Universidade de Évora - Escola de Ciências e Tecnologia

Mestrado em Engenharia Informática

Dissertação

Supergame, a system with data collection to support game recommendation

Francisco Ferreira Canhão Rodrigues

Orientador(es) | Francisco Manuel Coelho

Évora 2023

Universidade de Évora - Escola de Ciências e Tecnologia

Mestrado em Engenharia Informática

Dissertação

Supergame, a system with data collection to support game recommendation

Francisco Ferreira Canhão Rodrigues

Orientador(es) | Francisco Manuel Coelho

Évora 2023

A dissertação foi objeto de apreciação e discussão pública pelo seguinte júri nomeado pelo Diretor da Escola de Ciências e Tecnologia:

Presidente	José Saias (Universidade de Évora)
Vogais	Francisco Manuel Coelho (Universidade de Évora) (Orientador) Irene Pimenta Rodrigues (Universidade de Évora) (Arguente)

Évora 2023

UNIVERSIDADE DE ÉVORA

Escola de Ciências e Tecnologia

Departamento de Engenharia Informática

Supergame, a system with data collection to support game recommendation

Francisco Ferreira Canhão Rodrigues

Supervisor *Francisco Manuel Gonçalves Coelho*

Engenharia Informática

Dissertation

January 5, 2024

In the world you will have tribulation. But take heart; I have overcome the world.

Preface

In the ever-evolving world of video games, the pursuit of fun and engagement is an endless journey. As we all experienced the global shift during the COVID-19 pandemic, video games became a more prominent part of our lives, offering a plethora of options. Within this digital realm, filled with choices and opportunities, the concept of Recommendation Systems emerged as a practical solution.

My exploration into Recommender Systems for video games was driven by a simple goal: making the gaming experience more enjoyable. With countless casual games available, each with its own unique charm, the challenge was to help players find the right games for them, enhancing their enjoyment and engagement.

This journey had its fair share of challenges. Initially, I set out on an ambitious project, envisioning a comprehensive platform developed in Godot—an endeavor that required a deep understanding of game development, programming, and game engines. As the project unfolded, it became clear that some adjustments were needed. The path I followed led me to two critical areas. First, I delved into the world of Recommender Systems, exploring their complexities and practical applications in video games. Second, I developed an application within Godot, a testament to the learning, growth, and adaptability that defined this journey.

As a lifelong gaming enthusiast, I've witnessed the evolution of video games into a cultural phenomenon. Combining this passion with the rising relevance of Recommender Systems during the pandemic fueled the inspiration for this thesis. My goal was simple: to simplify the game selection process and help users discover games that match their preferences.

In the pages ahead, we'll dive into the realm of Recommendation Systems in video games, exploring their theoretical foundations and practical applications. While my initial vision included user testing and feedback, this thesis focuses on understanding Recommender Systems and their potential impact on the gaming world.

Acknowledgements

I would like to express my deepest gratitude to professor Francisco Coelho, whose guidance and expertise have been invaluable throughout this journey. Your mentorship has illuminated my path and shaped this thesis in multiple ways.

To my family, the cornerstone of my life, I owe a debt of gratitude that words alone cannot encompass. Your unwavering support, both emotional and practical, has been the bedrock upon which I built this thesis. Your encouragement, understanding, and sacrifices made it possible for me to pursue this endeavor. You've been my constant source of inspiration, reminding me of the importance of perseverance and the pursuit of knowledge. This work is a tribute to your belief in me.

To my friends, I extend my heartfelt appreciation for being pillars of strength during this journey. Your friendship, patience, and understanding have been a source of comfort and motivation. You've celebrated my successes, offered a shoulder during the challenging times, and shared in the excitement of each milestone achieved. Your presence in my life has made this academic pursuit all the more fulfilling.

This work would not have been possible without your collective support, and for that, I am truly grateful.

Contents

List of Figures

List of Tables

Acronyms

AI	Artificial Intelligence
ALS	Alternating Least Square
AR	Augmented Reality
ARPG	Action Role-Playing Game
BR	Binary Rating
CS	Cold Start
CB	Content-Based
CBRS	Content-Based Recommender System
CF	Collaborative Filtering
CFRS	Collaborative Filtering Recommender System
CIR	Continuous-Interval Rating
DFM	Deep Factorization Machine
DNN	Deep Neural Network
FM	Factorization Machines
FPS	Frames Per Second
GGG	Grinding Gear Games
GNN	Graph Neural Network
GUI	Graphical User Interface
HRS	Hybrid Recommender System
HUD	Heads-up Display
IBCF	Item-Based Collaborative Filtering
IBR	Interval-Based Rating
KBRS	Knowledge-Based Recommender System

KNN	K-Nearest Neighbours
KB	Knowledge Base
KBRS	Knowledge-Based Recommender System
LAN	Local Area Network
LR	Linear Regression
MDP	Markov Decision Process
ML	Machine Learning
MMO	Massively Multiplayer Online
NN	Neural Network
NPC	Non-Player Character
OR	Ordinal Rating
POE	Path of Exile
PvP	Player versus Player
QL	Q-Learning
RL	Reinforcement Learning
RM	Rating Matrix
RP	Recommendation Panel
RS	Recommender System
SCGRec	Social-Aware Contextualized Graph Recommender System
SVD	Singular Value Decomposition
UBCF	User-Based Collaborative Filtering
UE	Universidade de Évora
UI	User Interface
UX	User Experience
VR	Virtual Reality

Abstract

Supergame, a system with data collection to support game recommendation

Creating a recommender system for casual games

Recommender Systems are ever present in applications we use daily. Using Recommender Systems help users find what items best suit their interests, by providing them a customized experience. Machine Learning algorithms and processes are used to get models of user behaviour that base Recommender Systems (RSs) by predicting current preferences. In turn, accurate models contribute to increased user satisfaction. With advances in technology and easy access to better performing personal computers, video game design is more pronounced in smaller and/or independent video game studios now more than ever before. This shift has resulted in hundreds of video games being released every year. From the consumer point of view, trying all this games is not feasible. Our aim with this project is to design an application, using the Godot Engine, where several casual games are integrated with a system that recommends the "next game to play", based on the user's experience.

Keywords: Game Design, Recommender Systems

Sumário

Superjogo, um sistema com coleta de dados para apoiar recomendação de jogos

Criar um sistema de recomendação para jogos casuais.

Os Sistemas de Recomendação estão presentes nas aplicações que usamos diariamente, para ajudar a encontrar itens nos quais os utilizadores têm mais interesse através de uma experiência personalizada. Ao empregar algoritmos sofisticados de Aprendizagem Automática, estes usam características e comportamentos dos utilizadores para prever as suas preferências atuais oferecendo um apoio personalizado que aumenta a satisfação do utilizador. Hoje em dia, com avanços tecnológicos e acesso a computadores com melhor desempenho, os video jogos deixaram de ser exclusivamente criados por grandes estúdios para também serem criados em estúdios menores e mais independentes. Esta mudança levou a centenas de video jogos serem lançados todos os anos. Do ponto de vista do consumidor, jogar todos estes jogos não viável. O objectivo deste projecto é desenhar uma aplicação usando o Godot, onde possamos integrar vários video jogos casuais e construir um sistema que recomende jogos com base na experiência do utilizador com a plataforma.

Palavras chave: Design de Jogos, Sistemas de Recomendação

1 Introduction

Playing video games has become one of the most popular hobbies worldwide. With the lockdown, the video game industry has emerged as the largest entertainment sector in terms of revenue, making it a global phenomenon. As of 2021, the industry's revenue exceeded 174,000 million euros, surpassing the combined revenue of the book, movie, and music industries [Liq]. The impact of the Covid-19 pandemic further fueled the industry's growth, as people turned to video games to cope with boredom and stress during periods of quarantine and isolation. For example, Steam, a prominent video game distribution platform, has been at the forefront of this boom. In 2021 alone, more than 10,000 video games were released on Steam, and the platform has amassed a staggering user base of over 3 billion gamers worldwide [How23]. With over 120 million players online and more than 25 million concurrent players daily, Steam has become a thriving hub for the gaming community. This growth, however, is not as pronouced in 2023.

Over the years, a noticeable shift has occurred in the video game industry, with an increasing number of games being developed by smaller studios and teams consisting of five or fewer people. This trend has emerged in contrast to the past when AA and AAA companies dominated game development. Several factors contribute to this shift.

One significant factor is the improved accessibility of personal computers, which have become more powerful and readily available [hp21]. This accessibility empowers smaller teams to create games without the need for extensive resources and investments in high-end hardware. Additionally, the rise of digital distribution platforms has revolutionized the way games are published and distributed. Unlike the past, when physical copies were necessary for distribution, most video games today can be distributed digitally, reducing the dependency on major publishing and distribution companies. This enables smaller teams to reach a global audience without the traditional barriers associated with physical production and distribution.

RSs are purposefully designed to assist users in finding relevant items within platforms that offer a vast array of choices. By providing personalized recommendations, RSs alleviate the need for users to extensively browse through long lists of items, ultimately enhancing their overall satisfaction. A satisfied customer is more likely to revisit the platform, fostering customer loyalty, and proving beneficial for the business.

To enhance engagement with the application, a RS should adhere to certain parameters. The foremost aspect is the relevance of recommended items to the user, as this ensures the RS remains useful and effective. Additionally, the system should take into account the novelty of items, avoiding repetitive recommendations to prevent user boredom. Emphasizing serendipity is also vital, as even less relevant items can pique the user's interest, opening up opportunities for exploring new interests. Balancing these factors creates a more engaging and personalized user experience.

To assess an item's suitability for a user, RSs depend on item rating systems. These ratings can be implicit or explicit. Implicit ratings are derived from user actions, such as purchasing an item in an online shop, which indicates the user's interest in the item. On the other hand, explicit ratings involve users providing direct feedback, such as rating an item with stars or giving it a thumbs up or down. Both implicit and explicit ratings are crucial in enabling RSs to analyze users' preferences and adjust recommendations accordingly.

RSs are not without challenges, particularly when dealing with new users or items that lack sufficient ratings, commonly referred to as Cold Start (CS) problems. These challenges arise due to the absence of data to effectively personalize recommendations. While CSs are a common issue, they are not the only problem faced by RSs. To address or circumvent issues such as CSs, various types of RSs have been developed, each with its unique approach. In the upcoming section 2.1.3, we will explore some of these techniques, such as Content-Based Recommender Systems (CBRSs) and Collaborative Filtering Recommender Systems (CFRSs), which offer valuable solutions to the challenges encountered in recommender systems, as well as provide simple practical examples for each approach.

In addition to addressing technical challenges, it is crucial to consider ethical aspects when developing RSs. One notable concern is the risk of showing inappropriate or harmful content to users, which can have negative consequences. Maintaining a high standard of content curation is essential to ensure a safe and enjoyable user experience. Moreover, the transparency of the recommendation process plays a significant role in user retention. Users tend to trust systems that provide clear explanations for their recommendations. A lack of transparency may lead to a loss of confidence in the platform, impacting user engagement and satisfaction.

We will also explore of Game Design, with a primary focus on video games.

Games leverage the human capacity for play and imagination. They involve active participation in events where choices made by players influence the unfolding of the game's narrative. The act of pretending allows players to engage willingly in an imaginary reality that they can leave behind when the game concludes. Integral to games are clear objectives that players can pursue, whether it be achieving a high score or outperforming other players, leading to victories or defeats. These **goals** provide a sense of purpose and motivation for players. Crucial to the structure of games are **rules**, which dictate the **actions** players can take and govern the consequences of those actions. Rules define the boundaries of what is permissible and guide the flow of events within the game. In the context of video games, the rules are enforced and managed by computer systems. Computers allow for more complex and dynamic gameplay experiences and interactions beyond what is feasible in traditional tabletop or physical games. Video games thus offer a unique and captivating form of entertainment, blending human creativity with the power of technology.

Game Design involves the imaginative process of defining and shaping a game's concept, mechanics, and overall experience. A popular approach is Player-Centric Game Design, where designers put themselves in the players' shoes to better understand their needs and preferences, ultimately aiming to create engaging and enjoyable experiences. A good design delivers a compelling and captivating game that resonates with its target audience, fostering a sense of entertainment and satisfaction. To bring these ideas to life, game designers often collaborate with a team of developers to implement and refine the game's design.

When designing a game, rules serve as the foundation, guiding the gameplay and defining the boundaries of interaction. However, to implement these rules in a video game, they must be translated into core mechanics. Core mechanics are more specific and tangible elements that act as intermediaries between textual rules and algorithmic implementations. They provide the functional interactions and behaviors that players engage with while playing the game.

In the context of playing a game, effective communication of the player's actions and the corresponding responses is crucial. **User Interface (UI)** plays a significant role in facilitating this communication and should be a fundamental consideration in Game Design. A well-designed UI enhances the overall "feel" of the game, contributing to a richer **User Experience (UX)**, while a poorly-designed UI can lead to a less satisfying experience for the player. Therefore, careful attention to UI design is essential to ensure a seamless and enjoyable gameplay experience.

While not all games are designed solely for entertainment, a significant portion aims to provide fun and enjoyment to players. Fun can be an elusive concept, but studying what makes a game not fun can help in designing better games. Some factors that detract from the fun of a game include poor gameplay mechanics, outdated or irrelevant features, and unappealing aesthetics. By addressing these issues and focusing on engaging gameplay, relevant features, and appealing visuals, game designers can enhance the overall fun and enjoyment that players experience.

With the intersection of Recommender Systems and Game Design in mind, we conducted a thorough review of recent literature on these subjects. As a result, we developed an application that offers users the opportunity to play various casual video games while receiving personalized game recommendations.

Casual Games are known for their simplicity in rules and UI, allowing for relatively short play sessions. In the application, we aimed to cater to diverse user profiles by creating games falling into different categories: strategy, skill, and luck. We implemented four games based on existing ones: Floppy Bird, a skill-based game where the player navigates through walls until collision; Wordle, a word-finding game with a secret word; Slots, a luck-based slot machine game; and 2048, a strategy-based grid game where cells are merged to reach 2048.

The implementation of these casual games is being carried out using the Godot game engine. Godot is a widely used open-source game engine that facilitates the creation of games and interactive applications. It offers a visual editor, scripting support, and a diverse array of features for both 2D and 3D game development. The engine is compatible with multiple platforms such as Windows, macOS, Linux, Android, iOS, and HTML5, making it a versatile choice for various game development projects. Godot has gained significant popularity due to its user-friendly interface, robust capabilities, and the fact that it is freely available as an open-source software.[LMc21]

The application is designed with a flexible architecture that enables easy addition or removal of games with minimal effort. This modularity ensures that new games can be seamlessly integrated into the platform, providing a diverse gaming experience for users. Additionally, the application incorporates a mechanism for saving game data, such as highscores and time played, to disk. This feature ensures that the user's progress and achievements are retained across different gaming sessions.

The Recommender System in the application is designed to utilize user gameplay data. As the user plays the games, information about their highscores and playtimes is recorded and stored. The RS then generates two ordered lists of game recommendations for the player: one based on the most played games by the player and another based on the player's highscores. This personalized recommendation approach enhances the user's gaming experience by suggesting games that align with their interests and performance history.

The application has great potential for expansion and improvement. Adding more games to the platform would not only enhance its appeal to users but also provide a more diverse dataset for the recommender, leading to more accurate and personalized recommendations. Further development of the RS could involve exploring advanced recommendation techniques like Collaborative Filtering (CF) or Hybrid Recommender System (HRS) to refine the recommendations.

The implementation of online play would introduce new dimensions to the application. By enabling data sharing and user grouping, the RS could leverage collective user behavior to make even more insightful and relevant game suggestions. Features like "Most played games in the world" could further engage users by showcasing popular games across the platform.

Overall, the application serves as a solid foundation for future research and development, and with continuous improvements, it has the potential to become an even more engaging and user-friendly platform for casual gaming and personalized game recommendations.

Due to constraints in time and resources, conducting extensive user testing for the application was challenging. While we recognize the importance of gathering feedback from a diverse group of testers to enhance the application's design and RS performance, it was not feasible within the given timeframe.

In the future, if the opportunity arises to conduct user testing, providing testers with a feedback form would be beneficial. This feedback could offer valuable insights into various aspects of the application, including game design and the effectiveness of the recommender. By analyzing this feedback alongside the data collected from user interactions within the application, we could make informed improvements to enhance the application.

2 State of the Art

2.1 Recommender Systems

Recommender Systems play a crucial role in assisting users by sorting and providing relevant content from a vast array of options. A practical definition, as articulated by Falk, K., describes a RS as one that calculates and delivers content based on knowledge about the user, the content, and the interactions between the user and the items [Fal19]. These systems have gained prominence in recent years, contributing to improved user experiences in various applications.

To successfully implement both a game and a RS to fulfill the objectives of this project, it is essential to review recent literature on the subject.

2.1.1 What are Recommender Systems

RSs play a crucial role in addressing the challenge of selecting a single item from a vast collection of options. On the Web, users are exposed to an overwhelming amount of content, including videos, articles, and games. The sheer volume of available items makes it impractical for a user to browse through them all and consume them.

Fortunately, the Web also offers an opportunity to collect valuable user data, which can be leveraged as knowledge to filter out less relevant items and emphasize the most relevant ones. In some cases, providing additional contextual information alongside recommendations can enhance the user experience. For instance, when recommending a hotel or restaurant, including its location can be beneficial.

According to [Agg16], a RS should adhere to four technical and operational goals:

- **Relevance**: The primary objective is to provide the user with recommendations that align with their interests and preferences. This focus on relevance is crucial; however, it is not the sole factor to consider;
- **Novelty**: RSs should also prioritize recommending items that the user has not encountered before. Continuously suggesting popular or repeated relevant items may lead to a decrease in item diversity and eventually result in the user losing interest;
- **Serendipity**: In addition to Novelty, Serendipity focuses on recommending items that are truly "surprising" to the user, even if they are not particularly relevant. This element of surprise adds a sense of discovery to the user's experience. In the short term, this goal may not always pay off, as the user may mainly see items they are not interested in. However, in the long term, it can lead to the user discovering new interests they didn't know about or didn't think they would like.

  For example, in a movie recommendation setting, consider a user who typically watches action films. The system would mostly make recommendations of already seen action films (relevance) or unseen action films (relevance and novelty). However, the recommender may also suggest comedy and horror movies, which the user never watches. If one day the user becomes bored of watching action films, they might decide to try watching a comedy or horror movie. If they enjoy it, they might be more open to exploring other movie genres, which keeps them engaged with the system.
- **Increasing recommendation diversity**: Recommender systems often provide lists of top-k items as recommendations. To increase diversity in these recommendations, the items in the list should have different item profiles. This means that even if the user doesn't find some items appealing, there is a higher chance that they might like at least one of the suggested items. This approach ensures that users don't get bored with repeated suggestions and are more likely to discover new items of interest.

Typically, RSs use Rating Matrices (RMs) as the basis for their recommendations [Agg16]. A RM, denoted by R, has a size of $m \times n$, where m is the number of users and n is the number of items on the platform. For a user u and an item j, we denote the rating as r_{uj}, as shown in Table 2.1.

As a starting point, only a small subset of these ratings are observed, since no user has interacted with every item. These observed values are then used as training data in a Machine Learning (ML) model, which will predict the missing matrix values (test data). ML will be properly addressed in a following section. Completing the matrix with the predicted values allows the system to make recommendations to the user. When an interaction between the user and a predicted value is observed, the actual ratings can be compared, and the model can fine-tune itself to make more accurate predictions.

2.1.2 Rating Systems

In RSs, the items that are recommended, whether they are hotels, video games, movies, etc., are referred to as "items". User-item interactions can be observed in several ways, with the most common methods

	Item 1	**Item 2**	...	**Item n**
User 1	4	5	...	?
User 2	?	3	...	4
...	...	...	...	...
User m	2	?	...	3

Table 2.1: Example of a Rating Matrix where the '?' are the unobserved ratings

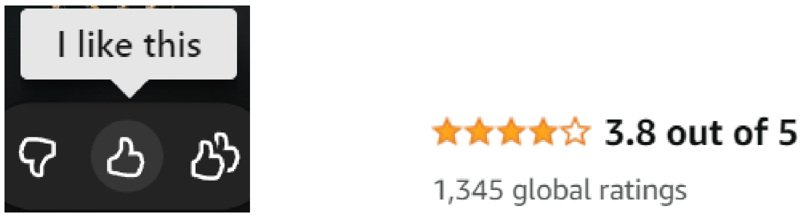

Figure 2.1: **Left:** Netflix's ternary method for rating a show; **Right:** Amazon's interval-based rating of an item

being Binary Rating (BR), Interval-Based Rating (IBR), and Ordinal Rating (OR). [Agg16]

In BR, the user indicates whether they like or dislike an item, often represented by a thumbs up or thumbs down. There are two common variations of this method: unary rating, where the user simply indicates that they like the item, and ternary rating, which is currently employed by Netflix, where the user can indicate that they love an item with a double thumbs up in addition to thumbs up and down options.

Another common interaction method is IBR, as seen in the five-star rating system on Amazon, where users rate an item by giving it one to five stars. Uncommonly, other methods are used, such as the Continuous-Interval Rating (CIR), where the user can choose fractional values, often obtained by sliding a bar. However, this method is rarely seen, as it puts a lot of pressure on the user to think of a precise numerical value for their rating.

The last method is the OR, where users can choose from options like "Strongly Disagree" to "Strongly Agree" This method is suitable when a broader range of responses is needed from the user and numerical values are to be avoided. However, in practical terms, this rating will be converted into a numerical variable for the recommender to utilize, making it similar to a five-star rating system.

The previously mentioned methods all fall into the category of explicit rating since they are directly specified by the user. Implicit rating, on the other hand, can take various forms. For instance, when a user browses items within a particular category on an e-shop, it may indicate interest in that category. If a user watches a video on platforms like Netflix or Youtube, the duration of their viewing and their subsequent choices can also be used to rate and recommend items.

2.1.3 Most Common Issues

As previously mentioned, RSs use machine learning techniques to address prediction problems, such as matrix completion, where missing values in a user-item interaction matrix are predicted. These techniques can also be used to classify users into specific interest groups based on their past interactions or observations. A group represents a profile with certain interest parameters, and users can be assigned to these groups. When a recommendation is made to a user, it can also be extended to other users within the same group. ML algorithms can be applied to solve classification problems in these scenarios. In the upcoming section, the main ML models and methods commonly employed in RSs will be explored. Before delving into those models, an overview of some of the common problems these methods seek to address in the context of recommenders will be provided.

The first problem that arises is related to the sparsity of the data matrix. When only a few values are observed in the matrix, it becomes challenging to predict the rest of the matrix using the observed data as training data. Sparsity occurs when the number of observed values for a user is significantly smaller than the total number of items, making it difficult to accurately predict other values due to the wide variation in subjects or domains of the items.

CS is another important problem in RSs, particularly concerning new users and new items. When users join the system, there is limited or no data available about their preferences, leading to empty or inaccurate values in the RM for that user. This makes it challenging for the system to accurately predict similarity values between new and existing users [TMAV21]. To address this, users are often directed to a questionnaire designed to gather data on their interests, such as movie genres they like, or asked to link a social media account to access additional information like age or user connections, providing a better starting point for the system.

Similar to cold users, there are also "grey sheep" users who are technically similar to cold users but not new to the platform. Grey sheep are users with highly specific or niche interests, making it difficult to fit them into a group profile. They pose the same challenges as cold users and may be handled similarly, such as using content abstraction into genres to start making recommendations.

On the item side, the cold item problem arises when new items are added to the system, and there is a lack of user interactions with them. This can lead to these new items being overlooked and not recommended, resulting in a cycle where having the item or not having it seems indistinguishable. To mitigate this problem, platforms can actively promote new items by sending email notifications to users or creating a dedicated section on the platform where new items are displayed, regardless of their rating, allowing users to discover them on their own.

Another approach to address the cold item problem is by "boosting" the item. This involves artificially giving a new item a high rating right from the start, so it can be recommended to users. While not publicly confirmed, Falk, K. speculates that Netflix may use this approach for some items. For example, when a highly popular series receives a new season, but user interactions with the series decline, Netflix can manually boost the show to display it in recommendations again, informing users about the new season and bringing renewed value to the existing item.

2.1.4 Machine Learning in Recommender Systems

Machine Learnings is a branch of Artificial Intelligence (AI) that allows software applications to be accurate in prediction and classification problems without being explicitly programmed to do so. The "learning" word consists on creating algorithms that automatically create representation models given a set of data. Training the model in one data set and giving it at least one performance metric makes the algorithm adapt its representation model of the data in an iterative way with the goal of getting the best performance. [Alp10]

ML is usually used for one of two tasks: classification and prediction. In classification, the algorithm's job is to "separate" the data. Given a data set that has different labeled classes, the algorithm must create a frontier between those. The model will be those frontiers that can be used to correctly classify new examples. Prediction is about the model's performance on unseen data. Once a model has been created, it's performance can be measured on new data.

A model's capability to properly classify new, unseen, data, falls under the concept of generalization, or abstraction. How well the model is able to abstract from the data and make correct prediction is the key to success. Poor generalization capability leads to two possible outcomes that make the model ineffective. Producing a model that corresponds too close to the training data set makes it incapable of generalizing. So, although the model's performance is great on the training data, it will fail to make predictions on new data, this is called **overfitting**. Inversely, if a model fails to capture the relationship between input and output accurately, sometimes because the training set was inadequate or more training time or features were needed (for example), it would fail to make predictions both in the training data and the unseen data, rendering it as useless as an overfitted one [Bis06].

Besides the prediction and classifications distinction. Depending on the feedback given to a learning algorithm we can also classify ML in four broad categories:

- **Supervised Learning**: In this approach the algorithm is given a data set containing inputs and their desired outputs. The goal is to find a model that maps the inputs to the outputs.[LW12] For example:
  - Linear Regression (LR): Computes a linear relationship between a dependent variable and one or more independent features. The goal is finding a linear equation to predict a dependent variable based on the independent ones.
  - NN: A method inspired by the human brain that uses interconnected nodes (neurons) in a layered structure. The first layer is the input layer where the data is processed and categorized and passed to the next layer. Then they have a hidden layer which can have one or more layer each one processing the data that comes from the previous one. The final layer is the output layer where the classification result is achieved. If the result is either a yes or no, the output layer can have just one node with values 0 or 1, or it can have multiple nodes representing any number of classes.

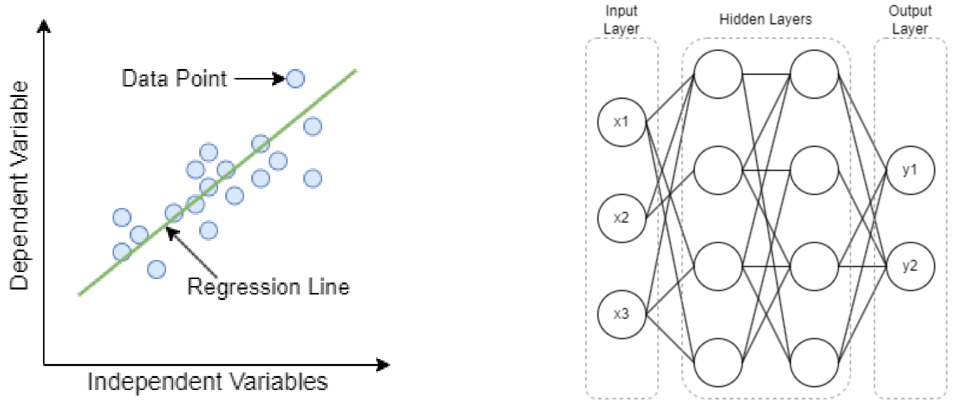

Figure 2.2: **Left:** Application of drawing a regression line that divides the data points; **Right:** Illustration of a NN with two hidden layers

- **Unsupervised Learning**: This approach is used on unlabeled data. The algorithm must then find hidden rules to group data without human intervention. Finding those hidden patterns can be a goal in itself and a step towards exploring the data by learning how to label it according to those rules (used in feature learning). [Dri21]

   - KNN: A simple type of clustering technique used in Unsupervised Learning is the KNN. To find what class to give to a data point, the algorithm calculates the Euclidean distance of k number of neighbours. From those k neighbours counts the number of data points in each class. The class with most data point in the set of k points will be assigned to the new data point.

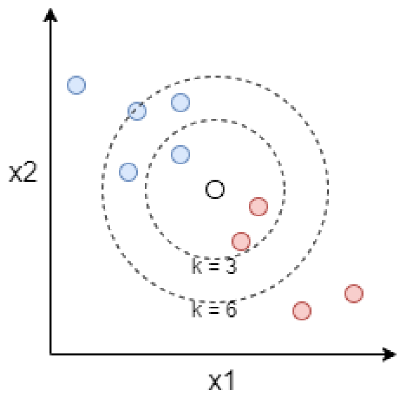

Figure 2.3: Example of a the KNN algorithm. There are two classes, represented in red and blue and a new data point, in white. If $k = 3$ the new data point will be classified as red, but if $k = 6$ it will be classified as blue.

- **Semi-supervised Learning**: This approach makes use of the previous ones that utilizes both labeled and unlabeled data. The problem with really big data sets is that it is time-consuming, and can be expensive, to label, in which case supervised learning might not be viable. On the other hand, using unsupervised learning on unlabeled data may not provide the accuracy desired. Semi-supervised learning is a hybrid technique between those two. From the big data set, a sample can be extracted and be labeled to train a model which will then be applied to a greater number of unlabeled data.[YSKX22]

- **Reinforcement Learning (RL)**: In this approach the developers devise methods of rewarding desired behaviours and punishing undesired behaviours. The developing of a model is accompanied by an agent that tries to maximize rewards and minimize penalties. Formulating a problem needs at least the following elements: environment where the agent operates; current state of the agent; feedback from the environment (reward); methods to map agent's states to actions (policies); the value of possible rewards that the agent can get by taking a particular action. Reinforcement Learning is mostly used in the fields of robotics and automation.[MBPJ23a]

   - Markov Decision Processes (MDPs): This is a type of model-based method. Here, the agent can navigate in a set finite environment states and where each state has a set of possible actions. Each transition between states has an associated reward (see 2.4). These methods are, however, not realistic, since rarely it is the case where the of environment mechanics is known in the real world.

   - Q-Learning (QL): A model-free algorithm that learns the value of an action in a particular state without having a model of the environment. From a known finite MDP, QL tries to optimize the policies to maximize the expected value from the current state to a final one, therefore maximizing the reward.

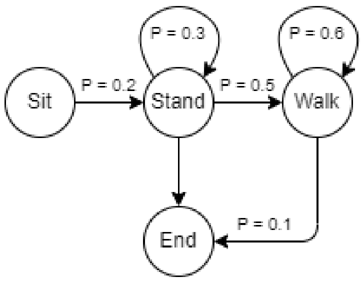

Figure 2.4: A Markov Chain representing an environment with states: sit, stand, walk and end. Each state has a transition with p value, representing the probability of changing from one state to the next. The agent will try to maximize the reward with p given, for example, a maximum number of steps.

One of the most important steps in creating an ML model is evaluating its performance. The quality of a model can be calculated through different metrics often known as evaluation metrics. Some of them are:

- Accuracy: One of the simplest metrics that can be used in a classification model is accuracy which gives the number of correct predictions in the total number of predictions. For the following equations, T_p, is the number of predicted positive cases that are actually positive, T_n, the number of predicted negative cases that are actually negative, F_p, the number of predicted positives that were actually negative and F_n the number of predicted negatives that were actually positive.

$$Acc = \frac{T_p + T_n}{T_p + T_n + F_p + F_n}$$

- Precision: A metric that is less limited that accuracy. Prediction determines the proportion of positive predictions that are actually correct, called True Positives. It is the ration of true positives to the positive prediction (true positives plus false positives).

$$P = \frac{T_p}{T_p + F_p}$$

- Mean Square Error: A method used in regression. In Linear Regression the model tries to draw a line that best separated data. Mean Absolute Error calculates the sum of all absolute differences between actual and predicted values over the number of data points. In the next formula Y represents the actual value, Y' the predicted value and N the number of data points.

$$mse = \frac{1}{N} \sum |Y - Y'|$$

There are many more evaluation metrics, some of which will be seen on the examples that follow.

In addition to performance there is also the matter of complexity. For certain algorithms and data sets, calculating a ML model can be very time consuming or not feasible at all. Considering the time to create and optimize models is also a job of the developers, which often have to settle on less accurate and less complex models instead of more accurate models that would take unrealistic amounts time to compute.[ZLJ21]

2.1.4.1 Content-Based Recommender Systems

CBRSs use the content of an item's description to predict it's relevancy based on a user's profile. The main goal is to recommend items to a user similar to previous items that the user rated highly. If a user likes an item, like a movie for example, the item's profile can be checked (such as the actors, the genre, the year it released...), usually built through keywords, to build a user profile with that information. A user's profile results from the history of interactions with the system and can be seen as a vector of features that describe the user. Now we can compare that profile to other items' profiles, check for similar features and make relevant recommendations based on them. This method is based on item representation, which means it is user independent [ZLJ21]. Therefore, this kind of system does not suffer from the data sparsity problem because when adding a new item, we can manually create a profile for it, which solves the new item cold start problem. Having a feature vector is also an advantage because we can use to explain to the user why that item is being recommended (recommend an action movie because the user liked another action movie), resulting in more transparency from the application, something we will talk more about in the ethical part of the review.

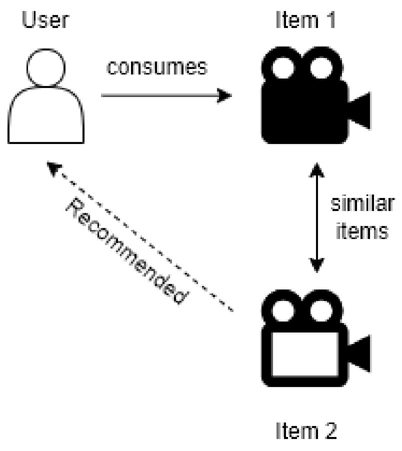

Figure 2.5: Item-Based Filtering method: the User consumes Item 1, which is similar to Item 2 that is then recommended to the User

A disadvantage of a system like this is that recommendations are always made based on similar items, which leads to overspecialized recommendations causing redundancy and lack of relevancy and novelty, making the users bored. These systems are more suitable to recommend articles and news items, since new ones come out frequently, than to recommend music or videos.

To better understand this concept, Gowtham S. R. has created an end-to-end practical example implementation of a simple CBRS using Python [Gow11]. The goal of this exercise is to create a CBRS for movies. Gowtham goes further into the example by providing a Graphical User Interface (GUI) for the application but for the purposes of brevity lets leave that part out. The data set used in the exercise is provided by Kaggle[1]. The data set is comprised of two tables:

- Movies.csv: data about a movie such as movie id, title, popularity, revenue, keywords, overview and others having a total of 20 columns;
- Credits.csv: data about the movies' cast and crew with four columns (including the movie id and title).

[1]Available at: `https://www.kaggle.com/datasets/tmdb/tmdb-movie-metadata?select=tmdb_5000_credits.csv`

Most of the code is about using pandas for manipulating the data, a Python library for working with spreadsheet like dataframes. That section isn't the focus of this work what needs to be known is that from the two initial tables we end up with a single table with three columns: *movie_id*, *title* and *tags* (2.2 shows how the table looks at this stage). Tags is a string composed of words from the overviews, cast, keywords, crew and genres of a particular movie.

	id	**title**	**tags**
0	19995	Avatar	in the 22nd century, a paraplegic marine...
1	285	Pirates of the Caribbean: At World's End	captain barbossa, long believed to be dead...
2	206647	Spectre	a cryptic message from bond's past sends...
3	49026	The Dark Knight Rises	following the death of district attorney...
4	49529	John Carter	john carter is a war-weary, former military...

Table 2.2: The first five rows of the dataframe, that result from the head() method.

The tags column has been created because one way to evaluate if an item is similar to another in CBRS is to see if they have similar profiles. This column will serve as the basis for that thought. The next step is to tokenize the tags. Scikit-learn provides provides several tools for ML. One such tool is the CountVectorizer which is a text feature extraction method used to convert a collection of text documents into a matrix of token counts. lets have a look at a simple example where we have a variable `text` that holds two strings:

```
text = ['Hello my name is John', 'This is my string']
```

CountVectorizer will lowercase all the words by default and will create a matrix $mtimesn$ where m is the number of strings, n the number of unique words and a value i_{mn} is the number of occurrences of a token in a string. For the above strings we get the following matrix:

	hello	**is**	**john**	**my**	**name**	**string**	**this**
0	1	1	1	1	1	0	0
1	0	1	0	1	0	1	1

Table 2.3: Matrix resulting from using CountVectorizer on `text`.

When CountVectorizer is applied to the table of movies the result will be a matrix with m movies and n unique tokens. Lets look at the following block of codeand explain it line by line.

```
from sklearn.feature_extraction.text import CountVectorizer
cv = CountVectorizer(max_features=5000, stop_words='english')

vectors = cv.fit_transform(new_df['tags']).toarray()
```

The first line is simply importing the CountVectorizer funtion from a scikit-learn library called Feature Extraction. In the next line a CountVectorizer object is created with two parameters:

- max_features: specifies the maximum number of features (words or tokens) to keep in the vocabulary. In this case, it is set to 5000, meaning only the 5000 most frequent words will be kept as features.
- stop_words: specifies that common English stop words (e.g., 'a', 'an', 'the', 'is', 'and', etc.) should be ignored during the tokenization process. Stop words are often removed as they do not carry much meaning and may not be helpful for ML tasks.

The last line This line fits the CountVectorizer to the 'tags' column of the DataFrame and transforms the text data into a numerical array representation using the token counts. The fit_transform() method first learns the vocabulary (unique words) from the 'tags' column and then converts the text data into a sparse matrix representation where each row corresponds to a movie, and each column represents a word from the vocabulary. The toarray() method is used to convert the sparse matrix to a dense NumPy array $movies \times features$ with size 4806×5000 for easy handling and exploration.

Each row of the array can be expressed as a vector of $length = 5000$. Since the goal of this exercise is recommending similar items given an item one metric that can be used to calculate similarity is the cosine similarity. Cosine similarity is the cosine between dot product of two vectors divided by product of their lengths.

$$similarity(\vec{a}, \vec{b}) = \cos\theta = \frac{\vec{a} \cdot \vec{b}}{||a|| \times ||b||}$$

What is needed now is to calculate the cosine similarities of each movie with one another. The following block of code does just that. First it imports the cosine similarity function from scikit-learn's metrics.pairwise module. The cosine_similarity function takes the input matrix vectors and calculates the cosine similarity between all pairs of rows in vectors. The output will be a new array, where each element (i, j) represents the cosine similarity between the $i - th$ and $j - th$ rows (movies) in the vectors matrix.

```
from sklearn.metrics.pairwise import cosine_similarity
similarity = cosine_similarity(vectors)
```

All that is left now is to create a function that given a movie recommends similar movies. Bellow is a function provided by Gowtham that recommends five movies.

```
def recommend(movie):
    index = np.where(new_df['title'] == movie)[0][0]
    similar_movies = sorted(enumerate(similarity[index]),
                     key = lambda x:x[1], reverse = True)[1:6]

    for i in similar_movies:
        print (new_df['title'][i[0]])
```

In the second line the function finds the index of the given movie (the row in the data frame). The second line finds the similar movies. The enumerate() function iterates over the values in the row of the similarity matrix corresponding to the given index. The sorted() function sorts the enumerated values based on the cosine similarity values (x[1]) in descending order (highest similarity first). The key argument specifies the function used to extract a comparison key from each element during sorting. After sorting, [1:6] slices the list of enumerated values, excluding the first element (since it represents the movie itself, which will have a similarity of 1 with itself) and taking the next five elements. The last part is a for loop that iterates the movies and prints out the recommendations.

Finally we can call the recommend function for the movie Avatar with:

```
recommend('Avatar')
```

Resulting in:

```
Titan A.E.
Small Soldiers
```

```
Independence Day
Ender's Game
Aliens vs Predator: Requiem
```

2.1.4.2 Collaborative Filtering Recommender Systems

CFRSs are some of the most popular and used techniques to this day. In contrast to Content-Based, which are user independent, CF-based techniques' main premise is that users who share similar interests consume similar items. Mainly, the most common problem is the matrix completion, where we predict a user's rating on unconsumed items. CF-based techniques can be categorized as memory-based or model-based. Memory-based CF techniques can be built with two main approaches: User-Based Collaborative Filtering (UBCF) and item-based Item-Based Collaborative Filtering (IBCF).[MBPJ23b] The UBCF approach is where users who have similar interests are recommended items rated by similar users. The IBCF approach is similar to UBCF but the item's profiles are taken into account, so items that share similar profiles are more likely to be recommended to a user who rated one of them. The core algorithm used in these techniques is the KNN [ZLJ21], a technique that given an item tries to classify it by measuring the distance to its neighbour items. k represents the number of neighbours that will be used to classify the item. To classify an item, with $k = 7$ (for example), the algorithm will calculate the distance from the item to other classified items and the class that is most common in those seven neighbours will be attributed to the item [PVG+11]. In RS, the system calculates the ratings of a target user on unconsumed items based on their neighbours' ratings.

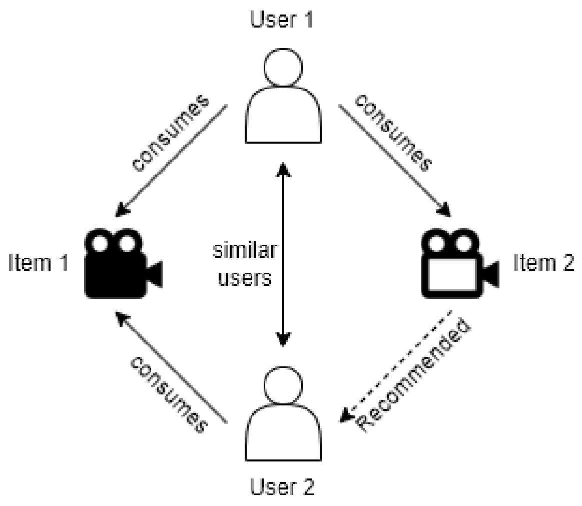

Figure 2.6: Collaborative Filtering method: here we assume User 1 and 2 are similar because they consumed Item 1. Since User 1 also consumed Item 2, this item will be recommended to User 2

A neighbour is another user that has rated the same items. If user A rated item a and b and user B rated item a, b and c, they are neighbours because of their shared rating of and a and b. From then on a recommendation of the item c to the target user can be created. The same can be done similarly on item-based techniques. CF has several inherent issues. Since CF-based recommender only provides recommendations based on previously rated items, if the data is too sparse[2], if it very difficult to recommend items that have never been interacted with. Although we mentioned ways to circumvent this problem, cold starts are also an issue with this systems, because new users have had no interactions and no profile has

[2]A matrix with mostly empty cells is called sparse, and a matrix mostly filled is called dense.

been assigned to them. Another big problem is their scalability, as the number of users and items grow on the platform and the amount of data increases, CF-based algorithms will see drops in performance, mainly due to the growth of the matrix and the complexity of the similarity computations [NVNG20]. Model-based techniques have been a successful way of trying to solve memory-based problems by creating a hybrid between CF and CBRS. This models use the profiling techniques of CBRS to predict ratings and filling the matrix.

Lets take a look at a practical example of CBRS. The data set is from Group Lens and is called MovieLens [HK15]. this particular data set is small having 100 000 ratings, 3600 tag applications applied to 9000 movies by 600 users. The only file needed for this exercise is ratings.csv. This is a table containing four columns: user id, movie id, rating and time stamp. Each row of the table represents an observed rating a particular user gave to a movie.

	userId	**movieId**	**rating**	**timestamp**
0	1	1	4	964982703
1	1	3	4	964981247
2	1	6	4	964982224
3	1	47	5	964983815

Table 2.4: Sample data from the MovieLens data set.

From this table its possible to build a sparse matrix similar to 2.1 with the observed values. The Python library Surprise was designed for building and evaluating RSs. It offers a collection CF algorithms, matrix factorization techniques and evaluation metrics for user-item interaction based RSs. Lets begin by getting the data from the file rating.csv:

```
from surprise import Dataset, Reader

data_path = '/content/ratings.csv'
reader = Reader(line_format='user item rating timestamp',
                sep=',', rating_scale=(0.5, 5), skip_lines=1)
data = Dataset.load_from_file(data_path, reader=reader)
```

The Reader class in surprise is responsible for parsing the input data file and specifying how the data should be read and processed. It allows to customize the format of the input file and set specific parameters related to the data, such as the rating scale which goes from 0.5 to 5, the separator used to separate values in the file. Skipping the first line means that the first line, the one with the labels (user id, movie id, rating and timestamp), won't be read. The load_from_file function is used to load user-item interaction data from a file and convert it into a Dataset object, which is a data structure used in surprise for handling recommendation datasets.

```
trainset = data.build_full_trainset()
```

The code line above creates a Trainset object, which is a specialized data structure in surprise designed to hold training data. It aggregates all the user-item interactions present in the dataset, regardless of whether they were used for training or testing. This object holds information about users, items, and ratings, as well as additional data structures that are optimized for efficient access during model training, such as assigning unique identifiers for users and movies for quick accessing.

Before going through the rest of the code lets talk about the method used, Singular Value Decomposition (SVD). SVD is a matrix factorization technique used in various fields, including RS, image compression, and data analysis. In the context of CBRSs, SVD is powerful because it reduces the dimensionality of the original user-item rating matrix and uncovers the latent structure in the data, enabling it to make predictions based on the discovered patterns. SVD may suffer from scalability issues when handling large datasets as it involves calculations over large matrices.[Bag20]

$$SVD = U\Sigma V^T$$

Without going too in-depth in the mathematics lets take a look at the formula above and break it down. SVD decomposes the sparce user-item rating matrix into three smaller matrices: U, Σ, and V^T.

- U: This matrix contains the representation of users in a lower-dimensional space (latent feature space). Each row in U represents a user, and the columns represent latent features. These latent features capture user preferences across different aspects.
- Σ: A diagonal matrix that contains the singular values in decreasing order. Singular values represent the importance of each latent feature.
- V^T: This matrix contains the representation of items (movies in our case) in the same lower-dimensional space. Each row in V^T represents an item, and the columns represent latent features. These latent features capture characteristics or attributes of the items.

The original user-item RM can be approximated by multiplying the three structures matrices obtained from SVD: $U\Sigma V^T$. Once the approximation of the RM is calculated, it can be used to make predictions for user-item interactions that were not present in the original matrix (i.e., missing ratings). These predictions represent estimated ratings for items that a user has not rated, which allows to make personalized recommendations for each user.

The next step is to create a SVD model and fit it into the training set. The method transform() that was used in the CBRS example is not applicable here since the only concern is to learn the patterns hidden in the matrix.

```
from surprise import SVD

model = SVD()
model.fit(trainset)
```

The only missing piece for completing the exercise it to calculate the missing ratings for a particular user a return the most relevant movies.

```
user_id = 100
movie_ids = trainset.all_items()

predictions = [model.predict(user_id, movie_id)
               for movie_id in movie_ids]
sorted_predictions = sorted(predictions, key=lambda x: x.est,
                            reverse=True)
```

Lets create predictions for user 100. The unique movie ids are stored in the variable movie_ids through the method all_items(). The variable predictions is a list comprehension to make predictions for a specific user. The list comprehension generates a list by going to each unique movie and generating a prediction (estimated rating). That list is then sorted from high to low based on the estimated ratings.

Iterating the list with a loop will produce the recommendations:

```
for i, prediction in enumerate(sorted_predictions[:5]):
    movie_id = prediction.iid
    estimated_rating = prediction.est
    print(f"Recommendation {i+1}: Movie ID {movie_id},
           Estimated Rating: {estimated_rating}")
```

Output:

```
Recommendation 1: Movie ID 0, Estimated Rating: 3.501556983616962
Recommendation 2: Movie ID 1, Estimated Rating: 3.501515184513213
Recommendation 3: Movie ID 2, Estimated Rating: 3.501487611681681
Recommendation 4: Movie ID 3, Estimated Rating: 3.501447891522447
Recommendation 5: Movie ID 4, Estimated Rating: 3.501446526526526
```

2.1.4.3 Hybrid Recommender Systems

HRSs are a class of RS that combine multiple recommendation approaches to provide more accurate and diverse recommendations to users. By leveraging the strengths of different recommendation techniques, hybrid systems aim to overcome the limitations of individual methods and enhance the overall recommendation quality.[WF20] There are several types of hybrid recommender systems, including:

- **Content-Boosted Collaborative Filtering**: In this approach, collaborative filtering is used as the primary recommendation method, and content-based filtering is used to augment the collaborative filtering results. Content-based features, such as item attributes or user profiles, are incorporated into the collaborative filtering model to provide more personalized and accurate recommendations.
- **Collaborative-Boosted Content Filtering**: In this approach, content-based filtering is the primary method, and collaborative filtering is used to complement content-based recommendations. Collaborative filtering can help address the cold-start problem, where content-based systems struggle to recommend items for new users with limited historical data.
- **Weighted Hybrid**: Weighted hybrid systems assign different weights to the recommendations generated by different algorithms. These weights can be static or dynamically adjusted based on the performance of each method. The final recommendations are a linear combination of the individual recommendation scores.
- **Switching Hybrid**: Switching hybrid systems dynamically select one recommendation method over another based on certain conditions or user preferences. For example, collaborative filtering may be used for active users with sufficient interactions, while content-based filtering is employed for users with sparse interaction data.
- **Feature Combination**: Feature combination hybrid systems merge the user-item features from content-based and collaborative filtering methods to create a unified representation. This joint feature representation is then used as input to a single recommendation model, such as a neural network or an ensemble learning algorithm.

HRSs offer several benefits. As they leverage the strengths of different approaches to compensate their weaknesses the accuracy of the recommendations increases as well as address cold start and sparsity problems. Considering both user preferences and item profiles can lead to more personalized recommendations, enhancing user satisfaction.[Agg16]

Some challenges HRSs offer are related to complexity in designing and implementing a system that integrates multiple algorithms and techniques which also leads to increased computational complexity and resource requirements. Another challenge is finding optimal combinations of algorithms and their weights which requires careful parameter tuning.

As done with the previous approaches, this one should also be studies. As such, to keep things simple, the next practical example is a HRS that utilizes both techniques used in Content-Based (CB) and CF to create a Content-Boosted Collaborative Filtering Recommender System. The only extra file needed is from MovieLens and its called movies.csv. Its a three column table: movie_id, title and genres. Each row corresponds to a unique movie (see 2.5).

	movieId	**title**	**genres**
0	1	Toy Story (1995)	Adventure Animation Children Comedy Fantasy
1	2	Jumanji (1995)	Adventure Children Fantasy
2	3	Grumpier Old Men (1995)	Comedy Romance
3	4	Waiting to Exhale (1995)	Comedy Drama Romance
4	5	Father of the Bride Part II (1995)	Comedy

Table 2.5: Sample data from movies.csv file from MovieLens.

The goal of the exercise is to recommend to a user five movies. Lets combine CF and CB in a weighted manner to create a hybrid recommendation approach. CF will use SVD to identify users with similar preferences and CB will use cosine similarity on the genres of the movies to recommend movies based on their intrinsic characteristics. Most of the code is the same so there is really no need to show most of it again. The important parts are:

1. Creating a SVD model and fit it into the training set which comes from the ratings.csv file:

```
model_cf = SVD()
trainset = data.build_full_trainset()
model_cf.fit(trainset)
```

2. Tokenize the genres in the movies.csv file using CountVectorizer and calculating a matrix with their cosine similarity:

```
cv = CountVectorizer(max_features=5000, stop_words='english')
genre_vectors = cv.fit_transform(movies_df['genres']).toarray()
similarities = cosine_similarity(genre_vectors)
```

3. Predict the movies for a given user using SVD:

```
cf_predictions = [model_cf.predict(user_id, movie_id).est
                  for movie_id in movie_ids_internal]
```

4. Combining both approaches to score the movies for that user:

```
scores = [0.5 * cf_pred + 0.5 *
          max(similarities[movie_ids.index(movie_id)])
          for movie_id, cf_pred in zip(movie_ids, cf_predictions)]
```

In the last step both approaches were given equal weight:

$$Hybrid\ Scores = 0.5 \times CF\ Prediction + 0.5 \times Maximum\ CB\ Similarity$$

The zip method is pairing each movie id from movie ids to their corresponding CF predictions in cf_predictions for iteration in the for loop. For each movie_id, movie_ids.index(movie_id) finds the index of that movie in the list of original movie IDs (movie_ids). Similarities is the matrix of content-based similarities between movies based on their genres and max is used to find the maximum similarity of the current movie with other movies.

By combining these two components, the hybrid scores represent a balanced measure of how well an item matches both the user's preferences (CF prediction) and the content characteristics (CB similarity). The equal weights (0.5) given to both the CF prediction and the CB similarity in the hybrid score ensure that both methods contribute equally to the final recommendation.

Now all that is left to do is sorting the scores and printing the results, which are all similar to how it was done in 2.1.4.1 and 2.1.4.2. For the user with id 1 the output is:

```
Recommendation 1: Andrew Dice Clay: Dice Rules (1991)
Recommendation 2: Bungo Stray Dogs: Dead Apple (2018)
Recommendation 3: Flint (2017)
Recommendation 4: No Game No Life: Zero (2017)
Recommendation 5: Black Butler: Book of the Atlantic (2017)
```

2.1.4.4 Knowledge-Based Recommender Systems

Knowledge-Based Recommender Systems (KBRSs) are an approach that relies on explicit knowledge about users' preferences and item characteristics to make recommendations. Unlike CFRS and CBRS, Knowledge-Based Recommender Systems do not rely solely on historical user-item interactions or item features. Instead, they use domain-specific knowledge, rules, and constraints to generate personalized recommendations.[KPAM20]

KBRSs aim to provide highly personalized and tailored recommendations to individual users based on their specific needs and preferences. The explicit rules consider various user attributes, such as demographics, preferences, and constraints, to generate relevant recommendations. They also take into account various item attributes, properties, and constraints. These attributes could include item specifications, availability, price ranges, compatibility, and more. The core component of a these systems is the rule engine that process the user and item information based on the predefined rules and generates recommendations accordingly. Domain experts play a crucial role in formulating the rules and constraints to ensure the system's accuracy and relevance. Since the recommendations are based on explicit rules they have the advantage of being able to provide explanations for their recommendations. Users can understand why a particular item is being recommended to them based on the activated rules.

Limitations of KBRSs:

- **Domain Expertise Required**: Building a Knowledge-Based System often requires domain experts who can encode relevant knowledge into explicit rules. This expertise can be challenging to acquire.
- **Scalability**: As the number of rules and knowledge base grows, the system's complexity can increase, making it challenging to scale efficiently.

- **Rule Maintenance**: Regular rule maintenance is essential to keep the system up-to-date with changes in user preferences, item characteristics, or business requirements.
- **Limited Discovery**: Knowledge-Based Systems rely on existing explicit knowledge and may not discover new or unexpected preferences or patterns that Collaborative Filtering and Content-Based Filtering can capture.

A brief disclaimer: while working on this project it was hard to find a simple example that could be implemented and tested. Unlike the previous methods, for KBRSs we opted to instead give a generic example that hopefully helps understand this approach a little better.

In this example, a travel agency wants to recommend personalized vacation packages to its customers based on their specific preferences, interests, and constraints. So, the domain of the Knowledge Base (KB) here will be Travel Agency Vacation Packages and its built on:

1. **User Preferences**: The travel agency collects information about its customers, such as age, budget, preferred travel dates, preferred destinations, travel style (adventure, luxury, family-friendly, etc.), and any specific activities or attractions they are interested in (e.g., beach resorts, cultural tours, outdoor adventures).
2. **Item Characteristics**: The vacation packages offered by the travel agency have various attributes, including destination, duration, price, type of accommodation, included activities, and special offers.
3. **Domain Expertise**: The travel agency's domain experts that work closely with the data science team to encode their knowledge into explicit rules and constraints that will be used by the KBRS.

Within the domain, some simple rules that can be applied are:

1. If the customer has specified a preferred destination, recommend packages for that specific location;
2. If the customer's budget is high and they prefer luxury travel, recommend a 5-star all-inclusive resort package in a tropical destination;
3. If the customer is an adventure enthusiast and prefers outdoor activities, recommend an adventure tour package with hiking, kayaking, and camping;
4. If the customer is a family with kids and prefers family-friendly activities, recommend a package with a family-oriented theme park and nearby accommodations.

To generate recommendations, when a user visits the agency's website, they are prompted to provide information about their preferences and aversions. The system considers the user's preferences and constraints, along with the attributes of the available vacation packages, to find the best matches based on the rules.

2.1.5 Ethical Requirements

A concern that arises when approaching recommendation topics and is often paired with cyber security is that of the intrusiveness of a platform. There are two classes of variables that are morally relevant: actions and consequences. The value of consequences is often measured in terms of the utility they contain [MTF20]. So, it is reasonable to assume that any aspect relative to RSs that could impact negatively any of its stakeholders constitutes a relevant feature to be considered when designing such a system. In terms of recommenders, there are two ways in which they can have ethical impacts: damage of its users or violate their rights. In RS, there are risks involved when exposing users to potential privacy violations or potentially damaging content. This exposure in itself may constitute a wrong even if no damage is materialized. This is familiar from other contexts such as medical ethics, for example, negligence in treating a patient constitutes a wrong, even if the patient ultimately recovers.

Here are some of the major ethical concerns with RSs:

- **Inappropriate Content**: The internet contains a vast amount of content, and users should have control over what they see. Souali et al. proposed an ethical database that follows a region's accepted norms, acting as a filter for recommendations [SEAF11]. Tang and Winoto take it a step further and propose a "user-adjustable" ethical filter. However, a problem with this kind of filter is the data it gathers. A filter that excludes certain content may reveal that such content could cause harm, distress, or be considered irrelevant [WT10]. Paraschakis, when focusing on e-commerce, identifies five ethically problematic areas: user profiling practices, data publishing, algorithm design, user interface design, and online A/B testing (which involves exposing groups of users to different algorithm variations to gather feedback). These areas can potentially breach user privacy through data leaks or behavior manipulation, leading to a lack of user trust in the platform [Par18].
- **Privacy**: One of the primary challenges in RSs is privacy. Privacy violations can occur at different stages: first, when data is gathered without the user's consent; second, when the collected data is leaked and could be used to identify individuals; third, when inferences are drawn from stored data, regardless of its security or accuracy; and lastly, when inferences are based on user profiling, which can have varying consequences depending on the domain. Adding privacy controls or security filters to the system can help mitigate privacy risks. However, even these measures may inadvertently reveal information about users, leading to potential privacy concerns. For instance, strong privacy settings might be interpreted as a sign that a user has something to hide. Addressing privacy concerns in RS requires careful consideration and measures to protect users' data and ensure their trust in the system.
- **Autonomy and Personal Identity**: In the context of advertisement or politics, RSs have the potential to influence users' interests and choices. While a RS should aim to empower individual agency and aid decision-making by filtering out irrelevant items and promoting relevant ones, there is a risk of malign behavior when the system manipulates or coerces users, attempting to create addictive patterns towards certain content. Moreover, RS operates in a dynamic environment, actively contributing to the construction of a user's identity rather than passively tailoring recommendations based on pre-established identities. This active role in shaping a user's identity raises ethical concerns about the extent of autonomy and control users have over their own preferences and personal identity within the system.
- **Opacity**: Ensuring transparency in recommenders is crucial to respect the end-user's autonomy. By providing explanations for why certain items are recommended, the system can mitigate the risk of exerting undue influence on the user's preferences. Transparency also fosters a better understanding of the user's current profile and builds trust in the platform. Explanations can be implemented in various ways. For instance, in CF-based systems, the explanation may revolve around informing the user that an item is recommended because other users with similar profiles have shown interest in it. However, depending on the system's application, there may be drawbacks to providing explanations. Recommending items solely based on their popularity among other users could lead to a "winner-takes-all" situation, where a few highly popular items dominate the recommendations. This may limit the variety of options and choices available to the user, potentially reducing the system's overall

utility and satisfaction for the user. Balancing transparency with diversity and personalization is essential to create a successful and ethically sound recommender system.

- **Social Effects**: RSs can have a transformative impact on society, leading to both positive and negative consequences. One discussed effect is the potential for overspecialization, where a few highly popular items dominate the recommendations, resulting in a "winner-takes-all" situation. This can lead to most users receiving very similar recommendations, creating a sense of herd mentality where users' experiences on the platform become homogenized. As a consequence, less popular items may receive fewer interactions, leading to a decrease in their visibility and ratings. This homogenization of recommendations can have social implications, as it may contribute to the spread of herd behavior in society. A well-known example of this phenomenon is the rise of anti-vaccine propaganda, which has been linked to a decrease in herd immunity.[Bur20] To counteract the problem of overspecialization and promote diversity in recommendations, strategies like promoting serendipity can be employed. This can be achieved by boosting certain items or creating pre-configured user profiles that differ significantly from the majority, encouraging novelty and diversity in the recommendations. By actively promoting a broader range of items, recommenders can help users discover content outside of their typical preferences, fostering a more diverse and engaging user experience.

2.1.5.1 Ethical Solutions on Youtube

As a prominent video-sharing platform, Youtube, faces the challenge of providing relevant and appropriate content to its users amidst the vast amount of videos uploaded daily worldwide. To achieve this, Youtube employs a UBCF approach, where users with similar interests are recommended similar videos. However, over the years, Youtube has encountered an increase in low-quality content, including misinformation, racy, or violent videos. To address this issue, they have implemented various measures.

In 2011, Youtube introduced classifiers to identify and prevent the recommendation of videos containing racy or violent content. In 2016, Youtube took further action by developing predictive models to assess the likelihood of videos involving minors in risky situations. These videos were subsequently removed from the platform to protect the well-being of young users. Additionally, in 2017, Youtube acknowledged the importance of fairness for marginalized groups, such as the LGBTQ+ community. They began evaluating their recommendation algorithm to ensure equitable treatment and avoid biases against these groups. Through these efforts, Youtube aims to provide a safer and more inclusive environment for its users, combatting inappropriate content and fostering a positive user experience.[Goo21]

2.1.5.2 Ethical Solutions on Spotify

The popular music streaming platform, Spotify, prioritizes user privacy and data protection in their RS. By offering privacy controls to their users, Spotify allows users to decide how much data they want to share, giving them greater control over their personal information.

To further protect user privacy, Spotify employs anonymization techniques on user data before using it in their recommender. This process ensures that individual user identities are removed from the data, making it impossible to associate specific recommendations with particular users. Instead, recommendations are based on aggregated patterns, preserving the privacy of individual users.

Additionally, Spotify implements differential privacy on user data. This involves injecting random noise into the data before aggregation, making it extremely challenging to identify the contributions of any single user. By applying differential privacy, Spotify enhances data privacy while still leveraging the collective information to improve their recommendation system.[MAI22]

These privacy measures demonstrate Spotify's commitment to ethical data handling and their dedication to maintaining user trust and confidence in their platform

2.1.5.3 Ethical Solutions on Amazon

Amazon places a strong emphasis on diversity in its recommendation algorithms to enhance user experience and promote fairness. To achieve this, they work towards collecting diverse and representative data that encompasses users from various demographic backgrounds, geographical regions, and cultural contexts.[ama] By considering a wide range of user perspectives, Amazon aims to deliver recommendations that cater to diverse interests and preferences.

To minimize bias in recommendations, Amazon employs fairness-aware algorithms. These algorithms are designed to ensure that all products or content, regardless of their source, have equal opportunities to be recommended to users. This means that products or content from different sellers, regardless of their size or popularity, are given fair exposure and consideration in the recommendation process.[Sta22]

By incorporating diversity metrics and fairness-aware algorithms, Amazon strives to create an inclusive and equitable recommendation system that meets the needs of its diverse user base. These efforts reflect Amazon's commitment to ethical and responsible use of data in providing personalized and unbiased recommendations to its customers.

2.2 Game Design

In the realm of Game Design, numerous subjects can be explored. The chosen approach for this project involves the exploration of some key topics. This exploration is based on the study of two influential books: "Fundamentals of Game Design" by Ernest Adams, and "Game Mechanics: Advanced Game Design" by Ernest Adams and Joris Dormans. These books will serve as guiding resources, providing definitions and explanations, which will be supplemented with additional research from more current and prevalent academic documents.

2.2.1 What is a Game?

The concept of a game can be initially defined by exploring human nature, which exhibits an inherent desire to play and the ability to engage in pretense. Play is considered a nonessential and recreational human activity, while pretense involves willingly immersing oneself in an imaginary reality, always aware of the pretense. Games encompass both of these elements.

According to Ernest Adams, a game can be described as an activity conducted within a pretended reality, where participants strive to achieve at least one arbitrary, nontrivial goal by adhering to established rules. While this definition may not fully capture the various aspects of games, it suffices for the scope of this work.

Based on the provided definition, four essential elements can be identified as the constituents of a game:

- **Play**: A participatory form of entertainment where participants make choices that influence the unfolding events. It grants the freedom to act and decide within the boundaries defined by the rules;

- **Pretending**: The capacity to conjure an imaginary reality in the mind when the game starts and disengage from it when the player leaves the game or it concludes. In multiplayer games, participants collectively engage in pretense, and crucially, they concur to imagine the same things and adhere to the same rules. Adams notes that occasionally, the demarcation between reality and pretend can become ambiguous, as exemplified by online gambling, which blurs the line by involving real money wagers on game outcomes;
- **Goal**: Games necessitate one or more goals, which can vary widely from achieving a checkmate in chess to defeating a boss in a computer game. The goals need not be overly rigid; for instance, in the game Minecraft, the Creative Mode offers an endless play experience centered around the goal of creation. Arcade games, on the other hand, often featured time limits or endless play with the objective of attaining a high score, either on a personal or global level. The attainability of goals is not critical; what matters is that they motivate players to strive for accomplishment through gameplay. Game designers establish and define these goals using rules, and they should offer an appropriate level of challenge. Even in Minecraft's Creative Mode, creation itself presents challenges, whereas simply betting on a coin flip does not. Goals can also involve a sense of victory or loss, although they are not a fundamental aspect of games. The inclusion of these elements can heighten excitement as they extend beyond the realm of the imaginary reality. Winning can be perceived as a significant accomplishment, leading to a sense of pride;
- **Rules**: The rules within a game represent the instructions and definitions that all players agree to follow upon entering. These rules can manifest in various forms:
  - Semiotics: Games utilize symbols to convey meanings and establish relationships between different elements;
  - Gameplay: These rules define the challenges and actions available to the player;
  - Sequence of Play: These rules govern how the game and its events unfold;
  - Termination Goals: Conditions that signal the end of the game;
  - Metarules: These are rules about rules, governing situations when rules can change or exceptions apply.

  To ensure a smooth experience, rules should be unambiguous and coherent, preventing conflicts over their interpretation.

The definition of a game provided earlier does not explicitly specify entertainment as a primary aspect. While most games are indeed designed for entertainment, there are also games created for educational or health improvement purposes. Adams acknowledges that his definition does not directly mention fun, as it is not an inherent property of the game but rather a response generated from the act of playing.[Ada10]

2.2.2 Video Games

The definition provided for games is broad enough to encompass conventional games like basketball and soccer, tabletop games such as Risk and Monopoly, as well as video games, which is the focus of this project. Video games represent a subset of the game universe where the rules are mediated through a computer, whether it be a small device like the Tamagotchi, an arcade machine, or a personal computer.

An advantage of video games is that the rules can be hidden from the player. With the computer enforcing the rules and managing victories and defeats, players can simply focus on playing and discover what they are allowed or not allowed to do. This can present both advantages and disadvantages. On the one hand, players who don't know the rules may find it challenging to optimize their choices, which can be frustrating for those seeking immediate success. On the other hand, it can be exciting for players who enjoy exploring and discovering the rules for themselves. In any case, basic rules should be easily learned, and most games often provide players with hints or tutorials to help them get started.

In video games, presenting the game world to the player can involve multiple layers. Visuals are provided through a screen, audio through speakers, and tactile stimulation through trembling a controller, all of which contribute to immersing players in an imaginary reality. This immersion can be further enhanced by using technologies like Augmented Reality (AR) in mobile games, where cameras are involved, or Virtual Reality (VR) with devices like the Oculus Rift. VR headsets and controllers enable players to feel "present" in a virtual world and interact with it. These technologies create a more immersive and interactive gaming experience.

AI has a significant role in video games, especially concerning strategy and gameplay:

- **Strategy**: In perfect information games like chess or tic-tac-toe, AI can determine optimal courses of action, making it suitable for creating opponents for players, which usually involves some type of Constraint Programming. In modern games, AI can be used to create dynamic opponents that adapt to the player's behavior. Instead of relying on predetermined strategies, they can learn from the player's actions and adjust its approach accordingly. This creates a more challenging and engaging gaming experience as the game becomes more responsive and unpredictable;
- **Pathfinding**: Another common application of AI in video games is pathfinding, which involves finding the best routes to objectives through obstacles. Non-Player Characters (NPCs), such as enemies or pets that follow the player, use pathfinding to navigate the game world. Advanced pathfinding algorithms can take into account dynamic obstacles and changing environments. A* Pathfinding is a popular algorithm used in computer science and video games to find the shortest path between two points in a graph or grid-based environment;
- **Natural Language Processing**: Although not yet widespread, natural language parsing and generation have potential in video games. Currently, interactions with NPCs involve reading or hearing recorded scripts with limited dialogue options. Advances in natural language processing, like OpenAI's ChatGPT, may lead to more immersive and dynamic interactions with NPCs as they become more conversational and responsive to player input;
- **NPC Behaviour**: NPC behaviors, particularly those of simulated people and creatures, are frequently learned from real examples and implemented in games to achieve more realistic character movements and actions. ML techniques, such as RL, can be used to train NPCs to behave more realistically and human-like. NPCs can learn from player behavior, player feedback, and other data sources to improve their decision-making and actions in the game world. This adds depth and authenticity to the game world.

Video games can have various personal and social impacts on players. In the past, concerns were raised about video games contributing to sedentary behavior and potential obesity, but recent studies have shown inconclusive results regarding the direct correlation between playing video games and obesity in children.[KJS20] With the advancement of VR technology, some studios are now creating games that encourage physical activity and help combat obesity. These games provide players with opportunities for exercise and movement, promoting a healthier lifestyle [Edw22].

Beyond physical health, video games have been found to have positive effects on cognitive functions, including enhancing brain activity and attention control [BG19]. Additionally, video games can offer social benefits by providing opportunities for social interaction and cooperation, especially in multiplayer or online gaming environments [HOM19].

During the COVID-19 pandemic, when people faced prolonged periods of isolation and lockdown, video games became even more important for many individuals. A study conducted by Barr, M. and Copeland-Stewart, A. found that a majority of respondents experienced cognitive stimulation, social opportunities, and mental health benefits such as reduced stress and anxiety levels from playing video games during the lockdown [BCS22].

Overall, video games have evolved to offer diverse experiences that can positively impact personal well-being and provide valuable social connections, in addition to being a source of entertainment.

2.2.3 What is Game Design?

Adams outlines game design and development in five steps: first, imagining the game; second, defining how it will function; third, describing the various elements of the game, including conceptual, functional, and artistic aspects; fourth, transmitting this information to a team that will build the game; and finally, refining the game through testing and iteration.

A design philosophy known as Player-Centric Game Design can greatly contribute to creating an exceptional gaming experience. In this approach, the game designer envisions a representative player for the game they wish to create. By adopting this philosophy, the designer takes on two important obligations: the duty to entertain and the duty to empathize:

1. The duty to entertain means that the primary goal of the game is to provide entertainment and enjoyment for the player. All other motivations or objectives are considered secondary to this main purpose.
2. The duty to empathize requires the designer to put themselves in the player's position and understand their preferences and desires for entertainment. By doing so, the designer can tailor the game to meet the player's expectations and deliver a more satisfying experience.

By following Player-Centric Game Design, game designers can create games that are more engaging, enjoyable, and tailored to the preferences of the players, leading to a more positive overall gaming experience.

There are other perspectives designers can take to develop games however. Dishon, G. and Kafai, Y. on their study of Perspective-Taking through game design offer other perspectives such as looking through the lens of a social group as a whole instead of just one person so that the design process has the interactions of the players in that group in mind [DK20].

The designer must adopt a multifaceted approach when developing a game. Considering a game solely as a form of art is too limited, as it is not just an expression of artistic creativity nor merely an engineering feat as some may perceive. Instead, it can be seen as a craft that combines both aesthetic and functional elements, resulting in an elegant and high-quality product.

2.2.4 User Experience and Interface

The UI is a familiar concept for those interested in computer software. It serves as the medium through which players interact with the game, using their input and output devices, and sometimes with other players. The UI's primary function is to facilitate the interaction between the player and the core mechanics of the game. When players engage with the game, they interact with the UI, which then communicates with the core mechanics to generate outputs, such as graphics on the screen or sound through the speakers, to inform and entertain the player.

A well-designed UI enhances the UX of the game, ensuring that players enjoy their interactions and overall gameplay. In contrast, poor UI design can lead to a subpar UX. Today, the process of designing these elements is often referred to as UX design, emphasizing the goal of entertaining the player. On the other hand, UI has traditionally been used to describe the creation of icons, screens, and menus. Therefore, it is crucial to distinguish between interface and experience. Experience occurs in the player's mind, influenced by the entertainment provided, while the interface is the software component that mediates between player actions and game responses.

UX holds a significant place in game design, and prominent companies like ArenaNet and BioWare, known for developing popular games like Guild Wars and Mass Effect, recognize its importance by employing UX Designers or dedicated UX Design Teams. Emphasizing Player-Centric development, UX allows designers to empathize with the end users, enhancing their overall experience and evoking emotions or reactions through various gameplay elements. This is where the concept of "game feel" comes into play, similar to how we can discern a movie or a dish made with "passion" and "soul". In the context of video games, UX plays a crucial role in creating a captivating and engaging gaming experience.

In practicality, we can observe how UX is employed in current video games. Take, for instance, the recent Spider-Man Miles Morales game. In this game, when the player is about to be attacked, a hint appears above Spider-Man's head. While the player could potentially see the enemy's animation, this hint serves a purpose in enhancing the gaming experience. Spider-Man's superpower, the spider sense, allows him to sense incoming danger. By providing this hint, the player can better predict and respond to attacks, adding to the immersion of being the character of Spider-Man.

Additionally, when the player successfully dodges an incoming attack at the right moment, the game employs a slow-motion effect for a brief moment. This conveys a sense of speed to the player, creating a more intense and thrilling experience [Har20]. These examples demonstrate how UX can be strategically used in video games to enrich the player's immersion and overall enjoyment.

The Heads-up Display (HUD) in a video game is a crucial component of the UI that presents important information and data directly on the screen while the game is being played. It provides players with real-time feedback and relevant information about their in-game status, resources, objectives, and other essential elements. The HUD is designed to be visually accessible and unobtrusive so that players can quickly and easily interpret the information without interrupting their gameplay. The HUD typically includes various elements such as: health, ammo, inventory, mini-map, quest/objective tracker, scores, crosshairs and skills.

The design of the HUD is critical for providing players with essential information without overwhelming or distracting them. A well-designed HUD should complement the game's visual style and theme while maintaining clarity and readability. Game developers often conduct usability testing to ensure that the HUD elements are effective and intuitive for players.

Numerous video games incorporate UIs as an essential component, and virtually all of them feature some form of user interface. For instance, Interface in Game (`https://interfaceingame.com`) is a website dedicated to gathering diverse UI elements from video games, serving as a source of inspiration for video game designers.

2.2.5 Making Games Fun

Most video games are primarily focused on providing entertainment, which is a broader concept than simply fun. While fun is a common outcome of gameplay, entertainment encompasses a range of human emotions, including learning, pride, fear, and more. To achieve emotional engagement, designers can employ Emotional Design, a practice that identifies elements within games capable of eliciting specific emotions in players.

Since the ultimate goal is to entertain players and ensure they have fun, it becomes essential to identify aspects that contribute to an enjoyable gaming experience. By doing so, designers can keep players engaged, increase the game's popularity, and ultimately boost its revenue. Adams emphasizes that the key to making a game fun lies not only in creative and imaginative ideas but also in the effective execution of those ideas. In essence, it is the proper implementation of concepts that prevents games from becoming boring, irritating, or awkward, rather than the inherent quality of the idea itself.

Following Adams' perspective, there are four key aspects that contribute to the fun factor of a game, ranked from most to least influential:

1. **Avoiding elementary errors**: The most crucial factor is to eliminate any issues related to bad programming, bugs, poor music, bad interfaces, and subpar art. Addressing these fundamental problems ensures that the game's foundation is solid and functional;
2. **Attention to detail**: The second most influential aspect involves focusing on small yet significant elements that enhance the player experience. This includes incorporating good animations, captivating UI design, appealing visual effects, and smooth transitions between levels or menus. These attention-to-detail aspects keep players engaged and elevate the game's overall polish;
3. **Creative variations on mechanics**: Offering creative twists and unique variations on the game's mechanics and design can set it apart from other similar games. These inventive elements help maintain the game's freshness and distinguish it from competitors that may share similar ideas;
4. **Design innovation from unpredicted interactions**: Lastly, design innovation can emerge from unexpected or unplanned interactions within the game. When making creative decisions or updating a game, it is crucial to identify and analyze these cases to preserve and potentially build upon any serendipitous discoveries.

Although formalizing fun is not the goal here, following these rules allows developers to take steps to make a game enjoyable and fun for players.

- Gameplay takes precedence above all else, as it is the mechanics and challenges that provide the most fun, while art direction, graphics, and storylines are considered secondary factors;
- Adopting a Player-Centric point of view is crucial. Designing the game as if the player is making decisions from that perspective ensures that fun is prioritized, and the player's experience is not lost sight of;
- If a feature isn't right or well designed or is optional, it is left out. Meaningless activities can detract players from the main fun-providing ones. If a feature is truly necessary and resources are available, it can be implemented, but it should be considered secondary, possibly even semi or fully automated;
- By following the designer's vision while being open to feedback and exploring other design options, a balance can be achieved, leading to a fun game with a clear goal in mind;

- While aesthetics may not directly generate fun, their absence can diminish the overall enjoyment of the game.

In 2019, Taborda et al. conducted a systematic review of the literature on fun; however, they did not find many metrics for gauging fun. This conclusion suggests that the topic requires further study and scrutiny.[TALC$^+$19]

2.2.6 Core Mechanics

When envisioning a game, designers create a set of rules, such as "When an enemy attacks the player, the player gets hurt and can die." These rules serve as the foundation for the game. A core mechanic is a more specific model that the designer creates to algorithmically implement the rule. For instance, using the previous example, a core mechanic could be formulated as follows: "When an enemy attacks the player, it deals 10 points of damage, and the player loses 10 points of life. If the player's life reaches 0, they die." These mechanics define what players can and cannot do, the challenges they may face, and how their actions affect the game world, ultimately constituting the core gameplay experience.

Various types of game mechanics warrant exploration, and the literature generally acknowledges at least five key ones that are worth mentioning[Ada10]:

- **Physics**: This mechanic provides the game with motion and force following (usually) Newtonian mechanics. Designers can tweak these mechanics, such as adjusting gravity, to accommodate their objectives;

- **Internal Economies**: These are rules that govern the creation, consumption, and exchange of resources within the game world. They should not be confused with player-driven economies seen in games like POE or World of Warcraft, where players exchange items through a marketplace. Instead, internal economies define elements like how many gold coins a monster will drop when killed, the resources dropped besides gold, how players will consume those resources, and the selling price of items at NPC vendors;

- **Progression Mechanics**: This mechanic focuses on how a player will progress through the challenges presented in the game. For example, progression mechanics can involve completing levels by defeating enemies or solving puzzles to unlock doors;

- **Tactics**: Tactical maneuvering is related to the positioning and planning of player movement. Some games are purely tactical, like chess or Go, while in video games like Total War and Age of Empires, players control units to conquer other civilizations. Other games may not be entirely focused on tactics but still implement some tactical mechanics, as seen in games like League of Legends, where managing when troops arrive can provide a significant advantage;

- **Social Interactions**: These rules govern how players can interact within the game. Social interactions encompass not only inherent social interactions players can have in games but also define how players can interact with each other. For example, in the game World of Warcraft, players can create parties (groups of five people to tackle minor challenges), raids (groups of up to 40 people to face greater challenges), or guilds (with up to 1000 players, focusing more on social interactions and role-playing). Other rules can include the ability for players to send messages or mail to other players [AD12].

2.2.7 Game Balancing

Game Balancing is a challenging concept to formalize. It typically involves a series of design and tuning processes aimed at creating a game that is fair to the player(s), striking the right balance between being neither too easy nor too hard. Essentially, a well-balanced game should fully utilize the player's skill and capabilities.

A Balanced Game should be a carefully crafted experience that considers various aspects to ensure fairness, engagement, and skill-based challenges for players. By providing meaningful choices, designers strive to maintain player interest and immersion. When faced with decisions, players should be presented with options that carry similar consequences or require comparable levels of skill. This design approach enhances the decision-making process and prevents players from consistently choosing an obviously superior path.

While chance and randomness can inject excitement and unpredictability into games, they should not overshadow the importance of skill. In Player versus Player (PvP) games, where player abilities are directly pitted against each other, skill should be the primary determinant of success. This design philosophy creates an environment where more skillful players are rewarded for their efforts and where victory feels earned.

Balancing a game also involves addressing catch-up mechanics. Providing opportunities for players who fall behind to make a comeback can inject tension and excitement into the gameplay. In competitive settings, preventing early leads from leading to inevitable victories not only keeps the game engaging but also fosters a sense of hope and excitement for the losing player.

Avoiding impasses or stalemates is another crucial aspect of game balancing. Such situations, especially between players with varying skill levels, can be disheartening and undermine the principle that player skill should drive the outcome of the game. By striving for dynamic and engaging gameplay, designers ensure that player skill remains a pivotal factor throughout the gaming experience.

Consistency in difficulty is vital to keeping players invested in the game. Gradual variations in difficulty, rather than abrupt changes, allow players to adapt and grow their skills progressively. Moreover, acknowledging players' improvement and providing suitable challenges can keep them motivated to continue playing and exploring the game's intricacies.

Perceived fairness is a challenging but essential aspect of game balancing. While designers can meticulously craft a balanced experience, players' perspectives can be influenced by various internal and external factors. Social opinion, public sentiment, and other subjective elements can shape how players perceive the game's fairness. Striving for clear and transparent design choices can help in achieving a sense of fairness, even if it might not be entirely within the designers' control.

Balancing in game design encompasses a wide range of activities, from fine-tuning specific aspects of gameplay, such as the damage of a skill, to addressing bugs and other issues that affect the game's overall performance. To achieve balance, thorough testing of the game is essential. Currently, human testing is a common approach, where game testers are employed by bigger studios, and smaller studios rely on platforms like Steam to reach a larger audience for testing.

However, human testing does have its limitations. It may be challenging to gather a diverse and sufficient number of testers, and the testers' preferences may be biased towards a specific genre, potentially missing critical feedback from different player demographics. To overcome these challenges, some researchers, like Politwoski et al., propose the automation of game testing as the next natural step. Automating tests can offer efficiency and scalability, but it requires creating parameters for abstract concepts like fun or balance, which can be complex and challenging.

After a game's release, particularly in online games, developers often release what is known as a "Balance Patch". This patch is intended to address specific aspects of the game that require tuning or adjusting due to player feedback or new discoveries. In PvP games, a new interaction or strategy may emerge that significantly alters the game's balance, making it unfair for players who do not use or counteract it. Balance patches aim to rectify such issues and maintain a level playing field for all players.

Balancing is an ongoing process that developers engage in to ensure that the game remains enjoyable, challenging, and fair to all players. By leveraging a combination of human testing and potential future automated approaches, game designers can strive to create an engaging and well-balanced gaming experience for their audience.[PPG21]

Game balancing is a multifaceted process that requires careful consideration of game mechanics, player skill, and the overall player experience. By adopting a Player-Centric approach and prioritizing meaningful choices and skill-based challenges, designers can create balanced games that captivate and entertain players across various skill levels. Achieving balance is a continuous journey, and as the gaming landscape evolves, so too will the art of game balancing.

2.2.8 Online Gaming

Online gaming is an interesting area that merits exploration. It is not a specific genre or pattern of gameplay but rather a technology that enables player cooperation and competition on a global scale. For the purpose of this discussion, online gaming refers to multiplayer distributed games where players connect through a network, specifically the Internet, as we are not focusing on Local Area Networks (LAN) here.

Online gaming offers a significant advantage in terms of player socialization. The act of socializing within games often enhances the overall enjoyment and experience for players, especially evident in Massively Multiplayer Online (MMO) games. In these MMOs, social interactions play a pivotal role in overcoming challenges and creating opportunities for players to engage in role-playing, which holds immense significance for many individuals. Role-playing provides a form of escapism from reality, allowing players to immerse themselves in an alternate reality and indulge in the act of pretending to be part of a different world.[Cas19]

Online play, especially in PvP games, presents a notable advantage by allowing players to engage in competitions against human intelligence rather than relying solely on artificial opponents. This dynamic adds an element of unpredictability and challenge, as players face off against each other, testing their skills and knowledge of the game. Unlike facing predictable NPCs, players in PvP games are up against the strategic decisions and maneuvers of real individuals, making each match a unique and exhilarating experience. In games such as League of Legends or Total War, while AI may still control most units, it is the player's intelligence and tactical prowess that drives the outcome of the battles. This interactive and dynamic aspect of online PvP gaming elevates the level of engagement and enjoyment for players, fostering a competitive environment that encourages growth and improvement.

One drawback of online gaming is latency. The internet is designed with a focus on redundancy rather than speed, prioritizing the delivery of data without corruption over rapid delivery. While faster internet connections have become more accessible in many households, the issue of latency persists. In turn-based games, this might not significantly affect the outcome of a match, but in real-time games, latency can lead to serious advantages or disadvantages for players. Unfortunately, there is no definitive solution to completely eliminate latency, as it is inherent to the nature of the internet. However, there are some methods that can help alleviate its impact. One popular approach, especially in online games with a vast player base, is to divide players into more localized servers, such as Americas and Europe. This way, players can choose to connect to the server closest to their geographical location, which can help reduce latency.[MKF+21]

Online video games possess a unique characteristic that demands regular maintenance and fresh content updates. The concept of '"Content Drought" has gained attention in recent years, referring to a situation where players or player bases feel there is a lack of engaging activities in the game. This could be due to having already experienced all available content or the absence of replayability value in the existing content. The social aspect of a game plays a crucial role, as players prefer to join a game with an active and vibrant community rather than one that feels deserted or "dead".

To retain their player base and prevent them from abandoning the game, developers frequently offer updates to introduce new and exciting features, such as expansions, game modes, or playable characters. For instance, in POE, a free action role-playing game , players can build their characters using intricate systems to enhance their power. The game's community is known for its openness and helpfulness, and the complexity is often seen as a positive aspect. To keep the game experience fresh, the developers, Grinding Gear Games, follow a 13-week cycle where they release a new league. Each new league introduces fresh mechanics for players to enjoy, along with new skills and a fresh start where players create new characters and engage in a new in-game economy. This approach ensures that players always have something new to explore and look forward to, preventing the game from falling into a content drought.

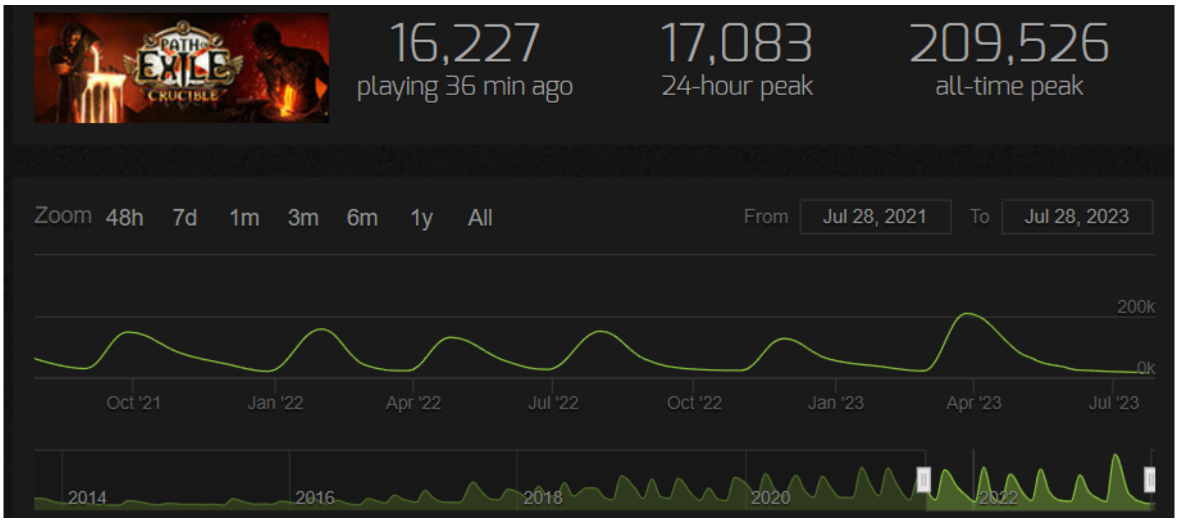

Figure 2.7: Steam statistics for POE.(Accessed: `https://steamcharts.com/app/238960`)

This method is proven effective and can be seen in 2.7. That graphic (in green) represents the amount of concurrent players in POE from July 2021 to July 2023. The peaks in the graphic represent the launch a new League and have the most players, then the players start to get bored and stop playing the game until the next League is released. In 2.7, below the graphic there is a full history of concurrent players since the game's release in 2014. In that graphic is clear that the peaks have been growing since release. Although it is arguable if this method is good or not, since some might prefer a more consistent player base, Grinding Gear Games have not publicly stated that they would be interested in changing their strategy.

2.3 Recommender Systems in Video Games

As of the current date of this document, the search for more specific information about RS and Game Design has yielded limited results. However, a notable example in the realm of video game recommendations is Steam, a platform developed by Valve Corporation. Steam offers a range of features, including account profiles, a game and software store, a library where players can access their purchased games, and community forums for various games. For the purpose of this project, the focus will be on the recommendation feature within the store known as "Your Discovery Queue".

The Discovery Queue offers users a collection of recommended games. When users open the queue, they are presented with ten games they haven't played or encountered before. Once they have browsed through the entire set, they can start a new queue to receive a fresh selection of game recommendations. Although not confirmed, it is assumed that these recommendations are generated based on a user profile that takes into account information such as the user's game library, playtime for each game, preferred game genres, and recent purchases.

Steam encourages users to actively participate in refining the recommendation system. Firstly, users have the option to blacklist specific games or entire genres and subgenres, which affects the games displayed in both the store and the queue. Secondly, when users browse a game, they are presented with three choices:

Figure 2.8: Steam's ternary rating system

- **Add to your wishlist**: This option allows users to add a game to their wishlist, creating a list of desired games. When a game is added to the wishlist, the user will receive an email notification when the game is released or when there is a sale or promotion for the game. Additionally, the user's friends can view their wishlist, which enhances the game's visibility among the user's social circle and potentially increases interest in the game;
- **Follow**: By choosing to follow a game, the user becomes part of a group or community dedicated to that specific game. As a follower, the user will receive notifications on their activity feed whenever new updates, announcements, or content are added to the group. This option provides more consistent updates and information compared to using the wishlist, allowing users to stay up-to-date with the latest developments and news related to the game they are following;
- **Ignore**: Selecting "Ignore" signals to the platform that the user is not interested in the specific item, and it is then added to a list for future reference. The user has the flexibility to revisit this list at any time and uncheck items that they might want to reconsider. Upon ignoring an item, there are two options available to the user:
  - Default Ignore: When an item is ignored using this option, it will no longer be displayed on the store, and it will not be considered in generating future recommendations. Essentially, it is permanently removed from the user's visible recommendations.
  - Hidden Ignore: With this option, the item is hidden from the store, meaning it won't appear in regular browsing or searches. However, it can still be used to influence the generation of future recommendations. This option provides the user with the choice to hide items they are not interested in from immediate view while still contributing to the platform's understanding of their preferences for future suggestions.

These three options are therefore a type of explicit rating that the user can give to a game and, in addition to the the blacklist, will help improve future recommendations.

In 2022, a research study titled "Recommender Systems for Online Video Game Platforms: the Case of STEAM", conducted by Cheuque et al., aimed to evaluate various recommendation algorithms on a set of

Steam games. Steam allows users to review games by providing brief comments and giving them a thumbs up or down. These reviews were utilized for sentiment analysis, which involves converting opinions into numerical measures to represent the level of positivity expressed.

The study employed several models for the evaluation, including Alternating Least Square (ALS) for matrix factorization, Factorization Machines (FM), Deep Neural Networks (DNNs) and Deep Factorization Machines (DFMs). The DFM model integrated a layer of DNN for feature learning, working in parallel with a layer of FM, which excels in recommendation tasks. These models were trained with and without the incorporation of sentiment analysis. The findings revealed that DNN, even though it was comparatively slower than some other models, demonstrated superior performance in terms of both recommendation novelty and diversity. On the other hand, the sentiment analysis using the pool of reviews might have been limited, potentially leading to noise rather than assisting in the recommendation process.

Overall, this study sheds light on the effectiveness of different recommendation algorithms for online video game platforms, offering insights into the importance of selecting suitable models and leveraging sentiment analysis to enhance the recommendation accuracy and relevance [CGP19].

Yang et al. conducted a study focused on game recommendation and identified three crucial characteristics that should be harnessed for optimal results: personalization, game contextualization, and social connection. These elements play a significant role in enhancing the user experience and engagement within the game platform.

In their research, the authors utilized sentiment analysis to gain insights into user preferences and opinions, a valuable tool for understanding player sentiments towards games. Furthermore, they introduced a novel model called the Social-Aware Contextualized Graph Recommender System (SCGRec), which proved to outperform other Graph Neural Network (GNN) in terms of recommendation performance. Despite its simplicity, SCGRec demonstrated remarkable flexibility and effectiveness in incorporating side information, which holds great relevance in the game industry. By considering social connections and contextual information related to games, the SCGRec model provides more personalized and engaging recommendations.

This research contributes to advancing the field of game recommendation by highlighting the importance of personalization, game contextualization, and social connections in designing effective RS. Additionally, the introduction of the SCGRec model showcases how sentiment analysis and graph neural networks can be integrated to enhance the overall game recommendation process [YH19].

3

Application

3.1 Godot

Godot is a cross-platform, free and open-source game engine released under the permissive MIT license. The Godot engine supports game creation on multiple platforms, including Windows, macOS, Linux, BSD, and Android. It allows developers to deploy their games directly to Windows, macOS, Linux, and Android. Additionally, for console platforms such as Nintendo Switch, PlayStation 4, and Xbox One, deployment can be achieved through third-party providers.

The basic features Godot provides for any project are the following:

- **Scripting**: In Godot, scripting is a vital part of creating gameplay mechanics and behaviors for the nodes. Godot supports multiple scripting languages, including GDScript (a Python-like language designed for Godot), C#, and visual scripting using the built-in visual script editor. Developers can choose the language that best suits their preferences and project requirements.
- **Scene Instancing**: Godot uses a concept called "instancing" to create multiple instances of the same scene. This is particularly useful when you need to duplicate objects or create instances of

complex scenes with shared functionality. Instancing allows for memory-efficient reuse of nodes and is beneficial in optimizing game performance.

- **Signals**: Nodes can communicate with each other using signals. Signals enable event-based communication, where a node can emit a signal when something significant happens, and other nodes can listen for and respond to those signals. This system facilitates decoupled and flexible interactivity between nodes.
- **Physics Engine**: Godot comes with a built-in physics engine that allows developers to create dynamic and realistic physics simulations for 2D and 3D games. This engine supports collision detection, rigid bodies, and various physics constraints, making it easier to create interactive and immersive environments.
- **Resource Management**: Godot offers a resource management system that allows users to organize and manage various assets like textures, audio files, 3D models, and more. These resources can be imported, modified, and shared across multiple scenes, enhancing the efficiency of the development process.
- **Exporting and Deployment**: Godot provides a range of export options for different platforms, including Windows, macOS, Linux, Android, iOS, web, and various game consoles. Developers can package their games for distribution on multiple platforms seamlessly.

3.1.1 Project Structure

Godot adopts a programming approach similar to Object Orientation, known as Nodes. These Nodes serve as the fundamental building blocks of any project and can be organized in a tree structure, where one node can be assigned as the child of another. Each Node must have a unique name within its parent. This collection of Nodes forms what is called a "scene," which can be saved to disk and then integrated into other scenes, granting significant flexibility.

During runtime, the active tree of nodes is managed by the SceneTree. Godot comes with a range of pre-built nodes, covering everything from 2D and 3D elements to Control nodes for user interface components. Additionally, users have the freedom to create their own custom nodes either from scratch or by extending existing nodes through inheritance. This allows developers to construct complex and diverse game environments using a combination of these Node types.

3.1.2 Types of Nodes

In the Godot engine, there is a collection of pre-built nodes available in the editor for easy use. They are the fundamental building blocks used to construct scenes. Each Node represents an entity or functionality, and they can be organized into a hierarchical tree structure. Nodes can be assigned as children of other nodes, and this parent-child relationship forms the basis for scene composition. There are various types of nodes available in Godot that cater to different aspects of game development.

Spatial Nodes (3D Nodes) are designed for use in 3D scenes and deal with spatial transformations. They represent objects in the 3D world and handle position, rotation, and scale. Spatial nodes can form complex 3D environments and are essential for creating 3D games.

2D Nodes are specifically used in 2D scenes and games. They handle 2D transformations and rendering, such as positioning, scaling, and rotation in 2D space. They are the main type of node used in this work. Examples include:

- **Sprites**: 2D image or texture-based nodes used to represent visual elements. They can display static images or be used to animate characters, objects, and other visual elements. Sprites are often used for characters, items, backgrounds, and any other 2D visuals in the game.
- **KinematicBody2D**: A physics node used for implementing custom movement and collision handling in 2D games. It allows to control its position and movement manually, responding to physics and collision signals.
- **Tilemaps**: Specialized 2D grids used to create levels, maps, and tile-based game environments. Instead of using individual Sprites for each tile, a Tilemap allows to efficiently lay out and manage large numbers of tiles, such as in a platformer or a top-down game. This makes level design and editing much more convenient.

Control nodes are responsible for managing the UI elements and their interactions. They handle user input and manage UI components like buttons, labels, text boxes, and more. These nodes are crucial for creating interactive and user-friendly interfaces.

In addition to these, there are other essential nodes like AnimationPlayers for animations, AudioPlayers for audio management (which were not used in this application since that wasn't the focus of the research), Timers for event scheduling, and Viewports to control camera views, among others. As mentioned before, nodes can be further customized and extended through inheritance. Developers can create custom nodes by either starting from scratch or by extending existing nodes. Inheritance allows the new node to inherit the properties and behaviors of the parent node, enabling code reuse and promoting a modular and organized project structure.

3.1.3 Signals

Signals in Godot are a way for nodes to communicate with each other when specific events occur, such as a button press or a character getting hit. When a node emits a signal, other nodes can connect to that signal and execute a function or respond to the event. This mechanism enables nodes to interact and respond to each other's actions without creating direct references between them, which promotes loose coupling and code flexibility.

For example, if a player's character emits a "hit" signal when it gets attacked, other nodes, such as an enemy AI or a health bar, can connect to that signal. When the character is hit, these connected nodes can react accordingly by reducing the character's health, playing a sound effect, or triggering an animation.

Using signals simplifies the development process and enhances code organization, as nodes can focus on their individual responsibilities without worrying about the specifics of other nodes they interact with. This decoupling of nodes contributes to cleaner and more maintainable code, making it easier to manage and extend the game's functionality as it evolves.

In Godot, built-in nodes often come with pre-defined signals that are triggered when specific events occur. For example, a Button node comes with a "pressed" signal, which is emitted when the button is clicked. These signals can be easily connected to other nodes' scripts through the editor. Additionally, developers can create custom signals or overwrite existing ones by declaring them within a script. For instance, you can define a custom signal named "my_signal" like this: signal my_signal. There are two ways to use signals: connecting them through the editor or programmatically.

In some cases, connecting signals dynamically through code can be more advantageous. To achieve this, users obtain a reference to the node that emits the signal and then call the connect function. The connect function requires three main parameters: the signal in the node that developers want to connect to; the target object to which the signal should be connected (using self if calling it within the target object itself); the name of the function that will execute when the signal is emitted. Optionally, users can also pass additional parameters after the function name to send arguments or data from one node to another during signal emission and handling.

Lets take a look at a realistic example. The player is at a shop and wants to buy an item. When he presses the item, he gets the item selected and its price is deducted from the player's inventory.

The signal is defined in the item's script like this:

```
signal buy(price)
```

For an item with price 100, when the item is clicked it emits the signal like so:

```
emit("buy", price)
```

In the player's script the connection to the signal can be defined as:

```
shop.connect("buy", self, "item_bought")
```

This will connect the signal "buy" from the node "shop" to the player and run the code in the function "item_bought". The function can be implemented in the player's script and will look something like this:

```
func item_bought(price):
    if player_money >= price:
        player_money -= price
```

Where "player_mone" is a variable that holds all the money the player has available.

3.2 Supergame

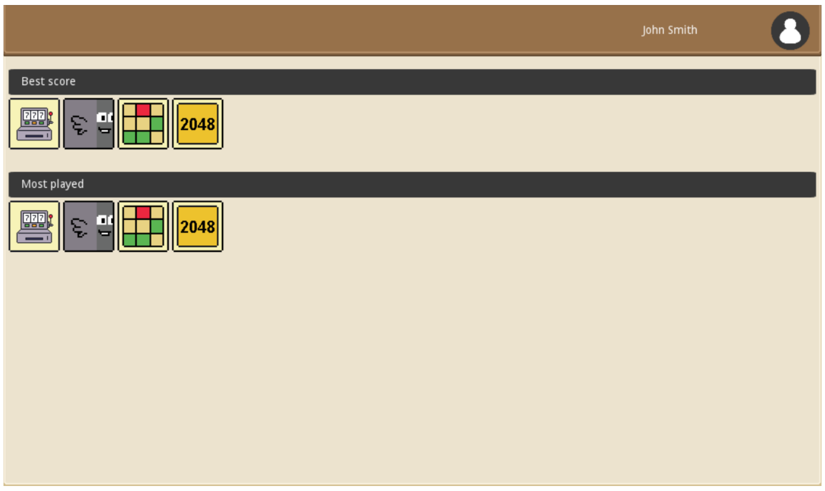

Figure 3.1: The Supergame Main Menu.

The Supergame is the overarching game and core of the project. It manages the casual games that the players can play, their accounts/profiles and the RS.

The Supergame is constituted by three main components:

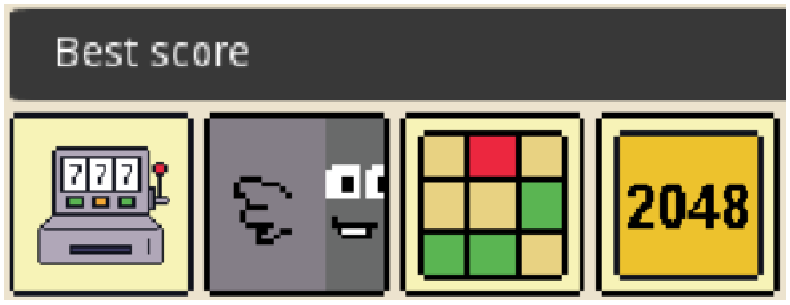

Figure 3.2: Panel of recommended games ordered by score with highest being to the left

- **Recommendation Panels (RPs)**: This node serves as a manager for all the game recommendations provided to the player. Its children are the individual recommender panels, which are separate scenes responsible for arranging the UI elements within a single panel. At present, the game includes two panels that recommend games based on the number of times played and the highest score achieved. Each panel is structured with a vertical box containing two sections. The top section consists of a simple bar texture and a label displaying the panel's title, such as "Most Played Games". Below the title, a scroll container is present, allowing its child elements to be scrolled if they exceed the screen's dimensions. Within the scroll container, an Horizontal Box container is utilized to populate the game icons themselves. To populate the panels with the sorted games, the RP is responsible for communicating with the RS component.. The RS calculates and sorts the games based on specific criteria and then passes this information to the RP, which in turn updates the panels accordingly with the appropriate game recommendations.

- **RS**: This serves as the logic layer of the Supergame. It is a generic node equipped with a script that handles all recommendation-related logic in the game. Currently, it features three functions: a sorting function and two recommendation functions. Upon starting up the Supergame, the RP requests the RS to provide recommendations based on the most played games and highest scores. The RS then processes the data and returns an array of games sorted from highest to lowest. The design of the RS allows for easy scalability and addition of new panels in the future. For example, if a new panel is needed, such as "Popular Games by Genre", it can be effortlessly implemented. All that is required is to create the new panel and add a corresponding function in the RS to return the sorted games. This modular approach enables the Supergame to adapt and expand with minimal effort when incorporating new recommendation criteria.

- **Profile Button**: Situated at the top right corner of the screen. When clicked, it reveals a dropdown menu with three options: "Account", "Settings", and "Log out". Although the game currently does not support multiple accounts, it is intended to do so in future versions. Hence, the button is already in place to provide users with the ability to manage their account-related information when the feature is added. Similarly, the "Settings" button is designed with future utility in mind. It is meant to allow users to customize various game settings, such as screen size, resolution, language, and themes, once these features are implemented. The functionality of the "Log out" button works as follows: When the user clicks it, a dynamic Popup element is created and added to the scene. This popup presents two options to the user. If they press the "Ok" button, they will exit the game. On the other hand, if they choose the "Cancel" or "X" button, the popup will close without closing the game, providing a way to abort the logout process.

The Supergame Scene includes other essential elements worth mentioning. There are two Texture Rectangles (TextureRect node) in the scene. The first one holds the background image, providing the visual

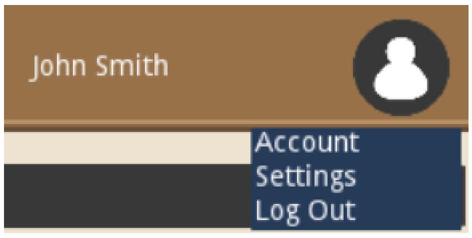

Figure 3.3: Profile button with its dropdown menu and options

background for the Supergame Scene. The second one serves as the bar at the top of the screen, containing the Profile Button (shown in brown in 3.3). The Profile Button is situated within the bar at the top of the screen. When clicked, it displays a dropdown menu with options related to account management and settings, as explained earlier. Above this bar, there is a Label that displays the user's name. Since account functionality is not yet set up, this Label acts as a placeholder, containing a default name like "John Smith". The Supergame Scene also includes a Popup element, which is initially hidden by default. When the user presses a game they wish to play, this Popup is shown. It serves as a confirmation dialog, asking the user whether they indeed want to start the game they selected. The Popup provides buttons to either proceed with starting the game or to cancel and abort the action.

Most of the nodes in the Supergame Scene are Control nodes. This choice is due to the interactive nature of the elements within the scene. Control nodes are specifically designed for managing player interaction and arranging UI elements. One important characteristic of Control nodes is that they feature a bounding rectangle that defines their extent, anchor, and position relative to their parents. This feature greatly facilitates changes and adjustments to the project's UI layout. Since Control nodes' arrangements are always defined by their parent nodes, modifications to the parent nodes will automatically propagate and affect the layout of their child Control nodes. This inherent hierarchy and positioning system of Control nodes promote ease of development and maintenance. By using Control nodes, developers can efficiently manage and arrange UI elements in a flexible and consistent manner, making it simpler to update and improve the user interface as the project evolves.

When the player starts the game, a splash screen[1] will appear. This can be done by going to Project $\rightarrow$ Project Settings $\rightarrow$ Application $\rightarrow$ Boot Splash and setting an image resource. For more complex splash screens, instead of using the above method, a dedicated scene tree can be created and instanced when the game starts. This allows for animations or sounds to be played in the screen. After a timer expires or the resources for the game have finished loading, the game can change to the main scene tree. In this project, the decision was made to display a static image using the first method.

3.2.1 FloppyBird

Floppy Bird is a game where the player controls a character by tapping the screen to make it flap and fly upward. The objective is to navigate through gaps between walls that move from right to left, scoring points for successful passes. Colliding with a wall or going out of bounds results in the game ending.

The root node in this scene is a 2D node named "World", a common convention in Godot for root scenes, often combined with the word "Game". This node takes charge of managing its children nodes, handling tasks such as setting their initial visibility in the viewport (whether they are drawn or not) and positioning them correctly. Moreover, the "World" node plays a crucial role in reloading the scene or returning to

[1]A splash screen is a graphical screen or image that appears briefly when a software application or game is launched to provide visual feedback to the user during the loading process.

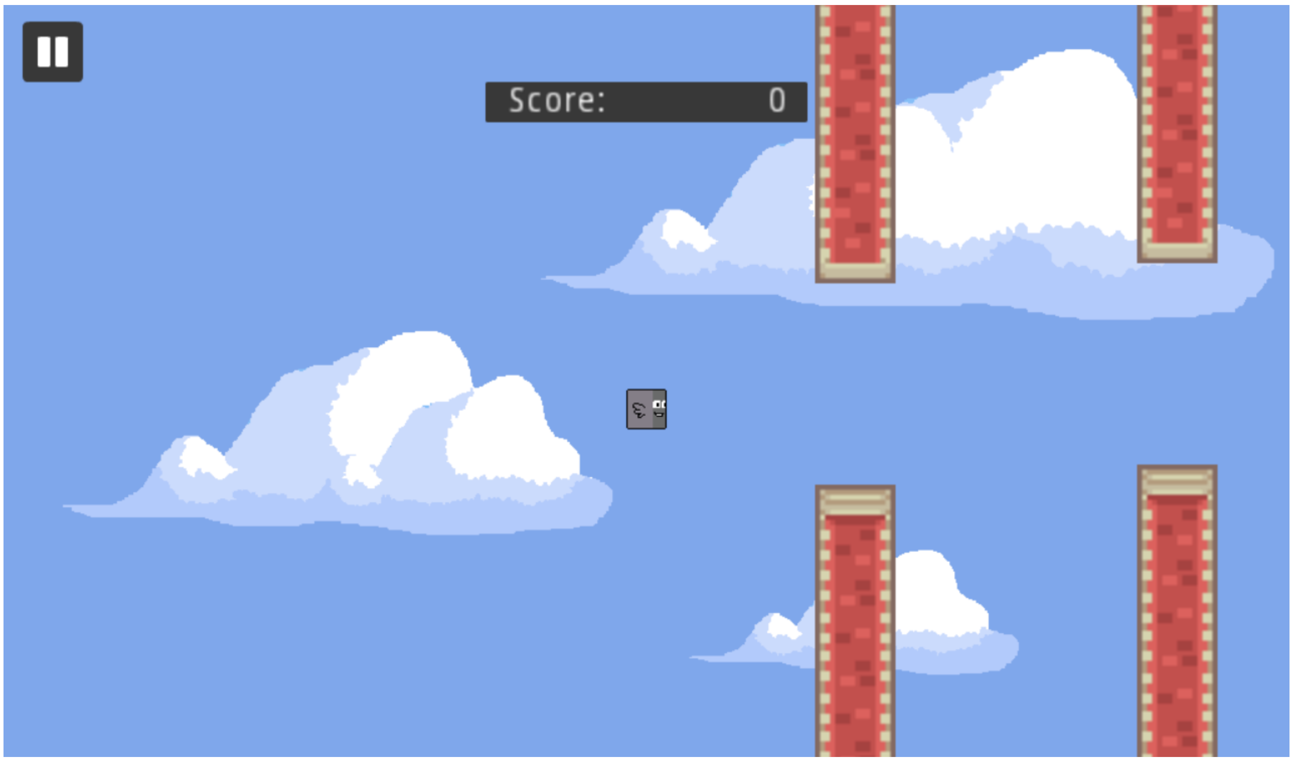

Figure 3.4: Screenshot of the mini game Floppy Bird. Contains the following UI elements: Player Character; Pause Button; Camera; Wall Nodes; Background Image and Score.

the main menu (Supergame SceneTree) based on user actions, particularly when the player loses the game or pauses it. When it receives the "game over" signal, it saves the current score before proceeding with appropriate actions: reloading the scene or transitioning back to the main menu.

Core Mechanics

In this section, we will outline the core mechanics that drive the gameplay. We will focus solely on the elements that directly impact the game experience, excluding any UI interactions that do not affect the gameplay. The values mentioned were derived from testing to create a game that feels good to play. The testing process might not have been extensive. It's essential to note that these values are not fixed and may require adjustments in the future to further optimize the gameplay experience.

1. The character experiences a constant downward force of 10 units, causing it to continuously fall with a maximum speed capped at 200 units;
2. When the player presses the "Flap" button, the sprite moves upwards with a force of 175 units;
3. The player always begins with a score of zero, and each time they successfully pass through a wall, the score increments by 1;
4. When the player collides with a wall, the game is over, and the score is displayed. At this point, the player is presented with the option to either start another game or return to the main screen;
5. When the player goes out of bounds (out of the screen), either up or down, the game is over;

6. The walls appear from the right side of the screen and disappear when they leave through the left side of the screen;
7. The walls move horizontally at a constant speed of -2.5 units per frame. (This negative value indicates that they move from right to left on the screen)

UI

Here, all the UI elements present in the game scene are listed. Figure 3.4 shows a picture of the game.

Element	**Description**
Background	The background is a 977x550px static PNG image serving as the visual backdrop of the game scene.
Player	A 32x32px PNG image at the screen center, used as player sprite with collider for wall collision.
Score	TextureRect node which displays the current score in the top middle part of the screen.
Countdown	A timer node is used to pause the game upon start, preventing the character from falling immediately. It initiates a three-second countdown and resumes the game when it reaches the end.
Walls	There are six walls, each equipped with colliders, arranged with open spaces between them. Figure 3.4 illustrates two walls positioned on the right side.
Camera	A node restricts the player's viewport to only display what the camera shows, limiting visibility to 50% of the scene. This node enforces the out-of-bounds rule, leading to a game over if the character goes beyond the camera boundaries.
Pause	A custom button that pauses the game state, further explained bellow.
Game Over (Popup)	Appears at game end with "Game Over!" title. Offers try again (reload scene) or cancel (return to main menu).

Table 3.1: UI elements present in Floppy Bird

In the future, more features can be added such as: sound effects for actions like flapping, wall collisions, and score increments, and background music that complements the game's theme and adds to the atmosphere.

Walls

The walls act as obstacles in the game, where the player can collide with them, resulting in a game over, or successfully pass through them to score a point. A wall is represented as a static body node (StaticBody2D). These nodes are designed to remain stationary (static), making them perfect for walls and platforms that do not move. However, they can be assigned a constant linear or angular velocity to impact bodies colliding with them, making them suitable for the game.

The node has three primary children. The first two, the upper and lower walls, behave similarly. Each consists of a sprite and a CollisionShape2D. The CollisionShape2D is rectangular to match the sprite, and it marks the collision area that ends the player's game if they come into contact with it. Between the upper and lower walls, lies a Point Area (Area2D) node. When the player enters this area, a point is scored. The Area2D node requires a CollisionShape2D child to detect collisions, enabling it to register when the player has entered this area. Consequently, a collision area in the shape of a rectangle was added between the upper and lower walls.

In this game, the player only moves up and down. The walls are the ones moving from right to left, creating the impression of the player moving forward. To achieve this, velocity is assigned to the walls. Attaching a script to the Wall node allows definition of the velocity. In the _physics_process() function, the velocity is added to the node's position. Similar to the _process() function, _physics_process() executes its code every frame, synchronizing it with the physics renderer. Consequently, the walls move a specific distance to the left each frame. Adjusting the velocity directly affects the game's difficulty. By increasing or decreasing the velocity, the walls can be made to move slower or faster, respectively, thus influencing the game's level of challenge.

In practice, the Wall script looks like this:

```
extends StaticBody2D

export var difficulty = Vector2(-2.5, 0)

func _physics_process(_delta):
    position += difficulty
```

In this code block, more Godot engine's syntax can be observed. A Wall is a type of static body, and it extends StaticBody2D. "Export" is used to make the "difficulty" variable accessible in the Editor. If the "difficulty" value needs to be modified, there is no need to rewrite the code. A Vector2 is a tuple holding the x and y coordinates. Since the position of the wall is also represented as a tuple with x and y coordinates, the "x" variable is used to move the wall. In the _physics_process() function, the velocity is added to the position during each frame. The "delta" variable represents the time elapsed within one frame.

Player

This subtree manages character behavior, score tracking, and wall resetting. Let's break it down to gain a better understanding. To facilitate the character's movement, the system relies on four constants: FLAP (determining how much the character ascends when flapping), UP (indicating the flap direction), GRAVITY (representing the strength of the downward force affecting the character), and MAXFALLSPEED (defining the maximum falling speed attainable).

In the *_physics_process()* function, the system applies the force of GRAVITY to the player every frame, causing the character to move downwards with a maximum velocity of up to MAXFALLSPEED. Upon pressing the "flap" action, the FLAP value is added to the character's motion, propelling it upwards. Subsequently, the *_move_and_slide()* function, a physics node function, is employed to handle collision behavior and, more importantly, to smooth out the character's velocities. As a result, the player experiences gradual deceleration or acceleration after flapping, instead of an instant upward motion followed by an immediate fall due to GRAVITY's influence on the body.

Another child of the Player node is "Detect", an Area2D responsible for detecting if the player passes through the gap in the wall or collides with it. As mentioned earlier, this area requires a collider, shaped to match the character, which is a square. When the player "enters" the gap in the wall, it emits a signal and increments the score. On the other hand, if the player collides with any part of the wall (either the upper or lower part), it emits a signal that marks the game over and sends the score along with it.

The Resetter is an Area2D placed at the left side of the screen, outside the player's sight, and it has a collision shape. When a wall collides with the Resetter, it emits the *_body_entered* signal, which is connected to the player node. Upon the signal being fired, that specific wall is released from memory, and a new wall is created at a predetermined position on the right side of the screen. By using this node, there are consistently five walls instanced, continuously approaching the player.

In 3.5 the Resetter is to the left of the background image (the blue rectangle with the clouds). The Walls are to the right and the spaces between upper and lower wall have a gap with scoring Area2D.

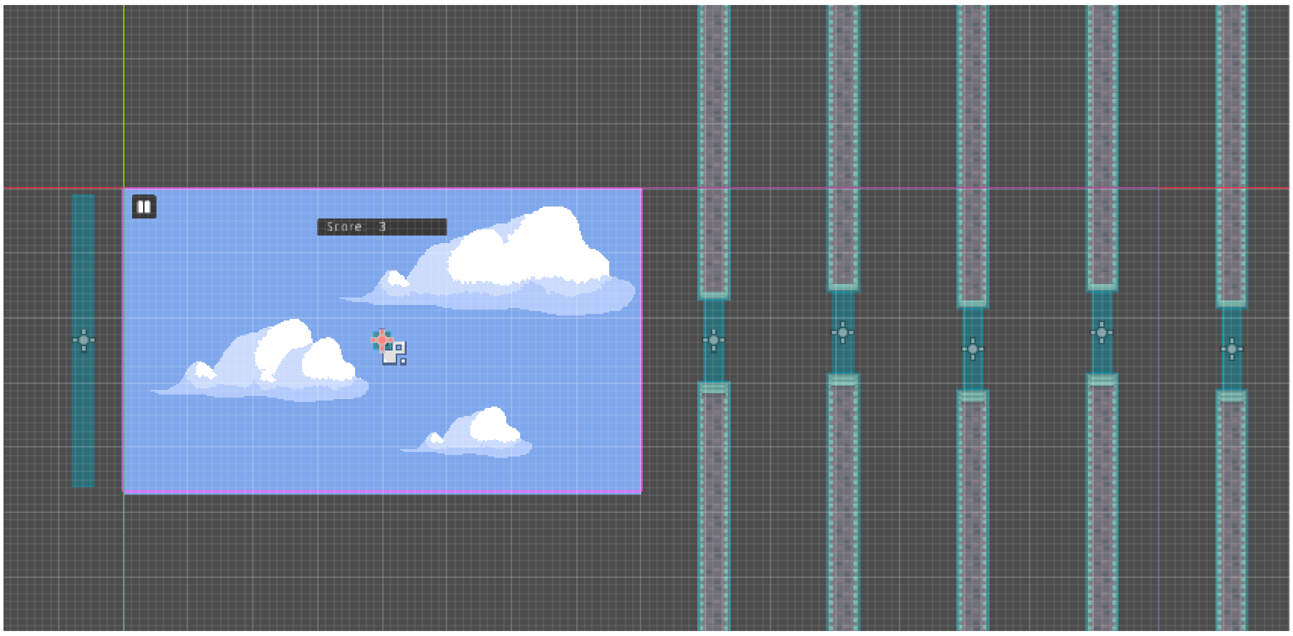

Figure 3.5: Screenshot taken from Godot. The rectangle with clouds is the camera where the game takes place. To the left is the Resetter and to the right the Wall nodes.

Pause Button

Godot provides a feature that allows the user to save a subtree as a scene, which can then be instantiated in other scenes. This particular is just a module, independent from other scenes, designed to be used in whatever scene the user sees fit. It consists of a TextureButton node, serving as the button itself, and a ConfirmationDialog node, which represents a popup.

The TextureButton acts as a toggle button. When pressed, it pauses the scene tree, effectively pausing the game. Additionally, it switches its texture from the pause icon to the resume icon and displays the popup window. The popup appears at the center of the screen with the title "Paused" and the message "Go back to the Main Menu?". If the player selects "Cancel" or clicks the "X" in the popup, the popup hides itself, and the texture in the pause button displays the pause icon again, resuming the game (i.e., the scene tree is no longer paused). However, if the player presses the “Ok” button, they are taken to the main menu.

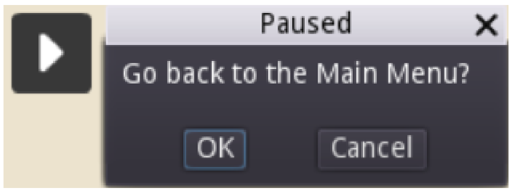

Figure 3.6: **Left**: Pause Button. **Right**: Popup that shows when the button is pressed

To create this button is a rather simple task. Lets focus solely on the logic behind it and not on the UI behaviour. The references to the TextureButton and the ConfirmationDialog nodes can be accessed like so:

```
onready var button = get_node("TextureButton")
onready var popup = get_node("ConfirmationDialog")
```

A variable is only assigned a value when the node is instanced. Onready is a keyword used to set the value of a variable only when that node is ready/instanced. It is a shortcut for:

```
var button = null

func _ready():
    button = get_node("TextureButton")
```

There are two ways of pausing the game, either the user clicks the button with the mouse cursor or they press the Escape key on the keyboard. Godot provides an easy way of mapping controls by going to Project → Project Settings → Input Map. The Escape key is mapped to the Input variable “ui_cancel”. The user can input this action whenever he wants, the button can listen to the input in the _process() function:

```
func _process(_delta):

    if Input.is_action_just_pressed("ui_cancel"):
            _on_TextureButton_pressed()
```

Whenever the Escape key is pressed the code in the function _on_TextureButton_pressed() will run. If the user clicks the button with the mouse cursos, the same funtion will run.

```
func _on_TextureButton_pressed():
    if not get_tree().paused:
        get_tree().paused = true
        pause_popup.popup()
```

If the SceneTree is not already paused, the scene pauses and the popup pops up. The popup() function simply makes the popup appear on the screen. To cancel the popup there are two buttons: the Cancel button that comes from the ConfirmationDialog node and the Close button that it inherits from the WindowDialog node. The buttons’ signals can be manually connected in the ready function:

```
func _ready():
    popup.get_cancel().connect("pressed", self, "_on_canceled")
    popup.get_close_button().connect("pressed", self, "_on_canceled")
```

Since the behaviour of both buttons is the same, the signals connect to the same function _on_canceled().

```
func _on_canceled():
    get_tree().paused = false
```

The popup has another button, Confirm. This button represents the “OK” button in the popup that changes the SceneTree from a game to the main menu, which can be achieved by connecting the signal in the script and executing the function _on_comfirm():

```
func on_confirm():
    get_tree().paused = false
    get_tree().change_scene("res://MainMenu.tscn")
```

An essential point to consider is that when transitioning from the game scene to the main menu, the user should also “unpause” the scene tree. If they fail to do so, the scenes will transition while still in a paused state, limiting interaction.

3.2.2 Slots

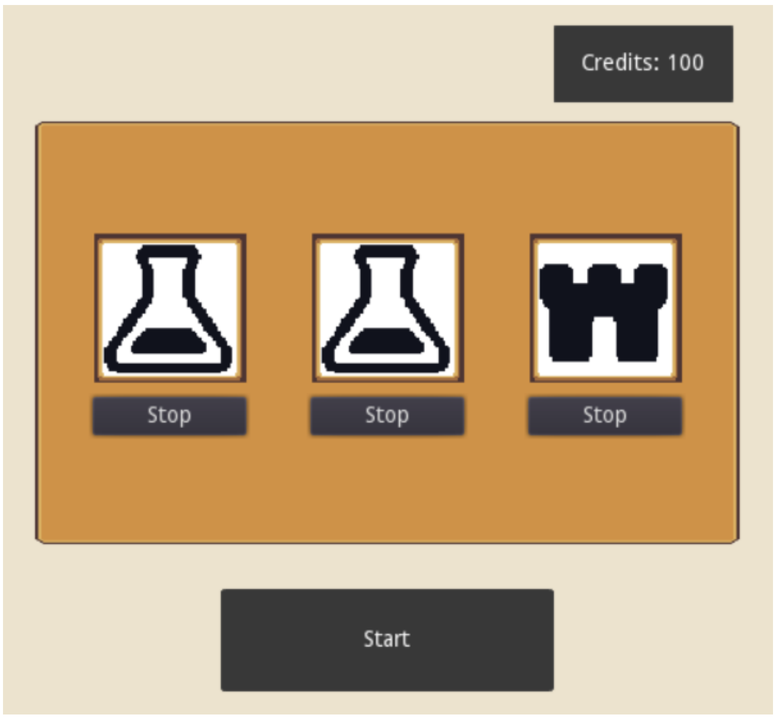

Figure 3.7: Screenshot of the mini game Slots. Contains the following UI elements: Background Image; the Player’s Credits; the Slot Machine; the three slots wheels; the Start Button.

In this game, the player is presented with a slot machine featuring three wheels. By default, the player starts with 100 credits and can spend 10 credits to initiate the slot machine by pressing the ”Start” button. Once activated, the wheels begin to rotate, and the player must press the ”Stop” button located under each wheel to bring them to a halt. If the player successfully stops all three wheels and aligns them with the same icon, they win 40 credits. However, if they fail to do so, nothing happens, and they can start a new game as long as they have enough credits to cover the cost. Upon exiting the game, the player’s score is determined based on the credits earned during playtime.

The game is composed of a background image represented by a sprite, the Slot Machine, including its Wheels, a Pause Button (the same scene used in Floppy Bird), and a Player. The following subsections explore the structure of the game in detail.

Core Mechanics

Here is the outline of the core mechanics that drive the gameplay. It's essential to note that values such as credits mentioned in this game are arbitrary and were defined during the game's implementation. They are included explicitly here to ensure that the core mechanics feel as authentic as they would be in any typical video game project. In the future, if adjustments to any values are necessary, any developer should have the flexibility to make those changes.

1. The player always starts with 100 credits;
2. Playing a round of Slots will cost the player 10 credits;
3. Should the player lack sufficient credits to play a round, the game concludes;
4. Upon winning a round, the player receives 40 credits, which can be used in subsequent rounds;
5. At the game's outset, all wheels must be in a stopped state. When the player initiates the game, all wheels simultaneously begin spinning at a speed of 5 Frames Per Second (FPS);
6. After the game starts, the player has the ability to stop each individual wheel, and once a wheel is stopped, the player cannot make it spin again;
7. When all wheels come to a stop, the game concludes. If all three wheels display matching symbols, the player wins the round; otherwise, they lose the credits.

UI

Table 3.2 lists the UI elements present in Slots. Although the UI is not very complex, being mostly built for representing the game and give feedback to the player regarding gameplay, this game has a lot of room for improvement when it comes to the UI.

As the game draws inspiration from a casino slot machine, incorporating visual and audio elements that evoke the authentic casino experience could be captivating. Audio effects could include the distinct sounds of the wheels spinning, the game coming to an end, credits being deducted or added, and celebratory sounds for special symbol combos. Animations should be visually appealing and leave a lasting impression on the player. By integrating these elements thoughtfully, the game can provide a more immersive and engaging experience, enhancing the player's enjoyment and involvement.

Wheel

In this project, a wheel is a node that comprises two children: an AnimatedSprite and a Button. The AnimatedSprite node facilitates animation creation using a set of sprites. For this work, a sprite sheet containing nine unique images is utilized to generate the animation. By partitioning the sprite sheet into squares of equal size, nine pictures, each representing a frame in the animation, are obtained. This enables the creation of the Rolling animation, wherein different icons are displayed in each frame as it rolls at 5

Element	**Description**
Background	The background is a 977x550px static PNG image serving as the visual backdrop of the game scene.
Credits	In the game, the score is represented as credits and is visibly displayed atop the slot machine using a Label node. The Label node updates dynamically whenever credits are earned or lost during gameplay.
Notices	Notices that inform the player of the game's outcome, such as "You have won 40 credits!" and "No luck this time. Try again!", are displayed above the three slot machine wheels. These notifications provide feedback to the player about their performance in the game.
Slot Machine	The machine space is represented by a darker brown PNG image (see Figure 3.7).
Wheels	The game features three wheels that the player will use to play. Each wheel contains a sprite sheet that changes the displayed image at a regular interval of 5 FPS.
Start	This button initiates the game by setting all wheels into motion, causing them to start spinning.
Pause	A button that when pressed, triggers a popup giving the player the option to return to the main menu.

Table 3.2: UI elements present in Slots

FPS. The speed at which the wheels roll determines the level of challenge of the game. If they roll faster the game gets harder, otherwise it gets easier.

The button is straightforward in its functionality. When the player presses the button, it emits the _pressed() signal, which is connected to the wheel. Consequently, the Rolling animation halts, and the wheel emits an is_stopped() signal connected to the slot machine, which includes the index of the frame where it came to a stop.

A node becomes ready when both itself and its children have entered the scene tree. In a node with children, the_ready() callbacks of the children are triggered first, and only after that does the parent node become ready. The _ready() function is typically used for initialization purposes. For the wheel, when it becomes ready, it registers itself in its parent node, which is the slot machine. Additionally, it randomizes its animation frames so that each wheel displays a different sequence of images. This function also establishes signal connections between the wheel and the slot machine, specifically connecting the is_stopped() signal. Furthermore, it stores a reference of the wheel in an array of the slot machine. This allows the slot machine to manage all the wheels dynamically. If the need arises to add or remove wheels in the future, duplicating an existing one will automatically register the new wheel in the machine. Consequently, the slot machine will keep track of all its children, and its functions will work seamlessly with any number of wheels.

Slot Machine

In the current state of the game, the Slot Machine includes three wheels, a sprite serving as the background where the wheels are displayed, a start button, and a pause button. Similar to Floppy Bird, the pause button is present but with a slight difference. In this game, the player's progress is only over when they run out of credits, and since the potential maximum credits they can acquire is "infinite", there has to be a way to save the score/credits earned in a single session. To address this, when nodes are exiting the scene tree (when their memory is being freed), a signal is emitted, that signal is utilized to save the game score. This ensures that the player's progress is stored and can be retrieved when needed.

On pressing the Start button, if the player has enough credits to play, those credits are debited from the player and the wheels start rolling.

The slot machine is responsible for managing the game itself. It registers the wheels by storing their references in an array and then proceeds to randomize them. After the wheels start rolling and the player has stopped all of them, the slot machine evaluates whether the player has won or lost. To determine the outcome, the slot machine goes through the array of wheels and checks if the current frame displayed is the same in all the wheels. If there is a match, it means the player has won credits; otherwise, they have not won. When the game is over, a message is displayed on the screen, informing the player about the result of the game.

Player

This node serves as the representation of the player in the game. The script is attached to the UI element displaying the player's credits on the screen. The player node is responsible for managing the player's funds. It includes functions to add or subtract credits and also checks if the player has enough money to play the next game.

Additionally, this node takes charge of saving the game, which involves storing the score or credits earned, particularly when the game exits the scene tree. This way, the player's progress and score are preserved for future gameplay.

3.2.3 Wordle

Wordle is a game where the player has a maximum of six attempts to guess a secret five-letter word. In the first attempt, they have no hints, so they usually have to write down a random word, and the game will provide feedback on the letters used in the attempt. The feedback indicates if the letters don't belong to the word, if they belong to the word but are misplaced, or if they are correctly placed. The player must use this feedback to deduce the secret word and achieve a better score with fewer tries.

In developing this game, our improved knowledge of Godot allowed us to organize it more efficiently. We divided the game into four different layers, which we will explain briefly before delving into the specific functionalities of each node and the overall flow of the game. The layers are presented from the bottom to the top, starting with handling resources such as files and concluding with the UI composition.

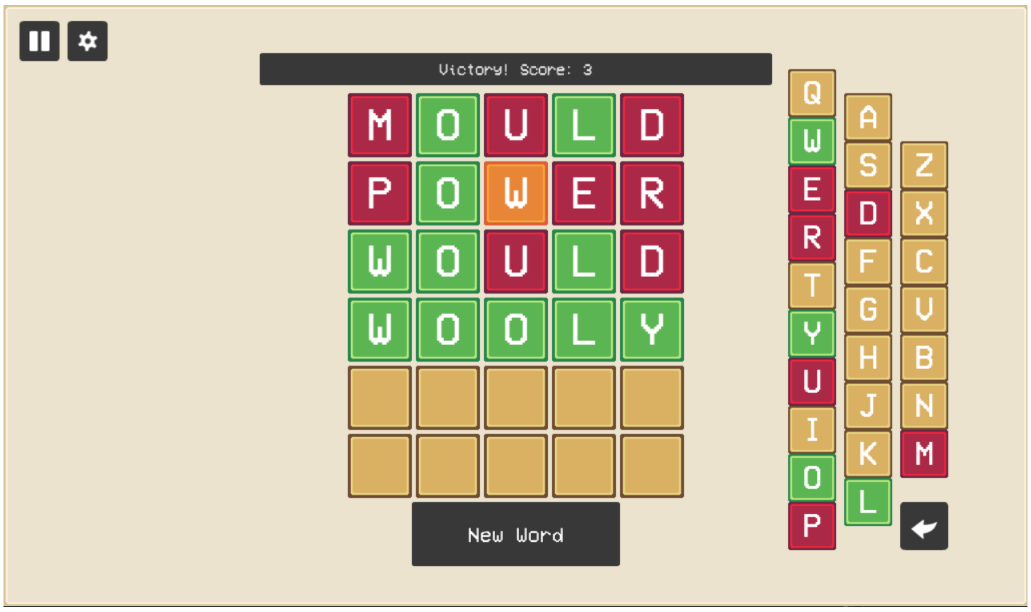

Figure 3.8: Screenshot of the mini game Wordle. Contains the following UI elements: New Word Button; Notice; Hints; Backspace Button; Guesses; Background Image; Pause and Settings Button.

Core Mechanics

In this particular game, there are no specific units to consider. However, when referring to aesthetic aspects, such as the highlight color of the letters, it's important to note that these choices were made arbitrarily and might be subject to change in future iterations of the project.

1. After the game starts, it randomly selects a word from a predetermined dictionary, and that chosen word becomes the secret word for the current round;
2. During the game, the player is allowed six attempts to guess the secret word;
3. If the player successfully guesses the secret word before using up all six attempts, they win the game;
4. If the player exhausts all six attempts without correctly guessing the word, the game ends in a loss;
5. When the player enters a new guess, the game must highlight the letters in the attempt accordingly:
   - Red: the letter is not in the secret word;
   - Orange: the letter is in the secret word but is misplaced;
   - Green: the letter is in the correct place.

Layer 1 - File Handler

This game requires dictionaries so that it can extract a secret word from them. Currently, there are four dictionaries: Portuguese, Portuguese simple, English, and English simple. These dictionaries were created to allow the player to choose a language in which to play (Portuguese or English) and the difficulty level of the game (simple or normal). In the simple dictionaries, words do not contain repeated letters within the same word (e.g., "jumps" belongs to both English and simple English dictionaries, but the word "kayak" does not appear in the simple dictionary, as it has two "a"s).

These dictionaries were constructed by extracting a comprehensive list of words, filtering only those with a length of five letters, and then removing any accents while converting all letters to lowercase. The layer works by using a dictionary data structure that holds the resources for the four dictionaries. When the function 'get_dictionary() is called, it loads the corresponding dictionary file into memory, reads it line by line, and appends each word to a list. Finally, the function returns the list containing all the words from the chosen dictionary.

Layer 2 - Dictionary

This layer is responsible for managing the current dictionary while it is in memory. It can find a given word in the dictionary and can request a different dictionary from the File Handler if the player decides to change it. Additionally, this layer holds a dictionary data structure called *occurrences*, which contains all the letters of the English alphabet. When given a secret word, it iterates through the word and increments the number of occurrences for each letter in the *occurrences* dictionary.

Layer 3 - Logic Layer

This layer handles the game logic. It begins by waiting for the game to be ready, ensuring that all nodes have been instanced. Once the game is ready, a new round starts, and this layer requests a random word from the Dictionary Layer, which becomes the secret word for this round. When the player enters a new guess, the input undergoes validation, which can result in three errors: the string does not have the same length as the secret word, the guess is invalid due to characters like white spaces or accents (parsed with a regex string), or the word is not found in the dictionary. If the input doesn't raise any errors, the layer proceed to score the guess.

To score the player's guess, this layer uses an array called score = [-1, -1, -1, -1, -1], where each index corresponds to a letter's position in the input. The word is iterated twice during the scoring process. Lets look at an example for a secret word "stump" where the player inputs the guesses "jump" and "trump" in this order:

```
score = [−1, −1, −1, −1, −1]
```

Now the player inputs the word "jumps". In the first iteration, the layer checks if any letters do not belong to the secret word or are in the correct place, assigning a score of 0 or 2, respectively, to the corresponding index in the score array. Since the letter J is not present in "stump", the first index of score will be zero.

```
score = [0, −1, −1, −1, −1]
```

In the second iteration, the layer looks for misplaced letters by checking if each letter in the input occurs in the secret word but at a different index. If a misplaced letter is found, the layer assigns a score of 1 to the corresponding index in the score array and decrements the occurrence of that letter in the dictionary. This step is crucial when more than one of the same letter occurs in the secret word. The letters M, P, S and U belong to "stump", although their are misplaced.

```
score = [0, 1, 1, 1, 1]
```

Now the player knows that those letters are indeed in the secret and inputs the word "trump". The process is the same:

```
First iteration:    score = [0, 0, 2, 2, 2]
Second iteration:   score = [1, 0, 2, 2, 2]
```

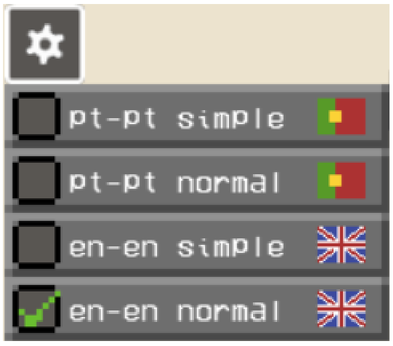

Figure 3.9: Dictionary Settings showing the four available options

With this information, the player now knows that the the only letter that they are missing is the first letter of the secret word.

When the score array is in the state [2, 2, 2, 2, 2], the player has correctly guessed the word and won the game. If the player reaches six attempts with a different score array, they have failed to guess the word.

The game's score is determined by finding the secret word in the fewest attempts possible.

Layer 4 - UI

Table 3.3 is a list of the various UI elements in the scene tree and some of their more relevant interactions and functions. Figure 3.8 is a screenshot of a round of the game where the secret word was correctly guessed in the fourth try.

By the time of implementation of this game , since aesthetics weren't the focus of the study, a decision was made to keep same overarching theme between the games. As such, the same brown hues were kept in this game and no sound or animations were added for the sake of time.

At this point, the idea of exporting the game to Android came to mind. The addition of the alphabet on the right side of the screen in 3.8 not only keeps the game true to its original version (which also has one) but also serves for the purpose of future touchscreen features.

Element	**Description**
Backgroung	The background is a 977x550px static PNG image serving as the visual backdrop of the game scene..
Pause	The same button used in the other games. On click, it pops up a window asking if the player wants to go back to the Main Menu.
Settings	On click, it displays four buttons representing language and difficulty options. Clicking a different button triggers a signal to change the dictionary and start a new round (see 3.9).
Guesses	Displays the player's current guesses and colors each letter based on the score array from the Logic Layer (green for correct place, red for not in the word, yellow for misplaced). The grid is dynamically filled during runtime, allowing flexible sizing and easy adjustments for testing or difficulty variations.
Player Input	A LineEdit node that allows the player to enter attempts with a maximum of five letters. It automatically converts input to lowercase and is hidden when the round ends.
Enter	Allows the player to submit their attempt to the Logic Layer. Can be triggered by pressing the Enter key. Provides a touch-friendly option for a potential future mobile version.
Backspace	Eliminates the last character in the Player Input, if there is one.
Hints	Displays all the letters in the English dictionary and provides two functions. Firstly, it colors the letters based on their score, similar to the Guesses section, aiding the player in making informed decisions for their next guess. Secondly, the letters can be clicked, functioning as buttons, and the pressed letter is added to the input string. This functionality is particularly useful for touchscreen devices.
Guess Arrows	Positioned on both sides of the Guesses container, these arrows point to the current attempt and add visual flair to the game.
New Word	Hidden by default, the "Next Round" button appears at the end of the round, overlaying the Player Input. When pressed, it initiates a new round with a different word.
Notice	A LineEdit node, similar to the Player Input, appears above the Guesses grid when a report signal is emitted. It informs the player about the game outcome, whether they won, lost, or made an invalid input (e.g., word not found in the dictionary).

Table 3.3: UI elements present in Wordle

3.2.4 2048

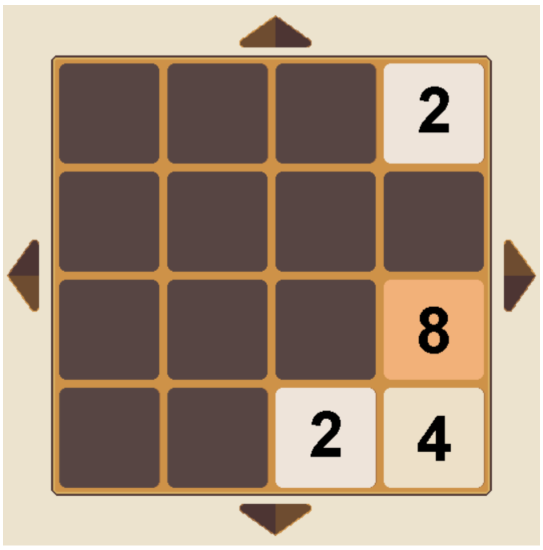

Figure 3.10: Screenshot of the 2048 game grid

2048 is a single-player puzzle game with the goal of combining tiles on a grid. The player can slide tiles in four directions (up, down, left, right), and when two tiles with the same number touch, they merge into one tile with their sum. The objective is to create a tile with the number 2048, and the game continues until this goal is achieved or no more moves are possible, resulting in a loss.

Like in Wordle, this game is divided into layers: Input Controller, Grid Logic and UI. Which will be explained in the following subsections.

Core Mechanics

Similar to the previous game, the flexibility for tweaking units and aesthetics remains a possibility in future iterations. For this specific game, the numbers featured on the tiles are powers of two, and the grid maintains a 4x4 configuration, staying faithful to the essence of the original version. This adherence to the classic format ensures a familiar and engaging gameplay experience while leaving room for potential enhancements and updates.

1. The game starts with the grid empty and generates two cells. These cells must have 90% (ninety percent) chance to be a cell with the number 2 and 10% (ten percent) to be 4;
2. The player can move the cells in four directions: Up, Down, Left or Right;
3. After the player moves and after any merges, the grid must spawn a new cell that follows the rules of in described in the first point;
4. After the player makes a move, if two cells with identical numbers come together, they will merge to create a single cell with a value equal to the sum of both cells;
5. If the player manages to get a cell with value 2048 (two thousand and forty eight) they win;
6. If the grid has no empty spaces and no merging opportunities the game is over.

Layer 1 - Input Controller

In the previous game, improvements were made in the development approach by introducing layers to handle different aspects of the game. Now, due to the need to enable touchscreen interactions, the Input Layer was created. These actions can be executed in two ways: using arrow keys on the keyboard or pressing on-screen arrow buttons. Whenever either action occurs, the Controller layer informs the Grid Logic layer of the corresponding event.

Layer 2 - Grid Logic

The flow of the 2048 game consists of the following steps, 3.11 is a flow chart that illustrates the following steps :

1. The grid begins with two cells on it, cell or tile refer to the spaces on the grid. These cells contain number 2 with the slight chance of spawning a cell with the number 4;
2. The player moves in a direction (UP, DOWN, LEFT or RIGHT);
3. The grid shifts all tiles into that direction and merges them if possible;
4. The layer checks if there is a cell 2048 and if the player won;
5. The layer checks if the player can move in any direction, if not the game is lost;
6. The grid spawns another cell (a 2 with a 10% chance of a 4);
7. Repeat from step 2 until steps 4 or 5 are true.

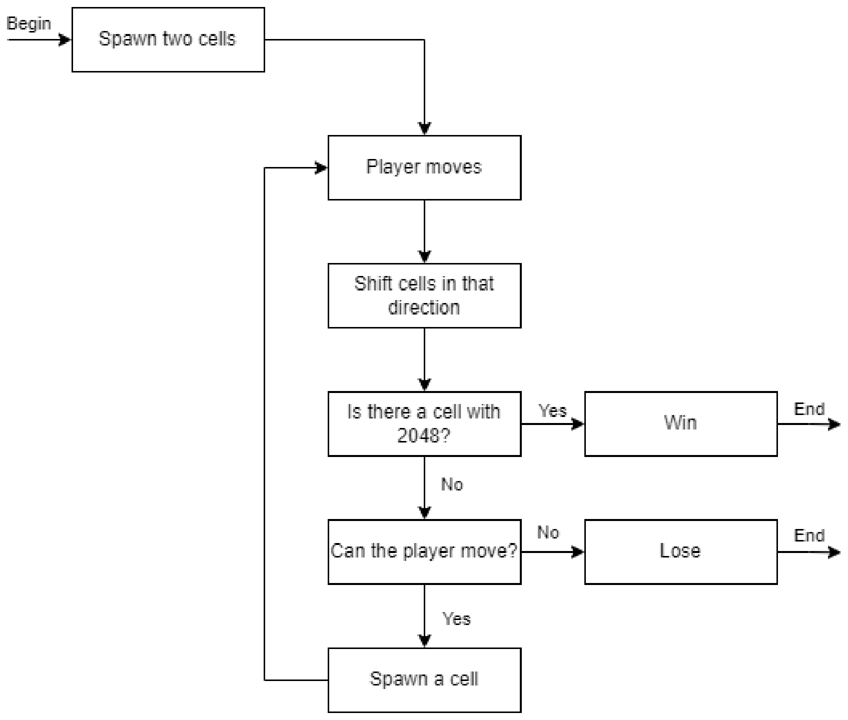

Figure 3.11: 2048 Flow Chart

In the Logic layer, at the beginning of a round, a 2D matrix (2D array) with a size of four by four is dynamically created. The grid's dimensions are not hardcoded, allowing easy adjustments through the editor. While it is possible to create any rectangular grid, it's essential to maintain the length and width equal to each other to replicate the typical square grid used in 2048 gameplay.

Once the grid is created, the spawn_cell() function comes into play, responsible for spawning two cells. This function selects a random empty cell on the grid and spawns a new cell with a value of 2 (90% chance) or 4 (10% chance). After this process, the Logic Layer emits the *switch_grid* signal, signaling to the UI Layer that the grid's state has been modified, prompting it to update the visual representation on the screen.

When the player makes a move, the Logic layer goes through three phases to handle cell shifting. First, the layer shifts all cells as much as possible in the specified direction to eliminate any empty cells between occupied ones. Next, it checks if any cells can be merged; a cell can merge with its neighbor in that direction if they have the same value. Importantly, the layer performs the merging in an order opposite to the direction the player moved[2]. Finally, since merging cells might create empty spaces between the recently merged cells and others that were previously shifted, the layer needs to shift the cells again in the same direction to fill those gaps. These three phases ensure that the game's grid is correctly updated after each move.

$$\begin{bmatrix}0&0&0&0\\2&2&0&2\\0&0&0&0\\0&0&0&0\end{bmatrix} \rightarrow \begin{bmatrix}0&0&0&0\\2&2&2&0\\0&0&0&0\\0&0&0&0\end{bmatrix} \rightarrow \begin{bmatrix}0&0&0&0\\4&0&2&0\\0&0&0&0\\0&0&0&0\end{bmatrix} \rightarrow \begin{bmatrix}0&0&0&0\\4&2&0&0\\0&0&0&0\\0&0&0&0\end{bmatrix}$$

Figure 3.12: Breakdown of the steps of shifting the grid in the LEFT direction.

After the player makes a move, the Logic layer checks if it can spawn another cell, meaning it looks for empty spaces on the grid, and does so if there are available spots. Once these changes are made, the layer emits the *switch_grid* signal, which prompts the UI to update and display the new grid state on the screen.

In the Logic layer, victory conditions are checked by iterating through the grid matrix and looking for a cell with a value of 2048. If such a cell is found, the player's score (representing the number of movements taken to achieve victory) is saved, and the signal *game_over(true, score)* is emitted to indicate a win.

To determine if the player has lost, the layer examines three conditions: the presence of at least one empty cell on the grid, the possibility of any horizontal merges, and the potential for any vertical merges. If any of these conditions is true, the player can still make moves. However, if all of them are false, the game is considered over, and the signal *game_over(false, score)* is emitted to signify a loss.

By efficiently evaluating these conditions, the game ensures that the player's movements and actions are appropriately tracked.

[2]Example: After the cells move to the left, the merging starts from right to left, ensuring that the leftmost cell has the higher number in the cases where there more than two cells with the same value. In step two of 3.12 the first 2 merges with the 2 in the middle.

Layer 3 - UI

Table 3.4 provides the essential UI elements present in 2048. Figure 3.13 is a screenshot of a 2048 game. In this image there are no empty cells nor possible merges left and the UI tells the player that the game is over.

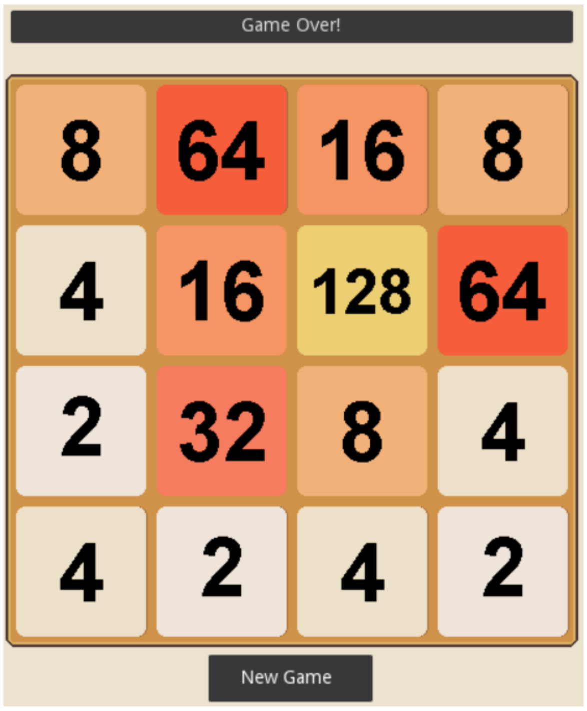

Figure 3.13: Screenshot of the mini game 2048 at the end of the game. Contains the following UI elements: Player Character; Pause Button; Camera; Wall Nodes; Background Image and Score.

It's worth mentioning that as the game is currently, when the player makes a move, the grid is redrawn, this means that the state of the grid changes instantly after a move, which can be confusing to the player. A future approach to this problem could be animating the cells by interpolating their movement from begin to end using a Tween node.

3.2.5 Saving and Loading the Game

Through out the section explaining the games in application there were mentions of saving the players' scores during their playing sessions. This section describes how the saving and loading games scores is done.

In Godot, the Autoload feature to creates a singleton, which serves as a centralized data storage solution for our project. Godot has limitations when it comes to storing information needed across multiple scenes, and using singletons is a convenient way to address this issue.

To achieve this, in Project → AutoLoad singletons can be created. This allows the storage of crucial game data, such as the player's score, in a file that acts like a singleton. This is especially useful as Godot does not support global variables by design.

By having this singleton, we can access and modify the player's score from any scene within the game. It

Element	**Description**
Background	A 440x440px PNG image serves as the background sprite.
Grid	The CenterContainer is a node that centers its children. It has two children: a darker brown background texture, serving as the grid's backdrop; the GridContainer, a node that arranges its children in a 4x4 grid layout, displays empty cells on the screen and updates the grid when it receives the *switch_grid* signal from the Logic Layer.
New Game	At the bottom of the grid, there is a button with the text "New Game" displayed on top of it using a Label node. The button also has a LineEdit, similar to the Notice in 3.2, which shows messages such as "Game Won" or "Game Lost" along with the player's score. The button is only visible when a game ends, and when pressed, it emits the signal *restart_game*, instructing the Logic Layer to start a new game.
Pause	The same button as in the other games. A popup window appears, giving the player the option to either return to the Main Menu or continue playing.
Movement	A Control node that contains four directional buttons, representing the possible grid movement directions for the player. Pressing any of these buttons signals the Input Controller about the chosen direction. When the game ends and the player can no longer move, the directional buttons are hidden, reappearing when a new game starts.

Table 3.4: UI elements present in 2048

provides a clean and reliable way to share and synchronize data across different parts of the project without resorting to complex workarounds like creating a "master" scene to load children scenes.

In the script named *GlobalScores.gd*, important game data is saved using a dictionary called *history*. This dictionary contains four sub-dictionaries, one for each game, and each sub-dictionary stores two arrays: timestamps and scores.

For example, in the game Floppy Bird, when the game ends, a signal is sent indicating that the game is over. GlobalScores then appends the score and timestamp of that game to the corresponding dictionary for FloppyBird.

By saving information in this manner, it is possible to leverage it for various purposes, such as collecting statistics. For instance, the length of the arrays can be examined to determine how many times the player has played the game. Similarly, analyzing the timestamp array allows for insights into how often the player plays the game or the duration of their gaming sessions.

This data storage approach allows for tracking and analyzing players' gameplay behaviors, providing valuable insights to enhance the overall gaming experience and make data-driven decisions for future improvements.

In the game-saving process, a function from the GlobalScores script is invoked to append scores and timestamps to the respective game. All this data manipulation takes place in memory. However, to preserve this information for future sessions, the data needs to be saved it to disk. When saving the supergame, the program opens a file named *savegame.save* in write mode. After converting the relevant data from the *history* dictionary to JSON format, it is stored in this file.

Upon launching the game, once the Main Menu is loaded, the function *load_game()* is called. This function attempts to open the save file in read mode. It then iterates through each line in the file, converts the JSON data back into its original format, and stores it in memory within the *history* array. This way, the game can retrieve and continue from the last saved state when it starts up again.

3.2.6 File Structure and Adding a New Game

Let's discuss the project file arrangement, starting with the most distinct one mentioned in section 3.2.5: the *savegame.save*. In Godot, there are two path notations: *"res://"* and *"user://"*. The *"res://"* path is intended for resources, while the *"user://"* path is designated for persistent data, such as save files and settings. This directory is automatically created on each user's device, even after exporting the project, and it is guaranteed to be writable. This path ensures that the game's save file is accessible and can be updated with the player's progress, regardless of the user's device.

In the project's *"res://"* directory, all other files are contained. Any folder that contains a *project.godot* file in it is considered a Godot project, and all files relative to it can be accessed through a path starting with *"res://"*. This allows easy access to the project's resources and ensures a clear organization of files within the project structure.

In this project, there are a substantial number of files, so let's begin by defining the typical organization of a project. Generally, there are three main types of files: scenes, scripts, and resources. Scenes encompass the entire scene tree, along with its children, and define their display and positioning. Scripts can be attached to scenes to define the behavior of elements within a scene. Resources encompass all other files, including textures, shaders, fonts, and audio files.

When starting a new project, conventionally, the main scene and its corresponding script are stored in the root folder as *"res://MainScene.tscn"* and *"res://MainScene.gd"* respectively. All other files can be organized into folders to improve accessibility. For instance, for a player scene and respective script, there can be a folder that holds all scripts and another for all scenes. Alternatively, there can be a single folder called "Player" and keep both the scene and script within it. In this project, we have opted for the former method.

In addition to organizing scenes and scripts, a folder called "assets" can be used to hold all other resources. For example, there can be subfolders within "assets" to store textures, audio files, or other specific resource types. This helps maintain a clean and organized structure for the project.

To ensure modularity and reduce coupling between mini-games and the Supergame, a systematic approach was developed for adding games to the project. During the application's development, it was realized that directly implementing all games within the Supergame could lead to a bloated and difficult-to-manage project. Therefore, an approach was sought to develop individual mini-games, like Floppy Bird, and seamlessly integrate them into the Supergame.

The rules established for this integration process are as follows:

1. The Main Scene of the Supergame must provide an option for the user to switch to the mini-game;
2. The mini-game must allow the user to switch back to the Main Scene of the Supergame;
3. The game can be saved using the same approach as other games.

By adhering to these rules, mini-games can be seamlessly integrated into the main Supergame project. The process is straightforward: first, each mini-game is created as a separate Godot project while following the folder convention mentioned earlier. Then, to include a mini-game in the main project, simply remove the *project.godot* file from the mini-game (as a project can only have one such file) and place the mini-game files into the main project. However, be cautious of potential file conflicts. Although there should be no issues if the directory rules were followed, there might be instances where the project can't find specific file paths, leading to crashes and rendering the game unplayable. To prevent such problems, it's crucial that while developing the mini-game, all file paths are used in relative terms. When these conditions are met, adding a mini-game becomes as simple as copying a folder into another.

To comply to the established rules, some minor adjustments are required in both the mini and super games. To transition from the Main Menu to the mini game, the game's thumbnail should be provided, and a button must be created to switch scenes when pressed (rule 1, estabilished through the RPs, see 3.2). Moving from the mini game back to the Main Menu can be more complex since each game is unique, but one straightforward approach is to utilize the generic Pause Button created in the super game (rule 2, see 3.2.1). As for saving the game, an implementation of a condition within it to handle game over or victory scenarios and calculate the score is necessary. Then, a new dictionary can be created in GlobalScores, using the game's name as the key, and added to the "history" dictionary, sending the score from the game to GlobalScores (rule 3, see 3.2.5).

4 Future Developments

The project involves a comprehensive exploration of Game Design and Recommender Systems, culminating in the development of an application that combines these areas. However, due to time limitations and the learning curve associated with Godot and its frequent updates, the project has only scratched the surface of its full potential. As a result, there are several exciting opportunities for future research and development that deserve attention.

One evident and valuable improvement for the application is the integration of new games. Initially, our goal was to include multiple casual games in the project. However, during the development process, we realized that creating a single game free from bugs and unexpected behaviors requires significant effort. As a result, we decided to focus on four video games, allowing us to concentrate on other aspects, such as research and additional features.

Expanding the collection of games would undoubtedly enrich the platform and enhance the user experience. Players would have a broader range of choices, leading to increased engagement with the application. Additionally, the RS would benefit from the greater diversity of game options, as a more extensive selection provides better support to users in finding games that match their preferences. Some game ideas we considered, which could serve as inspiration for future iterations, include Mahjong, Tetris, and Space Invaders.

As more games are incorporated into the application, the potential of the RS can be further explored. Currently, the system performs well with the limited data we have. However, as the item pool and player base expand, its effectiveness may diminish. To address this, future research could investigate various techniques, such as CF, NNs, or HRSs, and compare their performance.

By testing and comparing different approaches, we can optimize the RS to handle larger datasets and provide more accurate recommendations to users. This would lead to a more personalized and satisfying gaming experience for players, increasing overall user satisfaction with the application. As the project continues to evolve, ongoing research and improvements to the RS will play a crucial role in delivering a robust and effective gaming platform.

The incorporation of online interaction between players was a concept we had envisioned from the project's inception, but it was, unfortunately, left out due to time constraints. However, it remains a promising area for future development. Certain games within the platform may benefit from PvP or cooperative gameplay elements, which would enhance the overall gaming experience and foster a sense of community among users. One exciting addition could be the implementation of leaderboards, allowing players to compete with one another and compare their scores and achievements. This competitive aspect would likely increase user engagement with the platform and motivate players to strive for better performance in the games. To support these features and facilitate online interactions, the creation of a dedicated server would be beneficial. The server could serve as a central hub for storing and managing player data, as well as enabling real-time communication and coordination between players. Introducing online interaction and leaderboards would add a social and competitive dimension to the platform, attracting a wider audience and fostering a vibrant gaming community. This feature set has the potential to elevate the application's appeal and ensure its longevity.

User feedback is crucial for any project, and particularly for one that aims to provide a service to users. The benefits of having testers participate in evaluating the application are twofold:

Firstly, testers can help identify and pinpoint bugs, glitches, and unexpected interactions within both the Supergame and the casual games. Their feedback can play a significant role in refining the overall user experience and ensuring the application's stability and smooth functionality.

Secondly, the data collected from user testing can serve as valuable training data for the RS models. By observing how users engage with the platform, the RS can gain insights into their preferences, behavior patterns, and gaming habits. This information can be leveraged to enhance the RS's performance, making it more accurate and effective in recommending games that align with individual user interests.

Incorporating user feedback throughout the development process is essential for iterative improvements and continuous enhancement of the application. It allows developers to address any issues or concerns raised by users, leading to a more polished and user-friendly platform. Additionally, the feedback-driven approach empowers developers to adapt and evolve the application based on actual user needs and preferences, ensuring a more satisfying and engaging gaming experience for the target audience.

Finally, the release of the application to the market marks a significant milestone in the development process. While the application is currently available for PC, challenges were encountered when attempting to export it to Android. However, with the release of Godot 3.5, there is hope that exporting to mobile platforms will become more accessible and streamlined. Releasing the application to platforms like the Play Store, Steam, or other app marketplaces offers an opportunity for various studies and analyses. Beyond computer science-related aspects, the release opens up possibilities for social studies, providing insights into the players' demographics and preferences. Understanding the target audience can aid in tailoring future updates and game offerings to better suit their interests and needs. Additionally, business and economic studies become relevant when exploring the online video game marketplaces. Examining market trends, competition, and revenue models can inform strategic decisions for the application's monetization and growth strategies.

Overall, the application's release brings the potential for a range of research studies, encompassing diverse disciplines beyond computer science. These studies can provide valuable insights that help optimize the application's performance, engage users more effectively, and contribute to its success in the competitive gaming market.

5 Conclusion

We have presented a comprehensive study on Game Design and RS, showcasing our efforts in creating an application that integrates these two subjects. Our project aimed to explore the development of casual games while incorporating a personalized recommendation system to enhance user experience.

The significant expansion of online products and services has resulted in an overwhelming array of choices for users. In response to this challenge, RS have emerged as valuable tools that assist users in discovering relevant and personalized content. These systems, driven by AI and machine learning, are present in popular platforms like Netflix, Amazon, YouTube, and Steam. By analyzing user behavior and preferences, RS models can make informed predictions and recommendations.

In the exploration of RS, various techniques were encountered, including CBRSs, CFRSs, HRSs, and KBRS approaches. Each method has its limitations, and clever solutions, to overcome these challenges. Additionally, the importance of novelty and serendipity in retaining user interest by providing fresh and unexpected content was examined.

Despite the immense potential of ML-driven RS, there is a prevalent bias and mistrust among users towards these systems. Ethical concerns, particularly related to privacy, arise when personal data is utilized to make recommendations. To build trust, transparency with users is essential, fostering increased engagement and participation with the platform. RS thrive on constant interaction with users, as more data leads to improved recommendations. Therefore, the continuous feedback loop between users and RS models is critical for refining and enhancing the quality of recommendations.

Video games have become a widely embraced hobby, offering a form of entertainment that captivates people across the world. Their primary objective is to provide enjoyment, often achieved through well-defined rules and an immersive experience that transports players into imaginative realities. Designing such games is a complex task, requiring designers to empathize with the players and envision the interactions they will encounter. Central to player engagement is the UI, through which players interact with the game's elements such as buttons, menus, and sound effects. A well-designed UI can enhance the overall experience, ensuring a seamless and intuitive connection between the player and the game.

Achieving balance in a video game is crucial to its success. This involves tailoring the game's challenges to match the player's skill level, striking a delicate balance between being too mundane and too frustrating. In competitive online games, balance becomes even more intricate, as players' skills are pitted against one another.

A game's appeal is often defined by three key factors: balance, an engaging and user-friendly UI, and meaningful core mechanics. When these elements harmonize, a game can stand out in its genre and deliver an entertaining and memorable experience to players. As the world of video games continues to evolve, the pursuit of creating enjoyable and captivating games remains a driving force for game developers.

The development of this project has been a highly enlightening journey. When we first embarked on this endeavor, our ambitions were set high, which led to some challenges along the way. Learning to use Godot, navigating a new programming language, and adapting to frequent updates were significant hurdles that prolonged the development process. However, the experience also proved invaluable, as it honed our skills in Godot, and the difference between the implementation of the first and last games is remarkable.

On the research front, there were some complexities. While there is an abundance of literature on RS and Game Design, there are limited resources specifically focused on the implementation of RS in video games. Most existing studies revolved around platforms like Steam, primarily focusing on user interests. In contrast, the goal was to create a system that recommends games based on other metrics, such as skill level.

Despite facing some personal setbacks, the project managed to achieve its primary objectives, which included a comprehensive study of Game Design and RS. This endeavor enabled the development of a platform that, it is hoped, will serve as a foundational framework for future research and development.

While certain proposed goals had to be foregone, as mentioned in the "Future Directions" section, substantial progress has been made. This project has opened up new avenues for exploration and has shed light on the synergies between Game Design and RS in shaping a unique and engaging gaming platform. As the project concludes, further advancements in this dynamic field are eagerly anticipated.

Bibliography

[AD12] Ernest Adams and Joris Dormant. *Game Mechanics: Advanced Game Design*, volume 47. 2012.

[Ada10] Ernest Adams. *Fundamentals of game design*, volume 47. 2010.

[Agg16] Charu C. Aggarwal. Recommender systems: The textbook. 2016.

[Alp10] Ethem Alpaydin. *Introduction to Machine Learning*. The MIT Press, 2nd edition, 2010.

[ama] *What Is Fairness and Model Explainability for Machine Learning Predictions?*

[Bag20] R. Bagheri. Understanding singular value decomposition and its application in data science. 01 2020.

[BCS22] Matthew Barr and Alicia Copeland-Stewart. Playing Video Games During the COVID-19 Pandemic and Effects on Players' Well-Being. *Games and Culture*, 17(1):122–139, 2022.

[BG19] Daphne Bavelier and C. Shawn Green. Enhancing Attentional Control: Lessons from Action Video Games. *Neuron*, 104(1):147–163, 2019.

[Bis06] Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag, Berlin, Heidelberg, 2006.

[BS08] Brenda Brathwaite and Ian Schreiber. Challenges for game designers. 2008.

[Bur20] Talha Burki. The online anti-vaccine movement in the age of covid-19. *The Lancet Digital Health*, October 2020.

[Cas19] Ryan P. Castillo. Exploring the differential effects of social and individualistic gameplay motivations on bridging social capital for users of a massively multiplayer online game. *Computers in Human Behavior*, 91:263–270, 2019.

[CGP19] Germán Cheuque, José Guzmán, and Denis Parra. Recommender systems for online video game platforms: The case of steam. *The Web Conference 2019 - Companion of the World Wide Web Conference, WWW 2019*, 2:763–771, 2019.

[DK20] Gideon Dishon and Yasmin B. Kafai. Making more of games: Cultivating perspective-taking through game design. *Computers & Education*, 148:103810, 2020.

[Dri21] S. Dridi. Unsupervised learning - a systematic literature review. 12 2021.

[Edw22] Edward Probir Mondol. the Role of Vr Games To Minimize the Obesity of Video Gamers. *International Journal of Computations, Information and Manufacturing (IJCIM)*, 2(1):1–14, 2022.

[Fal19] Kim Falk. Practical recommender systems. Manning Publications, 2019.

[FCCP$^+$19] Alexis Fortin-Côté, Cindy Chamberland, Mark Parent, Sébastien Tremblay, Philip Jackson, Nicolas Beaudoin-Gagnon, Alexandre Campeau-Lecours, Jérémy Bergeron-Boucher, and Ludovic Lefebvre. Predicting video game players' fun from physiological and behavioural data. 2019.

[Goo21] C. Goodrow. On youtube's recommendation system. 09 2021.

[Gow11] S. R. Gowtham. Practical implementation of content-based recommendation system. August 2011.

[Har20] Alex Harbuzinski. Introduction to ux in game design. 2020.

[HK15] F. Maxwell Harper and Joseph A. Konstan. The movielens datasets: History and context. *ACM Trans. Interact. Intell. Syst.*, 5(4), dec 2015.

[HOM19] Yemaya J. Halbrook, Aisling T. O'Donnell, and Rachel M. Msetfi. When and How Video Games Can Be Good: A Review of the Positive Effects of Video Games on Well-Being. *Perspectives on Psychological Science*, 14(6):1096–1104, 2019.

[How23] Josh Howarth. How many gamers are there? (new 2023 statistics), January 2023. [Online; posted 18-January-2023].

[hp21] Specifications of personal computers over time, September 2021. Online; posted 28-September-2021.

[KJS20] Chelsea L. Kracht, Elizabeth D. Joseph, and Amanda E. Staiano. Video Games, Obesity, and Children. *Current obesity reports*, 9(1):1–14, 2020.

[KPAM20] Shristi Shakya Khanal, PWC Prasad, Abeer Alsadoon, and Angelika Maag. A systematic review: machine learning based recommendation systems for e-learning. *Education and Information Technologies*, 25:2635–2664, 2020.

[Liq] LiquidWeb.com. What the video game industry looks like in 2022. Online.

[LMc21] Juan Linietsky, Ariel Manzur, and Godot community. Godot documentation, 2021. Online.

[LW12] Qiong Liu and Ying Wu. Supervised learning. 01 2012.

[MAI22] MAIEI. Discover weekly: How the music platform spotify collects and uses your data. 05 2022.

[MBPJ23a] Thomas M. Moerland, Joost Broekens, Aske Plaat, and Catholijn M. Jonker. Model-based reinforcement learning: A survey. *Foundations and Trends® in Machine Learning*, 16(1):1–118, 2023.

[MBPJ23b] Thomas M. Moerland, Joost Broekens, Aske Plaat, and Catholijn M. Jonker. Model-based reinforcement learning: A survey. *Foundations and Trends® in Machine Learning*, 16(1):1–118, 2023.

[MKF$^+$21] Takato Motoo, Jiei Kawasaki, Takuya Fujihashi, Shunsuke Saruwatari, and Takashi Watanabe. Client-side network delay compensation for online shooting games. *IEEE Access*, 9:125678–125690, 2021.

[MTF20] Silvia Milano, Mariarosaria Taddeo, and Luciano Floridi. Recommender systems and their ethical challenges. *AI and Society*, 35(4):957–967, 2020.

[NVNG20] Senthilselvan Natarajan, Subramaniyaswamy Vairavasundaram, Sivaramakrishnan Natarajan, and Amir H. Gandomi. Resolving data sparsity and cold start problem in collaborative filtering recommender system using Linked Open Data. *Expert Systems with Applications*, 149, 2020.

[Par18] Dimitris Paraschakis. *Algorithmic and Ethical Aspects of Recommender Systems in e-Commerce*. PhD thesis, 03 2018.

[PPG21] Cristiano Politowski, Fabio Petrillo, and Yann Gael Gueheneuc. A Survey of Video Game Testing. *Proceedings - 2021 IEEE/ACM International Conference on Automation of Software Test, AST 2021*, (1):90–99, 2021.

[PVG$^+$11] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[SEAF11] Kamal Souali, Abdellatif El Afia, and Rdouan Faizi. An automatic ethical-based recommender system for e-commerce. *International Conference on Multimedia Computing and Systems -Proceedings*, pages 1–4, 04 2011.

[Sta22] Amazon Staff. How amazon works to make machine-learning tools fairer for everyone. 08 2022.

[TALC$^+$19] Jhonny Paul Taborda, Jeferson Arango-López, Cesar A. Collazos, Francisco Luis Gutiérrez Vela, and Fernando Moreira. Effectiveness and fun metrics in a pervasive game experience: A systematic literature review. *Advances in Intelligent Systems and Computing*, 932:184–194, 2019.

[TMAV21] Faryad Tahmasebi, Majid Meghdadi, Sajad Ahmadian, and Khashayar Valiallahi. A hybrid recommendation system based on profile expansion technique to alleviate cold start problem. *Multimedia Tools and Applications*, 80(2):2339–2354, 2021.

[WF20] Bogdan Walek and Vladimir Fojtik. A hybrid recommender system for recommending relevant movies using an expert system. *Expert Systems with Applications*, 158:113452, 2020.

[WT10] Pinata Winoto and Tiffany Y. Tang. The role of user mood in movie recommendations. *Expert Systems with Applications*, 37(8):6086–6092, 2010.

[YH19] Hsin Chang Yang and Zi Rui Huang. Mining personality traits from social messages for game recommender systems. *Knowledge-Based Systems*, 165:157–168, 2019.

[YLW$^+$22] Liangwei Yang, Zhiwei Liu, Yu Wang, Chen Wang, Ziwei Fan, and Philip S. Yu. Large-scale Personalized Video Game Recommendation via Social-aware Contextualized Graph Neural Network. *WWW 2022 - Proceedings of the ACM Web Conference 2022*, pages 3376–3386, 2022.

[YSKX22] Xiangli Yang, Zixing Song, Irwin King, and Zenglin Xu. A survey on deep semi-supervised learning. *IEEE Transactions on Knowledge and Data Engineering*, pages 1–20, 2022.

[ZLJ21] Qian Zhang, Jie Lu, and Yaochu Jin. Artificial intelligence in recommender systems. *Complex and Intelligent Systems*, 7(1):439–457, 2021.