue to see the conclusion of the narrative. Kapp [2012] also writes that often more advanced levels require the player to perform skills more quickly or under pressure to increase challenge. We will address this more in Section 3.9 when discussing the *time* game design pattern.

Obstacles are defined by GDP Wiki [2023] as “game elements that hinder players from taking the shortest route between two places in game worlds”. GDP Wiki [2023] remarks that overcoming unnecessary obstacles can be considered as one of the definitions of a game. Often, obstacles in games will slow or block the player’s progress in the game until they are moved, destroyed, or avoided by performing specific actions. This definition seems to be quite broad, since it includes even walls in the game world as an example [GDP Wiki, 2023]. Evidently, by this definition, the obstacles pattern is an essential pattern that is found in all games, at least those that have a game world. Interestingly, this pattern definition is quite similar to the enemies pattern definition. The key difference here is that enemies hinder the player actively.

Save points are locations in the game world where players can save the game state [GDP Wiki, 2023]. The saving can be initiated manually by the player, or the game can be made to do it automatically. Another consideration is whether to allow the player to save the state at any location in the game world, or only in places determined by the game designer. The Save points pattern is used with the Save-Load Cycles pattern, which is presented in Section 3.4.

Score, or *points* [Kapp, 2012], is used to numerically represent the player’s success in a game [Björk & Holopainen, 2005]. It allows the player to compare performance against other players or their previous attempts. When creating a score system it should be determined which actions or goals give score points and how much. Kapp [2012] suggests that the score pattern could be used to enforce the rules of the game. Björk and Holopainen [2005] state that the typical way players can compare score between attempts is the use of a *high score list* pattern. In video games score is often visible in the user interface as a game state overview, or extra information that extends information provided by game elements.

3.2 Resource and resource management

Patterns in this category are related to defining resources in games, and how their flow within and outside the game can be controlled [Björk and Holopainen, 2005]. An example is the Resources pattern, which describes “commodity” game elements that the player uses to enable actions in games. These include health, ammunition and money found in many games. It is worth investigating if and how much the patterns in this category bring value to educational contexts. Could student motivation be increased by awarding currency to the student for doing well, where the currency can be spent on decorating a public profile page for bragging rights? Perhaps. Patterns in this category won’t be discussed explicitly in this thesis. Instead, we will take a look at Reward Structures pattern discussed in Section 3.4, which is somewhat similar.

3.3 Information, communication, and presentation

This category contains patterns that focus on how information about the game state is presented to or hidden from the player [Björk and Holopainen, 2005]. Arguably, patterns that reveal something about the game state can make the game easier for the player. Hiding the state, on the other hand, could then make the game more difficult. I see that these patterns can be used when implementing the Difficulty Levels pattern presented in Section 3.10.

Feedback is a foundational feature of games that is almost constantly present [Kapp, 2012]. In video games the player receives real-time feedback on the state of the game, such as progress towards the goal or amount of resources, and especially in learning games the feedback is often designed to encourage correct behavior and actions. Feedback can also guide the player towards proper outcome with the information it provides. Brathwaite and Schreiber [2008] clarify that feedback can be visual, auditory or tactile, and common ways of providing visual feedback is in the user interface (UI), such as the HUD, including mini-maps of the surrounding area and the use of icons instead of text-based descriptions. HUD will be explained later in this section. Kapp [2012] notes that designers strive for something called “juiciness” when creating feedback. “Juicy” is a term used in game design that refers to “effective, exciting, and engaging feedback”. Juiciness certainly seems like something players would appreciate. However, it adds another level of analysis to game design patterns, so we will skip it for this thesis. Because feedback is a foundational element and present in all games, for the purposes of our analysis, we will concentrate on patterns derived from it that we will call **real-time instructive feedback** and **postponed feedback**. These names are references to the timing when the player receives the feedback in the game. More specifically, we will call **real-time instructive feedback** the kind of feedback the player receives during the game-play that explicitly and clearly informs the player of their mistakes or successes. Near miss indicators mentioned earlier fit this definition. An example of this pattern would be text boxes that appear in the HUD in reaction to player actions. The text in the box could then describe what the player did that warranted the box appearing. Icons might work as well if they are informative enough. Combining the two might be more viable. **Postponed feedback** is simply feedback that is given to the player after completing or failing a part of the game. These parts can be, for example, levels, challenges, or sometimes goals. An example of this would be an indicator on how close the player was to the goal, when the player’s distance from the goal is not revealed during the game-play. Another example would be a summary of the player’s accomplishments in the level, like the time it took for the player to complete the level.

Goal Indicators pattern provides information to the player about their current goals, which helps the player keep track of them [GDP Wiki, 2023]. The pattern can also help the player estimate how close they are to reaching the goals. One way to create a Goal Indicator is with the Progress Indicator pattern. Narration can also be used. We will focus on Progress Indicators, which will be introduced later in this section. Goal Indicators generally prevent players from defining and carrying out their own-made goals [GDP Wiki, 2023]. Arguably, this is good in educational contexts, because this means that the focus is on the educator-defined goals.

HUD interfaces are visual elements that are presented in a video game as if they were in a plane in front of the game world [GDP wiki, 2023]. They can be combined with game state indicators to provide information about the game state to the player. Example of this pattern is weapon ammo count shown at all times in a corner of the screen in first person shooter games. GDP Wiki [2023] notes that this definition is intended to be more generic than a HUD [see e.g.: Wikipedia, 2023a], but we will simply use "HUD" to refer to this pattern. Brathwaite and Schreiber [2008] suggest the following considerations to make information presented to the player, for example via the HUD, easier to access:

- Grouping several pieces of information that relate to each other and placing them near each other on the screen. For example, if magic points are used to heal hit points (health), or are otherwise equally important, their indicators should be placed so that the player can get all relevant information at a single glance.
- If there are events in the game that the player needs to know about, they must be indicated in some way too. An example would be to make the screen flash red, play a sound, make the controller vibrate, or combine all three, when the player takes damage from a hit.

Loading hints is a game design pattern in which hints about the game or game-play are presented to the player during loading sequences [GDP wiki, 2023]. Hints may contain information such as game controls, explanations on how some abilities work and so on. Loading sequences can be triggered when the game needs to halt the game-play to load data, for example, when the player enters a new level or world.

Mini-maps are elements in the user interface that provide an overview of the game world, for example to help the players to know their own or friendly players' location in the game world [GDP Wiki, 2023]. The pattern is an instance of the "game state overview" pattern [Björk and Holopainen, 2005]. The mini-map is separate from the primary representation of the game world, and can thus show a larger area of the game world, or maps of locations different to the player's location [GDP Wiki, 2023]. Mini-maps are a compact representation of the game world, so they are often used in games where the game world, or level being played, is too wast to be seen all at once. Mini-maps are very common in racing games, in which they can show the race track in small form. Many open-world games may also have a mini-maps. Mini-maps can show the locations of objectives, allies, and locations, among other things, on the map.

Progress Indicators are presentations outside the game world, such as in the HUD, that provide information to the player about their current progress towards “closures”, such as goals [GDP Wiki, 2023]. The pattern can be used, for example, to indicate how close the player is to completing a Combo, or a competition. Racing games sometimes have maps that indicate how far the player has traveled. A “negative Progress Indicator” can, for example, show how much time is left before a negative event occurs, possibly motivating the player to work faster to prevent it. We will focus on this pattern as the way to create Goal Indicators.

3.4 Actions and events

Most of the patterns in this category describe what kinds of actions are available to the player, and how they change the game state, or progress towards goals [Björk and Holopainen, 2005].

Reward structures, including badges, points and other rewards, can be used to help motivate the player to play the game [Kapp, 2012]. In general, rewards are positively perceived things that the player can receive, usually as encouragement for game progression, or for other changes in the game state [GDP Wiki, 2023]. Kapp [2012] argues that reward structures are a very important part of games. But they shouldn't be the focus of a gamification process as argued by both Bogost [2011] and Kapp [2012]. Kapp [2012] explains that there are two views on how to give out rewards to players. The first one suggests making rewards as easy as possible to get so that players get hooked and thus keep playing. The latter advocates not awarding easy rewards for activities that are by themselves rewarding. Kapp [2012] argues that rewards linked to activities within the game should be used over random rewards.

Save-load-cycles: In some video games, players can save the game state during game-play to disk. The saved game state can then later be loaded back by the game to return to the saved state and continue game-play from that point. These actions of saving and loading are called *save-load cycles* [Björk & Holopainen, 2005]. *Save-load cycles* allow players to return to different saved states in the game and thus replay them as many times as they wish. This can be done, for example, to successfully complete actions or to try to perfect how the actions are performed. Save-load cycles also allow player experimenting. Arguably, this pattern could be used in combination with or as a part of the **Replay** pattern, which will be discussed in Section 3.10.

3.5 Narrative structures, Predictability, and Immersion Patterns

These patterns here are designed to support the player’s immersion into and commitment to the game [Björk and Holopainen, 2005]. Arguably, commitment is something to pursue for educational contexts.

Character development describes the changes in player character's abilities, skills, or powers as an element of game-play [GDP wiki, 2023]. For our purposes, we combine the patterns **player levels** [Kapp, 2012], ***c*haracter levels** [GDP wiki, 2023] and **character development** [GDP wiki, 2023] as one pattern. This is done because of the patterns' close similarity with each other and to simplify game analysis. Player levels are often awarded for making progress in the game, such as by completing levels or otherwise gaining experience [Kapp, 2012]. Progression is usually measured, especially in role-playing games, using 'experience points' that can be awarded from completion of quests, overcoming obstacles and enemies, or for completing levels. Experience points are usually accumulated during the game and can give the player access to new features, such as special abilities or valuable items. This kind of progression can also be called character development, and usually makes the character under the player's control more powerful, such as by making the player more likely to succeed [GDP wiki, 2023]. Character development and experience points can be expressed with character levels, which indicates to the player their progression, such as a numerical value [GDP wiki, 2023]. Kapp [2012] suggests that gaining of player levels can bring the feeling of mastery and accomplishment to the player.

Storytelling is a game design pattern, where the game tells a story as the game progresses [GDP wiki, 2023; Kapp, 2012]. Kapp [2012] argues that this is an essential part of educational games, since it brings relevance, guidance, and meaning to the experience, as well as context to tasks. Kapp [2012] supports this with the fact that stories have throughout history been used to pass information from a person to another to guide behavior and thinking. I disagree with Kapp [2012] on the importance of this pattern for all educational games. This is because I think that a story is not necessary, for example when training to operate a machine or vehicle. This is not to say that the pattern doesn’t bring value. A counterargument to my view is if we use the broadest definition of storytelling and interpret the whole training experience as a story. Video games usually have stories, or can conjure stories in the player's mind, from simply through the name of the game found in older games, to having a complex story-lines with plot twists and surprises found in modern video games [Kapp, 2012]. Kapp [2012] argues that involving the player in the story makes learning more memorable. Kapp [2012] suggest four elements for building a story for an instructional game: characters, plot, tension, and resolution. Characters can be the player's coworker or a mentor who can guide the player to correct behavior, while the plot can simply mean that something happens, or has happened, which the player then has to react to.

3.6 Social interaction

This category covers patterns that describe ways that games can contain social behavior [Björk and Holopainen, 2005]. We will not address these patterns, since the games we are analyzing contain very little social interaction within the game especially among players. High Score Lists and characters in Storytelling shall fulfill the role of this category. We will instead leave the management of social interaction between students to educational institutions.

3.7 Goals

Björk and Holopainen [2005] use this category to identify different types of goals, which are arranged into four sub-categories. The categories are “Ownership and Overcoming Opposition”, “Arrangement”, “Persistence”, and “Information and Knowledge”. Technically the closest category here for the goal of educational games, learning, would be “Information and Knowledge”. As said before, every game, as well as the training simulator, has a goal, as do courses in education, so we will not concentrate on the abstract pattern that a goal is. However, we will discuss the definition of “goal” offered by Kapp [2012] since I think it offers useful observations, and we can link it to other patterns we discuss in this chapter.

A **goal** is an element of games that adds purpose, focus and quantifiable outcome to the activity [Kapp 2012]. Kapp [2012] suggests that giving players a way to visually determine how far they are from the goal creates incentive, feedback and an indication of progress that can be compared against other players. Kapp [2012] also says that a clear goal gives the player more choices on which techniques to use to achieve the goal thus more freedom. I would urge care that the freedoms given to the players shouldn’t interfere with the focus of the given training exercise. This means that the player mustn’t pass a training exercise using the ‘wrong’ technique that is not the point of the exercise. Instead, usage of the technique currently being taught in the exercise should be encouraged. Clear goals that specify what is and is not allowed is arguably an appropriate to support this. Kapp [2012] recommends that goals are structured and sequenced meaning that a terminal goal is supported by a series of enabling objectives, which function as incremental steps that allow the player to move from one completed objective to the next. Kapp [2012] justifies this by arguing that smaller goals leading to a larger one sustain the play, and that smaller goals help with building the skills necessary to achieve the larger goal. The smaller goals are often levels in the game [Kapp 2012]. Evidently, Progress Indicators can be implemented to accomplish issues related to keeping the player informed of larger goals and progress in them. This of course assumes the large goal can be defined as smaller steps.

3.8 Goal structures

This category inspects patterns that affect goals, explain how goals relate to winning the game, the interactions between goals, and the effect of goals on player's relations to each other in gameplay [Björk and Holopainen, 2005]. Since we excluded goals in general, and are generally interested in only one goal, we will skip this category as well.

3.9 Game sessions

This category's patterns describe characteristics of game instances and play sessions, as well as the limitations, possibilities, and features of player participation in games [Björk and Holopainen, 2005]. We focus on the first two.

Game Pauses cause the game-play and progression of game time to suspend [GDP wiki, 2023]. Typically, games progress automatically through a series of game events. Game pauses intentionally break this flow of progress [GDP wiki, 2023]. For example, many single-player video games implement the *game pauses* pattern by allowing the player to pause the game by entering a menu by pressing a specific pause-button. The player can then allow the game-play to continue by exiting the menu by pressing the pause-button again.

Time can be harnessed as an element of games, for example, to motivate the player to act [Kapp, 2012]. This can be done via creating consequences for the player by rewarding speed and penalizing slowness. This is also a form of the time limits pattern [GDP wiki, 2023]. One way to do this is through the use of the score: faster completion of goals leads to a higher and better score. Time could be used as a metric for the skill of the player: the more skilled the player is, the faster they can complete goals. An example of this is the concept of 'speed-runs', where a video game is completed from beginning to end as fast as possible [Wikipedia, 2023b]. To rank high in the speed-run community leader board, by achieving lower game completion times, the player, or 'speed-runner', has to be highly skilled in the game.

3.10 Game mastery and balancing

Björk and Holopainen [2005] define this category as a collection of "features related to how the players can use their skills and abilities in playing the game and how it is possible to balance the gameplay for players with different abilities". Most of the patterns given, such as Game Mastery, and Strategic Knowledge, are quite extensive in that almost the whole game's design needs to be considered. However, there are two interesting patterns that I think should be mentioned: "Right Level of Difficulty", and "Right Level of Complexity". I think one of the patterns we will discuss, Difficulty Levels, implements the first satisfyingly enough. The latter I argue, for education, should mainly be considered during the design phase of the training, such as the design of exercises/levels.

Difficulty levels [GDP wiki, 2023], or **playing levels** [Kapp, 2012], allow the player to adjust the difficulty level of the game to match their needs or skill level. Kapp [2012] explains that having different difficulty levels in the same game, such as easy, intermediate and difficult, allows more players to be able to enjoy and play the game. One way to do this in video games is by adding time constraints for completing the level or puzzle, faster obstacles and enemies, and overall making the game faster in the difficult or advanced difficulty level. On easier difficulty levels the game is slowed down and the game offers more guidance to the player with helpful text or other clues, and the challenges are usually simpler. The easier difficulty levels appeal to beginner players who are trying to learn the game, or to players who need simpler challenges or assistance with playing the game [Kapp, 2012]. For educational games, Kapp [2012] suggests adding three difficulty levels: demonstration, practice and test. In demonstrations the player can watch the proper procedure or technique being executed correctly. The practice levels guide the player through the procedure with feedback, clues and information. And finally, the test levels give no guidance to the player and are graded afterwards allowing the player to apply their learning in a similar environment as on the job. From this definition, we can identify patterns that are useful for implementing this pattern: Clues, Enemies, Obstacles, Real-time instructive feedback, and Time. Though arguably, all patterns from Sections 3.1 (Game elements); 3.3 (Information, communication, and presentation); and 3.9 (Game sessions) affect the difficulty of the game in some way, and should thus be considered when implementing this pattern.

Replay, or **do over** [Kapp, 2012], or **extra chances** [GDP Wiki, 2023], is a pattern that allows the player to restart the game, level or challenge. Replay is often implemented in the form of a button in the UI. It should be noted that replay pattern is distinct from replayability, which is a pattern of providing new challenges or experiences when the game is played again [GDP Wiki, 2023]. Kapp [2012] proposes that the replay pattern gives the player permission to fail and encourages, among other things, discovery-based learning, especially when the consequences of failure are minimal. It can be argued that this pattern helps in keeping the stakes low and thus maintaining a positive relationship with failure, which Lee and Hammer [2011] consider important. Practice levels could make use of the Save-load-cycles pattern and allowing player-defined Save Points, which would enable the player to choose where the replay begins, and thus enable them to practice things they find difficult. The other option is that the game designer identifies the difficult portions where Save Points should be placed. The replay pattern itself should be usable even in graded exercises when the exercise is started from beginning, just like exams in education in general can be retaken to improve grading.

3.11 Meta games, Replayability and Learning Curves

Björk and Holopainen [2005] describes patterns in this category as “issues that are outside the playing of a single game instance”. Like patterns in Section 3.11, the patterns here are generally quite abstract, so we will generally not discuss them, although with one exception.

Achievements are goals, whose completion state by the player is stored outside individual game sessions [GDP Wiki, 2023]. Achievements can be used to motivate the player to test all ways of playing provided by the game. This is especially useful if the core mechanics of the game are not enough. Simplest form of an achievement is a reward for completing a goal that is required for completing the whole game. These can thus measure how much progress the player has made in the game. Achievements can also be awarded for optional activities in the game. These include playing the game in a more challenging way, testing of optional activities, and perseverance. Perseverance here can mean that the achievement is awarded after the player has performed a certain task or action very many times. Achievements are often not stored on the player’s own device, such as a computer or game console, but are instead publicly viewable through the internet. It seems that achievements are a very common pattern in modern video games. Popular gaming platforms such as the PlayStation consoles (where achievements are called trophies) [PlayStation website, 2023], the Xbox consoles [Xbox website, 2023], and the Steam PC gaming platform and store [Steamworks, 2023] all have support for achievements in games. The consoles even require the game developers to use them [GDP Wiki, 2023]. Completing achievements can give the player bragging rights [Xbox website, 2023]. The act of striving to achieve every objective in the game, including all achievements, is called “completionism” [Wiktionary, 2019]. This pattern could also be considered an instance of Reward Structures, because they function as rewards for player actions. Achievements also function as goals.

4 Game design patterns in video games

In this chapter we analyze video games by identifying game design patterns found in them. Only focusing on one type of analysis was done to limit the scope of the research. The choice of focusing on video games for analysis was done because the video games chosen for analysis were easily available. Other training simulators or educational software could be found, for example, in educational institutions' premises, but they would not be as easily accessible.

All the games chosen for analysis here are simulations, more specifically dealing with vehicles. This is because the training simulator under study simulates the operation of a tram, a type of vehicle. Two differentiating factors are that, unlike the training simulator, these games are aimed at the general public and for recreational use. This means that the games are meant to be able to be played with common controllers, such as computer keyboards and mice, game-pads, joysticks and similar. Some of these games do, however, allow the use of a driving wheel type controller with pedals that can be connected to the game console or computer. The training simulator, on the other hand, is not meant for the public at large, and is instead intended to be used with real tram controls and dashboard connected to a computer. The training simulator will be discussed more closely in Chapter 5.

Even though the training simulator is not meant for recreational use like video games, it can still benefit from design patterns from video games that make the playing of the game more enjoyable, encouraging and memorable. The games discussed here do also contain some training elements, such as when guiding the player on how to play, thus teaching them on how to drive the vehicle used in the game.

The video games chosen for analysis, presented in their own following sections, are TramSim Vienna, City car driving, and Gran Turismo 7. Other games that were considered, but ultimately not included to limit the research scope, include the Bus Simulator series [see: Bus Simulator, 2023], the Euro Truck simulator 2 [see: Euro Truck Simulator, 2023], the Farming Simulator [see: Farming Simulator, 2023] series. The reason for choosing TramSim Vienna as one of the games was because its similarity to the training simulator, both being tram simulators. City car driving was chosen because its marketed educational features, and Gran Turismo 7 because of its popularity and recreational aspects. The final section in this chapter summarizes the findings.

4.1 TramSim Vienna

TramSim Vienna aims to be a modern tram simulator game with realistic graphics and tram functions [Steam, 2023d]. The target audience of the game is public at large including those who have never driven a tram or played a tram simulator before. The game can be played with keyboard and mouse, a gamepad, or in virtual reality.

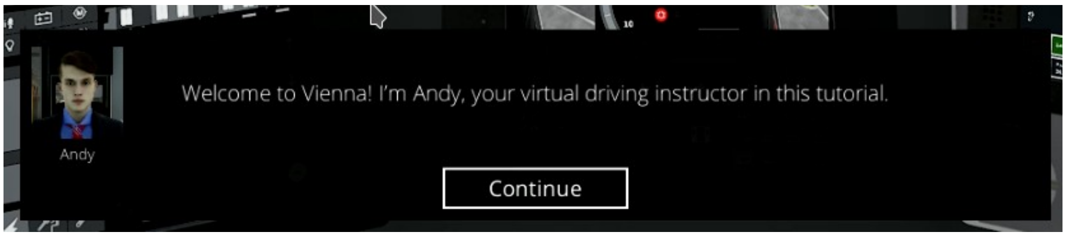

Welcome to Vienna! I'm Andy, your virtual driving instructor in this tutorial.

Figure 2: A HUD interface that is shown in TramSim Vienna tutorials whenever instructions need to be shown.

The game offers three interactive tutorials accessible from the main menu, as well as some general help documentation, to get started with the game. The tutorials are built so that the user does not need to read the manual of the game. The tutorials are game-play interspersed with pauses during which the game explains information about the game. This is done using the game pauses pattern when needed to halt the game-play and with the **HUD interface** pattern to show the tutorial instructions as text. Figure 2 shows an example of this HUD interface. The game has a virtual driving instructor who speaks and instructs the player via these HUD interface texts, so the text in the HUD interfaces can be thought of as coming from the virtual instructor. The virtual driving instructional qualifies the game to have light **Storytelling** elements. The game does not include spoken language, only the HUD interface texts. The HUD interfaces contain a "Continue" button that moves the tutorial forward when pressed.

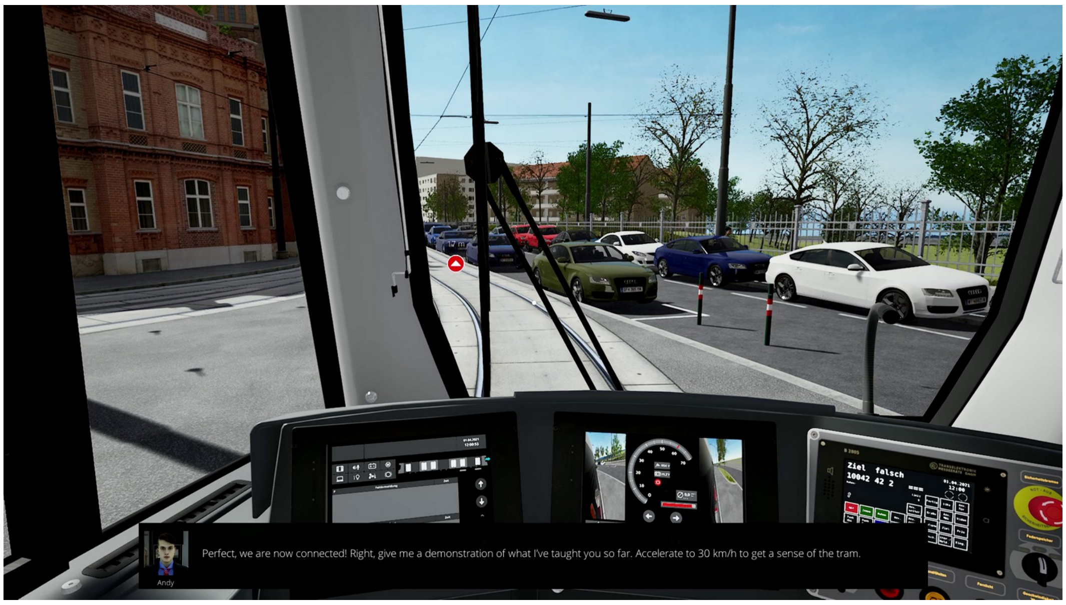

Perfect, we are now connected! Right, give me a demonstration of what I've taught you so far. Accelerate to 30 km/h to get a sense of the tram.

Figure 3: A red HUD interface element near the center of the screen which shows the player where to go next.

First of the tutorials, titled "Basics", offers overview of the main controls used in the game. The tutorial does this by showing button names, for example keyboard keys, in the HUD interface. Here the tutorial advances either by the player pressing the correct button on their controller, or by pressing the "Continue" button depending on the task. Topics covered by this tutorial include, for example, preparing the tram for driving, acceleration and braking, how to open and close the tram doors and so on. The tutorials also use HUD interface elements to show where the player should move the tram next. The HUD interface element is shown in Figure 3. Also seen in the figure are parts of the tram **Avatar,** which the player view is inside of. The whole tram model can also be viewed from the outside of the tram. The HUD elements can also be understood as a instance of the **Clues** pattern, and the instructions given are a form of a **Goal Indicator**. There are also **Clues** placed in the game world, such as traffic signs.

The second tutorial titled "Signals" teaches the player on how to read tram specific traffic signals. This time the HUD interfaces are combined with a **Game Pause** and a change in game view to show signal devices the interface text is talking about. The player can always pause and continue the game during game-play as well. The tutorial guides the player through all the different signal types and related traffic rules. The guiding of the player in small steps towards the larger goal, end of the tutorial, classifies as a **Progress Indicator.**

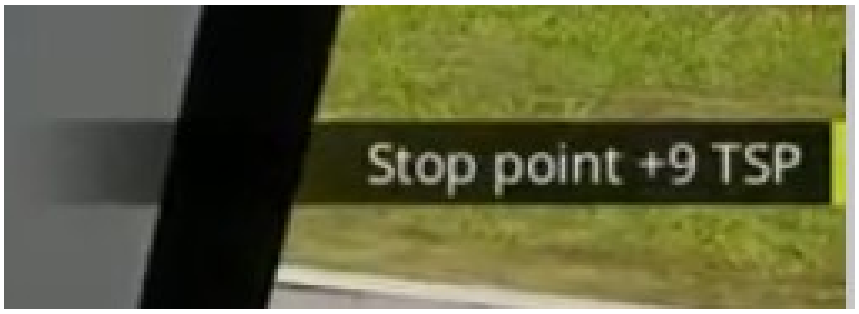


Figure 4: HUD interface element notifying the player of the TSP earned.

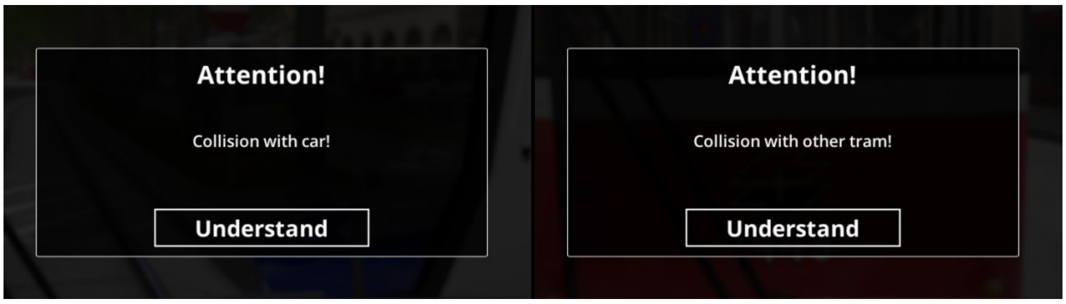
The third tutorial titled "Gameplay" mainly explains the game's scoring mechanic. The game uses a **Score** game design pattern to assess player actions in the game. The scoring mechanic is based on TramSim Points (TSP) that are awarded for actions in the game. TSP qualifies as an implementation of a **Reward Structure** as well. Actions that award TSP and their descriptions are listed in Table 3. As punctuality, how well the player sticks to the tram line schedule, is tracked, makes the **Time** pattern relevant. Whenever TSP is awarded a HUD interface in the right side of the screen is displayed showing the amount of TSP earned as shown in Figure 4. This is a form of **Real-time instructive feedback**, and gaining TSP to the player’s in-game profile, and the fact that it can be used to unlock new features, can be considered **Character development.** At the end of the trip the game shows to the player how much TSP they earned, which is an implementation of **Postponed feedback.** Different game modes and duration of trip affect the amount of TSP awarded. The three tutorials are separate sections accessed from the menus. Different driving routes, and challenges, also have their specific goal for that section of the game. Therefore, these can be considered as **Levels** in the game. Since there are different types of levels, such as tutorials and challenges, which could translate to “practice” and “test” levels, the game has some form of **Difficulty Levels**. The Levels can be **Replayed**.

Action	**Description and notes**
Stopping at a tram stop	Stopping at a tram stop. The tram stop has an optimal stopping point and more points are awarded the closer the player stops the tram to that optimal point. The optimal point is visible in the game world usually as a texture on the ground.
Punctuality	The game awards points for 'punctuality'. Neither the game nor the manual explains clearly what this means but it seems to be related to following tram schedule. The more punctual the player is the more points are awarded.
Completing a route or tutorials	Generally completing goals, like driving a route end-to-end, grants TSP.

Table 3: Actions that award TSP in TramSim Vienna.

The game simulates traffic as part of the simulation of driving a tram in the city. The traffic consists of cars, pedestrians, and other trams that follow the traffic rules. Whether to treat the other traffic participants as mere **Obstacles** or **Enemies** is more nuanced. They are active because they move, and the player is expected to avoid collisions with them. This should be easy as long as the player themselves follows traffic rules. If the other traffic “purposely” broke rules which would lead to dangerous situations, they could qualify them as **Enemies**. We will consider this as a possibility.

The game does not punish or take away player's TSP for making mistakes during game-play [TramSim, 2020]. The game doesn't inform the player about mistakes either, except for mistakes where the player collides with another traffic entity such as a car or another tram. When colliding with another traffic entity the game displays a warning to the player shown in Figure 5. This is done using the game pauses game design pattern combined with a HUD interface. When the player presses the "Understand" button in the HUD interface, the collided traffic entity is removed from the game world and the game-play continues as if the collision mistake never happened. In the real world, crashing with another traffic entity has serious consequences that realistically stops driving. However, not punishing the player lowers the stakes of the game and makes it more fun for a casual audience. The feedback from the traffic collisions and TSP accumulation are immediate so the feedback cycle in the game is rapid and a form of **Real-time instructive feedback**.


Attention!

Collision with car!

Understand

Attention!

Collision with other tram!

Understand

Figure 5: Warnings to the player when colliding with other traffic entities. The left warning is displayed on collision with a car while the right warning is displayed on collision with a tram.

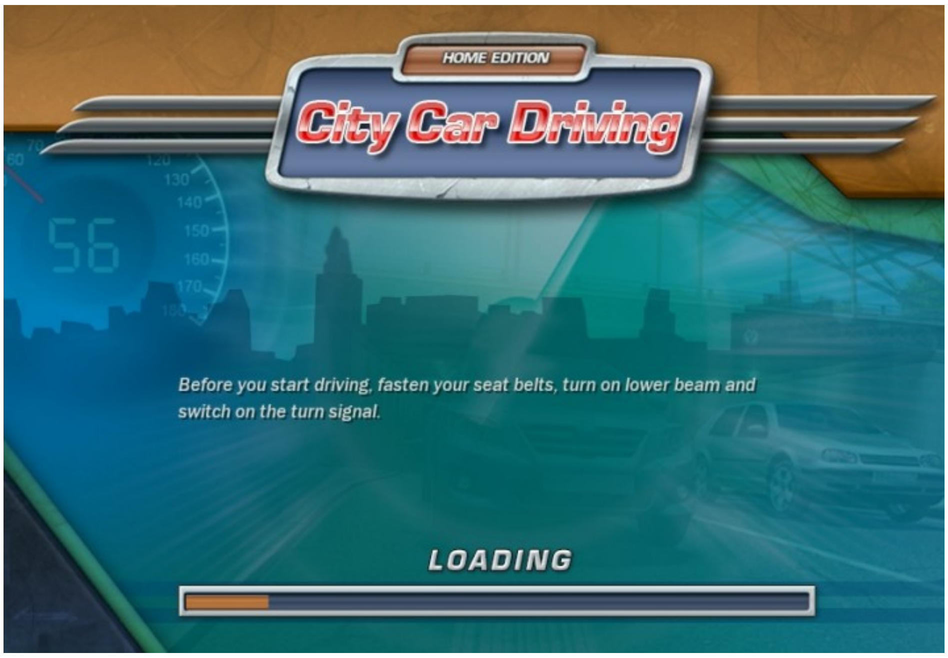
The game has **Achievements** via the Steam platform on PC [Steam, 2023b]. They can be mostly considered to be awarded from progression and perseverance.

The game allows saving the game state mid-route in player-defined **Save Points** and loading it later to continue driving with **Save-load-cycles** [Steam, 2023c].

4.2 City car driving

City car driving is a video game designed to train car driving skills in various environments and conditions [Steam, 2023e]. The game tries to simulate reality with immersive environment, sophisticated traffic including computer controlled other vehicles and pedestrians, different weather and road conditions as well as times of day. The traffic includes dangerous situations via other cars violating traffic rules such as other vehicles switching lanes improperly and suddenly, others driving on the wrong side of the road, traffic lights being sometimes broken and unpredictable pedestrians that may run in front of the player’s car, to name a few. Weather, road and time of day conditions range from easy clear skies and dry roads up to challenging heavy morning fog combined with slippery ice or snow.

The game assesses player's traffic rules compliance in real time while driving and gives continuous feedback [Steam, 2023e]. This fulfills the **Real-time instructive feedback** pattern. The game supports traffic rules from multiple regions chosen by the player such as the United States, European Union and Australia (with left-hand traffic) [Steam, 2023e]. In addition to city driving training the game contains different types of driving exercises from driving basics to practicing extreme driving conditions and counter accident training. These different types of exercises can be considered **Levels**, and their varying difficulties as **Difficulty Levels.** The Levels can have time limits for completion, thus bringing an element of **Time** into the game. The game can be played with keyboard and mouse, gamepads, gaming wheels and supports playing in virtual reality. The game has "Career" and "Free driving" modes [City car driving home edition user manual¸ 2019]. The career mode is separated into five levels, with each level consisting of a number of tasks. The Career mode could be considered to have elements of **Storytelling**. Another instance of Storytelling are in Free driving mode, when the player can work as a taxi driver, transporting passengers around the city. The passengers can comment the player's driving an can therefore be considered characters in the "story". Doing well in the Free driving mode can increase some player statistics in a positive direction, thus enabling **Character development**. The Career mode can also be thought of as an implementation of this.

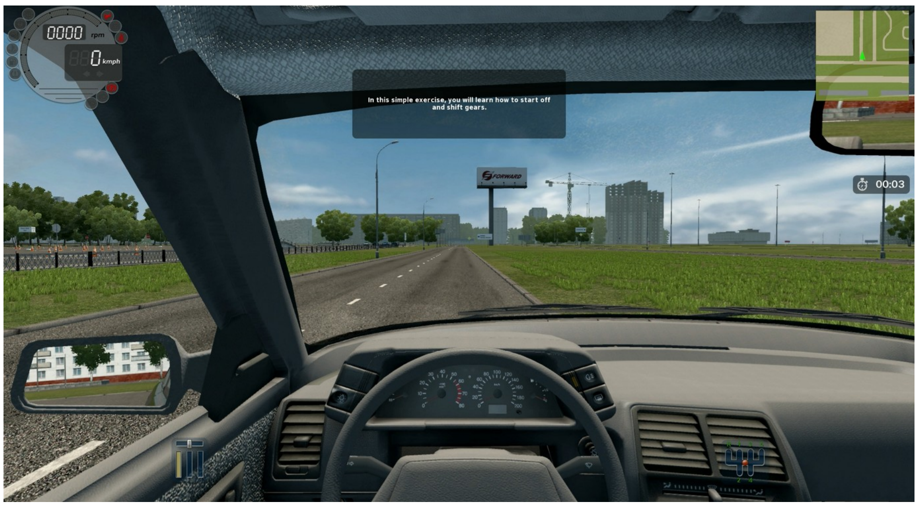


Before you start driving, fasten your seat belts, turn on lower beam and switch on the turn signal.

Figure 6: A loading screen containing a loading hint in City car driving.

The game implements the **Loading Hints** pattern by showing information about the gameplay during loading sequences. Examples of hints given are, for example, how certain driving and accident avoidance affects passenger satisfaction, what to do when the car is damaged and what to do before beginning to drive. Loading sequences are present when starting a new game or starting an exercise. Figure 6 contains an example loading screen with a loading hint.

The game has a **HUD**, which contains various elements. Always visible elements include a speedometer in the upper left corner, the car's transmission state and current gear in the lower right, a timer on the right that shows time elapsed since beginning of exercise, or alternative time left, and a **Mini-map** in the upper right corner. Figure 7 shows these elements as well as the overall gameplay view. Shown as needed elements are the instruction text box in the upper center of the screen and driving feedback text box in the right side of the screen. These two elements will be discussed in more detail later.

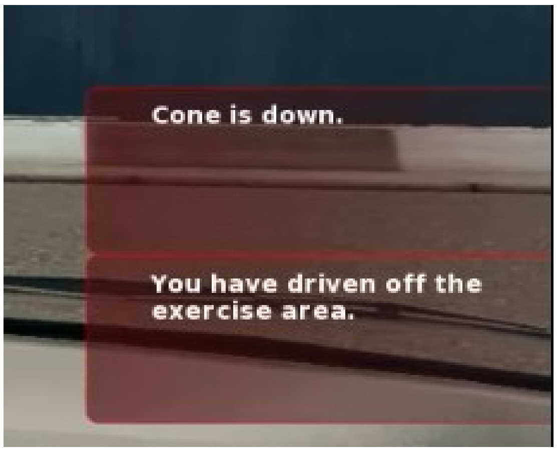


In this simple exercise, you will learn how to start off and shift gears.

Figure 7: Game view during gameplay.

The game implements the **Mini-map** game design pattern as an element in the HUD. The mini-map shows the location of the player's car as a green triangle and the streets the player can drive on. The mini-map may be useful when navigating a city and for clearly spotting intersections so the player can prepare to stop in case of traffic lights or other signs that call for stopping or letting crossing traffic pass. The map can also show an optimal route to reach the player's goal from their current position so it works also as a **Goal Indicator**, and a **Progress Indicator**.

The game contains many **Clues**. The two most obvious ones are the instruction text box and driving feedback text box implemented as HUD interfaces. Other clues in the game are in the game world itself: traffic signs, lights and road markings. The player is expected to obey these clues and will be given feedback based on traffic rules compliance. The instructions text box is mostly used in exercises where it contains instructions for the next step the player should do, such an action like fastening the seat belt, and possibly the buttons the player should press. This is also a form of **Goal Indicator**, and

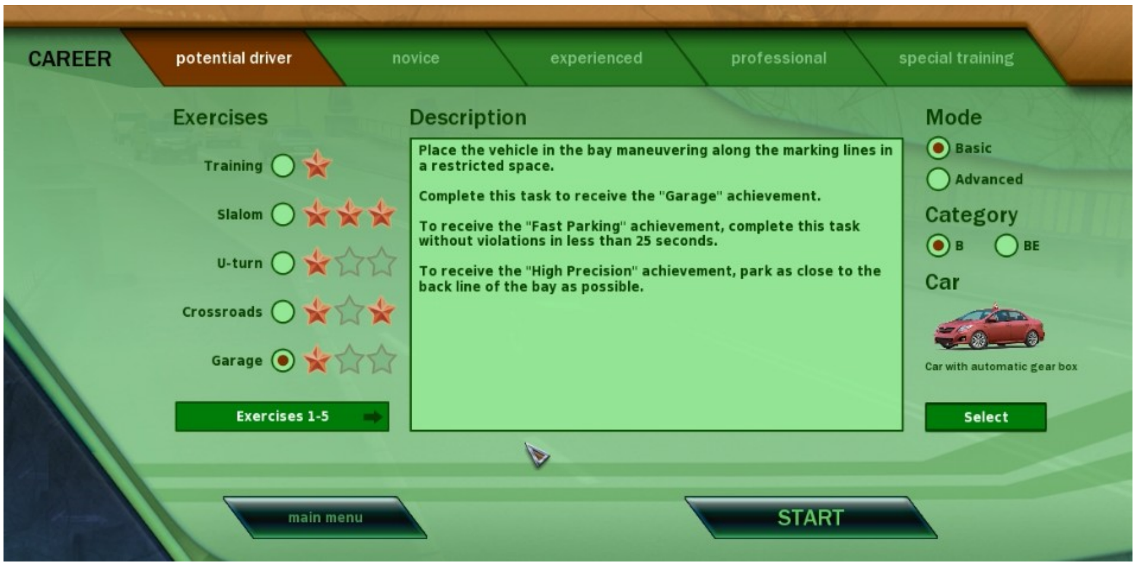

Cone is down.

You have driven off the exercise area.

Figure 8: Examples of negative feedback text boxes shown when violating exercise rules.

possibly a **Progress Indicator** as well because when the next instructions are shown it indicates progress in the exercise. The feedback text box is shown as a reaction to player actions. The most common feedback is a notification about traffic violations or about breaking exercise rules. These count as negative feedback and are shown as red boxes. Other feedback is shown in different color text boxes, such as green for encouraging feedback. Figure 8 shows examples of negative feedback text boxes. The feedback is given instantly so the feedback cycle in the game is rapid and is therefore **Real-time instructive feedback**.

The game shows goals visible in other places as well. An example can be seen in Figure 9 in he level instructions text box where the goals are listed. These include the main goal of the exercise, which is required to complete the level, and optional goals. When a goal is completed, the player is awarded an “achievement”. Completed achievements are indicated as colored stars next to the exercise selection. Missing achievements are indicated as empty star shapes. If the player seeks to complete all goals, the player can see what is missing at a glance.



Exercises: Training, Slalom, U-turn, Crossroads, [✓ Garage]

Place the vehicle in the bay maneuvering along the marking lines in a restricted space.

Complete this task to receive the "Garage" achievement.

To receive the "Fast Parking" achievement, complete this task without violations in less than 25 seconds.

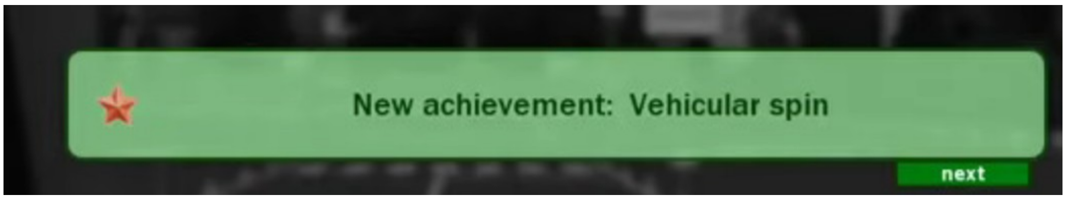
To receive the "High Precision" achievement, park as close to the back line of the bay as possible.

Mode: [✓ Basic] Advanced

Category: [✓ B] BE

Figure 9: Exercise selection screen showing the exercise instructions (goals) as well as current score (star rating) of all exercises.

After finishing a level the game tells the player which goals, and thus achievements, they completed. This is **Postponed feedback**. This can be seen in Figure 10. The achievements here satisfy some of the features of the **Achievements** design pattern as well, since they show the player’s progress and are awarded for achieving objectives. However, there is another system for achievements in the game, which is more fitting, because they can be seen in the player’s public profile, while the purely in-game achievements can not. The achievement stars here can be considered a form of **Reward structure** also, as well as a form of the **Score** design pattern.



New achievement: Vehicular spin

Figure 10: The completion of a goal in City car driving [Northernlion, 2021b].

The game offers **Achievements**, the “real” implementation according to us, that are awarded during the game-play with integration with the Steam platform [Steam, 2023a]. The main types of achievements in the game are for optional activities and perseverance. Activities related achievements encourage the player to try the different features of the game, while the perseverance related achievements are awarded from long-term activity, such as driving long distances in a certain area or country. Achievements awarded from playing the career mode could be considered progress related. Safe and correct driving is also awarded, such as from receiving the maximum passenger satisfaction score on a task.

Because the other traffic participants, cars and pedestrians, can sometimes actively hinder the player by causing accidents that lead to negative feedback, they can be considered **Enemies**. An example mentioned earlier is the event of a pedestrian breaking traffic rules and walking in front of the player's vehicle **Avatar**. This forces the player to react and evade the pedestrian either by emergency braking or driving off the road, whichever has a hope of preventing the accident. Cars can also do this as well, such as by unpredictably changing road lanes, possibly even to the wrong direction lane, which can lead to a head-on collision with the player's car, if they player doesn't react to prevent it.

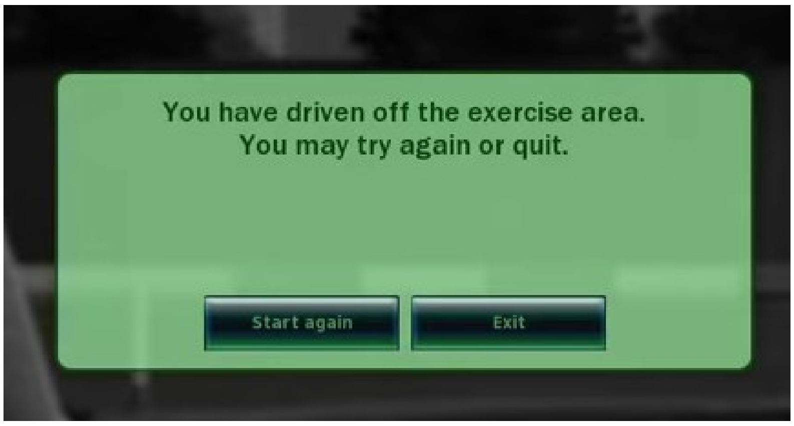

You have driven off the exercise area.
You may try again or quit.

Figure 11: Consequence of driving off the exercise area in a more strict level in City car driving [Northernlion, 2021a].

Replay: Some levels in the game have stricter rules for driving. Figure 11 has an example of the consequences of doing certain mistakes in a stricter level. The Figure also shows an implementation of the replay pattern, where the player can quickly try the level again by pressing a button. Providing an easy and quick way to try again lowers the stakes of the level. The player can also **Pause** and continue the player at any point during game-play.

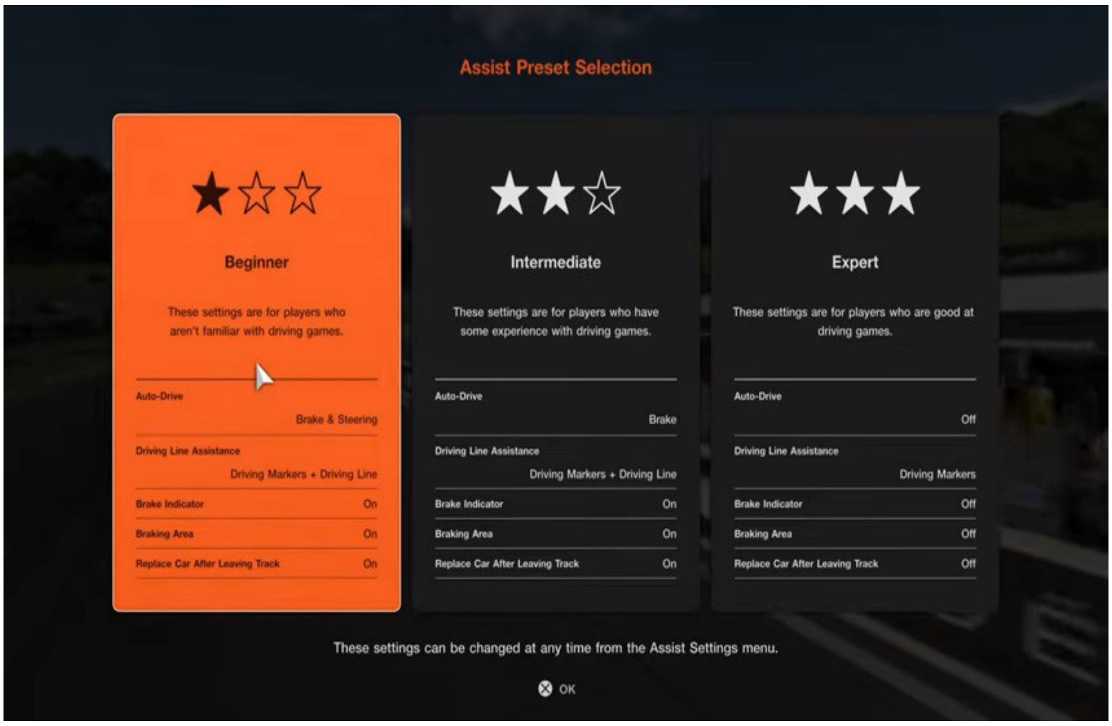
4.3 Gran Turismo 7

Gran Turismo 7 is a driving and racing game released in 2022 for the Sony PlayStation 4 and 5 game consoles, and developed by Polyphony Digital [PlayStation website, 2023b]. The game is marketed as a "real driving simulator" because it aims for realism and realistic feeling of driving [Gran Turismo website, 2023]. The Gran Turismo website [2023] lists the following features included in the game:

- Weather simulation that can alter the driving experience, such as by rain making the track surface wet thus affecting traction and vehicle handling.

- Wind direction and air turbulence and their effects on the vehicle are simulated.
- The game caters to both beginners to the Gran Turismo game series up to experienced players. This is done by including a variety of ways to play the game, such as participating curated races, playing car-based mini-games, learning racing basics in instructional type levels, and racing against other players in online races.
- The game is suitable for E-sports by offering competitive online daily races and an official online championship.

The game has a reward and rating system that is based on the player's in-game activity [Gran Turismo 7 online manual, 2023], therefore **Reward Structures.** The player can earn credits by participating races and events. The player can improve the credit amount they receive with the following ways: playing higher difficulty races, longer distance races and finishing in a higher position in the race. The game awards a "clean race bonus", if the player manages to complete the race without going off-course or colliding with other cars. The clean race bonus, and other rewards are awarded as **Postponed feedback** in a screen shown after the race. The credits can be used to buy cars and to pay for tuning and maintenance work. Purchasing cars accumulates the player's "collector points" which in turn increases the player's "collector level" when a sufficient amount of points are reached. Collector levels unlock new levels called "missions", parts used to tune the player's vehicles, and cosmetic items that the player can use to modify the look of their avatar, such as helmets and racing suits. Collector levels and the credit system count as **Character development.**


Assist Preset Selection

Beginner

These settings are for players who aren't familiar with driving games.

Auto-Drive: Brake & Steering
Driving Line Assistance: Driving Markers + Driving Line
Brake Indicator: On
Braking Area: On
Replace Car After Leaving Track: On

Intermediate

These settings are for players who have some experience with driving games.

Auto-Drive: Brake
Driving Line Assistance: Driving Markers + Driving Line
Brake Indicator: On
Braking Area: On
Replace Car After Leaving Track: On

Expert

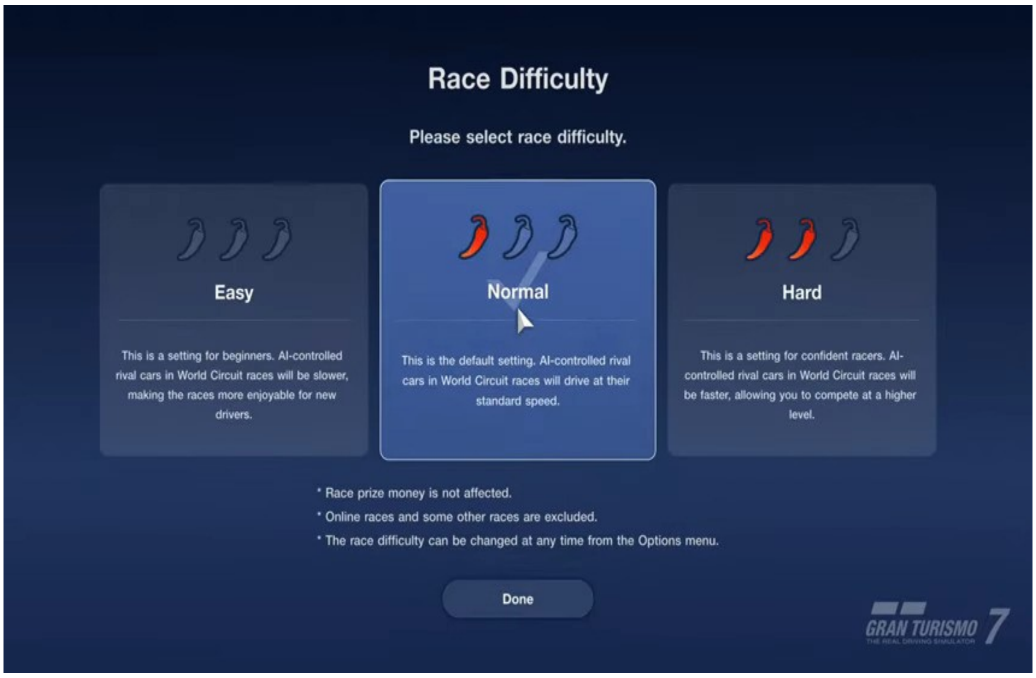
These settings are for players who are good at driving games.

Auto-Drive: Off
Driving Line Assistance: Driving Markers
Brake Indicator: Off
Braking Area: Off
Replace Car After Leaving Track: Off

These settings can be changed at any time from the Assist Settings menu.

Figure 12: Choosing the level of assistance and guidance features in Gran Turismo 7 [Error1355, 2022].

The game offers two ways to customize the **Difficulty Level** of the game to cater to beginner, intermediate, and advanced players. The game allows the player to choose these settings when the player starts the game for the first time. First, the player selects the level of assistance and guidance features, as seen in Figure 12. The easiest option, the beginner difficulty, is aimed for players new to driving games in general. It enables all of the assistance features. The intermediate difficulty setting disables some assistance features, and the most advanced setting, the expert difficulty, disables almost all assistance. These last two settings are meant for intermediately skilled, and very skilled and experienced driving game players respectively.


Race Difficulty

Please select race difficulty.

Easy

This is a setting for beginners. AI-controlled rival cars in World Circuit races will be slower, making the races more enjoyable for new drivers.

Normal

This is the default setting. AI-controlled rival cars in World Circuit races will drive at their standard speed.

Hard

This is a setting for confident racers. AI-controlled rival cars in World Circuit races will be faster, allowing you to compete at a higher level.

* Race prize money is not affected.
* Online races and some other races are excluded.
* The race difficulty can be changed at any time from the Options menu.

Figure 13: Choosing race difficulty level in Gran Turismo 7 [Error1355, 2022].

The second difficulty selection occurs after a short introductory game-play section, during which the player can test their skills, and possibly reassess their need for assistance features. The second difficulty setting is concerned with modifying the AI-controlled rival cars', the **Enemies** of the game, behavior by modifying their speed. Slower rival cars are more easily bypassed in the race and thus it is easier to reach higher rankings in the race by the player. Again, the game offers the player three different skill and difficulty level options, as seen in Figure 13. The race difficulty option affects the player's access to certain races and features, such as racing online against other players. The player can change these two difficulty settings from the settings menu. This allows the player to raise the difficulty of the game as their confidence and skill level increases helping the game stay interesting as the player becomes better at the game, and avoiding frustration caused by too much difficulty for the less skilled beginner players.

Another way the difficulty and complexity of the game is incrementally increased are the changes in the **HUD**. In the introductory game-play section the HUD has fewer elements than the normal HUD used in the main game during races. This can be seen in Figure 14 that shows the introductory section, the music rally, HUD above the normal racing HUD. The music rally game mode has some UI elements specific to that mode that don't appear in the normal racing UI.

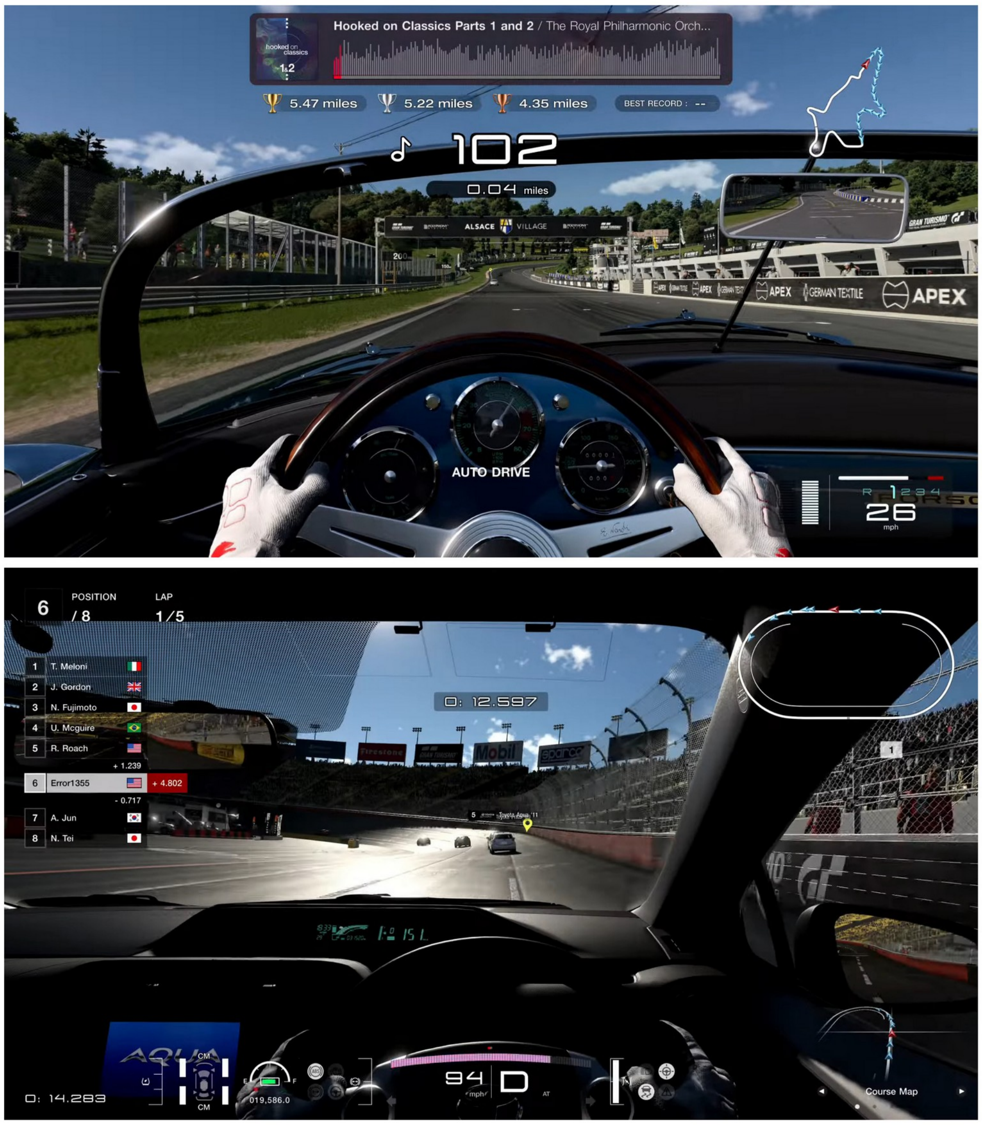
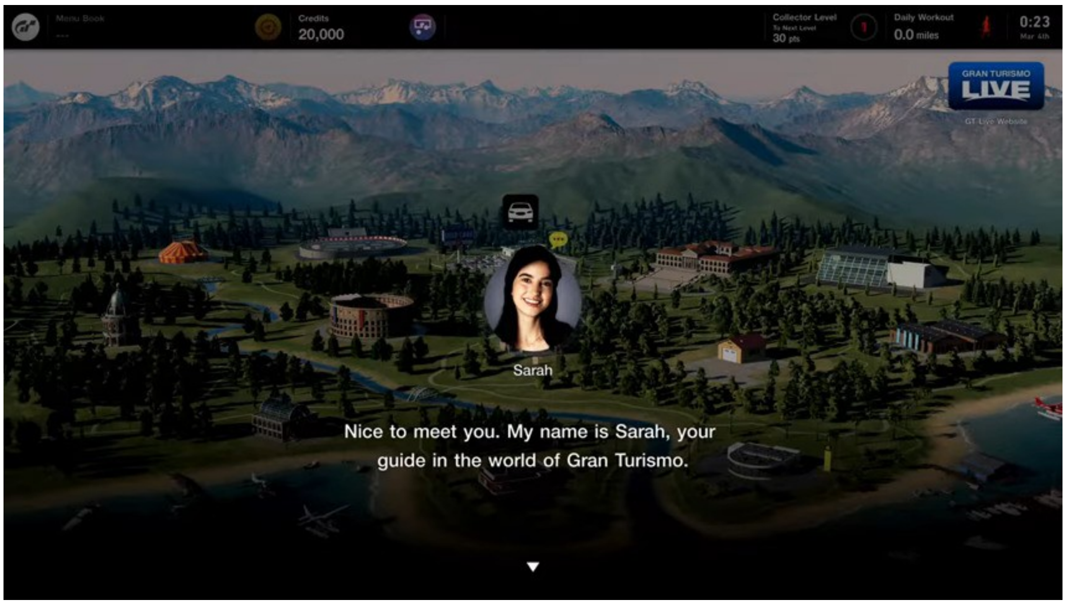


Figure 14: Gran Turismo 7 game-play user interfaces in the introductory Music rally game mode (above) [Error1355, 2022], and the UI used in standard racing mode (below) [Error1355, 2022].

In the upper-right area of both HUDs in Figure 14 we can see an implementation of a **Mini-map**, which depicts the race track and the player **Avatar**'s location in it. The player is shown as a red triangle pointing towards the direction of movement. Other rival cars, or **Enemies**, are shown as blue triangles. Another mini-map can be seen in the at the bottom right area of the standard racing mode UI. This mini-map has the same elements, but showing a more closer view of the player's surroundings and incoming turns on the race track. Both of these mini-maps make the game more predictable by showing the track and incoming turns in advance allowing the player prepare, such as by slowing down in advance. The map also work as **Progress Indicators**, because they show the player’s distance, how far ahead or behind the player is, compared to rival drivers. Other elements in the HUD also support the function of the map. The visibility of other cars on the maps adds to predictability by allowing the player to prepare for passing, or being passed by, a rival car. These features can arguably make the game more pleasant, easier, and less stressful.

Another element of both HUDs is the visibility of goals in **Goal Indicators**. In the music rally, the goals are laid out at upper center of the screen with small trophy symbols, from bronze to silver to gold, next to the description of the goal, the target distance to be traveled. Near the center of the screen is also the current distance traveled, which helps the player track and assess their progress towards the targets, and is therefore a **Progress Indicator**. The goal in the normal race is to place as highly as possible in the race. This goal is made visible with elements in the upper-left corner: the position indicator text, the lap count, and the leader board. The position indicator text and the leader board carry mostly the same information: the player's position in the race and distance from the goal. The lap count informs the player how many rounds on the race track the player has available to try to reach higher positions. Other ways the state of the game is portrayed are the UI elements around the screen, like the speedometer.



Nice to meet you. My name is Sarah, your guide in the world of Gran Turismo.

Figure 15: Sarah, the player's guide character in Gran Turismo 7.

The game contains some **Storytelling** elements in the single player campaign, such as characters. Examples of these are Sarah, the player's guide to the game, and Luca, who gives the player objectives. Sarah can be seen in Figure 15, and Figure 16 shows some dialogue with Luca, where he is giving the player a new objective to collect cars. The campaign consists mostly of collecting cars and completing races [Gran Turismo

Wiki, 2023a]. At the Figure 16 top right corner we can see an instance of the **Achievements**, called trophies here, system provided by the game and the PlayStation platform together [IGN, 2022]. The achievements are mostly related to progress towards and completion of goals in the game.

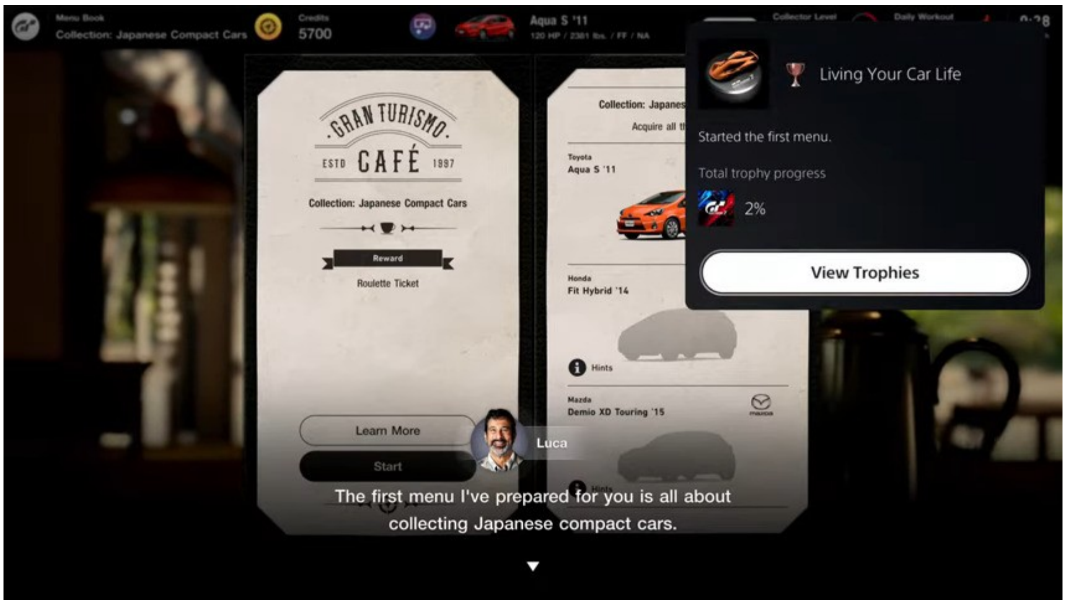


Figure 16: Luca character giving the player a new objective in Gran Turismo 7 [Error1355, 2022].

Gran Turismo 7 provides constant **Real-time instructive feedback** on the behavior and performance of the player. Figure 17 shows some examples of visual feedback in the game. Images 1 & 2 show the player their speed compared to previous rounds next to the stopwatch element at the center of the screen, with 1 informing the player it is their fastest round, and 2 informing that the player was a little slower. 5 tells the player they are going the wrong way on the race track with the traffic sign. Feedback on player mistakes is usually different in challenge levels than in races. In challenge levels a mistake can end the challenge, as seen in image 3. Races often use the penalty system. Examples of the penalty system feedback can be seen in image 4. The penalty information overlay appears under the stopwatch element, and is made up of two parts: text informing the player of the reason for the penalty, and the punishment caused by the penalty. The punishment causes the player to slow down, thus losing time and potentially risking rival cars passing the player. Penalties can be given from various mistakes and illegal actions, and the penalty system's purpose is to be a deterrent against unfair game-play [Gran Turismo Wiki, 2023b]. Images 1, 2, and 4 add the **Time** element to the game. The game can also be **Paused** during game-play by the player.

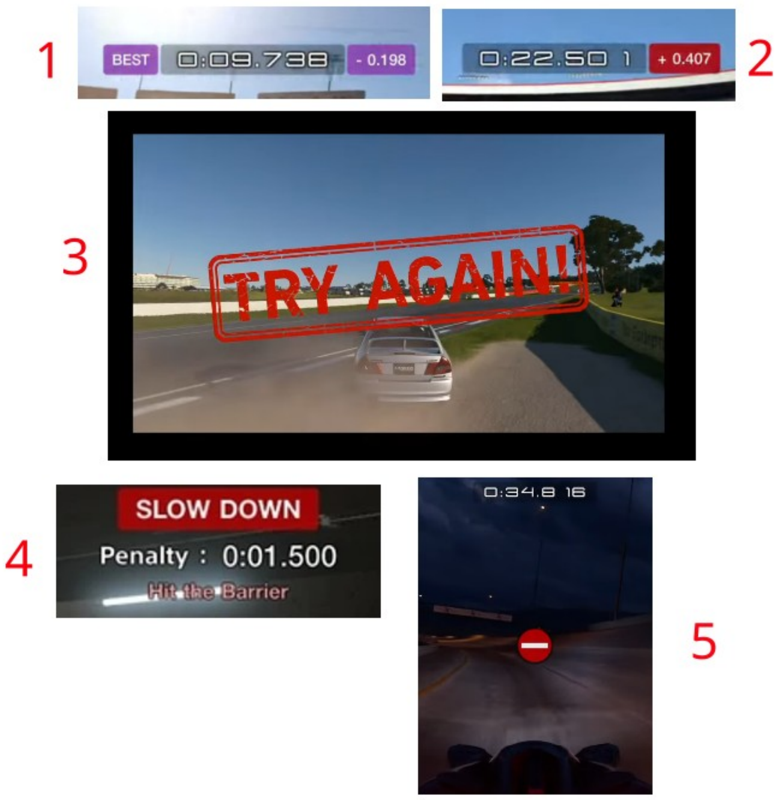


Figure 17: Examples of visual feedback in Gran Turismo 7. Images: 1 & 2 [Error1355, 2022]. 3 [Consistent Walkthroughs, 2022]. 4 [Gran Turismo Wiki, 2023b]. 5 [Nismonath5, 2022].

Image 3 in Figure 17 shows the potential result of a challenge. The challenges can be **Replayed**, as can races, to allow the player to try again for a better result. The challenges and races in the game can be considered **Levels**.

The game offers the **Ghosts** game design pattern to help the player improve their performance. The ghost appears as a translucent car on the track, which the player can drive through. An example of a ghost can be seen in Figure 18. The ghost can display the player's personal best and the demonstration driving [Gran Turismo 7 online manual, 2023]. The player can choose whether one, or both, are visible. Passing the ghost on the track means the player is faster than their personal best or the demonstration.

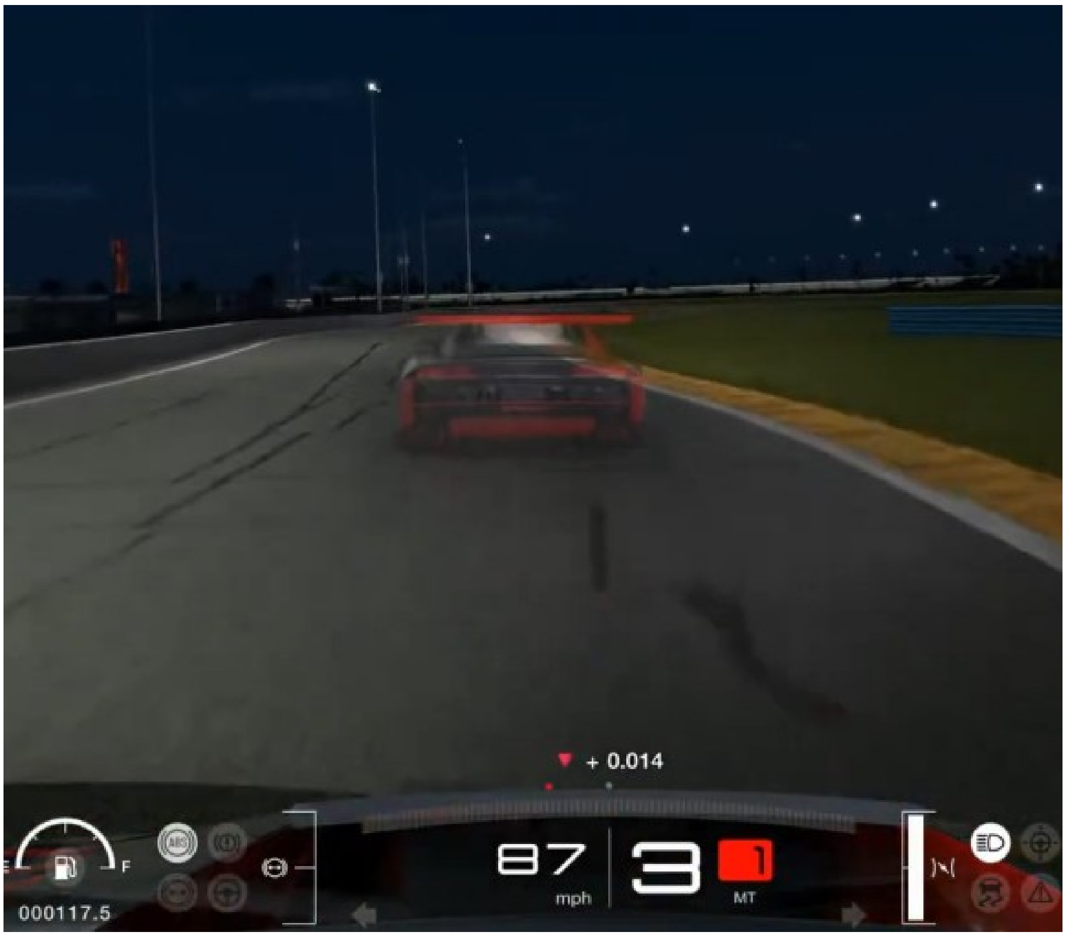


Figure 18: A ghost in Gran Turismo 7 [RA Shadow, 2022].

4.4 Observations

In this section we will briefly compare and find similarities, and differences, in the three games we have discussed thus far. Looking at Table 4, we can see that the games are very similar when considering that they contain mostly the same game design patterns. However, we used quite broad interpretations of what qualifies as the pattern in question. So differences can be found on how many different components have the pattern been implemented with, as well as how the pattern has been implemented in general. Something to note is that the focus of game analysis was on at most the first few hours of the game, so some details may be missing. We will start with patterns that have the most similar implementations and work towards those with more differences.

Achievements, Avatars, Clues, Game Pauses Levels, Reward Structures, Score, and Storytelling have very similar implementations across the games. Achievements can mostly be explained in that the game platforms dictate how they are implemented, and the pattern is pretty standardized in general. So are Game Pauses mostly. All games here have an avatar of the vehicle being driven. Also, all of them allow the player to view the avatar from the outside, be that by walking outside the vehicle, or by a third person view. The games all have Clues in the environment and in the HUD. The games provide multiple types of Levels as well: tutorials, challenges, as well as “real driving”, be that driving in simulated traffic or in a race against other cars. Score is always awarded from player actions. All the games contain minimal storytelling, including some characters at most, and not implementing full stories with strong narratives.

Differences in the presentation of the pattern implementation are found in Character development, Goal Indicators, HUD interfaces, Postponed feedback, Progress Indicators, Real-time instructive feedback, Reward Structures, and Time. The games differ in ways and metrics Character development is measured, from mostly only TSP in TramSim Vienna, to collector levels in Gran Turismo 7. The arguably clearest presentation of a Goal Indicator and Progress Indicators is in Gran Turismo 7, because the goal can be interpreted from multiple elements visible in the HUD. Next comes City Car Driving with the use of Mini-map and instructional texts. And last TramSim Vienna, which contains minimal indicators of the goal or progress, though the goal is likely already clear to the player. Time is also presented differently in the games with different kinds of HUD elements, like Gran Turismo 7’s penalty indicator, to the countdown timers found in City Car Driving. In Gran Turismo 7 the goal is to drive in the lowest time possible, while in TramSim Vienna it is to stay on schedule. HUD interfaces, and Postponed feedback differ in how many elements, or statistics about the player’s performance, are made reported and made visible. Evidently, the “prize” for the highest abundance of Real-time instructive feedback belongs to City Car Driving, since it provides constant informative feedback about the successes and mistakes made by the player.

Game design pattern	**TramSim Vienna**	**City car driving**	**Gran Turismo 7**
Achievements	X	X	X
Avatars	X	X	X
Character development	X	X	X
Clues	X	X	X
Difficulty Levels	X	X	X
Enemies	X	X	X
Game pauses	X	X	X
Ghosts			X
Goal Indicators	X	X	X
HUD interfaces	X	X	X
Levels	X	X	X
Loading hints		X	
Mini-maps		X	X
Postponed feedback	X	X	X
Progress Indicators	X	X	X
Real-time instructive feedback	X	X	X
Replay	X	X	X
Reward structures	X	X	X
Save-load-cycles	X		
Save Points	X		
Score	X	X	X
Storytelling	X	X	X
Time	X	X	X

Table 4: Game design elements found in analyzed games. “X” means that the pattern is found in the game, while empty cells indicate that the pattern is not in the game.

The second place goes to Gran Turismo 7 because it, although less than City Car Driving, offers both positive and negative feedback. TramSim Vienna offers minimal feedback in general, especially in negative types. Reward Structures differ in the amount of actions they can be received from, and how many types of rewards there are in the game.

Some implementation differences can be found too in Difficulty Levels, Enemies, and Replay. TramSim Vienna has fewer ways of customizing the difficulty of the gameplay compared to City Car Driving and Gran Turismo 7. They do all offer different types of Levels with different levels of difficulty. Enemies behave differently in the games. Our analysis is unclear whether TramSim Vienna has “real” Enemies that fit the pattern, while in City Car Driving they exist and actively hinder the player by trying to cause accidents and make the player do mistakes. The enemies in Gran Turismo 7 are more predictable and are not generally “hostile” towards the player, for example by trying to make the player drive off the track. The Gran Turismo 7’s multiplayer mode differs in this aspect, but that was excluded from the analysis.

Next pattern to discuss are Mini-maps, which are missing form TramSim Vienna. There are many differences in the maps in Gran Turismo 7 and City Car Driving. Things found in the first that are missing from the latter include: visibility of other cars, and the existence of another map elsewhere on the screen, as well as visibility of the whole track at once. Missing from Gran Turismo 7 is the highlighted optimal route.

And finally, patterns that are found in only one game. Ghosts are found only in Gran Turismo 7, Loading hints in City Car Driving, and both Save Points and Save-load-cycles in TramSim Vienna.

5 Training simulator

This chapter introduces the Tampere Tram Simulator, the training simulator to be gamified. The first section introduces the features and intended use case. The next section the software architecture of the simulator. And in the last section we look at game design patterns that are already implemented in the simulator.

5.1 Introduction

The simulator was placed inside the cabin of a tram replica, and consists of controls matching the real tram's, three large computer screens displaying the driver's view from the cabin, and other equipment related to operating the tram in the dashboard. All controls and equipment are configured to function the same in the simulation, as they do in the real tram. The tram mirrors, which the driver can use to see outside the tram, are displayed on the screens. Creanex Oy [2021] explain that this setup aims to be realistic and to mirror the experience of driving a tram as much as possible. This is to get drivers acquainted with the use of the controls in a real tram cabin. Figure 19 shows some of the simulator equipment and setup.

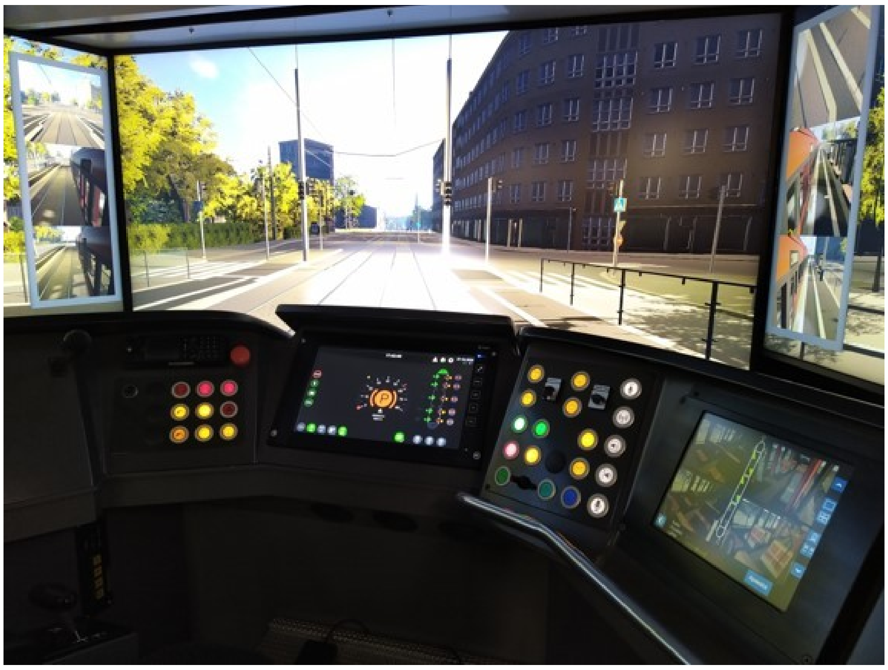


Figure 19: The Tampere Tram Simulator setup.

On release two tram lines were modeled and included in the simulation. The simulation can be updated to include future expansions to the Tampere tram network. Creanex Oy [2021] describe the tram training simulator as a safe environment for drivers and traffic controllers to practice and improve their skills. Creanex Oy [2021] maintain that the simulator had an especially great importance to the driver trainees when the tram lines were not yet operational.

5.2 Architecture

The training simulator software has been split into three separate applications: the training app, the simulator app, and one or multiple visual apps. These three applica-tions communicate with each other via an inter-process communication (IPC) frame-work. The training app has the user interface where the training material, and exercise instructions and results are displayed and managed. The training app user interface is the main entry to the simulator for trainees, and the program can send commands to the simulator app, such as to start an exercise. Teachers can use this app to manage course content and assign courses to trainees. The simulator app contains the majority of the simulation logic, and has the user interface for creating, saving, and running exercises. Teachers can use this user interface to create and edit individual exercises. The user in-terface also has options for controlling and monitoring simulation parameters, among other things. The simulator app controls the visual apps. The visual app is the display for the simulator app showing the simulated environment and game objects themselves, as well as other things needed to be displayed like user interface elements. The visual app has minimal logic and mostly applies textures and models to data sent by the simu-lator app. Figure 20 shows a visualization of the simulator software stack. The trainee and teacher mainly interact with the simulator via the training and visual app. The simu-lator and visual apps can be used without the training app, for example when creating exercises to be added to the training material, or for testing purposes.

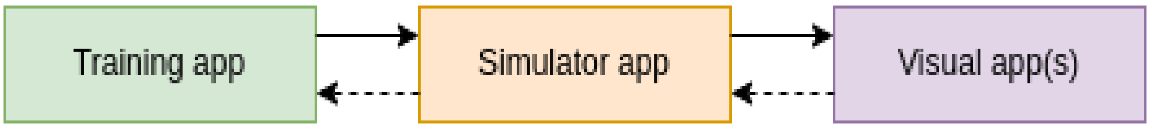


Figure 20:Visualization of the training simulator software stack, and communication between the different parts. Solid line represents the main communication direction, while the dotted line represents less communication.

All three applications use data saved on disk. The visual apps only read data, with the exception of writing into a log file. The visual application loads required assets when the simulator app starts an exercise. The assets stay in the application's memory until it is terminated. The simulator application reads configuration files, and writes and loads exercise files. The training app manages the education material and user data, such as exercise results.

5.3 Game design patterns

Here we will take a look at the game design patterns that are already found in the tram training simulator.

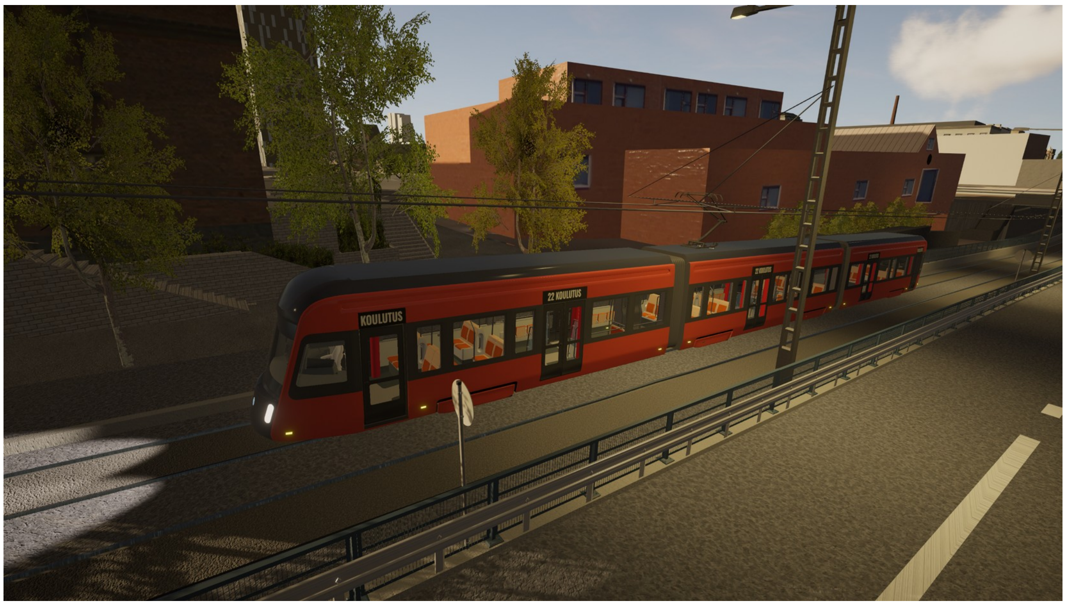


Figure 21: The avatar in the form of a tram model that the trainee controls in the training simulator.

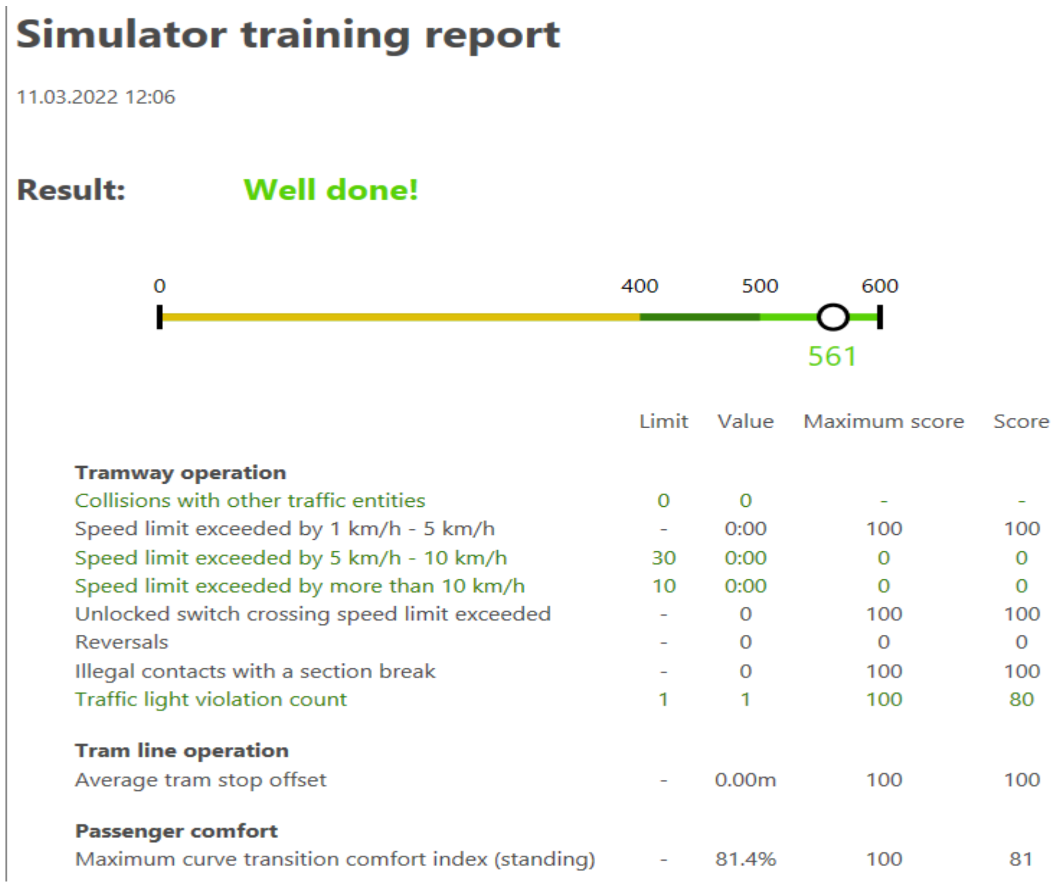
The **Avatar** pattern is present in the form of the tram model the player is driving seen in Figure 21. The tram is the only thing through which the player can affect the game world. The tram avatar's capabilities are not limited in any way in the beginning and it does not gain new capabilities as the training progresses. The simulator allows the user to see the tram model from the outside. This is done by providing a user-controllable flying camera, that when activated, allows the user to explore the game world without the tram. The camera could be used, for example, when the user is unsure of the state of the tram, such as which doors are currently open. Or it can also be used to scout what is ahead on the tram track. The game world contains **Clues** in the form of traffic signs and lights, including those specifically addressed to trams.

The simulation contains the **Enemies** in the form of cars and light traffic (pedestrians). The car traffic is automatically generated and simulated to mostly obey traffic rules such as traffic lights. The pedestrians are instead added to the level by the trainer. The trainer can set the pedestrian to walk in front of the tram while the tram is in motion. The trainee must then notice the pedestrian in time and stop the tram using brakes. Hitting any of the other traffic immediately stops the simulation and results in a training report with a fail marked on it.

Different training exercises can be thought of as **Levels.** Trainers can create exercises with different requirements for passing the exercise. This can be done, for example, by setting a minimum **Score** value that the trainee must exceed through their actions during the exercise. The parameters and actions which affect scoring can be configured. The trainer can add enemies and obstacles to the level as well. One other configurable property is setting target **Time** limits to the exercise. Ability to add levels with differing level of challenged means that **Difficulty Levels** pattern is supported. The trainer can **Replay** an exercise to try to get a better result.

The training simulator implements **Save-load-cycles.** This pattern is used when creating exercises. The **Save Points** can be defined by users, as well as the simulation designer. **Game Pauses** are possible to do during game-play.

The **Score**, the result, and statistics of an exercise are shown and recorded in the training report. The training reports are used by the trainer for tracking progress in the tram driver training. The trainee can access the reports to compare their different attempts. The training reports are generated and displayed after the exercise has ended and thus qualify as **Postponed feedback**. The report contains the scores for different criteria and assessment of the training exercise. An example of the training report can be seen in Figure 22. The colored Figure in the report shows the trainee's achieved Score compared to the target score. It therefore works as a **Progress Indicator**. Progression in the material managed by the training app could also be considered an indicator of progress.



Simulator training report

11.03.2022 12:06

Result: Well done!

	Limit	Value	Maximum score	Score
Tramway operation				
Collisions with other traffic entities	0	0	-	-
Speed limit exceeded by 1 km/h - 5 km/h	-	0:00	100	100
Speed limit exceeded by 5 km/h - 10 km/h	30	0:00	0	0
Speed limit exceeded by more than 10 km/h	10	0:00	0	0
Unlocked switch crossing speed limit exceeded	-	0	100	100
Reversals	-	0	0	0
Illegal contacts with a section break	-	0	100	100
Traffic light violation count	1	1	100	80
Tram line operation				
Average tram stop offset	-	0.00m	100	100
Passenger comfort				
Maximum curve transition comfort index (standing)	-	81.4%	100	81

Figure 22: A screenshot of the training report.

The data collected for the training report is organized into five categories: passenger comfort, startup and shutdown, time, tramway operation, and tram line operation. Some examples for items in the categories can be seen in Table 5.

Data category	**Examples**
Passenger comfort	Maximum curve transition comfort index (standing and seated), Longitudinal acceleration (5 levels)
Startup and shutdown	Successful start procedures, Failed start procedures, Failed shutdown procedures
Time	Total training time, Drive time, Braking time
Tramway operation	Collisions with other traffic entities, Speed limit exceeded (3 levels), Traffic light violation count, Failed checkpoints
Tram line operation	Average arrival time compared to target time, Number of late arrivals, Doors released at stop on the correct/wrong side Correct/Erroneous blinker usages at on-street stops

Table 5: Examples of items in the data categories present in the training report.

Passenger comfort measures things that affect the comfort of passengers, such as the amount of acceleration of the tram. Too much changes in speed can be uncomfortable for passengers, especially those not seated. Startup and shutdown category tracks how correctly the trainee starts the tram for operation, and shuts down the tram after driving. Time category measures times of certain things, such the duration of trainee actions with the tram. Tramway operation category's main interest is the trainee's adherence to traffic rules while driving the tram, and how many traffic offenses the trainee does. Tram line operation is concerned mainly with activities with tram stops, such as adherence to line timetable, blinker usage, and door usage. The trainer can set training ending conditions using these criteria. For example, *collision with other traffic entities* is usually set to abort and stop the training.

6 Evaluation

In this chapter we will discuss differences between the three games and the training simulator, as well as evaluate the suitability of the patterns collected from the games. The patterns in this chapter have been sorted into three categories: essential, recommended, and supplementary patterns, each category and its patterns discussed in its own section. In the final section we will evaluate where in the simulator software stack modifications should be made to implement a given pattern.

My role as an employee of Creanex Oy is software development. My main responsibility is to maintain, add new features, and help direct the evolution of the training application. The improvement suggestions offered in this chapter are intended specifically to the tram training simulator, but most of the suggestions are likely applicable to other simulators as well. I will participate in the implementation of accepted patterns.

The most similar game to the training simulator, at least in terms of the setting, is TramSim Vienna, itself being likewise a tram simulator. Similarity with City Car Driving is in one of the goals of the game, education. In the game's case it is to help the player learn to drive a car, which the training simulator aims to achieve but with a tram. TramSim Vienna and City car driving have simulated traffic like cars and pedestrians, which the player needs to be aware of. Gran Turismo 7 is the most dissimilar to the training simulator. Gran Turismo 7 and City Car Driving both simulate driving a car, and both provide instructive feedback in real-time. City Car Driving offers a lot more feedback, though clearly, driving in traffic has more things to be aware of compared to a race track. The most valuable game to take inspiration from is City Car Driving, simply due to the versatile instructive feedback system. The tutorial levels in TramSim Vienna provide a functional introduction to tram operation and key traffic rules. These could be used as inspiration for demonstration- or practice-type difficulty levels. Gran Turismo 7 has features that help in gaining mastery of the game. An example of this is the ghosts pattern, which is very useful to a player trying to improve their game-play. The character development and reward structure patterns motivate the player to keep playing.

A major limitation in the games we have analyzed is the control scheme: playing with a gamepad, or keyboard and mouse for that matter, does not really teach you on how to operate a car in City Car Driving or Gran Turismo 7. Neither does using a joystick in TramSim Vienna. However, this problem can be alleviated somewhat. For City Car Driving and Gran Turismo 7, players can purchase and plug into their computer or console a racing wheel controller, which includes common car controls like the steering wheel and pedals. TramSim Vienna can be played in virtual reality, which is a more realistic experience. Arguably physical controls, such as racing wheels, are tactile and thus the player can build muscle memory that they could translate to a real car. This is much more difficult to achieve in virtual reality. The player likely needs to look at their virtual “hands” when moving them to press buttons, which could mean taking eyes off the road in traffic.

Games can be played for as long and as many times as the player wants. The tram driving training, on the other hand, has a completion schedule to follow. The simulator is also only a portion of the training program. These facts should be considered when evaluating whether a pattern adds enough value to justify the effort of implementation.

6.1 Essential patterns

It can be argued that the game design patterns shown in Table 6 can be considered essential to training simulators in general. The list here is of course incomplete, since it's missing "foundational" patterns, such as the abstract "feedback" pattern, discussed in Chapter 3. However, to make the virtual experience on which the training simulator is built upon, the foundational patterns must have been implemented. Thus they are not included in this list. The list may also be missing some other important patterns not explicitly discussed in this thesis. Next we will take a closer look at some arguments on why these patterns should be considered essential.

Pattern	**Already implemented?**
Avatars	Yes
Clues	Yes
Difficulty Levels	**Yes***
Levels	Yes
Postponed feedback	Yes
Replay	Yes
Score	**Yes***
Time	Yes

Table 6: Essential game design patterns for training simulators. ***Could be improved**

Avatar of the vehicle being driven or used is required for the trainee to perceive the vehicle’s position compared to the surrounding area and other objects. Allowing the trainee to see the avatar from the outside can be helpful at beginning of the training might be valuable, though not required. Enabling the trainee to see the avatar around them and from mirrors in case of a large vehicle should be enough.

Clues, especially in the environment, are essential. This is because Clues can represent things found in the real world that guide and set restrictions to the operation of a vehicle. In the case of the tram training simulator, these would be the traffic signs.

Difficulty Levels can be considered essential because they allow increasing the difficulty of training exercises gradually. This hopefully prevents the player from getting frustrated when practicing. Having the three types of exercises suggested by Kapp [2012] is valuable. I think the Difficulty Levels in the training simulator could be improved by more clearly implementing and separating these three types. Patterns in Section 6.2, such as Real-time instructive feedback, should be considered when implementing this pattern. They can be used to adjust the difficulty of the simulation.

Levels are required because it's highly impractical in most cases to include the whole training into one game-play section. Being able to split the practice of different skills and maneuvers into separate parts and grading them separately is valuable. It allows the trainer and the trainee see which skills require more attention, among other things.

Postponed feedback is a key element for a training simulator, because the trainee must get information about their successes and mistakes after completing a simulator session. This is an important way for the trainee to know what to improve. This pattern ca be used to provide a clear indication of success is important, which helps the trainee know of what is expected of them. The postponed feedback should generally be stored for later use, so it can be compared to newer results when needed. This is useful for both the trainee, as well as the trainer. The trainer can use the feedback in their assessment of the trainee's skill. It is highly beneficial to combine the score pattern with this pattern. This is because actions in the game world could be made to affect score, positively or negatively, thus helping with the implementation of the postponed feedback. The score could be made visible in the feedback, as a way to give a numerical representation of the trainee's skill.

Replay is essential because it allows failure. And failure is a great way to learn. Arguably, it also lowers the stakes of exercises, thus reducing stress experienced by the trainee. This is especially important in practice-type exercises. Replaying an exercise dedicated to a specific skill repeatedly is one way to build muscle memory and skill.

Score is important because it can be used in grading trainee performance more easily. Actions can be tied to Score. Score is also a clear representation of performance in general. It should be included in Postponed feedback. For the training simulator, I think it is worth investigating whether making the currently accumulated total Score visible during game-play in some type of exercises. This could be done using patterns in Section 6.2.

Time is important because it measures skill. It also important after the simulator training period, for example when working with the machine in the real world in a company. For the training simulator here, it translates to sticking to the tram line schedule, for example. Being fast enough to do emergency procedures, like braking to avoid a crash, is also something to consider.

6.2 Recommended patterns

Pattern	**Already implemented?**
Enemies	**Yes***
Game Pauses	Yes
Ghosts	**No**
Goal Indicators	**Yes***
HUD interfaces	**No**
Progress Indicators	**Yes***
Real-time instructive feedback	**No**
Save-load cycles	Yes
Save Points	Yes
Storytelling	**Yes***

Table 7: Game design patterns that are recommended to be included in the training simulator. ***Could be improved**

The patterns in this section are recommendations on which patterns could be beneficial to a training simulator. The vast majority of patterns here work best in practice-type training sessions, because these patterns mostly offer corrective feedback, helpful information, and in general make the use of the simulator easier. For example, if the goal of the training session was to test the trainee's knowledge and skill in an exam-type session, then real-time instructive feedback informing the trainee about their mistakes could greatly affect the test results. This is because the trainee could then correct their behavior based on the feedback. Some of the patterns here, such as Goal Indicators, Progress Indicators, and Real-time instructive feedback, as well as Postponed feedback in the previous section, all help make the trainee aware of expected actions and behavior. The patterns and their implementation status can be seen in Table 7.

Enemies pattern is recommended for simulators whose setting has to do with places where other people can interfere with safe operation of the machine. Examples of this are traffic and construction sites. Both places can have people doing reckless things, such as jaywalkers in traffic, and careless workers at a construction site. Many machines, such as a tram, can cause major damage in accidents, maybe even death. The trainee's goal here is of course safe operation of the vehicle. In the training simulator, pedestrians have been added as something to look out for. Cars could be also considered as enemies. This would be especially true if they could be used in the same way as pedestrians in the simulator. For this, inspiration could be taken from City Car Driving. However, the traffic simulation as a whole would need to be somewhat deterministic.

Game pauses make the training simulator easier to train with. This allows the trainee to take a break from the simulator without abandoning the exercise. One way the pause can be used is to allow the trainee to stop to think. The pause could even be initiated by the trainer during instruction if the trainer thinks a pause might be beneficial, for example to correct behavior or if the trainer believes some explanation beforehand would better prepare the trainee for an upcoming situation in the simulator. In classroom-type settings an ability to take a break during the class could be appreciated as well. The "danger" of this pattern is that real life doesn't have pauses. Driving a vehicle in real traffic requires constant attention. For example, pausing to think is not possible when quick actions are needed to prevent a traffic accident. This would imply that pauses might not be appropriate in exam-type training sessions, or when the agenda is to mimic real life.

Ghosts may be an useful addition to the training simulation. A good way to use them would be to record professional tram drivers. This way trainees can be shown a ghost of the professional driver’s tram while practicing. Trainees can then try to match their driving to the ghost’s thus possibly enabling learning from professionals.

Visual **Goal** and **Progress Indicators** might be good to have. Both indicators technically exist in the Postponed feedback training report. The progression in the training material as indicated by the training app can also be considered as a Progress Indicator. However, I think implementing them so that they are visible during game-play could be valuable, especially in practice sessions. A Goal Indicator that explains what the trainee should do in a text box in the HUD can solve the problem of trainee’s awareness of what is expected of them. A **Progress Indicator** could simply be an indicator that shows the trainee's current score value in relation to the score threshold required to pass the exercise. For the training simulator, the graphic indicator in the training report might suit this purpose. Care should be taken to prevent trainees from doing the bare-minimum for passing the exercise using these indicators, if that is a concern. As with other patterns in this section, suitability for exam-type sessions should be evaluated, especially if the trainee is expected to remember the goal of the session, like in real life.

HUD interfaces: HUD interfaces are beneficial because implementing patterns such as real-time instructive feedback and visual goal indicators would be impossible to implement without them, at least visually. The HUD would be important as well if a simulator doesn't have separate equipment to display important information. For example, if the physical simulator setup doesn't have a speedometer, the same information should be displayed in the HUD of the program. This applies to other similar devices as well. Ability to hide certain HUD elements might become important when altering the difficulty of a simulator session. Especially exam-type session should hide real-time instructional feedback notifications, mini-maps, and such.

Real-time instructional feedback is arguably the most recommended pattern to implement. The ability to inform the trainee of their mistakes immediately is very valuable. The problem with Postponed feedback is that the trainee might not know when the mistakes were made. This pattern solves that problem. Positive feedback is not as important, but it could improve trainee motivation and maintain correct behavior. It is important that positive and negative feedback are clearly distinguished. In case of visual implementation, color is one way to achieve this, like in City car driving. Icons, such as thumbs up or down, could also be used. Different actions, such as speed limit related, could have different icons and symbols linked to them. Auditory feedback could perhaps also be used alone or in combination with visual feedback. It would be good to make sure the feedback is not too distracting to the trainee.

Save-load cycles: Save-load cycles are beneficial in the practice phase. If the trainer or trainee think that a certain situation or section in the training warrants special practice or training, then this pattern helps achieve that. Again, not appropriate for exam-type sessions. Implementation of **Save Points** is needed to implement this pattern.

Storytelling was recommended by Kapp [2012]. Might be a nice addition to provide some meaningfulness to individual exercises and make them more memorable. Overarching narrative is most likely unnecessary because it requires more effort to implement, and the purpose of the training should be clear to the trainees. The simplest way this pattern can be implemented is by naming exercises in the training material differently. Another simple way is to add narrative text to exercise instructions. This pattern technically already exists in the training simulator because exercise titles and descriptions can already be edited and then displayed. Perhaps adding characters or similar elements could be entertaining to the trainee.

6.3 Supplementary patterns

These patterns, listed in Table 8, are not essential to a training simulator, but could improve the user experience. These patterns mainly provide helpful information, or increase the trainee’s engagement during a training. Arguably, training with the simulator is not as “exciting” as operating the real thing, so these patterns may have some use from that point of view. However, it is appropriate to take care when implementing these patterns because they could make the simulation too easy for the trainee. Another concern is that they can distract the trainee from the real goal of training with the simulator. With these patterns, It is especially worth considering whether the value given by implementing a pattern here is worth the effort required. Though, these patterns should be somewhat easy to implement in most cases. This is because most simulator architectures might already support adding these patterns. Also, these patterns are quite common, so examples and templates for implementation are likely easy to come by. Issues mentioned before, namely the length of the training and the role of simulator, are relevant here.

Supplementary game design patterns	
Pattern	**Already implemented?**
Achievements	**No**
Character development	**No**
Loading hints	**No**
Mini-maps	**No**
Reward structures	**No**

Table 8: Game design patterns that are nonessential to the training simulator but could add value from the perspective of pleasurable user experience.

Achievements are very common in video games. However, the limitation with training simulators is that the simulator is only used during the training period. Therefore, after the training is done, the trainee's achievements would be lost, because the training provider would have no interest in using resources to store them in their systems after the training is over for that individual. Storing the achievement data on a trainee-owned storage medium is possible, but likely not useful. This is because a major point of achievements is for other people to see them. This might also lead to leaking of trade secrets, or other business data, if the achievements reveal something about the training or simulator architecture. Another disadvantage of achievements is that if there are optional ones, a trainee prone to completionism in games would get upset if they were unable to complete them all. A useful application of achievements could be to make them all obligatory to complete via progressing in the training. The trainee could then use them to track their progression towards the end goal, completion of the training.

Character development has some of the same problems as achievements, namely losing data about the progression after the training ends. This pattern has some synergy with the difficulty levels pattern because character development can simply mean gaining new abilities. Slowly gaining new abilities and ways of operation for the simulated machine makes the learning curve of using it potentially more manageable. If we use the pattern this way, then data loss similar to achievements would not feel so severe for the trainee. This is in combination with the other solutions to the problems with the achievements pattern.

Loading hints: Loading hints is a relatively "inoffensive" pattern in that it doesn't affect difficulty levels that much, or otherwise have much of an effect on a training session. Of course, use before an exam-type exercise might accidentally give away some information that is being tested. This of course depends on what kind of information the loading hints generally give to the trainee. Information to give might include things like how to operate the simulator, what functions are available in the simulator, hints about how to generally operate the simulated vehicle, or even hints specific to the exercise being loaded. In the case of the training simulator under study, these could be things like how to enter the flying mode to survey surroundings outside the tram, and what to do when starting the tram. Whether it is worth it to implement this pattern depends on how long it takes to load an exercise in the simulator, and whether there are enough hints to be given. Fast load times don't really give time to read the hints. This could be solved by having the trainee confirm with a button press that they wish to proceed, therefore allowing the trainee to finish reading the hint on the screen. Cycling through only a few hints might cause useless repetition, and thus not make the pattern worthwhile to implement either.

Reward structures: Reward structures, points and badges, are quite similar to Achievements, since Achievements could even be considered a reward structure. Though, Reward structures in general are likely more easily gained, unlike achievements. This is a benefit in a way because the loss of data is less significant to the trainee. Care should be taken to avoid patterns of this type becoming a big focus of the simulator experience, like Kapp [2012] warns.

6.4 Implementation considerations

To implement game design patterns not already present in the simulator requires modifications to the applications in the simulator software stack. An illustration of which applications would need modifications for given pattern can be seen in Table 9. For applicability for other simulators, the software stack could roughly be understood as an implementation of the model-view-controller software pattern, where the visual app would be the view, and the simulator app contains both the model and the controller, with the two parts separated in the simulator app's internal architecture. Next we will look at some justifications for the information presented in the illustration.

Since an **Achievement**'s completion state is user data, they should be mainly stored and managed by the training application, where the user can view them. Achievements could be awarded from both progressing in the training material, and from actions done during an exercise session. The former would be handled by the training app, while the latter would fall onto the simulator app. As mentioned earlier, awarding achievements from progressing in the training material would be more appropriate. However, completing simulator exercises is mandatory for progressing in the material, therefore the simulator application is needed, at least to report the completion of an exercise to the training app. If achievements were to be awarded from actions in the simulation, the visual apps could be used to notify the trainee. To implement the achievements pattern, the observer software design pattern could be used.

The best place for **Character development** pattern is likely, in the case of the training simulator, the training application. In general, the correct place is the component that handles permanent storage and user data. Character development metrics can be displayed in the training application. Restricting implementation only to the training app can make the character development simpler, especially if the only thing considered is advancement through the material. Optionally, character development could also be implemented in the simulator app for more complex character development metrics. These metrics can also be displayed in the visual apps, especially when advancements to character development happen during a simulator session as is likely if the simulator app is made to monitor trainee activities.

Improvements to the **difficulty levels** pattern require modifications to at least the simulator app. This is because the exercises created by the simulator app contain the data of what is included in the exercise. Data to be added could include flags on which HUD elements, or help in general, are visible or enabled. If HUD element visibility is to be changed per exercise, then modifications to the visual app are needed. Modifications to the training app software itself may also not be needed. The already supported changing of exercise instructions and descriptions in the training material are likely enough.

Pattern	**Training app**	**Simulator app**	**Visual app(s)**
Achievements	X	X	X*
Character development	X	X*	X*
Difficulty levels	X*	X	X*
Enemies		X	
Ghosts		X	X
Goal Indicators	X*	X	X
HUD interfaces		X	X
Loading hints		X	X
Mini-maps		X	X
Progress Indicators	X*	X	X
Real-time instructive feed-back		X	X
Reward structures	X	X	X*
Storytelling	X*	X*	X*

Table 9: Illustration of which applications in the training simulator software stack would need to be modified in order to implement patterns not already found in the simulator. *Modification optional.

Enemies are already implemented in the visual app, and modifications need to be directed to the simulator app, and more specifically to the traffic simulation. If the trainer should be able to add cars the same way as pedestrians, then the traffic simulation might need to be more deterministic. The other options is to add randomness to car and pedestrian behavior. This randomness would add unpredictability to the traffic experience, since cars and pedestrians could break traffic rules. Adding nondeterminism to the simulator might interfere with other patterns, however. Ability to configure the randomness could be appropriate to manage difficulty.

Ghosts pattern requires recording the machine's, in this case the tram's, movement and actions in the game world. With the tram this could include movement, acceleration and braking, as well as door operations. The main problem for recording movement is how and where to store the data generated by the recording. Perhaps streaming the recorded data to a file on disk is appropriate. This would be done by the simulator app. The visual app's job is to display the ghost based on the data sent by the simulator app. Another problem is the randomness of the traffic, which can make following the ghost difficult or dangerous. Making the traffic more deterministic might be required in the case of this pattern.

Goal and **Progress Indicators** that are visible during game-play first require the HUD to be implemented. Graphic, or even textual, presentation support needs to be added to the visual app, and the simulator app needs to update those presentations by tracking trainee actions to detect when the next step needs to displayed. Data about the steps should be built into the exercise files. The steps could also be shown in the training app, but modifications to it are not needed if it is simply typed into the exercise's description. Adding a way to hide these elements in the HUD is appropriate for exam-type sessions.

HUD interfaces pattern implementation is required for many of the patterns suggested. The visual app and the simulator app both need to be modified. The visual app renders the actual HUD, and updates and content to it are sent by the simulator app.

Loading hints are displayed by the visual app in the loading screen and their contents are sent by the simulator app. The simulator app could read these from a file on disk. Some logic needs to be implemented if certain hints are not to be displayed for certain types of exercises. Keeping track of which hints the trainee has already seen, or randomizing which hints are shown, could also be appropriate to reduce too much repetition.

Mini-maps: Mini-maps are displayed in the HUD, which in turn is rendered by the visual app. The simulator app needs to track and send the machine's location in the game world constantly so it can be displayed on the map correctly. The map itself could be made by hand, or automatically by processing the game world in some way. Adding some kind of invisible markers or elements to the game world for the processing could help.

Implementing **Real-time instructive feedback** presented during the training session game-play would require analyzing the trainee’s performance and presenting it on-the-fly. This may require collecting more data for analysis. The complexity of the data may require different ways of data storage structures. Quantity of data should also be accounted for storage. The level of detail in feedback in turn determines the complexity of the analysis. Statistical calculations are most likely adequate for simple feedback, such as speed limit calculations. More complex feedback that tracks, for example, chains of actions, could require some form of artificial intelligence type analyzer in some cases. Inspiration for analysis could be gathered from video games, such as the ones discussed in this thesis. Things to consider are:

- What data is relevant from the training point of view?
- What data can be collected?
- How to store the data?
- How to process the data?

Presenting the feedback visually should be done using the HUD.

Reward structures: Tracking actions that trigger rewards should be implemented in the simulator app. This information could then be stored in the training app for display either permanently or for a fixed time after the training session. The other option would be to document the rewards in the training report, which would eliminate, or at least reduce, the need to modify the training app. Displaying the rewards as they are triggered in the visual app is appropriate in cases where they don't interfere the session, such as in exam-type sessions. This pattern can be handled using the same methods, or as part of, real-time instructive feedback.

As said before, **Storytelling** could be improved without any modifications to the software just by changing text related to exercises. Characters with dialogue could be implemented in the same way as in TramSim Vienna and Gran Turismo 7. This would require modifications to the HUD in the visual app, with the simulator app being modified to send content for the dialogue or command which character picture is shown (if needed).

7 Conclusion

The thesis looked for answers to three questions. First, we have found “gamification elements” that should be useful from a trainee’s perspective. In the literature we chose these are called game design patterns. We sorted the patterns into three categories: essential, recommended, and supplementary. There were also some patterns that are foundational to games, and thus simulators because of the same technologies used. However, these patterns were not the focus of the thesis. The essential patterns are called so because they arguably the patterns that make a simulator useful for training. These patterns mainly allow assessment of trainee performance, help manage the difficulty curve of the training, and help divide training with the simulator into appropriate pieces. The patterns in this category are: Avatars, Clues, Difficulty Levels, Levels, Postponed feedback, Replay, Score, and Time. The recommended patterns improve the simulator training effectiveness with real-time feedback structures, as well as allow further adjustment of the training session difficulty. The patterns belonging to this category are: Enemies, Game Pauses, Ghosts, Goal Indicators, HUD Interfaces, Progress Indicators, Real-time instructive feedback, Save-load cycles, Save Points, and Storytelling. Achievements, Character development, Loading hints, Mini-maps, and Reward structures are supplementary patterns. These patterns help with trainee engagement potential commitment to the training, as well as make the simulator operation, depending on the machine, easier. With these patterns care should be taken to not distract the trainee from the goal of the training.

Many of the gamification elements listed above can be found in the training simulator, at least when the whole software stack is considered. The basic building blocks, the essential patterns, are there, as well as most of the recommended patterns. However, there is much room for improvement on the current implementations of the patterns in both categories. Supplementary patterns were not found in the simulator. When considering which patterns to improve or implement, some effort to value estimation should be done, to assess whether a given pattern adds enough business value to be worth the resource expenditure.

The patterns listed above were picked from simulation video games. The games analyzed were TramSim Vienna, City Car Driving, and Gran Turismo 7. The games do offer some usable ideas too. For the training aspect of the simulator, City Car Driving offers the most valuable content. The main reason for this is the feedback system that notifies the player of their mistakes and successes in real-time. Because the notification are immediate, they allow the player to accurately connect their actions to the feedback. TramSim Vienna gives a good introduction to tram driving, with some interactive tutorials on tram controls and relevant traffic rules. Gran Turismo 7 has a big focus on player engagement and commitment to the game. Inspiration could be taken to bring some of these aspects to a training simulator.

The research in this thesis was done with a mixture of literature review of game design patterns, analysis of their implementations in the software industry, and an assessment of their applicability to the tram training simulator under study. To keep the amount of work required by the thesis under control, limitations for the research were made. One of these was with the literature review.

Because there are vast amounts of game design patterns, as well as definitions of gamification, only a small number of them were discussed. Finding of the patterns was not done systematically either. One way the discovery of patterns could have been done more systematically, for example, would have been to focus on only one category or type of patterns. This would reduce some of the subjectivity of the pattern choices, but also make the analysis less diverse. Use of different sources for patterns was done for making the discussion more polyphonic, but the problem was that different sources had differing definitions of patterns.

Another issue concerns the choice of software industry products that were analyzed. Although the games discussed in this thesis can be considered simulators, it is arguable whether they can be considered training simulators. Though, City Car Driving is very close to the definition of a training simulator, if not an actual instance one. Better things to analyze would have been other training simulators. This was considered, but ultimately judged to be difficult. This is due to very limited access to training simulators, among other things. An option would also have been to do a more careful search for games with training elements. Nevertheless, I argue that the games discussed in the thesis offered some acceptable ideas.

There are also some other ways on how the research could have been done, or how the research could be expanded upon. One example is that of the case of usability and user experience design. This could be done by finding usability oversights in the simulator, or even in the game design patterns in general, and offering improvement suggestions. Another expansion would be a case study on the implementation phase of accepted patterns. Here, problems that are encountered during development would be documented, as well as the solutions to those problems. Following implementation, user testing could empirically prove pattern impact.

References

Basten, D. "Gamification," in *IEEE Software*, vol. 34, no. 5, pp. 76-81, 2017, doi: 10.1109/MS.2017.3571581.

Björk, S., and Holopainen, J. 2005. “Patterns in Game Design”. Charles River Media, Boston, MA.

Bogost, I. 2011. “Persuasive Games: Exploitationware”. Game Developer. Accessed 29.5.2023. Available at: *https://www.gamedeveloper.com/design/persuasive-games-exploitationware*

Brathwaite, B., and Schreiber, I. 2008. “Challenges for Game Designers”. Charles River Media, Boston, Ma.

Bus Simulator. 2023. Website. astragon Entertainment GmbH. Accessed 30.5.2023. Available at: *https://www.bussimulator.com/en/*

City Car Driving Home Edition User manual. 2019. “Version 1.5.9.”. Forward Global Group, Ltd.

City Car Driving website. 2023. Forward Global Group, Ltd. Accessed 10.5.2023. Available at: *https://citycardriving.com/*

Collins. 2023. “English Dictionary”. Accessed 29.5.2023. Available at: *https://www.-collinsdictionary.com/us/*

Consistent Walkthroughs. 2022. “Gran Turismo 7 PS5 Gameplay - MY FIRST TIME PLAYING! | Part 1 (GT7 Playstation 5 Gameplay)”. Available at: *https://www.y-outube.com/watch?v=PRMmyOLEIMQ*

Creanex Oy. 2021. “Tampere Tram Simulator”. Accessed 10.4.2023. Available at *https://creanex.fi/en/tampere-tram-simulator/*

Deterding, S., Dixon, D., Khaled, R., & Nacke, L. 2011. “From game design elements to gamefulness: defining gamification”. Proceedings of the 15th International Academic MindTrek Conference: Envisioning Future Media Environments, MindTrek 2011. 11. 9-15. 10.1145/2181037.2181040.

Duolingo. 2023. Website. Accessed 29.5.2023. Available at: *https://www.duolingo.com/*

Error1355. 2022. “Gran Turismo 7 - (PS5) - First Time Startup - Launch Day (March 4th 2022)”. Available at: *https://www.youtube.com/watch?v=p1RHMqWt2eY*

Euro Truck Simulator. 2023. SCS Software s.r.o. Accessed 30.5.2023. Available at: *https://eurotrucksimulator2.com/*

Farming simulator. 2023. Website. Giants software. Accessed 27.4.2023. Available at https://www.farming-simulator.com/about.php?lang=en&country=us&platform=pc

Game design patterns (GDP) wiki. 2023. “Patterns collection”. Accessed 30.5.2023. Available at: *http://virt10.itu.chalmers.se/index.php/Category:Patterns*

Google Trends. 2023. Accessed 27.4.2023. Available at *https://www.google.com/trends*

Gran Theft Auto V. 2023. Website. Rockstar games. Accessed 30.5.2023. Available at: *https://www.rockstargames.com/gta-v*

Gran Turismo 7 online manual. Accessed 6.4.2023. Available at *https://www.gran-turismo.com/us/gt7/manual/*

Gran Turismo website. 2023. Gran Turismo 7 product page. Accessed 6.4.2023. Available at: *https://www.gran-turismo.com/us/products/gt7/*

Gran Turismo Wiki. 2023a. “Gran Turismo 7”. Accessed 27.4.2023. Available at: https://gran-turismo.fandom.com/wiki/Gran_Turismo_7

Gran Turismo Wiki. 2023b. “Penalties”. Accessed 27.4.2023. Available at: https://gran-turismo.fandom.com/wiki/Penalties

Hunicke, R., Leblanc M., and Zubek, R.. 2004. “MDA: A Formal Approach to Game Design and Game Research.” *AAAI Workshop - Technical Report*. Vol. WS-04–04. N.p., 2004. 1–5. Print.

IGN. 2022. Gran Turismo 7 Trophy List. Accessed 31.5.2023. Available at: *https://www.ign.com/wikis/gran-turismo-7/Trophies*

Kapp, K. 2012. Gamification of learning and instruction : game-based methods and strategies for training and education. Pfeiffer.

Lee, J., & Hammer, J. 2011. Gamification in education: What, how, why bother?

Nismonath5. 2022. “Gran Turismo 7 Drag Racing - What Happens When You Drive The WRONG Way?”. Available at: *https://www.youtube.com/watch?v=QXrU-c_oOtg*

Northernlion. 2021a. “Where'd You Learn To Drive??? (City Car Driver)”. Available at: *https://www.youtube.com/watch?v=P5YmxOO3120*

Northernlion. 2021b. “Going Back To Driver's Ed (City Car Driver)”. Available at: *https://youtu.be/oFrfJvAVDVE*

PlayStation website. 2023a. “How to earn trophies on PlayStation consoles”. Accessed 12.5.2023. Available at: *https://www.playstation.com/en-us/support/games/how-to-earn-trophies-on-playstation--consoles/*

PlayStation website. 2023b. Gran Turismo 7. Accessed 6.4.2023. Available at *https://www.playstation.com/en-fi/games/gran-turismo-7/*

RA Shadow. 2022. “GT 7, using the Ghost to improve our times! Tips and Tricks”. Available at: *https://www.youtube.com/watch?v=2b58O94VWu8*

Salen, K. and Zimmerman, E. 2004. “Rules of play: Game design fundamentals”. MIT Press, Cambridge, Ma.

Steam. 2023a. “City Car Driving global achievements”. Valve corporation. Accessed 12.5.2023. Available at: *https://steamcommunity.com/stats/493490/achievements*

Steam. 2023b. “TramSim Vienna global achievements”. Valve corporation. Accessed 19.5.2023. Available at: *https://steamcommunity.com/stats/1314140/achievements*

Steam. 2023c. “TramSim | Patch 1.0.9”. Valve corporation. Accessed 19.5.2023. Available at: *https://store.steampowered.com/news/app/1314140/view/3057350018513745186?l=english*

Steam. 2023d. TramSim Vienna Store page. ViewApp GmbH. Accessed 31.5.52023. Available at: *https://store.steampowered.com/app/1314140/TramSim_Vienna__The_Tram_Simulator/*

Steam. 2023e. City Car Driving Store page. Forward Global Group, Ltd. Accessed 31.5.2023. Available at: *https://store.steampowered.com/app/493490/City_Car_Driving/*

Steamworks. 2023. “Step by Step: Achievements”. Accessed 12.5.2023. Available at: *https://partner.steamgames.com/doc/features/achievements/ach_guide*

TramSim. 2020. Manual. ViewApp GmbH and Aerosoft GmbH.

Wikipedia. 2023a. “HUD (video gaming)”. Accessed 30.5.2023. Available at: *https://en.wikipedia.org/wiki/HUD_(video_gaming)*

Wikipedia. 2023b. “Speedrunning”. Accessed 30.5.2023. Available at: *https://en.wikipedia.org/wiki/Speedrunning*

Wiktionary. 2019. “completionism”. Available at: *https://en.wiktionary.org/wiki/completionism*

Xbox website. 2023. “Tracking Xbox achievements in your game”. Accessed 12.5.2023. Available at: *https://support.xbox.com/en-US/help/games-apps/game-setup-and-play/tracking-achievements-in-your-game*