



Recommender systems for players of online video games

Final Degree Project

Òscar Pons Gallart

Director: David Sánchez Carreras

Video games design and development

Centre de la Imatge i la Tecnologia Multimèdia - UPC

2021-2022

Index

Summary	4
Keywords.....	4
Tables' indices.....	5
Figures' indices.....	6
Links' indices	9
Glossary	10
1. Introduction.....	12
1.1. Motivation	13
1.2. Problem statement	13
1.3. General objectives.....	14
1.4. Specific objectives	15
1.5. Project scope	15
2. State of the art.....	16
2.1. Video games.....	16
2.1.1. Online video games	16
2.2. Teammates finding sites.....	17
2.2.1. TeamTavern.....	17
2.2.2. TeamFind	18
2.3. Unity	18
2.4. Recommender systems	19
2.4.1. Collaborative filtering methods.....	19
2.4.2. Content-based methods	24
2.4.3. Collaborative vs content based.....	27
2.4.4. Evaluation of a recommender system.....	27
2.4.5. Hybrid Models and deep learning.....	29
2.4.6. Conclusions	32



3. Project management.....	34
3.1. Procedure and tools	34
3.1.1. GANTT.....	34
3.1.3. Asana	36
3.1.4. GitHub.....	36
3.2. Validation tools	37
3.2.1 Project development validation	37
3.2.2 Project content validation.....	37
3.3. SWOT analysis	37
3.3.1. Strengths.....	37
3.3.2. Weakness	37
3.3.3. Opportunities	38
3.3.4. Threats.....	38
3.4. Risks and contingency plans	38
3.5. Initial costs analysis	39
3.5.1. Human resources	40
3.5.2. Physical resources	41
4. Methodology.....	42
4.1. Working method	42
4.2. Project stages	42
4.2.1. Planning	42
4.2.2. Development.....	43
4.2.3. Closing	44
5. Project development.....	45
5.1. Definition and justification of the approach	45
5.2. Tools	46
5.2.1. Microsoft Word	46
5.2.2. Python.....	46
5.2.3. PyCharm	47
5.2.4. TensorFlow	47
5.2.4. Unity	47
5.2.5. Unity Barracuda library	48
5.2.6. Onnx library	48



5.3. Dataset	49
5.3.1. Populating the dataset.....	50
5.3.2. Defining the rating criteria.....	52
5.4. Recommender system	56
5.4.1. Retrieval.....	57
5.4.2. Ranking.....	64
5.4.3. Cross-ranking.....	68
5.5. Unity integration	77
5.5.1. TensorFlow Lite for Unity.....	77
5.5.2. TensorFlowSharp.....	77
5.5.3. Unity Barracuda.....	78
5.6. Game simulation	79
5.6.1. Find team screen.....	80
5.6.2. Last match summary.....	83
5.7. Problems and solutions	85
5.7.1. Dataset.....	85
5.7.2. Implementation of the recommender system model.....	85
5.7.2. Impact to the project.....	86
6. Conclusions	88
7. Next steps	90
8. References	91

Summary

The content in this project is the approach, exploration, analysis and use of recommender systems to integrate an implementation of one system that learns the players' behavior and recommends them to other players, to show recommender systems as a way of enhancing the player experience.

Keywords

Recommender system, algorithm, tensorflow, unity, connect, players, gamers, machine learning, deep learning, onnx, barracuda.

Tables' indices

Table 1: Project's tasks time estimation.....	35
Table 2: Risk and contingency plans	39
Table 3: Total initial costs analysis table.....	39
Table 4: Human resources costs table	40
Table 5: Physical resources costs table.....	41
Table 6: Dataset gender rating criteria.....	53
Table 7: Dataset character rating criteria.....	54
Table 8: Dataset character type rating criteria	54
Table 9: Results of model training and evaluation	62
Table 10: Training results of ranking stage	67
Table 11: Raw data before the manual pre-processing.....	71
Table 12: Raw data after the manual pre-processing.....	71

Figures' indices

Figure 1: TeamTavern logo	18
Figure 2: TeamFind logo	18
Figure 3: Unity logo.....	19
Figure 4: Representation example of collaborative filtering	20
Figure 5: Illustration of the user-user method	21
Figure 6: Illustration of the item-item method	22
Figure 7: Illustration of the matrix factorization from user-movie.....	23
Figure 8: Illustration of the matrix factorization method	24
Figure 9: Illustration of content-based approach	25
Figure 10: Illustration of the item-centered approach	26
Figure 11: Illustration of the user-centered method	26
Figure 12: Representation of deep learning stages	29
Figure 13: TensorFlow logo	30
Figure 14: LightFM logo	30
Figure 15: Spotlight logo	31
Figure 16: SeldonServer logo	31
Figure 17: Plot of the age distribution of the users.....	51
Figure 18: Plot of the ranking distribution of the users	51
Figure 19: Sample of the users file	51
Figure 20: Sample of the user interactions file	52
Figure 21: Diagram of the retrieval implementation's steps	57

Figure 22: Illustration of a two tower model	58
Figure 23: Diagram of parsing and pre-processing step in retrieval	59
Figure 24: Diagram of data processing step in retrieval.....	59
Figure 25: Diagram of training and evaluation step in retrieval.....	61
Figure 26: Diagram of the predictions and saving step	62
Figure 27: Diagram of the ranking implementation's steps	64
Figure 28: Diagram of parsing and pre-processing step in ranking.....	65
Figure 29: Diagram of data processing step in ranking	65
Figure 30: Diagram of training and evaluation step in ranking	66
Figure 31: Diagram of the implementation of cross-ranking	68
Figure 32: Diagram of following steps after the creation of the cross-ranking model.....	69
Figure 33: Illustration of the cross-ranking model	70
Figure 34: Diagram of data pre-processing in cross-ranked.....	71
Figure 35: Processing diagram of the cross-ranking	72
Figure 36: Diagram of model creation of cross-ranking.....	73
Figure 37: Visualization of internal weight matrix of cross-ranking.....	75
Figure 38: Game simulation - Main menu.....	79
Figure 39: Game simulation - Decorative props data.....	79
Figure 40: Game simulation - Recommender rating data.....	80
Figure 41: Game simulation - Differentiation of online and offline players	80
Figure 42: Game Simulation - Find team screen with game data.....	81
Figure 43: Game simulation - Find team screen with AI data.....	81

Figure 44: Game simulation - Find team screen: show offline user story82

Figure 45: Game simulation - Find team screen: add friend user story82

Figure 46: Game simulation - Last match summary screen with game data83

Figure 47: Game simulation - Last match summary with AI data84



Links' indices

Link 1: Data generation source code	50
Link 2: Recommender system implementation's source code	56

Glossary

Software: generic term used to refer to applications and programs that run on a device, that tell the device how to work.

Social game: a social game may refer to tabletop, other face-to-face indoor or outdoor games, or video games that allow or require social interaction between players as opposed to games played in solitude, games played at tournaments or competitions, or games played for money.

Cooperative game: A cooperative video game is one in which players must work together as a team to defeat one or more non-player character opponents. Other multiplayer modes, such as competitive multiplayer modes like player versus player or deathmatch, are not included.

Recommender system: is a type of information filtering system that attempts to predict a user's rating or preference for an item. A recommendation engine can display items that users might not have thought to search for on their own, but it can also help users find compelling content in a large corpus.

MMO: A massively multiplayer online game (MMOG) is an online video game that has hundreds or thousands of players on the same server.

LAN: A local area network is a computer network that interconnects computers within a limited area such as a residence, school, laboratory, university campus or office building

API: it is the acronym for Application Programming Interface, which is a software intermediary that allows two applications to talk to each other.

Bias: Bias is the tendency of a statistic to overestimate or underestimate a parameter.

Variance: variance measures variability from the average or mean.

Explainability: Model explainability refers to the concept of being able to understand the machine learning model.

Neighborhood: set of close values based on a point of reference

Dot product: Algebraically, the dot product is defined as the sum of the products of the corresponding entries of the two sequences of numbers.

Keras: It is an open-source software library that provides a Python interface for artificial neural networks.

PyTorch: PyTorch is an open-source machine learning framework based on the Torch library, used for applications such as computer vision and natural language processing.

Script: A script or scripting language is a computer language program capable of being executed without being compiled.

IDE: An integrated development environment is a software application that provides comprehensive facilities to computer programmers for software development.

Inference: Machine learning inference is the process of running data points into a machine learning model to calculate an output such as a single numerical score.

1. Introduction

The Internet has been the greatest hit in the last century, people around the world are connected instantaneously and with almost no effort. Like many advances in technology, games have also evolved and taken advantage of this famous invention of the internet, showing the wide and powerful potential of it.

Online games have an important role inside this huge industry and people are getting so used to them that every new launched game is expected to be online, or to have online features that bring people together.

Even with all this willingness to connect people, gamers find it difficult to make friends or game pals online because it requires so much effort to find someone appropriate for them.

This project proposes a way of connecting people with real ease using recommender systems, to let the players, just by looking at a username, see how good they will do playing with other players, or at least if there's a high probability to match in certain common aspects.

This statement is exemplified in two scenarios of online gaming: a lobby to find a team and the end of a match. These screens are two places where all the players are shown with the results and statistics of the game, and it is a relaxed zone for the players to get along between them. The work here is to show the recommender indicator next to the name of each player, so it is shown to the player the amount of similarity with each one. This scenario is represented by an imitation of a game UI made with the *Unity* game engine.

So, the content in this project is the approach, exploration, analysis and use of recommender systems to integrate an implementation of one system that learns the players' behavior and recommends them to other players, to show recommender systems as a way of enhancing the player experience.

1.1. Motivation

There is a lot of research about retrieving players data, clustering their personality traits, and collecting all kinds of information, using a large variety of methods such as artificial intelligence or mathematical models. But almost all of them address the same problem: they want to analyze players to recommend them video games to buy. [\[1\]](#)

The purpose of this project is to use these recommender systems not to sell something to them but to improve the user's experience of any online game, as it was the purpose of the study "*a team based player versus player recommender systems framework for player improvement*" to improve the performance of players [\[2\]](#). Here, the motivation is to ease the task of finding better matches of players with the same tastes, interests, or ambitions inside the same environment.

This is aimed to be an in-application content, letting the user have the full experience inside the same environment, removing the frustration of having to look for external sites, register, and fill some maybe not so necessary fields, just to share that they are looking for someone to play with, or to see a list of players as them.

1.2. Problem statement

The anonymousness and randomness of the internet makes players with the same interests meet, but most of the time, they don't notice, or they don't know how to keep in contact, or even the unlucky ones don't have such pleasure of meeting someone alike.

Teen gamers play games with different types of people: 89% of them play with friends they know in person, 54% play with friends they know only online, and 52% play online with others who are not friends [\[3\]](#). These numbers show the importance of friendships while playing online games, but they are clearly biased towards physical, already known, friends.

Studies say that most players play with their friends, [\[4\]](#) but what happens if their friends don't play the same games as them? Players have to look for teammates in other places such as team finding websites, or find a player in the game that has the same day schedule, or find a player that likes the same character, etc.

It is easy to see that there are too many details to consider finding a good friend to play online and, this fact is so frustrating for the users that they may eventually give up the search, leading them to finally quit and not come back to the game. [\[5\]](#)

This resembles the problem of many other fields on the internet where the user has to make one choice between a lot of possibilities, such as online shopping, watching YouTube videos or just using social media.

For the latest problem, there is already a current solution: recommender systems. There has been a lot of research to improve the recommendations that are given to the users, so they find more appealing the content presented in front of them, so they buy more things, watch more videos, or surf the social media for longer periods of time.

For the player's problem, the solution of recommender systems can be adapted and aimed to recommend players, as if they were the product, or stop at half of the algorithm's logic, where there is already a match between users, to output the recommended user itself instead of the data associated with it.

Recommendation of people on platforms where people are not the main content is not quite common. It is expected to see recommendations of people in social networks such as *Instagram* or *LinkedIn*, but it is not expected to see recommendations of players in a game, however, the data needed to provide this feature is, in most cases, already collected and being used for some other purposes such as recommending in-game purchases.

So, this project is about helping players that are willing to establish connection with other players, by providing a tool that game developers can use to solve this problem, and it is addressed using recommender systems as they have been an effective solution for content recommendation.

1.3. General objectives

The general objective of this project is to provide an implementation of a recommender system that propose a solution to this problem, embedded in a game simulation, to encourage the use of it to enhance the players' experience in terms of creating new relationships inside the game.

1.4. Specific objectives

As specific objectives, this project aims to:

- Provide an implementation of a recommender system that recommends players.
- Show to the players an evaluation of the profile of the other players. This way, players can quickly see the result of a depth analysis of the potential similar players.
- Extract conclusions about using recommender systems to recommend players.

1.5. Project scope

This approach of using recommender systems can be used in different scenarios, as it is seen in a lot of websites where people and new relationships can be possible, though here it is proposed as a part of a game, aimed at players of any age and gender. The scope can be so wide because it is the system who is narrowing the scope, as teens are more recommended to other teens and so on.

The beneficiaries of this project are the players of the games that include this system in it, as it will be provided with an extra feature that will enhance their overall playing experience.

Using this project as guidance will allow game developers to widen their range of vision about creating a solid game community of the game. Developers can decide to include the contents of this work into their game, so players are more interconnected, improving the solidity and range of the community, to stand out and build a good reputation of a game that takes care of its players.

2. State of the art

2.1. Video games

A video game is a computer game that can be played on a computing device such as a personal computer, gaming console, or mobile phone. Video games are classified into two types based on their platform: computer games and console games. However, the advent of social networks, smartphones, and tablets in recent years has given rise to new categories such as mobile and social games. Since the first games were released in the 1970s, video games have come a long way. [\[6\]](#)

The video gaming industry is now estimated to be worth \$178.73 billion in 2021, a 14.4% increase from 2020. This is a significant difference from what was predicted in 2016, which predicted a total worth of \$90.07 Billion for the same period – a 76.8% difference between the two figures demonstrating that growth could accelerate further. [\[7\]](#)

The industry is huge, and so is the community, and it is still growing, and since the eruption of the internet, the game industry has focused on the online and social part of it, bringing to the market more and more online games to increase the interconnection and competitiveness of them, setting a new standard where everybody is comfortable and willing to stay.

2.1.1. Online video games

Just to see the perspective of the huge amount of single player games there are, here it is shown the results of the searches of video games based on categories on one of the major video game stores, *Steam*. *Steam* is a video game digital distribution service by Valve. It was launched as a standalone software client in September 2003 as a way for Valve to provide automatic updates for their games and expanded to include games from third-party publishers. [\[8\]](#)

- Multiplayer cooperative: 11,340 results
- Multiplayer online competitive: 12,700 results
- LAN: 1,796 results
- Local and by group: 13,678 results
- Multiplayer: 27,786 results
- MMO: 2,024 results
- Single player: 93,750 results

Note that all these games are played online, meaning that although the category is single player, they are played in a lobby, match or sandbox with other players, but the main action of the game is aimed to be delivered to individual players. An example of it could be a war first person shooter game: it is played by a single player in a multiplayer environment.

Also note that the categories may be repeated among titles.

These numbers evoke us to the next section, the current solution of the problem stated in this project: team finding sites.

2.2. Teammates finding sites

Players who cannot find or have not found a person to play with and are willing to join a team or to find a partner to team up, usually join pages that are designed for that purpose.

These pages, in short, are a large database of players that have introduced their data in order to appear in the searches of other players that have done the same. So, with more people joining these sites, the higher the probability of finding a teammate.

2.2.1. TeamTavern

TeamTavern, born in 2019, is self-described as an esports team finding platform. Here, team or player profile can be created and then appear on the searches. They have a total of around 8800 entries combining players and teams' definitions.

They provide the basic features anyone would expect in the site: to look for a team and to look for players. Also, it has the option to search between 8 popular online video games.

To look for a team, an account is needed, and for that it is needed to fill a form to describe a little about the player and their gaming habits, choose the desired game and fill in more information about the game profile.

To look for players the process is very similar but instead of defining the gaming habits and profile, they are defined them referring to the team. [\[9\]](#)



Figure 1: TeamTavern logo

2.2.2. TeamFind

TeamFind is a team finding network and it has a large community with over 350.000 gamers, featuring 7 popular online video games. It was established in 2017 and has grown exponentially since launch. It has the same features as their competitors: team and player finding.

To look for both it is not needed to register, but if to interact and meet some new people, an account is needed.

The site is aimed game-wise, leaving more personal data such as gaming habits or schedule out of the record, although some players may find it an important piece of the puzzle of finding a new teammate. [\[10\]](#)



Figure 2: TeamFind logo

2.3. Unity

Unity is a cross-platform game engine developed by Unity Technologies, first announced and released in June 2005 at Apple Inc.'s Worldwide Developers Conference as a Mac OS X-exclusive game engine. The engine has since been gradually extended to support a variety of desktop, mobile, console and virtual reality platforms. It is particularly popular for iOS and Android mobile game development and used for games such as Pokémon Go or Monument Valley. It is considered easy to use for beginner developers and is popular for indie game development.

The engine can be used to create three-dimensional and two-dimensional games, as well as interactive simulations and other experiences. The engine has been adopted by industries outside video gaming, such as film, automotive, architecture, engineering, construction, and the United States Armed Forces. [\[11\]](#)

Unity will be used in this project to simulate an end of match user interface, taking advantage of the powerful yet simple engine, where the results of the recommender system are shown. Displaying the results in a demo is an easy way to see the potential use of the work of this project in a game.



Figure 3: Unity logo

2.4. Recommender systems

With the rise of YouTube, Amazon, Netflix, and other similar web services over the last few decades, recommender systems have become increasingly important in people's lives. Whether it's for e-commerce or online advertising, recommender systems are now an indisputable part of people's daily online lives.

Recommender systems, in a broad sense, are algorithms that aim to suggest relevant items to users (items being movies to watch, text to read, products to buy or anything else depending on industries). In some industries, recommender systems are critical because they can generate a significant amount of revenue or serve as a way to differentiate oneself from competitors.

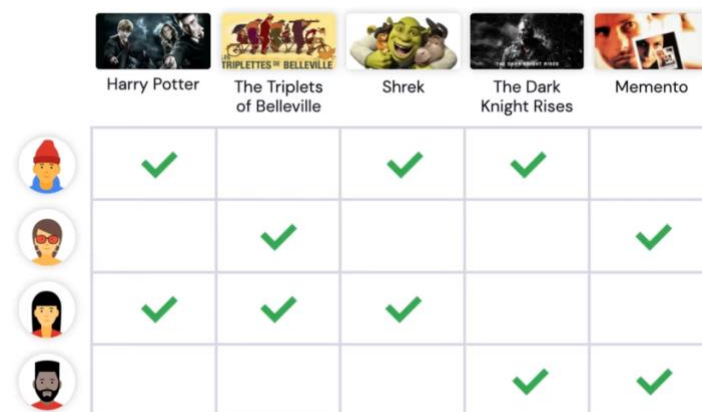
Recommendation engine algorithms are categorized in two kinds: collaborative filtering models and content-based models. They differ by the type of data involved.

2.4.1. Collaborative filtering methods

In order to generate new recommendations, collaborative methods for recommender systems are methods that are solely based on past interactions between users and items.

The main idea of collaborative methods is that past user-item interactions are sufficient for detecting similar users and/or similar items, as well as making predictions based on these estimated proximities.

Collaborative approaches have the advantage of requiring no information about users or items, allowing them to be used in a variety of situations. Furthermore, the more users interact with items, the more accurate new recommendations become: new interactions recorded over time bring new information and make the system more and more effective for a fixed set of users and items.



	Harry Potter	The Triplets of Belleville	Shrek	The Dark Knight Rises	Memento
User 1 (Red hat)	✓		✓	✓	
User 2 (Glasses)		✓			✓
User 3 (Black hair)	✓	✓	✓		
User 4 (Black beard)				✓	✓

Figure 4: Representation example of collaborative filtering

Collaborative filtering, on the other hand, suffers from the "cold start problem," in which it is impossible to recommend anything to new users or to recommend a new item to any users, and many users or items have too few interactions to be efficiently handled. This flaw can be addressed in a variety of strategies:

- Random: recommending random items to new users or new items to new users
- Maximum expectation: recommending popular items to new users or new items to most active users
- Exploration: recommending a set of various items to new users or a new item to a set of various users
- Non collaborative methods for the user's or item's early life.

Memory-based and model-based approaches are the two sub-categories of collaborative filtering algorithms.

Memory based methods work directly with the values of recorded interactions, assuming no model, and are essentially based on finding the closest users from a user of interest and suggest the most popular items among these.

Model based methods assume an underlying model that explains user-item interactions and attempt to discover it so that new predictions can be made.

2.4.1.1. Memory based collaborative approaches

2.4.1.1.1. User-user

To make a new recommendation to a user, the user-user method attempts to identify users who have the most similar interactions profile in order to suggest items that are the most popular among these users, and that are new to the user. This method is referred to as user-centered because it represents users based on their interactions with items and evaluates distances between users.

It is important to note that when calculating user similarity, the number of common interactions should be carefully considered, to avoid someone who has only one interaction in common with the reference user having a 100% match and being considered closer than someone who has 100 common interactions and agrees "only" on 98% of them. So, two users are considered similar if they have interacted with a large number of common items in the same way.

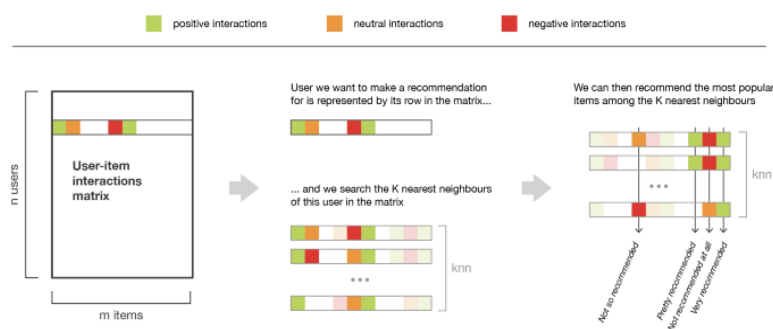


Figure 5: Illustration of the user-user method

2.4.1.1.2. Item-item

The item-item method seeks items similar to those with which the user has previously positively interacted, in order to make a new recommendation to the user. Two items are considered similar if the majority of users who interacted with both of them did so in a similar manner.

In order to obtain more relevant recommendations, this task is performed for more than just the user's favorite item and instead consider the n preferred items. In this case, it can suggest items that are similar to several of these preferred items.

This method is referred to as item-centered because it represents items based on interactions with them and calculates distances between them.

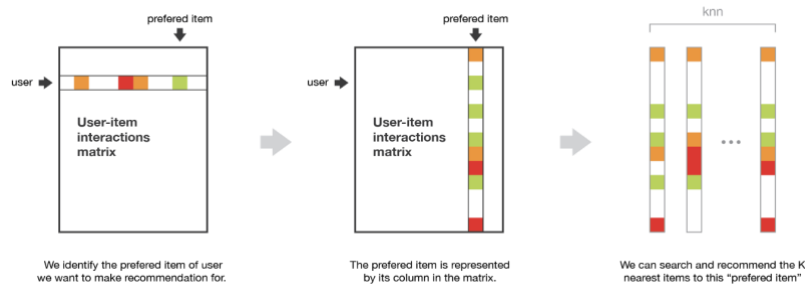


Figure 6: Illustration of the item-item method

2.4.1.1.2. Complexity and side effect

One of the most serious shortcomings of memory-based collaborative filtering is that it is difficult to scale: generating a new recommendation can take a long time in large systems. For systems with millions of users and millions of items, the nearest users search step can become intractable if not carefully designed. To make computations more tractable for large systems it can be used approximate nearest users.

Most recommendation algorithms must be extremely cautious to avoid a *rich-get-richer* effect for popular items. In other words, the system must not recommend more and more only popular items, nor do give recommendations for items that are extremely close to the one they already liked, with no opportunity to learn about new items they might like.

While these issues can arise in most recommendation algorithms, they are especially prevalent in memory-based collaborative ones, as previously stated. In the absence of a model to regularize, this type of phenomenon can be amplified and observed more frequently.

2.4.1.1.3. Comparing user-user and item-item

The user-user method is based on the search for users who have had similar interactions with items. Because each user has only interacted with a few items in general, the method is very sensitive to any recorded interactions (high variance). However, because the final recommendation is based solely on interactions recorded for users similar to the user of interest, it is obtained more personalized results (low bias).

The item-item method, on the other hand, is based on the search for similar items in terms of user-item interactions. Because a large number of users have interacted with an item in general, the neighborhood search is far less sensitive to single interactions (lower variance). As a result, interactions from all types of users are taken into account in the recommendation, making the method less personalized (more biased). As a result, while this approach is less personalized than the user-user approach, it is more robust.

2.4.1.2. Model based collaborative approaches

Model-based collaborative approaches rely solely on information about user-item interactions and assume a latent model to explain these interactions. Matrix factorization algorithms, a method to narrow the search, decompose the massive and sparse user-item interaction matrix into a product of two smaller and denser matrices: a matrix containing user representations and a matrix containing item representations.

2.4.1.2.1. Matrix factorization

The main assumption underlying matrix factorization is that there exists a latent space of features in which it can be represented both users and items, and that the interaction between a user and an item can be computed using the dot product.

For example, in order to model user-movie interactions, it can be assumed that there are some features that describe and distinguish movies fairly well, also, these characteristics can also be used to describe the preferences of the user.

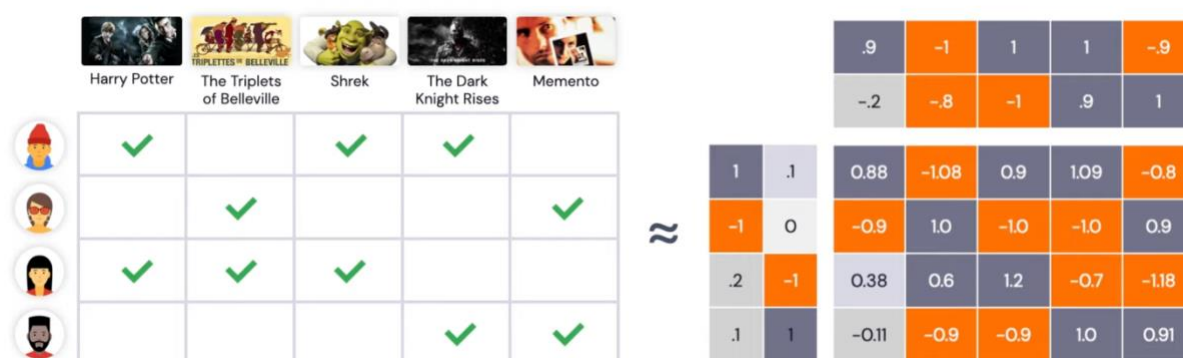


Figure 7: Illustration of the matrix factorization from user-movie

However, these features are not explicitly provided to the model. It is let the system discover these useful features on its own and create its own representations of both users and items. Because they are learned rather than given, extracted features have a mathematical meaning but are impossible to understand as human.

It is not uncommon for structures resulting from that type of algorithm to be extremely close to intuitive decompositions that humans could consider.

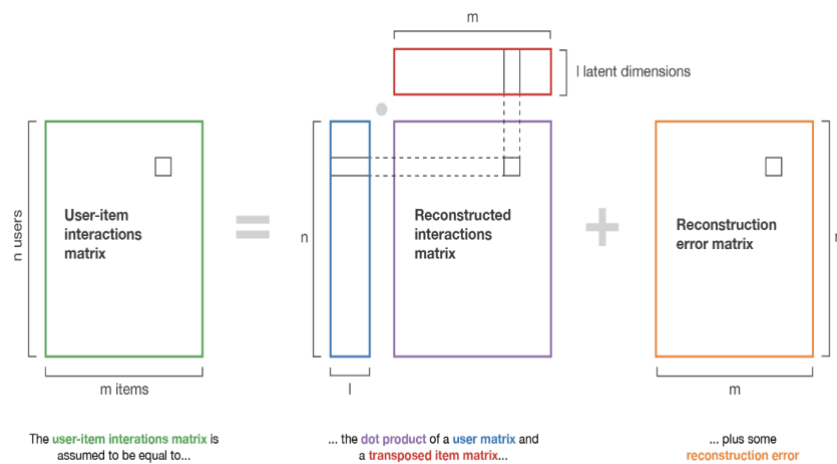


Figure 8: Illustration of the matrix factorization method

2.4.2. Content-based methods

Content-based methods, unlike collaborative methods that rely solely on user-item interactions, make use of additional information about users and/or items. In the case of a movie recommender system, this additional information could include the user's age, gender, occupation, or any other personal information, as well as the movie's category, main actors, duration, or other characteristics.

The goal of content-based methods is to try to build a model that explains observed user-item interactions using the available features. If such a model is obtained, then making new predictions for a user is relatively simple: all it has to be done is to look at the user's profile and determine relevant items to recommend based on this information.

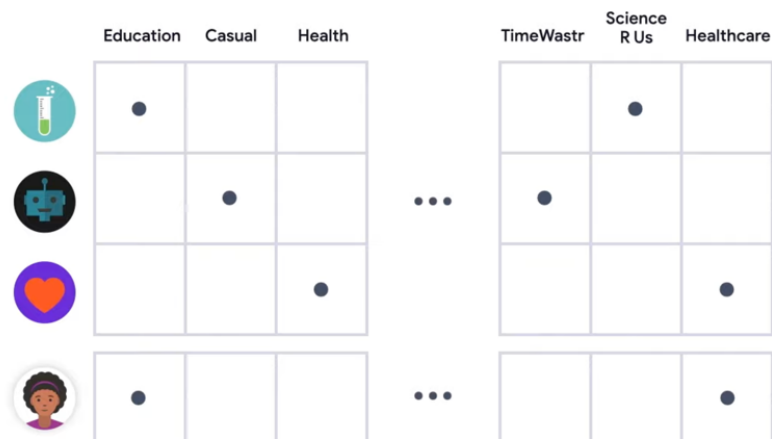


Figure 9: Illustration of content-based approach

New users or items can be described by their content, allowing relevant suggestions to be made for these new entities. Content-based methods suffer far less from the cold start problem than collaborative approaches, only new users or items with previously unseen features will logically be affected by this flaw, but once the system has matured, this is unlikely to occur.

2.4.2.1. Content based approaches

It has been mostly talked about user-user, item-item, and matrix factorization approaches in the previous two sections. These methods are part of the collaborative filtering paradigm because they only consider the user-item interaction matrix. Let us now go over the content-based paradigm.

The recommendation problem in content-based methods is casted into either a classification problem or a regression problem, predicting whether a user likes or does not like an item or predicting the rating given by a user to an item, respectively. In both cases, it is created a model based on the user and/or item features available to us, the content.

2.4.2.1.1. Item-centered

If the approach is based on user features, this approach is called item-centered: modeling, optimizations, and computations can be performed by item. In this case, the model is build and learnt by item based on user features in order to answer the question "what is the probability for each user to like this item?" or, for regression, "what is the rate given by each user to this item?".

The model associated with each item is trained on data related to that item, which leads to fairly robust models in general because many users have interacted with the item.

The interactions considered to learn the model, come from each user, and even if these users have similar traits, their preferences can differ. This means that, even though this method is more robust, it is less personalized (highly biased) than the user-centered method.



Figure 10: Illustration of the item-centered approach

2.4.2.1.2. User-centered

The method is user-centered if it is working with item features: modeling, optimizations, and computations can all be performed by the user. Then, based on item features, one model is trained per user to try to answer the question "what is the probability for this user to like each item?" or, for regression, "what is the rate assigned to each item by this user?"

Then a model trained is attached on the user's data to each user: the model obtained is thus more personalized than its item-centered counterpart because it only considers interactions from the considered user. However, because most of the time a user has interacted with a small number of items, the model it is obtained is far less robust than an average model. Finally, content-based methods can be neither user-centered nor item-centered

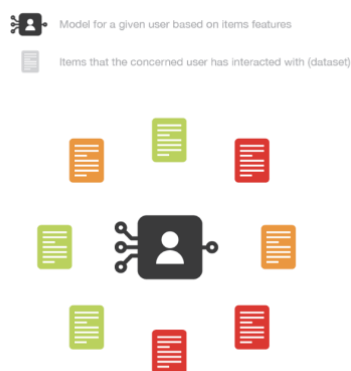


Figure 11: Illustration of the user-centered method

From a practical standpoint, it is much more difficult to ask some information to a new user than it is to ask a lot of information about a new item, people adding them have an interest in filling this information in order to make their items recommended to the right users. [\[12\]](#)

2.4.3. Collaborative vs content based

Let's take a closer look at the key differences between the two approaches. Let's look at the impact of the modeling level on the bias and variance.

In memory-based collaborative methods, no latent model is assumed. The algorithm works directly with user-item interactions: users are represented by their interactions with items, and suggestions are generated using a nearest search of users on these representations. Because no latent model is assumed, these methods have a low bias but a high variance in theory, meaning more personalized with more sensitivity to the interactions.

In model-based collaborative methods, a latent interaction model is assumed. From its own representation of users and items, the model is trained to reconstruct user-item interaction values. This model can then be used to generate new suggestions. This method has a higher bias but lower variance than methods assuming no latent model because a model for user-item interactions is assumed and users tend to be grouped. This means that the results are less personalized and also less sensitive to sporadic interactions.

In content-based methods, some latent interaction model is assumed. However, in this case, the model is given content that defines the representation of users and/or items. Users are represented by given features, and the model tries to fit for each item the type of user that likes or dislikes this item. As with model-based collaborative methods, a user-item interactions model is assumed in this case. However, this model is more constrained and, so, the method tends to have the highest bias but the lowest variance, meaning less personalized and less sensitive to unique interactions. [\[13\]](#)

2.4.4. Evaluation of a recommender system

In order to determine which algorithm is best suited to the situation, it must be able to assess the performance of the recommender systems. There are two types of evaluation methods for recommender systems: evaluation based on well-defined metrics and evaluation based primarily on human judgment and satisfaction estimation.

2.4.4.1. Metrics based evaluation

If the recommender system is based on a model that produces numerical values such as ratings predictions or matching probabilities, the quality of these outputs is assessed in a very traditional way by employing an error measurement metric such as mean square error. In this case, the model is trained on a subset of the available interactions before being tested on the remainder.

If it is considered a recommender system that is not based on numerical values and only returns a list of recommendations, such as user-user or item-item approaches, it can still be defined a metric by estimating the proportion of recommended items that truly suit the user.

To estimate this precision, it cannot be considered recommended items with which the user has not interacted, and it should only consider items from the test dataset available.

2.4.4.2. Human based evaluation

When designing a recommender system, it may be aimed not only in obtaining models that produce recommendations that it is confident in, but also in other desirable properties such as recommendation diversity and explainability.

As mentioned in the collaborative section, users must not be trapped in what it is referred to as an information confinement area. The term "serendipity" is frequently used to describe a model's tendency to create such a confinement area.

Serendipity should not be too low because it would create confinement areas, but it should also not be too high because it would indicate that it does not take users' interests into account enough when making recommendations. Thus, in order to add variety to the suggested options, recommended items should both suit the user well and are not overly similar to one another.

Another important factor in the success of recommendation algorithms is their explainability. It has been demonstrated that when users do not understand why they have been recommended a specific item, they lose trust in the recommender system.

So, this is why sometimes it is included a small sentence explaining why an item was recommended: "people who liked this item also liked this one".

2.4.5. Hybrid Models and deep learning

Deep learning is used to combine collaborative filtering and content-based models in the most modern recommendation engine algorithms. It can learn much finer interactions between users and items using hybrid deep learning algorithms. They are less likely to oversimplify a user's tastes because they are non-linear.

Deep learning models are capable of representing complex tastes across a wide range of items, even from cross-domain datasets, for instance covering both music, movies and TV shows.

Users and items are modeled in hybrid deep learning algorithms using both embeddings learned via the collaborative filtering approach and content-based features. Once the embeddings and features have been calculated, the recommendations can be served in real time.

The main disadvantage of deep learning models is that extensive parameter optimization is required. In comparison to the abundance of computer vision architectures, recommendation engines have almost no well-known architecture or pre-trained model.

Deep learning algorithm usually goes through three main stages, excluding optimization: retrieval, ranking and post-ranking.

The retrieval stage is responsible for selecting an initial set of hundreds of candidates from all possible candidates. The main objective of this model is to efficiently weed out all candidates that the user is not interested in.

The ranking stage takes the outputs of the retrieval model and fine-tunes them to select the best possible handful of recommendations. Its task is to narrow down the set of items the user may be interested into a shortlist of likely candidates.

Post-ranking is an even more refined stage, and the output is reduced to a few dozens.



Figure 12: Representation of deep learning stages

2.4.5.1. TensorFlow Recommenders

TensorFlow Recommenders (TFRS) is a library for building recommender system models. It helps with the full workflow of building a recommender system: data preparation, model formulation, training, evaluation, and deployment. It's built on Keras and aims to have a gentle learning curve while still giving the developer the flexibility to build complex models. It is the library used in this project.

TFRS makes it possible to:

- Build and evaluate flexible recommendation retrieval models.
- Freely incorporate item, user, and context information into recommendation models.
- Train multi-task models that jointly optimize multiple recommendation objectives.

[\[14\]](#)



Figure 13: TensorFlow logo

2.4.5.2. Light FM

Light FM is an implementation of several popular recommendation algorithms for implicit and explicit feedback. It's easy to use, fast and produces high-quality results.

It also makes it possible to incorporate both item and user metadata into the traditional matrix factorization algorithms. It represents each user and item as the sum of the latent representations of their features, thus allowing recommendations to generalize to new things, via article features, and new users, via user features.



Figure 14: LightFM logo

2.4.5.3. Spotlight

Spotlight uses *PyTorch* to build both deep and shallow recommender models. By providing both a slew of building blocks for loss functions, representations, and utilities for fetching (or generating) recommendation datasets, it aims to be a tool for rapid exploration and prototyping of new recommender models.

Spotlight provides a range of models and utilities for fitting next item recommendation models.



Figure 15: Spotlight logo

2.4.5.4. Seldon Server

Seldon Server is a machine learning platform and recommendation engine.

It provides an open-source data science stack. *Seldon* can be used to deploy machine learning and deep learning models into production on-premise or in the cloud.

It includes an API with two key endpoints:

- Predict: Build and deploy supervised machine learning models created in any machine learning library or framework at scale using containers and microservices.
- Recommend: High-performance user activity and content-based recommendation engine with various algorithms ready to run out of the box.



Figure 16: SeldonServer logo

[\[15\]](#)

2.4.6. Conclusions

With the well-known mathematical algorithms, it is possible to achieve really good results with great performance and robustness. Choosing between the different approaches makes the recommendations vary from more personalized to less personalized and from having more different results or narrowing them.

To decide on which method will be the best to implement for this project, it is needed to recapitulate the situation of the problem stated: recommending players to other players with rather no information of the player itself, being the main source of data the interactions the player makes while playing, navigating, or chatting inside the game.

This is the most user-friendly approach to make because, users, or in this case players, tend not to be willing to give much information more than necessary to play, and it is an enhancement that not all players want to benefit from. So, keeping the user less concerned with the information it is needed is a good proposal.

There is a clear positive presumption of defining certain features of the users inside the model such as age or language, as well as the items that they use (characters, settings, playing habits, etc.), so a method where it can be defined with some characteristics to be based on seems a good method to seek.

Collaborative memory-based methods seem powerful but also not so controllable and scalable. Collaborative model based methods seem a good approach to follow for this problem, but, the fact that features cannot be defined seems a turning point because some features like age or language spoken should be considered and they may not be taken into account using a blind system where most of the in-game interactions will have no relation to the user profile.

On the other hand, content based methods seem to be a good solution for the problem: it can have features to be assessed and it learns from the interactions. What may cause problems is the clustering of the users, the system might try to group as much as it can the largest number of users in the same profile and therefore, the recommendation has no variety because the same players will be recommended again and again.

Majority of online sites use a hybrid approach with machine learning involved. [\[16\]](#) They are powerful and flexible, adapting to almost any situation if they are well trained.

Using libraries that implements these systems also reduces the complexity of the problem because there are modern ones such as *TensorFlow Recommenders*, that simplifies the problem of having to implement the mathematical models behind these systems as they are presented in a high-level programming language, hiding the intrinsic of the calculations. [\[17\]](#)

Hybrid models seem to satisfy all the necessities of the problem, as they accept the definition of features for users and items, libraries implementing these systems are ready to use and have a good ratio of personalized-diversity results. That's why they are the choice for solving the problem in this project.

Between all the recommender open-source libraries, the one it is used in this project is the *TensorFlow Recommenders* library. It is behind the powerful company of *Google*, and it is open source, with a lot of implemented features out of the box and with a huge community that can be really helpful in a lot of scenarios.

3. Project management

To plan and prepare a project is almost a guarantee that it will flourish. It is necessary to visualize the management of it and that is what it is seen in this section, from the overall planification of tasks to the risks and contingency plans.

3.1. Procedure and tools

3.1.1. GANTT

This table shows the overall time and task context of the project to understand the later planification of it.

Task	Hours
Planning	40
Project statement	3
Objectives and scope	2
Current technologies research	11
Management tools research	1
Market study	2
Tasks specifications	1
Learning of technologies	20
Implementation	280
Creation of the model and defining parameters to be used	50
Implementation of the model	80
Tweaking and adjustments	80
Polishing	40
Presenting the results	30

Closing	40
Documenting last parts	20
Final revision	5
Final presentation preparation	15

Table 1: Project's tasks time estimation

3.1.3. Asana

Large projects have a lot to plan about, and it's easy to mess up and forget things. Asana helps in development keeping it organized.

Asana is a web and mobile work management platform designed to help teams organize, track, and manage their work. It is a free tool, with a paid version, that structures in columns, lists or even in a time chart the different tasks a job will go through.

This project's tasks will be divided in five columns, depending on their state: backlog, to do, doing, review, done.

In the backlog it is placed all the tasks at first, separated by sprints, and then the tasks go through the states of to do, doing, reviewing, and done, depending in which state they are.

3.1.4. GitHub

GitHub is a provider of internet hosting for software development and version control using Git that is software for tracking changes in any set of files, usually used for coordinating work among programmers collaboratively developing source code during software development.

This is a perfect tool to use in this project along with the *git flow* branch management, where there are these branches:

- Release or master: Production, stable and release versions are uploaded here
- Development: Where all the new features are being stacked once developed till there is a release ready to be made in the master branch.
- Feature: New features are implemented in different branches till they are ready to merge to the development branch.

3.2. Validation tools

To validate it, it is used several methods:

3.2.1 Project development validation

At the end of each sprint, there may be two reviews, one is a personal review acting the developer as a client that wants to see how the product is going and if the functionalities expected are working properly.

And the other is a review with the project director, where it is discussed the development of the project, and if anything needs to be adjusted or if it is going differently as expected.

3.2.2 Project content validation

The validation of the content of the project is considered the evaluation of the model implemented and it is done using the methods detailed in the section [2.4.4 Evaluation of a recommender system](#). As the recommender provides a numerical output, the evaluation of the system is done using the Root Mean Squared Error, that is the average squared difference between the estimated values and the actual value.

The evaluation of the final system can be found in the section [5.4.3.5. Training and evaluating](#).

3.3. SWOT analysis

3.3.1. Strengths

- I have been a player since very young and developer for some years, so I can see both perspectives, being the beneficiary and the user of the results of this project
- Having used several applications that provides similar services, such social media or ecommerce

3.3.2. Weakness

- No experience in implementing recommender systems
- Little knowledge of the libraries and services I am using in this project
- One man army: development, user interface, user experience, testing... All the work may be overwhelming for a single member

3.3.3. Opportunities

- No game offers this service in the same approach it is proposed
- Already existing applications do not focus on user experience, and they do not coddle users and prefer them to leave rather than offering a better service

3.3.4. Threats

- Oversaturated market, difficult to stand out just by exhibiting this feature
- Technologies change fast, and the final proposal may be deprecated quickly
- Data required for the system is expensive to collect

3.4. Risks and contingency plans

It is crucial to detect risks that can affect the development and to have a solution planned just in case it is needed to reconduct the project due to the events.

The possible identified risk of this project, and the matching solutions, ordered from less to more importance, are the following:

Risk	Solution
Problems developing the application, lack of knowledge of the technologies and libraries or problems in developing certain parts of it.	Limit the range of the implementation. It could be reducing the functionalities of it or the visual design.
The necessary devices to develop the application break and stop working.	There are alternatives devices that can be used at the university
The project extension can be too wide to provide a high fidelity, almost best solution.	Narrow down the range of characteristics the system must consider, to simplify the model and with so, the solution.

The provision of functionalities of the project is not fully accomplished.	Follow a path of consistent development to ensure that from a certain point the the implementation remain functional even though it is not fully developed
Data available is not suitable for a recommender system	Adapting the data or the system to fit together

Table 2: Risk and contingency plans

3.5. Initial costs analysis

In this chapter it is detailed the associated costs of the project. These costs include human resources, hardware and software and services.

Resource	Cost
Human	5.400€
Physical	6.275€
Total	11.843€

Table 3: Total initial costs analysis table

3.5.1. Human resources

As defined in the project planification, a total of 360 hours is needed to finish it.

The team is formed by one member, and it has the main role of a developer, without taking into account all the other jobs that are not the main topic of the application such as the job of the designer, or the user experience specialist. To simplify the overall calculations, it will be obviated. One approximation of the salary of a developer could be 15€/hour.

Stage	Hours	Cost
Planning	40	600€
Implementation	280	4.200€
Closing	40	600€
Total	360	5.400€

Table 4: Human resources costs table

3.5.2. Physical resources

Resource	Price	Use	Cost
Acer Nitro N50	1100.00 €	6 months	1.100€
Electricity	0,35€/kWh	500 kW	175€
Rental	800€/month	6 months	4.800€
Internet	50€/month	6 months	300€
Total			6.375€

Table 5: Physical resources costs table

4. Methodology

This planning is made considering that the project starts at 14th February and the final delivery of it is on 30th June and, considering that the final degree project corresponds to 12 ECTS credits, and one credit is equivalent of 30 working hours, it is estimated that a total of 360 hours is available for the realization of the project.

4.1. Working method

For this project it is planned to use an agile methodology. An agile methodology provides to the client the value of the project in a fast manner, so risks and inconveniences are seen in time and they can be prevented. [\[18\]](#)

Most famous agile methodology is Scrum, but in a single member team it needs to be adapted as there are no meetings, there is no need to share documents or to explain parts of the project to the rest of the team.

However, the methodology used is almost the same as the original. The project is divided into objectives, then told apart into tasks which are shared out between the sprints.

4.2. Project stages

The project stages consist of three main stages: planning, development and closing.

4.2.1. Planning

At this stage of the project, there is already an idea of the project, it is refined and detailed. Also, the execution of it is planned and all the main aspects to be known before the implementation of the project are documented.

4.2.2. Development

The objective of this stage is to develop the project. While being in this stage, divided in sprints, the focus is in completing objectives. They are detailed and time estimated, divided in sprints and respecting the initial planning but being flexible in the time-wise planning.

At the end of this project the dataset is being fed to the model, that is in its final iteration with advanced functionalities, providing a proper recommendation based on its learned criteria, presented in a game simulation that has been built using *Unity*.

This means that the project succeeded in following the planning, including all the prepared steps and stages of the development to provide a final polished result.

4.2.2.1. Sprints

In every sprint there are tasks involved, that are the steps and parts of the objective to reach. The quality of code is analyzed, and it is evaluated on the achievement and completeness of the objectives, extending the documentation of the project, and appointing meetings with the project director.

The objectives in the sprints are divided in this arrangement:

- Sprint 1: Creation of the model and defining parameters to be used
- Sprint 2: Implementation of the model
- Sprint 3: Tweaking and adjustments of the model
- Sprint 4: Polishing
- Sprint 5: Presenting the results

Considering the steps presented in section [3.1.1 GANTT](#), the first steps of the development of the project were gathering the data and implementing the recommender model (sprint 1, 2). These steps were successfully executed on time. Next steps (sprint 3, 4, 5); tweaking and adjusting, polishing and presenting the results were slightly modified timewise because of the increasing complexity of the algorithm and the problems encountered during the development of it.

However, the project followed its route map though the problems that have been encountered that lead to adapt the initial planning, with no apparent repercussion on the costs of the project.

4.2.3. Closing

Closing of the project, this is the last stage of the project. Once the implementation is done, it is written the conclusions of it, extracting the result of all the work.

In academic terms, the documentation is about to be finished with the last details of it, and finally the final presentation of the project.

5. Project development

This section contains the description of the process of the development and tools used, as well as the implementation of the model and problems that arose.

The recommender system has been in various stages:

The very first implementation is the stage of retrieval, implemented during the sprint number two. The recommender at this point can provide user recommendations for a given user with a basic model.

Though, the recommendations provided by the system are limited in terms of personalization and they need to be tweaked and polished. The interactions provided by the dataset are treated as binary information, if an interaction existed it was implicitly considered as a good interaction and the algorithm used it to learn, discarding it otherwise.

The second stage is the stage of ranking, implemented during the sprint number three. The recommender at this point can make a rating's prediction of a user towards another user. The system evolved to include, process, and show more data as it is introduced in the system the rating feature. However, this rating output has almost no personalization and the algorithm learned very little about the interactions itself nor the profiles of the users.

The final stage is the stage of cross-ranking. The recommender predicts the rating that a user will make to another one and comparing this stage to the previous one, the difference is that now the algorithm considers all the profile features available, such as age, gender, or language and also it considers the in-game features such as favorite character or mode. So, it can predict ratings based on multiple factors, enriching the quality of the predictions and making them more personalized.

The main gap between the stages is from the ranking implementation to the cross-ranking. It can be observed the power of a cross recommender system that considers multiple features to make a personalized prediction.

5.1. Definition and justification of the approach

In previous sections there has already been stated that the project includes a stage of learning the technologies. Some of the learning is reflected at the [2.4. Recommender systems](#), where a fundamental understanding of what types of algorithms are there to approach recommender systems. The rest of the knowledge is acquired during the development so the deepness' limitations of it are strong.

Before the release of the library of *TensorFlow Recommenders*, there already existed recommender systems and they were developed using deep learning, neural networks and even with combinations of them with reinforcement learning. Nowadays these strategies still remain but new ones arise along the new technologies.

Moreover, the project is not only about implementing the recommender, but it also aims to deliver it using *Unity* game engine, to simulate a user interface of the game, and demonstrating that it can be ported to an actual game with real users.

Considering these points, the solution proposed for this problem is led to a fast and relatively simple system. This means that the bases are fulfilled as it is clearly defined the type of model and why it is that way, but it rather stands for simplicity and effectiveness over complexity and efficiency.

5.2. Tools

A brief overview of the tools used in this project and why they have been chosen.

5.2.1. Microsoft Word

A well-known software for editing text documents, it is a basic software used on any documentation. Developed by *Microsoft* leading the rank of text editing software.

5.2.2. Python

Python is a high-level, interpreted, general-purpose programming language. Its design philosophy emphasizes code readability with the use of significant indentation.

Talking about machine learning algorithms it is implicit to talk about python. Most of the libraries and frameworks are designed in python as they are aimed to be used by mathematicians, scientist, and developers.

Having these targets forces the implementation to be in a high-level programming language as is python so it overcomes the difficulties of low-level programming and allowing users to develop the algorithms without a deep level of it.

5.2.3. PyCharm

PyCharm is an integrated development environment used in computer programming, specifically for the Python programming language. It is developed by the Czech company JetBrains.

There are available a lot of free IDE for all kind of languages. For python the most famous ones are *Spider*, *Visual Studio Code*, *PyCharm*, *Atom* and so on. The reason *PyCharm* is being used in this work is because the company developing it is focused on IDE and have a lot of them, even *Android Studio* from *Google* is powered by *JetBrains*, the company behind *PyCharm*. Moreover, I have a solid background using it and it has powerful features such as debugging.

5.2.4. TensorFlow

In terms of machine learning, there are two main libraries: *TensorFlow* and *PyTorch*. They both are powerful and with a big company developing it, *Google* and *Facebook* respectively. They offer a lot of possibilities and paths to implement all kind of machine learning algorithms and it is a hard choice to make when choosing between them.

There is a turning point that makes *TensorFlow* more appropriate to be used in this project, and it is that it has a dedicated library for recommender systems since September 2020, and *PyTorch* released its dedicated library in February 2022. [\[19\]](#)

At the moment of *PyTorch* releasing its library, the project was already started and considering switching to it was a risky move, taking into account that it involved using a new technology without much background knowledge of it, both the library and the theory of algorithms behind recommenders, and it was a dangerous approach to consider.

5.2.4. Unity

Unity is one of the most used game engines worldwide. Demonstrating that a recommender system built with powerful libraries such as *TensorFlow* can be used inside a game developed with it, is a game changer in terms of user experience for the games developed in this engine. It widens the vision of developers as they can include a recommender inside their game because the implementation of it can be in their favorite game engine, and not in a particular limited engine built exactly for that purpose.

Unity also has its own machine learning library: machine learning agents. But they are targeted to reinforcement learning algorithms, that to keep it simple, they use the trial-error method to learn. This kind of approach is not suitable for this project, and it is not considered as an option although it would be ideal as the implementation would not leave the final *Unity* environment where the project is intended to be placed.

5.2.5. Unity Barracuda library

Unity Barracuda is a lightweight cross-platform neural networks inference library for *Unity*. It can run Neural Networks both on GPU and CPU. It is production-ready for use with machine learning agents, the own Unity library of artificial intelligence, and number of other network architectures such as onnx.

Barracuda is the library used in this project to use the model generated by *TensorFlow*. It must be converted to onnx format as Barracuda only accepts onnx standards for external machine learning applications.

5.2.6. Onnx library

The Open Neural Network Exchange is an open-source artificial intelligence ecosystem of technology companies and research organizations that establish open standards for representing machine learning algorithms and software tools to promote innovation and collaboration in the AI sector.

Onnx is an open format built to represent machine learning models. Onnx defines a common set of operators, the building blocks of machine learning and deep learning models, and a common file format to enable AI developers to use models with a variety of frameworks, tools, runtimes, and compilers.

5.3. Dataset

To have access to a large game' statistics database, populated with real in-game data of any game is almost impossible, because it is an internal characteristic that is harvested from the game itself, and for privacy protection.

There are some databases with useful data [\[20\]](#) but, they lack two important things: players' profile and behavior and being a large dataset. These features are a must when acquiring data for this project because it is intended to recommend players to other players based on their behavior inside the game and their gaming habits, and also, a machine learning algorithm needs a large dataset to train and test in order to be trustable.

To overcome this difficulty, the database used for this project is generated. This way it is ensured that the data needed is there and that there are enough entries to work with. In contrast to this project, where there is no actual game, developers implementing this recommender system will have it easier to gather the data as they can implement in-game tracking methods and then retrieving the data to accumulate a large database with real players' data.

The data generated is a symbolic representation of all the data that can be gathered from a game and kept in a database. There's no implementation of the database nor the calls to it. However, the dataset represents the response of a query to it, where it is combined raw data and aggregate fields, defining so, a limited and bounded context that makes sense in this project.

The dataset is composed by two files: `users.csv` and `users_interactions.csv`.

`Users.csv` has some basic information of the user as well as game traits.

`Users_interactions.csv` has 11 columns. The first columns represent a user and the user it has interacted positively and the rating of this interaction. The next 8 columns are added to simplify the feeding of the recommender while training and they contain the equivalent data of the `users.csv` file of the interacted user.

A positive interaction and its real rating may be considered when a user team up with another one, or when a user talks to another one or if their characters stay together for most of the match time. This decision is unapproached in this project, and it is treated as a black box that returns a list of positive interactions, decoupling the logic of the algorithm from the criteria of positive interactions.

Both files are needed because two reasons; it is needed to have the information of the users to deliver more accurate predictions, and predictions themselves. And it is needed feedback from the user to let the algorithm know what his or her preferences are and learn from them.

5.3.1. Populating the dataset

The dataset is filled with the information of 75.000 users, and 4.654.201 interactions with an average of 62, a minimum of 25 and a maximum of 100 interactions per user

The traits selected to be inside the dataset are the ones that are more common on most games and also have the major impact when creating relationships between gamers, such as the age or the language spoken.

The generation of it is done using the python packages *Faker* and *NumPy* and can be found here:

https://github.com/ponspack9/UserRecommenderSystem/tree/main/faker_generator

Link 1: Data generation source code

Faker is a Python package that generates fake data. It is used in many situations such as to bootstrap databases, create good-looking XML documents, fill-in your persistence to stress test it, or anonymize data taken from a production service. [21]

NumPy is a library for the Python programming language, adding support for large, multi-dimensional arrays and matrices, along with a large collection of high-level mathematical functions to operate on these arrays. [22]

With both libraries it is possible to generate a decent dataset to work with. These are the fields of users.csv file that are created to later use on the recommender system, based on the game *Overwatch* [23] to define the characters' names and types and game modes:

- User ID: unique user identifier
- Age: age
- Gender: gender
- Language: language in which the game is set by the user
- Login_time: estimated recurrent login time (minutes passed since 00:00 from that day)
- Character: Most played character
- C_type: Most played character type
- Mode: Most played game mode
- Ranking: User in-game score

For the fields Age and Ranking, it is used a normal distribution, to make the data closer to what can be in the real world. The other fields are just randomized between certain constraints such as the available characters and modes of the game used as reference, a sample of 50 languages and the minutes of one day.

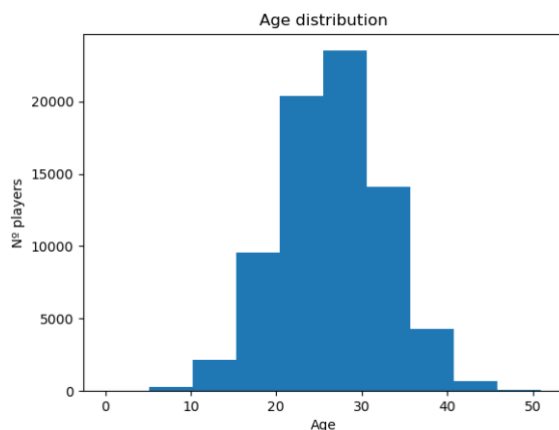


Figure 17: Plot of the age distribution of the users

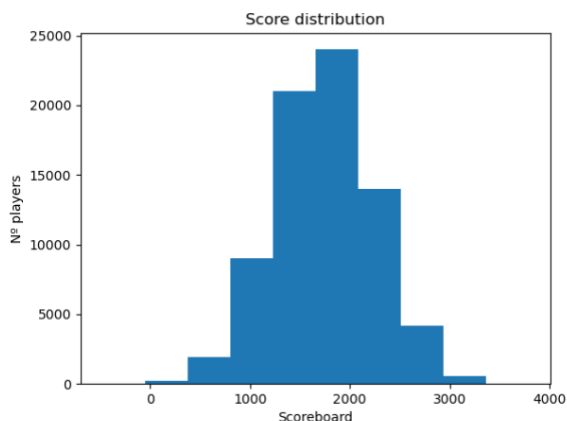


Figure 18: Plot of the ranking distribution of the users

user_id	age	gender	language	login_time	character	c_type	mode	ranking
1	19	Other	Hungarian	90	Tracer	Healer	Ranked	1139
2	32	Male	Bosnian	741	Ashe	Damage	Mayhem	2028
3	27	Male	Danish	1350	Soldier	Healer	CTF	2178
4	20	Female	German	1256	D.Va	Healer	Mayhem	1850
5	21	Male	Macedonian	469	Wrecking	Damage	CTF	690
6	19	Male	Catalan	1420	Echo	Tank	3VS3	1973
7	27	Male	Danish	225	Ana	Tank	3VS3	867
8	28	Male	Arabic	1033	Genji	Damage	Solo	2334
9	18	Other	Latvian	354	Wrecking	Tank	3VS3	1360
10	23	Male	Mongolian	673	Moira	Damage	Solo	1765

Figure 19: Sample of the users file

There are 11 fields in the user's interactions dataset:

- User_id: the user that made the positive interaction.
- Interacted_id: the user who the positive interaction is made with.
- Rating: the rating given of this interaction.
- User features [3,11]: the traits of the interacted user.

user_id	interacted_id	rating	age	gender	language	login_time	character	c_type	mode	ranking
1	44519	2.91	23	Female	Mongolian	858	Reinhardt	Damage	Quick	1998
1	27272	3.34	16	Male	Dutch	230	Winston	Tank	3VS3	1585
1	36986	3.12	19	Male	Swedish	461	Tracer	Tank	Ranked	2670
1	10628	2.15	24	Male	Belarusian	900	Echo	Tank	CTF	2282
1	48093	1.78	37	Other	Marshallese	624	Echo	Tank	Quick	2089
1	50721	2.74	26	Male	Armenian	1259	Lucio	Healer	CTF	1561
1	6350	2.09	28	Male	Belarusian	385	Zarya	Healer	Solo	2245
1	24454	2.5	33	Female	Albanian	1181	Zarya	Healer	CTF	1544
1	67220	2.74	23	Female	Arabic	482	Mei	Healer	CTF	1191
1	39324	2.83	30	Male	Indonesian	684	Brigitte	Damage	Quick	1498
1	3175	3.26	38	Male	Romanian	1224	Moira	Tank	Ranked	1976

Figure 20: Sample of the user interactions file

5.3.2. Defining the rating criteria

The rating of any interaction could be random as the other data generated, but then no conclusions of the model could be made. To avoid that and eventually see that the recommender has learned from the interactions, the rating is forced to be higher or lower by using the following criteria:

The total rating is weighted out of 120 points. Each feature weights 10 points of maximum value except for the language trait, that weights a maximum of 50 points. This difference is justified because language is a key element in communication, and it is not expected to have a fluid interaction of two users if they cannot speak the same language. Note that it is not considered if the users know a second common language.

The implementation of this criteria can be found in the file "leveraged_rating.py".

5.3.2.1. Age

From 0 to 10 points.

- Maximum: 10 points if both ages are the same.
- Minimum: 0 points if the ages differ for more than 10 years.
- In-between: Each year of difference subtract 1 of 10 till a minimum value of 0.

5.3.2.2. Gender

From 7.5 to 10 points. Female gender is better valued because of its scarcity inside the gaming community.

- Maximum: when the user is a male, 10 points are given if the interacted user is a female. When user is a female, 10 points are given if the interacted user is also a female.
- Minimum: 7.5 points are given if the maximum conditions are not met.

	Male	Female	Other
Male	7.5	10	7.5
Female	7.5	10	7.5
Other	7.5	7.5	7.5

Table 6: Dataset gender rating criteria

5.3.2.3. Language

From 0 to 50 points.

- Maximum: 50 points if both users speak the same language
- Minimum: 0 points if they don't speak the same language

5.3.2.4. Login time

From 0 to 10 points.

- Maximum: 10 points if both users connect at the same time
- Minimum: 0 points if the login time differs more than 90 minutes
- In-between: The points given are proportional of how their login times differ, for example if the absolute value of the difference of the login times is 45 minutes, 5 points are given.

5.3.2.5. Character

From 0 to 10 points.

A complementary type is any other type than the one the character belong. For example, a character belonging to damage has as complementary tank and healer. It is less valued the character belonging to damage because it is harder to be help each other in-game.

- Maximum: 10 points if the character belongs to a complementary character type.
- Minimum: 0 points if both characters are the same.
- In-between: 3.3 points if both users' character types are damage, 6.6 points if they have the same other character types.

	Damage	Tank	Healer
Damage	3.3	10	10
Tank	10	6.6	10
Healer	10	10	6.6

Table 7: Dataset character rating criteria

5.3.2.6. Character type

From 6.6 to 10 points.

The rating criteria is similar to the character criteria, having greater valuation the complementary types. In this case, damage is not penalized because it is considered that two damage characters can team up and make combos.

- Maximum: 10 points when types are complementary
- Minimum: 6.6 points when types are the same

	Damage	Tank	Healer
Damage	6.6	10	10
Tank	10	6.6	10
Healer	10	10	6.6

Table 8: Dataset character type rating criteria

5.3.2.7. Mode

From 0 to 10 points.

- Maximum: 10 points if the game mode is the same.
- Minimum: 0 points if the game mode is different.

5.3.2.8. Ranking

From 0 to 10 points.

- Maximum: 10 points when the rankings are the same.
- Minimum: 0 points when the rankings differ more than 1000
- In-between: The points given are proportional of how the rankings differ, for example if the absolute value of the difference of the rankings is 500, 5 points are given.

5.4. Recommender system

The code of the implementation can be found here:

<https://github.com/ponspack9/UserRecommenderSystem>

Link 2: Recommender system implementation's source code

The development of the recommender system is divided in 3 main stages: implementing the retrieval stage, implementing the ranking stage and implementing the final cross-ranking stage.

It is described the process of the three of them, to see the evolution and enhancement of the recommender, implicitly justifying the stages and the need of them. Note that some concepts are described in the first stages and apply for all of them.

The use of the stages in this project is not the same described previously in the state of the art. They are supposed to be concatenated one to another, meaning that the retrieval stage feeds the ranking stage and then it is provided the final prediction in the post-ranking stage. This method described is to manage really large database where the input is at the magnitude of millions and there has to be a massive discard of the possible elements to provide a prediction.

Also, in this project the approach of presenting the recommendations is different. The usual way of recommending items is to blind recommend, meaning that new content is presented to the user hoping that the user would like it not giving the predicted rating, in this project the latest is used, and the recommender is used to present the internal rating as an approximation of what the user would rate. So instead of presenting new users inside the context of “You would like to play with ...” the approach of this work is to present the users inside the context of “Your affinity to this users is ...”.

5.4.1. Retrieval

Here it is presented an abstraction of the steps of the retrieval implementation developed:

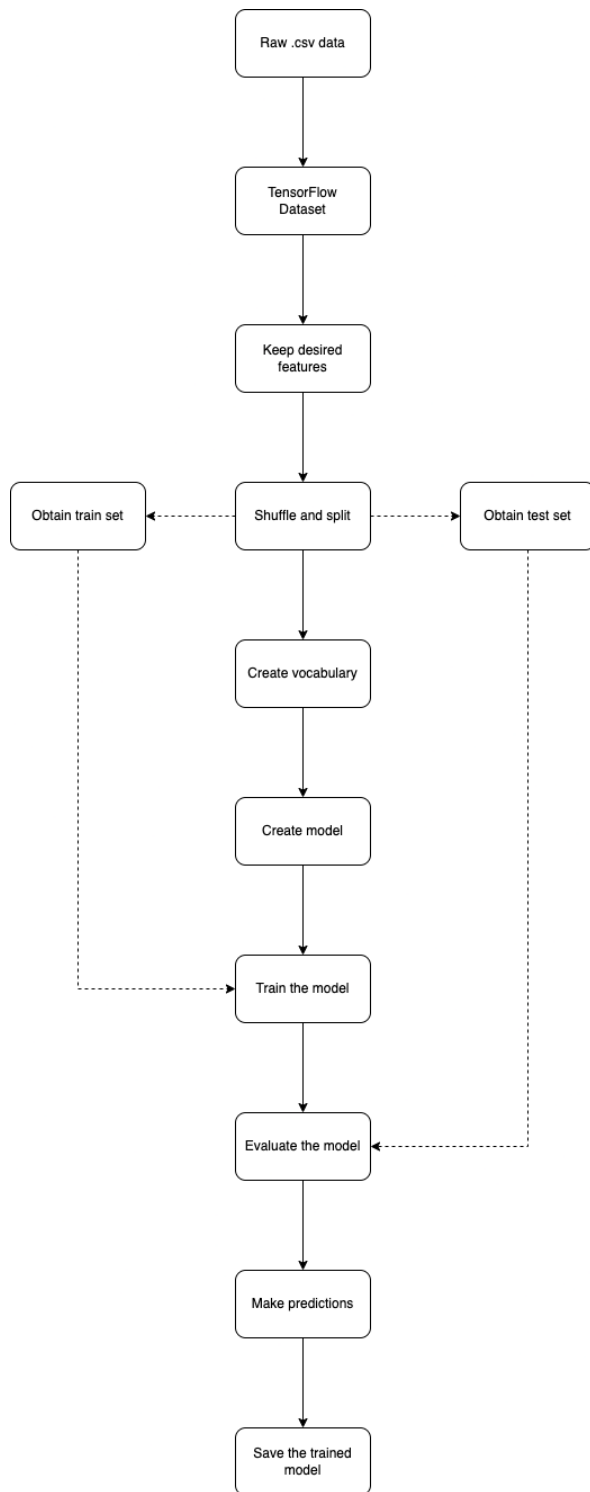


Figure 21: Diagram of the retrieval implementation's steps

5.4.1.1. Model definition

The implementation of the algorithm uses a content-based item-item approach and it is represented with a two-tower model. This means that there is one query tower and one candidate tower.

The query tower is what it is given as an input to the model, expecting it to return a parameter from the candidate tower, being it the output of the query.

For example, in another scenario where the system recommends movies to a user, the query tower would be the users and the candidate tower would be the movies.

In this case, being the users both of them, the query tower is the users and so it is the candidate tower. This means that given a user as input, it is returned another user as output.

The positive interactions take an important role inside the model as they are used to train the model to learn how the user behaves.

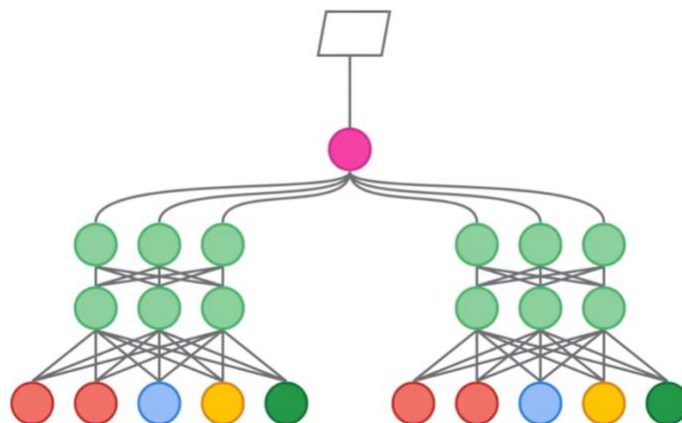


Figure 22: Illustration of a two tower model

5.4.1.2. Data parsing and pre-processing

To implement the model, first of all, it is needed to parse the dataset. To do this, the library provides some functions to easily convert raw data to an object that it can use.

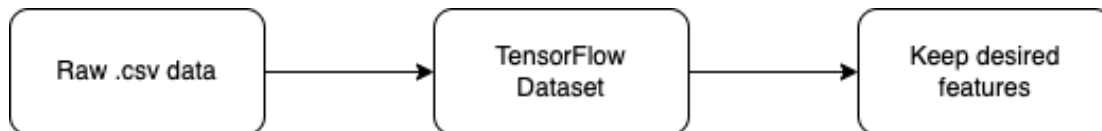


Figure 23: Diagram of parsing and pre-processing step in retrieval

Once the data is parsed, some of the features are left out of the dataset. This is a common step as the dataset may not be clean or the developer wants to test only some features instead of all of them. In this point of the development, the objective is to provide a simple model that is functional, so it is only kept the user identifier (user_id) from the users dataset and the user identifier for both of the users from the interactions dataset.

That means that the system recommends users based only on past interactions with other users without taking into account any other features or data.

5.4.1.3. Data processing

The simplified data is shuffled and then split into subsets, the train and the test set.

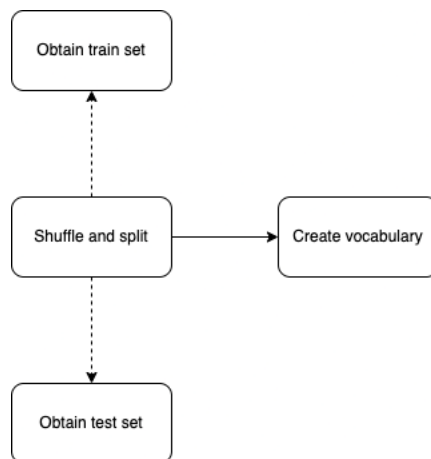


Figure 24: Diagram of data processing step in retrieval

The shuffling is made because the set may have consecutive entries for the same user, as seen in [figure 20](#), and if it is so, this implies that the algorithm would never learn from the latest entries of the dataset meaning that some users only appear in the test set and can never be used for learning. And vice versa, that some user will never be used to test the algorithm because they appear in the beginning of the file.

Once the full dataset is shuffled, it is ready to be split.

The train set is the one that will be used when training the model, or also known as fitting the model. This set usually contains most of the data and so does in this case, containing the 80% of the total interactions.

The test set is used when evaluating the model once it has learnt from the train set and it contains the remain 20% of the data. This process returns the same values of the training stage, but the model is no longer modified.

The vocabulary is the unique “words” that the algorithm is allowed to use. It is used as an identifier because it only contains the unique data available. For example, from the interactions set seen in [figure 20](#), the vocabulary based on that figure would be only “1”.

Once it is created the data is ready to be pushed to the model.

5.4.1.4. Model creation

The model is a two-tower model. It has two sequential models inside it, representing the query and the candidate towers. A sequential model is appropriate for a stack of layers where each layer has exactly one input value and one output value, as it is the approach of it right now. A layer is an object that process some data and modifies its internal parameters to adjust and learn from the input and once done, it gives the output to the next layer.

The model receives one input (the user identifier) and gives one output (another user identifier) as right now, it is prioritized keeping the simplicity of the model. This model implementation change in the next iterations as it will accept more parameters as input.

The training data has positive user-interactions pairs. To figure out how good the model is, it is used a “top score” metric. The affinity score that the model is producing is compared to all the other possible candidates to conclude if it is accurate or not, if the score for the positive pair is higher than for all other candidates, our model is highly accurate.

5.4.1.5. Training and evaluating

At this point the model has the data ready, knows how to manage it and how to figure out if it is doing well. The next step is to actually train and test it.

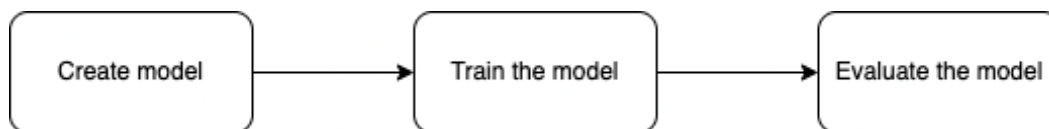


Figure 25: Diagram of training and evaluation step in retrieval

To train it is mandatory to define the number of epochs and the batch size. One epoch is one complete cycle of processing or training the data. The batch size is the partitions in which the data is divided.

These two parameters are relevant because a low epoch could mean that the model does not learn enough from the data, especially in small datasets, and a large epoch could lead to the model to be overfitted, meaning that it has learn “by heart” all the training data and it only performs well with already-seen data, and giving nonsense results to the new one.

The batch size is related with the modification of the internal parameters of the model as at the end of every batch the model adjusts its parameters. Low batch sizes could lead to the model trying to adapt and fit every single entry of the dataset, going around without actually making any progress at generalization. Large batch sizes could lead to the model trying to fit, in fewer steps, large amounts of data and the individual traits of each entry are lost and merged in the immensity.

Tweaking these two numbers are a tricky job and there’s no must-follow rule, every model and situation has a different pair of them. This model uses 3 epochs and a batch size of 8192.

This table has the results of the metric that evaluates the model. As the model trains, the loss is falling, and a set of top-k retrieval metrics is updated. These tell us whether the positive candidate is in the top-k retrieved items from the entire candidate set. For example, a top-5 accuracy metric of 0.2 would tell us that, on average, the positive candidate is in the top 5 retrieved items 20% of the time.

The following table contains the results given from the training and evaluation of the model.

	Epoch 1	Epoch 2	Epoch 3	Evaluation
Top 1	0,0045	0,0263	0,1148	0,5572
Top 5	0,2893	0,6445	0,8439	0,8581
Top 10	0,3577	0,6855	0,8592	0,8671
Top 50	0,5196	0,7672	0,8895	0,8906
Top 100	0,5892	0,8439	0,9028	0,9022

Table 9: Results of model training and evaluation

The results show an increasing accuracy of the model through the epochs. This means that the model is learning but a too high accuracy for low top-k metrics, such as top-1 or top-5 may mean that the model is overfitted.

5.4.1.6. Making predictions and saving

Once the model is trained and evaluated, the last step is to use and save it.



Figure 26: Diagram of the predictions and saving step

To use it, it is only needed to be passed an input, the user identifier and it will return the recommendations for this user, being them the identifiers of the recommended users.

Recommendations for user 42: [26759 74055 15415 3685 38633 68451 23496 34575 19173 3309]

5.4.1.7. Conclusions

To extract the conclusions once reached this stage of the project, we need to analyze which were the objectives and if they are accomplished.

The main objective to reach at this point was to have the dataset gathered and a functional early version of the recommender system that has the capability of recommending users but with no personalization, as tweaks and adjustments will be made in next iterations.

The project has achieved the planned stage at this point although it has encountered some problems that delayed the development of it. These delays have made the implementation of the recommender system to be very basic and to be presented as an early release, although being a good starting point, currently there is no real use of it as the recommendations that gives are quite impersonalized.

Also, it is relevant to note that the proposed solution of the model suffers from the *cold start* problem, where it would be impossible to deliver proper recommendations to new users as they have no data to be based on.

This is a common issue of recommender systems, and this project does not approach a solution for it. In a logical sense makes sense not to recommend significant users till the system has modelled a “profile” for that specific user, but in a practical point of view new users should also be given the opportunity to see recommendations, in this document it is explained some of the techniques used to solve this, but it is assumed that it will be solved in the future iterations of it.

Though these facts, the system learns and it is prepared to incorporate new features, having a good potential.

5.4.2. Ranking

The steps in this stage resemble to the retrieval stage, having slight differences in the implementation:

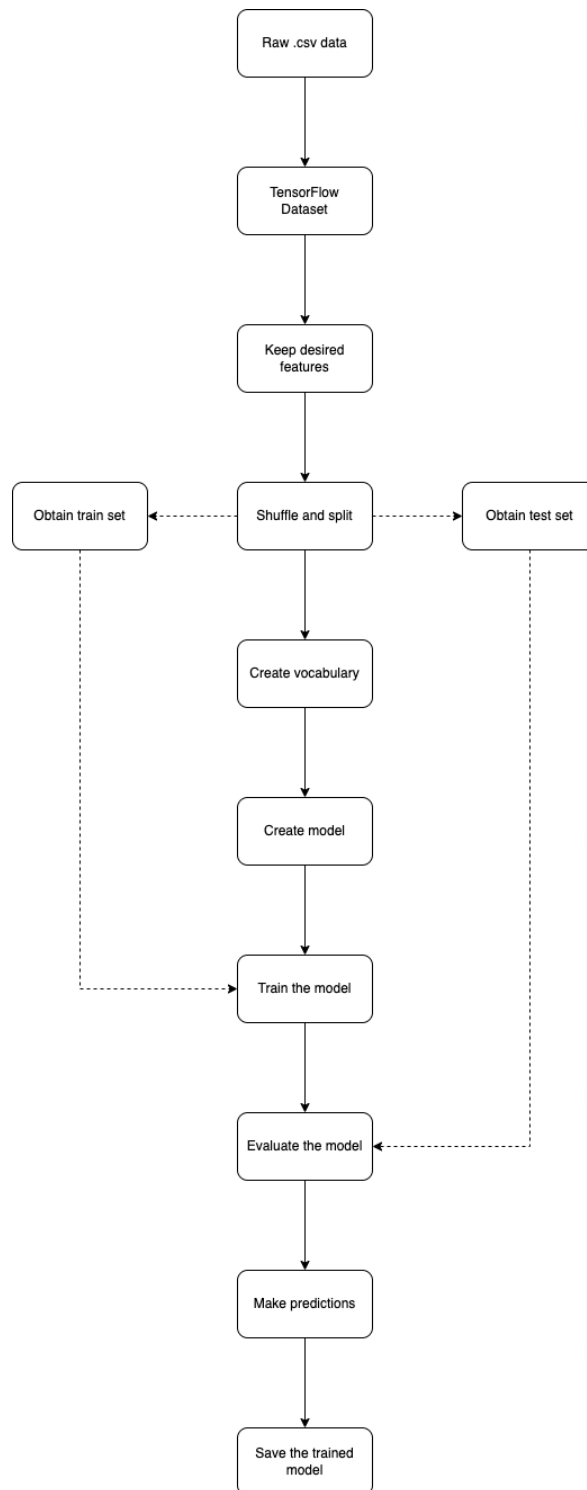


Figure 27: Diagram of the ranking implementation's steps

5.4.2.1. Model definition

The definition of the model varies respect to the retrieval stage. It is introduced the rating feature, and the model uses it to learn to later produce a rating output.

The model is still based in a two-tower model, the first tower contains the users, and the second tower contains the interacted users, but this time it outputs the rating instead of the interacted user.

The ranking model has two inputs and one output, so, given a user and another user it outputs the prediction of the rating.

It is easy to see that there's little, not to say, no information that the model is trained by. The quality of the predictions in terms of personalization are low as it learns "the greatest interacted users" and tries to predict other interactions based on that.

5.4.2.2. Data parsing and pre-processing

The data parsing and pre-processing proceeds the same way it does in the retrieval stage, with the difference that there is this one extra feature introduced that now is being considered.

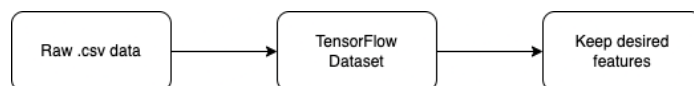


Figure 28: Diagram of parsing and pre-processing step in ranking

5.4.2.3. Data processing

This step is exactly the same as the retrieval stage. The dataset is split in 80% training and 20% testing.

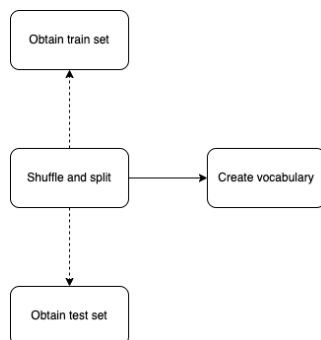


Figure 29: Diagram of data processing step in ranking

5.4.2.4. Model creation

To implement the two-tower architecture and to take into account the ratings it is used 3 Sequential models. The first two models are there to pre-process the data and give it to the final deep sequential model where all the calculations are made.

The weights used in the calculations are modified during the training process to achieve the result of giving a rating out of two users as an input.

5.4.2.5. Training and evaluating

Once the model is ready, the next step is to train it.

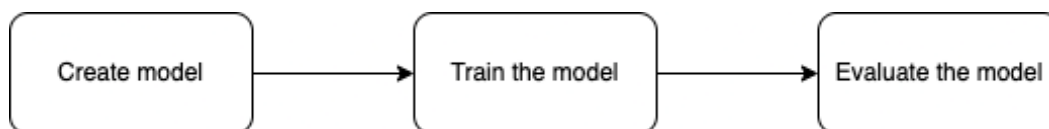


Figure 30: Diagram of training and evaluation step in ranking

It is no longer used the same metrics chosen in the retrieval stage.

The ranking stage provides a number as output, so it can be used some mathematical estimations such as the mean squared error (MSE) or root mean squared error (RMSE). In statistics, the mean squared error or mean squared deviation, measures the average of the squares of the errors, that is, the average squared difference between the estimated values and the actual value. MSE is used as a loss function and RMSE as a metrics function to evaluate the model.

The target value using these formulas as metrics is as close to 0 as possible, because that means that the lower the mean squared error metric, the lower the error and this implies the more accurate our model is at predicting ratings.

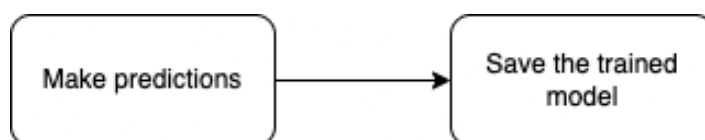
Through the iterations or epochs, the loss function (RMSE) is decreasing and so does the total loss, so the model is learning. The total loss is the function that computes the distance between the current output and the expected output. Although it is not a great result, the model is learning, and it can make better predictions than its antecessor because it uses a new feature.

Epoch	RMSE	Total loss
1	5.5432	30.7271
2	4.5326	19.6915
3	4.7889	22.9335
4	3.6145	15.0592
5	3.2609	13.4362
6	2.7124	9.5754

Table 10: Training results of ranking stage

5.4.2.6. Making predictions and saving

Last step to complete the model is to make predictions and save it for later use.



To make a prediction it is needed to users as an input, for example: 42 and 4524. The resulting output of this query is: 3.6475356

5.4.2.7. Conclusions

The recommender has evolved since its beginnings, being able not to work with users but also with ratings, learning and predicting them. This is clearly a step forward the final model when it uses more features.

At this point the recommendations, or predictions, made by the model are still quite irrelevant and impersonalized as the information used by the algorithm is limited. It only works with the rating between users, but knows nothing about their profile, activity, or traits. In a really large database, it could lead to a more significant predictions as it can eventually learn the “sociability” of users, treating them as a black box with no extra information but with an idea of how well two users would interact based only in their past activities and history to other users.

Even though at this stage the recommender has potential, there’s no reason to stop here because the potential hidden in a cross-ranking model is much more flexible, powerful and trustful. This kind of model is what it is explained in the next and final stage.

5.4.3. Cross-ranking

The final stage of the recommender. At this point is where all the elements are combined to produce a powerful and more advanced algorithm that uses users features as well as the rating of the interactions. This model is much more complex and requires more detail than its previous versions, moreover, in this section it is explained the resolution of the problem when converting the saved model into a *Unity* compatible format, stated in the section [5.7.2.2 Using the recommender in Unity](#).

Here it is illustrated an abstraction of the steps of the cross-ranking implementation developed:

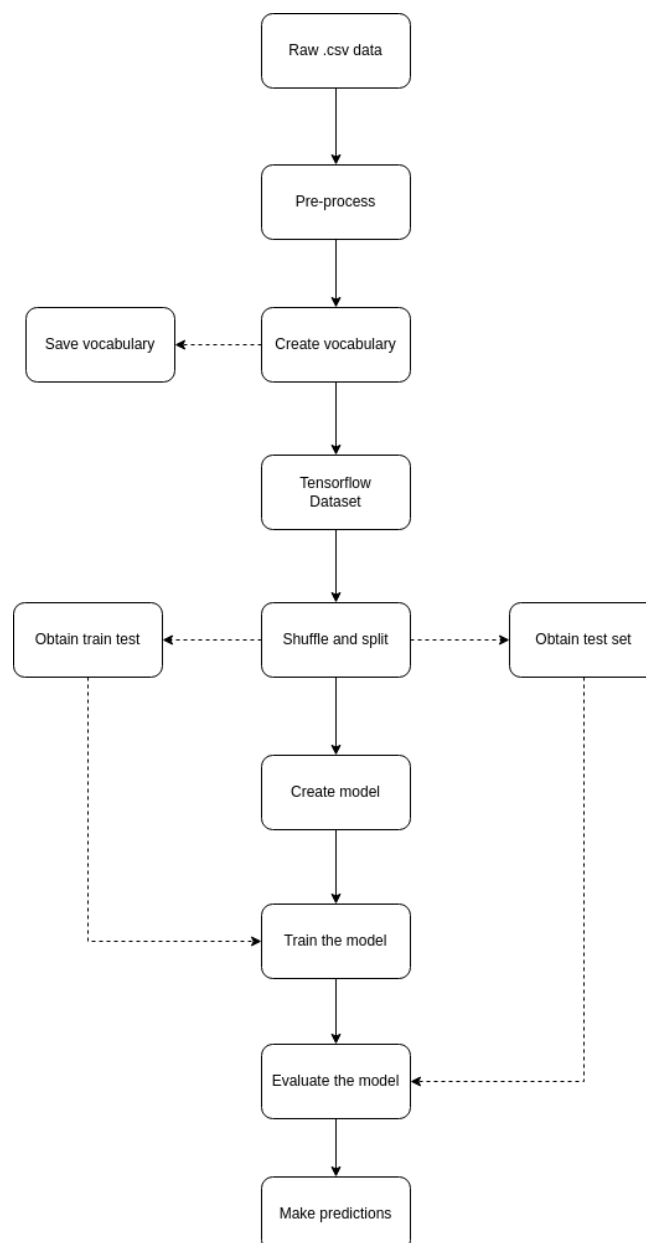


Figure 31: Diagram of the implementation of cross-ranking

Here it is illustrated an abstraction of the steps after the model is created till the final use inside the game simulation:

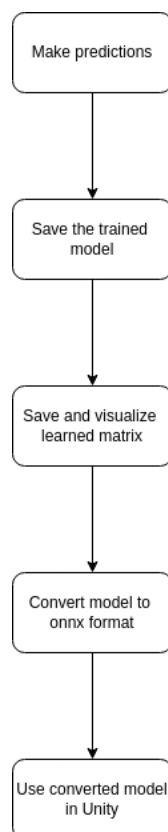


Figure 32: Diagram of following steps after the creation of the cross-ranking model

The first part resembles to the steps of the other stages of the recommender but differs in implementation and in order in some steps. The creation of the vocabulary is created in an early stage because it was needed to pre-process the data differently.

Note that this order and implementation is not the best solution and does not pretend to be like this. This way of solving the problem is the result of a lot of searching of the alternatives to use the ideal layers, seeing that the solutions to keep them are out of my knowledge. So, analyzing what these layers are, what are they used for, and when they are used, it is implemented a workaround to use them externally, so the model does not have to incorporate them in its insides, being able then to be convertible to the *onnx* format used by *Unity*.

More details of this are explained in the section [5.4.3.2. Data parsing and pre-processing](#).

The second part is new respect to the other stages as the cross-ranking model was the only one exported and tested in *Unity*.

5.4.3.1. Model definition

The model no longer uses the two-tower architecture used till now. It uses a combination of concatenated layers to finally converge to a single dimension layer to produce the output. To follow the analogy of the tower, it uses a multi-tower architecture where each one is the feature used as input, so it has 10 towers as input to then pass the values to the internal layers.

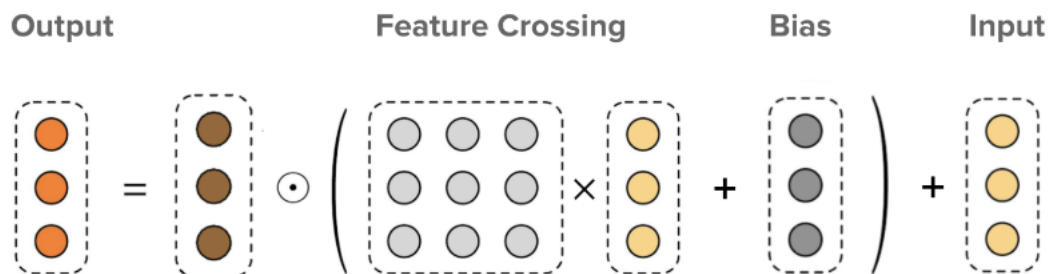


Figure 33: Illustration of the cross-ranking model

5.4.3.2. Data parsing and pre-processing

By default, there is no need to manually pre-process the data as the pre-processing layers do it, this is reflected in the previous stages where the raw data is directly converted to a TensorFlow Dataset without any previous modifications.

This could continue to be possible if the model is kept in the python environment, but as it is mandatory to be converted into another format such as onnx or TensorFlow Lite, rather than keeping the original saving format of TensorFlow to be used inside *Unity*, it is not an option anymore. May I repeat that this pre-processing step might not be the optimal nor the best solution, but it is the solution adopted for this problem.

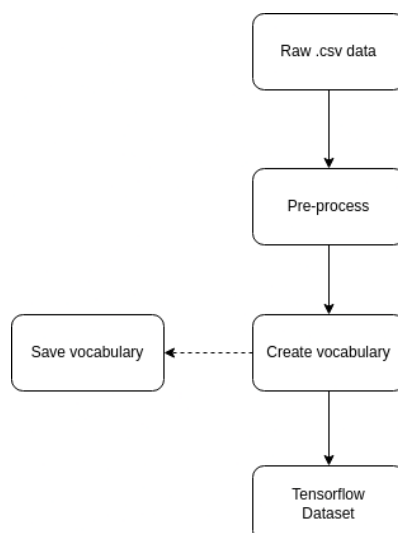


Figure 34: Diagram of data pre-processing in cross-ranked

As stated in the following section [5.5 Unity integration](#), the aimed format is onnx format and there are some limitations when converting a model to it.

The model was using pre-processing layers that are available in the newest versions of the library and the onnx format does not support them yet. To overcome this problem, the solution was to do manually the pre-process the data with the same layers that would be using the model but doing it before the model is created. This way the model receives the data already preprocessed and does not have to do it internally.

This solution emerged new problems: The dataset is modified and codified in numbers instead of the human-readable values. To have a better visualization of the problem, here is an example of how the dataset fed to the algorithm changed from this:

User_id	...	Language	Login_time	Character	Mode	Ranking
32	...	Spanish	425	Hanzo	Healer	1353

Table 11: Raw data before the manual pre-processing

To this:

User_id	...	Language	Login_time	Character	Mode	Ranking
32	...	3	425	12	1	1353

Table 12: Raw data after the manual pre-processing

This conversion is typically made internally by the model so there is no problem but now, it is needed to keep track of which feature value maps to which number.

To ensure that the indices used by the algorithm are the same that will be used to later feed it to produce predictions, there is an extra step when pre-processing the data. This step is to save the vocabulary and its individual index generated by the pre-processing layers to a file, to later be used when making predictions inside the game simulation.

Having this dictionary eases the conversion of natural, raw feature values such as “Damage” to its value, let’s say “1”, to minimize the modifications of the code already developed in the game and the data generator, acting like an adapter to talk to the recommender system.

Once all the data is pre-processed and saved, it is converted to a TensorFlow Dataset ready to be used by the model.

5.4.3.3. Data processing

The dataset is now processed as usual, splitting it to obtain the train set and the test set. There’s no difference in this step.

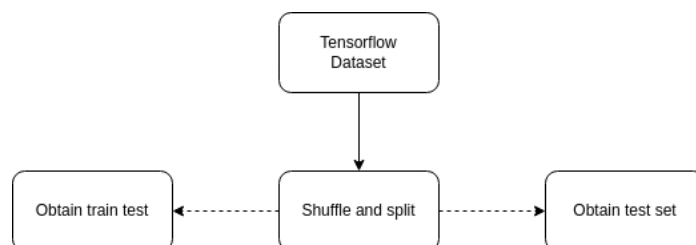


Figure 35: Processing diagram of the cross-ranking

5.4.3.4. Model creation

The model implements a combination of Sequential layers as inputs, one for each feature excluding the rating as it is not an input itself, where each feature is passed and then translated to a vector to be used on the following layers. The next layer is a Cross layer where all the inputs are mixed up and connected to combine them and use the correlations found. After that, there are the Dense layers, the regular hidden layers of any machine learning algorithm where all the weights and biases are modified while the algorithm is learning to finally converge to a single Dense layer to output the result.

The flow has no difference once the model is created, sharing the steps with the previous stages.

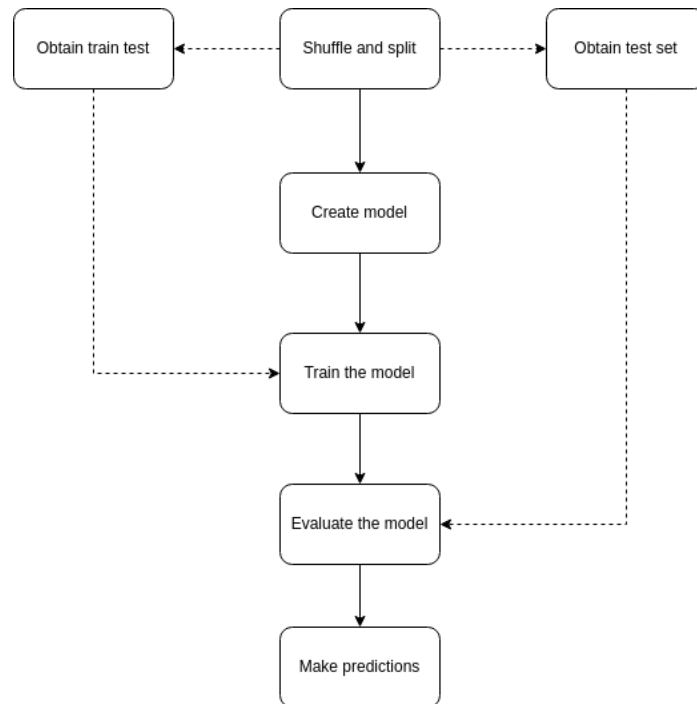


Figure 36: Diagram of model creation of cross-ranking

5.4.3.5. Training and evaluating

The training process is the same of the ranking stage. It uses the RMSE metric and to choose a better generation of the algorithm trained, it is tested with different dense layer's sizes and tested three times each model. The results presented here are not expected to be improving each iteration, because the modification of this values are made while the model is training and here is stated the final result of it, not the process of improvement as it can be seen in previous stages.

Dense layers with sizes [192, 192]

Iteration	RMSE	Total loss
1	0.7218	0.5211
2	0.6620	0.4383
3	0.5466	0.2985

Dense layers with sizes [128, 256, 256, 128]

Iteration	RMSE	Total loss
1	0.7320	0.5359
2	0.7371	0.5438
3	0.7747	0.6002

Dense layers with sizes [192]

Iteration	RMSE	Total loss
1	0.7642	0.5838
2	0.7317	0.5356
3	0.7088	0.5028

Dense layers with sizes [512, 512]

Iteration	RMSE	Total loss
1	0.6131	0.3763
2	0.6405	0.4101
3	0.6096	0.3718

Basing the decision of choosing a model based on the RMSE metric, the closest evaluated value to 0 is the highlighted in bold, the 3rd iteration of the model defined with 2 dense layers of 192 nodes each one. This is the model picked as the best one and from now on the references will be towards it.

5.4.3.6. Making predictions and saving

Querying for a recommendation has the same procedure as the other stages. The difference here is that instead of using one or two inputs, it is used the 10 inputs used for training that are: user id, interacted user id, age, gender, language, login time, character, character type, mode, and ranking. The inputs apart from the first one, user id, refers to the features of the interacted user.

An example would be:

Inputs: [5421, 12532, 24, Male, Spanish, 241, Hanzo, Damage, Quick, 2421]

Output: 4.29

And considering that the model takes pre-processed data, the “real” query would be:

Inputs: [5421, 12532, 24, 2, 35, 241, 19, 2, 4, 2421]

Output: 4.29

To try to understand the learning of the algorithm, it can be visualized the weight matrix of the ponderations of each feature to see if it actually makes sense and have correlation with the forced ratings that are produced with the data generator.

The interpretation is presented in a blue scale and the darker the tones the more importance the algorithm gives to this combination of features.

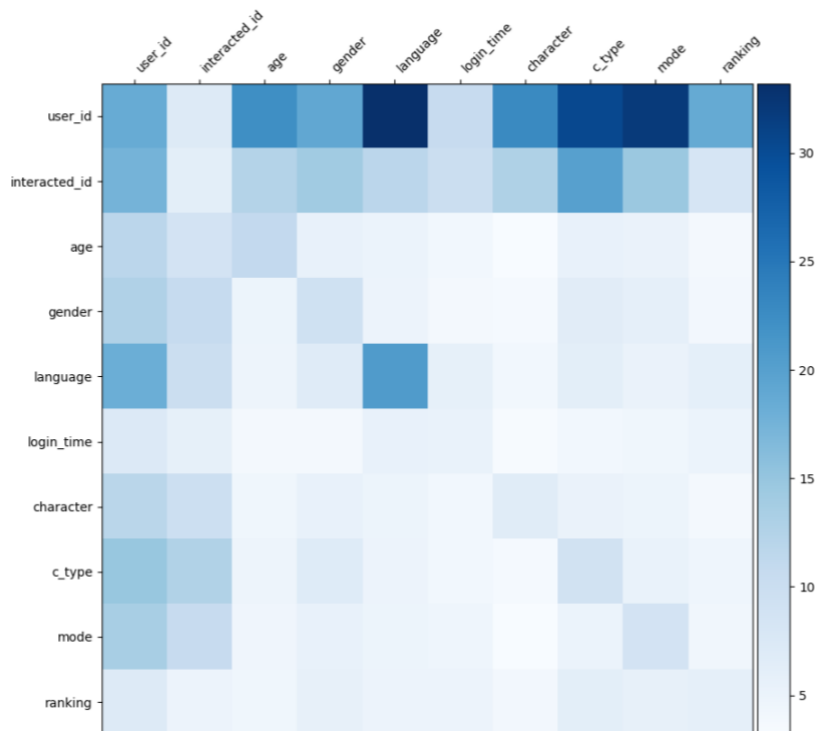


Figure 37: Visualization of internal weight matrix of cross-ranking

Analyzing the visualization of the factorization matrix, the model considers that language is a really powerful trait, as well as the mode and the character type.

They are more intense in the first rows and columns of the matrix where the ids of the users are located, this means that the algorithm have learned that each user has a “profile” with specific tastes and that to predict a rating, the user itself must also be considered and not focusing only on the combinations of the features, treating them as an unrelated separated thing from the user, although they themselves influences in the final prediction as it can be seen in the darker mid diagonal where each feature match itself.

This visualization confirms that the model is correct logic-wise. The dataset forced the language to be the strongest parameter, and it is also reflected in the matrix meaning that the recommender understood that. The other parameters with more importance such as *c_type* or *mode* are relevant when making a prediction.

Note that the dataset is generated randomly, so this randomness may have produced a dataset that is prone to have better rating when the feature *mode* satisfies the conditions implemented in the dataset or when the *login_time* does not satisfy the conditions of the generator, and this may be the reason the algorithm learned that way.

5.4.3.7. Conclusions

The final model can be presented as a success, it can produce a reasonable rating using an input that can be satisfied inside a game environment. The visualization of its’ internal matrix shows that the learning process has been made in an expected way, although there are some values that for some reason are not as important as others and in the dataset, they have the same weight, but it could be attributed to the fact that the dataset is generated randomly.

On the other hand, there’s a downside with this implementation because now the game simulation needs the adapter to convert the natural, human-readable raw data to the model pre-processed raw data. In terms of code complexity does not scale too much and either does it in terms of efficiency but it needs an extra step that may be not required using another solution.

5.5. Unity integration

Once the model is saved in TensorFlow format (.pb), it is ready to be inferred. The next step is to export it to Unity where it is placed inside the game to be used there. To do so there are some alternatives described below.

5.5.1. TensorFlow Lite for Unity

To understand this library first it is needed to explain TensorFlow Lite.

TensorFlow Lite is a mobile library for deploying models on mobile, microcontrollers and other edge devices. The target devices are Android, iOS and Raspberry Pi and it provides a set of tools that enables on-device machine learning by helping developers run their models on these devices. [24]

TensorFlow Lite for Unity is the port of the TensorFlow Lite to Unity and C#. This port is made by *asus4 (Koki ibbukuro)* a freelance technical designer. This port is an open-source project incubated by an augmented reality company that works with *Unity* and machine learning, and although it works well with *Unity* the pitfalls of using it can go from no further support to making the project private.

[\[25\]](#)

The model made in this project is convertible to TensorFlow Lite format, but it is not the alternative selected to be final as the code complexity rises, the library is maintained by an external developer that any time may stop giving support to new features, and the documentation is limited to examples.

5.5.2. TensorFlowSharp

TensorFlowSharp is a runtime to run existing models, and is mostly a straight binding to the underlying TensorFlow runtime. It is still experimental. While it is possible to embed trained models into Unity games, Unity Technologies does not officially support this use-case for production games currently. Both the maintainers of the library and *Unity* state that no guarantees are provided regarding the quality of experience.

Although it can be an alternative, there are a lot of potential problems when using an experimental library and being the development in such early state made this library not an option to consider implementing the model inside *Unity*. [\[26\]](#)

5.5.3. Unity Barracuda

Unity Barracuda is the standard and official way to proceed when using machine learning models generated outside the *Unity* environment.

The game engine has its own artificial intelligence libraries, called machine learning agents, and they can be trained and used without leaving the engine environment, but they can only be trained using the reinforcement-learning method, that keeping it simple, is trial-error learning, so it is not suitable for the scope of this project.

Knowing the limitations of its own library, *Unity* provides Barracuda, and as explained in section [5.2.5 Unity Barracuda library](#), it is a lightweight cross-platform neural network inference library for *Unity* and it is the option used in this project to implement the generated model into the game simulation.

To do so, there is a previous step to do because Barracuda works with onnx format models, so the saved model by TensorFlow needs to be converted to onnx format. Once it is converted it is ready to be used in the *Unity* project. This step is the responsible of transforming the raw human-readable data to numeric raw data, because as described in previous sections some latest layers are not ready to be converted to onnx format.

The implementation of the recommender inside the game can be found in the “Recommender.cs” file of the *Unity* project. [\[27\]](#)

5.6. Game simulation

The idea of this project is to present the algorithm as game ready, as a feature that can be implemented in an online game, so it is a must to present it in at least a game simulation. The ideal showcase would have been to implement it in a real game with real data, but this approach is out of the scope of this work, and it is finally presented in a simplified simulation of a game.

The game simulation consists in the main menu interface that could be inserted in almost any online game. The player can navigate to the *find team* screen, the *last match summary* screen, and visualize some information about the project.

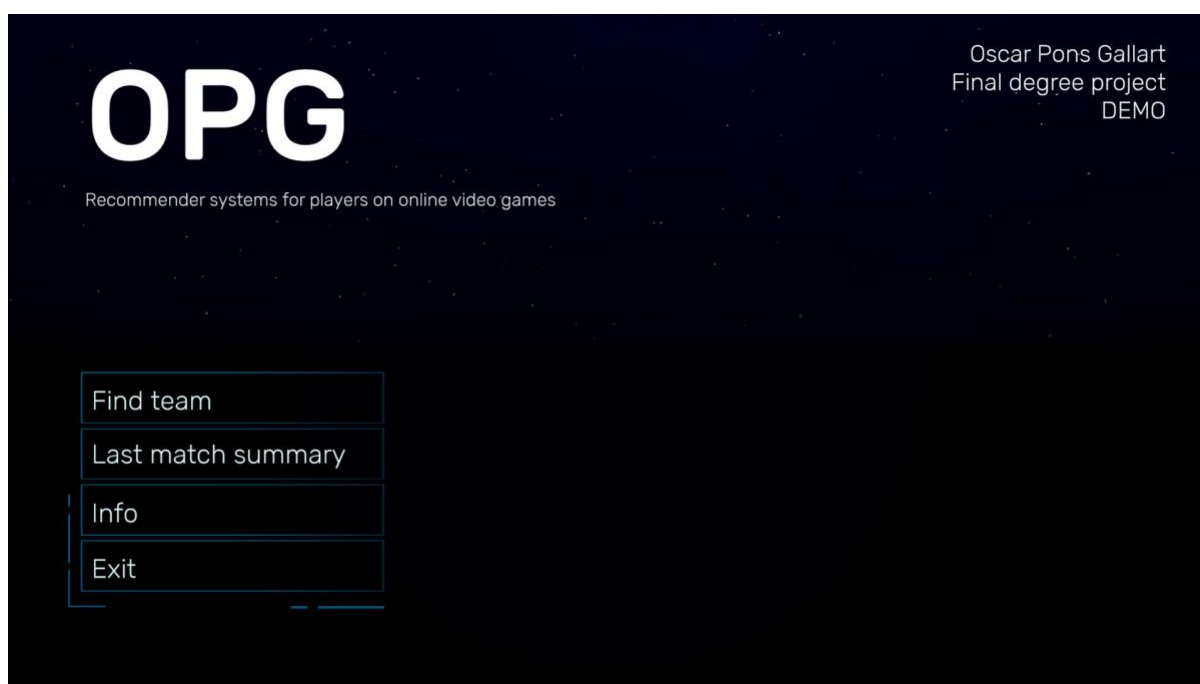


Figure 38: Game simulation - Main menu

The numbers shown below the icons of a dart board, a heart, a dollar sign and three rectangles in the screens when showing “game data” are fictional and irrelevant for the project. They are props that try to simulate possible real game data and must not be considered. They are there to simulate as close as possible the screen of the hypothetical game.

			
13568	12422	456222	19132
6607	6292	53704	26256
2490	17763	213434	98063
18066	12557	176928	35483
11012	6545	632507	62641

Figure 39: Game simulation - Decorative props data

The relevant number is the last one, the one below the yellow star icon, highlighted in a light-yellow background. This is the recommender rating for each user, from the point of view of the current user highlighted in blue background.

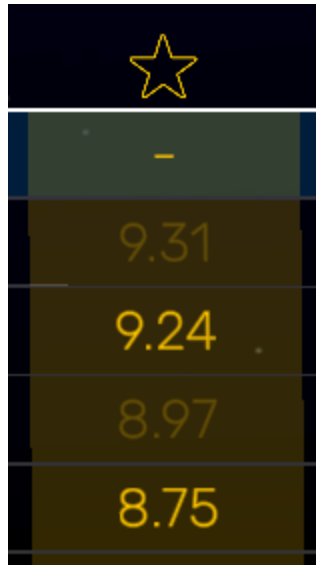


Figure 40: Game simulation - Recommender rating output

The demo also implements the feature to differentiate between online and offline players. This differentiation is with the text color of the row. Online players have white text and offline players have gray text.

[24133] Isaura Coronado Royo.	13047	5615	750494	46460	9.24	⊕
[66024] Michael Peters	15905	114	652960	4047	8.97	⊕

Figure 41: Game simulation - Differentiation of online and offline players

The game is seen with the point of view of the user 73, Leyre Elías Verdejo, highlighted with a blue background in each of the screens.

5.6.1. Find team screen

This is the implementation of the main purpose of the recommender system, where it can be seen the power of it. This screen represents the list of all the players in the game, only visualizing a few of them, ordered by the rating the recommender has given to them.

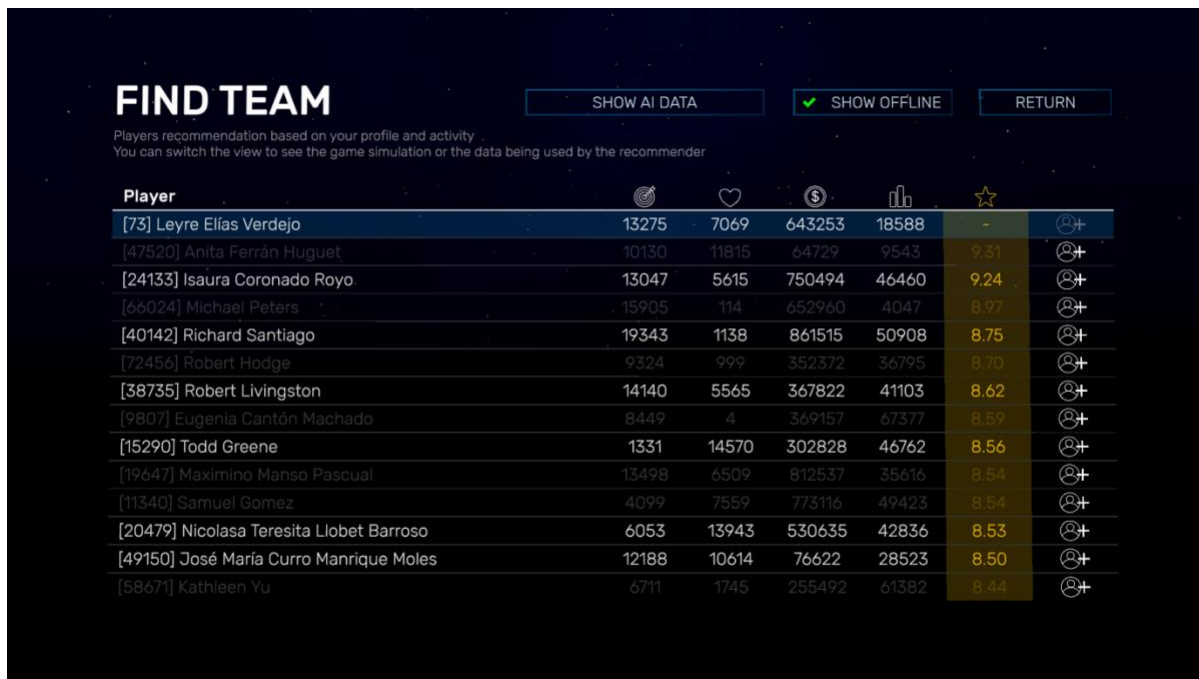


Figure 42: Game Simulation - Find team screen with game data

To help to understand what the algorithm is seeing and what are the user traits, as a helper tool (not expected in the real game) the view can be switched to see the data the recommender is using.



Figure 43: Game simulation - Find team screen with AI data

In this screen it is implemented two user stories:

- As a player I want to be able to see only the online players so that I can play now with them



Figure 44: Game simulation - Find team screen: show offline user story

- As a player I want to be able to send a friend request to any player so that we can play together



Figure 45: Game simulation - Find team screen: add friend user story

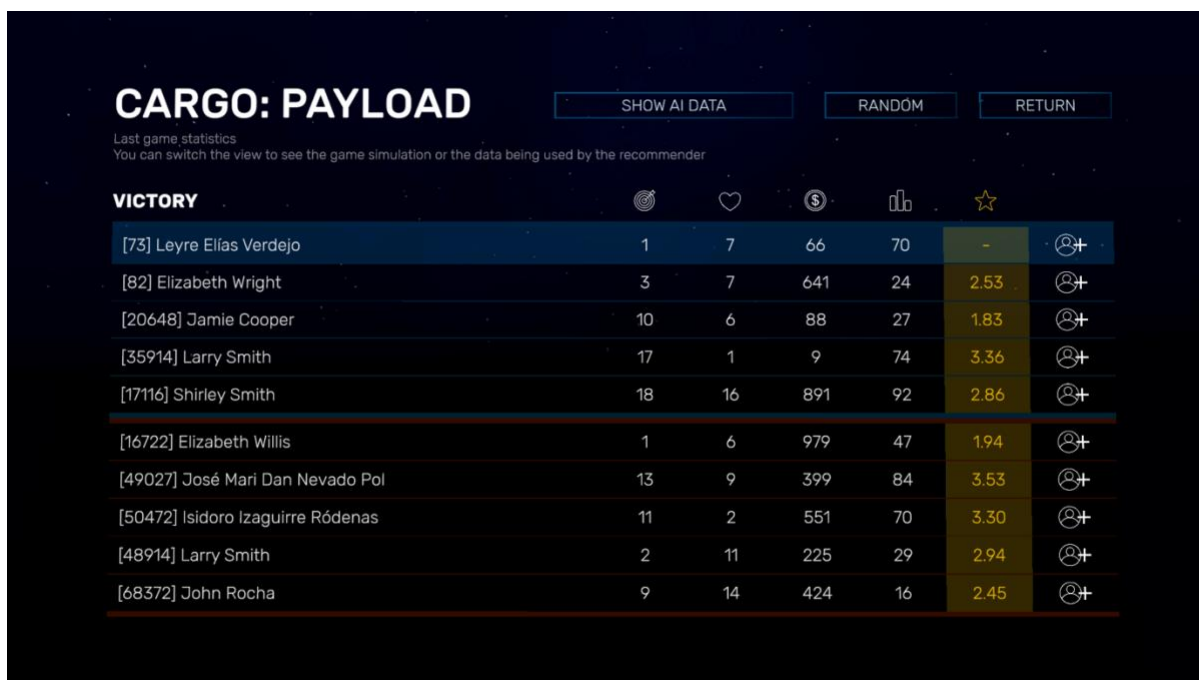
The first one can be executed pressing the button checkbox “Show offline” and it switches the view to see the desired output list.

The second one can be executed by pressing the button at the side of each player, there is then a visual feedback confirmation in the player’s name that shows that a friend request has been sent.

Note that these user stories are fictional, and they do not make any changes a part from the visual feedback of it.

5.6.2. Last match summary

This can be another great example of where the recommender can be used although in this case the power of it is not as meaningful as inside the *find team* screen, because the players of any given match are likely to be random in a certain degree but having similitudes in the scoreboard and abilities of the game, so it is a balanced match. This screen represents the end of a match where all the players are presented in a two lists, allies, and enemies, with each one’s statistics of the match summarized, and the rating of the algorithm at last.



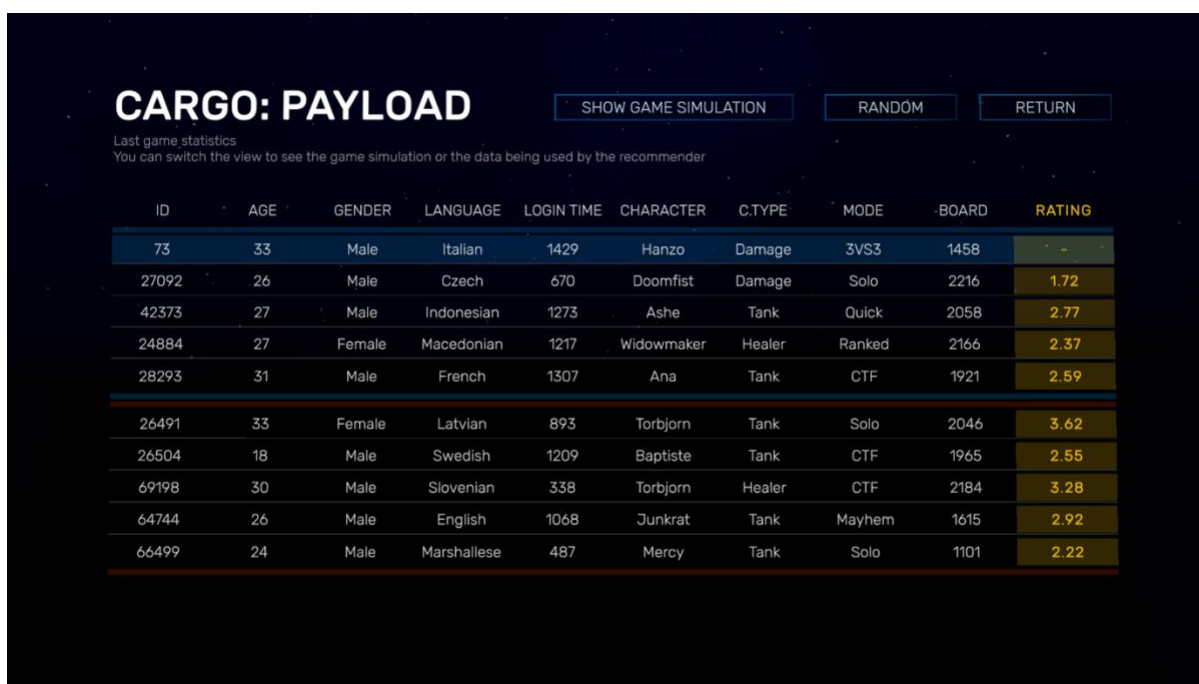
CARGO: PAYLOAD		SHOW AI DATA	RANDOM	RETURN		
Last game statistics You can switch the view to see the game simulation or the data being used by the recommender						
VICTORY						
[73] Leyre Elías Verdejo	1	7	66	70	-	⊕
[82] Elizabeth Wright	3	7	641	24	2.53	⊕
[20648] Jamie Cooper	10	6	88	27	1.83	⊕
[35914] Larry Smith	17	1	9	74	3.36	⊕
[17116] Shirley Smith	18	16	891	92	2.86	⊕
[16722] Elizabeth Willis	1	6	979	47	1.94	⊕
[49027] José Mari Dan Nevado Pol	13	9	399	84	3.53	⊕
[50472] Isidoro Izaguirre Ródenas	11	2	551	70	3.30	⊕
[48914] Larry Smith	2	11	225	29	2.94	⊕
[68372] John Rocha	9	14	424	16	2.45	⊕

Figure 46: Game simulation - Last match summary screen with game data

The player has just played a game and may have an idea of the other players. Analyzing the situation when he got along with another one or another player drawn his attention, the usual step could be to go to the other's profile, check if they do match and after confirming, send a friend request if the confirmation is successful. Here it is presented another scenario, the same conditions happened, the player wants to keep playing with another player and now he can rely on the algorithm criteria to see if they would be good partners.

There is also another scenario where the player sees a high rating of one player that did not draw his attention at the beginning, and he sees the rating of the recommender and he relies on it and sends a friend request as he knows that the algorithm knows his criteria and is trustful.

To help to understand what the algorithm is seeing and what are the user traits, as a helper tool (not expected in the real game) the view can be switched to see the data the recommender is using.



CARGO: PAYLOAD SHOW GAME SIMULATION RANDOM RETURN

Last game statistics
You can switch the view to see the game simulation or the data being used by the recommender

ID	AGE	GENDER	LANGUAGE	LOGIN TIME	CHARACTER	C.TYPE	MODE	BOARD	RATING
73	33	Male	Italian	1429	Hanzo	Damage	3VS3	1458	-
27092	26	Male	Czech	670	Doomfist	Damage	Solo	2216	1.72
42373	27	Male	Indonesian	1273	Ashe	Tank	Quick	2058	2.77
24884	27	Female	Macedonian	1217	Widowmaker	Healer	Ranked	2166	2.37
28293	31	Male	French	1307	Ana	Tank	CTF	1921	2.59
26491	33	Female	Latvian	893	Torbjorn	Tank	Solo	2046	3.62
26504	18	Male	Swedish	1209	Baptiste	Tank	CTF	1965	2.55
69198	30	Male	Slovenian	338	Torbjorn	Healer	CTF	2184	3.28
64744	26	Male	English	1068	Junkrat	Tank	Mayhem	1615	2.92
66499	24	Male	Marshallese	487	Mercy	Tank	Solo	1101	2.22

Figure 47: Game simulation - Last match summary with AI data

To add exploration of the recommender, there is the *random* button to generate a new random list of users. This can be thought as a new game with another users and acts as a helper tool to demonstrate the algorithm, it is not expected in the real game.

5.7. Problems and solutions

This section reviews in detail the problems encountered, and the approach used to solve them.

5.7.1. Dataset

One of the problems is that the data available online is not suitable for the recommender system. After a lot of research, many websites offer datasets or APIs that gather information about games but almost none of them were specific, useful, and relevant user information. With this kind of data, the recommender system does not have the guarantee to learn how the users interact with each other.

Proper data is hard to retrieve for external application users or developers and easy to retrieve for the game developers themselves as they can place in-game events that collect relevant information, therefore the solution executed for this problem seems a good approach.

So, following the risk and contingency planning, the solution executed is the same as the proposed one in the table: Adapting the data. In order to do this, the data is generated with the set of fields that provide useful information for the system, giving the necessary number of entries, more freedom to consider other parameters combinations and the on-demand availability of the dataset. This solution is explained in detail in [5.3 Dataset](#) section.

5.7.2. Implementation of the recommender system model

5.7.2.1. Knowledge limitations

The main problem while implementing the model was the huge possibilities and combinations of how a model can be defined. The best recommender systems are built by engineers that are specialized in this field and have a vast knowledge of the topic, being able to represent the same model in such a variety of styles, and this freedom of representing the model evokes insecurity in newbies developers.

This project contemplates the part of acquiring new skills and learning new technologies and although this learning has been accomplished, there are a lot of features of the technology that are out of the reach with my current level, and the degree of “correctness” of the solution proposed may be far from perfect.

So, the implementation of the algorithm is based in beginner level examples and documentation, where the new developer acquires the basic skills to enter the big world of deep learning and recommender systems.

5.7.2.2. Using the recommender in Unity

The recommender final step was to be presented inside a simulation of a game made with *Unity*. To use the final trained model into the game engine, it was needed to use some libraries that adapted the model generated by *TensorFlow* to another format and then use it inside the game.

This step had the project stuck for weeks. As stated in the previous section, there are a lot of ways to propose the recommender and at first it was aimed to the most straight forward one. The problem it has been encountered is that two pre-processing layers that fed the model are too new to have a conversion to the *Unity*'s format and a workaround has been needed to overcome this difficulty.

It has been tested three different libraries for using the model inside the game with no initial success. Also, it has been proposed the idea of developing the game demo with the python *PyGame* library, so no conversion would be needed. Finally, a preprocessing of the data outside the model did the trick and the model is final exportable to *Unity*.

5.7.2. Impact to the project

The first problem experienced has no negative impact to the development of the project more than the time invested in searching for a dataset that is useful.

The solution of a dataset being generated on demand offers more possibilities than gathering an already-collected immutable online dataset. Moreover, this enriches the project because the feeding part of the algorithm can be easily manipulated to satisfy new needs.

The lack of knowledge of the technology do suppose a major impact to the development of it. This had an impact in the iterations as the validation stage had to be replaced with both learning and development of the system. This means that although the system is functional, with personalized features, the mathematics behind it are not checked nor validated, so the solution may not be optimal or may be computational inefficient.

Not being able to successfully export the final model to *Unity* has supposed a major impact to the project. The roadmap has been frozen and branched at this point, investing a lot of time and effort to look for a solution that arrived in a late stage. The solution achieved may not be the optimal but overcomes the problem of using new technologies that aren't yet implemented in all the necessary steps of this project.

6. Conclusions

The recommender system finally produced works in a predictable way, meaning that the expected output rating that the algorithm makes is correct considering how the dataset is produced. It values, with some differences, the same things that the dataset has forced to be valued so, analyzing the recommender system it can be said that it is a success and is prepared to be used in production.

This statement emerges one question: why the need of a recommender system then? Why is not possible to define a criteria, and base the rating of the recommended players on that? In a certain way is what this project have done when generating the dataset; it defined a rating criteria of how a good interaction is valued and then it fed the algorithm with that dataset. Using common sense, it is correct, and it could be implemented to the game without the need of a recommender. This question has a simple answer: there is no master criteria that works with all the players, everyone is different. With that in mind, what developers should do? Define rules and apply some of them to some players and the other rules to the other players? It could be a solution, but I may dare to say that it's not the best one.

The recommender system proposed in this work adapts to the players and although it is conditioned with the dataset fed with, it learnt from them and gives personalized results that may satisfy more than an unbreakable rule. In this project the rating valuation is fixed but in the real world there are no rules to write, everyone gives more value to certain things and this system would adapt to them with much less effort than trying to understand the player behavior through deep analysis and data interpretation, easing the life of developers as they do not need to be concerned about this.

Considering that the algorithm has “only” been trained with 4 million entries and 75 thousand players in a one go, it gives a great result, and shows the true potential of using it in a recurrent way, constantly learning from the players, incorporating new ones, and correcting its weak points. With a large database and a team improving the implementation its performance would probably be even better.

In terms of code complexity and if it is suitable to embed a recommender system inside a game made with *Unity*, it has been proven that is possible, and that there is no performance pitfall that does not allow the proper execution of the game. The game simulation made for this project lacks a real cloud database where most of the computations could be made, to minimize the wait time of the player, and also lacks of parallel execution that can improve the player experience and the speed of execution.

This project has successfully made a minimum valuable product of a recommender system that can be embedded into a *Unity* game and encourages developers to implement it inside their games to improve the user experience and grow a better community as the number of possibilities expand in a lot of areas; from implementing this player recommender system to making personalized game chapters, create matches of recommended players, or to generate a helper-buddy program, for example, and the list can go on.

Bringing players together is one of the objectives of this project and using a recommender system can be a great solution to do so, wiping out the frustration that players feel when they search for teammates, hoping that they will find them in lucky strike in a random match or in an external website. This solution has focused on easing and simplifying this task as much as possible, enabling team finding inside the game and with a recommendation made by an artificial intelligence, this time used not to sell something but to bring people together.

7. Next steps

Although the recommender system is ready to be implemented in a game, it is the first release and have a lot of things to improve.

Next steps of this project could be to enable the model to do recurrent training, keeping it updated and constantly improving and learning from new inputs. Currently, the system is static, it is delivered as a fixed product that can be embedded to the game and it would be a good point to make the appropriate modifications for it to create snapshots and keep training. That way it would be a step closer to what an ideal solution would be for a real game.

It also would be really eye-opener if it could be tested in a real game environment, with real player data to see if the system responds well and keep up the results made in the safe environment. Inside a controlled environment the results tend to be both more predictable and correct but, in a real scenario it may give unexpected predictions or answers and a big improvement of the implementation of it might be made.

8. References

- [1] Cheuque, G., Guzmán, J., & Parra, D. (2019). Recommender systems for online video game platforms: The case of steam. *The Web Conference 2019 - Companion of the World Wide Web Conference, WWW 2019*, 763–771. <https://doi.org/10.1145/3308560.3316457>
- [2] Joshi, R., Gupta, V., Li, X., Cui, Y., Wang, Z., Ravari, Y. N., Klabjan, D., Sifa, R., Parsaeian, A., Drachen, A., & Demediuk, S. (2019). A Team Based Player Versus Player Recommender Systems Framework for Player Improvement. *ACM International Conference Proceeding Series*, 7. <https://doi.org/10.1145/3290688.3290750>
- [3] Lenhart, A. (n.d.). *Video Games, Teen Boys and Building Social Skills and Friendships* / Pew Research Center. Retrieved March 20, 2022, from <https://www.pewresearch.org/internet/2015/08/06/chapter-3-video-games-are-key-elements-in-friendships-for-many-boys/>
- [4] Fishman, A., MSW, LSW, Clinician, & Response Center for Teens. (n.d.). *Video Games Are Social Spaces: How Video Games Help People Connect* / ResponseCenter. Retrieved March 22, 2022, from <https://www.jcfs.org/response/blog/video-games-are-social-spaces-how-video-games-help-people-connect>
- [5] Wiederhold, B. K. (2021). Kids Will Find a Way: The Benefits of Social Video Games. *Https://Home.Liebertpub.Com/Cyber*, 24(4), 213–214. <https://doi.org/10.1089/CYBER.2021.29211.EDITORIAL>
- [6] LimeLight. (n.d.). *Market Research: The State of Online Gaming – 2019*. Retrieved March 25, 2022, from <https://www.limelight.com/resources/white-paper/state-of-online-gaming-2019/>
- [7] WePC. (n.d.). *Video Game Industry Statistics, Trends and Data In 2022* / WePC. Retrieved March 25, 2022, from <https://www.wepc.com/news/video-game-statistics/>
- [8] *Single player games*. (n.d.). Retrieved March 25, 2022, from <https://store.steampowered.com/category/singleplayer/#p=0&tab=TopSellers>
- [9] *Esports Team Finder* / TeamTavern. (2017). Retrieved March 25, 2022, from <https://www.teamtavern.net/>
- [10] TeamFind. (2017). Team and Player Finding for CS:GO, LoL, Overwatch, RL, CoD, Dota 2 and Halo | Teamfind. <https://teamfind.com/>

- [11] Dealessandri, Marie (January 16, 2020). *What is the best game engine: is Unity right for you?*. GamesIndustry.biz. Gamer Network. Retrieved March 18, 2022, from <https://www.gamesindustry.biz/articles/2020-01-16-what-is-the-best-game-engine-is-unity-the-right-game-engine-for-you>
- [12] Rocca, Baptiste. (2019, June 3). *Introduction to recommender systems*. <https://towardsdatascience.com/introduction-to-recommender-systems-6c66cf15ada>
- [13] Roy, Abhijit. (2020, June 31). *Introduction To Recommender Systems- 2: Deep Neural Network Based Recommendation Systems*. <https://towardsdatascience.com/introduction-to-recommender-systems-2-deep-neural-network-based-recommendation-systems-4e4484e64746>
- [14] Google. (n.d.). *TensorFlow Recommenders*. Retrieved March 25, 2022, from <https://www.tensorflow.org/recommenders>
- [15] Walia, M. (n.d.). *Top 5 Open-Source Machine Learning Recommender System Projects With Resources/ Becoming Human: Artificial Intelligence Magazine*. Retrieved March 25, 2022, from <https://becominghuman.ai/top-5-open-source-machine-learning-recommender-system-projects-with-resources-50583c4007c3>
- [16] Crossing Minds. (n.d.). *What are today's top recommendation engine algorithms? | by Crossing Minds | ITNEXT*. Retrieved March 25, 2022, from <https://itnext.io/what-are-the-top-recommendation-engine-algorithms-used-nowadays-646f588ce639>
- [17] Google. (n.d.). *Introducing TensorFlow Recommenders — The TensorFlow Blog*. Retrieved March 21, 2022, from <https://blog.tensorflow.org/2020/09/introducing-tensorflow-recommenders.html>
- [18] Metodología “scrum” en “Agile”: ¿Qué es un “sprint” y cuánto dura? (2019, March 1). <https://www.bbva.com/en/scrum-methodology-what-is-a-sprint/>
- [19] PyTorch. (2022, February 23). *Introducing TorchRec, a library for modern production recommendation systems | PyTorch*. <https://pytorch.org/blog/introducing-torchrec/>
- [20] Project Awesome. (n.d.). *:video_game: Awesome Game Datasets | Curated list of awesome lists | Project-Awesome.org*. Retrieved May 13, 2022, from <https://project-awesome.org/leomaurodesenv/game-datasets>

- [21] Faker. (n.d.). *Welcome to Faker's documentation! — Faker 13.11.1 documentation*. Retrieved May 13, 2022, from <https://faker.readthedocs.io/en/master/index.html>
- [22] Numpy. (n.d.). *NumPy*. Retrieved May 13, 2022, from <https://numpy.org/>
- [23] Blizzard. (n.d.). *Heroes - Overwatch*. Retrieved May 13, 2022, from <https://playoverwatch.com/es-es/heroes/>
- [24] TensorFlow. (n.d.). *TensorFlow Lite | ML for Mobile and Edge Devices*. Retrieved June 20, 2022, from <https://www.tensorflow.org/lite>
- [25] Koki Ibukuro. (2022). *asus4/tf-lite-unity-sample: TensorFlow Lite Samples on Unity*. <https://github.com/asus4/tf-lite-unity-sample>
- [26] lISourcell. (n.d.). *Unity_ML_Agents/Using-TensorFlow-Sharp-in-Utity-(Experimental).md at master · lISourcell/Unity_ML_Agents*. Retrieved June 18, 2022, from [https://github.com/lISourcell/Unity_ML_Agents/blob/master/docs/Using-TensorFlow-Sharp-in-Utity-\(Experimental\).md](https://github.com/lISourcell/Unity_ML_Agents/blob/master/docs/Using-TensorFlow-Sharp-in-Utity-(Experimental).md)
- [27] Unity. (2022, March 1). *Introduction to Barracuda | Barracuda | 3.0.0*. <https://docs.unity3d.com/Packages/com.unity.barracuda@3.0/manual/index.html>
- [28] Harrison Kinsley, & Daniel Kukiela. (2021). *Neural Networks from Scratch*. <https://nnfs.io/>