



Thomas Olip, BSc

# **Implementation of Trading as a Collecting Mechanic in a Novel Racing Game**

## **Master's Thesis**

to achieve the university degree of

Diplom-Ingenieur

Master's degree programme: Software Engineering and Management

submitted to

**Graz University of Technology**

Supervisor

Ass.Prof. Dipl.-Ing. Dr.techn. Johanna Pirker, BSc

Co-Supervisor

Dipl.-Ing. Georg Arbesser-Rastburg, BSc

Institute of Interactive Systems and Data Science

Head: Univ.-Prof. Dipl.-Ing. Dr.techn. Frank Kappe

Graz, February 2024



**GAME LAB GRAZ**

---

## Affidavit

I declare that I have authored this thesis independently, that I have not used other than the declared sources/resources, and that I have explicitly indicated all material which has been quoted either literally or by content from the sources used. The text document uploaded to TUGRAZonline is identical to the present master's thesis.

---

Date

---

Signature

## Eidesstattliche Erklärung

Ich erkläre an Eides statt, dass ich die vorliegende Arbeit selbstständig verfasst, andere als die angegebenen Quellen/Hilfsmittel nicht benutzt, und die den benutzten Quellen wörtlich und inhaltlich entnommenen Stellen als solche kenntlich gemacht habe. Das in TUGRAZonline hochgeladene Textdokument ist mit der vorliegenden Dissertation identisch.

---

Datum

---

Unterschrift

# Abstract

Video games are more popular than ever. In many games, collecting items is an important part of the gameplay, as collecting and completing collections is a key driver and a great motivator to keep playing these games for a long time and to have a fulfilling gaming experience. However, a mechanism for swapping items with other players is much less common in games, and research in this field is rather untouched.

This master's thesis offers basic research on the topic of collecting and exchanging followed by an implementation of an exchange function as a game mechanic. The exchange mechanism is built into a 3D car racing game developed specifically for this master's thesis. To implement this game, a literature review was carried out on the subject of basic game mechanics and racing games. The game offers a street-racing simulation in which players try to set new speed records without colliding with traffic or other obstacles. The high scores are then compared between all players on a leaderboard. Additionally, cars and achievements can be collected.

To ensure that the goals of the work were achieved and the effects of the trading mechanics could be verified, the mechanic was compared with a conventional gathering mechanism in which players collect cars through the game progress. For this, an A/B study was carried out with 39 participants. Depending on the group, the players tested the game either with the traditional gathering mechanism through game progression or with the new trading mechanism. The evaluation showed that both test groups achieved similar results. The differences were rather nuanced and hardly influenced the behavior of the players. In summary, the study showed that adding a trading feature does only have a small influence on the overall game experience, and for most users, the primary game mechanics like racing continued to be in the foreground. Nevertheless, an exchange mechanism can be a good complement to existing gathering mechanisms.

# Kurzfassung

Videospiele sind so verbreitet wie nie zuvor und in vielen Spielen ist das Sammeln von Gegenständen ein wichtiger Bestandteil des Gameplays. Das Sammeln und Vervollständigen von Sammlungen ist eine wichtige Triebfeder und Motivator, um Spiele lange zu spielen und ein wesentlicher Faktor zu einem erfüllten Spielerlebnis. Im Gegensatz dazu ist ein Mechanismus, um Gegenstände mit anderen zu tauschen, deutlich seltener in Spielen zu finden und die Forschung auf diesem Feld eher unberührt.

Diese Masterarbeit bietet eine Grundlagenforschung zum Thema Sammeln und Tauschen, gefolgt von der Implementierung einer solchen Tauschfunktion als Spielmechanik. Der Tauschmechanismus wurde dafür in einem eigens für diese Masterarbeit entwickelten 3D Auto-Rennspiel eingebaut. Um dieses Spiel zu entwickeln, wurde eine Literaturrecherche zum Thema grundlegende Spielmechaniken und Autorennspiele durchgeführt. Bei dem Spiel handelt es sich um eine Straßen-Rennsimulation bei der es das Ziel ist neue Geschwindigkeitsrekorde aufzustellen, ohne dabei mit dem Verkehr oder anderen Hindernissen zu kollidieren. Die Highscores werden dann auf einem Leaderboard verglichen. Zusätzlich können Autos und Achievements gesammelt werden.

Um sicher zu stellen, dass die Ziele der Arbeit erreicht wurden und die Auswirkungen der Tauschfunktion überprüft werden können, wurde die Tauschfunktion mit einem herkömmlichen Sammelmechanismus, bei dem durch Spielfortschritt gesammelt wird, verglichen. Dafür wurde eine A/B Studie mit 39 Teilnehmerinnen und Teilnehmern durchgeführt. Diese testeten, je nach Gruppe, das Spiel entweder mit dem herkömmlichen Sammelmechanismus durch Spielfortschritt oder mit dem neuen Tauschmechanismus. Die Evaluierung zeigte, dass beide Testgruppen ähnliche Ergebnisse erzielten, die Unterschiede eher nuanciert waren und das Spielverhalten kaum beeinflussten. Zusammenfassend zeigt die Studie, dass das Hinzufügen einer Tauschfunktion nur einen geringen Einfluss auf das gesamte Spielerlebnis hatte und für die meisten Spielerinnen und Spieler die primäre Spielmechanik, in dem Fall Rennen fahren, weiterhin im Vordergrund stand. Dennoch kann ein Tauschmechanismus eine gute Ergänzung zu bestehenden Beschaffungsmechanismen sein.



# Acknowledgements

First and foremost, I would like to thank my supervisor Johanna Pirker, who enabled me to choose the topic based on my interests in motorsport and collecting. I would also like to thank Georg Arbesser-Rastburg for proofreading my work and his detailed feedback, as well as Michael Holly and Stephan Keller, who were always helpful and offered support when needed.

Furthermore, I would like to thank all my friends, colleagues, and study participants who supported me, tested my app, and gave feedback.

Finally, I would like to thank my family, without their support, my studies would probably not have been possible.

# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	Goals and Motivation . . . . .	2
1.2	Methodology and Structure . . . . .	2
<b>2</b>	<b>Background and Related Work</b>	<b>3</b>
2.1	Collecting in Video Games . . . . .	3
2.1.1	Collecting in General . . . . .	3
2.1.2	Collecting of Digital Goods . . . . .	4
2.2	Trading in Video Games . . . . .	7
2.3	Fundamental Aspects of Game Design . . . . .	8
2.3.1	Game Mechanics . . . . .	9
2.3.2	3D Engines and Tools . . . . .	13
2.3.3	User Interface . . . . .	14
2.4	Racing Simulations and Games . . . . .	16
2.4.1	Vehicle Physics . . . . .	16
2.4.2	The Perception of Speed . . . . .	17
2.5	Related Games . . . . .	20
2.6	Summary . . . . .	24
<b>3</b>	<b>Design &amp; Conceptual Model</b>	<b>25</b>
3.1	Starting Point and Motivation . . . . .	25
3.2	User Target Group . . . . .	25
3.3	Requirement Analysis . . . . .	26
3.3.1	Functional Requirements . . . . .	26
3.3.2	Non-Functional Requirements . . . . .	26
3.4	Conceptual Architecture . . . . .	28
3.4.1	Storyboard . . . . .	28
3.4.2	Client Architecture . . . . .	28
3.4.3	Server Architecture . . . . .	32
3.5	Game Mechanics . . . . .	32
3.5.1	Level Design . . . . .	34
3.5.2	Racing . . . . .	35
3.5.3	Collecting . . . . .	35
3.5.4	Achievements . . . . .	35
3.5.5	Trading . . . . .	36

3.5.6	Modifications . . . . .	36
3.5.7	In-Game Currency . . . . .	37
3.5.8	Leaderboards . . . . .	37
3.6	Summary . . . . .	37
<b>4</b>	<b>Implementation</b>	<b>38</b>
4.1	Used Software . . . . .	38
4.2	Vehicles . . . . .	39
4.2.1	Vehicle Design . . . . .	40
4.2.2	Vehicle Physics . . . . .	41
4.2.3	Vehicle Data . . . . .	42
4.3	Racing Environment . . . . .	44
4.3.1	Environment Design . . . . .	44
4.3.2	Procedural Environment Generation . . . . .	44
4.3.3	Traffic Generation with Artificial Intelligence . . . . .	49
4.4	Garage and Store Environment . . . . .	52
4.5	User Interface . . . . .	54
4.5.1	Navigation Bar . . . . .	54
4.5.2	Collection . . . . .	54
4.5.3	Market Place . . . . .	59
4.5.4	Events . . . . .	60
4.5.5	Profile . . . . .	62
4.5.6	Racing Interface . . . . .	64
4.5.7	Additional Interfaces . . . . .	64
4.6	Database and Server . . . . .	65
4.6.1	Car Data . . . . .	65
4.6.2	User Data Database . . . . .	66
4.7	Visual Appearance . . . . .	73
4.7.1	Visual Effects . . . . .	73
4.7.2	App Appearance . . . . .	75
4.8	Quest System . . . . .	77
4.9	Summary . . . . .	78
<b>5</b>	<b>Evaluation</b>	<b>80</b>
5.1	Material and Setup . . . . .	80
5.2	Method and Procedure . . . . .	80
5.2.1	Pre-Questionnaire . . . . .	81
5.2.2	A/B-Test . . . . .	81
5.2.3	Post-Questionnaire . . . . .	85
5.3	Participants . . . . .	86
5.4	Results . . . . .	87
5.4.1	System Usability Score . . . . .	87
5.4.2	Game Experience Questionnaire Score . . . . .	89

## Contents

---

5.4.3	Player Activity Statistics . . . . .	91
5.4.4	Player Progress Statistics . . . . .	93
5.5	Discussion . . . . .	94
<b>6</b>	<b>Lessons Learned</b>	<b>95</b>
6.1	Literature Research . . . . .	95
6.2	Design and Implementation . . . . .	95
6.3	Evaluation . . . . .	96
<b>7</b>	<b>Future Work</b>	<b>98</b>
7.1	Research . . . . .	98
7.2	Implementation . . . . .	98
7.3	Evaluation . . . . .	100
<b>8</b>	<b>Conclusion</b>	<b>101</b>
	<b>Bibliography</b>	<b>103</b>
	<b>Ludography</b>	<b>109</b>

# List of Figures

2.1	Structure of seasons. Each season consists of consecutive events, which are composed of several stages. After each season the progress is reset. . . . .	12
2.2	User interface design pattern for scrollable elements on mobile devices. The arrows show the scrolling direction while the color of the items shows the similar elements that are grouped.	15
3.1	Storyboard of the game as a virtual reality game with street and race track mode. . . . .	29
3.2	Quick test of the cars look in Unity using the High Definition Render Pipeline and the demo project. . . . .	30
3.3	Quick test of the cars look in Unreal in the car at the beach scene. . . . .	30
3.4	Overview of all components of the game in the client application and database, grouped in Unity scenes and further broken up into UI screens, collected data, databases, and web interfaces. . . . .	31
3.5	Overview of the requirements for playing the levels, the level division, and the rewards for completing levels. . . . .	34
4.1	A selection of the chosen colors to offer differentiation and variation for the vehicles. . . . .	39
4.2	The Teron Kanaani sports car with opened wing doors. Modeled in Blender and rendered in Blender's render engine Eevee.	40
4.3	The interior of the Kanaani, developed in Blender and rendered in Blender's render engine Cycles. . . . .	41
4.4	The screenshot from the configuration of Edy's Vehicle Physics Pro shows the performance curve of the configured sports car.	42
4.5	Content of the cars.json file, showing all important variables of the vehicles. . . . .	43
4.6	Autobahn as an environment for the traffic simulation, modeled and rendered in Blender. . . . .	44
4.7	Autobahn with varying decoration models, set up in Unity. .	45

## List of Figures

---

4.8	A selection of sections to generate the Autobahn. From left to right: noise barriers on narrow lanes, lane widening, lane with concrete barriers and emergency lanes, lane narrowing and tunnel. . . . .	46
4.9	The main functions of the Road Generator script show the initialization of new sections and removal of old sections by Sobaihi (2017). . . . .	47
4.10	The improved and extended Fixed-Update function of the Road Generator script. This shows the random loading of new layouts for the street and the corresponding placement in the order. . . . .	48
4.11	The red, yellow, and orange dots on the street show the points for the A-Star path. Each line has a group of dots. . . . .	50
4.12	The generated street with decoration and randomly spawned vehicles on it. . . . .	50
4.13	Functions created to implement random spawning vehicles on the street. . . . .	51
4.14	The garage is implemented as a plateau on an island with a rotating base plate to be able to view the vehicle from all sides without changing the viewing angle. . . . .	52
4.15	The store environment is implemented as a simple dark room to keep the focus on the vehicle. . . . .	53
4.16	Standard view which shows the car in the garage and the navigation bar. . . . .	54
4.17	The user interface of the collection shows several entries with different states. The first car in the row is owned, the second was owned at one point but was traded away, the third field shows an unknown vehicle with the number 3 and the first entry in the second row shows a vehicle that was seen, but was never owned. . . . .	55
4.18	A pop-up with an animated background celebrating the receipt of a new vehicle. . . . .	56
4.19	The car progress screen shows the distance covered, as well as new speed records and trophies obtained with the vehicle. . . . .	57
4.20	The car's level is dynamically computed through a simple process, where queries add up the number, resulting in the current level. . . . .	57
4.21	Detail view of the car and user interface showing a description, statistics, mastery, and upgrades with the particular car. A previous owner history and a separate upgrade screen are displayed with the note 'coming soon...'. . . . .	58

## List of Figures

---

4.22	View of the integrated marketplace. On the left side is the inventory from which vehicles can be selected for sale. On the right side, the player will find current offers from other players or the manufacturer. . . . .	59
4.23	Alternative and simplified interface of the marketplace with a more elegant and simplified presentation. . . . .	60
4.24	The event page shows current running events including a rank, leader board, and medal count. . . . .	61
4.25	Preview of upcoming events for different vehicle classes and on different maps and terrains. . . . .	61
4.26	The profile screen shows the user name, avatar, statistics, world rank, and achievements. . . . .	62
4.27	This code shows the calculation and user interface generation of the achievements. . . . .	63
4.28	User interface during the speed run showing a gauge cluster in the lower middle of the screen, a pause button in the upper left corner, and the level progress in the upper middle. . . . .	64
4.29	The pause screen displays a button to continue the race and one to end it, on a darkened screen. . . . .	65
4.30	Visual presentation to give an overview of the data exchange between application, user, and server. Temporary data stands for data related to the exchange function that is stored centrally on the server and is only displayed. . . . .	66
4.31	C# code in Unity that transmits the data to the server using Unity WebRequest. The transmission is only partially encrypted but will be fully encrypted in the future. . . . .	67
4.32	This code shows the GetScores function where a string from the server is split to show the results on the leaderboard. . . . .	68
4.33	Excerpt from the function to register new users, programmed in PHP. . . . .	69
4.34	Shortened version of the PHP code for data transmission to the server using the update function. . . . .	70
4.35	Excerpt from the DisplayLeaderboard PHP function loading data to show the top ten leaderboard entries. . . . .	71
4.36	PHP code to display the user's position in the leaderboard, including the high score, the vehicle, and the date the record was set. . . . .	72
4.37	A screenshot of the database for the ranking lists and with data to evaluate the study. The upper entries show the old logging implementation while the last entry shows the updated implementation with user accounts with unique user IDs and log-in functionality. . . . .	72

## List of Figures

---

4.38	Settings screen showing the keymap, audio settings, and quality settings. . . . .	74
4.39	The app icon, which was developed for iOS, Android, macOS, and Windows. It is composed of the title of the app and the Teron Kanaani headlight. . . . .	76
4.40	The cover is optimized for websites and game sections in app stores and shows the alternative title. . . . .	76
4.41	The figure shows the welcome screen where the testers choose a username, and specify their age and sex. . . . .	77
4.42	Example of a quest displayed in the upper right corner of the in-game screen. . . . .	78
4.43	The figure shows the screen for the questionnaires SUS and GEQ in a scroll view. . . . .	78
5.1	Flowchart of the A/B test sequence from start to the questionnaires which mark the end of the test. . . . .	82
5.2	Comparison of the SUS results from the test group without a trading function (Group A, blue bar) and the test group with a trading function (Group A, orange bar), extended with Bangor et al. (2009)'s adjective rating scale. . . . .	88
5.3	Comparison of the Game Experience Questionnaire results of the two test groups. . . . .	90
5.4	This figure shows the click statistics of the UI tabs in the game. The clicks were counted when clicking on the corresponding tabs. . . . .	91



# List of Tables

3.1	Intended Game Mechanics for the game including their description. . . . .	33
3.2	Decision-making based on the play experience cycle. . . . .	33
5.1	List of quests that users receive during testing. The letters after the numbers show the test group the quests are shown to.	83
5.2	Listing of the data collected during user testing. . . . .	84
5.3	Questions of the System Usability Scale Test. . . . .	85
5.4	Questions of the Game Experience Questionnaire (In-Game Module). . . . .	86
5.5	Participants broken down by age and sex. . . . .	87
5.6	Results of the System Usability Score Questionnaire, showing the mean, standard deviation, and the statistical significance with $\alpha = .05$ . . . . .	89
5.7	Breakdown of the results of the Game Experience Questionnaire, showing the mean, standard deviations, and the statistical significance with $\alpha = .05$ . . . . .	90
5.8	Detailed display of user activity statistics including mean, standard deviations, and the statistical significance with $\alpha = .05$ .	92
5.9	Comparison of the player progress statistics of the two test groups, showing the mean, standard deviations, and the statistical significance with $\alpha = .05$ . . . . .	93

# 1 Introduction

Collecting and trading have always been an essential part of human culture. The acquisition, possession, and exchange of items have shaped societies and driven progress. In the digital world, collecting and trading are widespread, including in video games. Collecting in video games has a long history, and sometimes trading is used as a gathering mechanism to expand or complete collections. A prominent example of this is Niantic's *Pokémon Go*<sup>1</sup>, one of the most well-known representatives of the Pokémon game series, in which Pokémon can be collected and traded (Niantic, 2022).

In car racing games and simulations, one of the integral genres since the beginning of the history of video games (Siemens et al., 2021), collecting vehicles is very popular. The idea of collecting cars in racing games has gained popularity and sophistication over time. Players can collect a wide range of vehicles in most modern car racing games. These vehicles can be acquired by players through game progression, in-game currency purchases, real money transactions, and sometimes also by trading with other players.

Trading of cars has been found in racing games for a long time too, but the function is not nearly as widespread as other gathering mechanisms. For example, *Gran Turismo 5*<sup>2</sup> included such a car trading function (Gran Turismo Wiki, 2022), but this is no longer available in the subsequent titles of the franchise. The reasons for the removal of the feature remained largely unclear. Possible reasons could be economic considerations, too high complexity, problems with game balance, or even abusive or fraudulent use of the feature. For this master thesis, a study was conducted to shed light on this and to answer the question of whether these potential disadvantages outweigh the potential advantages of a more complex and interesting gaming experience.

---

<sup>1</sup> Niantic, 2016. <https://pokemongolive.com>.

<sup>2</sup> Polyphony Digital, 2010. <https://www.gran-turismo.com/de/products/gt5/>.

## 1.1 Goals and Motivation

To answer what the advantages and disadvantages of such a trading function are and whether it is worth integrating such a feature in video games, a literature review should be conducted that explores the psychological backgrounds behind trading and collecting in general.

Furthermore, the literature review should collect information on what other fundamental aspects need to be considered in order to implement a successful collection and trading game. Additionally, games that successfully use these features should be highlighted.

After determining the advantages, disadvantages, and other requirements of a collecting and trading game, it should then be determined how such a game could be implemented. Using examples, it should be shown how potential problems with game balance, increased complexity, abuse, fraud, or economic losses could be avoided and at the same time, the positive aspects of a trading function could be implemented.

To test the actual feasibility of implementing such a trading and collecting feature in a game, a reference solution should be implemented within a practical example that should then be checked by user tests. The user tests should answer the questions about usability, user experience, and the general impact of a trading feature in video games.

After evaluating the user tests, a clear picture should be able to be given as to whether such a trading feature makes sense for user satisfaction and consequently also for the game manufacturers.

## 1.2 Methodology and Structure

The structure of the master's thesis essentially follows the sequence of the objectives. The following Chapter 2 explains the background and related work, followed by Chapter 3 in which the design and concept of the reference implementation are described. In the subsequent Chapter 4, it is described how the game was implemented. In the evaluation chapter (Chapter 5), the methodology and procedure are discussed and the results of the study are presented. The lessons learned follow in Chapter 6 and an outlook on future work in Chapter 7. Finally, Chapter 8 summarizes the findings in a conclusion.

## 2 Background and Related Work

Scientific research in the field of video games is a fast growing domain that combines various disciplines, including psychology, computer science, and neuroscience. Researchers are particularly interested in understanding the cognitive processes involved in gameplay, such as problem-solving skills, spatial awareness, and reaction times. While there already exists much literature about collecting, there is almost no literature available about trading in the context of video games or digital assets. Therefore, this chapter takes a step back and focuses on collecting and trading in general before diving into the fundamentals of game design, which are the foundation for implementing trading successfully. The topic of racing games and simulations is then discussed, which deals with driving physics and the perception of speed.

### 2.1 Collecting in Video Games

When considering the topic of trading digital goods in video games, there is no getting around the topic of collecting - a major reason and motivator for trading assets, as trading is often used as a gathering mechanism to expand or complete a collection. Collecting is also a big topic outside of video games and is present in many areas of people's lives (McKinley, 2007; Snow et al., 2015). It is therefore not surprising that there is a lot of literature on the subject. However, since this work focuses on video games, the scope of this chapter will be mostly confined to the collection of digital goods like those described by Koh and Kerne (2006) and Keene (1998). Keene described building digital collections for museums and Koh and Kerne how students collected and managed collections.

#### 2.1.1 Collecting in General

A collection, in general, is defined by Belk et al. (1991) as follows:

"The selective, active, and longitudinal acquisition, possession, and disposition of an interrelated set of differentiated objects (material things, ideas,

beings, or experiences) that contribute to and derive extraordinary meaning from the entity (the collection) that this set is perceived to constitute.”

The practical utility, aesthetic appeal, and sentimental value of an object are key factors that influence how people relate to material possessions. This concept was explored in depth by Danet and Katriel (1989), who discussed the various motivations and behaviors associated with collecting objects. Danet and Katriel suggested that the act of collecting can be driven by a range of factors, such as the inherent value or usefulness of the items, their aesthetic qualities, or the emotional attachment they inspire.

In addition to these factors, the personality traits of the individuals also play a significant role. Gosling et al. (2003) discussed the Big Five personality dimensions, which include openness, conscientiousness, extraversion, agreeableness, and neuroticism. These traits can influence how individuals interact with objects and value objects. For example, someone high in openness might be more inclined to appreciate the aesthetic qualities of an object, while someone high in conscientiousness might value the practical utility of an object.

Pearce (1994b) underscored the complexity of defining the subject of collecting, which is probably why there is a lot of literature on the subject. More reasons to collect were found by McKinley (2007), who delved into the vast diversity in people’s collecting interests, ranging from common items to exotic and even bizarre objects. He postulated that people collect for various reasons, including hobby, investment, and professional collecting. Rykwert (2001) echoed this sentiment, considering the historical and cultural significance of collecting. He suggested that people have been driven to collect a variety of objects, sometimes unlikely ones, throughout the ages. In a more contemporary context, Watson et al. (2014) discussed the design of a gameful system that supports collectors. They drew attention to the significance of the collection’s story, structure, and authenticity and stressed the necessity of encouraging emergent behavior.

### 2.1.2 Collecting of Digital Goods

The process of collecting digital objects is intrinsically tied to the perceived value and the meaning these objects hold for the collector. This concept was first discussed by Pearce (1994a), who emphasized the significance of the connection between collectors and their collections. Pearce suggested that collections could be divided into three major categories: Souvenir, Fetishistic, and Systematic Collections. This categorization provides a comprehensive

understanding of why individuals collect objects and the value they attribute to them.

Building on Pearce's work, Livingston et al. (2014) and Manninen and Kujanpää (2007) further explored this concept. They focused on the topic of digital objects, particularly related to Massively Multiplayer Online Games (MMOGs). In these virtual environments, players often collect digital assets, such as game characters or avatars. The players collect or produces those assets while playing the game through game progressing (Manninen & Kujanpää, 2007). The value of these digital objects is not just functional but also symbolic, representing the player's achievements, status, and identity within the game world.

Yee (2006) provided a valuable framework for understanding different types of values that can be attributed to virtual objects. Yee identified various motivations for players in online games, which can also be applied to understand the value attributed to digital objects. These motivations include aspects like achievement, immersion, social interaction, and more.

Livingston et al. (2014) used Yee's framework to identify values of MMOGs. These values, detailed in their work, serve as key motivators for collectors. They identified different values, players associate with their game characters. These values are summarized: Utility, Investment, Communication, Memory, Enjoyment, Relationship, New Experience, Creativity, Sociability, and Self-Expression.

In a follow-up study conducted by Toups Dugas et al. (2016), an in-depth analysis was carried out on the types of game objects that players preferred. The game objects included characters, gears, skins, and vehicles. This research aimed to understand player behavior and preferences in a more nuanced way, contributing to the development of more engaging and satisfying gaming experiences.

In addition to the types of value and the different types of digital items, there are also different types of collections, according to Watkins et al. (2015):

- **Pursued Collections:** A pursued collection is characterized by clear boundaries and a goal of what it should look like in the end. Examples include collecting special cars in racing games or collecting achievements. It depends on the knowledge and skills of the player, and is goal-oriented. The players see meaning, a kind of achievement, and mastery in it.
- **Evolving Collections:** A developing collection has flexible boundaries and the goal is rather vague. The player sees the collection more as a review of their activity, or, using the example of a music collection,

as a representation of his taste. When collecting, excitement and anticipation predominate, especially when surprising discoveries are made.

- **Emerging Collections:** An emerging collection does not contain a concrete goal and only arises passively during the activity without thinking about the collecting itself. An example would be digital films, photos, or books. Nevertheless, the collection can have value as it represents taste and provides a look back.

A different approach is offered by Hamari and Tuunanen (2014), who introduced the concept of player types in the context of digital games. They proposed seven primary dimensions to categorize players: Intensity, Achievement, Exploration, Sociability, Domination, Immersion, and In-game demographics. These player types can influence how individuals value and interact with games, similar to how personality traits can influence how individuals value and interact with objects. Hamari and Tuunanen's player types were described in detail in their paper. For instance, Achievers might collect items as a way to demonstrate their progress and status. Explorers, on the other hand, might be driven by the thrill of discovering new items or completing collections. Sociability players could engage in collecting as a means to interact with others, trading or gifting items. Domination players might collect rare or powerful items to gain advantages over others. Immersion players might collect items that contribute to the richness of the game world and their in-game experience. Lastly, Intensity and In-game demographics could influence the pace and style of collecting, with more intense players potentially collecting at a faster rate, and demographic factors like in-game roles influencing what types of items are collected.

In summary, collecting is a complex activity with various motivations and methods. It can involve a wide range of objects. The act of collecting has historical and cultural significance, and the design of video games that support collecting can benefit from considering factors like narrative, organization, and authenticity. The value individuals assign to objects or games are influenced by a number of factors, including the inherent qualities of the object or game, the individual's personality traits, and their player type. The interplay of these factors creates a compelling need to collect, driving individuals to seek out and acquire these objects. This dynamic is not just limited to digital collections but is a fundamental aspect of the broader phenomenon of collecting.

## 2.2 Trading in Video Games

To shed light on the topic of trading in video games, one should deal with the topic of virtual currencies like Asadi and Hemadi (2018) who gave an overview of how virtual currencies work in video games. Because tradable goods can also be considered as currency and one of the five types of video games according to Adams and Dormans (2012) besides physics, progression mechanics, tactical maneuvering, and social interactions is the internal economy. This includes resources that function as virtual currency within video games.

When trading between players in a game is enabled, it should be noted that with the tradability, markets outside the game can also emerge and these can influence the game (Guo & Barnes, 2007). This need for virtual goods has been investigated by Hamari and Lehdonvirta (2010). In the paper by Snay (2021), the influences and effects on a game are described when players have the option to purchase in-game currency with real money or from external sources, instead of earning it by investing time into the game. In many games, the playing time can be drastically shortened and the goals in the game can be achieved faster. From a certain desirability of the game assets, it is therefore to be expected, whether wanted or unwanted, that such a market could arise outside the game. Therefore, it is necessary to consider how to deal with this phenomenon and which strategies are applicable, as described by Lehdonvirta (2010). For example, limitations on trading could make buying with real money less attractive or the tradable resources are only limited to game components that do not offer competitive advantages over fellow players and serve only visual purposes, for example.

A real-life example of a game with such a restricted trading feature is the Pokémon series. There, in most cases, the collection of Pokémon can only be completed using the swap function. Without swapping and the associated interaction with other players, completing the Pokédex is not possible or only possible with increased difficulty. Even though there is a secondary market where Pokémon are sold for real money, this is heavily restricted. For example, by the threat of bans from Niantic, the developer of *Pokémon Go*<sup>1</sup>. Another method to restrict the secondary market is by limiting trading so that each Pokémon is only tradable once and by changing the individual values of Pokémon during the trade. This prevents the targeted trade and collection of Pokémon with perfect values (Game Press, 2018). Due to the continuous integration of this game mechanic in the Pokémon series, one might assume that this feature is successful and popular among players of this game series.

---

<sup>1</sup> Niantic, 2016. <https://pokemongolive.com>.



Another way to prevent an uncontrolled secondary market would be to integrate a market within the game. An uncontrolled secondary market and so-called Real Money Trading (RMT) can lead to numerous side effects for game manufacturers. These include a worse gameplay experience and imbalance in the game. They could also undermine game mechanics and give wealthier players advantages. Furthermore, they could trigger inflation of the game currency. This is due to the easier purchasing option through real money, as described by Snay (2021).

Another unwanted effect could be, according to Olivetti (2011), a focus on trading itself, away from the actual purpose of playing the game for enjoyment. However, there are also positive effects as described by Zhang (2014), where it is described that a free economy and its efficiency can also be attractive for players. It's important to consider the impact on the external market when a game, which was initially free, introduces chargeable items that are tradable (Oh & Ryu, 2007). The introduction of these items might influence the dynamics of the market outside the game. This change can be attributed to nested network effects, the types of accessories offered, the shift towards monetization through virtual items, and player motivation. The interplay of these factors necessitates a careful evaluation of the potential impacts on the market.

Urschel (2011) explored the topic of real money trading in Massively Multi-player Online Role-Playing Games (MMORPGs). He investigated the impact of real money trading on game balance and player behavior, as well as the motivations and attitudes of the trading participants. He also analyzed the legal and ethical issues of real money trading and proposed some guidelines and recommendations for game developers, regulators, and players.

To sum it up, trading items in games with other players is a common and popular activity in many online games. It can offer benefits such as acquiring desired items, making friends, and feeling rewarded. However, it can also pose risks such as losing money, items, or accounts, breaking the game rules or design, and may lead to a focus on trading. Therefore, players should be careful and responsible when engaging in this kind of trading, and follow the game's policies and guidelines.

### 2.3 Fundamental Aspects of Game Design

To develop a game based on the previous findings on the topic of trading and collecting in the practical part, it is necessary to consider the basic factors that make a good game. Based on the many factors that make a successful game (Ahmad et al., 2017), there are different frameworks to

capture these (Aleem et al., 2016). There is the Design, Play, and Experience (DPE) framework (Winn, 2009), the Mechanics, Dynamics, and Aesthetics (MDA) framework (Hunicke et al., 2004), and the Design, Dynamics, and Experience (DDE) framework based on it (Walk et al., 2017). They divide the games into different components. The papers aim to better define gameplay and fun in video games. Underlying all of this are the game mechanics, which is the basic functioning of the game.

### 2.3.1 Game Mechanics

In general, game mechanics are very important in video games and simulations. They are defined as follows, according to Boller (2013):

“A game’s mechanics are the rules and procedures that guide the player and the game response to the player’s moves or actions. Through the mechanics you create, you define how the game is going to work for the people who play it. So just to be clear, the mechanics describe rules the player follows and the rules the game itself follows.”

Salen and Zimmerman (2004) described that games are a construction of rules too. Game mechanics are a crucial aspect of game design, but their implementation alone is not sufficient. They must also provide an enjoyable experience to the player to be considered effective. According to Fabricatore (2007), the cycle of play experience should be the primary source of enjoyment in games. This cycle encapsulates the player’s journey through the game, from understanding the rules to strategizing and playing. The cycle consists of information gathering, information analysis, decisions, and interactions with the gaming world. This cycle then repeats itself throughout the entire game. Each game mechanic should ideally align with this cycle to enhance the overall fun factor. By adhering to this principle, game designers can ensure that their mechanics not only function as intended but also contribute positively to the player’s experience. A positive, more engaging, and satisfying gameplay experience, ultimately could make the game more successful.

For instance, the dynamics of a car racing video game can be considered. Initially, players are presented with a variety of car models, each with distinct distributed performance parameters. Players must then engage in a process of analysis, weighing the pros and cons of each model based on factors such as speed, handling, and durability. This decision-making process is a crucial part of the game experience, adding a strategic element to the gameplay. Once players have made their choice, they proceed to purchase their preferred model, marking the beginning of their racing

journey. This can add depth to the gameplay, enhancing player engagement and satisfaction.

### **Levels Design**

Level design is crucial for creating variety in games so that the game mechanics don't become monotonous. Multiple levels with different maps and designs can provide variety. Also increasing the difficulty of the levels over time or a story can help to keep players motivated and engaged. A storytelling aspect can add depth to the game, making it more immersive and emotionally engaging (Lebowitz & Klug, 2011). Additionally, varying levels of difficulty across different stages can pose new challenges to players, requiring them to adapt their strategies and skills. A game's level is based on two factors, namely a reward for completing a level and the difficulty of the level:

"Each element has two critical features: a reward (e.g., earning an item or being able to watch a cinematic) and a degree of difficulty (e.g., how much energy or focus is needed to interact with the game element)." - Li et al. (2021)

These must then be set in a reward-to-difficulty relation to optimize the level design. A popular variant in level design is increasing the difficulty with each additional level, for example in a linear way. Typically, the rise in difficulty is balanced out by the enhancement of player skills or the availability of optional upgrades that offset the heightened challenge.

### **Upgrades and Farming**

To keep up with the increasing difficulty of the levels, in addition to constantly improving player skills, there is also the possibility that the player uses upgrades to cushion the increased difficulty. The principle of upgrading in games is a delicate balance that developers must maintain. Upgrades are typically limited to a certain extent to prevent any unforeseen impacts on gameplay. One possible way to implement upgrades is to implement an in-game currency that can be accumulated through successfully completing levels. This repetitive method of collecting is a common practice in video games and is also called farming (Rouse, 2017).

If all in-game objects are upgraded to their maximum potential, the game may lose its appeal, as players no longer have goals to strive for. To counter this, developers often introduce new items or challenges to keep players engaged and invested in the game.

One solution to this issue is the introduction of upgrade drops with a certain probability, also called Random Reward Mechanisms (Holmberg & Modée, 2021; Yin & Xiao, 2022). Unlike traditional purchases, these drops add an element of chance to the acquisition of upgrades. This means that players do not know exactly what they will receive when they get new items, which can make the game more exciting and unpredictable. These dropped upgrades could also benefit from an exchange function using which they can be exchanged with other players to complete vehicles.

This mechanic could not only sustain player interest but also extend the lifespan of the game. Even without the addition of new content, the random nature of these item drops can keep players coming back in the hopes of obtaining rare or powerful upgrades. Therefore, through careful implementation of upgrades and mechanics like this, developers can ensure a more sustainable and engaging gaming experience. Although Bergeron (2010) criticized how current implementations of loot do not improve the overall gameplay and suggests an improvement in which the probability of rare items dropping should be slightly increased with each drop.

### **Leaderboards**

Another popular game mechanic is the use of leaderboards. As Butler (2013) pointed out, these ranking systems are simple to incorporate into games and provide a means of quantifying 'bragging rights', which in turn encourages competition. The study also described a polarization of players and, among other effects, an effect that players at the bottom of the leaderboards play the game more often than players in other positions.

### **Seasons**

Establishing a trading feature requires not only a successful game and a large user base but also a long runtime of the game. But levels and upgrades pose a problem: When all levels have been played through and all upgrades have been installed, the game is usually over, it is played through. Some game manufacturers rely on so-called Downloadable Content (DLC) to extend the lifecycle of the game (Dey & Lahiri, 2013). Especially with games that consist of very complex levels, the continuous creation of new levels is often not economical and at a certain point is also limited by the available device memory for the game.

Therefore, game developers must develop successors to their games. Recently, however, many manufacturers have switched from their upfront

payment business model to microtransactions, battle passes, and other forms of monetization (Joseph, 2021; Petrovskaya & Zendle, 2020). These forms of monetization increase the lifespan of a game without having to make further large investments in development because the game is kept alive with small content updates. These content updates are often combined with battle passes and so-called seasons. The term 'season' in video games is typically a set of events or activities where players can compete on a ranked ladder, win rewards, or take part in specialized activities in a given timeframe.

The so-called battle pass is mostly a premium bundle that unlocks this content in exchange for in-game transactions of the users. It increases the lifespan of the game by introducing new content like skins or by simply resetting the leaderboards. Real racing series and other tournaments could be seen as prototype models: Every year, races are held on mostly the same routes. The racetracks are the same, but the rules might be slightly modified, the drivers and manufacturers' points are to zero again, and the drivers and teams fight for a new iteration of the championship. Figure 2.1 illustrates how such a structure can look like. The stages are different car races, while the events are race weekends and the seasons are world championships, for example.



Figure 2.1: Structure of seasons. Each season consists of consecutive events, which are composed of several stages. After each season the progress is reset.

### Quests

A quest system serves a dual purpose in the context of a study as well as in the onboarding of new players in a game. For study participants, it provides a structured pathway, guiding them through various stages of the study. This systematic approach ensures that the participants understand and follow the study's procedures, thereby enhancing the quality of the research findings.

On the other hand, for new players, a quest system acts as an interactive tutorial. It introduces them to the game's mechanics and features in a fun and engaging manner. By completing quests, players learn how to navigate

the game environment, use items, interact with characters, and more. This playful learning process not only helps players understand the game better but also enhances their overall gaming experience. A design pattern for a quest system is the so-called Arrowhead Questioning, in which many broad quests lead to a small number of specific quests with greater significance (Smith et al., 2011). For example, in a racing game, these would be simple goals such as completing a certain distance or beating a certain lap time. More advanced quests might set harder-to-achieve goals, such as beating a certain lap time in combination with the requirement to finish first and not cause collisions.

In conclusion, a quest system is an effective tool that benefits both study participants and new players by providing guidance and facilitating learning in an enjoyable way.

### 2.3.2 3D Engines and Tools

“With the growing maturity of computer technology and the rapid development of the video game industry, players have higher needs for the game display technology. They are no longer satisfied with the 2D game, yet hope to experience a more authentic 3D scene in the 2D display.”

- Su (2018)

According to the research conducted by KBV Research (2022), it has been observed that 3D game engines have a significant market dominance over 2D game engines. The growth of the market is steady and they also predicted a further increase in the future. The factors contributing to this dominance could be attributed to the immersive and realistic gaming experience that 3D engines provide, which is highly sought after in the current gaming market.

### 3D Game Engines

Among these 3D game engines, the Unity Game Engine (Unity Technologies, 2021) and the Unreal Game Engine (Epic Games, 2021) stand out as the most popular choices. The Unity Game Engine, known for its versatility and user-friendly interface, is widely used by both indie developers and large gaming studios. It supports a broad range of platforms and offers a robust set of tools for game development. On the other hand, the Unreal Game Engine is renowned for its high-end graphics capabilities and is often the go-to choice for AAA game titles. It provides developers with a powerful platform to create visually stunning and complex games. Both engines support Virtual

Reality (VR) headsets, which are one of the latest hardware innovations in gaming and simulation.

In conclusion, the shift towards more immersive and visually engaging games has led to the rise of 3D game engines, with Unity and Unreal dominating the market. This trend is expected to continue as advancements in technology further enhance the capabilities of these engines.

### 3D Creation Tools

3D models form the visual foundation of any 3D game. They are the digital structures that give life to the characters, environments, and objects within the game world. In games where the primary focus is on collecting, the aesthetic appeal of these 3D models becomes particularly crucial. Players are drawn to collect these exquisitely made models because of the way they look and not only because of their functionality.

Developing these 3D models and animations requires specialized software, and one such tool is Blender (Blender Foundation, 2021). Blender is a free tool for developing 3D content. The functions range from 3D modeling to rigging, animation, and rendering to functions such as video editing and motion tracking.

Blender's versatility and robust features make it an ideal choice for game developers. Its ability to produce high-quality 3D models and animations can significantly enhance the visual appeal of a game. Moreover, being open-source, it has a large community of users who continually contribute to its development and improvement. This ensures that Blender stays up-to-date with the latest trends and technologies in 3D content creation.

### 2.3.3 User Interface

User interfaces are an essential part of software development that should not be underestimated. The importance of the topic was pointed out by Dillon (2003). This is also important in video games, as pointed out by Kristiadi et al. (2017) who discovered that users love games because of their good user interface.

In addition to being easy to understand, the displayed content must not be overloaded and unnecessarily complicated. An application must be effective and easy to use. In addition, a user interface should also be consistent and well-integrated into the game. Good user interfaces also provide clear feedback for interactions. Therefore several design patterns were developed

by Hooper and Berkman (2011). They have conducted research and compiled 76 best practices covering everything from using screens, lights, and sensors to creating pages and presenting information. Every pattern comprises an explanation of the design issue and its resolution, as well as antipatterns, specifics about interactions and presentations, and variations.

Especially the development of user interfaces on smartphones is a challenge due to the small display area. This is particularly important because the designed game will also be ported to mobile devices in a later phase. If this is planned from the start, the user interface does not need to be changed. A popular way to efficiently present a lot of content is to use a combination of scrollable horizontal and vertical lists. This makes it possible to group similar content in a meaningful way to give the user access to the desired content as quickly as possible, like shown in Figure 2.2.

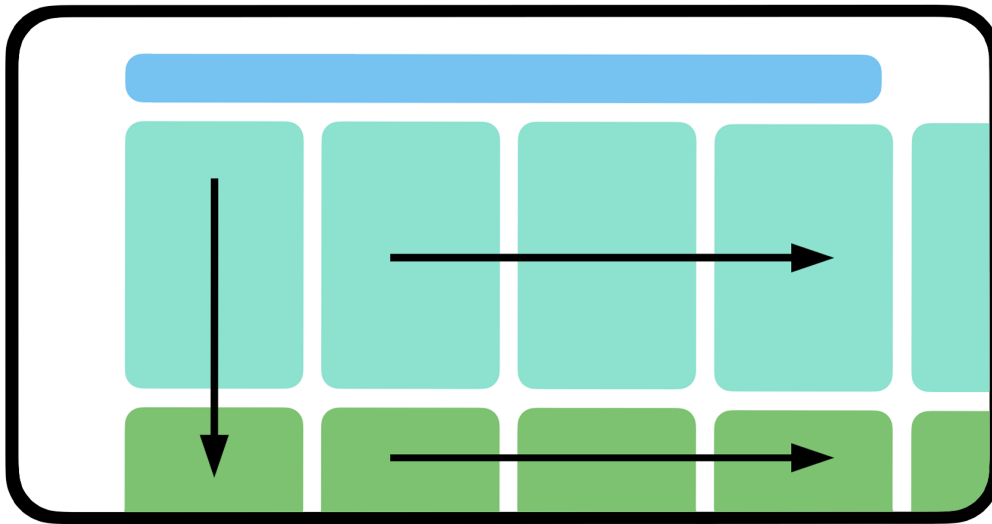


Figure 2.2: User interface design pattern for scrollable elements on mobile devices. The arrows show the scrolling direction while the color of the items shows the similar elements that are grouped.

Another important topic in user interface design is the avoidance of unnecessary text and words. On the one hand, these texts and words take up a lot of space, but it is also more difficult to maintain because it has to be translated into many languages. The use of icons provides a remedy here. However, caution is required when using icons, as they are not always understood and therefore do not achieve the desired effect if used incorrectly.



## 2.4 Racing Simulations and Games

Racing games are mainly divided into two categories. On the one hand, there are arcade games that focus on maximum fun and entertaining game experience, and, on the other hand, there are racing simulations that are intended to depict driving behavior that is as realistic as possible. However, for developers, there is a wide range of implementation options in between to find the right balance to appeal to a broader audience. The foundation for both is vehicle physics.

### 2.4.1 Vehicle Physics

Driving physics is a key element of racing game design and development, contributing significantly to the realism and immersive experience of game-play. Over time, the physical models became more sophisticated and closer to reality. This meant that motorsport fans could use the simulations to practice and improve their driving skills and lap times on race tracks. The game developers recognized early on, that very realistic driving behavior significantly increases the level of difficulty and therefore limits the fun of the game for casual players. Players and drivers have different preferences, therefore, when balancing, game developers always have to consider where they want to position their game on the spectrum between simple and realistic driving physics.

The complexity of the topic has led to a significant amount of literature and resources dedicated to its development. One of these works is from Bahaweres et al. (2014), who presented a method for simulating the dynamics of a four-wheeled vehicle. Their approach was based on a spring-mass-damper system combined with a fuzzy logic controller. This combination allowed for a more nuanced simulation of vehicle dynamics, contributing significantly to the field.

Another substantial contribution comes from Monster (2003), who described a comprehensive physics engine. This engine was characterized by the integration of a rigid body model, a tire model, a suspension model, and a collision detection system. The holistic approach taken in this work provided a robust framework for understanding and simulating vehicle dynamics.

Further expanding on this theme, Beckman (1998) delved into the physics model of a game. Their model employed a finite element method, a tire model, a suspension model, and an aerodynamics model. The inclusion of an aerodynamics model, in particular, added a layer of realism to the simulation, enhancing its applicability to real-world scenarios.

In a similar context, Hecker (2000) discussed the physics engine of a game. Their engine distinguished itself by using a mass-spring system, a tire model, a suspension model, and a collision response system. The use of a mass-spring system and a collision response system added depth to the simulation, making it a valuable resource for those studying vehicle dynamics.

Lastly, Bourg and Bywalec (2013) provided a guide to creating realistic and physics-based driving simulations. Their guide covered a wide array of topics, including vehicle physics modeling, steering, braking, traction, suspension, aerodynamics, and collision detection.

These works collectively contributed to a rich and diverse body of literature on vehicle dynamics and simulation. Each offered unique insights and methodologies, furthering our understanding and ability to accurately simulate vehicle dynamics. However, for developers who may not have the time or expertise to build their own driving physics from scratch, there are ready-to-use solutions available.

One notable example is Edy's Vehicle Physics Pro (Edy, 2022), a comprehensive solution available on the Unity Asset Store. This package offers a ready-made solution for setting up realistic driving models, saving developers considerable time and effort. These pre-built solutions are designed with a high degree of realism and can be easily integrated into a game, allowing developers to focus on other aspects of game design. An alternative would be Edy's Vehicles Physics from the same creator, which offers a simpler, game-like approach (Edy, 2011).

### 2.4.2 The Perception of Speed

While games and simulations are becoming more sophisticated, there are still some limitations:

- Firstly, the physical feedback in a simulation is not as nuanced as in real life. Despite advancements in haptic technology, it's challenging to replicate the exact feel of a car's movement, the vibration of the engine, or the resistance during gear shifts.
- Secondly, simulations are limited by the accuracy of their models. They rely on mathematical models to replicate real-world physics. These models, while sophisticated, can't account for every variable that might be encountered on a real-world road, such as changing weather conditions or unexpected obstacles.

- Thirdly, there's the hardware limitation. The quality of the simulation experience heavily depends on the user's hardware setup. Not everyone has access to high-end steering wheels, pedals, or VR headsets that can enhance the realism of the simulation.
- Lastly, there's a learning curve associated with simulations. They require a certain level of skill and understanding of driving mechanics, which might not appeal to casual gamers looking for a quick, easy-to-play racing experience.

Especially due to the lack of physical feedback, it is important to compensate for this with other effects. For example, MuYe (2021) criticized the lack of sensitivity to speed in racing games. He addressed the topic in a video and made some suggestions on how racing simulations could close the gap to arcade games and make the perception of speed in simulations more realistic.

The suggestions mainly related to visual effects and post-processing. Visual effects (VFX) are computer-generated effects that enhance the appearance and realism of video games. They include elements such as explosions, smoke, fire, water, and particles. VFX can also be used to create dynamic weather, lighting, and shadows that affect the gameplay and the aesthetics of games. Post-processing refers to the refinement and enhancement of outputs from a computer simulation or any digital process. The following effects can be added to improve the gaming experience:

### **Field of View (FoV)**

Field of view refers to the visible field of the game being visible on the screen at any given moment. In racing games, a wider FoV can give players a better sense of speed and spatial awareness, but it might also make the game more challenging as objects appear smaller.

With a larger field of view, the subjective perception of speed changes, as studied by Lidestam et al. (2019). This change in the field of view angle allows speeds to be displayed on the screen in a more immersive and action-packed manner.

The influence of the field of view on speed perception has also been investigated by Pretto et al. (2009). According to his findings, the field of view below 60° leads to an underestimation of speed.

### **Motion Blur**

This is a visual effect that blurs moving objects to simulate the high speed at which the player's car is moving. It enhances the perception of speed and direction, making the game feel more realistic.

"Motion blur is a fundamental cue in the perception of objects in motion. This phenomenon manifests as a visible trail along the trajectory of the object and is the result of the combination of relative motion and light integration taking place in film and electronic cameras." - Navarro et al. (2011)

Motion blur is especially important in racing games, where it creates a sense of speed and motion and may increase the overall excitement for the game.

### **Camera Shake and Distance**

Camera shake can be used to simulate the impact of collisions or the rumble of the car engine, adding to the immersion. If the camera is far away from the vehicle in 3rd person mode, this ensures a more intense perception of speed. This is especially true when the distance increases with increasing speed.

### **Sound**

These are used to enhance the realism and immersion. Wind noise and wooshing sounds increase with speed, giving auditory feedback to the player about their velocity. When a car passes by, there is a change in the frequency of a sound relative to the observer, which is called the Doppler Effect. It was first described by Doppler (1842).

Music and sound effects are essential elements for creating immersive and engaging experiences in video games. They can influence the atmosphere, the emotions, and the immersion of the player in the game world (Sexton, 2007; Zehnder & Lipscomb, 2006). They can also create tension, drive the plot, and give feedback to the players. The music in a game sets the tone and the mood and draws us into the game universe while sound effects increase the realism of the game.

### Narrow Streets

In racing game design, narrow streets can increase the difficulty level. They require more precise steering and can create tense overtaking maneuvers, enhancing the challenge and excitement of the game. This is also the reason why in many games the streets are unnaturally wide. It should be noted that race tracks have a fixed width and this only applies to racing games that take place on public roads.

These points could serve as optional features to help simulations better adapt to the subjective perception of the players to better represent real driving. Arcade games have the opposite problem, in that critics say the simplicity and lack of consequences take the excitement out of the games.

TimePlayer (2023) also pointed out that too wide roads and low traffic make games too easy and therefore boring. The reduced risk of driving off the road or colliding with other vehicles reduces the scare and adrenaline factor, which makes the games less engaging. However, there are different results in the literature as to how difficulty affects players' motivation. Lomas et al. (2017) said that it is advantageous to keep games easy but interesting to achieve the highest motivation. However, one must remember that this statement refers to educational games.

TimePlayer (2023) also highlighted that in *Forza Horizon 5*<sup>2</sup>, for example, the high-end vehicles can be earned very quickly at the beginning of the game. This led him to the feeling that he had achieved the goal very fast and he felt a lack of progress in the game after that. This phenomenon was also described by Koster (2013). He states, for example, that the human brain is goal-oriented and tends to choose the easiest path through the game. So, if the best vehicles are earned at the beginning, how much motivation is there to earn worse ones? The game could quickly lose interest. In summary, the key to an entertaining game is to find the balance between easy and difficult and to offer a good learning curve to pick up the players.

## 2.5 Related Games

In the course of this work, some games were tested and looked at to examine different mechanics like car racing, collecting, trading, and ranking. The most relevant games are listed here:

---

<sup>2</sup> Playground Games, 2021. <https://www.xbox.com/de-AT/games/store/forza-horizon-5/9nnx1vvr3knq>.

- *Need for Speed: Pro Street*<sup>3</sup>. A classic car racing game developed by EA Black Box and released for consoles and PCs in 2007. The game offers classic car races on closed public roads, structured in the form of events where racers compete against each other and compete in several stages for victory. There is also the option of equipping vehicles with randomly dropping upgrades and changing the vehicle setup.
- *Gran Turismo* 5<sup>4</sup>. Another classic racing game developed by Polyphony Digital and was released for PlayStation 3 in 2010. The game offers realistic racing experiences, over 1000 cars to collect, and a simulated marketplace to buy used cars for a cheaper price. The game also includes the ability to swap vehicles with other players.
- *Race Driver: GRID*<sup>5</sup>. Developed by Codemasters and published for consoles and desktop computers in 2008. Special features of the game are convincing damage models of the vehicles and a career built from a racer's point of view. This means the player starts the career as a race car driver including taking care of income and sponsorships. According to Hegevall (2022), it stands out as one of the most complete and entertaining arcade racing games of all time.
- *Need for Speed: Hot Pursuit*<sup>6</sup> is an arcade racing game developed by Criterion Games and was released for consoles and desktops. It's the 16th title in the series and offers a map with a large road network on which the player can carry out chases with the police or even as a police force.
- *Need for Speed: Undercover*<sup>7</sup> is a racing game for console and desktop and was released in 2008. Developed by EA Black Box, it offers an open-world racing principle with street racing elements and police chases. There are different event types such as highway battles on busy highways and sprint races.
- *Need for Speed: Shift*<sup>8</sup> is a follow-up title after Undercover but focuses on simulation rather than the arcade racing of previous titles. It was developed by Slightly Mad Studios and EA Bright Light and was released for desktops and consoles in 2009.

---

<sup>3</sup> EA Black Box, 2007.

<sup>4</sup> Polyphony Digital, 2010. <https://www.gran-turismo.com/de/products/gt5/>.

<sup>5</sup> Codemasters, 2008. <https://www.ea.com/games/grid>.

<sup>6</sup> Criterion Games, 2010. <https://www.ea.com/de-de/games/need-for-speed/need-for-speed-hot-pursuit-remastered>.

<sup>7</sup> EA Black Box, 2008.

<sup>8</sup> Slightly Mad Studios and EA Bright Light, 2009.

- *Assetto Corsa Competizione*<sup>9</sup> is a racing simulation game for desktops and consoles, developed by Kunos Simulazioni. The simulation focuses on GT3 and GT4 series races and offers licensed racing series like the Total 24 Hours of Spa and Endurance Cup. In addition to online multiplayer, it features an career mode, special events, and custom races.
- *Pokémon Go*<sup>10</sup> was developed by Niantic and released for mobile devices in 2016, Pokémon Go features hundreds of collectible Pokémon that can also be traded among players. The swap function is already a central mechanic in the classic Pokémon games.
- *League of Legends*<sup>11</sup> is a mulitplayer online battle game, was developed by Riot Games, and published for desktop computers in 2009. Collecting is also an important component in this game. In addition to the many heroes that can be collected and then mastered, there are also many skins and achievements to collect. In addition, the game is divided into so-called seasons in which the players can receive a rank.
- *SimCity: BuildIt*<sup>12</sup> is a mobile game, developed by TrackTwenty and released in 2014. The game, based on the classic city-building simulation SimCity, features mining and resource creation. These resources can be used to build the city or to trade with other players at a trading center.
- *Real Racing 3*<sup>13</sup> is a mobile racing simulation with arcade elements developed by Firemonkeys Studios and was released in 2013. The game offers a realistic racing experience with over 400 cars and 42 car manufacturers. In contrast to earlier games in the series, players must maintain and service their cars with in-game currency or by accepting idle times.
- *iRacing*<sup>14</sup> is an online-based racing simulation developed by iRacing.com Motorsport Simulations. It was released in 2008 and uses a subscription model. The game simulates car races on real tracks with rules based on real racing events. It provides a realistic driving experience and supports both racing wheels and gamepads. Vehicles and race tracks can be purchased individually through in-app purchases.

---

<sup>9</sup> Kunos Simulazioni, 2019. <https://assettocorsa.gg/assetto-corsa-competizione/>.

<sup>10</sup> Niantic, 2016. <https://pokemongolive.com>.

<sup>11</sup> Riot Games, 2009. <https://www.leagueoflegends.com/>.

<sup>12</sup> TrackTwenty, 2014. <https://www.ea.com/games/simcity/simcity-buildit>.

<sup>13</sup> Firemonkeys Studio, 2013. <https://www.ea.com/de-de/games/real-racing/real-racing-3>.

<sup>14</sup> iRacing Motorsport Simulations, LLC, 2008. <https://www.iracing.com>.

- *Forza Motorsport 8*<sup>15</sup> is a modern racing game that was created by Turn 10 Studios and released for Xbox and PC in 2023. The game offers a realistic racing experience with advanced graphics, over 500 cars, and a wide range of customization options. It also features different weather conditions and times of day, which affect the driving physics. The vehicles have a mastery system in which more and more vehicle upgrade features are unlocked the longer the car is driven.
- *Project Cars 3*<sup>16</sup> is a modern racing game that was created by Slightly Mad Studios and released for consoles and PC in 2020. The game offers a realistic racing experience with over 200 cars and 120 racetracks. It also features dynamic season- and weather conditions, and a career mode.
- *Automobilista 2*<sup>17</sup> is a racing simulation developed by Reiza Studios and released for PC. It has a large selection of vehicles and racetracks, primarily featuring Brazilian content. The simulator supports Virtual Reality headsets, multi-screen, and full-motion racing simulator setups.
- *Forza Horizon 5*<sup>18</sup> was developed by Playground Games and released for Xbox and PC in 2021. The game offers a realistic racing experience with a large amount of cars and tracks. It also features dynamic weather conditions and times of day, affecting the driving physics. In *Forza Horizon*, swapping between two players is not implemented. However, there is a so-called Action House where vehicles can be offered for sale. Although one-to-one swapping is not implemented, players willing to swap organize themselves in forums and provide instructions on how swapping can still be carried out in a detour.
- *EVE Online*<sup>19</sup> is a free MMORPG sci-fi strategy game where the player can choose their own paths and engage in combat, exploration, industry, and much more in a vast and dynamic open world. It is a space game with a large community and a long history of 20 years.
- *Grand Theft Auto 5*<sup>20</sup> is an action-adventure game, featuring three protagonists involved in heist sequences. The game allows driving around with different vehicles in an open world, including city environments.

---

<sup>15</sup> Turn 10 Studios, 2023. <https://www.xbox.com/de-AT/games/forza-motorsport>.

<sup>16</sup> Slightly Mad Studios, 2020. [<https://de.bandainamcoent.eu/project-cars/project-cars-3>].

<sup>17</sup> Reiza Studios, 2020. <https://www.game-automobilista2.com>.

<sup>18</sup> Playground Games, 2021. <https://www.xbox.com/de-AT/games/store/forza-horizon-5/9nnx1vvr3knq>.

<sup>19</sup> CCP Games, 2003. <https://www.eveonline.com/>.

<sup>20</sup> Rockstar North, 2013. <https://www.rockstargames.com/de/gta-v>.



The realistic implementation of cities and their streets ensures a more realistic driving experience than racing games that rely on cities that have been specially adapted for car racing games.

### 2.6 Summary

This chapter first surveyed literature about collecting in general and then delved into digital collections. The importance and special status of collecting in video games were highlighted.

Furthermore, the topic of swapping in video games was discussed, and what advantages and disadvantages swapping brings in the context of collecting. It became clear that the success of an exchange function depends heavily on its implementation. Therefore, ways for possible implementations were shown.

General aspects of game design were then discussed. These were game mechanics, user interfaces, and the choice of implementation tools, which are all important factors in game development and therefore provide an important basis for successfully implementing an exchange function.

The topic then moved on to racing games and the differentiation between car racing games and simulations. Clear differences in structure and target group were shown, but also points of connection between the game types. It was found that balancing between the two alignments is an important topic and is highly debated. Problems of both types of games were identified and various approaches to solving them were shown. The chapter was finally rounded off with a list of relevant games that are relevant in the context of the topics discussed.

## **3 Design & Conceptual Model**

To answer the research goals and based on the theoretical background, the design and requirements of the implementation were defined. First, the starting point and objectives were discussed, followed by the requirements which were divided into functional and non-functional requirements. Finally, the architecture of the app and the server, as well as the planned game mechanics, were discussed.

### **3.1 Starting Point and Motivation**

The goal was to create a racing simulation where players can race, collect, and trade vehicles. Since the research focused on collecting and trading aspects of the game, an already working car physics script was used. Aside from that, the project was built from the ground up. Self-developed 3D models such as cars, environmental elements, and user interface elements were used to give the app a unique look. The game was organized in a modular structure. This meant clean programming and division of the code into appropriate functions but also the organization of the code files and other game assets in folders sorted by modules. This should enable easy adaptation of features and later changes and extensions. In addition, the development should take into account the display on smartphones, tablets, and virtual reality devices in order to allow easy porting to those platforms.

### **3.2 User Target Group**

The target group of players were casual gamers who enjoy racing games, are interested in vehicle design, and like to collect. In addition, to expand the target market, demanding enthusiasts of racing games and car collectors were addressed, by adding advanced realism to the game and pushing it in the direction of realistic racing simulations. Moreover, these enthusiasts often have a real-life interest in collecting cars in any form, like real cars or

scale car models. They take pride in acquiring and showcasing a wide array of vehicles, each with its unique design and specifications. For this target group, the application may serve as a virtual extension of their tangible passion.

### 3.3 Requirement Analysis

At the beginning of the development of a software project, the requirements need to be defined. These are in general divided into functional and non-functional requirements. The former defines the specific functions while the latter defines general requirements.

#### 3.3.1 Functional Requirements

The functional requirements were divided into two main groups. On the one hand the racing features and on the other hand the collecting and trading features.

- Requirements for the racing features
  - Players should be able to participate in 3D race events.
  - Players should be able to get scores after events which they can compare with other players in leaderboards.
  - Players should be able to earn achievements for completing and winning races.
- Requirements for the collecting and trading features
  - Players should be able to collect cars and earn badges for them.
  - Players should be able to gather new cars through a trading mechanism.
  - Players should be able to earn achievements for completing and winning races.

#### 3.3.2 Non-Functional Requirements

In addition to the functional requirements, non-functional requirements were formulated too.

- Availability

- The racing simulation should be downloadable from the internet and the servers for trading and leaderboards should be available at all times.
- Scalability
  - The software should work independently of the number of users. Trading should be possible even when no other players are online and also if there are many users online.
- Compatibility
  - All functions should be platform-independent. This means a responsive user interface, different input methods, and platform-independent servers for leaderboards.
- Usability
  - The game and all of its functions should be easy to use. Trading and collecting should be as effortless as possible.
- Maintainability
  - The software should be clear and easy to maintain to simplify further development. Maintenance should also be possible for other developers.
- Reliability
  - The servers should be as resilient as possible. This means that the servers are accessible around the clock, with a minimum of downtime.
- Performance
  - The performance of the software should always be sufficient. Performance problems like lags on desktop devices or increased heat development of mobile devices should be avoided.
- Security
  - The servers and the communication between the end device and the server should be secure to prevent data from being falsified.
- Integrability
  - The code developed for trading should be able to be easily integrated into other games.
- Reusability/Configurability
  - The game should be developed in such a way that it can be used in the future and can serve as a basis for further projects.
- Extendability

- The software should be programmed in such a way that it can be expanded to include additional features in the future.

### 3.4 Conceptual Architecture

This section is about the architecture of the game, starting with the structure of the client app, followed by the server side, and finally, the game mechanics that should be integrated into the game.

#### 3.4.1 Storyboard

Based on the objectives and the target group, a storyboard was developed to define the basic structure and main functions of the game, seen in Figure 3.1. When developing the concept, both the implementation in virtual reality and the implementation as a conventional computer game were taken into account. At the beginning of the game, there is a trailer to get the player in the mood for the game. However, the central element of the game is the garage, in which the player's vehicle is parked, which also functions as the main menu. From there the player can look at the car in possession, view the collection, enter the marketplace, and make settings. If the player wants to start a race, the scene switches into the cockpit of the vehicle and from there the player can choose whether to drive a street race or a race on the track. After each race, the player returns to the main menu in the garage. It should be noted that the races on the racetrack were only taken into account in the concept, but were not implemented in order not to complicate the test process and to keep the focus on collecting and swapping.

#### 3.4.2 Client Architecture

The first step after creating the storyboard was to think about how the game could be implemented most easily so that all requirements could be met. The core element of every racing game is the driving script. After extensive tests of different driving scripts, ranging from the basic driving scripts provided by the developers of the game engines Unity (Unity Technologies, 2021) and Unreal (Epic Games, 2021), to free ones offered at the Unity and Unreal Asset Store, the decision for the most fitting solution fell on the Edy's Vehicle Physics Pro script (Edy, 2022). Combined with other factors like a subjectively easier and faster implementation the Unity game engine was chosen to realize the project. In addition to being easy to use, the engine

### 3 Design & Conceptual Model

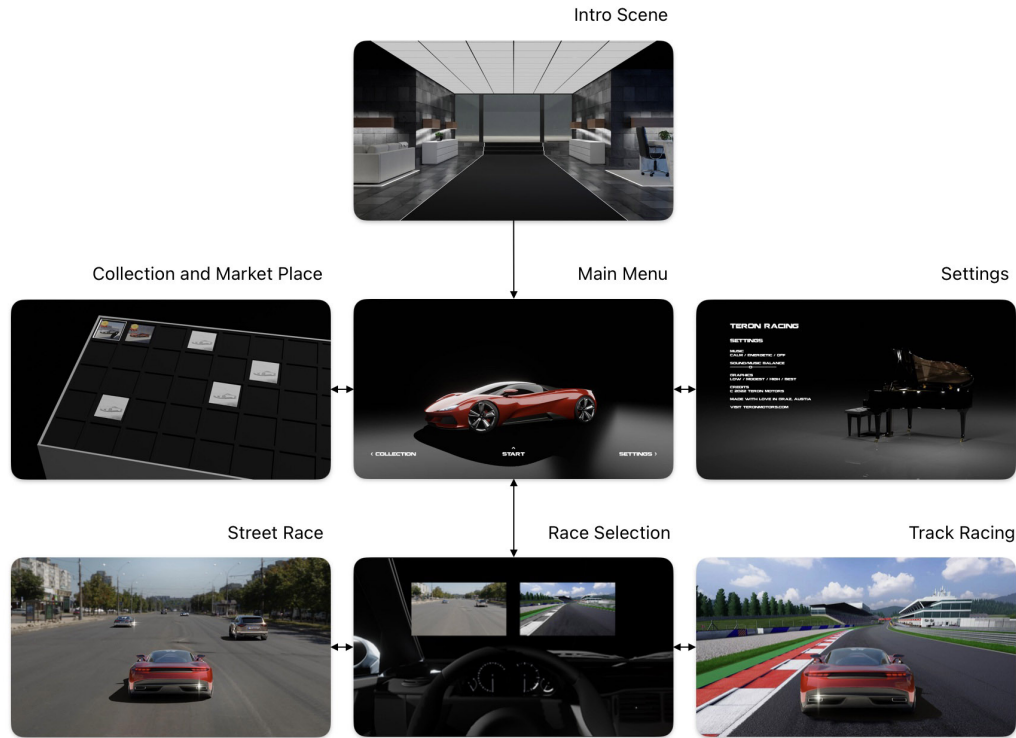


Figure 3.1: Storyboard of the game as a virtual reality game with street and race track mode.

also should offer an easy way to publish the game on different platforms, like the targeted desktop and mobile platforms.

During the engine evaluation, a graphics comparison was created and checked which quality could be achieved in a short time and without specific knowledge. The results are shown in Figure 3.2 for Unity and Figure 3.3 for Unreal.

The following Figure 3.4 shows more information about the architecture of the game. The components are broken down as follows: Unity scenes (turquoise), UI screens (orange), menus (yellow), databases (light blue), recorded study data (dark blue), the test group-dependent UI (red), and web interfaces (violet). The arrows show the possible navigation within the game or the data transfer. A high number of users is required for a functioning trading function and this cannot be achieved in the short time of the study period. Therefore, the design of this feature for this study is limited to the fact that the trading only takes place with the computer.



Figure 3.2: Quick test of the cars look in Unity using the High Definition Render Pipeline and the demo project.



Figure 3.3: Quick test of the cars look in Unreal in the car at the beach scene.

### 3 Design & Conceptual Model

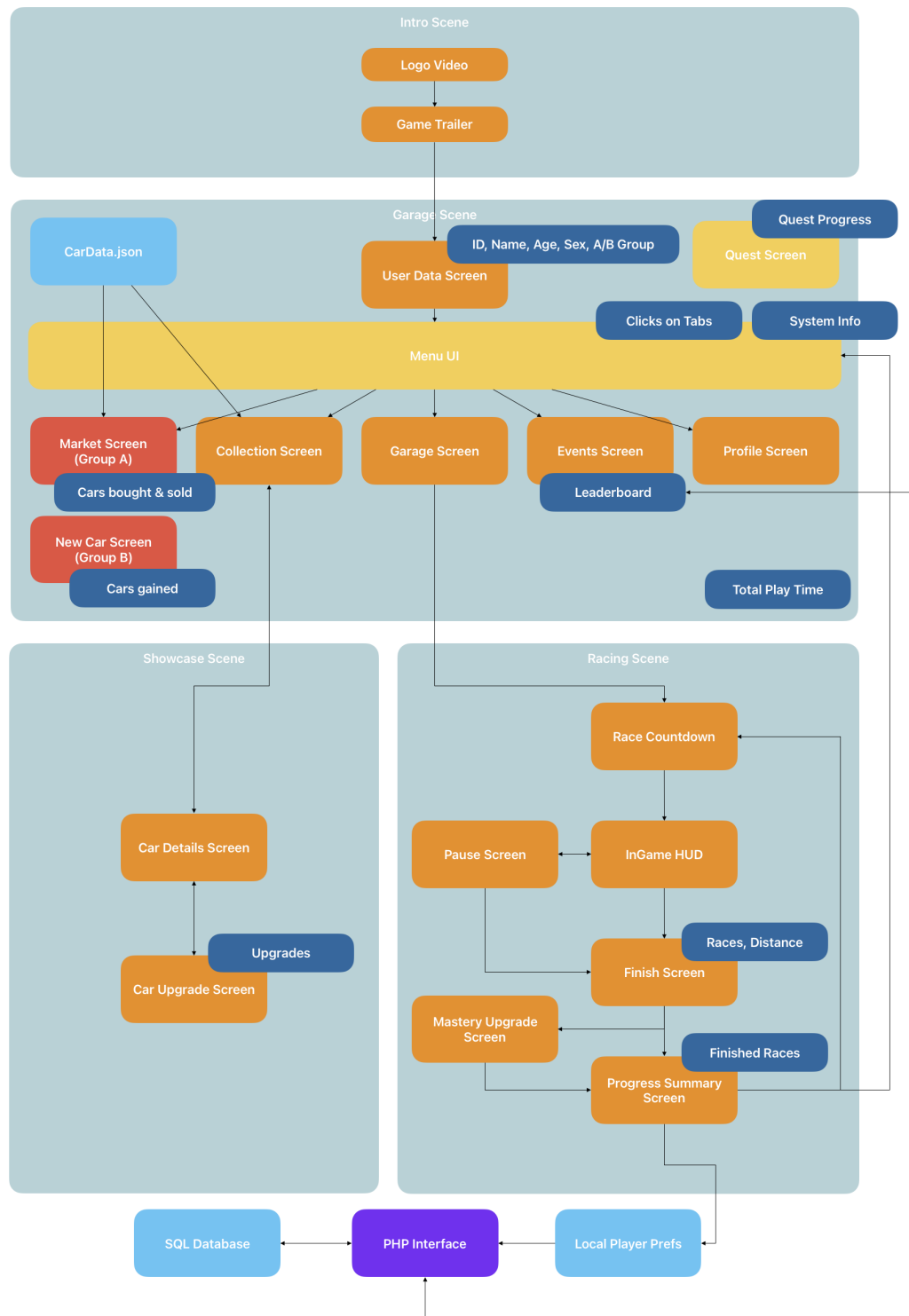


Figure 3.4: Overview of all components of the game in the client application and database, grouped in Unity scenes and further broken up into UI screens, collected data, databases, and web interfaces.



### 3.4.3 Server Architecture

Even if the trading function within this study was not implemented via the server, a server connection was still designed to implement a ranking system and to record study data and upload it for evaluation to a database on the server. To realize the rankings, all high scores were uploaded into the database. This server acts as a central repository for all important game data, ensuring that every player's achievements are recorded and stored securely. The data was provided with an ID so that it can be converted into user accounts in the future through extensions so that players can restore their scores. The server uses MySQL (Oracle Corporation, 2023), a popular open-source relational database management system, to manage and organize this data. MySQL is known for its speed and reliability, making it an excellent choice for handling high volumes of data.

To create the communication between the server and the game engine, a PHP interface was used. PHP is a versatile programming language that is very useful for server-side web development and works perfectly with HTML. This interface acts as a bridge between Unity and the server, allowing for seamless data exchange. When a player achieves a new high score in the game, this information is sent via this interface to the MySQL database on the server.

This setup not only allows for real-time updating of rankings but also provides a robust and scalable solution for managing game data. As more players join the game and new high scores are set, the system can easily accommodate this growth, ensuring that every player's accomplishments are recognized in the rankings.

## 3.5 Game Mechanics

The game concept and requirements demanded the development of various game mechanics, which are listed in Table 3.1. These mechanics are integral to the gameplay experience, influencing how players interact with the game environment and achieve their objectives. The table serves as a guide, outlining each mechanic and its role within the game's structure.

Based on Fabricatore's cycles of play experience (Fabricatore, 2007), mentioned in Chapter 2, the game mechanics can be described as in the following Table 3.2.

Table 3.1: Intended Game Mechanics for the game including their description.

Mechanic	Description
Racing	Driving the car on a street while trying to reach the highest possible speed without colliding with traffic or the surrounding area.
Collecting	Collecting cars, achievements, medals, trophies, and car parts.
Trading	Buying and selling cars and parts on the marketplace.
Leaderboard	A high score list to compare the best players.
Modifications	Modify the car with new parts to improve the performance.
Quests	Completing different quests to learn about the game features.

Table 3.2: Decision-making based on the play experience cycle.

Gathered Information	Decisions
Traffic situation change	Adjust speed, choose lane or take the risk to collide with other cars in traffic and fail the top speed attempt?
New top speed reached	Finish the run or continue?
Money reward	Spend money on new car or car parts?
Car part reward	Should it be equipped, discarded, or sold?
Offered car and car parts	What should be bought from the offer?
Impact of modifications	Which modification improves performance?
Car/Parts market prices	Sell unneeded cars and parts to make a profit?

### 3.5.1 Level Design

The level design is based on real racing events and thus offers individual races that are combined as stages in events. All events are evaluated and, over a season, summarized in a championship, resulting in an evaluation that is recorded in a ranking list (Figure 3.5). A requirement for participation in the races is a vehicle and depending on the event upgrades on it. With the prize money from the events, research can be carried out to upgrade the vehicles. The research is implemented as a random item drop, where a purchase with in-game currency, different improvements can be gained for the vehicle. In addition to the in-game currency reward, players can also receive trophies and increase the rank of their vehicle described in the next paragraph. The trophies are awarded for achieving the main goal of the event. In this case, that means reaching the target speed.

To better reflect the experience and skill of the players and to motivate them to drive each car, a vehicle-level system was introduced. Different tasks have to be completed, such as reaching the vehicle-specific maximum speed, completing a certain number of kilometers, or winning races. For each goal achieved, the player receives badges and the level of the vehicle is increased. By increasing the driver level, the player receives additional badges to increase the prestige of the vehicle.



Figure 3.5: Overview of the requirements for playing the levels, the level division, and the rewards for completing levels.

There are different parameters to complete events. These are:

- **Precision:** The most important criterion is precision. The player must not collide with other vehicles or objects. In other event types with scoring, there may also be deductions for standing times and reversing the vehicle.
- **Top Speed:** If the first criterion is met, the maximum speed achieved during the race counts. In classic races on the race track, the lap time can also be measured.

- Time: Depending on the event, a minimum time or a maximum time on the road can be set as a limit. Alternatively, in races on the race track, the fastest lap is counted.

### 3.5.2 Racing

As already outlined in the storyboard, two racing modes were planned. On the one hand, classic races on the racetrack, and on the other hand, high-speed races on motorways. However, since the study is primarily about swapping and collecting, only the high-speed races on the street were implemented here for the study. Here, the gaming experience is based on attempts to set top-speed records. The player starts next to the motorway and accelerates the car before getting into traffic. The goal is to reach the highest possible speed without endangering other road users or causing an accident. To ensure safe driving, a record attempt is only counted if no collision is caused, the motorway is left again after the record attempt, and the vehicle is safely brought to a standstill. To further increase the level of difficulty, there will be traffic on the road. Police vehicles will also be present on the Autobahn, which can intervene if the driving style is dangerous despite the unlimited permitted maximum speed. For example, so-called tailgating, in which drivers drive too close to cars in front of them or overtake in the right lane. It should also be mentioned that risky high-speed driving maneuvers should only take place on closed roads. On public roads, careful driving is necessary and no unnecessary risks should be taken.

### 3.5.3 Collecting

The collection feature was implemented on several levels. The basis is collecting the cars themselves. On top, medals and trophies can be collected for every owned car. The players can collect experience medals for the total distance traveled, as well as loyalty medals for the time they have already owned the car. The trophies can be earned by achieving exceptional performance in events. As a third level of collecting, there are cross-vehicle achievements to reward general achievements in the game.

### 3.5.4 Achievements

The achievements were structured in such a way that, depending on the progress, there is a bronze, silver, or gold medal for reaching the goals. A detailed list of all planned achievements can be found in the following list.

- Expert Driver: This medal honors your driving experience. Drive many kilometers or collect experience points to improve this medal. (10, 1.000, 10.000 km)
- Event Traveller: Take part in races to improve this medal. (10, 100, 1.000 races)
- Quest Master: Absolve quests to earn this medal. (2, 4, 7 quests)
- Enthusiast: A medal to honor your total playtime. (1, 1.000, 10.000 hours)
- Ultimate Speed: Reach extreme speeds without crashing the car to earn this medal. (200, 300, 400 km/h)
- Tuner: Install car parts to upgrade your car to earn this medal. (10, 100, 1.000 upgrades)
- Collector: Collect cars to earn this medal. (2, 10, 100 cars)
- Car Flipper: Sell cars to earn this medal. (2, 10, 50 vehicles)
- Trophy Hunter: This medal honors your enthusiasm to collect trophies. (1, 10, 100 trophies)

#### 3.5.5 Trading

The third core feature and at the same time the most complex one provides a trading system between the players to enable swapping of their cars. Since the vehicles have different rarity values, the vehicles must be swapped one for one but can be offered for sale on a marketplace for an in-game currency. To keep the trading active, the number of vehicles a player can own is limited. So at some point, a player has to throw vehicles back onto the market to buy new ones. A market maker buys back the vehicles and offers other vehicles for sale.

#### 3.5.6 Modifications

To increase the complexity of the game, it was also planned that vehicle parts could be bought at a later date. These then make it possible to modify and adjust the vehicle properties to adapt the vehicle's handling to one's own needs and driving style. The players receive the vehicle parts randomly, with different rarities, to ensure that the game lasts over a longer period. This game mechanic, with random car parts instead of parts that can be bought, is more modeled after racing where money has to be put into the research and development of parts, and finished parts cannot simply be bought in a shop. This is also ideal as a money sink where excess in-game currency can be used up.

### 3.5.7 In-Game Currency

For the prototype implemented in the study, the monetary system remained largely deactivated and each player received a fixed capital to enable the trading of cars. In future versions, the players will be able to earn the in-game currency by racing and can invest it in upgrading the cars.

### 3.5.8 Leaderboards

The leaderboards should be designed in such a way that they work platform-independent. To be completely flexible and independent from third-party solutions, a separate database was set up and hosted on a server. The scores of the players are automatically uploaded there and can be displayed in leaderboards. In the future, cloud backups and user accounts can also be implemented with this database.

## 3.6 Summary

In this chapter, the starting point and motivation were summarized, and the goals for implementation were set. Based on this, the target group was defined and a requirements analysis with functional and non-functional requirements was created.

After the requirements were defined, a storyboard was developed to paint a rough picture, vision, and guidelines for the app. When it became clear that the chosen values were based on a realistic, already existing driving script and the visual appearance of the app, a suitable engine was sought for implementation. The decision was made to use the Unity game engine and to use an already existing driving script for the client application.

Finally, the game mechanics that should be used in the game were then determined. A level system, car racing, collecting, trading, achievements, upgrades as modifications, an in-game currency, and a leaderboard were selected.

## 4 Implementation

This chapter unfolds the journey from the initial design phase to the final game, presenting the results of the implementation and providing insights into the process. It illustrates the transformation of conceptual ideas into tangible outcomes, culminating in the final game.

### 4.1 Used Software

The Unity Engine (Unity Technologies, 2021) version 2021.3 LTS (Long Term Support) and subsequently version 2022.3 LTS was used as the base for developing the game. Unity currently offers three different render pipelines. Before starting the project, it's crucial to consider which of these is most suitable. The Standard Render Pipeline (SRP) was chosen for this project to maintain the flexibility of transitioning to the Universal Render Pipeline (URP) or the High Definition Render Pipeline (HDRP) in the future. This makes a later upgrade less complicated because a switch from URP to HDRP or vice versa is significantly more complex. Furthermore, the SRP is still the most widely used, and therefore most shaders are available for it.

The open-source software Blender (Blender Foundation, 2021) was used to model the vehicle and the environment. To use the 3D models from Blender, new materials had to be developed or used because the Blender materials were not compatible with Unity. Otherwise, importing from Blender to Unity via the FBX format worked very well.

## 4.2 Vehicles

In most racing games, the central elements are undoubtedly the vehicles. They do not only define the gameplay but also significantly influence the player's experience. To create variation in the fleet without having to develop a huge amount of vehicle models and to keep the memory consumption of the game low, only a single car was chosen to be integrated into the game. This car is presented in unique, unchangeable color coatings, shown in Figure 4.1. This creates a clear differentiation between the models without relying on using different car models. The colors are stored as HEX codes and variables for the surface (for matte and glossy) in a JSON file along with all other vehicle data and information (Figure 4.5) which allows a huge amount of different variations from a single car.



Figure 4.1: A selection of the chosen colors to offer differentiation and variation for the vehicles.



### 4.2.1 Vehicle Design

The vehicle used for this implementation is called Teron Kanaani and was modeled by the author using Blender. The name of the car is derived from Terra and the Kanaani cat breed, which, matching the sports car, is particularly characterized by its slim but strong physique. The most striking features of the vehicle are the unmistakable silhouette of a mid-engine sports car, the gullwing doors, and the large glass dome that encloses the driver's cabin based on the carbon monocoque. The logo of the fictional vehicle manufacturer is a crown, inspired by the King Crown that is awarded in Need For Speed: Pro Street when the player masters all racing categories.

The task of developing such a car required a considerable investment of time and effort to ensure the model's accuracy and suitability for the game environment. Throughout the bodywork modeling process, the author received invaluable support from Richard Kastner. His expertise and guidance were instrumental in refining the model's bodywork and rims.

Once the modeling of the bodywork of the Teron Kanaani was completed, it was then adapted for use within the game. This involved expanding and rebuilding certain aspects of the model, like an interior, several car details, and cut-out bodywork to ensure extended functionality like opening doors and the rear spoiler. The result was a well-integrated, game-ready vehicle model that significantly enhanced the visual gaming experience, shown in Figure 4.2.



Figure 4.2: The Teron Kanaani sports car with opened wing doors. Modeled in Blender and rendered in Blender's render engine Eevee.

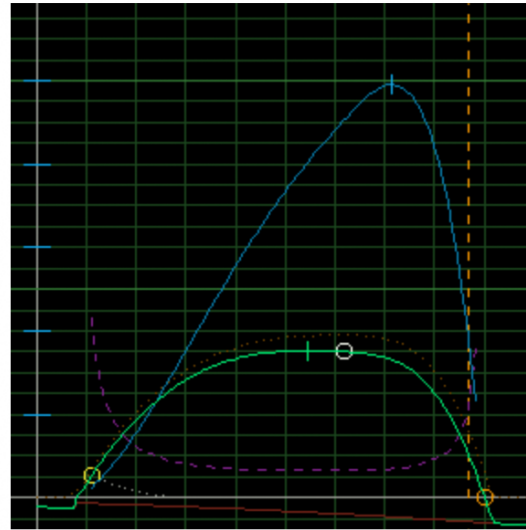
In addition to the exterior, an interior was designed and modeled to allow playing the game not only in the third-person but also from the driver's perspective (Figure 4.3). This relies on a combination of classic display elements and modern, minimalistic touchscreens. The interior also features a classic sports steering wheel with a 12 o'clock marking, a gear shift lever, and a clutch pedal. The interior allows a switch to a new camera located in the cockpit, which enables a particularly immersive gaming experience, especially in virtual reality mode.



Figure 4.3: The interior of the Kanaani, developed in Blender and rendered in Blender's render engine Cycles.

### 4.2.2 Vehicle Physics

The Vehicle Physics Pro script was used as the foundation for the driving script (Edy, 2022). The free community version of the script, which was used for the implementation of realistic driving behavior, already offered all the features necessary for the project. However, it was necessary to adjust vehicle parameters to obtain the desired performance data. For example, one can approach the desired driving performance by altering the maximum torque and adjusting the performance curves (Figure 4.4). Other values such as weight, air resistance, behavior of the brakes, tires, drive, steering, clutch, and gearshift had to be adjusted according to the desired performance to effectively transfer the engine power to the road.



Hover here to legend	
Max Power	495.3 Kw (664.2 HP) @ 7137 rpm
Max Torque	705.8 Nm @ 5445 rpm
Limit Rpm	9162 rpm
Idle Rpm	1100 rpm
Stall Rpm	756 rpm
Friction at stall	-50.1 Nm
Friction at idle	-27.6 Nm
Friction at limit	-122.7 Nm
Specific fuel consumption (BSFC):	
	129.9 g/kWh @ 5445 rpm

Figure 4.4: The screenshot from the configuration of Edy's Vehicle Physics Pro shows the performance curve of the configured sports car.

### 4.2.3 Vehicle Data

All data about the vehicles were stored in the JSON file, which allows for easier modifications. This was primarily used for storing the different colors. In the future, however, as the number of supported vehicles increases, the integration can be further deepened so that extensive engine data is saved in the JSON file and the vehicle script is configured automatically. The following code (Figure 4.5) shows the implementation of the first car in the JSON file. To simplify the data input and to reduce proneness to errors, the colors were specified in the file in the form of HEX codes and not in the RGB format required by Unity. For this purpose, a simple HEX-RGB converter was written that converts the data accordingly.

## 4 Implementation

```
{
  "cars": [
    {
      "model": "Teron Kanaani",
      "type": "Sports Car",
      "details": "The Kanaani is a puristic super sports car with innovative technology and the latest vehicle from Teron Motors. Production is strictly limited and each model has a unique livery.",
      "maxpower": "664", "maxtorque": "705", "weight": "1600", "drivetrain": "RWD", "gearbox": "manual",
      "units": [
        {"hex": "F4F3EF", "smoothness": "0.94", "metallic": "0.0", "color": "Arctic White"},
        {"hex": "610000", "smoothness": "0.94", "metallic": "0.0", "color": "Cherry Red"},
        {"hex": "61FF00", "smoothness": "0.94", "metallic": "0.0", "color": "Poison Green"},
        {"hex": "FF8400", "smoothness": "0.94", "metallic": "0.0", "color": "Ocre"},
        {"hex": "5FC8C1", "smoothness": "0.94", "metallic": "0.0", "color": "Turquoise"},
        {"hex": "5D1E1F", "smoothness": "0.94", "metallic": "0.0", "color": "Wine"},
        {"hex": "FFC0EE", "smoothness": "0.94", "metallic": "0.0", "color": "Bubblegum Pink"},
        {"hex": "003FE0", "smoothness": "0.94", "metallic": "0.0", "color": "Blue"},
        {"hex": "FF4526", "smoothness": "0.94", "metallic": "0.0", "color": "Orange Red"},
        {"hex": "CEEAFF", "smoothness": "0.94", "metallic": "0.0", "color": "Frost White"},
        {"hex": "C95300", "smoothness": "0.94", "metallic": "0.0", "color": "Orange"},
        {"hex": "909090", "smoothness": "0.94", "metallic": "0.0", "color": "Grey"},
        {"hex": "002945", "smoothness": "0.94", "metallic": "0.0", "color": "Navy Blue"},
        {"hex": "958276", "smoothness": "0.94", "metallic": "0.0", "color": "Pebble Grey"},
        {"hex": "59337E", "smoothness": "0.94", "metallic": "0.0", "color": "Violet"},
        {"hex": "FFD91B", "smoothness": "0.94", "metallic": "0.0", "color": "Sun Yellow"},
        {"hex": "0E2A59", "smoothness": "0.94", "metallic": "0.0", "color": "Gentian Blue"},
        {"hex": "004160", "smoothness": "0.94", "metallic": "0.0", "color": "Green Blue"},
        {"hex": "2E353C", "smoothness": "0.94", "metallic": "0.0", "color": "Denim Grey"},
        {"hex": "A60000", "smoothness": "0.94", "metallic": "0.0", "color": "Deep Red"},
        {"hex": "01672B", "smoothness": "0.94", "metallic": "0.0", "color": "Styria Green"},
        {"hex": "000000", "smoothness": "0.94", "metallic": "0.0", "color": "Black"},
        {"hex": "925A3E", "smoothness": "0.94", "metallic": "1.0", "color": "Bronze"},
        {"hex": "EEEEEE", "smoothness": "0.94", "metallic": "1.0", "color": "Silver"},
        {"hex": "EAC76B", "smoothness": "0.94", "metallic": "1.0", "color": "Gold"}
      ]
    },
    {
      "model": "Teron Kanaani R",
      "type": "Sports Car",
      "details": "tba",
      "maxpower": "700", "maxtorque": "750", "weight": "1600", "drivetrain": "RWD", "gearbox": "manual",
      "units": [
        {"hex": "F4F3EF", "smoothness": "0.94", "metallic": "0.0", "color": "Arctic White"},
        {"hex": "4169E1", "smoothness": "0.94", "metallic": "0.0", "color": "Royal Blue"},
        {"hex": "0A0A0A", "smoothness": "0.94", "metallic": "0.0", "color": "Jet Black"},
        {"hex": "013220", "smoothness": "0.94", "metallic": "0.0", "color": "Dark Green"},
        {"hex": "151515", "smoothness": "0.50", "metallic": "0.0", "color": "Mat Black"}
      ]
    }
  ]
}
```

Figure 4.5: Content of the cars.json file, showing all important variables of the vehicles.

### 4.3 Racing Environment

The core element of the environment is the motorway, shown in Figure 4.6. A modular structure was chosen here so that different landscapes, road borders, and decorations can be automatically combined. Guardrails, guide posts, concrete road barriers, and noise protection walls were modeled as boundaries. Other decoration elements including forests, houses, and tunnels are shown in Figure 4.7.

#### 4.3.1 Environment Design

The goal was to represent the Central European, especially German, street scene, as only there on the highway unlimited speeds are allowed to be driven in certain areas. The road consists of three lanes and a breakdown lane for each direction. The directions of travel are usually separated by guardrails. However, there are also interruptions to offer more possibilities for players.



Figure 4.6: Autobahn as an environment for the traffic simulation, modeled and rendered in Blender.

#### 4.3.2 Procedural Environment Generation

The environment was designed in such a way that it is automatically generated and also changes in appearance. It is thus possible to create an endless





Figure 4.7: Autobahn with varying decoration models, set up in Unity.

highway without having to model endless variations or having to load an entire highway into the device memory. The whole landscape was implemented by dividing it into blocks and then lining them up in a chain. The algorithm is based on the Road Generator script by Sobaihi (2017) and was further developed to load different segments and to enable the concatenation of different sequences of elements as shown in Figure 4.8 as an example. The segments can spawn in a specific order. This is required for transitions between different amounts of lanes and the implementation is shown in Figure 4.9 and 4.10.

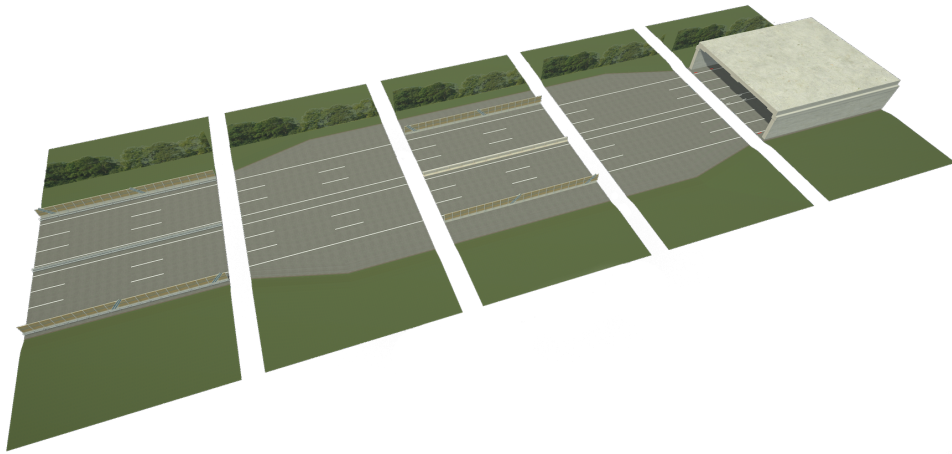


Figure 4.8: A selection of sections to generate the Autobahn. From left to right: noise barriers on narrow lanes, lane widening, lane with concrete barriers and emergency lanes, lane narrowing and tunnel.

## 4 Implementation

---

```
private void InitializeChunksList()
{
    _chunks = new Queue<Transform>();
    for (int i = 0; i < numberOfChunks; i++)
    {
        GameObject chunk = Instantiate<GameObject>(roadChunk);
        chunk.transform.position = NextChunkPosition();
        if (i != 0)
            chunk.SetActive(false);
        _chunks.Enqueue(chunk.transform);
    }
}

private void SweepPreviousChunk()
{
    Transform chunk = _chunks.Dequeue();
    chunk.gameObject.SetActive(false);
    chunk.position = NextChunkPosition();
    _chunks.Enqueue(chunk);
}

private Vector3 NextChunkPosition()
{
    float position = _currentChunkPosition - WorldOffset;
    _currentChunkPosition += (int)chunkLength;
    return new Vector3(0, 0, position);
}

private Transform GetCurrentChunk()
{
    Transform currentChunkObj = null;
    foreach (Transform c in _chunks)
    {
        if (Vector3.Distance(CalcOffset(player.position), c.position) <= (chunkLength / 2))
        {
            currentChunkObj = c;
            break;
        }
    }
    return currentChunkObj;
}

private int GetIndexOfCurrentChunk()
{
    int index = -1;
    for (int i = 0; i < _chunks.Count; i++)
    {
        if ((_chunks.ToArray()[i]).Equals(_currentChunk))
        {
            index = i;
            break;
        }
    }
    return index;
}

private Vector3 CalcOffset(Vector3 position)
{
    return new Vector3(0, 0, position.z - WorldOffset);
}
```

Figure 4.9: The main functions of the Road Generator script show the initialization of new sections and removal of old sections by Sobaihi (2017).



## 4 Implementation

---

```
private void FixedUpdate () {

    if (!player) return;

    _currentChunk = GetCurrentChunk();
    _indexOfCurrentChunk = GetIndexOfCurrentChunk();

    for (int i = _indexOfCurrentChunk; i < (_indexOfCurrentChunk+drawingAmount); i++)
    {
        i = Mathf.Clamp(i, 0, _chunks.Count-1);
        GameObject chunkGo = (_chunks.ToArray()[i]).gameObject;

        if (!chunkGo.activeInHierarchy) {
            chunkGo.SetActive(true);

            // Decoration
            for (int x = 1; x <= Options-1; x++) {
                chunkGo.transform.GetChild(x).gameObject.SetActive(false);
            }

            if (_decoChange == 0) {
                _decoSet = Random.Range(1, Options);
                _decoChange = DecoChangeFrequency;

                //Debug.Log("Street Style: " + _streetStyle + ", Deco Set: " + _decoSet);
                if(_entryExit > 5)
                    chunkGo.transform.GetChild(_decoSet).gameObject.SetActive(true);
            } else {
                if(_entryExit > 5)
                    chunkGo.transform.GetChild(_decoSet).gameObject.SetActive(true);

                _decoChange--;
            }

            for (int y = 0; y <= StreetStyles-1; y++) {
                chunkGo.transform.GetChild(0).transform.GetChild(y).gameObject.SetActive(false);
            }

            if (_entryExit == 0) {
                chunkGo.transform.GetChild(0).transform.GetChild(1).gameObject.SetActive(true);
                _entryExit++;
            } else if (_entryExit > 0 && _entryExit < 5){
                chunkGo.transform.GetChild(0).transform.GetChild(2).gameObject.SetActive(true);
                _entryExit++;
            } else if (_entryExit == 5) {
                chunkGo.transform.GetChild(0).transform.GetChild(3).gameObject.SetActive(true);
                _entryExit++;
            } else if (_entryExit > 5) {
                chunkGo.transform.GetChild(0).transform.GetChild(0).gameObject.SetActive(true);
                _entryExit++;

                if (_entryExit > ExitChangeFrequency)
                    _entryExit = 0;
            }
        }
    }

    if (_indexOfCurrentChunk > 0)
    {
        float distance = Vector3.Distance(CalcOffset(player.position),
            (_chunks.ToArray()[_indexOfCurrentChunk - 1]).position);
        if(distance > (chunkLength * .75f))
            SweepPreviousChunk();
    }
}
```

Figure 4.10: The improved and extended Fixed-Update function of the Road Generator script. This shows the random loading of new layouts for the street and the corresponding placement in the order.

### 4.3.3 Traffic Generation with Artificial Intelligence

Traffic was implemented by placing points on each lane along the traffic lanes (Figure 4.11). The vehicles head for these points using the A-Star algorithm. Thus, the vehicles are also able to follow curves, change lanes, or return to the lane when the vehicle leaves the lane.

#### A\* Algorithm

A common pathfinding algorithm in computer science and game development is the A\* (A-Star) algorithm. It was introduced by Hart et al. (1968). The algorithm plots a walkable path between multiple points or nodes efficiently on the graph. A\* discovers the least-cost path (among one or more feasible goals) from an initial node to a target node via a best-first search. To calculate the cost of the path from a particular node to the target, it uses a heuristic estimate, allowing it to prioritize nodes that are likely to be part of the final shortest path. The algorithm, unlike other pathfinding algorithms, guarantees the shortest path due to this heuristic. As a result, A\* is widely used in various applications, including maps, robotics, and video games, where an efficient and optimal solution is required.

Based on the A-Star algorithm, there are improved variables that provide better and earlier detection for avoiding obstacles, described by Erke et al. (2020).

To optimize performance and also for the reason that the road appears and disappears variably, it is just as necessary for the vehicles to appear and disappear variably. To hide this implementation from the players and to not disturb the immersion of the game, the vehicles spawn far enough away to be out of the player's field of vision. However, the vehicles do not spawn too far away from the player, otherwise they would appear before the road spawns and thus fall below the road.

#### Police Cars

In addition to normal traffic, police cars have also been integrated into the spawning traffic. These do not differ from regular traffic but are equipped with special functionality. In the event of negative behavior, in this case, collisions, the police cars turn on the blue lights and roadblocks subsequently appear. Intermediate steps such as the police cars trying to intercept the player have not yet been implemented.

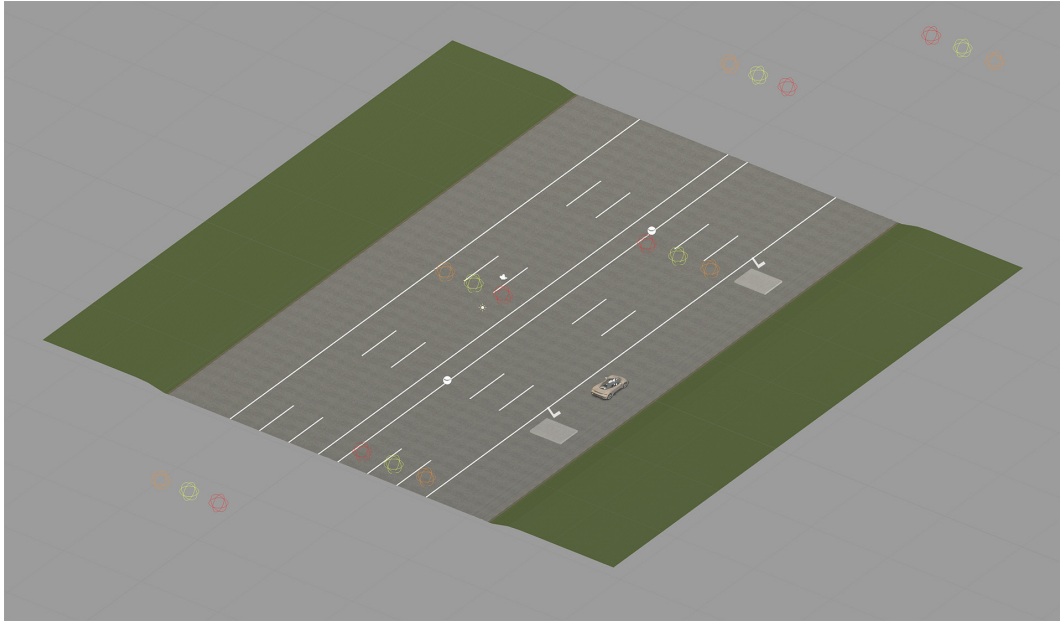


Figure 4.11: The red, yellow, and orange dots on the street show the points for the A-Star path. Each line has a group of dots.



Figure 4.12: The generated street with decoration and randomly spawned vehicles on it.

## 4 Implementation

---

```
private void Awake()
{
    _targetPointsList = this.gameObject.GetComponentsInChildren<Transform>();
}

private void Start()
{
    SpawnNewVehicles(true);
    InvokeRepeating("SpawnNewAIVehicles", 5.5f, 3.5f);
}

private void Update()
{
    for (int i = 0; i < _spawnedVehicles.Count; i++)
    {
        GameObject vehicle = _spawnedVehicles[i];
        if (Vector3.Distance(vehicle.transform.position, _playerVehicle.transform.position) > maxSpawnDistance)
        {
            _spawnedVehicles.RemoveAt(i);
            Destroy(vehicle);
        }
    }
}

private float MinDistanceToOtherVehicles(Vector3 position) {
    float minDistance = float.MaxValue;
    for (int i = 0; i < _spawnedVehicles.Count; i++)
    {
        GameObject vehicle = _spawnedVehicles[i];
        float currentDistance = Vector3.Distance(vehicle.transform.position, _playerVehicle.transform.position);
        if (currentDistance < minDistance) {
            minDistance = currentDistance;
        }
    }
    return minDistance;
}

private void SpawnNewAIVehicles()
{
    SpawnNewVehicles(false);
}

private void SpawnNewVehicles(bool startInit)
{
    Physics.SyncTransforms();
    for (int i = 0; i < _targetPointsList.Length - 1; i++)
    {
        Vector3 spawnPosition = _targetPointsList[i].position;
        Vector3 playerPosition = _playerVehicle.transform.position;
        float distance = Vector3.Distance(spawnPosition, playerPosition);

        if ((MinDistanceToOtherVehicles(spawnPosition) > 25f && ((Random.Range(0, 10) > 7 && distance > minSpawnDistance)) || startInit)
            && distance < maxSpawnDistance && _spawnedVehicles.Count < maxVehicles)

        {
            _vehicleToSpawn = trafficCarArray[Random.Range(0, trafficCarArray.Length)];
            GameObject vehicle = Instantiate(_vehicleToSpawn, spawnPosition, _targetPointsList[i].rotation);

            Material myNewMaterial = new Material(Shader.Find("Reflective/Diffuse"));
            myNewMaterial.SetColor("_Color", carColors[Random.Range(0, carColors.Length)]);
            myNewMaterial.SetColor("_ReflectColor", carColors[2]);
            myNewMaterial.SetTexture("_Cube", cubeMap);

            if (vehicle.name == "CarA_AI(Clone)")
                vehicle.transform.GetChild(0).transform.GetChild(0).transform.GetChild(22).GetComponent<Renderer>().material = myNewMaterial;

            if (vehicle.name == "CarB_AI(Clone)")
            {
                Material[] matArray = vehicle.transform.GetChild(0).GetComponent<Renderer>().materials;
                matArray[1] = myNewMaterial;
                vehicle.transform.GetChild(0).GetComponent<Renderer>().materials = matArray;
            }

            UnityStandardAssets.Vehicles.Car.CarAIControl p = vehicle.GetComponent<UnityStandardAssets.Vehicles.Car.CarAIControl>();
            p.targetpointsiterator = i + 1;
            p.m_Target = this.transform;
            _spawnedVehicles.Add(vehicle);
        }
    }
}
```

Figure 4.13: Functions created to implement random spawning vehicles on the street.

## 4.4 Garage and Store Environment

While the vehicle can be driven on the road as discussed in the last chapter, a scene was designed where the vehicle can be viewed without distraction. For this purpose, the garage was designed (Figure 4.14) where the current vehicle is displayed. Additionally, a store was created where new vehicles are displayed and can then be purchased (Figure 4.15). While the garage scene is composed of license-free textures and meshes, the store scene is based on simple black walls and a mixture of fog and post-processing to allow the desired look.

The two scenes pursue different approaches. While the vehicle in the island garage scene is to be simulated in daylight to thus create a natural and friendly impression on players when opening the app, the store scene focuses on a dimmed, minimalist appearance to bring the focus entirely to the vehicle. In addition, the doors can be opened at the push of a button, which then moves upwards in an animation and results in a view like in Figure 4.2.



Figure 4.14: The garage is implemented as a plateau on an island with a rotating base plate to be able to view the vehicle from all sides without changing the viewing angle.



Figure 4.15: The store environment is implemented as a simple dark room to keep the focus on the vehicle.

## 4.5 User Interface

The user interface was designed in such a way that it can be adapted for both mobile and desktop devices. The input is possible via touch screens and with the keyboard. The main color scheme is dark blue and black with white lettering. The user interface has been designed so that most of the elements are generated to allow easy expansion of the elements in the future. For example, this allows the display of an error-free user interface, if car data are updated via a JSON file on the server, but not the application itself.

### 4.5.1 Navigation Bar

The main component of the user interface is the navigation bar, which can be used to navigate through the different tabs. The navigation bar allows easy access to the 'Market', 'Collection', 'Garage', 'Events', and 'Profile' menus, shown in Figure 4.16.

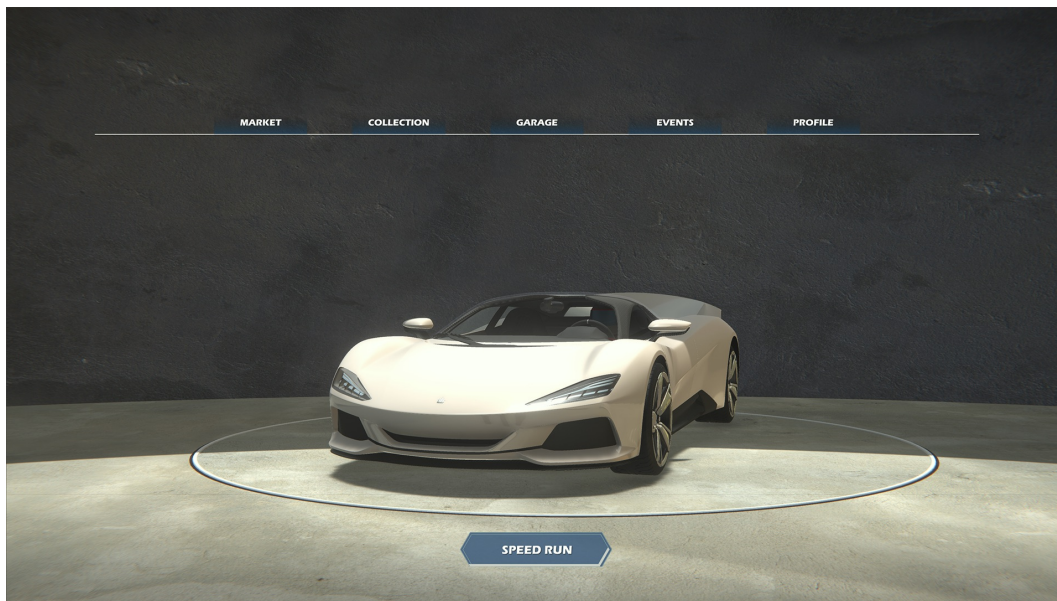


Figure 4.16: Standard view which shows the car in the garage and the navigation bar.

### 4.5.2 Collection

All vehicles are displayed in the collection with five entries per row (Figure 4.17). The entries have three different states and a distinction was made between unseen, seen, and owned. In the unseen entry, only the number of



the vehicle appears, if the vehicle has been seen, an outline of the vehicle appears and if the vehicle is or was owned, the corresponding vehicle is displayed. If the vehicle is currently in possession, the border of the car tile appears in blue. A car key also appears in the upper right corner if the vehicle is currently owned. In addition, the mastery of the vehicle is displayed in the upper left corner of each entry. The mastery is based on different missions which have to be completed to increase the mastery level. Clicking on the vehicle opens a new view, where the player can view the car in 3D and look up detailed information like the car's stats, history, or installed upgrades.

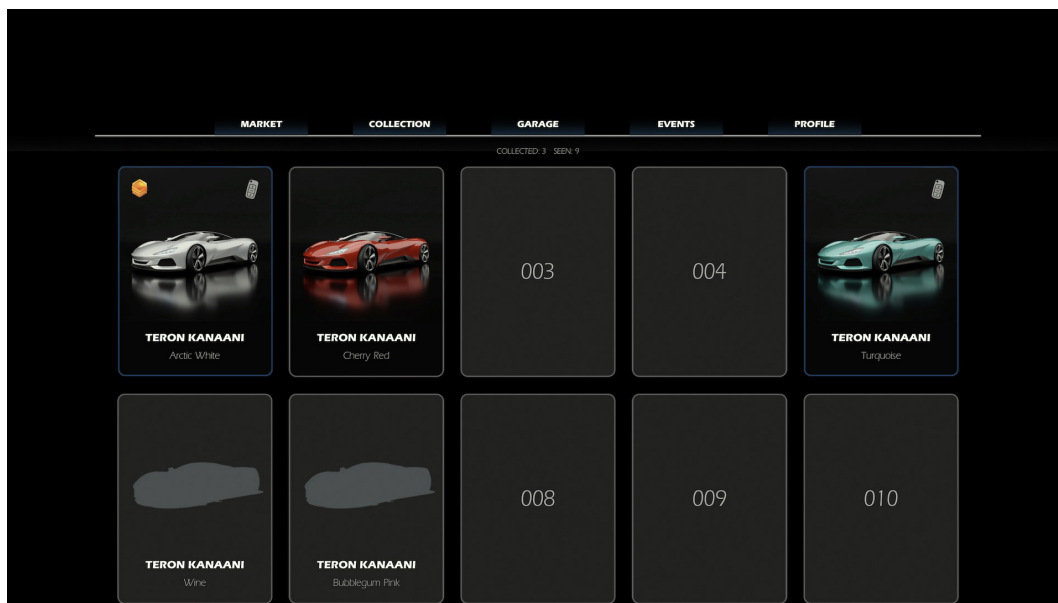


Figure 4.17: The user interface of the collection shows several entries with different states. The first car in the row is owned, the second was owned at one point but was traded away, the third field shows an unknown vehicle with the number 3 and the first entry in the second row shows a vehicle that was seen, but was never owned.

To increase the feedback of the game, additional pop-ups appear for new achievements, when purchasing new vehicles (Figure 4.18) or after each race when new medals are won or new speed records are achieved (Figure 4.19).

The animated elements such as a rotating background as seen in Figure 4.18 and animated texts make the user experience livelier and more exciting. The progress screen in Figure 4.19 shows the car, level, trophies won, the highest speed achieved, distance driven, and the time in possession.

The vehicle-specific data that was recorded during the game and saved in the player preferences is displayed on this screen, shown in Figure 4.20.



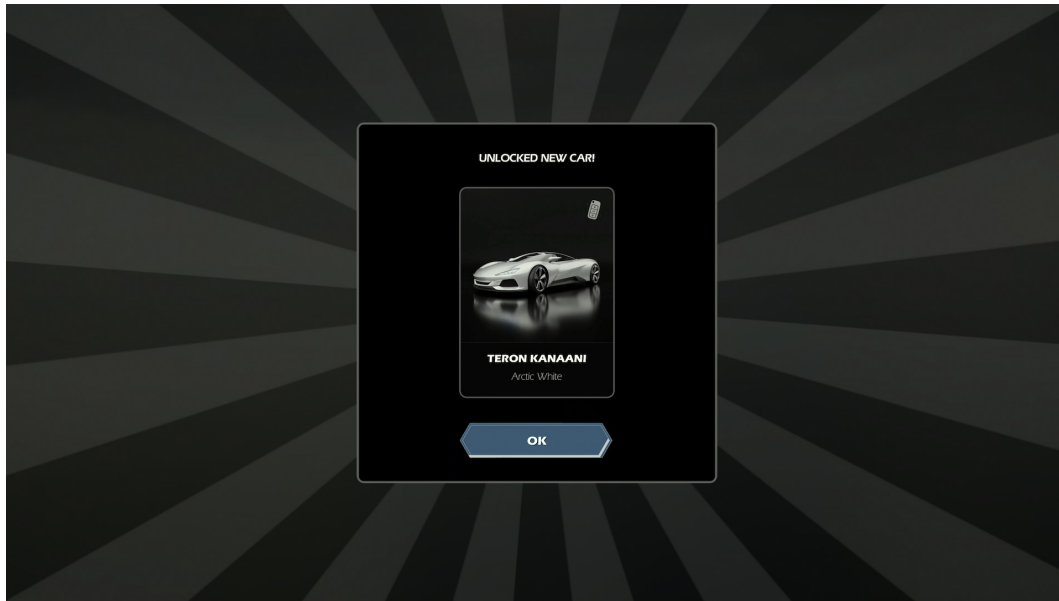


Figure 4.18: A pop-up with an animated background celebrating the receipt of a new vehicle.

On the left side, the vehicle is shown with the mastery badge, ownership identification, picture, name, and color. The progress is broken down on the right. These are divided into trophies received, highest speed achieved, total distance driven, and the time the vehicle has already been owned.

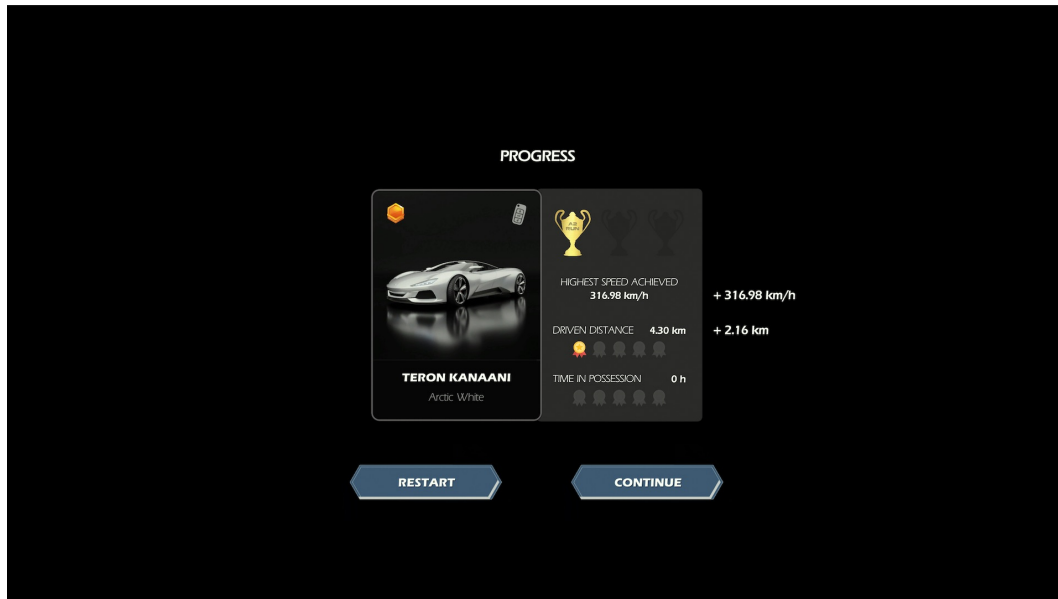


Figure 4.19: The car progress screen shows the distance covered, as well as new speed records and trophies obtained with the vehicle.

```
public static int UpdateMastery(int carModelID, int carUnitID)
{
    int mastery = 0;

    if (CarStats.GetPossessionTime(carModelID, carUnitID) > 24)
        mastery++;

    if (CarStats.GetDistanceDriven(carModelID, carUnitID) > 0)
        mastery++;

    if (CarStats.GetDistanceDriven(carModelID, carUnitID) > 10)
        mastery++;

    if (CarStats.GetDistanceDriven(carModelID, carUnitID) > 100)
        mastery++;

    if (CarStats.GetTopSpeedRecord(carModelID, carUnitID) > 200)
        mastery++;

    return mastery;
}
```

Figure 4.20: The car's level is dynamically computed through a simple process, where queries add up the number, resulting in the current level.

### Car Detail View

In the car detail view (Figure 4.21), the vehicle is displayed in the center, which can be viewed in a 360-degree view. Vehicle data is displayed in the left section, specifically the performance of the vehicle, as well as the weight, the type of drive, gearshift, color, and serial number. In the upper middle section, the user gets a short description of the vehicle, information about the model, and the unique selling points of the model such as the paintwork. In the right segment of the screen, further stats, the car mastery, and upgrades of the car are showcased.

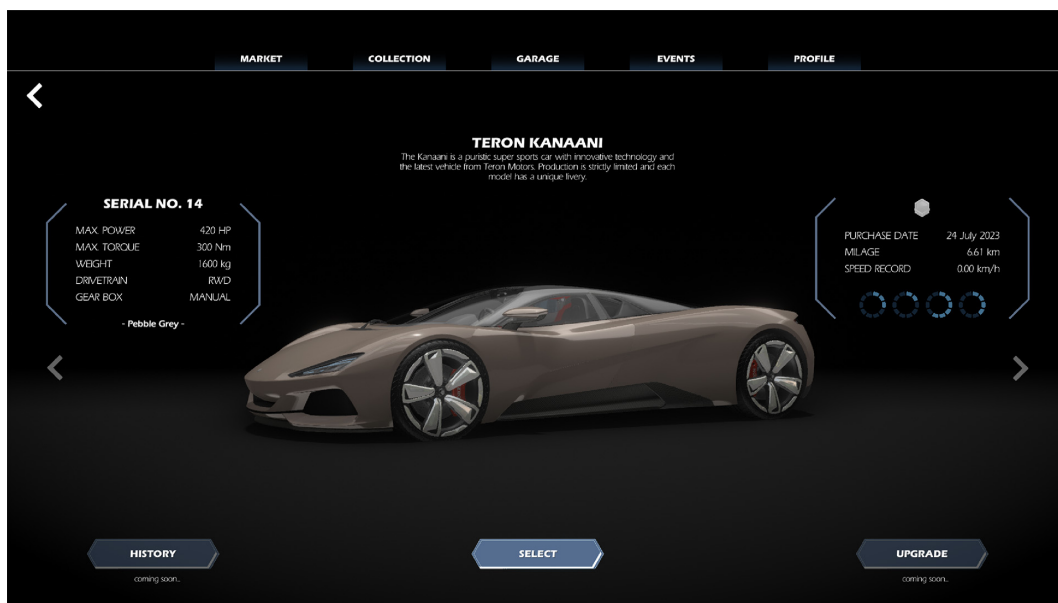


Figure 4.21: Detail view of the car and user interface showing a description, statistics, mastery, and upgrades with the particular car. A previous owner history and a separate upgrade screen are displayed with the note 'coming soon...'.

### 4.5.3 Market Place

On the marketplace, the owned vehicles are displayed on the left and the vehicles offered for purchase by other players are displayed on the right (Figure 4.22). The area on the right is automatically updated at regular intervals to always be able to offer new vehicles. On the left side, it is intended that the players can later offer their vehicles at a desired price. In the current implementation, however, they could only be sold at a given price. The price and the seller are displayed under each vehicle.

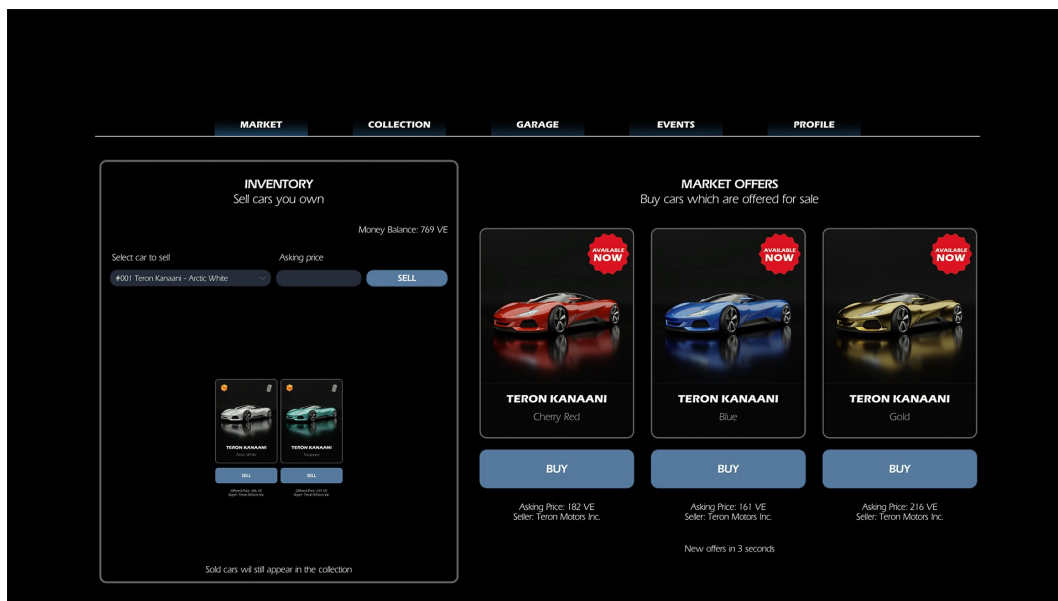


Figure 4.22: View of the integrated marketplace. On the left side is the inventory from which vehicles can be selected for sale. On the right side, the player will find current offers from other players or the manufacturer.

In addition, an alternative, simplified interface was created. This provides a better overview of the player's inventory and simplifies the trading process (Figure 4.23). The layout is now horizontal instead of vertical, and the purchase button is no longer located under the image but is the image of the vehicle itself. When buying or selling, a separate confirmation window appears.

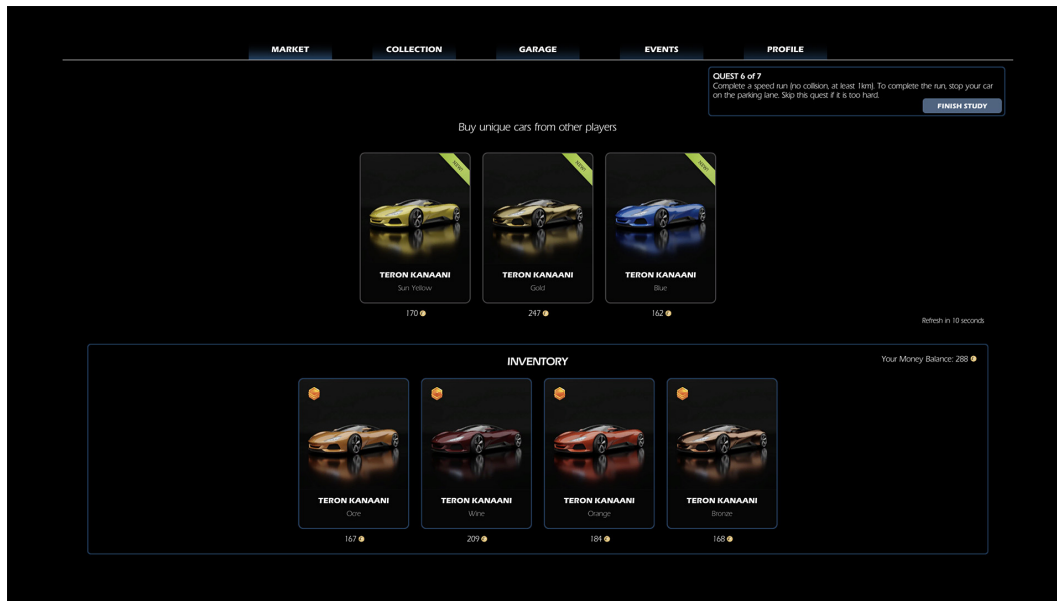


Figure 4.23: Alternative and simplified interface of the marketplace with a more elegant and simplified presentation.

### 4.5.4 Events

The event page shows the currently running event (Figure 4.24) and previews of future events (Figure 4.25). Events are limited in time and have a ranking system in which the players are ranked among other players based on their best times. In addition to the leaderboard, there are also prestige levels that can be increased within each event. The division into events also ensures that players have more chances to achieve good positions on the leaderboard and do not have to compete with high scores achieved years ago. In addition, a subdivision into several vehicle classes is implemented through the events. For example, in the future, there could be an event in which only low-powered vehicles are allowed or for off-road events previewed in Figure 4.25.

## 4 Implementation

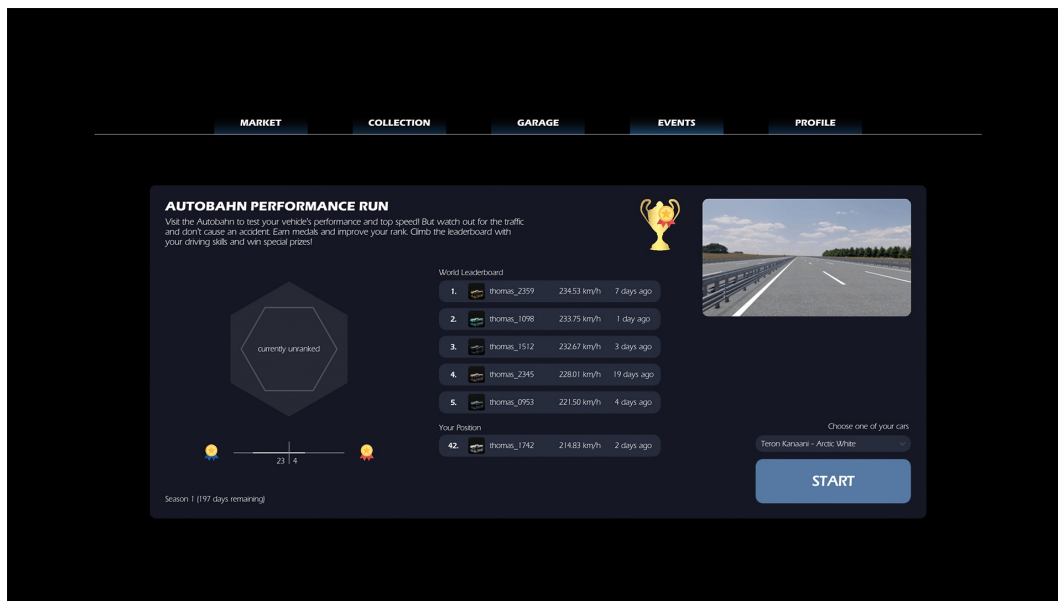


Figure 4.24: The event page shows current running events including a rank, leader board, and medal count.



Figure 4.25: Preview of upcoming events for different vehicle classes and on different maps and terrains.

### 4.5.5 Profile

The profile tab is divided into two areas. On the left side, next to the user-name, the players will find the game statistics, such as the races completed and the total playing time. Achievements are in the right section, containing medals that can be obtained and upgraded by completing certain tasks. An example is a medal for distance covered. A bronze medal is awarded for 100 km, a silver medal for 1.000 km, and a gold medal for 10.000 km. Another example is reaching a certain number of completed races or hidden trophies that can only be obtained with a bit of luck without specifically working towards it (Figure 4.26).

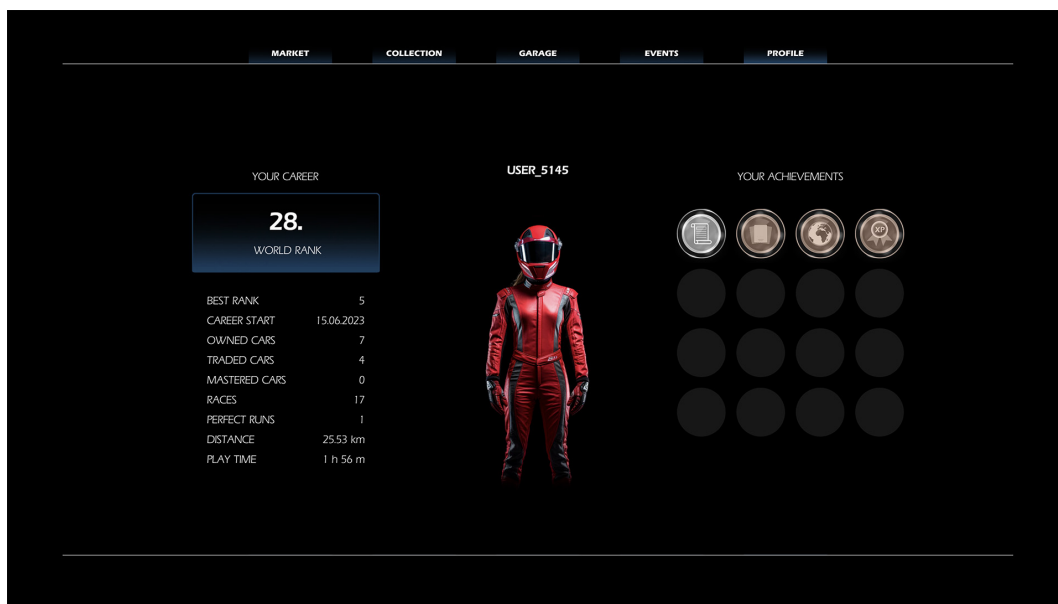


Figure 4.26: The profile screen shows the user name, avatar, statistics, world rank, and achievements.

The following code (Figure 4.27) shows parts of the implementation to calculate the medals based on the user data and generate the corresponding user interface elements. Only the motive is different for each medal, while the color and the background of the medal are put together via coding. The 'CalcAchievement' function offers a simple and clear interface to create new achievements. All the developer has to do is add a graphic, give a title, description, as well as threshold values for the individual medals, and the new achievement will be implemented.

## 4 Implementation

```
private void RefreshAchievements()
{
    CalcAchievement(0,"Racing Master",
        UserStats.GetTotalRaces() >= 10,
        UserStats.GetTotalRaces() >= 100,
        UserStats.GetTotalRaces() >= 1000,
        "Win races to improve your medal.");

    CalcAchievement(1,"World Traveller",
        UserStats.GetTotalDistance() >= 10,
        UserStats.GetTotalDistance() >= 100,
        UserStats.GetTotalDistance() >= 1000,
        "Drive many kilometers to improve your medal.");

    CalcAchievement(2,"Enthusiast",
        UserStats.GetTotalPlayTimeSeconds() >= 3600,
        UserStats.GetTotalPlayTimeSeconds() >= 36000,
        UserStats.GetTotalPlayTimeSeconds() >= 360000,
        "A medal to honor your total play time.");

    CalcAchievement(3,"Quest Master",
        UserStats.GetAbsolvedQuests() >= 2,
        UserStats.GetAbsolvedQuests() >= 4,
        UserStats.GetAbsolvedQuests() >= 7,
        "Absolve quests to earn this medal.");
}

private void CalcAchievement(int i, string title, bool bronzeCondition, bool silverCondition,
    bool goldCondition, string description)
{
    string trophyName = title.Replace(" ", "");
    pnlUserAchievements.transform.GetChild(i).GetComponent<Image>().sprite = Resources.Load<Sprite>("Achievements/base_empty");
    Transform achievement = pnlUserAchievements.transform.GetChild(i);
    achievement.transform.GetChild(0).gameObject.SetActive(false);
    achievement.transform.GetChild(1).gameObject.SetActive(false);
    achievement.transform.GetChild(2).gameObject.SetActive(false);

    if (bronzeCondition)
    {
        achievement.GetComponent<Image>().sprite = Resources.Load<Sprite>("Achievements/base_silver");
        achievement.transform.GetChild(0).gameObject.SetActive(true);
        achievement.transform.GetChild(1).gameObject.SetActive(true);
        achievement.transform.GetChild(0).GetComponent<Image>().sprite = Resources.Load<Sprite>("Achievements/" + trophyName);
        achievement.transform.GetChild(1).GetComponent<Image>().sprite = Resources.Load<Sprite>("Achievements/overlay_bronze");
    }
    if (silverCondition)
    {
        achievement.GetComponent<Image>().sprite = Resources.Load<Sprite>("Achievements/base_silver");
        achievement.transform.GetChild(0).GetComponent<Image>().sprite = Resources.Load<Sprite>("Achievements/" + trophyName);
        achievement.transform.GetChild(1).GetComponent<Image>().sprite = Resources.Load<Sprite>("Achievements/overlay_silver");
    }
    if (goldCondition)
    {
        achievement.GetComponent<Image>().sprite = Resources.Load<Sprite>("Achievements/base_gold");
        achievement.transform.GetChild(0).gameObject.SetActive(false);
        achievement.transform.GetChild(1).GetComponent<Image>().sprite = Resources.Load<Sprite>("Achievements/overlay_gold");
        achievement.transform.GetChild(2).gameObject.SetActive(true);
        achievement.transform.GetChild(2).GetComponent<Image>().sprite = Resources.Load<Sprite>("Achievements/" + trophyName);
    }

    pnlUserAchievements.transform.GetChild(i).GetComponent<Button>().onClick.AddListener(delegate () {
        ShowMessageBox(pnlMessageBox, title, description);
    });
}
```

Figure 4.27: This code shows the calculation and user interface generation of the achievements.



### 4.5.6 Racing Interface

During the race, the distance covered and the maximum speed achieved are displayed. In the event of a crash, a message on the screen will indicate that the speed record is rated invalid. Likewise, if the player drives in the wrong direction, a warning appears to turn the vehicle otherwise the race will be aborted. It is possible to pause the game by pressing the pause button in the upper left corner or by pressing the escape key. In the lower part of the screen, there is a section to display the vehicle data. In addition to the driving speed, engine speed, and the current gear, acceleration forces are also displayed visually. Assistance systems can be switched on or off by pressing the displayed buttons (Figure 4.28). The buttons are modeled based on the buttons on racing steering wheels.

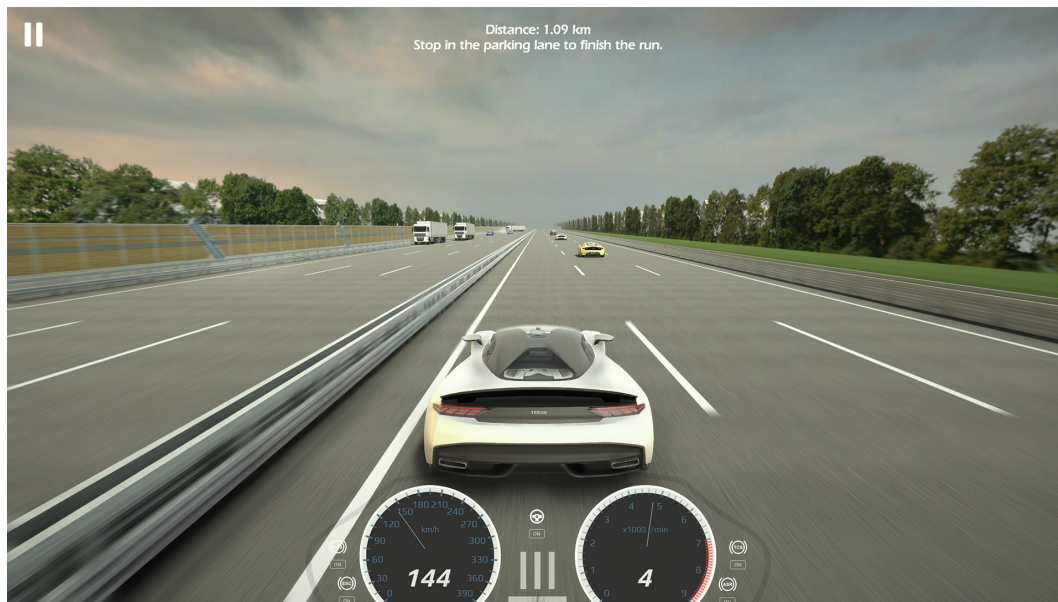


Figure 4.28: User interface during the speed run showing a gauge cluster in the lower middle of the screen, a pause button in the upper left corner, and the level progress in the upper middle.

### 4.5.7 Additional Interfaces

In addition to the interfaces already mentioned, several other small interfaces were developed, such as the pause screen, seen in Figure 4.29. Furthermore, interfaces were designed for the end of the race, loading screens, a screen for a mastery level-up of the vehicles, and an intro scene. To elegantly connect all scenes, a fade-out transition was implemented, in which the current scene is gradually darkened. At the end of the game or in the event of a crash, a

slowdown effect is additionally used, which slows down the time until it comes to a standstill.

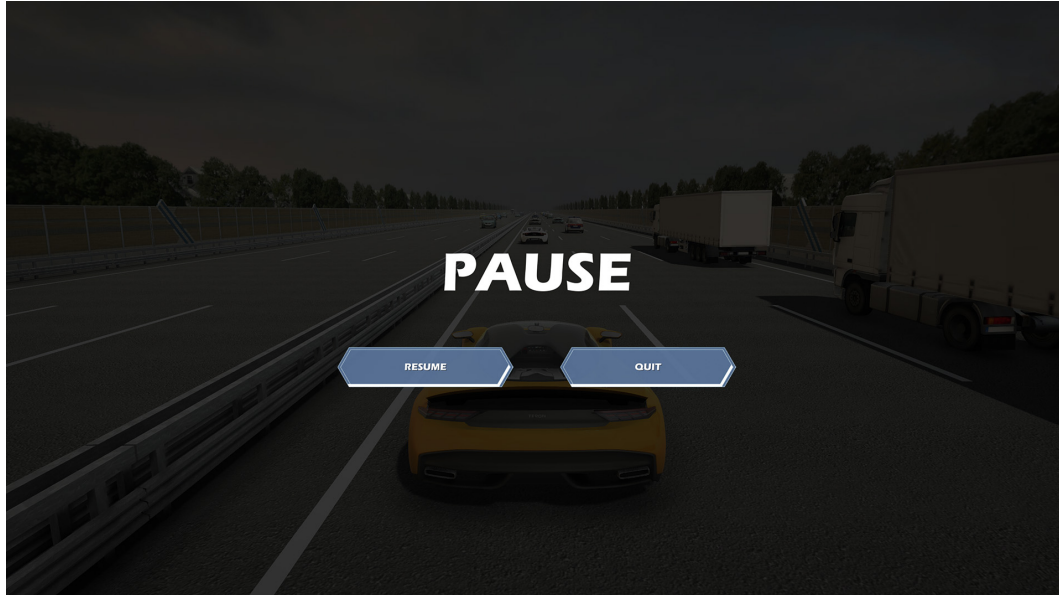


Figure 4.29: The pause screen displays a button to continue the race and one to end it, on a darkened screen.

## 4.6 Database and Server

To enable trading between players, it is necessary to use a server to exchange data between the players. This is implemented by using Unity's local player preferences and a JSON file on the user's device and asynchronous pushing the data with Unity web requests through a PHP interface to a MySQL database visualized in Figure 4.30.

### 4.6.1 Car Data

The basic structure of the game's local database is composed of a JSON file for storing the car data and the built-in player preference function from Unity. All data about the vehicles, which is unchangeable by the player, is stored in the JSON file, while the statistics about the vehicles and players are stored in the player preferences. By separating the data from the game and saving them in a JSON file, it is possible to update the game content simply by updating this file, without having to update the entire application

## 4 Implementation

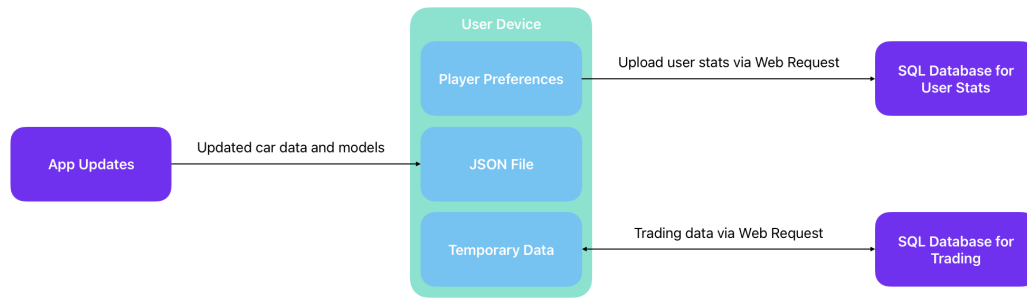


Figure 4.30: Visual presentation to give an overview of the data exchange between application, user, and server. Temporary data stands for data related to the exchange function that is stored centrally on the server and is only displayed.

at once. For example, new paint finishes and new vehicle variants based on existing 3D models can be easily distributed.

To simplify the data input and to reduce the probability of errors, the colors are specified in the JSON file in the form of HEX codes and not in the RGB format required by Unity. For this purpose, a simple HEX-RGB converter was written that converts the data accordingly.

### 4.6.2 User Data Database

To be able to evaluate the user data for the study and at the same time to be able to offer leaderboards, a server was set up on which data is transferred to a MySQL database using PHP forms.

#### Unity Web Requests

The following code shows the PostScore function which is called in Unity and sends the user data to the server (Figure 4.31).

## 4 Implementation

---

```
public static IEnumerator PostScores(string userName, int score, int carIntID, string datetime)
{
    Debug.Log("Post Scores...");
    string hash = HashInput(userName + score + AccessDatabase.SecretKey);
    string postURL = AccessDatabase.AddScoreURL + "name=" + UnityWebRequest.EscapeURL(userName) +
        "&score=" + score +
        "&hash=" + hash +
        "&car=" + carIntID +
        "&datetime=" + datetime +
        "&distance=" + Convert.ToInt32(UserStats.GetTotalDistance()) +
        "&races=" + UserStats.GetTotalRaces() +
        "&carsbought=" + UserStats.GetTotalCarsBought() +
        "&playtime=" + UserStats.GetTotalPlayTimeSeconds() +
        "&cleanruns=" + UserStats.GetTotalCleanRaces() +
        "&quests=" + UserStats.GetAbsolvedQuests() +
        "&clicksmarket=" + UserStats.GetClicksOnTab("toStore") +
        "&clickscollection=" + UserStats.GetClicksOnTab("toCollection") +
        "&clicksgarage=" + UserStats.GetClicksOnTab("toGarage") +
        "&clicksevents=" + UserStats.GetClicksOnTab("toEvents") +
        "&clicksprofile=" + UserStats.GetClicksOnTab("toProfile") +
        "&carssold=" + UserStats.GetTotalCarsSold() +
        "&id=" + UserStats.GetPlayerID() +
        "&testgroup=" + UserStats.GetPlayerTestGroup();

    UnityWebRequest hsPost = UnityWebRequest.Post(postURL, hash);
    yield return hsPost.SendWebRequest();
    if (hsPost.error != null)
        Debug.Log("There was an error posting the high score (AccessDatabase.cs): "
            + hsPost.error);
}

public static string HashInput(string input)
{
    SHA256Managed hm = new SHA256Managed();
    byte[] hashValue = hm.ComputeHash(System.Text.Encoding.ASCII.GetBytes(input));
    string hashConvert = BitConverter.ToString(hashValue).Replace("-", "").ToLower();
    return hashConvert;
}
```

Figure 4.31: C# code in Unity that transmits the data to the server using Unity WebRequest. The transmission is only partially encrypted but will be fully encrypted in the future.

The next function shows the loading of the data from the server in the `GetScores` function (Figure 4.32). The data is sent from the server to the application as a string, then the string is split and the data is output accordingly in the ranking table.

```
IEnumerator GetScores()
{
    UnityWebRequest hsGet = UnityWebRequest.Get(AccessDatabase.HighScoreURL);
    yield return hsGet.SendWebRequest();
    if (hsGet.error != null)
        Debug.Log("There was an error getting the high score (AccessDatabase.cs): " + hsGet.error);
    else {
        string dataText = hsGet.downloadHandler.text;
        MatchCollection mc = Regex.Matches(dataText, @"_");
        if (mc.Count > 0) {
            string[] splitData = Regex.Split(dataText, @"_");

            int column = 1;
            int row = 0;
            for (int i = 0; i < mc.Count; i++) {
                rankEntry.transform.GetChild(row).transform.GetChild(0).GetComponent<Text>().text =
                    row + 1 + ".";
                rankEntry.transform.GetChild(row).gameObject.SetActive(true);
                if (column == 1)
                {
                    rankEntry.transform.GetChild(row).transform.GetChild(2).GetComponent<Text>().text =
                        splitData[i];
                    column++;
                    continue;
                }
                if (column == 2) {
                    float speed = float.Parse(splitData[i]) / 100;
                    rankEntry.transform.GetChild(row).transform.GetChild(3).GetComponent<Text>().text =
                        speed + " km/h";
                    column++;
                    continue;
                }
                if (column == 3) {
                    rankEntry.transform.GetChild(row).transform.GetChild(1).transform.GetChild(0)
                        .GetComponent<Image>().sprite =
                        Resources.Load<Sprite>("cars/" + Converter.IntToString(Int32.Parse(splitData[i])));
                    column++;
                    continue;
                }
                ...
                if (column == ColumnCount) {
                    column = 1;
                    row++;
                }
            }
        }
    }
}
```

Figure 4.32: This code shows the `GetScores` function where a string from the server is split to show the results on the leaderboard.

### PHP Interface

The collected data is transferred to the server and subsequently processed by the following PHP scripts. The first script registers the user in the database and checks whether the entered username is already being used by another user or whether it is still available. Then, a unique user ID is returned (Figure 4.33).

```
...
try
{
    $databaseHandle = new PDO('mysql:host='. $hostname .' ;dbname='. $database, $username, $password);
}
catch(PDOException $exception)
{
    echo '<p>Error</p><pre>', $exception->getMessage(), '</pre>';
}

$hash = $_GET['hash'];
$trueHash = hash('sha256', $_GET['name'] . $secretKey);

if($trueHash == $hash)
{
    try {
        date_default_timezone_set("Europe/Vienna");
        $serverTime = date("Y-m-d H:i:s");
        $name = $_GET['name'];

        // Check if the name already exists in the database
        $checkName = $databaseHandle->prepare('SELECT * FROM 'scores' WHERE 'user_name' = :user_name');
        $checkName->bindParam(':user_name', $name, PDO::PARAM_STR);
        $checkName->execute();

        // If the name already exists, add an incrementing number until a unique name is found
        $i = 1;
        while ($checkName->rowCount() > 0) {
            $name = $_GET['name'] . "_" . $i;
            $checkName->bindParam(':user_name', $name, PDO::PARAM_STR);
            $checkName->execute();
            $i++;
        }

        $statementHandle = $databaseHandle->prepare('INSERT INTO 'scores'('user_name', 'stat_register_date', 'stat_platform',
        'stat_app_version') VALUES (:user_name, :stat_register_date, :stat_platform, :stat_app_version)');
        $statementHandle->bindParam(':user_name', $name, PDO::PARAM_STR);
        $statementHandle->bindParam(':stat_register_date', $serverTime, PDO::PARAM_STR);
        $statementHandle->bindParam(':stat_platform', $_GET['stat_platform'], PDO::PARAM_STR);
        $statementHandle->bindParam(':stat_app_version', $_GET['stat_app_version'], PDO::PARAM_STR);

        $statementHandle->execute();

        // Get the unique ID of the inserted row
        $unique_id = $databaseHandle->lastInsertId();

        echo $unique_id;
    } catch(Exception $exception) {
        echo '<p>Error</p><pre>', $exception->getMessage(), '</pre>';
    }
}
}
```

Figure 4.33: Excerpt from the function to register new users, programmed in PHP.

After successful registration, the data will be transferred to the database using the update function (Figure 4.34).

```

...

if($trueHash == $hash)
{
    try {
        date_default_timezone_set("Europe/Vienna");
        $serverTime = date("Y-m-d H:i:s");

        // Define an array of column names and values
        $columns = [
            'user_name' => $_GET['name'],
            'user_sex' => $_GET['usersex'],
            'user_age' => $_GET['userage'],
            'user_email' => $_GET['user_email'],
            'achievement quests' => $_GET['quests'],
            'cars_inventory' => $_GET['inventory'],
            'cars_history' => $_GET['cars_history'],
            'event_sle1_date' => $serverTime,
            ...
        ];

        // Generate the UPDATE statement
        $updateColumns = [];
        foreach ($columns as $column => $value) {
            $updateColumns[] = "$column'=:column";
        }
        $updateStatement = implode(', ', $updateColumns);
        $sql = "UPDATE 'scores' SET $updateStatement WHERE 'id' = :id";

        $statementHandle = $databaseHandle->prepare($sql);
        foreach ($columns as $column => $value) {
            $statementHandle->bindValue(":$column", $value);
        }
        $statementHandle->bindValue(":id", $_GET['id']);

        $statementHandle->execute();

    } catch(Exception $exception) {
        echo '<p>Error</p><pre>', $exception->getMessage() , '</pre>';
    }
}

```

Figure 4.34: Shortened version of the PHP code for data transmission to the server using the update function.

## 4 Implementation

---

The next code section shows the loading of the leaderboard. Unlike the previous scripts, no separate encryption of the data is necessary here. The data is transferred in a string separated by hashtags. This string is then broken down into individual values in Unity (Figure 4.35).

```
try
{
    $databaseHandle = new PDO('mysql:host='. $hostname .' ;dbname='. $database, $username, $password);
}
catch(PDOException $exception)
{
    echo '<p>Error in display.php</p><pre>', $exception->getMessage(), '</pre>';
}

$statementHandle = $databaseHandle->query('SELECT * FROM scores ORDER BY event_s1e1_score DESC LIMIT 10');
$statementHandle->setFetchMode(PDO::FETCH_ASSOC);

$result = $statementHandle->fetchAll();

if (count($result) > 0)
{
    foreach($result as $r)
    {
        echo $r['id'], "#";
        echo $r['user_name'], "#";
        echo $r['event_s1e1_score'], "#";
        echo $r['event_s1e1_car_model'], "#";
        echo $r['event_s1e1_car_unit'], "#";
        echo $r['event_s1e1_date'], "#";
        echo $r['distance'], "#";
        echo $r['races'], "#";
        echo $r['carsbought'], "#";
        echo $r['playtime'], "#";
        echo $r['cleanruns'], "#";
        echo $r['quests'], "#";
        echo $r['clicksmarket'], "#";
        echo $r['clickscollection'], "#";
        echo $r['clicksevents'], "#";
        echo $r['clicksprofile'], "#";
        echo $r['carssold'], "#";
        echo $r['testgroup'], "#";
        echo "1", "#"; /*data base version */
    }
}
```

Figure 4.35: Excerpt from the DisplayLeaderboard PHP function loading data to show the top ten leaderboard entries.



## 4 Implementation

The following PHP function prepares the data to determine the player's position on the leaderboard. This is also transmitted as a string and then broken down and output in Unity (Figure 4.36).

```
if($trueHash == $hash)
{
    try {
        $id = $_GET['id'];
        $statementHandle = $databaseHandle->prepare('SELECT (SELECT COUNT(*) + 1 FROM scores WHERE event_s1e1_score >
        (SELECT event_s1e1_score FROM scores WHERE id = :id)) as position, user_name, event_s1e1_score,
        event_s1e1_car_model, event_s1e1_car_unit, event_s1e1_date FROM scores WHERE id = :id');
        $statementHandle->bindParam(':id', $id, PDO::PARAM_INT);
        $statementHandle->execute();
        $statementHandle->setFetchMode(PDO::FETCH_ASSOC);

        $result = $statementHandle->fetch();

        echo $result['position'], "#";
        echo $result['user_name'], "#";
        echo $result['event_s1e1_score'], "#";
        echo $result['event_s1e1_car_model'], "#";
        echo $result['event_s1e1_car_unit'], "#";
        echo $result['event_s1e1_date'], "#";

    } catch(Exception $exception) {
        echo '<p>Error</p><pre>', htmlspecialchars($exception->getMessage()), '</pre>';
    }
}
```

Figure 4.36: PHP code to display the user's position in the leaderboard, including the high score, the vehicle, and the date the record was set.

The next figure (Figure 4.37) shows example entries in the database in the phpMyAdmin (The phpMyAdmin Project, 2023) user interface. The database contains user statistics generated during testing. Columns for the inventory of vehicles, money, and other features to enable trading are also present and partially implemented. These extended functionalities are not further described here because the implementation was not completed because of the focus on simulated trading with the computer and not real trading between users.

name	score	car	datetime	distance	races	carsbought	playtime	cleanruns	quests	clicksmarket	clickscollection	clicksgarage	clicksevents	clicksprofile	carssold	id	testgroup
Thomas	7915	20	2/8/2023 6:24:17 PM	NULL	0	0	0	0	NULL	NULL	NULL	NULL	0	NULL	NULL	NULL	NULL
Max Mustermann	6143	16	2/8/2023 7:17:58 PM	3	0	0	0	0	NULL	NULL	NULL	NULL	0	NULL	NULL	NULL	NULL
Max Mustermann	6743	16	2/8/2023 10:00:03 PM	3	0	0	0	0	NULL	NULL	NULL	NULL	0	NULL	NULL	NULL	NULL
Max Mustermann	8609	1	2/8/2023 10:16:00 PM	3	8	2	0	0	NULL	NULL	NULL	NULL	0	NULL	NULL	NULL	NULL
Max Mustermann	6270	14	2/9/2023 12:09:09 AM	3	9	3	892	8	NULL	NULL	NULL	NULL	0	NULL	NULL	NULL	NULL
Max Mustermann	10106	10	2/9/2023 3:14:32 PM	3	13	4	1125	12	6	3	NULL	NULL	0	NULL	NULL	NULL	NULL
Max Mustermann	11091	10	2/9/2023 3:33:30 PM	3	14	4	1203	13	6	3	10	63	37	6	NULL	NULL	NULL
Max Mustermann	10215	10	2/9/2023 4:18:22 PM	4	16	4	1285	15	6	3	10	73	45	6	0	NULL	NULL
Sepp	8529	1	2/9/2023 4:43:47 PM	0	0	1	9	1	2	0	1	3	0	0	0	787050	2

Figure 4.37: A screenshot of the database for the ranking lists and with data to evaluate the study. The upper entries show the old logging implementation while the last entry shows the updated implementation with user accounts with unique user IDs and log-in functionality.

## 4.7 Visual Appearance

To make the overall project coherent, some visual effects were added. Important components are reflection probes, which enable real-time reflection, which primarily ensures that the landscape is reflected in the vehicle paintwork. In addition, a post-processing stack was added that provides a visually more realistic representation with effects such as motion blur and ambient occlusion. In addition, an atmosphere in the form of fog was used for a more realistic view into the distance. To improve the game acoustically, energetic music for the races and calm background music for the garage were added in addition to the sound effects for vehicles and buttons.

### 4.7.1 Visual Effects

So-called high dynamic range textures can be used to improve the visual quality of the app. To use this, it is necessary to leave the default gamma color space and switch to the linear color space. This is at the expense of compatibility with older devices. To be able to display the linear color space, the device needs Metal support on iOS and support for OpenGL ES 3.0 on Android. However, this does not result in any major restrictions, as all iOS devices with the A7 Chip up from the iPhone 5S support Metal and hardware manufacturers for Android have been able to use it since Android 4.3.

Since the various effects, especially the Post Processing Stack, require a lot of computing power, a settings menu has been designed that allows the deactivation of the Post Processing Stack (Figure 4.38). This ensures a better gaming experience on weaker hardware. In addition, volume controls for sound and music have been added, as well as an explanation of the key assignment in case there are uncertainties about the controls.

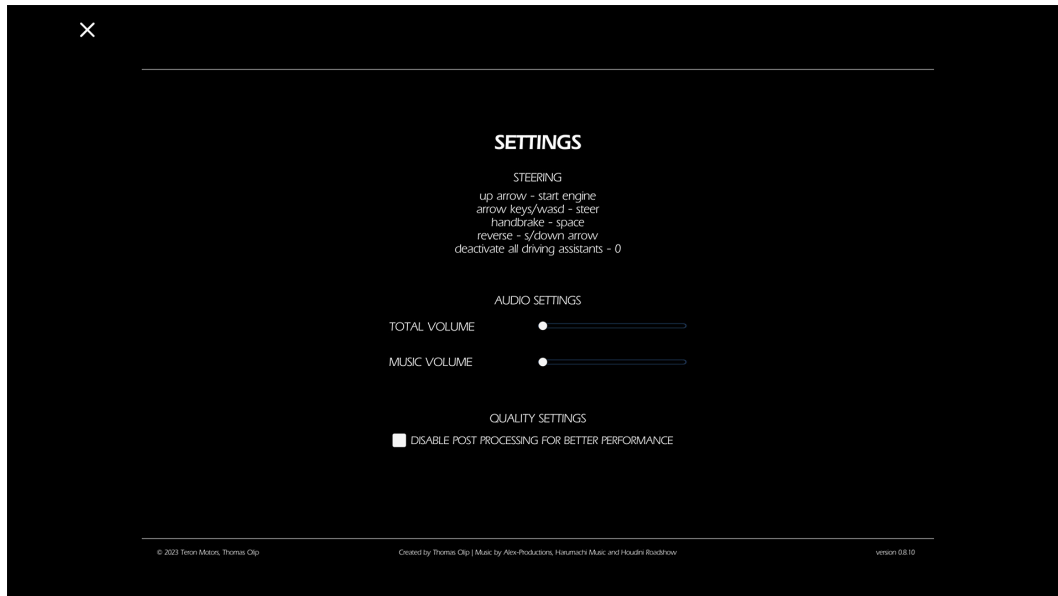


Figure 4.38: Settings screen showing the keymap, audio settings, and quality settings.

### Perception of Speed

To enhance the perception of driving speed and thus improve the user experience, five points have been selected to amplify this effect. These are:

- Post-Processing
  - The post-processing stack in Unity consists of the post-process volume and the post-process layer. The layer is responsible for anti-aliasing, while volume contains some effects. Of these, motion blur, ambient occlusion, and a vignette effect were activated. Other effects include Depth of Field, Grain, Bloom, Screen Space Reflections, and Chromatic Aberration. These were left out for efficiency reasons.
- Field of View
  - Field of View is a setting of the cameras and is already implemented in Unity. This feature was extended with a function to automatically increase the Field of View during runtime, depending on the increasing vehicle speed.
- Sound
  - Wind noise is already implemented in the driving script provided by Edy's Vehicle Physics Pro. Vehicle traffic was also equipped with engine sounds to provide a more realistic sound experience.
- Camera Shake and Distance
  - To simulate the fine unevenness of the road that results in small shocks at high speeds, the code was prepared so that a function can be implemented that simulates this in the form of increasing camera rotations with increasing speed. From 120 km/h, the camera distance in 3rd person increases gradually to increase the intensity of speed perception.
- Realistic Environment
  - To accurately represent the speed of the vehicle, care was also taken to ensure that the environment, other vehicles, and the roads were made and placed to scale. Otherwise, incorrect scaling could potentially lead to distortions in the perceived speed.

### 4.7.2 App Appearance

To round off the app visually, an icon (Figure 4.39), a cover (Figure 4.40), and various media graphics were created. The cross-platform distribution of

the app and a meaningful, simple design to reflect the content of the app well were taken into account.

Two variants have been developed for the app's name. The first, 'Teron Speedrun', specifically describes the type of events featured in the game. The second, 'Teron Racing', is more general, offering greater flexibility for incorporating different types of events. The term 'Teron' represents the fictional vehicle manufacturer whose vehicles are used within the app.



Figure 4.39: The app icon, which was developed for iOS, Android, macOS, and Windows. It is composed of the title of the app and the Teron Kanaani headlight.



Figure 4.40: The cover is optimized for websites and game sections in app stores and shows the alternative title.

## 4.8 Quest System

To enable the study to be automated as far as possible, a quest system was implemented that guides the player through the study. In the first step, the users select a username, and provide their age, and their sex (Figure 4.41). The study participants remain anonymous, the username is only used for display in the profile screen and the leaderboard and can be freely chosen. The user interface and the quests then adapt to the test group. More details on this are described in the evaluation chapter. The quests are displayed in the upper right corner of the screen throughout the entire test so that the testers always know what to do (Figure 4.42).

Every important action in the game is provided with a trigger so that the quests are automatically updated when the activities, required to complete the quest, have been completed. The UI will then be updated with the new quest. The data is transmitted to the server after each race to additionally capture participants who have dropped out of the study.

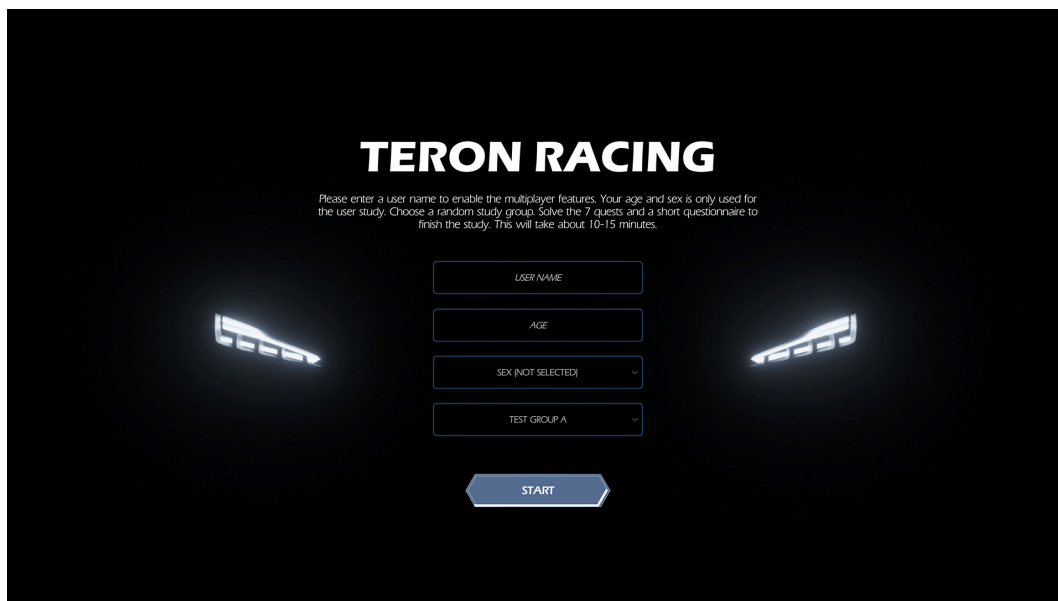


Figure 4.41: The figure shows the welcome screen where the testers choose a username, and specify their age and sex.

After all the quests are completed and the test run is thus finished, users are asked to participate in two final questionnaires. These were also implemented in Unity and the data is also transmitted to the server along with the other data (Figure 4.43).

The questionnaire user interface was developed directly in Unity and consists of single-choice questions and a submit function that checks whether all

### QUEST 3 (GROUP B)

Visit the 'MARKET' and buy another car.

Figure 4.42: Example of a quest displayed in the upper right corner of the in-game screen.

questions have been answered to subsequently transmit the data.

Questionnaire  
Just this one page, no more pages

System Usability Scale Test

	Strongly Disagree					Strongly Agree
1. I think that I would like to use this system frequently. (system means this game)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
2. I found the system unnecessarily complex.	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
3. I thought the system was easy to use.	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
4. I think that I would need the support of a technical person to be able to use this system.	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
5. I found the various functions in this system were well integrated.	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
6. I thought there was too much inconsistency in this system.	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
7. I would imagine that most people would learn to use this system very quickly.	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
8. I found the system very cumbersome to use.	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
9. I felt very confident using the system.	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
10. I needed to learn a lot of things before I could get going with this system.	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

Figure 4.43: The figure shows the screen for the questionnaires SUS and GEQ in a scroll view.

## 4.9 Summary

The game was developed from the ground up based on the driving script and a previously designed vehicle shell. The vehicle was created in Blender and further developed with moving parts, an interior, and numerous details. The model was then imported into Unity and linked to the driving script, which was then heavily adapted to achieve the desired driving behavior and performance goals.

A modular landscape was then modeled in Blender. Using a heavily extended road generator script, the modules are combined in Unity at runtime to provide a new, varied landscape.

After the basic gameplay worked smoothly, the streets became populated with traffic and police cars. These follow the lanes using the A\* algorithm and appear as varied traffic that moves at different speeds depending on the lane. The police cars react to collisions with blue flashing lights and roadblocks.

To be able to look at the cars in peace, the garage and the store were created, where the players can view their own and those available for purchase.

The most important component was then developed: the user interface, which provides the foundation for the collection and trade mechanics. This task turned out to be very extensive, as databases had to be created and a lot of data and states of the vehicles had to be recorded on top of the user interface which was dynamically created.

In addition, the market maker was implemented, which handles the local vehicle trading. The server functionality was initially tested, but ultimately only used to evaluate the study and display the leaderboards.

Particular attention was also paid to Chapter 2.4.2 about the perception of speed, which was discussed in the background. Individual effects described in this chapter were implemented with the help of the provided Unity functions.

After implementing the prototype, a test phase was carried out where several persons tested the gameplay and pointed out errors before the start of the user study. Over 30 improvements were implemented before the game was finalized and the evaluation of the trading mechanism began.



## 5 Evaluation

The methodologies and processes that were used are presented in this chapter, and then the research outcomes are shown in detail. The chapter goes into further detail on data review, statistical analysis, and observable distribution patterns. The results are examined in detail in the discussion section to ensure that they support the hypothesis.

### 5.1 Material and Setup

To conduct the study, no additional tools were needed apart from the implementation, as users could perform the test on any regular computer. Thus, testers could carry out the test at any time from home on their computers running current Windows, Mac, or Linux versions supported by Unity.

### 5.2 Method and Procedure

To answer if the research goals, defined in the objectives, were reached, several quantitative research methods were carried out in an A/B case study. The study was composed of a short pre-questionnaire, followed by a random assignment to Group A or B, followed by two post-questionnaires. Each test group received a different gathering mechanic in the game to test. Both the pre- and post-questionnaires were integrated into the Unity game and implemented with the Unity UI system. The results were automatically sent to a server set up specifically for this purpose, stored in a database there, and were then manually transferred to an Excel sheet for evaluation. Mean values and standard deviation were then calculated from the data and significance was determined using *t*-tests. Cohen's *d* was calculated using a Python script.

### 5.2.1 Pre-Questionnaire

In the pre-questionnaire, only the age and sex of the participants were recorded. Users could also choose a username, but this had no influence on the study or evaluation and was only used for display in the profile and on the leaderboard.

### 5.2.2 A/B-Test

An A/B test, shown in Figure 5.1, was carried out with the following test structure: Participants in group A received a randomly selected car after each race. Group B participants, in contrast to group A participants, had to purchase the vehicles at the marketplace but had the option of exchanging the vehicles they had collected for other vehicles at the marketplace at any time. To encourage trading, Group B testers had limited garage slots and limited in-game currency.

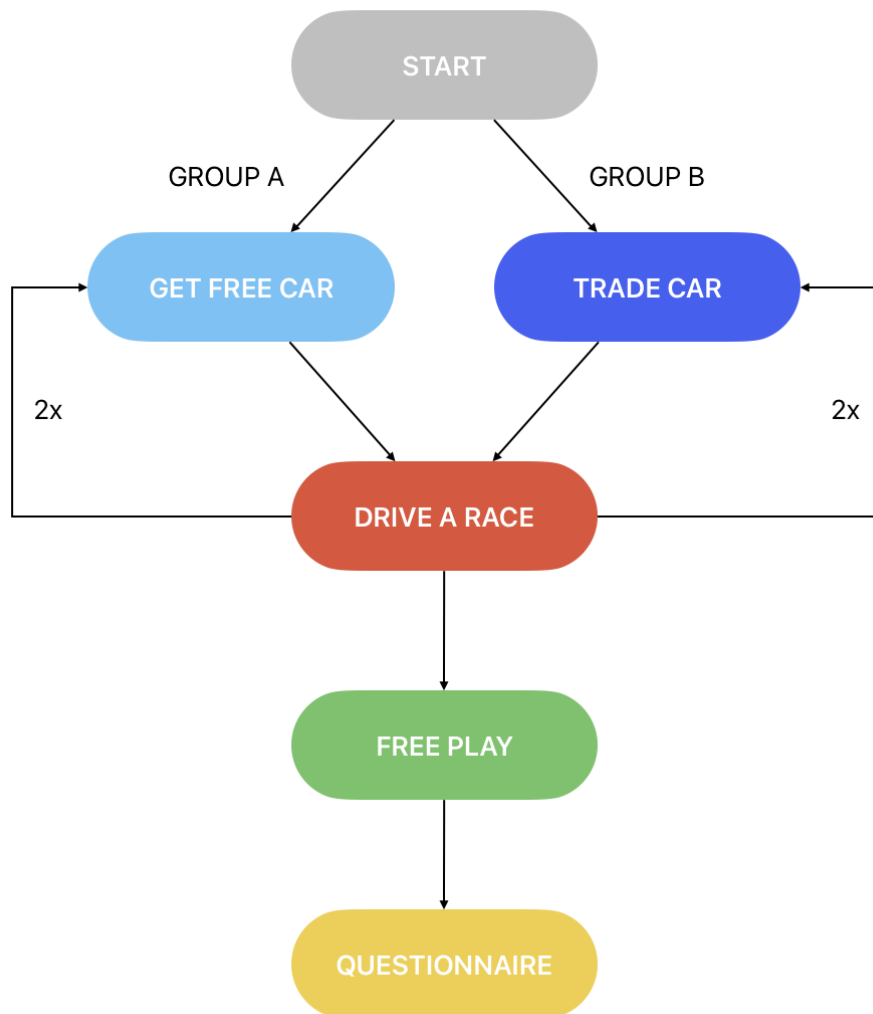


Figure 5.1: Flowchart of the A/B test sequence from start to the questionnaires which mark the end of the test.

The testers were guided through the study using the quest system. Each user received six quests, shown in Table 5.1, which must be completed one after the other. At the end of the quests, there was an open test phase in which testers could continue the game as long as they like. During the test, data on gaming, collecting, and trading behavior was automatically recorded. A detailed listing of the recorded data can be seen in Table 5.2.

Table 5.1: List of quests that users receive during testing. The letters after the numbers show the test group the quests are shown to.

Quest Number	Test Group	Quest Description
1.	A	Select your new car in the 'Collection' tab.
1.	B	Visit the 'Market', buy a car, and select it in the 'Collection' tab.
2.	A/B	Start a speed run in the 'Garage' tab. Drive at least 1 km.
3.	A	Change your car in the 'Collection' tab.
3.	B	Visit the 'Market' and buy another car.
4.	A/B	Start a speed run and drive at least 2 km. Try not to crash!
5.	A	Change your car again in the 'Collection' tab.
5.	B	Visit the 'Market' and buy another car.
6.	A/B	Complete a speed run (no collision, at least 1km). To complete the run, stop your car in the parking lane. Skip this quest if it is too hard.
7.	A/B	Feel free to play some more races. When you are done, press 'Finish' :)

Table 5.2: Listing of the data collected during user testing.

Name	Description
Play Time	Total time spent in the program, measured in seconds.
Distance	Combined driven distance with the cars, measured in meters.
Races	Total driven races, including aborted races.
Finished Runs	Total finished races without crashes or disqualification.
Cars Collected	Total cars obtained through winning or trading.
Market Visits	Clicks on the market tab in the user interface.
Collection Visits	Clicks on the collection tab in the user interface.
Garage Visits	Clicks on the garage tab in the user interface.
Event Visits	Clicks on the events tab in the user interface.
Profile Visits	Clicks on the profile tab in the user interface.
Finished Quests	Amount of quests the user absolved.

### 5.2.3 Post-Questionnaire

After the quest-guided test, users were asked about the implementation in two questionnaires. First, the System Usability Scale Test (SUS), developed by Brooke (1995) and then the Game Experience Questionnaire (GEQ) - In-Game Module, developed by IJsselsteijn et al. (2013).

#### System Usability Scale Test

The System Usability Scale Test consists of ten questions and yields a score between 0 and 100 when evaluated, with 100 being the best rating and 0 the worst. In this test, the following questions were provided (Table 5.3) and could be answered from 'Strongly Disagree' (1) to 'Strongly Agree' (5).

Table 5.3: Questions of the System Usability Scale Test.

Number	Question
1.	I think that I would like to use this system frequently.
2.	I found the system unnecessarily complex.
3.	I thought the system was easy to use.
4.	I think that I would need the support of a technical person to be able to use this system.
5.	I found the various functions in this system were well integrated.
6.	I thought there was too much inconsistency in this system.
7.	I would imagine that most people would learn to use this system very quickly.
8.	I found the system very cumbersome to use.
9.	I felt very confident using the system.
10.	I needed to learn a lot of things before I could get going with this system.

### Game Experience Questionnaire

The Game Experience Questionnaire uses the shortened version (In-Game Module) of the core module to keep the duration of the test short. The questions are shown in Table 5.4 and are rated with values between 0 (not at all) and 4 (extremely). The questionnaire consists of 14 questions, resulting in six values when evaluated: Competence, Sensory and Imaginative Immersion, Flow, Tension, Challenge, Negative affect, and Positive affect.

Table 5.4: Questions of the Game Experience Questionnaire (In-Game Module).

Number	Question
1.	I was interested in the game's story
2.	I felt successful
3.	I felt bored
4.	I found it impressive
5.	I forgot everything around me
6.	I felt frustrated
7.	I found it tiresome
8.	I felt irritable
9.	I felt skillful
10.	I felt completely absorbed
11.	I felt content
12.	I felt challenged
13.	I had to put a lot of effort into it
14.	I felt good

## 5.3 Participants

A total of 39 participants took part in the study. Of these, 8 were female and 31 were male. The age of the participants ranged from 14 to 41 years, with an average age of 25. More details in Table 5.5. Since the link to take

part in the study was freely distributed, anyone could take part in the study, which resulted in some people entering an arbitrary, unrealistic number for their age (e.g. 120 years). These values were removed before calculating the average age. Of the 39 participants whose game data was recorded, 30 filled out the subsequent questionnaires. Technical knowledge was not necessary to participate in the study. The only requirement was the independent installation of the application on the tester's computer.

Table 5.5: Participants broken down by age and sex.

	Total	Female	Male	Age	
				<i>M</i>	<i>SD</i>
Group A	17	5	12	24.07	7.43
Group B	22	3	19	25.67	5.03
Total	39	8	31	25.00	6.10

## 5.4 Results

In this section, the results of the A/B study are presented. These consist of the results of the two questionnaires and the evaluation of user data recorded during the study. The results were evaluated according to the respective calculation key and then examined for significance using two-sample *t*-tests and Cohen's *d*. Due to the different number of testers per group, unpaired *t*-tests were used. Since the null hypothesis is assumed, equal variance is used instead of unequal variance. The value for alpha is 0.05 and the hypothesized mean difference is 0.

### 5.4.1 System Usability Score

The ten questions in the questionnaire result in one score per person, which is then averaged. The results can be seen in Figure 5.2. Group A (blue bar, without trading) scores 69.29 (*SD* = 22.69) out of 100 points, while Group B (orange bar, with trading) scores 80.54 (*SD* = 7.73) out of 100 points, which indicates slightly better usability in the trading test group.



To better classify the results, Bangor et al. (2009) added an adjective rating scale to the test. According to this classification, Group A's score is 'OK', while Group B's score is 'Good'.

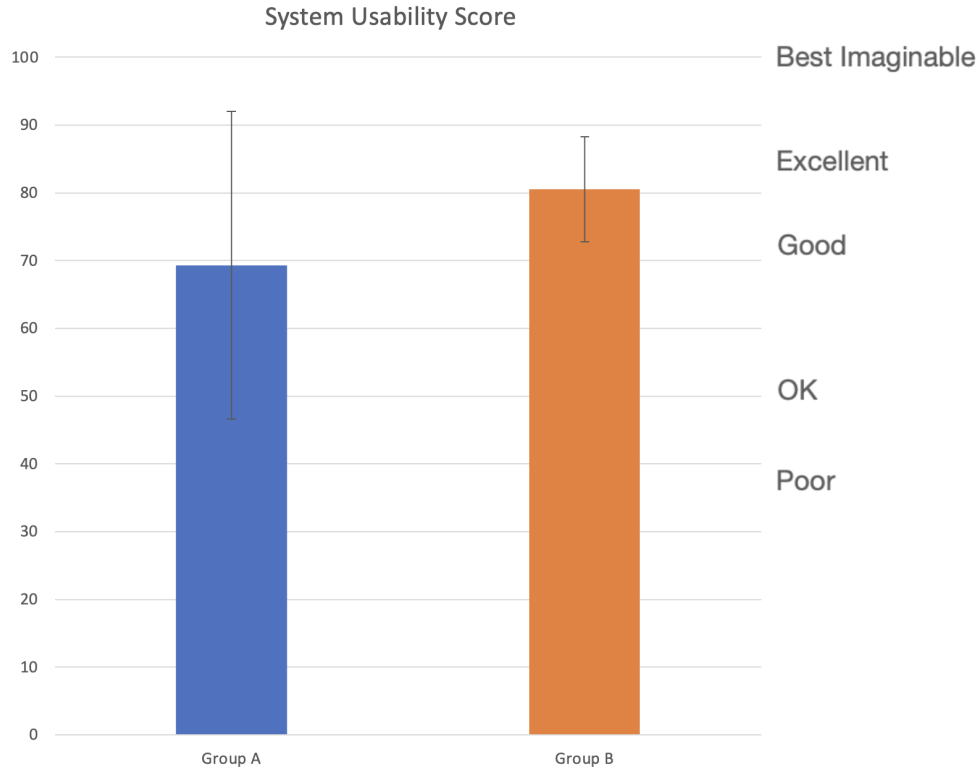


Figure 5.2: Comparison of the SUS results from the test group without a trading function (Group A, blue bar) and the test group with a trading function (Group B, orange bar), extended with Bangor et al. (2009)'s adjective rating scale.

The  $t$ -test assuming equal variances shows that the differences between group A and B are not statistically significant ( $t(26) = 1.76$ ,  $p = .090$ ,  $\alpha = .05$ ), also visible in Table 5.6.

In the System Usability Score Test, the questions are framed once positively and once negatively. The questions are contradictory, which is why if the testers tick the same answer everywhere, the ratings cancel each other out. This is designed to prevent response bias, where participants might otherwise tend to select the same response for all questions. However, if a participant consistently selects the same response for all questions, it could indicate a misunderstanding of the questionnaire or a lack of attention to the questions. This was the case for two participants who picked the same answer for each question and another participant who also rated

Table 5.6: Results of the System Usability Score Questionnaire, showing the mean, standard deviation, and the statistical significance with  $\alpha = .05$ .

	Group A		Group B		$t(26)$	$p$	Cohens's $d$
	$M$	$SD$	$M$	$SD$			
System Usability Score	69.29	22.69	80.54	7.73	1.76	.090	-0.66

the contradictory questions largely the same. All three participants were removed from the data set before the analysis. The decision was made, based on the view that their responses did not accurately represent their user experience.

#### 5.4.2 Game Experience Questionnaire Score

The results of the Game Experience Questionnaire can be seen in Figure 5.3. The results of the two test groups were very similar but showed a large standard deviation. Generally, it can be said that the negative affect was very small and the positive affect was rather high while Competence and Challenge also tend to be large and the other values move in the midfield.

In detail, Tension was rated higher ( $M = 1.85$ ,  $SD = 1.30$  vs.  $M = 1.54$ ,  $SD = 0.91$ ) in Group B, the group with the trading mechanic. However, the negative affect in the group has also been rated higher ( $M = 1.72$ ,  $SD = 1.46$  vs.  $M = 0.96$ ,  $SD = 0.77$ ). In Group A, the group where vehicles are obtained through progress in the game, the value for Challenge, has been rated higher ( $M = 2.82$ ,  $SD = 0.67$  vs.  $M = 2.38$ ,  $SD = 1.07$ ).

Unfortunately, none of the samples were statistically significant. This was true for all values, also shown in detail in Table 5.7.

## 5 Evaluation

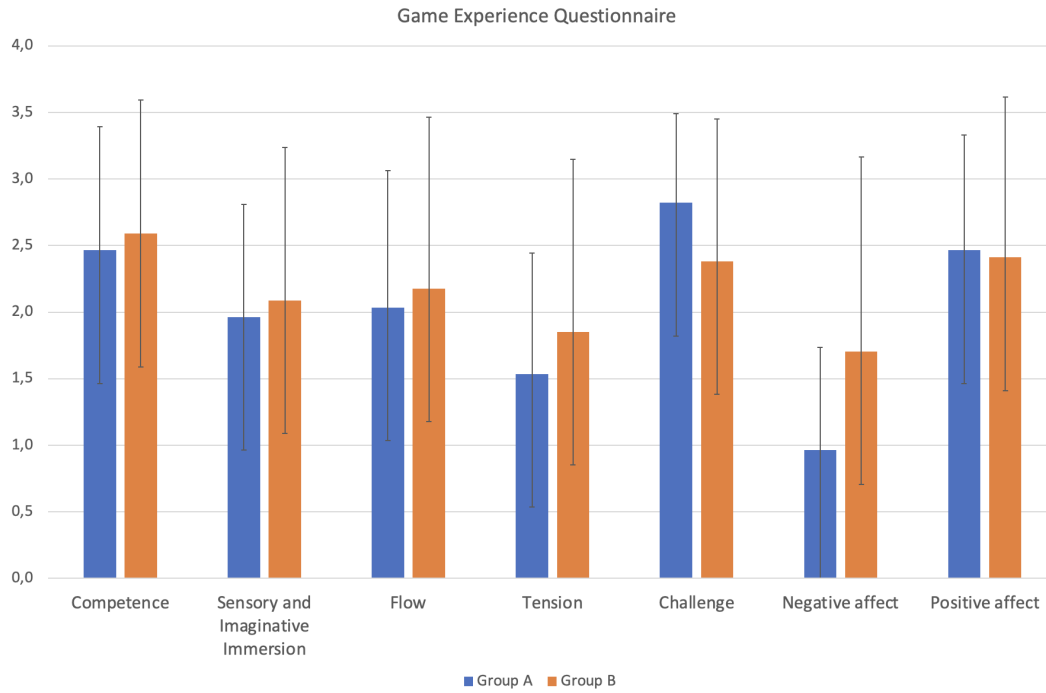


Figure 5.3: Comparison of the Game Experience Questionnaire results of the two test groups.

Table 5.7: Breakdown of the results of the Game Experience Questionnaire, showing the mean, standard deviations, and the statistical significance with  $\alpha = .05$ .

	Group A		Group B		$t(29)$	$p$	Cohens's $d$
	$M$	$SD$	$M$	$SD$			
Competence	2.46	0.93	2.59	1.00	-0.35	.726	-0.13
Sensory and Imaginative Immersion	1.96	0.84	2.09	1.15	-0.33	.740	-0.12
Flow	2.04	1.03	2.18	1.29	-0.33	.742	-0.12
Tension	1.54	0.91	1.85	1.30	-0.77	.446	-0.28
Challenge	2.82	0.67	2.38	1.07	1.34	.192	0.48
Negative affect	0.96	0.77	1.71	1.46	-1.71	.098	-0.62
Positive affect	2.46	0.87	2.41	1.20	0.14	.892	0.05

### 5.4.3 Player Activity Statistics

The following statistics (Figure 5.4) show the click activity of users in the game with the standard deviation. Since Group A did not have a marketplace, the value here was zero. Otherwise, it can be said that the activity was higher in the group without the trading function. Collection and Garage were important tabs that must be visited to complete the quests and thus the study. Tab Events and Profile did not appear in the quests and were therefore visited less.

In order not to distort the click statistics, the data set was also cleaned of insufficient data. This was done by removing data sets with a total test duration of less than 250 seconds from the evaluation. This only affected four participants who had an average playing time of just 2.5 minutes and did not complete a single quest.

The results of this evaluation must also be classified as not statistically significant. A detailed list of results is shown in Table 5.8.

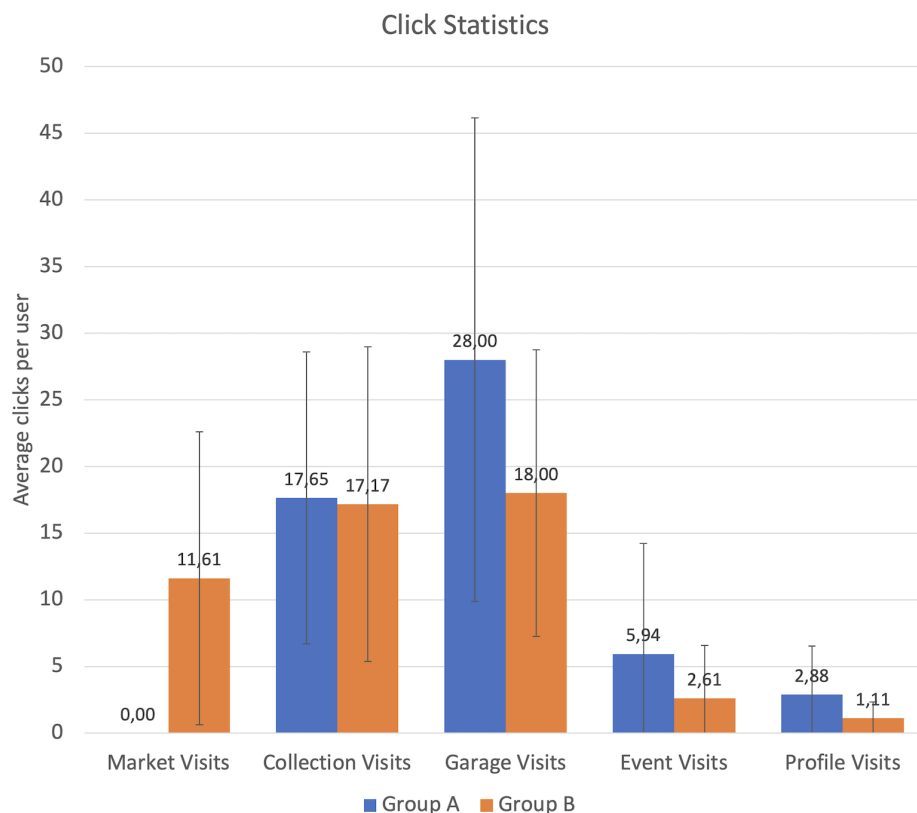


Figure 5.4: This figure shows the click statistics of the UI tabs in the game. The clicks were counted when clicking on the corresponding tabs.

Table 5.8: Detailed display of user activity statistics including mean, standard deviations, and the statistical significance with  $\alpha = .05$ .

	Group A		Group B		$t(33)$	$p$	Cohens's $d$
	$M$	$SD$	$M$	$SD$			
Market Visits	0.00	0.00	11.61	10.98	-4.36	< .001	-1.47
Collection Visits	17.65	10.94	17.17	11.80	0.13	.901	0.04
Garage Visits	28.00	18.14	18.00	10.75	2.00	.054	0.67
Event Visits	5.94	8.30	2.61	3.97	1.53	.136	0.52
Profile Visits	2.88	3.66	1.11	1.23	1.94	.060	0.65

#### 5.4.4 Player Progress Statistics

The player progress statistic shows the player activity, measuring progression and playtime. Different values were summarized in Table 5.9 and compared between Group A and Group B. It appeared that there might have been less user engagement in Group B than in Group A. It should be noted, however, that this is an average and contains significant outliers. The player with the most playing time spent over 140 minutes in the game and covered over 187 kilometers on the highway. 53 percent of players exceeded the estimated 15 minutes of test duration. 28 percent of players even spent more than twice as much time in the game. 18 percent even three times as much, 10 percent four times as much. It is noticeable that fewer vehicles were collected in Group B with the trading function. As a reminder: In Group A, the players got a new vehicle after each race, while in Group B, the players had to manually trade to gather vehicles. Furthermore, in Group A without the trading function, no vehicles could be traded, which is why the value was zero.

Table 5.9: Comparison of the player progress statistics of the two test groups, showing the mean, standard deviations, and the statistical significance with  $\alpha = .05$ .

	Group A		Group B		$t(33)$	$p$	Cohens's $d$
	$M$	$SD$	$M$	$SD$			
Distance (km)	55.71	60.99	26.50	25.39	1.87	.070	0.63
Play Time (min)	43.21	40.22	23.81	13.42	1.93	.061	0.66
Races	27.53	26.43	14.44	10.01	1.96	.058	0.66
Finished Runs	2.24	2.02	1.50	1.42	1.25	.219	0.42
Cars Collected	11.53	6.35	4.94	3.42	3.85	.001	1.30
Cars Sold	0	0	1.61	3.09	-2.15	.039	-0.73

Due to the large standard deviation, a definitive statement about the trading mechanic was not possible. Similar to the results of the click statistic, this evaluation must also be classified as not statistically significant too. Further details are in Table 5.9.

In the player progress statistics, there was a statistically significant result and this referred to 'Cars Collected' ( $t(33) = 3.851, p = .001, \alpha = .05$ , Cohen's  $d = 1.30$ ). However, it has to be kept in mind that collecting cars was a passive mechanism in Group A, but requires active swapping in Group B.

As with the previous player activity evaluation, the 'Cars Sold' was also significant ( $t(33) = -2.15$ ,  $p = .039$ ,  $\alpha = .05$ , Cohen's  $d = -0.73$ ) here due to the deactivated marketplace in one test group.

## 5.5 Discussion

The outcome of both questionnaires was very positive. High scores were achieved in System Usability, and the positive results also predominated in the Game Experience questionnaire. The research goals of planning and developing a game with trading features, high usability, and great game experience, were achieved. This was confirmed by the System Usability Score and the Game Experience Questionnaire.

In general, it was surprising that the players spent much additional time in the game. It only takes about 10-15 minutes to complete the quests and fill out the questionnaires carefully, but this time was greatly exceeded by many players.

Further insights were gained by looking at the player activity statistics in combination with the questionnaires. It turned out that players who gave the game good scores in usability and game experience were more likely to play the game much longer. The longest playtime of one user was about 2.3 hours. This is backed up by a positive correlation between increased playtime, enhanced usability, and game experience in the results.

Additionally, based on verbal feedback, it can be concluded that the game was better evaluated by players who in general enjoy playing car racing games. This rules out the possibility that the playtime was longer than necessary due to poor usability or too difficult tasks. Instead, it is more plausible that the extended playtime was a reflection of a positive user experience and an effective usability design.

At first glance, the usability score was higher in the group with the trading function, but the  $t$ -test showed that the difference was not significant. The same applies to the game experience and user activity. The initial research goals to review if such a trading feature makes sense can now be answered. The difference between the trading mechanic and the gathering mechanism through game progression is not statistically significant. Due to the unproblematic implementation effort and the high usability, the cost-benefit ratio can still be right and it can be implemented in a game. The real effects can then be measured in a large-scale A/B study.

## 6 Lessons Learned

This chapter deals with the insights gained during this work. These are divided into literature research, design and implementation, and evaluation.

### 6.1 Literature Research

It was surprising that the topic of trading, especially one-to-one swapping (i.e. not via a marketplace) in video games, has hardly played a role in the literature so far. However, a lot of insights could be gained about trading in general. On the other hand, the topic of collecting digital items is researched very well. The psychological backgrounds and types of collectors were of particular interest. Many other insights were also gained on how to design good gameplay and a realistic racing simulation.

The variety of racing games, which were all positioned slightly differently and filled a gap in the market, is particularly remarkable. The research into the rise and fall of game series and the shift in target groups due to changes in the balance between arcade games and simulations was very exciting. It was also exciting to see how eagerly the racing game community discusses changes made by game manufacturers and how many suggestions for improvements are made by the community. It was surprising how players' ideas of what a good game should be, differ from current games.

### 6.2 Design and Implementation

The design and implementation was a highly iterative process, as many ideas were implemented on a trial and error basis. Different approaches were implemented and tested for the visual design, user interface design, and also algorithms for the game mechanics. The goal was to develop the game to a certain level of market readiness so that when testing it, players have the feeling that they are looking at a fully-fledged game and not just a prototype. A lot of time went into this development, which could have been shorter if the concept had been clearly defined at the beginning. The user interface in



particular has been redesigned many times, due to visual improvements or the need for new features that have been added. Ultimately, however, this iterative process greatly increased the quality of the software.

To equip the game with a very good driving script, numerous free driving scripts and also a paid one for Unity and Unreal were tested. Surprisingly, none of the tested scripts could meet the requirements. So the choice was made to use the Vehicle Physics Pro driving script from Edy (2022), which is only free in the community version and only available for Unity.

The game engine Unity turned out to be the right choice for the project and the objectives could be quickly implemented. The workflow between Blender and Unity was also very problem-free and features like the post-processing stack were very suitable for quickly achieving beautiful visual effects. The simple creation of builds was already expected due to the author's previous experience, but it was realized that builds for Linux were also very easy to implement.

A LimeSurvey integration was considered but later discarded due to the effort required to familiarize with the tool. The effort was considered greater than a quick implementation in the already familiar Unity user interface system. Implementing the questionnaires in the game engine paid off and was done very quickly. However, reading the data from the database and transferring it to an Excel file for subsequent evaluation had to be done manually due to the quick implementation, which represented a small additional effort.

The implementation of swapping between players with online registration and accounts in the cloud, which was initially implemented using the SQL database, has turned out to be a very extensive topic. Increased development effort is to be expected here to bring the feature to product maturity.

In summary, the implementation phase was very pleasant and fun, despite the many different features and interactive development including some revisions.

### 6.3 Evaluation

In this master thesis, the quest system proved to be a robust guidance system that guided the testers through the study. The game was very well received by the testers with high scores in usability and game experience. The results of the study did not show major differences between the group with trading mechanics and the group with no trading mechanics. The different workloads of the tasks depending on the group could be responsible for

these results, which is why further studies could focus on more evenly distributed workloads.

## 7 Future Work

This chapter offers a few approaches and ideas for further research and implementation improvements. It offers a direction for researchers and developers interested in this topic.

### 7.1 Research

Further research could look at the impact of trading features in games with a larger test group, one-to-one swapping between real players, and whether the impact on the game experience is more significant there and how the feature is received by players in general. This could again be done with a quantitative analysis.

Additionally, qualitative analysis through surveys and interviews could be used to understand players' perceptions and experiences with the trading features, providing insights into how well the feature is received by the players and what improvements could be made.

Furthermore, it could be examined how the rarity of vehicles influences their desirability, i.e. whether rare vehicles are more desirable simply because they are rare or whether this factor has little or no influence on desire. It could be investigated whether players feel happy when receiving particularly rare vehicles.

One could also examine how swapping between two players affects the social component. Swap appointments could take place via chats and social interaction could be measured. It could be investigated whether this social component strengthens the community around the game and thus contributes to the success of the game.

### 7.2 Implementation

The implementation meets all set goals and already achieves a high level of user satisfaction. However, some points can still be improved. The biggest

point here is the introduction of vehicle swaps between real players. This would require the implementation of user accounts including all the security measures that come with it. In addition to real user trading, social features could be integrated such as a chat function or a platform on which players can communicate and exchange ideas and suggestions.

The game could be made more beginner-friendly by starting with less powerful vehicles to get the players used to the driving physics and reduce the frustration caused by the high level of difficulty at the beginning. To make the game a little easier, the vehicle settings can also be adjusted, which are currently designed for maximum speeds. Possible optimizations could be achieved, for example, through an improved gear ratio to better transfer the vehicle's performance to the road.

A further possible improvement for the game is the implementation of the planned upgrade system. Through upgrades, the vehicle's performance could be improved and tailored to different event requirements, which gives the game more complexity. In addition to the cars, upgrades, for example, racing tires or suspension parts, could also be offered for exchange on the marketplace. This is particularly useful if upgrades drop with a random probability and then can be easily combined and completed by swapping.

One might improve the traffic AI to reduce the number of accidents involving AI vehicles to ensure a realistic simulation. Currently, cars always spawn at 0 km/h, which makes heavy trucks in particular susceptible to rear-end collisions. A possible solution would be to spawn at driving speed. The vehicles could also be equipped with reinforcement logic to enable the vehicles to navigate independently and thus represent traffic more realistically. This would significantly increase the quality and complexity of the gaming experience.

The intended virtual reality support could be implemented using a plugin in Unity. This would mainly require making minor adjustments to the menu. The cockpit camera can already be freely rotated using the mouse and should work out-of-the-box with virtual reality devices.

A true multiplayer feature could also be implemented, where two or more drivers are on the same track. This feature would enrich the game with another social aspect.

Ultimately, content such as new landscapes and vehicles could be added to round off the gaming experience. Additional content can also be created by adding new achievements and events. New game modes like time trials and races against opponents on a racetrack have already been implemented in basic terms, but still need to be optimized and integrated using an appropriate user interface. Alternatively, additional game modes could be

created, for example by measuring the average speed instead of the top speed as in the current implementation.

### 7.3 Evaluation

In future studies on this topic, care should also be taken to ensure that both test groups in an A/B study have the same workload. In this way, distortions in the test results due to an imbalance in the workload between the different collection mechanisms could be reduced. This could be achieved, for example, by purchasing vehicles in a shop using earned in-game currency, rather than simply obtaining them through game progress. In this way, passive collecting is transformed into an active activity.

## 8 Conclusion

Despite the lack of literature in the area of trading as a game mechanic in video games, basic research was conducted by using literature in the area of collecting and trading in general. This made it possible to explore the psychological background behind the urge to collect. The advantages and disadvantages of such a trading function were then discussed. It was found out, that swapping is very often wanted by the players and was also implemented by the players in a roundabout way, even if the function was not intended by the developers. Numerous disadvantages of a trading function have also been shown, which can, however, be greatly reduced with the right measures and limitations.

The literature review also dealt with the fundamental aspects of game design, such as level design, upgrades, farming, leaderboards, seasons, and quests, and was primarily concerned with important game mechanics in general, the fundamental decision between 2D or 3D games, and relevance of good user interface designs. Valuable basics for later implementation were collected here.

Upon closer examination of racing games, it has also been discovered that the balance between arcade and racing simulation is a much-discussed topic in the industry, and the decisions made by game developers are controversial for some players. The weak effects used to represent speed are particularly criticized because, according to some critics, they do not reflect the feeling of speed in real car races.

In the design chapter, a storyboard and the most relevant mechanics were designed to test the collection and trading mechanics based on this. This should avoid that the results of the study are distorted by poor basic game mechanics and that users develop frustration before they even start collecting and trading. All problems identified in the background chapter were taken into account in the design and solutions were developed to circumvent the problems.

The implementation was carefully developed in several phases to provide a good gaming experience on the one hand and to further stimulate collecting behavior through visual aspects as chosen value. During the implementation, features such as the racetrack and upgrades were implemented to some

extent but were ultimately deactivated for the final release. This meant that the focus on collecting and trading could be narrowed down. Since trading with real players in the small test group posed too many hurdles, a market maker was used instead to emulate trading. The System Usability Score underlined the success of the implementation with a very good rating. The Game Experience Questionnaire also suggested a good implementation of the game mechanics for a prototype.

From the A/B study and the user data of the two test groups, it could be concluded that the trading function had a rather small impact on the game. The two test groups hardly differed in the values achieved and contained a very high standard deviation and were therefore not significant. The primary game mechanics - in this case car racing and collecting - continued to be in the foreground.

Overall, the research goals defined in the beginning were achieved: The literature research revealed new insights into collecting and trading and, with the research into game mechanics and racing games, provided a basis for the design and implementation. The goals of designing and implementing the game were also achieved and the success was confirmed using the questionnaires. The A/B study carried out showed that the trading feature has little influence on the gameplay, which means that the aim of examining the influence was successful. The final question as to whether such a trading feature should be integrated into a game can be answered based on the research results. It can be implemented if the game developers want to give the game another facet. The actual effects then have to be checked again with many users.

Further research is necessary to highlight the nuances of the influence of a trading function. Whether the trading function between users increases user engagement or improves the game experience, remains open. The current implementation provides a good basis for further research. In the future, the server functionality could be expanded to test the trading function not only against the computer but also against other players in a multiplayer mode to draw refined conclusions. For example, by adding a chat function, the interaction between players could be increased and, thus, user engagement.

Finally, the work might form a cornerstone in the research of trading functions in video games, which can be used as a basis for in-depth work that better examines the processes and influences. This should contribute to the development of high-quality entertaining games, with a potential positive impact also on the quality of the gamified education sector in the future.

# Bibliography

- Adams, E., & Dormans, J. (2012). Game Mechanics: Advanced Game Design, pp. 1–22 (cit. on p. 7).
- Ahmad, N. B., Barakji, S. A. R., Shahada, T. M. A., & Anabtawi, Z. A. (2017). How to launch a successful video game: A framework. *Entertainment Computing*, 23, 1–11. <https://doi.org/https://doi.org/10.1016/j.entcom.2017.08.001> (cit. on p. 8)
- Aleem, S., Capretz, L. F., & Ahmed, F. (2016). Critical Success Factors to Improve the Game Development Process from a Developer’s Perspective. *Journal of Computer Science and Technology*, 31(5), 925–950. <https://doi.org/10.1007/s11390-016-1673-z> (cit. on p. 9)
- Asadi, A.-r., & Hemadi, R. (2018). Understanding Virtual Currencies in Video Games: A Review. *2018 2nd National and 1st International Digital Games Research Conference: Trends, Technologies, and Applications (DGRC)*, 109–117. <https://doi.org/10.1109/DGRC.2018.8712047> (cit. on p. 7)
- Bahaweres, R., Pratama, A., & Sitohang, B. (2014). Implementation of physics simulation in serious game, 1–6. <https://doi.org/10.1109/ICTSS.2014.7013157> (cit. on p. 16)
- Bangor, A., Kortum, P., & Miller, J. (2009). Determining what individual SUS scores mean: Adding an adjective rating scale. *Journal of usability studies*, 4(3), 114–123 (cit. on pp. xii, 88).
- Beckman, B. (1998). *Physics of racing series*. Stuttgart West. <https://books.google.at/books?id=VZO6jwEACAAJ>. (Cit. on p. 16)
- Belk, R., Wallendorf, M., Sherry, J., & Holbrook, M. (1991). Collecting in a consumer culture, pp. 178–215 (cit. on p. 3).
- Bergeron, L. (2010). Improving Random Loot Drops. *Game Developer*. <https://www.gamedeveloper.com/design/improving-random-loot-drops> (cit. on p. 11)
- Blender Foundation. (2021). Blender. <https://www.blender.org/> (cit. on pp. 14, 38)
- Boller, S. (2013). Learning Game Design: Game Mechanics. <http://www.theknowledgeguru.com/learning-game-design-mechanics/> (cit. on p. 9)
- Bourg, D. M., & Bywalec, B. (2013). *Physics for Game Developers: Science, math, and code for realistic effects*. " O’Reilly Media, Inc." (Cit. on p. 17).



- Brooke, J. (1995). SUS: A quick and dirty usability scale. *Usability Eval. Ind.*, 189 (cit. on p. 85).
- Butler, C. (2013). The Effect of Leaderboard Ranking on Players' Perception of Gaming Fun, 129–136. [https://doi.org/10.1007/978-3-642-39371-6\\_15](https://doi.org/10.1007/978-3-642-39371-6_15) (cit. on p. 11)
- Danet, B., & Katriel, T. (1989). No two alike: Play and aesthetics in collecting. *Play & Culture*, 2, 253–277 (cit. on p. 4).
- Dey, D., & Lahiri, A. (2013). The Zero-Day DLC Strategy: A Case for Versioning to Facilitate Product Sampling. *SSRN Scholarly Paper* (cit. on p. 11).
- Dillon, A. (2003). User Interface Design. London: Macmillan. <http://hdl.handle.net/10150/105299>. (Cit. on p. 14)
- Doppler, C. (1842). Über das farbige licht der doppelsterne und einiger anderer gestirne des himmels. *Abhandlungen der königlichen böhmischen Gesellschaft der Wissenschaften*, 2, 465–482 (cit. on p. 19).
- Edy. (2011). Edy's Vehicle Physics. <https://www.edy.es/dev/vehicle-physics/> (cit. on p. 17)
- Edy. (2022). Vehicle Physics Pro. <https://vehiclephysics.com> (cit. on pp. 17, 28, 41, 96)
- Epic Games. (2021). *Unreal Engine* (Version 4.27). <https://www.unrealengine.com>. (Cit. on pp. 13, 28)
- Erke, S., Bin, D., Yiming, N., Qi, Z., Liang, X., & Dawei, Z. (2020). An improved A-Star based path planning algorithm for autonomous land vehicles. *International Journal of Advanced Robotic Systems*, 17(5), 1729881420962263. <https://doi.org/10.1177/1729881420962263> (cit. on p. 49)
- Fabricatore, C. (2007). Gameplay and game mechanics design: A key to quality in videogames. <https://doi.org/10.13140/RG.2.1.1125.4167> (cit. on pp. 9, 32)
- Game Press. (2018). Trading for IVs and Lucky Pokemon. *Pokemon GO Wiki - GamePress*. <https://gamepress.gg/pokemongo/trading-ivs-and-lucky-pokemon> (cit. on p. 7)
- Gosling, S. D., Rentfrow, P. J., & Swann, W. B. (2003). A very brief measure of the big-five personality domains, pp. 504–528 (cit. on p. 4).
- Gran Turismo Wiki. (2022). Defunct Online Features, Gameplay, Trade. <https://gran-turismo.fandom.com/wiki/Trade> (cit. on p. 1)
- Guo, Y., & Barnes, S. (2007). Why People Buy Virtual Items in Virtual Worlds with Real Money. *ACM SIGMIS Database*, 38, 69–76. <https://doi.org/10.1145/1314234.1314247> (cit. on p. 7)
- Hamari, J., & Lehdonvirta, V. (2010). How Game Mechanics Create Demand for Virtual Goods (2010). 5, 14–29. <https://ssrn.com/abstract=1443907> (cit. on p. 7)

- Hamari, J., & Tuunanen, J. (2014). Player types: A meta-synthesis. *Transactions of the Digital Games Research Association*, 1(2), 29–53 (cit. on p. 6).
- Hart, P. E., Nilsson, N. J., & Raphael, B. (1968). A formal basis for the heuristic determination of minimum cost paths. *IEEE transactions on Systems Science and Cybernetics*, 4(2), 100–107 (cit. on p. 49).
- Hecker, C. (2000). Physics in computer games (title only). *Communications of the ACM*, 43(7), 34–39 (cit. on p. 17).
- Hegevall, P. (2022). Racing Dreams: The world’s biggest racing disappointments. *Gamereactor*. <https://www.gamereactor.eu/racing-dreams-the-worlds-biggest-racing-disappointments/> (cit. on p. 21)
- Holmberg, M., & Modée, A. (2021). Random rewards in video games and their impact on player engagement. (Cit. on p. 11).
- Hoober, S., & Berkman, E. (2011). *Designing Mobile Interfaces*. O’Reilly Media, Inc. (Cit. on p. 15).
- Hunicke, R., Leblanc, M., & Zubek, R. (2004). MDA: A Formal Approach to Game Design and Game Research. *AAAI Workshop - Technical Report*, 1 (cit. on p. 9).
- Ijsselsteijn, W. A., de Kort, Y. A. W., & Poels, K. (2013). The Game Experience Questionnaire. *Technische Universiteit Eindhoven* (cit. on p. 85).
- Joseph, D. (2021). Battle pass capitalism. *Journal of Consumer Culture*, 21(1), 68–83 (cit. on p. 12).
- KBV Research. (2022). Global Game Engines Market Size, Share & Industry Trends Analysis Report By Type (3D Game Engines, 2D Game Engines and Others), By Component (Solution and Services), By Platform (Mobile, Console, Computer and Others), By Regional Outlook and Forecast, 2022 - 2028, 179. <https://www.kbvresearch.com/game-engines-market/> (cit. on p. 13)
- Keene, S. (1998). *Digital Collections, Museums and the Information Age*. Routledge. (Cit. on p. 3).
- Koh, E., & Kerne, A. (2006). I keep collecting: College students build and utilize collections in spite of breakdowns. *Proceedings of the 10th European Conference on Research and Advanced Technology for Digital Libraries (ECDL’06)*, 303–314. [https://doi.org/http://dx.doi.org/10.1007/11863878\\_26](https://doi.org/http://dx.doi.org/10.1007/11863878_26) (cit. on p. 3)
- Koster, R. (2013). *Theory of fun for game design*. " O’Reilly Media, Inc." (Cit. on p. 20).
- Kristiadi, D. P., Udjaja, Y., Supangat, B., Prameswara, R. Y., Warnars, H. L. H. S., Heryadi, Y., & Kusakunniran, W. (2017). The effect of UI, UX and GX on video games. *2017 IEEE International Conference on Cybernetics and Computational Intelligence (CyberneticsCom)*, 158–163 (cit. on p. 14).
- Lebowitz, J., & Klug, C. (2011). *Interactive Storytelling for Video Games: A Player-Centered Approach to Creating Memorable Characters and Stories*. Elsevier/Focal Press. (Cit. on p. 10).

- Lehdonvirta, V. (2010). Real-Money Trade of Virtual Assets: New Strategies for Virtual World Operators (2008). 5, 113–137. <https://ssrn.com/abstract=1351782> (cit. on p. 7)
- Li, Y., Ryan, C. T., & Sheng, L. (2021). Optimal sequencing in single-player games. <https://doi.org/10.2139/ssrn.3883557>. (Cit. on p. 10)
- Lidestam, B., Eriksson, L., & Eriksson, O. (2019). Speed perception affected by field of view: Energy-based versus rhythm-based processing. *Transportation Research Part F: Traffic Psychology and Behaviour*, 65, 227–241. <https://doi.org/10.1016/j.trf.2019.07.016> (cit. on p. 18)
- Livingston, I. J., Gutwin, C., Mandryk, R. L., & Birk, M. (2014). How players value their characters in world of warcraft. *Proceedings of the 17th ACM Conference on Computer Supported Cooperative Work & Social Computing*, 1333–1343. <https://doi.org/10.1145/2531602.2531661> (cit. on p. 5)
- Lomas, J. D., Koedinger, K., Patel, N., Shodhan, S., Poonwala, N., & Forlizzi, J. L. (2017). Is Difficulty Overrated? The Effects of Choice, Novelty and Suspense on Intrinsic Motivation in Educational Games. *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*, 1028–1039. <https://doi.org/10.1145/3025453.3025638> (cit. on p. 20)
- Manninen, T., & Kujanpää, T. (2007). The Value of Virtual Assets - The Role of Game Characters in MMOGs. *International Journal of Business Science and Applied Management*, 2(1), 21–33 (cit. on p. 5).
- McKinley, M. B. (2007). The psychology of collecting. *The National Psychologist* (cit. on pp. 3, 4).
- Monster, M. (2003). Car physics for games. Retrieved September, 1, 2009 (cit. on p. 16).
- MuYe. (2021). Why Racing Games Feel Slow. <https://www.youtube.com/watch?v=e9OriySqLjQ> (cit. on p. 18)
- Navarro, F., Serón, F. J., & Gutierrez, D. (2011). Motion Blur Rendering: State of the Art. *Computer Graphics Forum*. <https://doi.org/10.1111/j.1467-8659.2010.01840.x> (cit. on p. 19)
- Niantic. (2022). Trading Pokémon. <https://niantic.helpshift.com/hc/en/6-pokemon-go/faq/96-trading-pokemon/> (cit. on p. 1)
- Oh, G., & Ryu, T. (2007). Game Design on Item-selling Based Payment Model in Korean Online Games. *DiGRA Conference*, 650–657. <http://www.digra.org/wp-content/uploads/digital-library/07312.20080.pdf> (cit. on p. 8)
- Olivetti, J. (2011). The dangers and allure of real money trading in MMOs. <https://www.engadget.com/2011-09-03-the-dangers-and-allure-of-real-money-trading-in-mmos.html> (cit. on p. 8)
- Oracle Corporation. (2023). MySQL. <https://www.mysql.com>. (Cit. on p. 32)
- Pearce, S. M. (1994a). Collecting reconsidered. In S. M. Pearce (Ed.), *Interpreting objects and collections* (pp. 193–204). Routledge. (Cit. on p. 4).

- Pearce, S. M. (1994b). The urge to collect. In S. M. Pearce (Ed.), *Interpreting objects and collections* (pp. 157–159). Routledge. (Cit. on p. 4).
- Petrovskaya, E., & Zendle, D. (2020). The battle pass: A mixed-methods investigation into a growing type of video game monetisation. *OSF Preprints, Sep* (cit. on p. 12).
- Pretto, P., Ogier, M., Bühlhoff, H. H., & Bresciani, J.-P. (2009). Influence of the size of the field of view on motion perception. *Computers & Graphics*, 33(2), 139–146 (cit. on p. 18).
- Rouse, M. (2017). What is farming? - definition from techopedia. *Techopedia*. <https://www.techopedia.com/definition/19278/farming> (cit. on p. 10)
- Rykwert, J. (2001). Why collect? Joseph Rykwert considers what has led people through the ages to make collections, sometimes of the most unlikely objects, and discusses the value of their activities. *History Today*, 51(12), 32–38 (cit. on p. 4).
- Salen, K., & Zimmerman, E. (2004). *Rules of play: Game design fundamentals*. MIT Press. (Cit. on p. 9).
- Sexton, J. (2007). *Music, Sound and Multimedia: From the Live to the Virtual*. Edinburgh University Press. (Cit. on p. 19).
- Siemens, S., Plass-Fleßenkämper, B., & Beyer-Fistrich, M. (2021). Die Evolution des Rennspiel-Genres: Von den 1970ern bis heute. *PC Games*. <https://www.pcgames.de/Panorama-Thema-233992/Specials/die-besten-Rennspiele-und-ihre-Geschichte-von-den-70-er-Jahren-bis-heute-1378118/> (cit. on p. 1)
- Smith, G., Anderson, R., Kopleck, B., Lindblad, Z., Scott, L., Wardell, A., Whitehead, J., & Mateas, M. (2011). Situating Quests: Design Patterns for Quest and Level Design in Role-Playing Games. In M. Si, D. Thue, E. André, J. C. Lester, T. J. Tanenbaum, & V. Zammitto (Eds.), *Interactive storytelling* (pp. 326–329). Springer Berlin Heidelberg. (Cit. on p. 13).
- Snay, A. (2021). Impact of real-world trading into online video games (cit. on pp. 7, 8).
- Snow, S., McMahon, B., McKenzie, S., Radke, K., Verlaet, I., & Buys, L. (2015). Designing for Collections: Building Histories, Sharing the Spectacle. *Proceedings of the Annual Meeting of the Australian Special Interest Group for Computer Human Interaction*, 299–303. <https://doi.org/10.1145/2838739.2838798> (cit. on p. 3)
- Sobaihi, R. (2017). Road Generator for Unity3d. <https://github.com/Redouane64/Road-Generator-Unity3d> (cit. on pp. x, 45, 47)
- Su, Y. (2018). The application of 3d technology in video games. *Journal of Physics: Conference Series*, 1087(6), 062024. <https://doi.org/10.1088/1742-6596/1087/6/062024> (cit. on p. 13)

- The phpMyAdmin Project. (2023). phpMyAdmin. <https://www.phpmyadmin.net> (cit. on p. 72)
- TimePlayer. (2023). Why Forza Horizon 5 feels boring. <https://www.youtube.com/watch?v=7mNh8wZU2Go> (cit. on p. 20)
- Toups Dugas, P. O., Crenshaw, N. K., Wehbe, R. R., Tondello, G. F., & Nacke, L. E. (2016). "The Collecting Itself Feels Good": Towards Collection Interfaces for Digital Game Objects. *Proceedings of the 2016 Annual Symposium on Computer-Human Interaction in Play*, 276–290. <https://doi.org/10.1145/2967934.2968088> (cit. on p. 5)
- Unity Technologies. (2021). Unity. <https://www.unity.com/> (cit. on pp. 13, 28, 38)
- Urschel, A. (2011). Understanding real money trading in MMORPGs. *New World Disorder* (cit. on p. 8).
- Walk, W., Görlich, D., & Barrett, M. (2017). Design, Dynamics, Experience (DDE): An Advancement of the MDA Framework for Game Design. [https://doi.org/10.1007/978-3-319-53088-8\\_3](https://doi.org/10.1007/978-3-319-53088-8_3). (Cit. on p. 9)
- Watkins, R. D., Sellen, A., & Lindley, S. E. (2015). Digital collections and digital collecting practices. *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*, 3423–3432. <https://doi.org/10.1145/2702123.2702380> (cit. on p. 5)
- Watson, D., Valtchanov, D., Hancock, M., & Mandryk, R. (2014). Designing a gameful system to support the collection, curation, exploration, and sharing of sports memorabilia. *Proceedings of the First ACM SIGCHI Annual Symposium on Computer-Human Interaction in Play*, 451–452. <https://doi.org/10.1145/2658537.2661322> (cit. on p. 4)
- Winn, B. M. (2009). The Design, Play, and Experience Framework. <https://api.semanticscholar.org/CorpusID:17093595> (cit. on p. 9)
- Yee, N. (2006). Motivations for play in online games. *Cyberpsychology & Behavior*, 9(6), 772–775. <https://doi.org/10.1089/cpb.2006.9.772> (cit. on p. 5)
- Yin, M., & Xiao, R. (2022). The reward for luck: Understanding the effect of random reward mechanisms in video games on player experience. *Proceedings of the 2022 CHI Conference on Human Factors in Computing Systems*, 1–14 (cit. on p. 11).
- Zehnder, S. M., & Lipscomb, S. D. (2006). *Playing Video Games: Chapter 17: The Role of Music in Video Games* (Vol. 1). Routledge. (Cit. on p. 19).
- Zhang, W. (2014). Real World Trading and MMOs: Positive Impact. <https://www.freemmostation.com/features/real-world-trading-mmos-positive-impact/> (cit. on p. 8)

# Ludography

- CCP Games. (2003). EVE Online [game; CCP Games, Atari SA, Reykjavík, Island]. <https://www.eveonline.com/>. (Cit. on p. 23)
- Codemasters. (2008). Race Driver: GRID [game; Codemasters, Southam, England]. <https://www.ea.com/games/grid>. (Cit. on p. 21)
- Criterion Games. (2010). Need for Speed: Hot Pursuit [game; Electronic Arts, Guildford, UK]. <https://www.ea.com/de-de/games/need-for-speed/need-for-speed-hot-pursuit-remastered>. (Cit. on p. 21)
- EA Black Box. (2007). Need for Speed: Pro Street [game; Electronic Arts, Burnaby, British Columbia, Canada]. (Cit. on p. 21).
- EA Black Box. (2008). Need for Speed: Undercover [game; Electronic Arts, Burnaby, British Columbia, Canada]. (Cit. on p. 21).
- Firemonkeys Studio. (2013). Real Racing 3 [game; Electronic Arts, Melbourne, Australia]. <https://www.ea.com/de-de/games/real-racing/real-racing-3>. (Cit. on p. 22)
- iRacing Motorsport Simulations, LLC. (2008). iRacing [game; Racing.com Motorsport Simulations, LLC, Chelmsford, USA]. <https://www.iracing.com>. (Cit. on p. 22)
- Kunos Simulazioni. (2019). Assetto Corsa Competizione [game; Kunos Simulazioni, Campagnano di Roma, Italy]. <https://assettocorsa.gg/assetto-corsa-competizione/>. (Cit. on p. 22)
- Niantic. (2016). Pokémon Go [game; Niantic, Inc., San Francisco, California, U.S.]. <https://pokemongolive.com>. (Cit. on pp. 1, 7, 22)
- Playground Games. (2021). Forza Horizon 5 [game; Microsoft, Rossmore House, 9 Newbold Terrace, Leamington Spa CV32 4EA, UK]. <https://www.xbox.com/de-AT/games/store/forza-horizon-5/9nnx1vvr3knq>. (Cit. on pp. 20, 23)
- Polyphony Digital. (2010). Gran Turismo 5 [game; Sony Computer Entertainment, Tokyo, Japan]. <https://www.gran-turismo.com/de/products/gt5/>. (Cit. on pp. 1, 21)
- Reiza Studios. (2020). Automobilista 2 [game; Reiza Studios, Maringá, Paraná, Brazil]. <https://www.game-automobilista2.com>. (Cit. on p. 23)
- Riot Games. (2009). League of Legends [game; Riot Games, Los Angeles, California, U.S.]. <https://www.leagueoflegends.com/>. (Cit. on p. 22)

- Rockstar North. (2013). Grand Theft Auto 5 [game; Rockstar North, Edinburgh, Schottland]. <https://www.rockstargames.com/de/gta-v>. (Cit. on p. 23)
- Slightly Mad Studios. (2020). Project Cars 3 [game; Bandai Namco Entertainment, London, UK]. [<https://de.bandainamcoent.eu/project-cars/project-cars-3>]. (Cit. on p. 23)
- Slightly Mad Studios and EA Bright Light. (2009). Need for Speed: Shift [game; Electronic Arts, London, England]. (Cit. on p. 21).
- TrackTwenty. (2014). SimCity: BuildIt [game; Electronic Arts, Helsinki, Finland]. <https://www.ea.com/games/simcity/simcity-buildit>. (Cit. on p. 22)
- Turn 10 Studios. (2023). Forza Motorsport 8 [game; Microsoft, Redmond, USA]. <https://www.xbox.com/de-AT/games/forza-motorsport>. (Cit. on p. 23)