POLITECNICO DI TORINO

Master's Degree in Computer Engineering - Artificial Intelligence and Data Analytics

Master's Degree Thesis

Designing the AI for The Number Farm: a Gaming Application for treating Dyscalculia

Supervisor

Prof. Andrea SANNA

Candidate

Francesco BLANGIARDI

Co-Supervisor

Prof. Vito DE FEO

Academic Year 2022-2023

Summary

Dyscalculia is one of the most common developmental disorders, and it leads to difficulties in learning mathematics, arithmetic operations and in representing and manipulating quantities. As all these areas are used constantly in everyday life, identifying and treating such condition at a young age can have a great impact in the school experience of those affected by it. A possible way to do so is through the use of suitably designed gaming applications, which were shown to be effective in helping children with mathematical difficulties on several publications. This thesis will present a new application of such kind, called “The Number Farm”, that implements a numerical comparison task aimed at training the perception of numerosity in children affected by Dyscalculia. As a novelty with respect to previous works however, the proposed project makes use of a study conducted in 2018, named "Learning to Focus on Number", which proposes a new model for how children perceive non-symbolic quantities. In particular the paper states that, whenever asked to indicate the more numerous out of two sets of elements, individuals are influenced not only by the ratio of the cardinalities involved, but also by non-numerical properties of the displayed information, such as the size or placement of the items composing the sets. Moreover, the experiments analyzed within the paper also suggest that the degree of these influences can vary between children affected by Dyscalculia and those who are not. To implement these discoveries into the design of The Number Farm, an AI-based system was developed, with the goal of understanding the player’s ability in coherence with the paper’s theory while also offering a balanced gaming experience in terms of difficulty. In this thesis, the design of this Artificial Intelligence will be thoroughly discussed, and with the help of a suitable testing environment, its capabilities will be shown to be up to the task: these results enforce the belief that The Number Farm can truly be helpful in the detection, and possibly treatment, of Dyscalculia.

Acknowledgements

Everyone who knows me is well aware that I am a man of few words, and I'm especially not too good at making speeches or thankings. Normally, if we were celebrating in front of a meal, I would simply propose a toast and get away with a few hasty words to express my gratitude, but since this thesis is the final step of my degree I should probably take the opportunity to properly thank everyone who had a part in this work and in my life over the past 5 years.

As they are a fundamental part of this project, I should probably start from the people that made this thesis possible: prof. Sanna, prof. De Feo, Katerina, Gaia, Bruno, Wale, Kudi, Jayalakshmi. Thanks to all of you for giving me this opportunity and for working with me during these months.

Thanks to my parents, Vincenzo and Carmelina. Thank you for everything, for putting up with my rants, problems and with the trivial matters I wouldn't dare to discuss with anybody else.

Thanks to my brother Giovanni. I decided to come study at Politecnico di Torino following in your steps, and you're also the person I spent the most time with. This path would not have been possible without you.

A huge thanks to my aunt Grazia. You've been basically a third parent to me, and it's entirely thanks to you that I was able to face this path with confidence and serenity.

I'm also very grateful to my aunt Maria. Thank you for always offering me your help, for showing me your support, and for spoiling me whenever you have the chance.

I want to also thank the rest of my relatives, for believing in me and encouraging me to keep pursuing this goal.

A big "thank you" all the colleagues that worked with me during my bachelor and the master's degree. Thanks to you, dealing with exams and projects was not as hard as it should have, I wish you all the best.

Thanks to all my friends who, like me, have been to Turin during these last 5 years.Your company has always made me feel at home, no matter how far it was.

Of course, a huge thanks goes also to all my friends from Ramacca. Sometimes it's difficult for us to meet all-together, but knowing that you are there for me whenever I come back home fills me with sincere joy.

It goes without saying that I'm also grateful to the "Dotari" people. Thanks to you I've never been afraid of failing in university, as I've always had my "backup plan" of becoming a professional player together with you.

Last but not least, I also wanted to thank everyone who lent me a "paw" during my stay in Turin. You guys surely know how to be annoying, but I would be lying if I said I didn't appreciate your company.

I dedicate this work to all of you. Thank you for accompanying me in this journey and for making it easier for me to achieve this goal.

Sincerely, Francesco

Table of Contents

List of Tables

List of Figures

Acronyms

ISA	item surface area. The area occupied by a single element in a numerical discrimination task
TSA	total surface area. The total area occupied by all the element in a numerical discrimination task
ND	Numerical Dimension. In a numerical discrimination task, It is a property of a given trial and it is equal to the logarithmic ratio of the numerosities of two proposed sets
NND	Non-Numerical Dimension. Similarly to ND, this is equal to the log ratio of the Non-Numerical Variable of the two sets
PD	Perceived Difficulty, explained in subsection 5.2.
EP	Error Probability. See subsection 5.2
SVM	Support Vector Machines, a well known learning model.
AI	Artificial Intelligence
GA	Genetic Algorithms.

Chapter 1

Introduction

Learning disorders are a persistent condition that hinder one of the most human defining features, which is the ability to learn and improve not only through stimuli and experience, but also through formal education.

The process of absorbing knowledge from oral or textual teachings is one of the key factors of humanity's progress, and with the passage of time it is becoming more and more a mean to establish oneself. An obstacle in this process can greatly affect the way an individual experiences growth as a person, as it is, rightfully, a daily activity starting from a very early age. School is in fact the most important place to understand how to behave in the world and with other people; experiencing difficulty in such an ambient, other than hampering education, can both have a negative effect regarding inter-personal relationships as well as leading to an underestimation of one's capabilities, especially when such difficulty is caused by an undetected neurological issue.

One of these conditions is Dyscalculia, which specifically affects the ability to acquire basic arithmetic skills [1] and results in difficulty in understanding numbers, learning how to manipulate them, performing mathematical calculations, and remembering concepts related to mathematics. As mathematical operations are a part of everyday life, the effects of such disorder should not be underestimated, since it is also shown to persist in adulthood [2], and especially because dyscalculia is also often comorbid to other learning and developmental disorder such as dyslexia [3] and ADHD [4].

Identifying these disorders is of fundamental importance, and doing so at an early age can both improve the child's experience at school while encouraging possible treatments. In the case of dyscalculia, a widespread theory is that it *"is at heart a deficiency of basic Number Sense"* [5], which is the ability to understand and manipulate quantities. Based on this observation, it is believed that sharpening this skill can greatly help dyscalculic children overcoming the symptoms; a suitable tool for achieving this is represented by gaming applications aimed at children, as they provide an experience that can be both educational as well as entertaining. An example of such gaming apps can be found in The Number Race [6], which was observed to be successful in increasing the number sense of several children with mathematical difficulties. This study was in fact the first of many regarding the matter of Dyscalculia and it was able to inspire the development of many more gaming applications, despite its short duration and the limited knowledge about the disorder available at that time.

Following its result, it became clear that a continuous use of a gaming app at an early age could be an interesting case of study for the treatment of dyscalculia. The idea that time and practice could help with the internal representation of numerosity and with sharpening the number sense has been established for quite some time and has been explored extensively [7]. However, in more recent times, an alternative hypothesis on how children improve with time and education was proposed in the paper *Learning To Focus On Number* [8] with the name of **"sharpening and filtering hypothesis"**, according to which the children *"improve to focus on the relevant dimension of number and to avoid interference from irrelevant but often co-varying quantitative dimensions"*. In the paper, this hypothesis was validated through the analysis of data belonging to a number comparison task, in which several people of different ethnic and age groups were asked to discriminate two set of elements with different irrelevant properties (i.e. size of each element, distance between them etc.) by selecting the more numerous one. To create a model of the subject when proposed with such kind of question, the paper also proposes also defines parameters, named α and σ within this thesis' work, that are indicative of the effect of the irrelevant non numerical properties and of the noise in the internal perception of numerosity in the comparison process.

Given that the aforementioned concepts were still not explored in the context of gaming applications, the project described in this thesis is a possible first implementation of its principles. **The Number Farm** is in fact a game targeted at kids that attend the last years of kindergarten or the first years of primary school, whose main novelty is the contribution in its design of the theory explained in Learning to Focus on Number. The aim is to provide a way to detect some of the symptoms of dyscalculia as well as a mean to train dyscalculic children in order to reduce them, all while providing an engaging experience tailored around its target audience. Moreover, as the game is meant to be played over an extended period of time, the application will give the opportunity to better understand how individuals improve over time in their perception of numerosity, in a way that is compatible with the parameters defined by Learning to Focus on Number. Just as in the paper, the game therefore consists in a number comparison task, which in this case is translated in a setting that can be appealing to its target. In short, the player will be presented with two fences, each of which contains a certain amount of animals, and will be asked to tell which of the two contains more animals. As time passes, the player will be proposed with several different trials of this kind, and their responses will be collected. Similarly to many other applications before it, The Number Farm will also make use of techniques belonging to **Artificial Intelligence** and **Machine Learning**, with the purpose of evaluating the player as well as adapting the difficulty of the game to their level.

As this project has been in development for several years, this thesis will focus mainly on the candidate's contribution, which is focused on (but not limited to) the development of an AI that follows the theory of Learning to Focus on Number in order to achieve the aforementioned goals. To aid this process, a simulation environment was also implemented, consisting of a component capable of emulating the behavior of a child according to the sharpening and filtering hypothesis.

In order to discuss each one of the project's aspects, this thesis is structured as follows:

- **Dyscalculia**. The problem of dyscalculia is explained in detail in chapter 2. The chapter presents a discussion of its symptoms and its impact on the mathematical skills developed by individuals, and also presents some ways this disorder can be diagnosed and treated. Moreover, an in depth description of Learning to Focus on Number will be carried out, focusing on the relevant aspects for this project.

- **The State of the Art**. The state of the art of previous training apps meant to treat dyscalculia is discussed in chapter 3, where a few examples of applications will be described.

- **General Aspects of the Project**. Chapter 4 is dedicated to the general description of the project. To better explain the individual contribution of the candidate, the project's structure will be analyzed, and the starting point of the application will also be discussed.

- **The AI**. Chapter 5 is where the new AI developed by the candidate will be presented. Its functionalities will be explained, and particular focus will be given on how it uses the knowledge gained from Learning to Focus on Number in order to carry out its tasks.

- **The Simulation Environment**. The previously mentioned simulation environment will be discussed in section 6. All its functionalities will be showcased, and an assessment of the AI's capabilities will also be made through its use.

- **Conclusions and Future Works**. Lastly, chapter 7 will be reserved for the conclusions and to discuss what possible other improvements can be brought to the project by future members.

Chapter 2

Dyscalculia

As explained in the introduction, before going into the details regarding the project it's worth to elaborate on the core problem that it is trying to solve. This section is dedicated to giving dyscalculia a proper introduction, ranging from its symptoms to its treatments. Also, a thorough explanation of Learning to Focus on Number will be made, as it is the paper that inspired this project the most.

2.1 Main Characteristics

Developmental Dyscalculia, often abbreviated to **DD**, is categorized as a Specific Learning Disorder (SLD) that results in difficulties learning number-related concepts or performing arithmetic calculations. As all SLDs, it is a developmental disorder that shows its first symptoms at an early age and that generally persists throughout the individual's whole life. In the following paragraphs, a possible theory for its cause is described, as it's particularly relevant for this thesis' work, which is followed by a brief explanation of its symptoms and treatments.

2.1.1 The root in the Number Sense

Despite its effect on the mathematical abilities of the individual, the cause for DD lies on a more fundamental concept than arithmetical skills, which is defined in several papers as the **"Number Sense"**. Such concept is described as a *"short-hand for our ability to quickly understand, approximate, and manipulate numerical quantities"* [9], and is a core skill that is normally functioning automatically within an individual. According to its theorizer, in fact, individuals have the ability to understand numerosities in a way that is detached from the actual symbolic representations of number, and the mechanism that allows for their internal representation is called **"Approximate Number System" (ANS)**;

starting from this definition, a possible cause for DD has been therefore identified in deficits in the acuity of this system.

This theory assumes the name of “Magnitude Representation Hypothesis”, and its implications have been particularly explored within tasks related to numerosity comparisons: the acuity of the ANS is in fact observable from the ability of the individual in comparing non-symbolic numerosities, with the ratio of the two compared quantities being a measure of the difficulty of the task. Slowness or inaccuracy in these kind of operations when the two quantities are close in number can be viewed as a deficit in the acuity of the ANS. Moreover, since such acuity is often observed to be co-related to math achievements, this suggests that the presence of a mathematical disorder like Dyscalculia is **deeply liked with the individual’s skills in numerical comparisons**. This piece of information will be very useful in the next chapters, as The Number Farm proposes a mini-game representing a non-symbolic numerical comparison task; as for the effects of dyscalculia, a brief overview will be given in the subsequent paragraphs.

2.1.2 The Symptoms

The symptoms brought by dyscalculia are several, and are not limited to just difficulties in performing mathematical operations. In the following, the most important ones mentioned in previous publication are summarized:

- Difficulty in **subitizing** which is the ability to know, without explicitly counting, how many objects there are in a small group [10]
- Difficulty in comparing numbers correctly, as described in previous paragraphs.
- Difficulty in relating mathematical concepts to normal life situations.
- Difficulty in handling different representations of number (ex. symbolic, number words etc.)
- Inconsistent results and exertion when performing arithmetic operations.
- Difficulty in estimating and approximating numbers.
- Problems in telling time and reading analog clocks.
- Difficulty in remembering mathematical concepts.
- Lose of focus during mentally intensive tasks.

All these symptoms greatly influence working areas related to mathematics, while also presenting problems in every day life. As both of these things can also have an impact in the individual's academic and career path in adulthood, it is vital to promptly detect these symptoms so that action can be taken starting at an early age.

2.1.3 Diagnosis and Treatment

Dyscalculia is widely considered to be at its fundamental level a deficit in the development of arithmetical and mathematical skills [11], but as of now, there is no widely accepted criteria for diagnosing it. Moreover, the world of mathematics is rather complex, as even at early stage of education it involves several different concepts like arithmetic and geometry: due to this variety in areas affected by dyscalculia, researches identified several ways to diagnose the disorder.

Generally, the diagnosis procedures involve the analysis of domain-specific tests, that are meant to understand the child's capability in executing specific functions or to hold numerical information in mind. Such tests are generally analyzed jointly with evaluations coming from the school teachers, as they allow for a more comprehensive diagnosis. As an alternative way to diagnose dyscalculia, an analysis of the child's brain activation patterns can also be performed. In fact, dyscalculic children have been demonstrated to show differences in the activation of the prefrontal cortex[12]; as these procedures generally have a high cost and require time to be performed, it is however unlikely that this method will be used as a general criteria for diagnosis, despite being very effective.

Moving on to the treatments, the focus of research has been mainly devolved into training basic number concept in the children. At first, methodologies involving interactions between specifically trained teachers and small groups of individuals were shown to be successful among children with generalized math learning difficulties. Due to practical reasons however, the research quickly moved to computer-based methodologies, with the goal of achieving similar results while also providing a mean to practice more than it would be possible with dedicated teachers.

Under this idea, many software-based programs were developed with the purpose of training children with numerical disabilities, by implementing games where the player would have to practice with operations related to number manipulations. An example of these games can be found in The Number Race, which will be discussed more in detail in chapter 3: the general approach of these games is to develop mini-games aimed at training specific areas affected by dyscalculia; moreover, the game's difficulty is generally controlled by adaptive algorithms, which are aimed at understanding the ability of the player. As these training programs have proven to be effective among children with general difficulties in mathematical development, The Number Farm has been designed to follow in their step; however, a difference with previous works lies in its theoretical principles: the project proposed in this thesis is in fact based on the theory of the paper "Learning to Focus on Number", of which the main points are explained in the next section.

2.2 Learning To Focus On Number

Now that the general aspects of dyscalculia have been introduced, the last section of this chapter will be focused around Learning to Focus on Number, a paper published in 2018 that strongly inspired the development of this project. As will be shown, the paper borrows from the theory of the “Number Sense” and reiterates on the importance of the performances in non-symbolic numerical comparisons in the context of dyscalculia.

2.2.1 Premises

Learning to Focus on Number is a study mainly focused on how age and education change the way individuals perceive numerosity, whether dyscalculic or not, and how such changes happen with maturation. As an established methodology, the acuity of such perception can be assessed based on the subjects’ decisions when proposed with a numerical comparison or discrimination tasks, which is generally implemented in the form of a non-symbolic numerical comparison. Several studies prior to the paper have demonstrated that, even in the absence of training or education, the perception of number is present in infants similarly to other quantitative dimensions of the environment[13]; however, when it comes to numerical discrimination tasks, infants can discern well only sets that differ in numerosity by 300%, while accuracy improves drastically with adults, who usually are able to differentiate numerical changes within 15% and 20%.

Given that education is also considered to play a role [14] [15], the most accepted explanation for this behavioral improvement is the so called *“sharpening hypothesis”*, according to which maturation and education make the internal representation and the perception of numerosity more accurate. In the paper however, an alternative possibility is explored, suggesting instead that the difference in performance is due to a gap in the ability of discarding task-irrelevant features while focusing only on the numerical quantity of the two sets. This hypothesis, referred to as *“filtering hypothesis”*, is validated by previous publications, showing that individuals are less accurate in number comparison tasks when the “size of the items or the inter-item distance was incongruent with number”[16].

The paper introduces the principles of these two hypotheses and proposes a mathematical model of their implications. Moreover, their plausibility is discussed from a series of experiments involving groups of individuals of different ages and education levels, where each participant was asked to solve a numerical comparison task implemented as a series of “trials” as described in figure 2.1.

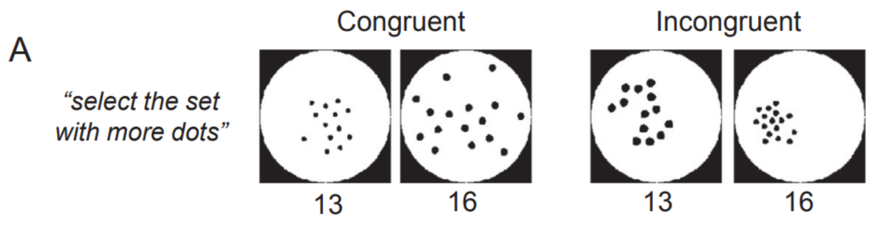


Figure 2.1: Two examples of trial presented within the experiments. The participants were shown a picture consisting of two arrays of dots with different numerical and non numerical features (see subsection 2.2.2) and were asked to indicate, without counting, the area that contained more of them. In the picture, the trial on the left is defined as "Congruent", which means that its more numerous set also presents predominant non numerical features, whereas the trial on the right presents the opposite situation, defined as "Incogruent". No information regarding the congruity of the trials was disclosed to the participants during the experiments.

2.2.2 The Filtering and Sharpening hypotheses

According to Weber's law [17], the difficulty of non- symbolic numerical comparison tasks such as the one described in the paper is related to the logarithmic ratio of the numerosities of the two sets: this quantity will be referred to as the **Numerical Dimension**, **ND** in short, for the given trial. The introduction of the filtering hypothesis however assumes several other quantities that contribute to the overall difficulty of the task. These are referred to as **non-numerical dimensions**, which are related to each single set of elements shown within the trials, and were designed starting from a set of dimensions used in previous literature [18], namely:

- **Item Surface Area**. Abbreviated to **ISA**, it is a measure of the area occupied by a single dot.
- **Field Area**. Abbreviated to **FA**, it is the area of the white disc where dots are placed into
- **Total Surface Area**. Abbreviated to **TSA** it is the total area occupied by the dots within the disc.
- **Sparsity**. It is a measure of the distance between each element, computed as the ratio between FA and the number of dots within the disc, abbreviated as **Spar**

By applying dimensionality reduction (see appendix B for details), it is possible to compute a single dimension that summarizes the orthogonal component with respect to number of all of these non-numerical dimensions into a single variable. In the same way as the purely numerical value, the authors defined as **Non-Numerical Dimension**, abbreviated to **NND**, the log ratio of the two non-numerical variables extracted from the two discs of the same trial. As a convention, both ND and NND for a given trial are computed by using the related quantity of the element shown in the right as the numerator of the logarithmic ratio.

The sign of ND and NND formalizes the concept of congruity introduced in figure 2.1: congruent trials are in fact trials characterized by an ND and NND with the same sign, and a measure of congruity of a trial can be also defined as the numerical distance of the two values. Moreover it is now possible to describe each trial proposed during the task inside a two dimensional space with ND and NND as its axes, as shown in figure 2.2.

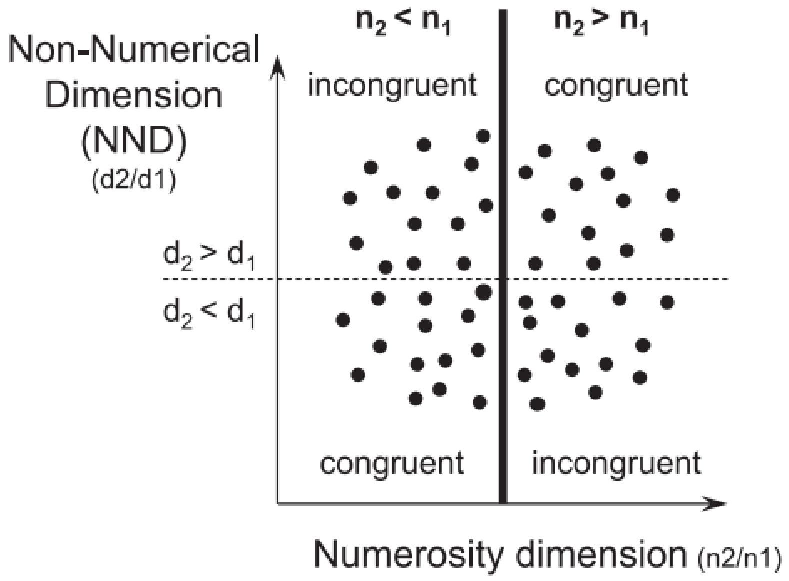


Figure 2.2: The ND-NND space for trials. In the picture, each dot represents a trial, while the “2” and “1” suffixes are used to describe the numerosity and the summarizing non numerical variable for the right and the left area of the trial respectively

With this new representation, it is now easier to delve into the details of the sharpening and the filtering hypotheses:

- The sharpening hypothesis, as stated before, implies that maturation and education increase the accuracy with which the individual perceives and internalizes numerical quantities. In the ND-NND space, the participant is considered to be able to discriminate trials based solely on their ND variable, and can therefore be considered as a linear classifier with a decision boundary coinciding with the y axis; however, since human perception is never perfectly accurate, each individual is also characterized by an internal noise that acts on their representation of the trial in terms of its ND and NND values. This hypothesis therefore states that this internal noise is the only factor that determines the quality of the numerical decisions made by the individuals; as an explanation to the improvement of said quality, the hypothesis instead suggests that generally such noise is very high at an early age and tends to decrease with the age and education.

- The filtering hypothesis suggests instead that the individual's perception of numerosity is also affected by the other non numerical features of the two sets. In this case, the participant's response can be modelled as a linear decision boundary passing through the origin of the ND-NND space that is characterized by an angle with respect to the y axis: if the angle is high, the participant's decision is likely to be influenced by the NND variable of the proposed trial; as the angle decreases however, the decision boundary approaches the y axis, which represents the optimal decision boundary for the given task as it completely filters out non numerical information. With this premise, the better performances of older and more educated groups reported in other papers is not explained by a decrease of the internal noise for the numerical representation, but rather by a decrement of this angle.

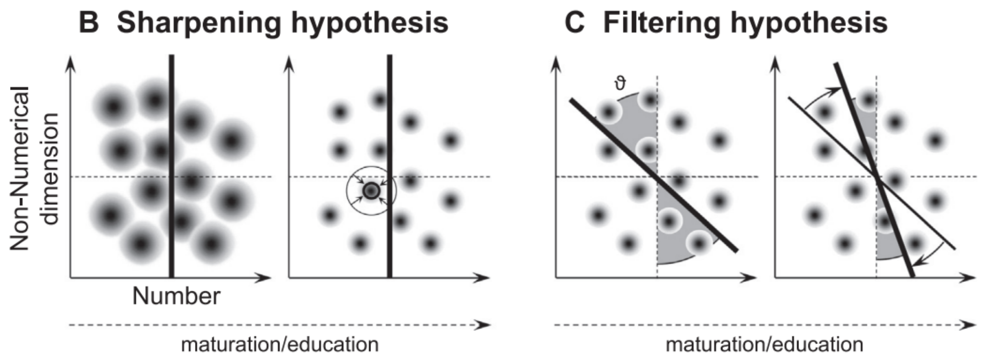


Figure 2.3: Visual representation of the effect of development according to the filtering and the sharpening hypotheses in a numerical representation task

With these premises, the authors of the paper made some predictions regarding what the analysis of the collected data would show according to each hypothesis. If only the sharpening hypothesis is valid, the groups of participants taken into consideration will give significantly different answers only for the trials that are close to the y axis in the ND-NND space, with an overall improvement in terms of accuracy for older or more educated groups. In the case where only the filtering hypothesis is correct, it is instead expected that these group will perform better only for incongruent trials, while for congruent trials they may even show a decrease in accuracy as they can no longer rely on non-numerical helping variables. Finally, the third possibility is that the filtering and sharpening hypotheses are not mutually exclusive: the individuals improve according to both hypotheses, and will therefore achieve better results both in incongruent and congruent trial conditions.

2.2.3 Methods

The data on numerical comparison trials was collected from several groups, consisting of subjects of different ages, education and culture: 44 Italian kindergartners (ranging from 3.6 to 6.2 years old); 29 Italian school-aged children (between 8 and 12 years old); 20 Italian educated adults (aged between 22 and 33 years); 25 Italian dyscalculic children (between 8 and 12 years old), and 38 Munducucù Indians with a wide age range (from 3 to 62 years, averaging at 24.6 years old). The experiments were performed in occasion of previous publications [14] [19], but were previously only analysed according to the Weber's law regarding solely the numerical dimension of the trials.

The stimuli space of the trials consisted in, as mentioned earlier, two arrays of dots with different numerical and non numerical features. For one of the two arrays, the number of elements could be either 16 or 32, while the second array could assume a value between 10 and 22 or 20 and 44 respectively. Moreover, the trials were generated in order to provide a balance between congruent and incongruent situations, spanning a wide range of values for each of the four non numerical features described in previous literature (FA, ISA, TSA and Spar). Moreover all four of them were expressed in terms of number of pixels and turned into a single variable orthogonal to number, as explained in appendix B.

2.2.4 Analyses and Results

At first, an analysis focused solely around accuracy was performed. From the collected data it emerged that the overall accuracy was significantly better in congruent trials with respect to incongruent ones for all age groups except for the 10 year old school kids. In any case it is worth noting that accuracy was still higher in congruent conditions and that, with respect to preschooler children, the 10 year old kids showed a slight decrease in accuracy for congruent trials and a significant increase for incongruent ones, which is precisely **what was predicted regarding the filtering hypothesis**.

On the other hand, another analysis of the participant's performances was carried out following both a logistic regression approach as well as a mutual information approach (also known as Shannon Information). In particular, in the case of logistic regression, the authors trained some models for each of the groups of participants by using the collected trial data (with suitable pre-processing) as the training dataset. After the models had been trained, the authors then extracted for each group the best fitting decision boundary in the ND-NND space of trials, which was assumed to be a line passing through the origin characterized by its angle with respect to the NND axis, as mentioned previously.

This analysis showed that such angle was close to the optimal 90° value for the adults and 10 year old children (~85°), while the kindergarteners and the Munducurcù groups presented a lower angle representing a less optimal decision boundary for the task (~54° and ~75° respectively), **which further validates the filtering hypothesis**. An interesting result would also involve the dyscalculic children, who demonstrated an angle of around 10° less than the non-dyscalculic kids of the same age.

In conclusion, the paper proposed and gave empirical evidence for the filtering hypothesis, cementing it as a relevant mechanism in non-symbolic numerical decision making and as the main object of improvement during development, although the sharpening hypothesis is still necessary to account for the trends shown in previous papers. Moreover, the analysis of the data showed that dyscalculic children tend to be affected by non numerical data more than other kids of the same age, thus detecting a new possible symptom of such condition. The concepts introduced in this paper represent valuable knowledge for the purpose of this thesis' project. However, before discussing how this paper is used by The Number Farm, it can be useful to first take a look at some other applications that were aimed at treating dyscalculia, which will be the main topic of chapter 3.

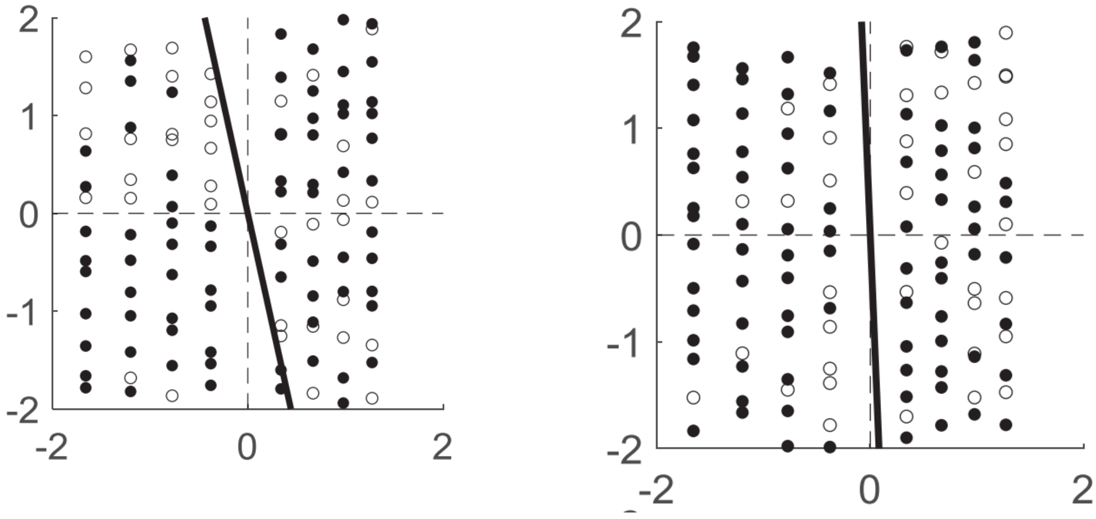


Figure 2.4: The measured decision boundary in the ND-NND space for the group of dyscalculic Italian children (left) and the group of 10 year old children (right). As shown in the figures, the group of dyscalculic children was defined by a less optimal decision boundary

Chapter 3

The State of the Art

The proposed gaming app is, of course, not the first one of its kind. Several game apps targeted at kids were in fact developed prior to it with the aim of either teaching a concept or, more specifically and similarly to this project's objective, helping them getting used to arithmetic and mathematical concepts. Gaming apps are exceptionally suitable for this as they provide an engaging environment where the kid can have fun while learning the core skill that is being taught.

Before delving into this thesis' game, it can be useful to analyze the standard of applications aimed at treating DD: in this chapter, an introduction to both **The Number Race** and **Calcularis** will be carried out, as they both share several aspects in common with the project developed by the candidate.

3.1 The Number Race

3.1.1 General Aspects

The Number Race is a gaming application developed in 2006 by Wilson et al. [20], and is one of the first example of games based on an *adaptive algorithm* with the purpose of training children with dyscalculia and mathematical deficits in general. The goals of the application were in fact to help them improve on several aspects that are affected by the disorder, all while keeping the experience as entertaining as possible by adjusting the difficulty of each stage with respect to the skill of the player. In detail, the authors had three main objectives:

- **Enhance number sense**. As discussed previously, the Number Sense has been identified as an important factor in mathematical and arithmetical skills. For this reason, the game implements a game where the children have to estimate numerosity without counting explicitly.

- **Cementing the links between representations of number**. Among the issues of dyscalculic children there is also a difficulty in learning the relations between symbolic representations of number (i.e. Arabic symbols, words etc.) with the concept of numerosity itself. Hence, the authors also implemented the game so that the player has to deal with both symbolic and non-symbolic representations of number in order to reach their goal.

- **Conceptualizing and automatizing arithmetic**. Since dyscalculic children also tend to show a developmental delay in procedures related to simple additions and subtractions, an additional mechanic was added in the game that would include these kind of operations as a way to help children automate them over time.

To meet all of these requirements, the core game mechanic was implemented as a **number comparison** task, which is encapsulated in a broader board game where the player's performances in the comparisons mini-game allow them to advance and eventually win against a CPU-controlled opponent. As an incentive to keep playing, the player is also rewarded with new playable characters the more times they win the game.

In the comparison mini-game, the player has to choose the set with more elements, as shown in figure 3.1. Sometimes the mini-game involves also performing an addition or a subtraction in order to get the correct value for the number of elements; moreover, in order to prevent the player from counting explicitly, a limited amount of time is provided to make the decisions, after which the opponent (the CPU) will choose the more numerous set for itself. Depending on how many elements the player and the CPU selected, they will advance of a proportional amount in the board game, with the goal of reaching the end of the board before the opponent.

3.1.2 The adaptive algorithm

Regarding the adapting algorithm that controls the difficulty of the mini-game, the authors identified three dimensions to regulate it:

- **The distance**, intended as a measure of how the two sets differ in terms of number.

- **The speed**, which determines the time available for the player to make the decision.

- **The conceptual complexity**, which is a composite dimension whose difficulty is increased by decreasing the ratio of non-symbolic to symbolic information available and by introducing additions and subtractions.

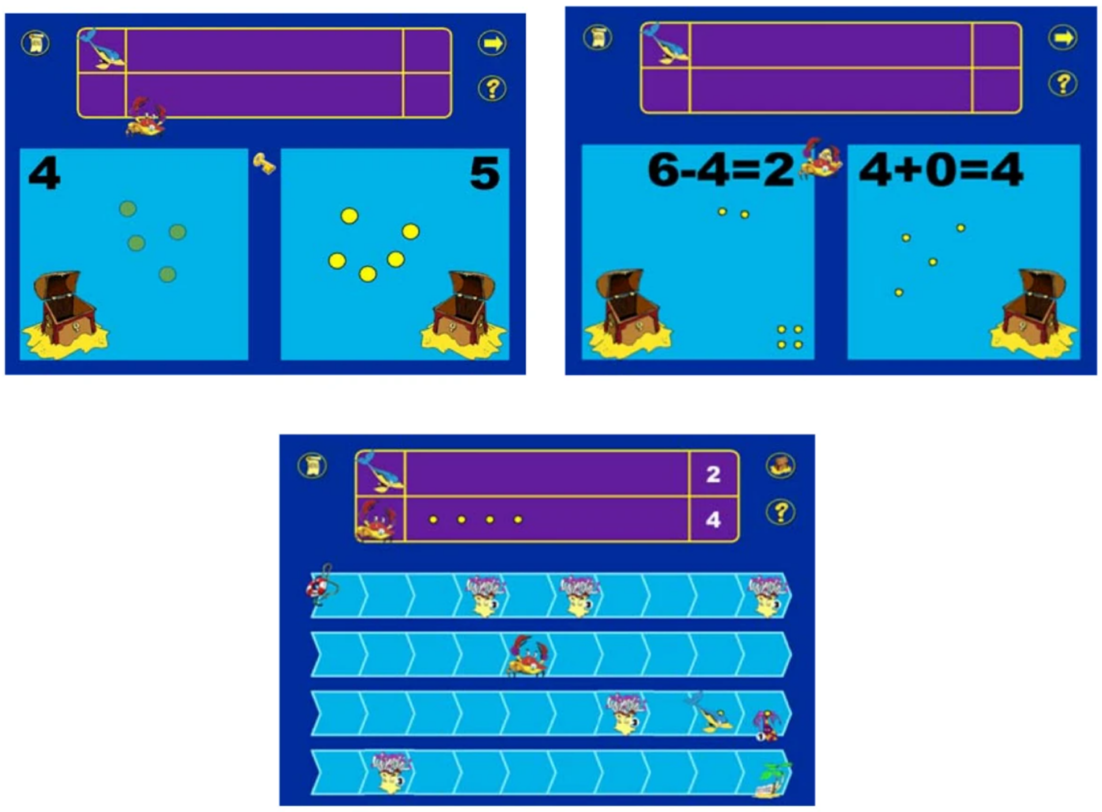


Figure 3.1: Screenshots taken taken from the game. The two top figures show the comparison mini-game (both with and without arithmetic operations), while the one at the bottom displays the board game.

The combination of these three dimensions defines a three dimensional *"learning space"* by giving each one of them a difficulty coefficient between zero and one. Each trial will be represented as a point in this space, and the adaptive algorithm was designed to understand how likely each player will provide the correct answer based on their displayed skills in each dimension. More in detail, the algorithm tries to estimate the *"knowledge space"* of the player, which is a subspace of the aforementioned learning space and defines the set of trials that have a high probability of being guessed correctly. Out of that knowledge, the algorithm will then propose trials so that the player can improve gradually on each dimension while achieving an overall satisfying accuracy.

3.1.3 Results

The algorithm was first tested through a simulation environment able to reproduce the behavior of a child with a certain knowledge space. The simulations were both performed by keeping the knowledge space fixed and by setting different levels of learning rates for each dimension, in order to assess the performances also in the case of a player who improves over time.

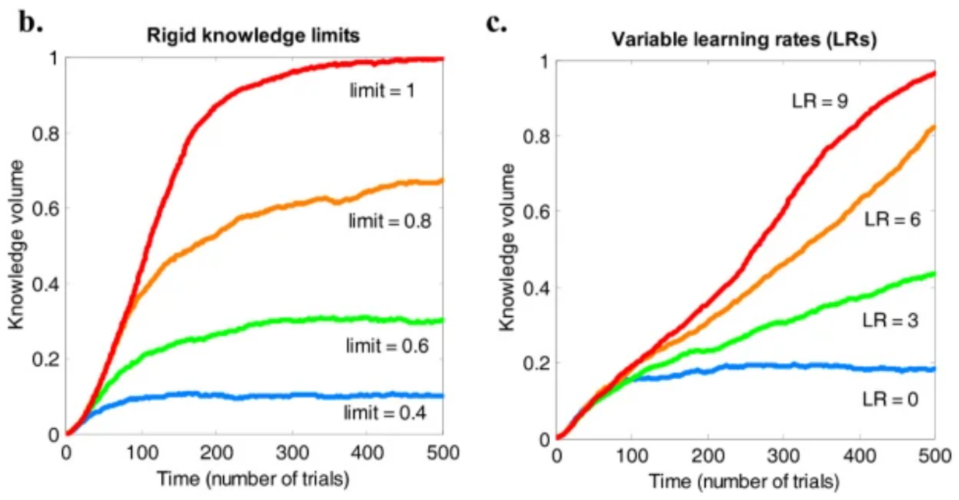


Figure 3.2: Graphs produced after the simulations. Both of the plots use the "knowledge volume" as the main metric, which stands for the volume of the knowledge space estimated by the algorithm. In panel b. each curve represents different simulations with a given limit for all three knowledge dimension of the simulator. In panel c. instead are shown simulations with varying learning rates for the simulator's knowledge space, without any limit on the actual knowledge volume

.

As shown in figure 3.2, the algorithm was able to estimate fixed knowledge spaces relatively well while also being able to adapt to several different learning rates for each knowledge dimension.

Moreover, the game was also tested on a group of 9 children with difficulties in learning mathematics. After a 5 week study, the collected data proved that the accuracy eventually converged to the desired fixed quantity (75%) for all children, and that some of them did experience an increase in their estimated knowledge volume, as shown in figure 3.3 and figure 3.4.

In addition to these results, the group of children who participated in the playthrough were also examined on their performance in several tasks such as counting, symbolic comprehension and arithmetical operations [6]. The assessments

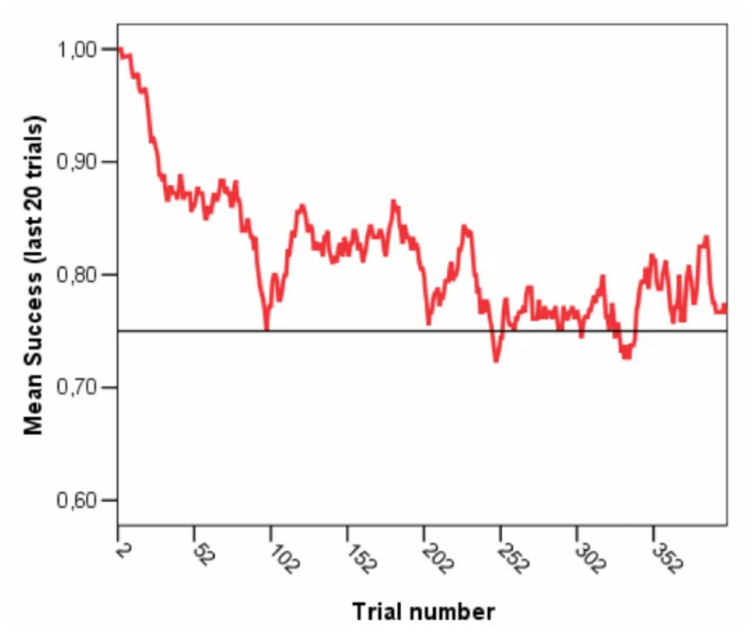


Figure 3.3: the accuracy attained by the players, measured as a running average of the last 20 trials for each child, and averaged across all nine participants

.

were made both before and after the use of the software, which leads to the most relevant results of this study: not only the adaptive algorithm was able to understand the children's skills, but the exposure of the children to the game also led to an overall improvement in other mathematical tasks that were not strictly related to the game, thus validating the effect of gaming apps in the treatment of Developmental Dyscalculia.

3.2 Calcularis

3.2.1 Main Principles

Calcularis [21] is another computer-based training program targeted at children with DD or difficulties in learning mathematics, with an heavy inspiration from another game named "Rescue Calcularis" [22]. Just like The Number Race, the game combines the training of basic numerical cognition with the training of arithmetical abilities through the use of an adaptive algorithm.

The program was designed with 3 design principles in mind:

- **Design of numerical stimuli**. In order to strengthen the links between various representations of numbers, properties such as colors and shape are

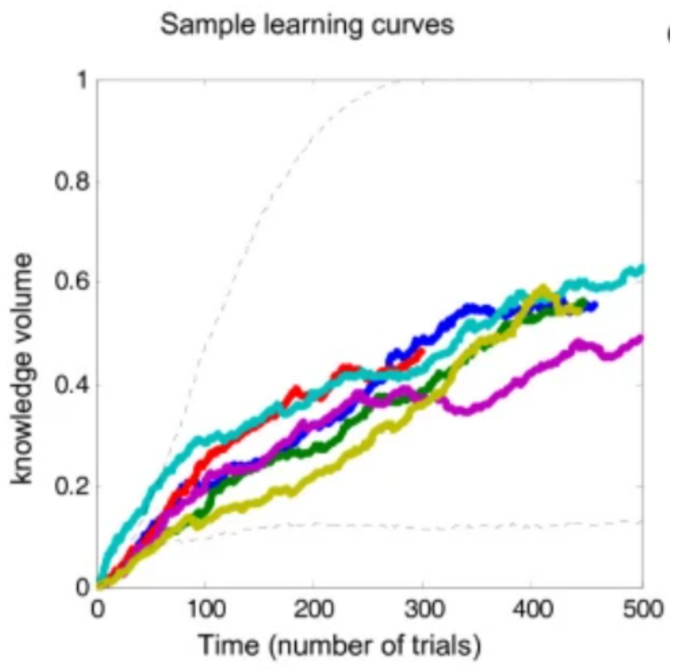


Figure 3.4: The estimated knowledge volumes over time for all the children

.

consistently used to give visual cues on the quantities displayed by the several mini-games.

- **Adaptability and scaffolding**. As every individual learns at their own pace, the program is designed in order to teach fundamental knowledge first, and to later introducing more advanced and difficult concepts, when the player has shown sufficient abilities to understand them.
- **Different types of knowledge**. The training program has the aim to help children both with acquiring conceptual knowledge as well as with automating mathematical operations.

Following these principles, the authors developed several mini-games that can be divided in three hierarchically ordered areas:

1. **Number Representations**. In this area, the games involve tasks aimed at strengthening the links between the various representations of numbers (i.e. Arabic symbols, number words etc.). The authors also established the cardinality (quantity), ordinality (position in a sequence) and relativity (difference between two numbers) as the main factors related to the difficulty of these games.

2. **Arithmetic operations**. This area focuses on training arithmetic skills, with task difficulty being determined by the magnitude of numbers involved and by the visual aids provided to the player.

3. **Word problems**. An area where the player is given verbal instruction, out of which they have to understand both the quantities involved and the operations to be performed.

It is worth noting that the second and third area require a good understanding of the topics belonging to the previous ones in order to be dealt with properly. Additionally, the games can also be categorized on their relative complexity, with some games requiring a combination of abilities to be solved, while other ones only train specific skills within their area. A typical training path would therefore consist in playing games from each area by also spanning several intervals of possible cardinalities of the numbers used.

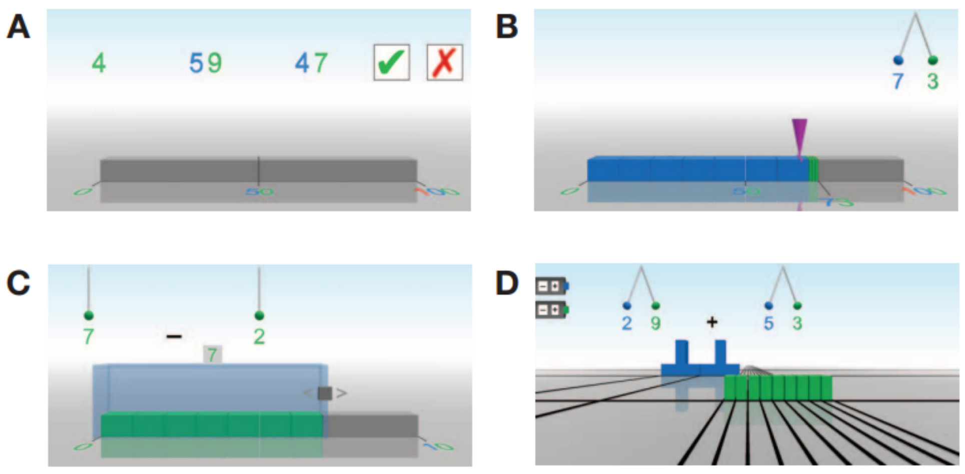


Figure 3.5: Examples of games implemented in the training program. Panel A and B show games related to the Number Representation area with cardinality in the range of 0 to 100. In panel C and D it's possible to see two other games belonging to the area of Arithmetic operations, with relatively simple cardinality in the range from 0 to 10. It's worth noting that tens are coloured in blue while units are colored in green, according to the numerical stimuli principle mentioned before.

3.2.2 How to model the player

Regarding the adaptive algorithm, it is based on a **student model** and a **controller** which were developed starting from concepts already established by previous literature [23].

The player's mathematical knowledge is modelled as a directed acyclic graph, where each node represents a skill and the links represent dependencies between each of them. Skills are therefore linked with a "precursor and successor" relationship, where successor skills are more complex, but require that their precursor ones are already achieved in order to be mastered. The controller will consider each skill as either "learnt" or "not learnt" based on the player's performances on the tasks assigned to said skill, and at each input from the player it can decide whether to keep training the current skill (to understand better if it has been learned or not), to go back to one of its precursor skills (if it's likely that they were not achieved) or to go to one of the successor skills (when the current skill has been established as learned). For more details, refer to figure 3.6.

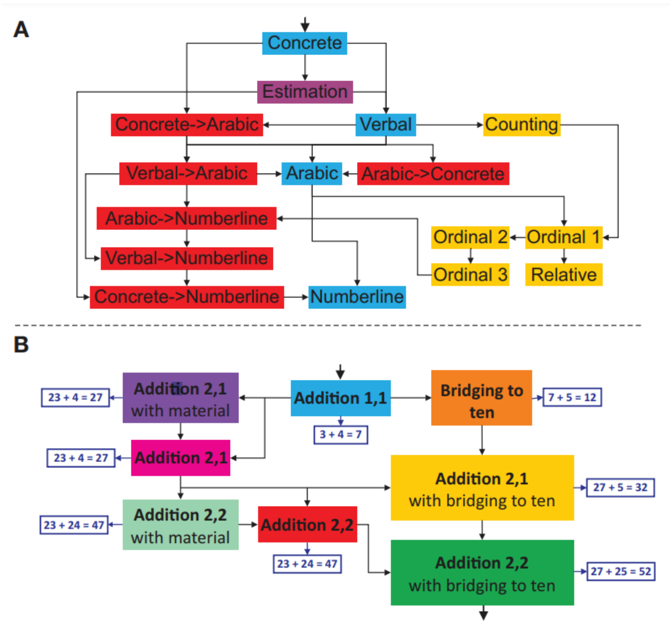


Figure 3.6: The knowledge graphs for two different areas in the cardinality interval from 0 to 100. Panel A shows the graph for the Number Representation area, where the various representations are colored in blue, the trans-coding skills in red and the ordinal ones in yellow. Panel B shows instead the sub-graph for the addition skill with examples of related tasks, where the root node represents easy operations while successors require more difficult tasks to be achieved

.

3.2.3 Results

The effects of the training program were assessed on a group of 41 children, which were divided into a training group (n = 20) completing a training of 12 weeks and a (n = 21) starting after a rest period of 6 weeks. The mathematical abilities of each group were evaluated by means of computer-based tests and HRT (Heidelberger Rechentest), which were performed at three different points in time during the training: at the start, after 6 weeks and at the end of the program. Moreover, the children were asked to play the game at least 5 times per week for at least 20 minutes per session, and were asked to fill a feedback questionnaire at the end of the training.

Based on the estimated knowledge graphs and the performance tests, the children were already able to solve more complex subtractions after a training period of 6 week, and also showed an improvement in the response time for these kind of questions. An overall increase of accuracy was also reached for number ordering tasks, and even if such trend was not replicated for number comparison ones, the authors observed that this was due to ceiling effects (i.e. children were already quite capable to begin with). Moreover, the prolongation of the program from 6 to 12 weeks was observed to be mostly beneficial, and according to the feedback questionnaire the program was able to adapt well to the skills of the players, since the difficulty was mostly described as appropriate by the children themselves.

Despite these beneficial effects however, the authors also pointed out how an appropriate comparison among the children was difficult to do due to the nature of the program: given the proposed adaptive algorithm and the wide set of skills that were subject of training, each child followed a different training path during the training period, which could lead to training some specific skills more than others; moreover, the training time for each specific skill is reduced due to their high number, which was given as an explanation for the benefit of an extended training period. Nevertheless, these observation are not meant to devalue the results of the game, but rather to enforce the fact that further research in computer-based program can provide a valuable alternative to conventional treatments for mathematical disorders.

In conclusion, just like The Number Race, Calcularis represents an important example of how a carefully designed training application should be. Now that their aim and design principles have been discussed, it is time to delve into the main topic of this thesis, which is the gaming application developed by the candidate in collaboration with the University of Essex.

Chapter 4

General Aspects of the Project

Now that both the problem of Dyscalculia and the state of the art have been described, it is now time to delve into the main subject of this thesis' work. The following sections will be dedicated to an overview of the general aspects regarding the proposed game, which is named **The Number Farm**. First of all the user experience will be treated, which is followed by the discussion of the theoretical aspects and of the project structure. Finally, to better understand the candidate's individual contribution, a section will be dedicated to describing the project state before the beginning of this thesis' work.

4.1 The User Experience

As a first step in the introduction to The Number Farm, it may be useful to discuss what is the kind of experience that the game wants to provide to its target users, which consist mainly in preschool children as well as dyscalculic children who have difficulty in performing operations related to mathematics.

Other than providing a mean to training mathematical skills, the game was designed also in order to be an entertaining activity for the player, as to increase motivation without the risk of them losing interest in the game. For this reason, the game has been decided to be set in a farm, just like the name suggests: this is because children are exposed to such environment from a very young age, be it at school or by other entertainment media.

The player will first interact with the owner of the farm, a character named James, who will ask for help with a task involving different types of animals that can be easily recognized by the player. Moreover, since it may be possible that the target audience is not yet able to read or may have difficulty in doing so, the amount

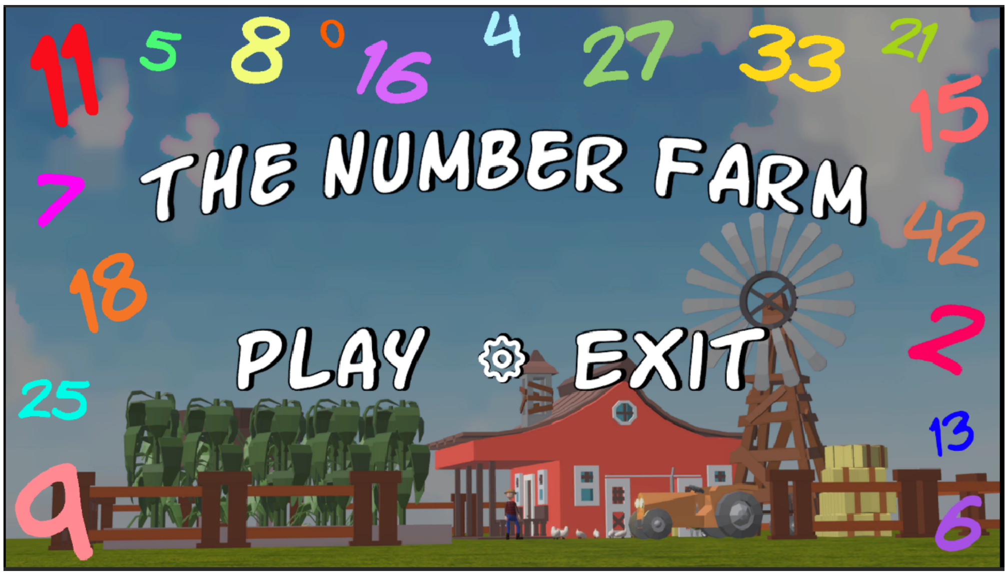


Figure 4.1: The main screen of the game.

.

of text within the game has been reduced to a minimum, which is complemented by full voice-acting of farmer James who will guide the player through the game.

The idea is to develop several mini-games that pertain different areas affected by dyscalculia, all encapsulated into a broader game taking place within the farm. As of now, a number comparison mini-game has been implemented, where the player has to help farmer James to understand which one out of two fences contains more animals. To make the mini-game clearer to the player, a brief tutorial is displayed before its start, where farmer James briefly explains the rules of the game. The animals that appear within the mini-game can be either chickens or cows, and the two sets of animals will differ according to features related to the ones described in section 2.2.2, as shown in figure 4.2. Additionally, there are two other parameters that can vary between trials, which are the show time for the animals, and the maximum time provided to the player to decide which fence to pick.

The game rewards the player with a success screen whenever they are able to perform the task correctly; when the answer is wrong however, a secondary screen is displayed where the player is asked to count the animals explicitly by dragging them at the side of the fence, as shown in figure 4.3. This is done to both provide a secondary activity to the game and to help the player understanding better when a mistake is made.

In order to provide an experience as accessible as possible, the game is also meant to be played from a mobile device. Because of this, all actions required to play the game are designed to be compatible with touch screen devices, as

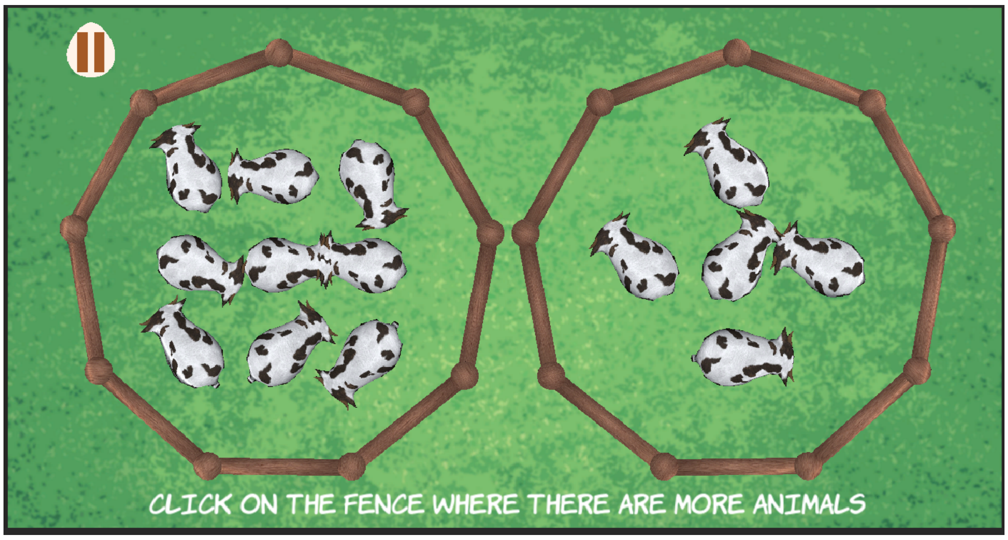


Figure 4.2: The screen where the comparison mini-game takes place.

.

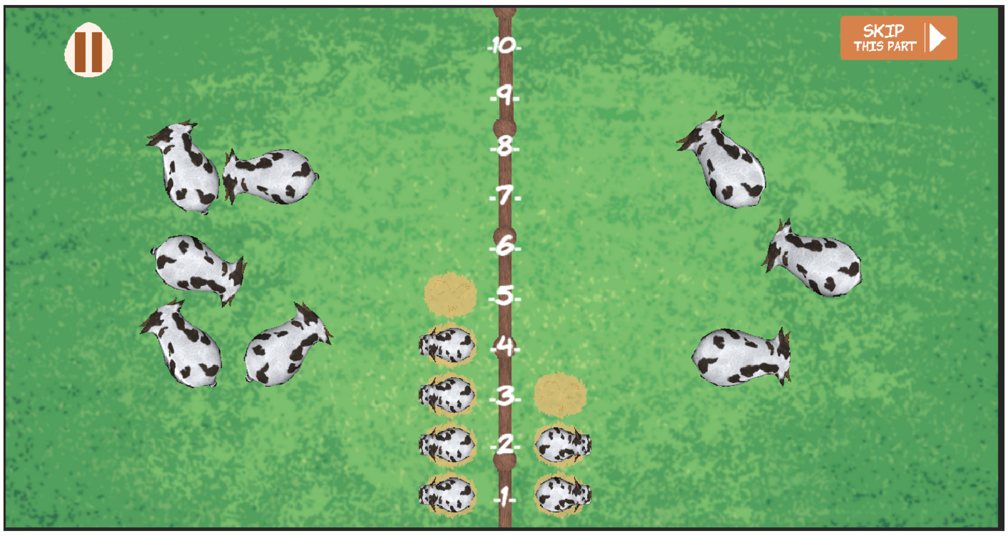


Figure 4.3: The counting mini-game displayed when the comparison task is unsuccessful.

.

it is believed to be a more motivating and intuitive technology for children [24]. Finally, although the game is split in a client-server fashion, no registration or authentication is required in order to load the player's progress, as such procedures are done automatically through characteristics of the device itself.

4.2 Theoretical Aspects

To summarize, the aim of the project is to offer a way to both detect symptoms of dyscalculia at a young age and to treat it in the case where it has already been diagnosed. As discussed in chapter 2 and 3, a possible way to do so is by putting the children in an environment where they need to manipulate numbers and quantities, so that they can get used to operations related to them.

It is however not wise to come up with a training program from scratch; hence, before describing how the project is structured and performs its task, some more attention will be dedicated to what is exactly used out of the theory that was discussed so far.

4.2.1 The contribution of Learning to Focus on Number

The main inspiration and difference with respect to previous similar works to The Number Farm is definitely the contribution brought by Learning To Focus on Number. In fact, this almost 2 years old project has been developed with the paper as its main reference since its early stages, and the theory regarding the filtering and sharpening hypotheses was included in almost all aspects of the game, starting from the visual components up to the AI.

As already mentioned in the previous section, in fact, the main game consists in a number comparison task, which was implemented to be as similar as possible to the one explained in section 2.2: the player is presented with two sets of elements displayed in the form of animals within a fence, and they have to decide which one of the two sets contains more by also avoid the influence of other features that have a correlation with the numerosity itself. The concept of **Numerical Dimension ND**, **Non-Numerical Dimension NND** as well as **congruity** will be borrowed from the paper and will be used to describe each trial in the related ND-NND space. Same can be said about the non-numerical features that characterize each of the trial's fences, that are reported here for completeness:

- **Item Surface Area**, or **ISA**.
- **Field Area**, **FA** in short
- **Total Surface Area**, abbreviated to **TSA**.
- **Sparsity**, also called **Spar**

A difference however lies in how these features are displayed from the client [25] [26]. In the paper all of these features were measured in terms of pixels, but since in the current project all of the assets are three dimensional, another set of parameters had to be designed in order to capture the non-numerical dimensions of each set, without losing their relationship with respect to the parameters used in literature. This matter was handled by the candidate's direct predecessors, who came up with the following set of parameters to define each trial within the client:

- **Size of the Animals**. This is strictly related to the ISA, but is expressed in a different measure.
- **Circle Radius**. A property that defines the space in which the animals will be placed within each fence.
- **Average Space Between**. Defines how spaced the animals will be between each other. It is also used to assign a position to the animals.

All three of them are directly related to the non-numerical features described in Learning to Focus on Number, more specifically to **ISA** and **FA**. For this reason, the candidate's predecessors decided to let the AI process the NND in terms of these two non-numerical features, while the client will translate both of them in its internal dimensions. More details on how the NND is computed within the project are explained in appendix B.

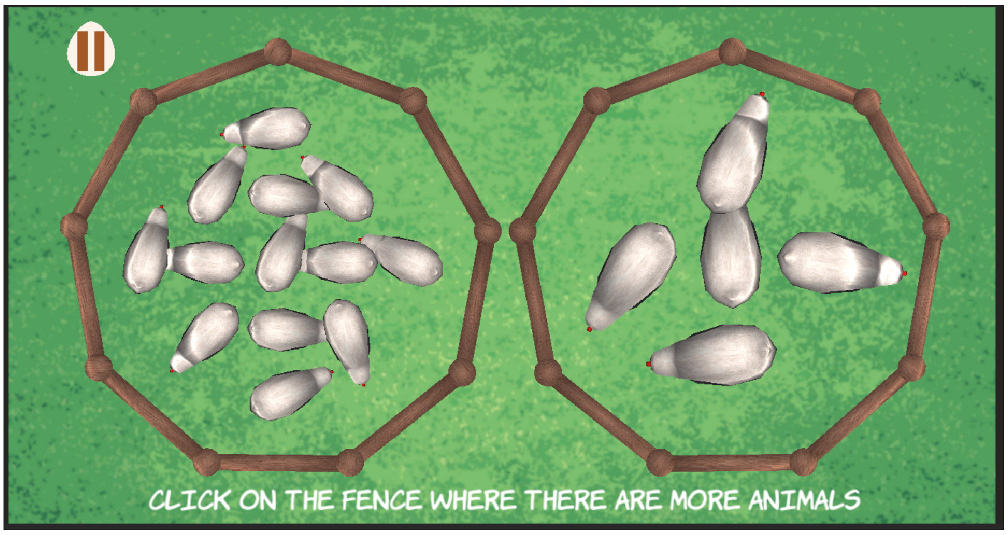


Figure 4.4: An example of incongruent trial. The two sets of animals differ both in numerosity, size of the animals and circle radius, with the non-numerical features being less prominent in the more numerous set on the left

.

Finally, the paper brings its most important discovery as contribution to this project: the AI, which will be described and showcased in chapter 5 and chapter 6 respectively, is in fact modelled after the filtering and the sharpening hypotheses.

The player is modelled as a classifier defined by both a filtering angle and a noise related to their internal representation of number within the ND-NND space, and the role of the AI will be to estimate these two parameters and propose suitable trials according to such estimation. Moreover, as the paper implies, the player will be considered to be able to improve, both according to the filtering and sharpening model.

4.2.2 Relationship with the State of the Art

The proposed game draws heavy inspiration from both the previously described training programs, since all of them are aimed at training children in mathematical operations.

Just like The Number Race, The Number Farm has the purpose of training the number sense through a number comparison mini-game, although devoid of the additional difficulty of having to perform additions and subtractions as the game progresses. This is done in order to better conform to the target of the game, which also comprehends kindergarteners who are not yet introduced with arithmetic operations. Training arithmetic skills and the links between the various representations of number are therefore not the objectives of this project, although mini-games related to such tasks may be introduced in the future. As a compromise however, the peculiarity of The Number Farm consists in the acknowledgement of the influence of non-numerical features in the comparison decisions, which is unprecedented for previous training applications. Together with that, the proposed game also is the first one to make use of the filtering hypothesis when it comes to the way children improve over time, as discussed in the previous section.

As already stated, The Number Farms aspires to be a full game comprised of several mini-games, each pertaining to a certain area related to mathematical understanding just like Calcularis. This is not only because it's vital to treat every aspect of dyscalculia, but also because introducing variety in the game-play is also the key to keep the player interested in the activity over a long period of time. It's worth noting however that The Number Farm already presents a step forward in that direction with respect to Calcularis, as it brings all such activities in a setting that can easily be appreciated by young children.

Another thing that sets The Number Farm apart from the two proposed predecessors is how the game is meant to adapt to its player: while The Number Race and Calcularis defined an adaptive algorithm in order to do so, The Number Farm has the goal of using tools related to **Artificial Intelligence**. Details on the AI will be discussed in the related chapter, and will exploit both **Machine learning** techniques as well as **Genetic Algorithms** to perform its task. Due to the lack of data specific to the filtering hypothesis, **Deep Learning** options were not explored within this thesis, although they may be taken into consideration in the case where the game is distributed to a first batch of children, which is entirely possible thanks to its accessibility.

4.3 Project Structure

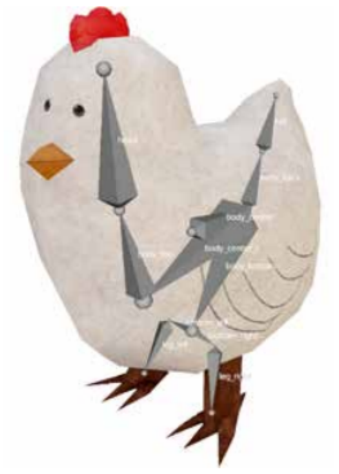
Before talking about the main features developed by the candidate, it can be useful to give some context about how the game works from the point of view of implementation. In section 4.1 it was stated that the game is supposed to be played from a mobile device; however, as the game requires AI methodologies in order to work properly, enclosing all the game's features inside a single mobile application can be highly impractical. Moreover, operations such as collection of data related to the game would be hard to manage if each device can only store the application's data locally. For these reasons, the game has been split into two parts following a client and server approach, both being managed as a single project by several contributors through the use of Git.

4.3.1 The Client

The client is the part of the game that runs in the user's device, and although the candidate was not in charge of its development, it's important to discuss briefly its main functionalities. Due to its compatibility with a wide range of devices, the client was developed inside the Unity game engine (version 2020.1.9f1), by using C# to write the scripts that manage all the Game Object of the application. All graphical aspects were developed inside this part of the project, with all asset being three-dimensional and with proper animations. In order to create them and related models, the candidate's predecessors used Blender and Maximo, due to their ease of use, while the UI was made by using several software including Photoshop and Premiere Pro [25].

The client's code is split in several scenes, each of them composed of several Game Objects with their related scripts. Other than showing the user the graphical aspects of the game, the client also needs to manage user input and to run the comparison mini-game. The server's contribution is however fundamental for running such mini-game, so right after displaying the tutorial, the client opens a TCP connection with the server which either authenticates or registers the user based on its IP address. To facilitate development and testing of the devices running the client, a screen dedicated to specifying the server's IP address as well as the amount of trials to play within the gaming session has been added to the game, as shown in figure 4.6

After the connection has been established, the client will refer to a set of commands in order to communicate with the server, which are handled by a suitable script (named ClientToServer.cs, handling the connection from both directions):



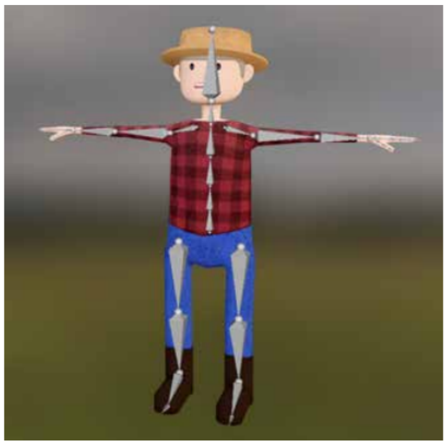


Figure 4.5: The models used for the chickens and for farmer James

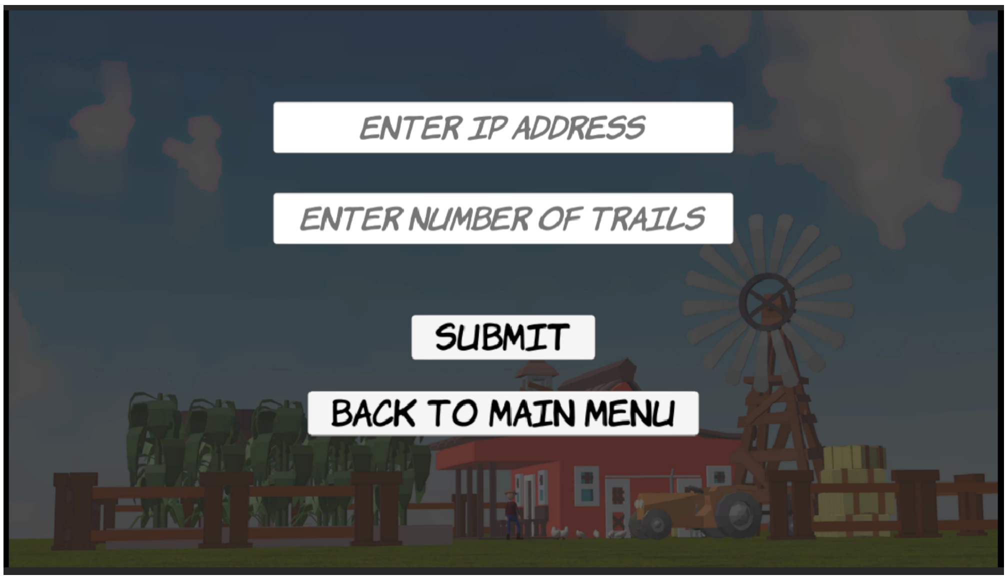


Figure 4.6: The settings menu, which can be opened from the main screen, where it is possible to specify the server's IP address and the amount of trials that will be proposed during the session

.

- **"TRIALS"**. This command can be used to ask for the next trial that needs to be shown to the player. The decision of the numerical and non-numerical features of each trial is in fact made by the server which, after receiving this command, selects a suitable trial, translates the features of each of its sets in the parameters used in Unity, and sends it to the client as a JSON object, so that it can be easily displayed inside the mini-game.

- **“COMPLETE”**. After the player has selected one of the two fences for the proposed trials by touching on one of the fences, the client runs a quick validation to understand if the answer was correct or not. The command “COMPLETE” is reserved to notify the server of the trial’s outcome, and is therefore sent together with a JSON object containing details on the player’s response, which are used by server to update its internal statistics.
- **“END”**. This command is sent when the player closes the game, and is used to communicate the server that it’s possible to close the TCP connection.

The client therefore alternates between the TRIALS and the COMPLETE command, proposing the trials to the player and communicating the answer to the server. When the player’s answer is incorrect, the client also runs the counting mini-game, for which the server’s intervention is not required.

4.3.2 The Server

The server is the part of the project under the candidate’s supervision, and its functionalities can be summarized in three main points:

1. Manage client connections.
2. Generate and send trials to each connected client.
3. Interact with the Database

In order to accommodate for each of these requirements, the server’s code has been developed in Python, as it provides packages to handle TCP communications through python sockets, database queries as well as several libraries related to Artificial Intelligence. All the required python packages are listed in a dedicated file, to facilitate their installation. MySQL was chosen to run the database, which needs to be installed in the same machine running the server, and requires to be initialized with a set of scripts in order to create all the tables required by the application.

After all requirements are met, the server can be launched through a script (main.py). Moreover, as the server needs to work under several different conditions, the main script accepts also a set of arguments (defined in argument_parser.py), of which the main ones are listed in the following:

- **use_lan**. Specifies that the python socket clients can connect to has to be bound to the machine's local IP address. It allows the server to be reachable from its local area network, which is necessary whenever the client is run from a mobile device. If this argument is not specified, the server's socket is bound on the localhost IP address and can be reached only by processes running on the server's machine (i.e. Unity IDE, for testing purposes). As for the port number, the socket is bound to a default port in all cases.

- **evaluator**. As will be discussed in the following section, the server implements two different versions for the AI. This parameter can be used to specify which one needs to be used.

- **usability_test**. Launches the server in usability test mode: the server will treat every connection as a separate user, even when coming from the same device, and will save separate entries in the database for each of them.

For completeness, the implementation for each of the server's main functions will now be briefly explained.

Managing client connections

The server has to be able to support communications with several different clients, as the game is meant to be played by several users at the same time. In order to do that, the main script instantiates an object of the `GameServer` class, whose main method binds a python socket to the IP address and port specified from the arguments. After that, the server's main thread starts a loop where a non-blocking check is made on the socket. Whenever a new connection is accepted, the main thread creates a new socket linked to the client and checks whether their IP address is already present in the database: if it is present, it determines the related user and their current progress; if not, the server performs a query in order to register it.

Once that is done, the main thread launches a new thread that will take care of the client that requested a connection, which is managed by the `PlayerHandler` class. In its initialization method, this thread will instantiate an object of a class implementing the `PlayerEvaluator` interface, which will be referred to as **evaluator**, and will define methods for all tasks that need to be performed by the AI. The player thread then runs a loop with a blocking read operation on the client's socket: whenever the client sends one of the commands listed in the previous section, the `PlayerHandler` thread will be unblocked and deal with the request. Once the "END" command is received, the player thread will finally terminate, and the server's main thread will release its resources.

Generate and send trials to each connected client

The server's most important task consists in the generation of trials to be sent to clients, so that they can be proposed to the end users. This is done whenever the "TRIALS" command is received by the `PlayerHandler` thread. As already mentioned, this class instantiates an object of one of two classes implementing the `PlayerEvaluator` interface, based on the arguments provided to the main script. These two classes are named `SimpleEvaluator` and `PDEP_Evaluator`, and each of them implements a working version of the AI: the `SimpleEvaluator` will be explained better in the next section, and represents the AI prior to the candidate's contribution; the `PDEP_Evaluator` will be instead discussed in chapter 5, and is the main topic of this thesis' work.

Both the AI classes take care of the generation of trials and the player's evaluation in their own ways; however, before forwarding the generated trials, they both need to interact with another component, which will be referred to as **"Lookup Table"**.

To better explain the role of this component, it should be pointed out that the client is not able to handle trials with just any set of numerical and non-numerical features. In fact, due to the size of the screen, the client can only place a certain amount of animals inside each fence, which can vary based on their non-numerical features like their size. As the client trusts blindly the server's generation of each trial, the AI has to be able to generate trials where each set of elements can be shown inside the fences, in order to avoid interpenetration of the models or even a crash on the client side.

This problem has been solved by the candidate's predecessors. As a first step, they analyzed the client's implementation for all the admissible values of the number of animals and of the non-numerical parameters, namely the Size of the Animals, the Circle Radius and the Average Space Between, as explained in subsection 4.2.1. These ranges of values were then translated into the space of features used within the server, which are the same **FA** and **ISA** used in Learning to Focus on Number together with the numerosity of the set: this translation therefore defines a subspace of admissible feature combinations that can be easily processed by the server's AI, as shown in figure 4.7.

Going back to the Lookup Table, it is basically a CSV file containing a subset of all the trials that can be displayed within the client: starting from the aforementioned admissible values for each fence, this subset is in fact generated by combining some of them into pairs. Each entry of the Lookup Table therefore contains a triplet $\{number, ISA, FA\}$ for each of the two fences, along with other features related to the trial like its ND and NND values. Moreover, after receiving some feedback from a psychologist, a secondary Lookup Table was also generated, which was meant to be used for younger players as it comprised trials with an overall lower amount of animals.

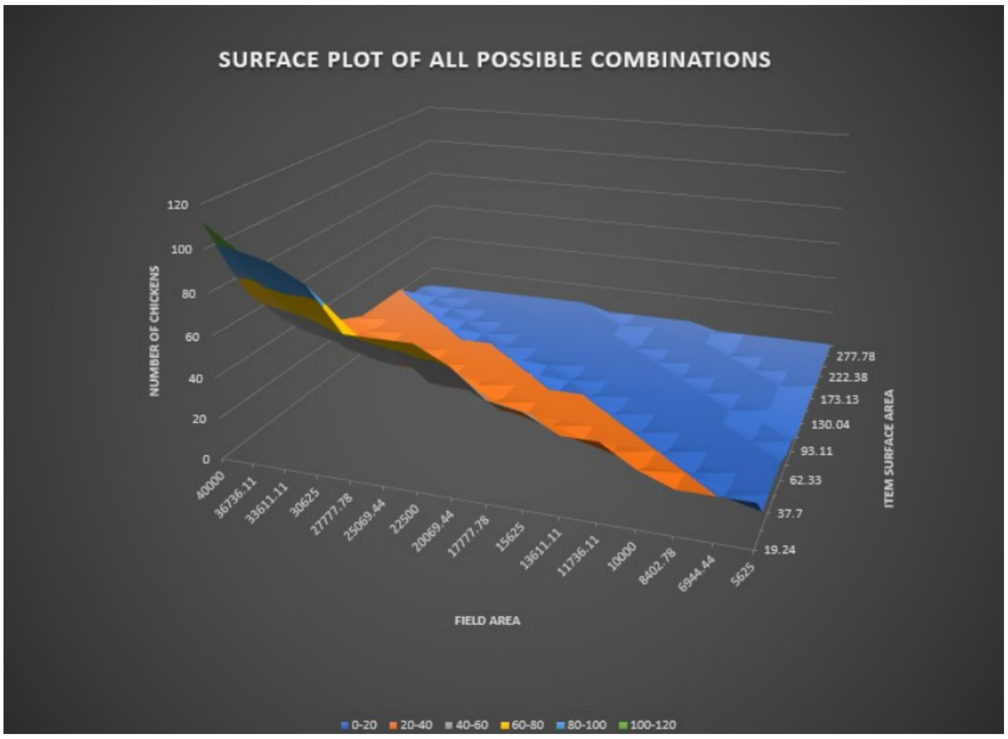


Figure 4.7: The 3D surface defining all possible combinations of features. Every point below the curve represents a triplet of number, FA and ISA that can be displayed within a fence in the client

.

The default behavior of the AI will therefore consist of generating a trial following its internal rules, and then to pick the actual trial that will be proposed to the client from the Lookup Table, according to some inter-trial distance criterion: after translating the trial in the set of parameters used in Unity, the server can safely send it without risking any unwanted behavior from the side of the client.

Interacting with the Database

The Number Farm is a game meant to follow the player as they grow, in order to better understand their skill in handling numerosity and whether the interaction with the game is useful in improving them. This implies that the game has to be played over a long period of time, where the data related to the player's answers has to be collected so that it can be processed by the game's AI.

For this reason, the server requires also the support of a database, which is installed inside the server's machine through MySQL. The application's database contains a table for the players (identified through their IP address) as well as two different sets of tables related to the two versions of the AI used withing the application. To summarize, the data stored in the database concerns the history of trials proposed to the player, together with their response and with additional statistics internal to the selected AI version.

Other than the main table for players, which is handled by the server's main thread, the tables related to the AI are updated by the `PlayerHandler`'s thread. A specific class, named `DBConnector`, has been developed to interact directly with the MySQL database, and implements all possible queries required by the AI. A single object of this class is instantiated in the main script, and a reference to it is passed to each player thread; then, whenever the client sends the "COMPLETE" command, the player thread first lets the evaluator object process the response, and finally calls the methods of the evaluator that, by accepting the `DBConnector` object as an argument, will take care of updating the database with the required data.

4.4 The Starting Point

As inferred from the previous sections, The Number Farm has been in development since before the candidate's arrival. This project was in fact started back in 2021 as an initiative of the University of Essex, and its first contributor already laid out a first version of the client and the server [27]. The project was then led by a working group comprised of students both belonging to Politecnico di Torino and University of Essex, including the candidate himself. In this section, the state of the game at the beginning of this thesis' activity will be described, with some emphasis also on the work performed by the candidate regarding general matters.

4.4.1 Overall Structure

At the start of this thesis' activity, the client was already implementing all the required main functionalities, with all assets used being three-dimensional [25]. The tutorial as well as the mini-game were already present, together with the client-server communication within the same machine. The user experience did receive some tweaks after the candidate's arrival, with the addition of voice over for all parts of the game as well as quality of life changes; these changes were added by other contributors however, as the development of the client side was not the candidate's area of responsibility. Therefore, the candidate's contribution on the side of the client is limited to the communication between client and server over a remote connection, which allowed the installation and testing of the game on mobile devices.

Regarding the server's code [26], the connection with clients had already been dealt with along with most of the database interaction; only some adjustments were performed by the candidate to allow for the correct handling of multiple clients at the same time. The admissible combinations of features for the trials were already detected by previous contributors, and a first version of the Lookup Table was already available. Moreover, all the code regarding the pre-processing of the trials before being sent to the client had already been developed, and remained nearly unchanged over the course of this thesis' activity. Some additional scripts were also present to run simulations of the server's code, but in this case they were re-implemented from scratch by the candidate, as reported in chapter 6.

4.4.2 The initial state of the AI

At the time of the candidate's arrival, a single version of the AI was already implemented and working [28]. This version was based on the concept of **"filtering and sharpening difficulty"**, according to which the trial, defined by its ND and NND, could be assigned two different difficulty coefficients between 0 and 1 related to the filtering and sharpening hypotheses: the filtering difficulty could be summarized as a normalized measurement of the **congruity** of the trial; the sharpening difficulty was instead simply a normalized value that was inversely proportional to the ND of the trial. In figure 4.8 are reported some graphical interpretations of the filtering difficulty.

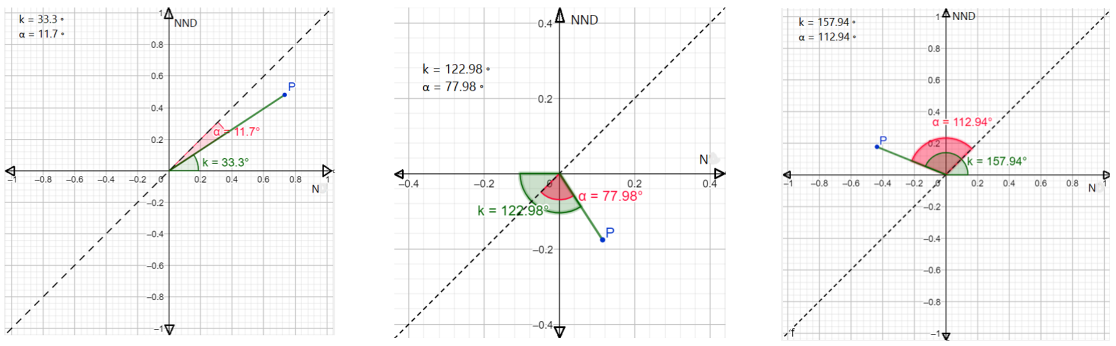


Figure 4.8: Graphical representation of the filtering difficulty for trials in the ND-NND space. In these figures, the line passing through the origin forms a 45° angle with both the ND and NND axis. The filtering difficulty is the angle (α in the figure) between this line and the line connecting the origin with the trial. The part of the line from which the angle is computed depends the NND of the trial, which brings some inconsistencies in this difficulty coefficient as discussed in the next subsection

In this version, the player is modelled with two parameters, the **filtering and sharpening skill**. More in detail, they are both a number between 0 and 1 and are strictly related to the aforementioned concepts of difficulty: for example, if the player has a filtering skill equal to 0.5, the AI will consider them to be able to answer correctly to trials that have a filtering difficulty lower than such value, while trials with higher filtering difficulty will be considered to be hard; the same relation exists between the sharpening skill and the sharpening difficulty. The role of the AI is therefore to estimate the sharpening and filtering skills of the player, and to use these estimated values in order to generate balanced trials to be sent to the client.

The criterion for generating suitable trials and to estimate the two skills can be summarized in the following steps:

1. First, the AI selects a “mode”, that can be either related to the filtering or sharpening difficulty. The method for selecting the mode was initially a simple alternation, and was later changed into a more complex function by other participants of the project [29].

2. Given the mode, the related filtering or sharpening skill that was estimated so far for the player is selected.

3. A search in the Lookup Table is then performed: as each one of its entries contains the parameters for the related trial together with its two difficulty coefficients, the AI selects the trial whose difficulty related to the current mode is the closest to the skill selected in step 2, and forwards it to the client.

4. Once the player’s response is received, the AI updates the estimated skill of step 2 by increasing or decreasing it by a fixed amount, based on the correctness of the answer.

4.4.3 The first activities

The first period of this thesis’ activity consisted in studying the theory behind the paper Learning to Focus on Number and in understanding the general structure of the project, as the candidate was appointed to be the main contributor of the entirety of the server’s code.

Soon after, the candidate’s first contributions were brought to the project which, other than the aforementioned remote connection between client and server, were related to the database support for the AI mentioned in the previous subsection. As the candidate was in charge of the development of the AI, a thorough assessment of the current version’s capabilities was then carried out, with the aim of understanding whether expanding on its principles would benefit the application or not. The assessment was conducted by taking the theory behind Learning to Focus on Number into account, and therefore required recomputing the Lookup Table with the addition of the ND and NND values for each trial.

Unfortunately, some inconsistencies were found within the AI's working principles. First of all, the alternation of the proposal procedure in the two working modes led to an important problem in the generation of the trials and on the update of the player's estimated parameters. The trial to be proposed was in fact chosen according only to the current mode, and the update was applied only to the related player's skill: the possibility that the player answers incorrectly because the trial is too difficult according to the other "inactive" mode can't be, in principle, neglected; yet, this version of the AI could potentially propose very imbalanced trials in terms of the inactive mode, and still penalize the current mode's skill in the case of an incorrect answer. Moreover, other than being hard to relate to the theory of Learning to Focus on Number, the principles behind the sharpening and filtering difficulties led to some further problems, which are discussed in figure 4.9 and 4.10.

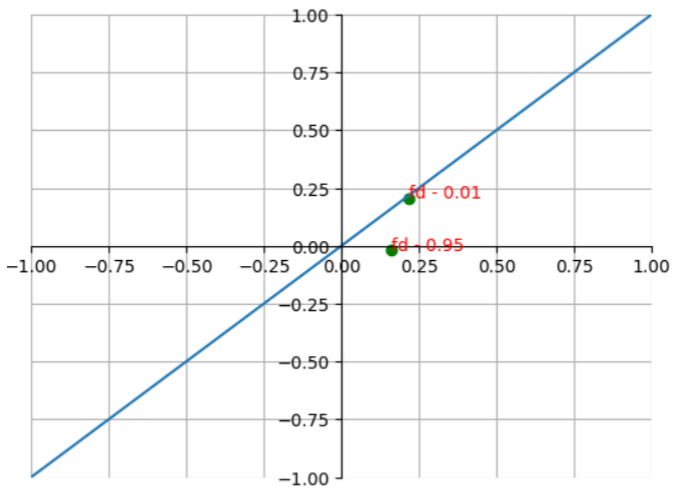


Figure 4.9: An inconsistency of the filtering difficulty. The figure shows two trials within the ND-NND space and a line passing through the origin with a 45° angle with respect to both axes. In red is instead reported the filtering difficulty for both trials: it's easy to see that, although the two trials are very close in the ND-NND space, they are assigned with two completely different filtering difficulties. This is because such metric depends on two different angles based on the NND of the trial: if positive, the angle is measured with respect to the part of the line in the first quadrant; if negative, the angle is instead measured with respect to the part of the line in the third quadrant

.

Moreover, a primitive version of the simulation environment described in chapter 6 was developed, and was used to test this AI's performances: the various simulations proved that the AI was not able to come up with consistent estimations for the player's skills and therefore couldn't propose balanced trials to the player.

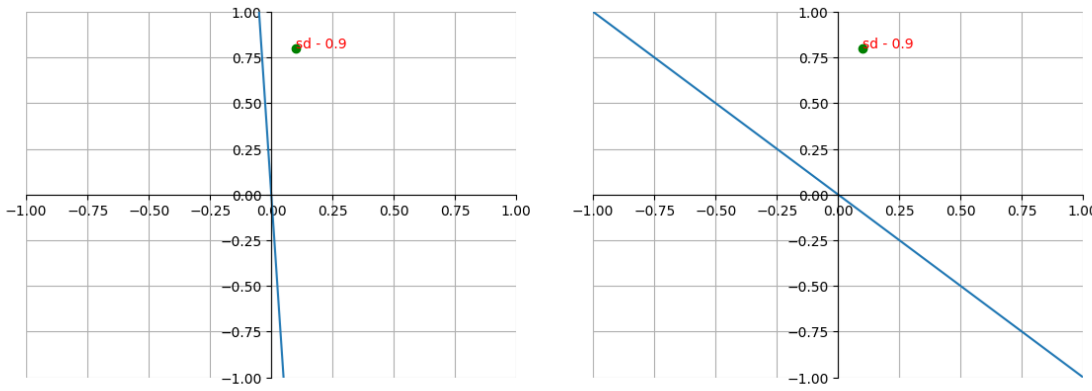


Figure 4.10: An incongruence of the sharpening difficulty with respect to the filtering hypothesis. In the two figures is shown the same trial that, having a ND value close to zero, has a high sharpening difficulty coefficient. However, according to the filtering hypothesis, the player's decision boundary (the blue line in the two figures) does not necessarily coincide with the NND axis, and is instead defined by an angle with respect to it. The shown example identifies a problem in the definition of the sharpening difficulty because the player with a less optimal decision boundary (right) is very likely to answer correctly to the proposed trial, as it's very distant from its decision boundary, while the more skilled player (left) can possibly give a wrong answer due to their noise in numerical representation.

For these fundamental reasons, the candidate decided that instead of improving the current AI, it could be wise to design a new version from scratch that would follow the filtering and sharpening hypotheses more rigorously. This decision was not to be taken lightly however: The Number Farm is in fact a project expanding in several different directions, and some contributors were still working on the initial version of the AI. Moreover, a usability test where the game had to be used by real children in the project's laboratory was scheduled around in the middle of this thesis' activity. As the new AI was estimated to take some time to be finished, these factors meant that its development could not just replace the one of the starting AI, but rather **had to go in parallel while the latter was still maintained by the candidate**.

This led to the necessity of reorganizing the whole code related to the AI and to its interaction with the rest of the server's code. First of all, the set of arguments mentioned in subsection 4.3.2 were introduced to distinguish the several operating modes of the server, including the desired AI version. Then, a new package solely related to the AI was brought in the project, and the `PlayerHandler` class was changed to interact with the `PlayerEvaluator` super-class contained in it as an interface for accessing the functionalities of the AI. Finally, all the code related to the AI was moved inside the proper package, and two subclasses of `PlayerEvaluator` were introduced: the first one, called `SimpleEvaluator`, stands for the initial version of the AI, and implements its trial proposal and statistics update procedures in a way that is compatible with the super-class; the second one, named `PDEP_Evaluator`, was instead meant to implement the new AI's behavior.

In this way, the candidate was able to leave the initial AI in an accessible and working state and to maintain it in collaboration with other contributors. Moreover, this version of the AI, implemented as the `SimpleEvaluator`, was the one used during the aforementioned usability test. This, of course, was done in parallel with the development of this thesis' main topics: the `PDEP_Evaluator` which will be described in detail in the following chapter, and its simulation environment, that will be discussed in chapter 6.

Chapter 5

The AI

As this thesis' title suggests, the main topic of the candidate's work is related to the AI, for which they are the sole developer. This chapter is therefore the most important of this thesis, as it's meant to describe the principles of such AI together with the implementation of its functionalities. For simplicity and for coherence with respect to the actual implementation, this version of the AI will be referred to as `PDEP_Evaluator`. This chapter will therefore start by introducing the general ideas behind the `PDEP_evaluator`, followed by a detailed explanation of how it performs its functionalities. Moreover, a section will be dedicated to how it interacts with the client, while the chapter will be closed with the explanation of how it adapts to the changes in the player's ability.

5.1 Introduction

In the previous chapter the main idea behind The Number Farm was explained. The game is in fact meant to provide a non-symbolic number comparison game where the player is asked to select one of two fences containing more animals, and in doing so, the game has to follow the theory of Learning to Focus on Number. Within this system, the AI has the role of generating the trials to be proposed to the client so that their difficulty is balanced and adapted to the player's skills. The contribution of Learning to Focus on Number was already discussed regarding the entirety of the project; now, for completeness, its concepts that are fundamental for the understanding of the `PDEP_evaluator` are reiterated in greater detail:

- The trials, which are composed of two sets of animals, can be described by two properties named **ND** and **NND**: the former is defined as the logarithmic ratio of the number of elements respectively of the right fence over the left one; the latter is instead the logarithmic ratio of the non-numerical features (i.e. the **FA** and the **ISA** reduced to a single value) of the set displayed on the right fence over the ones of the set displayed on the left fence. These two values allows trials to be visualized as points in the ND-NND space, where the ND's value determines the correct answer to each trial: if ND is greater than zero, the correct answer for the trial is **"right"**, as in this case the right fence is the one containing a greater number of animals; if ND is instead negative, the correct answer will therefore be **"left"** for similar reasons.

- The player can be modelled as a linear classifier in the context of the proposed mini-game that, by receiving as input the trials in the ND-NND space, is meant to label each one of them as either "left" or "right" according to the player's perception. This perception is modelled as the combination of a linear decision boundary (i.e. a line passing through the ND-NND origin) and of a numerical representation noise: the former is characterized by the angle formed with the NND axis, which will be called α, and represents the influence of the NND in the player's decisions mentioned by the filtering hypothesis; the latter instead is an interpretation of the sharpening hypothesis and is modelled as a two-dimensional Gaussian noise in the classifier's perception of the trial within the ND-NND space. Moreover, the aforementioned Gaussian noise will be considered to have a covariance matrix equal to the identity matrix times a constant, which represents its standard deviation σ. This model allows to represent the player through the couple of parameters $\{\alpha, \sigma\}$.
- The player is considered to be able to improve with time and practice according to both the filtering and sharpening hypotheses. Given the modelling of the player of the previous point, this can be considered as a decrease in the player's α and σ parameters as the game is being played.

These principles define a solid set of rules for the `PDEP_Evaluator`, as they allow the AI to interpret the game and the player in a space that is easier to analyze while still being consistent with the theory. As opposed to the SimpleEvaluator, this new version will "think" in the ND-NND space, normalized in the range $[-1{,}1]$ for simplicity, by generating the trials as points within it; moreover, the definition of their difficulty will not just be a fixed property related to their position, but it will also depend on the player's α and σ parameters. It goes without saying that, in order to properly adapt to the player's skill, the AI will also have to estimate these two parameters, which are considered to be in the range of $[0, 90]°$ and $[0.0, 0.5]$ units respectively, based on the player's answers to the previously proposed trials. Additionally, the evaluator will also have to understand whether the player is improving as they are playing, and if they do, it's also important to understand how fast they are learning: reaching such understanding can greatly help the α and σ estimation by giving a criterion on how much and how old the trial data used should be.

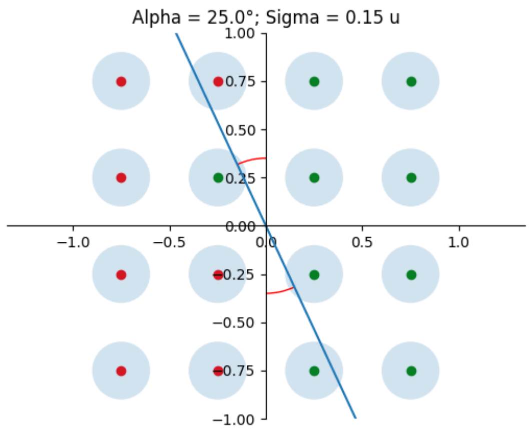


Figure 5.1: Graphical representation of the theoretical principles in the ND-NND space. The blue line represents the player's decision boundary, with the red arcs representing the α angle. Each point represents a trial, with green ones being labelled as "right" and red ones as "left". Each trial is surrounded by a blue disk with radius equal to σ, which represents the standard deviation of the noise applied to them: this makes the classifier's labelling less accurate for trials near to its decision boundary

.

To perform these tasks properly, the `PDEP_Evaluator` can be divided logically into several interacting components, whose typical workflow is described in the following:

- **Difficulty Controller**. In order to be effective, the training program has to both provide easy and difficult trials, according to a certain criterion. At each mini-game iteration, this component generates a set of parameters that define the requested difficulty of the current trial, based on the difficulty of the previously proposed ones.

- **Trial Proposer**. This component takes as input the currently estimated values for α and σ together with the difficulty parameters generated by the Difficulty Controller, and uses them to generate a pair of values related to the ND and the NND, both representing the next trial to be proposed to the client.

- **Trial Adapter**. It takes the trial generated by the Trial Proposer as input and processes it: first, the trial is translated into a suitable couple of

$\{Number, FA, ISA\}$ parameters, following the requirements for its admissibility; then the component will interact with the rest of the server's code to translate the trial into the parameters used in Unity.

- **Player Estimator**. By defining the trial proposed at the ith mini-game iteration as t_i and the related player's response as r_i, this component's input at the nth iteration will be represented by the sets $\{t_{n-x}, t_{n-x+1}, ...t_n\}$ and $\{r_{n-x}, r_{n-x+1}, ...r_n\}$, where x is an integer value dependent on the player's improvement rate for both α and σ. This component will have the task of estimating both the α and σ parameters after every trial proposed, as accurately as possible so that the Trial Proposer can perform its task properly.

Each of these components will be described in detail in the following sections.

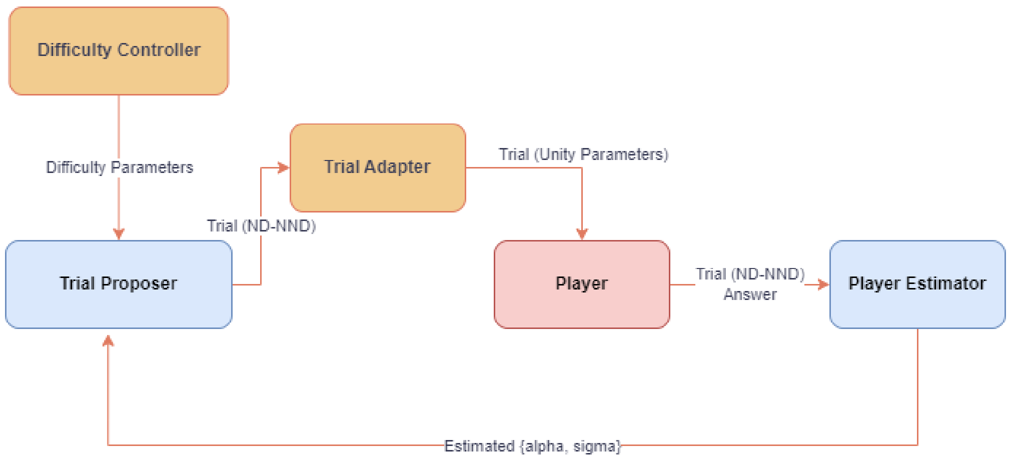


Figure 5.2: Graphical illustration of the AI's components and their interactions.
.

While the `SimpleEvaluator` proposes trials by only considering the player's skill, the `PDEP_Evaluator` is designed to both acknowledge the player's α and σ parameters as well as a degree of difficulty for the proposed answer. The "PDEP" belonging to its name stands for "Perceived Difficulty and Error Probability", which is exactly the principle implemented by the Difficulty Controller and followed by the Trial Proposer at trial generation time. In the following section, this idea will thoroughly explained, while the way it's used by the Trial Proposer will be discussed in section 5.3.

5.2 Controlling the Difficulty

The **Perceived Difficulty (PD)** and the **Error Probability (EP)** are the two main ideas behind the difficulty calibration of the trials proposed by the application.

Each one of them is expressed by the Difficulty Controller as a single value in the range $[0, 1]$, and are meant to both control the spontaneity of the player's answer as well as the overall accuracy reached by the player as they play the game. Moreover, other than calibrating the difficulty of the proposed trials, the Difficulty Controller has also a secondary goal: as will be described in the following sections, the difficulty parameters determine the distribution of the ND-NND values of the proposed trials, which are required to reasonably span the ND-NND space so as to provide the Player Estimator with sufficiently good data to work with. Before discussing how the Error Probability and the Perceived Difficulty are interpreted by the Trial Proposer, it's worth discussing the meaning of these two metrics and how they are regulated by the Difficulty Controller.

5.2.1 The Error Probability

The Error Probability is rather self explanatory: whenever a new trial has to be proposed to the client, the Difficulty Controller will select a value between 0 and 1, called **target error probability**. This value represents the desired probability that the player answers incorrectly to the current proposal, which is later used by the Trial Proposer together with the player's estimated α and σ to generate a trial reflecting such probability. The value is picked in a Round-Robin fashion according to the following simple procedure:

1. At initialization time, an array of N "candidate values" for the target error probability is generated.

2. The first time the player starts the game, the first target error probability p_1 is selected randomly from the N candidate values, while also being removed from the array of candidate values.

3. p_1 is used as the target error probability for a certain amount of mini-game iterations, which will be referred to as M.

4. At the iteration number M+1, p_1 is discarded, and a new value p_2 is selected randomly out of the N-1 values left in the array of candidate values, from which it is then removed.

5. Step 4 is repeated is repeated until the array of candidate values becomes empty, and each time the selected value is used as the target error probability for M trial proposition iterations.

6. After that, the Difficulty Controller will keep selecting the values from the original array of candidates based on which one was used the most further back in time. As usual, the selection will last for M trial propositions.

The array of candidate values is generated so that its mean value is equal to the desired overall accuracy to be achieved by the player: if the Trial Proposer is good enough at following the Difficulty Controller's guidelines, the player's accuracy should in fact converge to the array's average value. Moreover, the Trial Proposer is considered to be able to reasonably perform its task: the Difficulty Controller does not use in any way the actual accuracy measured during the playthrough.

As of now, the AI allows to calibrate the overall desired accuracy to either 50% or 70%. Although the implementation can be easily changed to allow a generic value for the target overall accuracy, the array of candidate values is hard-coded for both the provided difficulty calibrations: this was done to carefully control the difficulty of each trial proposed to the player, but also to completely avoid certain values from being selected as the target error probability. In fact, as will be later discussed, target values too close to 0.5 imply trials that go too in contrast with the other difficulty metric, which is the Perceived Difficulty; moreover, trials generated with such target error probability generally are not very useful for the Player Estimator, as will be explained in section 5.5.

5.2.2 The Perceived Difficulty

If the Error Probability is meant to control how "tricky" the trial has to be to the current player, the Perceived Difficulty is instead a metric that is supposed to define how hard it should be for the player to give an answer to it, regardless of its correctness. To better explain this concept, here's an example: assuming a fixed target error probability, a trial with a low Perceived Difficulty is a trial that should be answered instinctively by the player; a trial with a high perceived difficulty is instead supposed to not be intuitive to the player, and should be perceived as more challenging regardless of the outcome of the answer.

Just like the target error probability, the Perceived difficulty is expressed as a number between 0 and 1, named **target perceived difficulty**. In principle, this value should not be co-related with the Error Probability, and most importantly it should not affect the overall desired accuracy scheduled by the Difficulty Controller. Given the player's model described in previous section, this idea could be put into practice by, for example, assigning a higher perceived difficulty value to trials with higher cardinality of the numerical feature, as the ND of the trial is dependent just on the ratio of the cardinality of the two fences. In practice, however, such relationship cannot just be assumed by the candidate, but should rather be inferred from real data related to a similar task as the proposed comparison game: a possible way to do so would be, in fact, to relate the perceived difficulty to the time taken by the player to come up with an answer, which can then be used as a label to understand what kind of features of the trial have an impact on the overall response time.

As already mentioned before, however, no data coming from real number comparison experiments was provided for the development of the AI; for the time, the candidate therefore came up with a reasonable mathematical meaning for the target perceived difficulty, which is rather similar to the concept of sharpening difficulty of the SimpleEvaluator. The Difficulty Controller keeps it at a fixed low value, and the way it is used by the Trial Proposer will be discussed in the following section, along with its implications.

5.3 Generating the trial

In section 5.1 it was stated that the generation of the trials was handled by the Trial Proposer, by using as input both the estimated α and σ parameters of the player as well as the difficulty parameters produced by the Difficulty Controller. Now that also the **target error probability** and the **target perceived difficulty** have been discussed, it is now time to explain how these parameters are used by the Trial Proposer to generate suitable trials in the ND-NND space following their specifications.

5.3.1 Computing the Difficulty for a generic trial

The Difficulty Controller tells the Trial Proposer the set of properties that the generated trial should have, expressed as the probability of being guessed incorrectly by the current player and the "perceived difficulty" that such player will experience when giving the answer; how these two metrics are computed for a generic trial is however still an open question, which will be answered in the following.

Computing the Error Probability

Let's address first the computation of the **error probability**. At this point, the Trial Proposer is made available of the α and σ parameters for the player that were estimated during the previous mini-game iteration: given these two parameters and a generic trial t in the ND-NND space, its perception by the classifier can be graphically explained by figure 5.3.

For simplicity of the explanation, the same situation is also proposed in figure 5.4, which also displays the **normal vector** $\vec{n}$ to the player's decision boundary.

Given the normal vector, a more rigorous interpretation to the player's labelling procedure can be defined. By considering the trial as a vector $\vec{t}$, in the ND-NND space, the player's labelling is related to the sign of the projection of $\vec{t}$ onto $\vec{n}$: when it has a positive sign, the trial is labelled as "right"; if negative, the label "left" is instead provided as an answer. This procedure can be also viewed as the translation of the trial in a new set of coordinates, with its axis translated

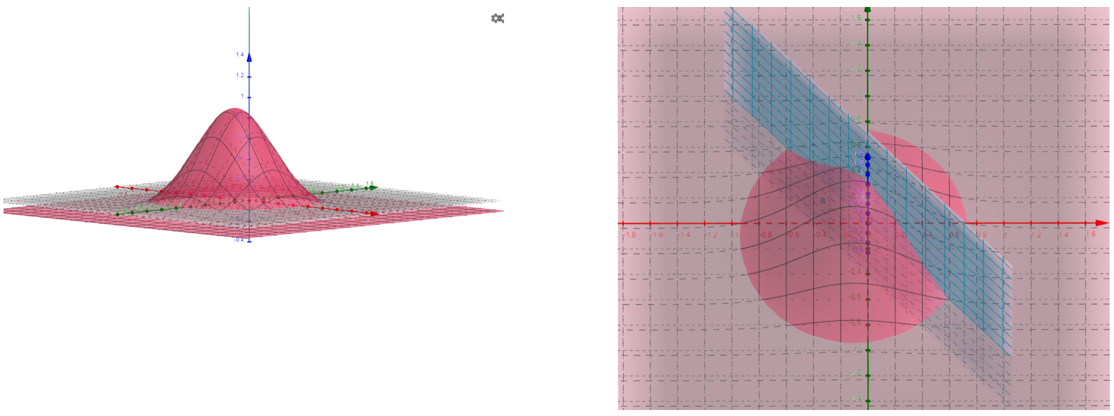


Figure 5.3: Graphical 3D representation of the trial's perception by the player, from two different points of view. The red and green axis represent the ND and NND, while the z axis represent a probability density. Moreover, the blue plane represents the player's decision boundary characterized by α. The trial, due to the noise related to the σ parameter, can be seen within the ND-NND space as a random variable characterized by a two dimensional Gaussian distribution centered on the trial's ND-NND coordinate and with uniform standard variation equal to σ

.

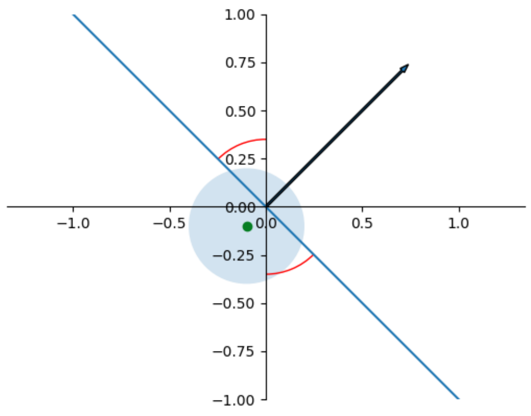


Figure 5.4: Simplified representation of figure 5.3. The arrow represent the unit vector that is perpendicular to the decision boundary, which by convention is considered to always lie in the first quadrant

.

with respect to the ND-NND ones by an angle α and representing the decision boundary and its normal vector: in this set of coordinates, the aforementioned projection is represented by the trial's component with respect to $\vec{n}$, which can also be interpreted as the **signed distance** of the trial with respect to the decision boundary.

Going back to the trial shown in the figure, it can be observed that it has a negative ND value, which means that the correct answer for it would be "left": through the aforementioned interpretation of the labelling, it's easy to see that the trial should be normally labelled correctly; due to the σ parameter however, the trial has only a limited probability of being perceived on the correct side of the ND-NND plane, that in the figure is numerically equal to the volume of the Gaussian bell below the player's decision boundary.

Given these observations, it is possible to give a more concrete definition to the Error Probability for a generic trial, which is the **fraction of volume of the Gaussian bell on the "wrong" side of the decision boundary**, where the wrong side is defined by the ND of the trial in question (i.e. the side of $\vec{n}$ when ND is negative, and the one hosting $-\vec{n}$ when ND is positive).

As the python's `scipy` library provides a set of functions to perform two-dimensional integration, the first implementation of the Error Probability computation was designed to compute explicitly the fraction of volume of the bell. This implementation was rather computationally demanding, and although it was not shown to impact the user experience significantly, it was considered to be too slow when simulations of the game had to be run. For this reason, the computation was further optimized, as shown in figure 5.5. Basically, as the bell's covariance matrix has been assumed to be equal to the identity matrix times σ (i.e. similarly to a multivariate Gaussian distribution defined as $\mathcal{N}(t, \sigma I)$), the problem is mathematically equivalent to its one-dimensional visualization where the trial is defined solely by its **signed distance d with respect to the decision boundary** : the noise in the representation of the trial is just viewed with according to its distance with respect to the boundary, and is now interpreted as a one-dimensional Gaussian distribution $\mathcal{N}(d, \sigma)$. With this interpretation of the problem, the Error Probability can be easily computed through the use of the erf function, suitably adjusted to the aforementioned distribution. Although no proof is provided for the equivalence of the two methods for computing the Error Probability, its correctness has been verified empirically through a set of experiments, where both methods would produce nearly identical values under various conditions.

Computing the Perceived Difficulty

As mentioned earlier, the current interpretation of the Perceived Difficulty is rather similar to that of the "sharpening difficulty" described in subsection 4.4.2, but with

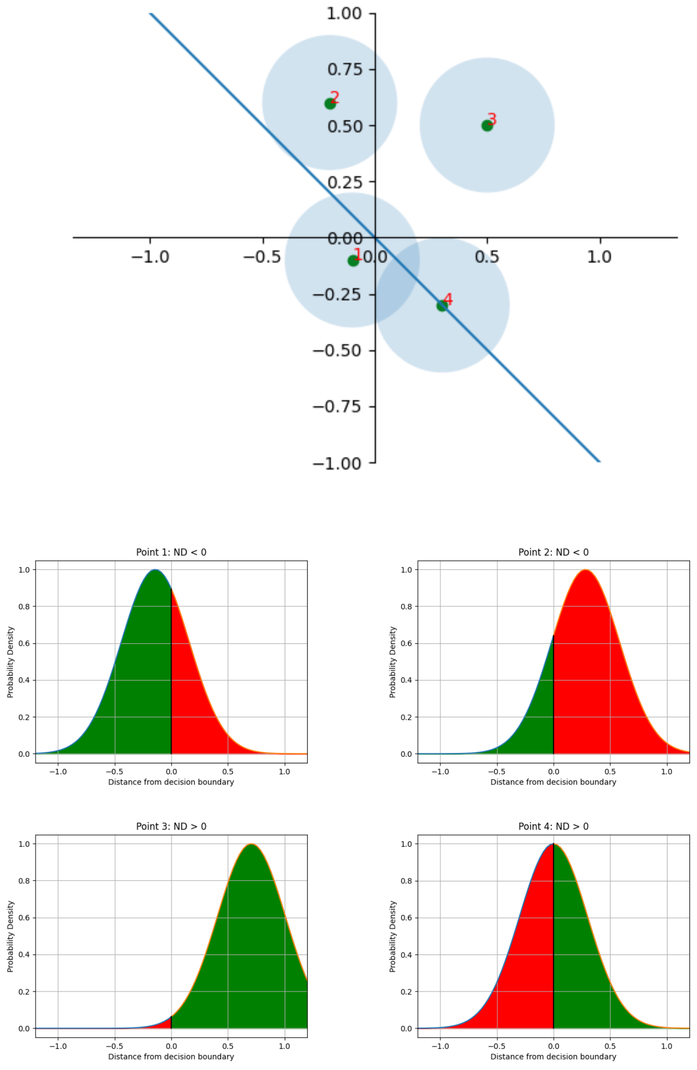


Figure 5.5: Graphical explanation for the optimized Error Probability computation for several trials. In the top figure, the trials are shown in the ND-NND space, with the player's α and σ being represented by the decision boundary and the disks around the points. In the bottom figure lies the one-dimensional simplification for the Error Probability of each point, where its value is represented by the red areas. It's worth noting that trials with a high error probability (i.e. > 0.5) must lie in the space between the boundary and the NND axis.

the additional feature of being dependent also on the player's α parameter: instead of being inversely proportional to the distance with respect to the NND axis, it is in fact computed as 1 minus the normalized distance of the trial with respect to the player's decision boundary, as shown by figure 5.6.

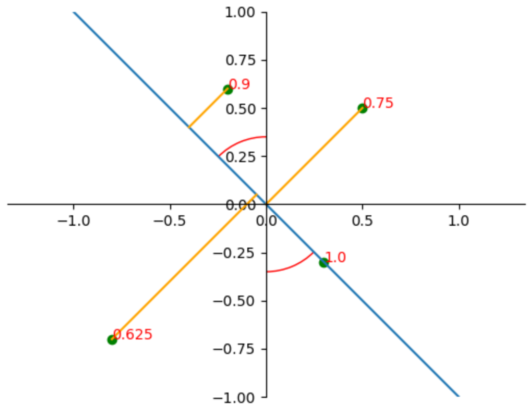


Figure 5.6: The Perceived Difficulty computed for several trials, which depends on their distance with respect to the decision boundary in the ND-NND space

.

Given all the background explained in the previous paragraphs, it shouldn't be necessary to explain the details on how this value is computed. However, as anticipated in section 5.2, its definition is not perfectly fit for being used in the current application as by design it depends on the **distance with respect to the decision boundary**, which is a quantity that the Error Probability also depends on. This means that, although the Error Probability depends also on the **sign of this distance** and on the ND variable of the trial, optimizing the Trial Proposer's choice according to the target perceived difficulty may go against the optimization for the target error probability. This can happen when, for example, the target perceived difficulty is set to be low, but the target error probability is a value close to 0.5: optimizing for the first metric requires moving the chosen trial far from the boundary, which changes its Error Probability; optimizing for the target error probability instead requires the trial to be as close as possible to the boundary, which makes the trial's Perceived Difficulty become close to its maximum value. To mitigate for this, the target error probability is never chosen to be too close to 0.5, while the target perceived difficulty is always kept at a low value; moreover, as will be shown in the next subsection, the target perceived difficulty is treated as a secondary criterion for the proposal's optimality, with the target error probability being the main one. When a new method for computing the Perceived Difficulty will be introduced, possibly disentangled by any dependence on the ND and NND values of each trial, it will be possible to perform a separate optimization for the target perceived difficulty, possibly being regulated by the player's measured response time.

5.3.2 Searching the ND-NND space

Since the way the two difficulty metrics are computed for a generic trial has been explained, it is now time to discuss how these two are combined into an optimality score, and how the actual search for an optimal trial within the ND-NND space is performed.

Obtaining the Optimality Score

Finding an exact solution for the optimization of the **target error probability** and the **target perceived difficulty** may be, in principle, possible. The problem is in fact tied to several mathematical relationships that can be probably solved from a mathematical point of view, given some criterion. Implementing an ad hoc solution for that, however, would make this part of the AI lacking in scalability, and as the definition of Perceived Difficulty will likely change in the future, it's simply not wise to design an algorithm from scratch to perform the optimization.

For this reason, it may be a better idea to follow a more modular approach, where the optimization algorithm is independent of the actual quantity that it is trying to optimize. The usual starting point for such approaches is the definition of a function that takes as input a generic solution, which is a trial in this case, and outputs a **numerical measure of its quality**. By naming the target error probability and the target perceived difficulty as T_{ep} and T_{pd} respectively and by joining them with α and σ into a single set of parameters named Θ, the candidate decided to define the **Optimality Score** S_Θ for a trial t as

$$S_\Theta(t) = w_{ep}|T_{ep} - EP_\Theta(t)| + w_{pd}|T_{pd} - PD_\Theta(t)| \tag{5.1}$$

where EP_Θ and PD_Θ represent the Error Probability and the Perceived Difficulty of a generic trial given the α and σ parameters contained in Θ, while w_{ep} and w_{pd} represent two weights in the range [0,1]. The proposed score can be simply seen as a weighted sum of the distances of a generic trial's Error Probability and Perceived difficulty from the two respective target values. Moreover, as anticipated in subsection 5.3.1, the dominating term in this score is the one pertaining to the Error Probability, since its weight w_{ep} has been chosen to be equal to 1 as opposed to the weight w_{pd}, which is instead equal to 0.2.

The proposed Optimality Score represents a function that can be used as the objective function for a suitable solver implementing a **minimization optimization**: in fact, the closer the trial's properties are to the respective target values, the closer the actual score is to zero. For reasons that will be clearer in the next paragraphs, it is better to also define the **Fitness F** for a generic trial t as

$$F_\Theta(t) = \frac{1}{S_\Theta(t)} \tag{5.2}$$

which is the inverse of the Optimality Score. As such, this function is suited for a maximizing optimizer, as optimal trials will be characterized by a high Fitness, and is exactly the objective function that will be used by the optimizing procedure chosen by the candidate. Moreover, the range of values that can be attained by this function are not reported, since it is simply going to be used as a comparison metric between different trials.

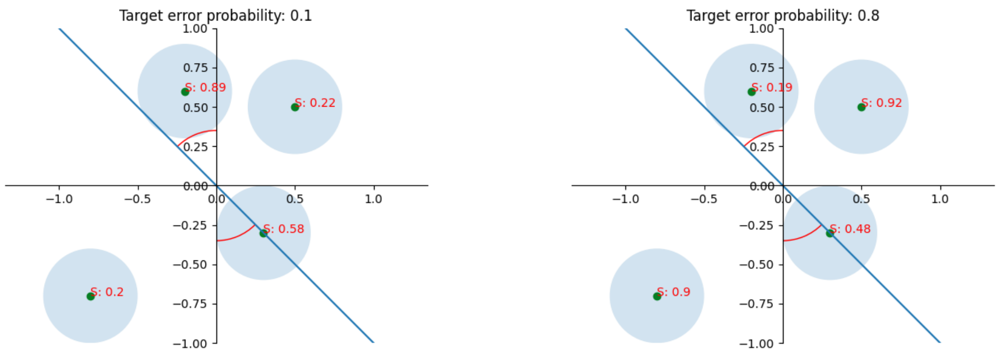


Figure 5.7: Optimality Scores of several trials for a player with α equal to 45° and σ equal to 0.3. On the left, the target error probability is set to 0.1, which leads to very congruent trials being more optimal. On the right, the target error probability is instead set to 0.8, which leads to the trial located between the decision boundary and the NND axis to have the lowest Optimatlity Score.

Performing the Optimization

Now that the Fitness function has been defined, the last thing that needs to be addressed regarding the proposition of the trials is how the ND-NND space is searched for local maxima of the Fitness given the set of parameters Θ. With this in mind, it's best to point out some observations first:

- **The solution is not unique**. There is, in principle, more than one trial in the space reaching the maximum value for the Fitness.

- **Diversity is a requirement**. As the Difficulty Controller keeps the same target error probability for several trial propositions, it is also important that the the optimization procedure doesn't always produce the same trial over and over when performed with respect to the same set of parameters Θ.

- **There is noise in the Θ parameters**. Θ in fact contains also the α and σ parameters estimated by the Player Estimator, which should be considered to not be perfectly accurate. Because of this, it's important to note that a perfect optimization will not necessarily lead to a perfect result.

- **The Fitness is not differentiable over the whole ND-NND space**. This is clear by simply considering the Error Probability, as it depends on the trial's correct answer which is in turn dependent from the sign of its ND.

These observations, together with the fact that a compromise has to be reached in terms of computational time, suggest that the goal of the optimization is not to find the trial corresponding to the global maximum of the Fitness, but rather to **find suitably good local optima**. The diversity requirement is also exceptionally important for the Player Estimator, as it needs to be provided with meaningful trial data in order to work properly. Moreover, the last observation suggest that gradient-based methodologies are not suited for solving this kind of optimization.

As foreshadowed by the chosen objective function's name (i.e. the Fitness), the candidate therefore identified a possible way to perform the optimization by using a **Genetic Algorithm (GA)**. As a meta-heuristic that makes no assumptions on its objective function, GAs in fact provide a good compromise in terms of quality of the solution and computational time, as well as a way to introduce randomness, and therefore diversity, during the search for the local maxima of the Fitness over different iterations.

A thorough description of GAs is not within the scope of this thesis, which is instead provided to the interested readers in the bibliography [30]. In any case, in the following is provided a basic description of how the proposed GA achieves the optimization procedure by maximizing the Fitness function, which was implemented by using the `pygad` python library:

1. At initialization time, as set of N "candidate solutions" is generated randomly at the beginning of the GA, with each solution representing a trial in terms of its ND-NND values. As part of the initialization for the first iteration, the Fitness is also computed for each of these trials.

2. At the beginning of the iteration, a subset of these candidate solutions is selected according their quality in terms of Fitness. This subset of trials will be referred to as "parent solutions".

3. The parent solutions are organized in pairs, and a "recombination operator" is applied to each one of these pairs: by taking as input both parents, the operator generates a new trial, named offspring, whose features are a recombination of the parents' ones (ex. it will be characterized by the first parent's ND and by the second one's NND). The parents are regrouped and recombined until a set of offspring trials of size M has been generated.

4. The offspring trials undergo a "mutation operator" with a given probability p equal to 0.1: the operator takes as input the single offspring trial and performs a random change to either its ND or NND value (ex. by adding a random value).

5. The Fitness is calculated for all the generated offspring trials, and they are collected together with the parent solutions into a single set of N+M trials.

6. A "selection operator" is applied to this newly formed set of trials: the top N trials with the highest Fitness are kept as the "candidate solutions" for the next iteration, while the M trials with the lowest Fitness are discarded.

7. Step 2 to 6 are then repeated for a fixed number of iterations K.

8. Finally, the trial with the highest Fitness among the candidate solutions produced after the last iteration is provided as the final answer of the optimization process.

As a trade-off between computational time and quality of the generated trials, the values N, M and K were chosen to be equal to 6,4 and 10 respectively. Moreover, the search space has been limited to the subspace of the ND-NND plane characterized by a positive NND value: in order to generate trials also with a negative NND, a simple mirroring transformation with respect to the ND-NND origin is performed to the generated trial with a probability equal to 0.5, as it preserves the Fitness of the transformed trial.

Some examples of trials generated by this procedure with varying target error probability are shown in figure 5.8: as it's possible to see, the GA can find several different trials with reasonably similar properties, while also adapting to different values for the player's α and σ parameters.

With these last bits of information, the role of the Trial Proposer has been thoroughly discussed. Since the entirety of its functionalities depends on the α and σ estimation performed by the Player Estimator, it may seem that the Trial Proposer completely depends on its result. As it will be shown however, the interdependence of these two components is not asymmetric: the Player Estimator does not simply provide its results to the Trial Proposer, but it's also dependent on the data generated by it with the help of the Difficulty Controller. Proceeding the

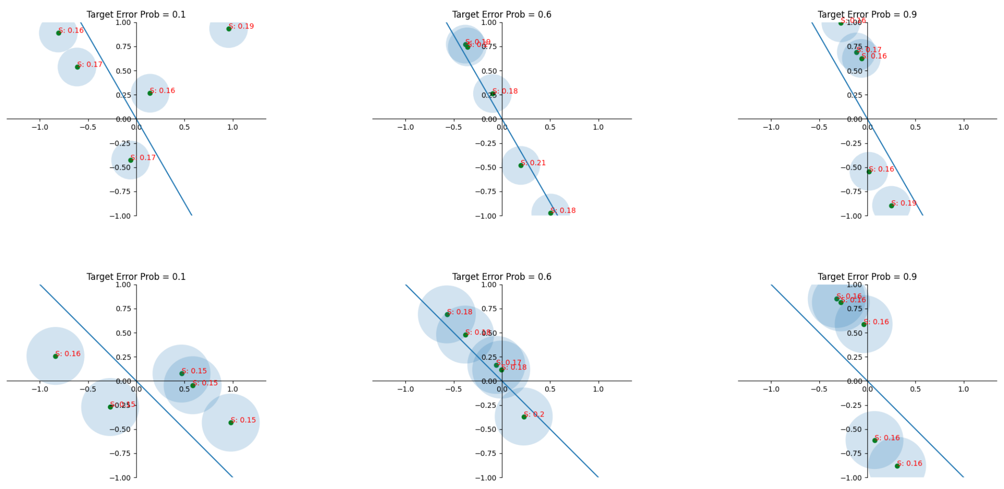


Figure 5.8: Trials generated by varying the set of parameters Θ. From top to bottom, the α and σ parameters increase from $\{30°, 0.2\}$ to $\{45°, 0.3\}$. From left to right, the target error probability increases from 0.1 to 0.6 to 0.9. The trials generated by the GA are both diverse while maintaining the same difficulty properties.

discussion of the `PDEP_Evaluator` however, it is first necessary to explain the task performed by the next component of the AI's workflow, which is the Trial Adapter.

5.4 Interacting with the Client

In the previous section it was made clear that, although the trials can be represented according to different sets of features, the AI treats trials simply as points within the ND-NND space. In fact, that is the format in which they are generated at the end of the Trial Proposer's task, and as will be shown in the section regarding the Player Estimator, the same counts even when the response to the trial is received and processed by the AI. By recalling the concepts introduced in chapter 4, the two other representations of the trials used by the whole application are:

1. **The set of $\{number, ISA, FA\}$ for each of the two fences**. This is the representation from which the actual ND and NND feature depend, as explained in detail in appendix B.

2. **The set of $\{number, SizeofAnimals, CircleRadius, AverageSpaceBetween\}$ for each of the two fences**. This is the set that can be interpreted by the client side and is directly related to the set of representation 1.

With that being said, the Trial Adapter is simply the component that takes care of finding a trial in accordance to the first representation that is suitably close the input trial generated by the Trial Proposer, which is instead defined just by its ND and NND values. This can be viewed as a **dimensionality expansion task**, starting from a two dimensional space and ending in a six-dimensional one: as there are infinite trials that can reproduce the desired ND and NND values, the Trial Adapter simply has to select one of them according to some possible additional criteria. Once that is done, the rest of the server's code will perform the translation of said trial in the second representation, and will later forward it to the client so that it can be shown to the player.

5.4.1 Selecting a suitable trial

As explained in subsection 4.3.2, the AI cannot just propose any trial to the client, but rather has to be careful in selecting only those ones that can be shown within it. Moreover, a sufficiently big subset of these trials has already been stated to be present inside the Lookup Table, with each one of its entries representing a single admissible trial. In the following, all the information contained in these entries is reported:

1. The set of $\{number, ISA, FA\}$ for the trial's left fence.
2. The set of $\{number, ISA, FA\}$ for the trial's right fence.
3. The filtering and sharpening difficulty coefficients used by the SimpleEvaluator, along with other related parameters.
4. The ND and NND values for the trial, computed from the parameters of the two fences as described in appendix B.

Obviously, the `PDEP_Evaluator` makes no use of the two difficulty coefficients, as it was developed specifically to overcome their limitations. On the other hand, the Trial Adapter can exploit the ND and NND information to meet two of its requirements at the same time: by finding an entry whose ND and NND values are close to the input ones, the Trial Adapter is immediately made available of one of their possible realization in the target six-dimensional space; moreover, by being selected from the Lookup Table itself, such trial will also be admissible for being shown to the client.

The only thing that is left to do is defining a criterion for selecting an entry based on the ND and NND of the input trial. As it also determines a vicinity in terms of the Error Probability for the selected trial, the chosen entry is the one with the smallest **Euclidean Distance** in the ND-NND space between its trial and the input one.

The implementation is done through the functionalities made available by the `pandas` python library, as it allows to efficiently manipulate large CSV files like the Lookup Table. In detail:

1. At initialization time, the Trial Adapter is told which version of the Lookup Table needs to be used, which is later read and stored in a DataFrame object allowing its management.

2. At each mini-game iteration, the DataFrame computes the Euclidean Distance between each entry's ND and NND values with respect to the input trial.

3. The DataFrame sorts the whole Lookup Table in ascending order according to said distance.

4. By cycling through the sorted Lookup Table, the Trial Adapter selects as its final proposition the first trial that passes a set of additional controls.

This means that the Trial Adapter will try to find the closest trial in the Lookup Table in a best-effort fashion: in principle, it can't be guaranteed that the Trial Adapter will find trials that are extremely close in the ND-NND space to the target one. This represents a limitation that can't be completely removed from the application without changing the client, as shown in figure 5.9.

5.4.2 Additional Controls and Observations

In order to complete the description of the Trial Adapter, it is also necessary to mention what other checks are performed on the final trial, so as to complement the aforementioned distance criterion.

First and foremost, the Euclidean distance alone can bring to an important problem: in the case where the Trial Proposer gives an input trial with a ND value close to 0, the Trial Adapter may possibly find as its closest trial in the Lookup Table an entry with a different sign for its ND value. In this case, the Trial Proposer's intention to propose a trial with a certain Error Probability would be completely averted, as the change in the sign of the ND value implies a change in the actual correct answer. As a mitigation for this, the Trial Adapter doesn't just choose the closest entry in the ND-NND space, but it first discards any entry with a different sign for its ND value.

Other than this, the Trial Adapter also takes precautions related to the user experience, which are mainly related to additional controls to explicitly avoid reproposing the same entry within the same gaming session. Moreover, the component manages which one of the two Lookup Tables mentioned in subsection 4.3.2 needs to be used, by properly communicating with the rest of the server's code.

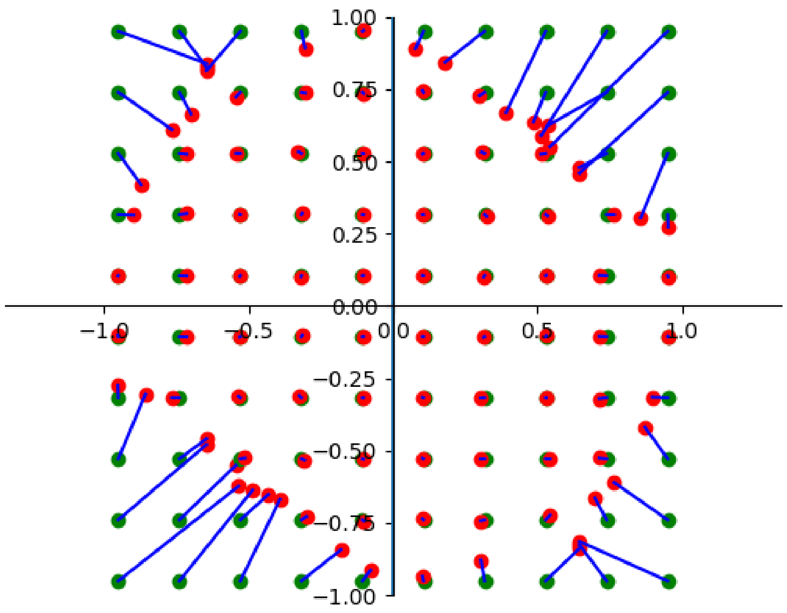


Figure 5.9: The coverage of the normalized ND-NND space attained by the trials present in the Lookup Table. The green points represent possible input trials of the Trial Adapter, while the red ones represent its related output. The output trials are linked to the respective input ones through the displayed blue lines. As shown in the figure, the Trial Adapter can reasonably accommodate for a wide range of input trials, but struggles when these are strongly congruent: this is because the client's limitations don't allow to show these kind of trials, as sets of animals with too high non-numerical features tend to occupy more space and can be shown only in a limited amount within a fence

.

As it is not the most important component of the `PDEP_Evaluator` and it is mainly based on concepts already explained in chapter 4, this is basically all there is to know about the Trial Adapter. Some more words can be spent however on its future developments: as this component has direct access to the numerical and non-numerical features of the trials, this is likely where a possible optimization for the target perceived difficulty will take place in the future; moreover, a more fine-grained control on the maximum value for the numerical features of the proposed trials will be probably added, as this is currently done manually in the main script by specifying which Lookup Table needs to be used.

With these last words, it is time to move on with the next section, which will conclude this chapter with the discussion of one the most important components of the AI: the Player Estimator.

5.5 How to Evaluate the Player

Even though the Player Estimator is the last component in the `PDEP_Evaluator`'s pipeline, it is definitely one of the most important parts of the whole AI. In section 5.1 it's stated that this component's task consists in the estimation of the player's α and σ parameters at every iteration of the mini-game, while in sections 5.2 and 5.3 it was anticipated that in order to do so, the Player Estimator uses the trials proposed during the playthrough together with the player's responses. The basic idea behind this component is therefore to understand the player's parameters at different points in time, which is done both to propose more balanced trials in the subsequent iterations and also to understand whether the player shows signs of improvements over time according to the sharpening and filtering hypotheses.

Some more remarks were made to the fact that the Difficulty Controller and the Trial Proposer were designed also in order to generate meaningful trial data for this component to work properly: in the following paragraphs, the general procedure for estimating α will be explained, which will make this statement clearer; following such explanation, an overview of the estimation procedure of σ will be carried out, which will be shown to depend on α's estimation. Given this need of trial data in order to work, at the beginning the Player Estimator returns a default value for both α and σ for a certain amount of mini-game iterations, so that the Trial Proposer can still reasonably perform its task without being provided with actual estimated values. After these iterations are over, the Player Estimator will then start executing the estimation procedures from the trial data produced so far.

Moreover, a distinction has to be made in the estimation procedure, which comes in two versions:

- The first one, named **first pass estimation**, is a version that is only partially addressing the possibility of an improving player.
- The second version, named **second pass estimation**, instead is also adaptive to the player's learning rate and it is based on the first pass estimation's result.

As both these versions share a lot in common, the general procedure for the first pass estimation will be explained in the next subsections, while this section's last part will be dedicated to the additions brought by the second pass estimation.

5.5.1 Estimating the Alpha angle

The first part addressed by the Player Estimator during the first pass estimation is α, the parameter derived from the filtering hypothesis that represents the influence of non-numerical features in the player's decision. As shown in previous sections, this parameter is strictly related to the player's decision boundary, which is assumed to be a line passing through the ND-NND origin, and numerically represents the angle between such boundary and the NND axis within the ND-NND space. Following this definition, the problem at hand can be interpreted under a new light: rather than trying to explicitly estimate the α angle, it can be more convenient to estimate the boundary itself instead, similarly to how the procedure was carried out in the paper Learning to Focus on Number. Before moving on, it is worth to make some observations:

- The trial data produced over several mini-game iterations can be seen as a dataset of pairs $\{t_i, y_i\}$, where t_i represents a feature vector composed of the ND and NND value of the ith proposed trial, while y_i represents the player's answer, which is a label representing either "left" or "right".
- The aforementioned dataset contains data that, due to the design of the Difficulty Controller and the Trial Proposer, is centered at the origin and reasonably spans the whole ND-NND space.
- The player is assumed to be able to improve over time, which means that trial data that is too old may not necessarily be beneficial to the current iteration's estimation.

From these observations, it follows that the current problem can be solved through traditional **machine learning techniques**, since what needs to be done is basically solving a binary classification task with the additional tweak that **only a subset of the whole dataset can be used**. In particular, the first pass estimation's main goal is to not introduce any old polluted data to the actual dataset used for training, so the amount of data used at each iteration is limited to the last N trials proposed in previous ones, with N suitably small (i.e. 180 in the implementation). The dataset obtained from these N trials will be called D_N, and by definition, it varies at every mini-game iteration.

As for the machine learning technique used to model the player's decisions, the choice fell on **SVM** with a linear kernel: not only do linear SVMs perform explicitly the estimation of the required boundary, but they are also suitable for dealing with noise, which can be widely present in D_N proportionally to the player's actual σ parameter. As for the hyper-parameter C required for their functioning, an ablation study has been performed to generate a mapping between the pair $\{\alpha_{i-1}, \sigma_{i-1}\}$

found in the previous iteration and the optimal C leading to the best estimation, of which the details are discussed more in detail on appendix A.

The implementation was designed by using the `sklearn` python library, which provides access to classes implementing SVMs for classification tasks with a linear kernel. In order to force the related class into finding a separation boundary passing through the origin, the dataset D_N is first pre-processed through a mirroring augmentation transformation, which doubles the size of the dataset by adding, for each trial, its specular one with respect to the origin. Moreover, in the unlikely case that the estimation would produce a boundary defined by an angle outside of the $[0, 90]°$ interval, the value found would be clipped to be within such range. Figure 5.10 shows an example of boundary estimation, according to the procedure described just now.

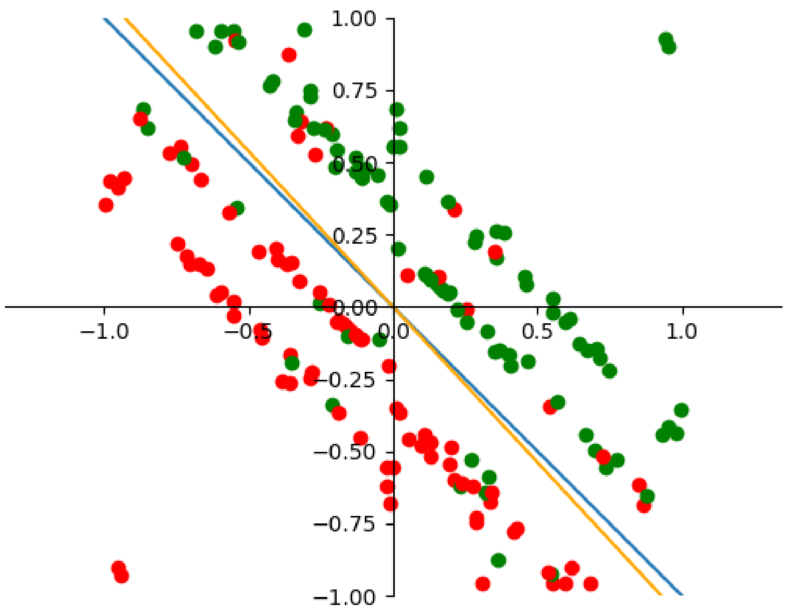


Figure 5.10: The SVM's estimation of the player's decision boundary. The data were generated from the Trial Proposer as if the player had a fixed α and σ equal to 45° and 0.2 respectively, while the labelling, represented as green dots for "right" and red ones for "left", was produced from a player model coherent with those parameters. The blue line represent the actual player's boundary, while the orange one is the boundary estimated by the SVM

.

Once the boundary has been estimated, it comes in the form of a vector starting at the origin, from which the related α angle can be easily derived through trigonometric formulas.

It is now clearer why the Difficulty Controller and the Trial Proposer had to provide diverse trials at proposition time: the presence of multiple trials with the same ND-NND values within D_N does not help the SVM algorithm, and it may even harm the estimation in the case where their labels are different due to the σ parameter, as that would bring further noise within the dataset without the advantage of leveraging from more data.

5.5.2 Estimating the Sigma standard deviation

The decision boundary is not the only factor composing the player's model as, from the beginning of this chapter, the σ parameter has also been identified as the main factor representing the sharpening hypothesis. Recalling its definition, the σ parameter is the standard deviation of the noise in the player's internal representation of the trial within the ND-NND space, and together with the player's decision boundary, it defines an Error Probability for each trial that represents how likely the player will answer incorrectly to it.

Looking back at the interpretation of the previous subsection, the σ parameter can be seen as a measure of the noise within the dataset D_N. So, how can this noise be estimated?

Generally, machine learning techniques are not aimed at estimating the noise within the data, but rather at filtering it out and produce models that are as least as possible affected by it. Therefore, instead of searching for a mathematical formula or for a library implementing this task for this specific problem, the candidate decided to design an ad hoc solution starting from some solid theoretical background. Therefore, given that

1. At this point the estimated decision boundary has already been computed.
2. There is a well established and optimized way to compute the Error Probability and its complementary for each trial.

the problem has been interpreted as a **Maximum Likelihood Estimation** for the σ parameter given the newly estimated separation boundary (identified by α) and the dataset D_N.

Defining the Likelihood function

As the knowledgeable reader would know, the Likelihood is a widely used term in statistics to define a specific kind of probability related to random variable realizations, which is also used in the field of machine learning to define certain kinds of model estimations. In the context of the proposed estimation procedure, the Likelihood L of a dataset D can be defined in its most generic formulation as

$$L(D|\alpha,\sigma) = P(D|\alpha,\sigma) \tag{5.3}$$

which can be read as "given α and σ, the Likelihood of a dataset D is equal to the **probability of observing D from a model defined by the parameters α and σ**". This is not to be intended as the probability of generating D itself but rather to generate the labels contained in D for each one of its elements. To have a clearer idea, it is possible to relate this formula to the dataset reserved for the first pass estimation D_N, which is recalled to be composed of the set of couples $\{t_i, y_i\}$ representing the ND and NND values of each proposed trial together with the player's labelling. In that case, by considering each y_i as independent from all the others, equation 5.3 can be written as

$$L(D_N|\alpha,\sigma) = P(Y_1 = y_1, ..., Y_n = y_n|\alpha,\sigma) \tag{5.4}$$

$$L(D_N|\alpha,\sigma) = \prod_{i=1}^{N} P(Y_i = y_i|\alpha,\sigma) \tag{5.5}$$

where Y_i is the random variable related to the player's prediction of trial t_i. This formula is already detailed enough to be computed by the Player Estimator, as each term of the product can be derived in a similar way to how the Error Probability is computed for each trial by the Trial Proposer. However, in order to describe more in detail the actual implementation in code, some more changes can be made to the equation. As a preliminary operation, the dataset D_N is divided into two parts, named D_c and D_w, of which a graphical representation is shown in figure 5.11: the former is the set of trials where the noise due to σ did not play a role in the label assigned by the player; the latter is instead the set of trials whose labelling was influenced also by the noise.

It's worth to point out that the separation of these two datasets is done according to the player's **estimated decision boundary** and that it is performed without taking into account the actual correctness of the labels in the comparison mini-game.

With that being said, equation 5.5 can now be written as

$$L(D_N|\alpha,\sigma) = \prod_{d_i}^{D_c}(erf_{|d_i|,\sigma}(0)) * \prod_{d_i}^{D_w}(1 - erf_{|d_i|,\sigma}(0)) \tag{5.6}$$

where d_i is the "signed distance" of the ith trial from the estimated decision boundary introduced in section 5.3.1 (which depends on α), and $erf_{|d_i|,\sigma}$ is defined

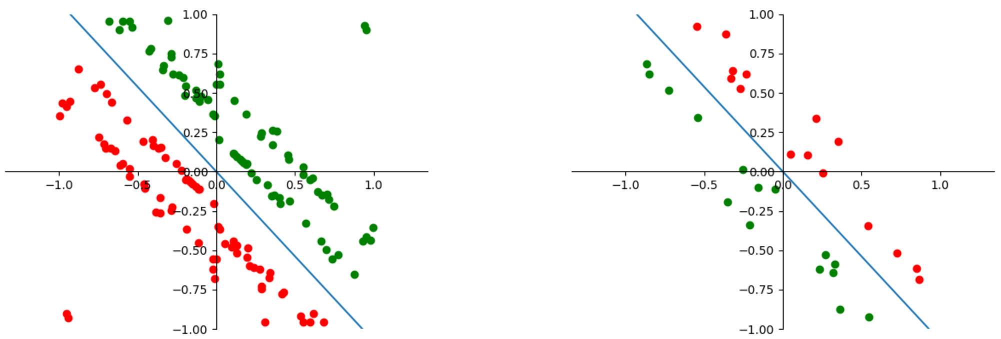


Figure 5.11: Examples of the D_c (left) and the D_w (right) datasets. Both of them were generated from the same dataset D_N displayed in figure 5.10, with the labelling following the same rules and with the blue line representing the estimated separation boundary. As shown in the right figure D_w collects all trials that, because of the noise due to σ, were labelled as "left" while being on the "right" side of the decision boundary and vice-versa. In the left instead is shown that D_c contains all trials that were seemingly labelled without the influence of the noise.

as the integral of a normal distribution with mean $|d_i|$ and standard deviation σ computed in the range $[x, +\inf]$.

Finally, as a general good practice to avoid numerical problems, the Likelihood is just turned into a Log-Likelihood defined as

$$l(D_N|\alpha,\sigma) = \ln L(D_N|\alpha,\sigma) = \sum_{d_i}^{D_c} \ln erf_{|d_i|,\sigma}(0) + \sum_{d_i}^{D_w} \ln 1 - erf_{|d_i|,\sigma}(0) \qquad (5.7)$$

which is the sum of the log-probabilities of each single trial's labelling and represents the final function that will be maximized by varying the parameter σ.

Maximizing the Log-Likelihood

Given equation 5.7 it is now time to design a way to perform the **Maximum Likelihood Estimation** for the parameter σ. In other words, by keeping α fixed to the estimated value in the previous subsection, this implies finding the value σ_e that maximizes the Log-Likelihood function for the datasets D_c and D_w. This can be written as

$$\sigma_e = \arg\max_{\sigma}(l(D_N|\alpha,\sigma)) = \arg\max_{\sigma}(\sum_{d_i}^{D_c} \ln erf_{|d_i|,\sigma}(0) + \sum_{d_i}^{D_w} \ln 1 - erf_{|d_i|,\sigma}(0)) \tag{5.8}$$

where σ_e is also the value for σ that is provided as the final estimation for the current iteration.

Equation 5.8 can probably be simplified further, but as the computation of the Log-Likelihood has been deemed to be sufficiently optimized, the candidate decided to perform the Maximum Likelihood Estimation as a simple linear search by generating an array of candidate values V_σ in the range $[0, 0.5]$ and by selecting the one that would attain the highest Log-Likelihood.

As a closure to the explanation of the first pass estimation, the whole procedure is summarized in pseudo-code in algorithm 1.

5.5.3 Handling a player that improves over time

Now that the first pass estimation has been explained, it is time to discuss what the **second pass estimation** does differently in order to adapt for the possible changes in the player's abilities in terms of α and σ.

When it comes to the algorithms used for the estimation of both parameters, the first and second pass estimations share a lot in common, as all the procedures that take care of that are exactly the same. The idea, however, is that the second pass estimation will be adaptive with respect to the player's improvements by using a **different different dataset**: while the first pass estimation uses D_N, which is the dataset composed of the trial data related to the previous N propositions, the second pass estimation will use a dataset D_z, which instead comprehends the **last z proposed trials**. As the second pass estimation has to be adaptive with respect to different players, z will therefore be a variable number depending on the player's estimated improvement rate for both α and σ.

The assumptions

Before going into the details of how this improvement rate is estimated, it's better to first explain how the player is assumed to improve over time.

As stated by the filtering and sharpening hypotheses, children are considered to be able to improve in their acuity in numerical representation and in the filtering of the non-numerical features in numerical comparison tasks. These improvements are supposed to happen thanks to time and practice, and have already been stated to be modelled as a decrease of the player's α and σ values over time; however, how the AI models the decrease of both these parameters was still not given a proper

Algorithm 1 The first pass estimation of α_i and σ_i performed by the Player Estimator at the ith iteration

```
 1: procedure ESTIMATION($D_N, i, \alpha_{i-1}, \sigma_{i-1}, V_\sigma$)
 2:     ▷ Mirroring Augmentation
 3:     $D_N \leftarrow MA(D_N)$
 4:
 5:     ▷ Find the optimal C for the SVM from $\alpha i - 1$ and $\sigma_{i-1}$
 6:     $C \leftarrow OC(\alpha_{i-1}, \sigma_{i-1})$
 7:
 8:     ▷ produce the SVM model
 9:     $model \leftarrow SVM(D_N, C)$
10:
11:     ▷ Use the model to produce $D_c$ and $D_w$ from $D_N$
12:     $D_{temp} \leftarrow classify(model, D_N)$
13:     $D_c, D_w = CS(D_N, D_{temp})$                    ▷ Compare and Separate
14:
15:     ▷ Extract $\alpha_i$
16:     $\alpha_i \leftarrow EX_\alpha(model)$
17:
18:     ▷ Find optimal $\sigma$
19:     $\sigma_{best} \leftarrow -1$
20:     $ll_{best} \leftarrow -\inf$
21:
22:     for $\sigma_{curr}$ in $V_\sigma$ do
23:         ▷ Find Log-Likelihood for current $\sigma_{curr}$
24:         $ll_{curr} \leftarrow l(D_c, D_w | \alpha_i, \sigma_{curr})$
25:
26:         ▷ Compare and swap
27:         if $ll_{curr} \geq ll_{best}$ then
28:             $ll_{best} \leftarrow ll_{curr}$
29:             $\sigma_{best} \leftarrow \sigma_{curr}$
30:         end if
31:     end for
32:
33:     $\sigma_i \leftarrow \sigma_{best}$
34:     return $\alpha_i, \sigma_i$
35: end procedure
```

explanation. The set of assumptions made by the AI to model such changes are therefore listed in the following:

- Other than possible temporary fluctuations, the player is assumed to improve following a linear trend, which can be characterized by a fixed value named **learning rate**.
- The player can improve independently according to the two hypotheses. This means that the AI will have to estimate two different learning rates, one related to α and one related to σ
- No explicit assumption is made on the "sign" of the learning rate, which means that the AI can handle also situations where the player gets worse in time.

In reality, it's more likely that a player may improve according to non-linear relationships that may be better described for example by exponential functions, and only until a certain limit. For simplicity however, the current version of the AI assumes that the player's α and σ over time can be described as a function of the kind

$$f(x) = max(ax + c, 0) \tag{5.9}$$

where c represent the starting value for the related parameter, a represents its learning rate and x is a quantity representing both the passage of time and the amount of practice done by the player.

Estimating the improvement rate

In equation 5.9, the meaning of x was left unclear. According to the filtering and sharpening hypotheses, an improvement has to be related to either the passage of time or the education received by the child. By design, the AI is made to be unaware of these two factors explicitly, but it can still model them through the **amount of mini-game iterations executed since the start of the playthrough**: the game is in fact meant to be played over a long period of time, and at each one of its iterations the player is effectively made to exercise by solving the related trial proposed by the AI. Moreover, as each iteration implies the proposal of a single trial, these two concepts will be used interchangeably to refer to this internal representation of the passage of time.

According to this interpretation, the learning rate will be measured internally as the increment or decrement of the related quantity per each mini-game iterations performed, expressed as either $degrees/it$ or U_{ND-NND}/it for α and σ respectively,where U_{ND-NND} represents the unit of measure of the ND-NND space.

The main idea on how AI estimates these learning rates is rather simple: since the Player Estimator already performs a first estimation of α and σ after every trial proposal, the result of the first pass estimation can be used as a general indicator of the learning rate for both parameters. This is done by performing two steps:

1. First, the first pass estimation is visualized in the value-iteration space, for each of the two parameters
2. Then, the AI fits a polynomial function of degree equal to one (i.e. a line) through all these points, for each parameter separately.

This is implemented through the `polyfit` function of the `numpy` python library, which returns the slope of the line in the value-iteration space that best describes the input data. Moreover, as the AI will need to compare the two estimated learning rates, this fitting is done with respect to the vector of **normalized estimated values** for both α and σ, obtained by dividing the actual data vectors by 90° and 0.5 respectively. By doing so, the procedure therefore provides two **normalized slopes that represent the normalized learning rates of the two parameters**.

This procedure implies that the results of the first pass estimation from the first to the current iteration have to be available; furthermore, leveraging from more of these kind of data will rightfully lead to a better estimation of the learning rates. This, together with the fact that the second pass estimation will be used for a different purpose than the first pass one, leads to the AI performing this procedure only when explicitly requested, rather than executing it automatically at every mini-game iteration.

Selecting the right z

At this point, the normalized learning rates for the player's α and σ parameters have already been estimated. What is left to do now is mapping both of these learning rates to the optimal amount z of trials to be used for the second pass estimation.

The first step to do so consists in understanding which of the two learning rates is steeper, as it would represent a more restrictive condition on the amount of usable trials: in fact, the same dataset D_z will be used for estimating both parameters just like in the first pass estimation. As the two learning rates extracted during the learning rate estimation come in a normalized form, this procedure is trivial.

Once the steeper learning rates has been identified, the last thing that needs to be done is **mapping such normalized learning rate to an optimal amount of trials z to be used for the estimations**: as a general idea, steeper learning rates will limit the value z, as trial data generated more further back in time is likely too old to be used; if instead the more restrictive learning rate has a low absolute value, the Player Estimator will allow the usage of more data for the estimation, thus leading to results that are much more accurate than the first pass' ones. The mapping of the normalized learning rate with the optimal amount of trials is discussed in appendix A, of which the main result is reported in figure 5.12

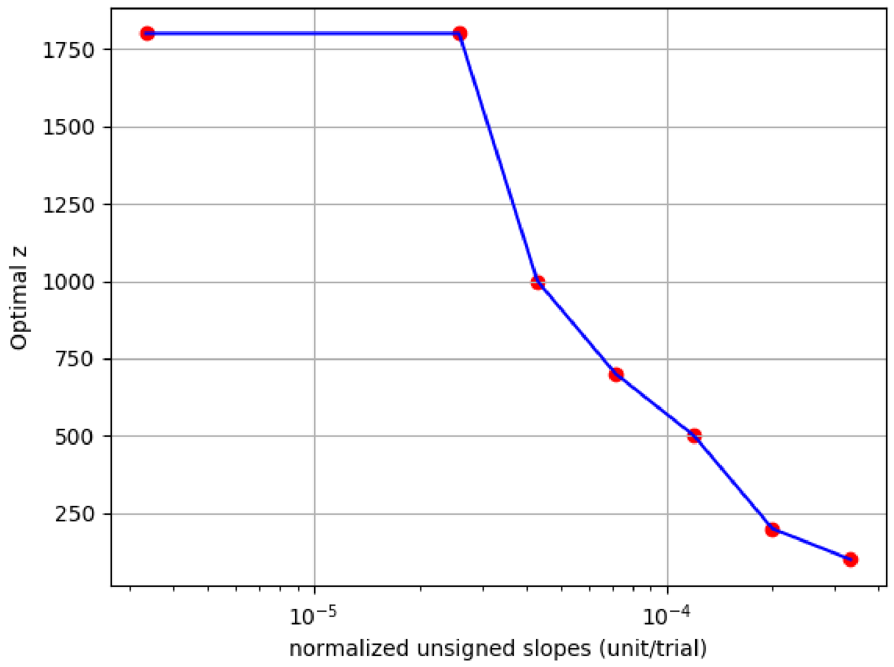


Figure 5.12: Optimal values of z for different normalized learning rates. Learning rates that are lower in absolute values (less steep) are mapped to higher values z, as the AI can afford to use older data without hindering the quality of the estimations. On the other hand, steeper learning rates force the Player Estimator to use less trials, as using data that is any older would mean introducing trials whose labelling is too incoherent with respect to the current player's parameters

.

How the second pass estimation is used

As already anticipated, the second pass estimation is more accurate at the later stages of the playthrough, when more data have been generated by the AI that can be used to perform the learning rate estimation. This, along with the fact that the learning rates have been assumed to be constant in time, somewhat incentivizes the usage of the second pass estimation only at the **later stages of the playthrough**.

In addition to this, the candidate empirically assessed that the result of the first pass estimation, although not as accurate as the second pass one's, still allows the Trial Proposer to guarantee the Error Probability requirements as dictated by the Difficulty Controller. As such, the actual α and σ parameters used as the input of Trial Proposer were kept to be the ones produced by the first pass estimation also in the later stages of development.

With that being said, the second pass estimation should not be seen as the main mechanism allowing the AI to work, but rather as the **main tool for understanding the player's abilities at different points in time**. For this reason, the second pass estimation is not performed automatically at every iteration, but is made available by the AI whenever it is requested by the player or by the simulation environment. In detail, whenever the request is made the Player Estimator will:

1. Estimate the learning rates by using **the entirety of the α and σ data** derived by the first pass estimation from the start of the playthrough until the current iteration, referred to as i_c.

2. Find the optimal amount of trial z_c to be used according to said estimation.

3. Perform the second pass estimation **for all iterations** $\{i_0, ...i_c\}$ by using z_c at every point in time.

This means that the AI will use the most accurate knowledge available about the player's learning rate to **retroactively reinterpret the entirety of the playthrough**.

With the introduction of these last concepts, all the aspects related to the theory and the implementation of the AI have been presented. It is therefore time to see how these methodologies perform in practice, which is the main topic of the next chapter together with the description of the simulation environment that is meant to aid this process.

Chapter 6

The Simulation Environment

Whenever an application like The Number Farm is in development, the correctness of its design and of its implementation is never sufficient to understand how well it will perform in a real use-case scenario. The way the application behaves when it is used by a real person, in fact, is not something that can be normally tested by the developer alone, and in the particular case of AI-related functionalities, it is not easy to understand whether the application is capable of doing what it was designed to do.

The client side and the related user experience had already been tested on a series of occasions, by holding usability tests in the laboratory room dedicated to the project. In these usability tests, several groups of children were invited in the University of Essex to play a certain amount of rounds of the mini-game with a dedicated mobile device: at the end of these tests, the candidate's predecessors and colleagues would ask the children to answer several questions regarding how the game felt to be played and other aspects related to the user experience. As it is not the main topic of this thesis, the client side can be therefore considered to be sufficiently good in regards of the user interaction.

When it comes to the candidate's area of responsibility however, the discussion of this thesis cannot lack an in depth description of the additional work required for testing its capabilities. While in general the server's code is not very "intelligent" and requires only minimal testing regarding the correctness of its implementation, the same can't be said with respect to the AI, as it came in a new version developed by the candidate. So, **how can the AI's performances be assessed?**

In the field of machine learning and deep learning, the usual approach is to design suitable procedures that test the generated models on a dedicated dataset, that generally contains data related to the task that needs to be solved. In this thesis' activity however, no such data was provided, regardless of whether it could be used for testing purposes. Moreover, it is needless to say that using real children for testing the AI's performance would be highly impractical, as the game is meant to be played for long periods of time. There is therefore only one way to perform a thorough assessment of the AI, which was already employed by previous applications as mentioned in chapter 3: given a certain mathematical model of the player, a suitable component that simulates their behavior has to be implemented; after that, this simulator is made to interact with the AI in several different simulations; finally, from the results of these simulations, a first assessment of the AI's performances can be performed.

This chapter is dedicated to the Simulation Environment developed by the candidate in order to do so. The chapter will begin with the discussion of the **Child Simulator**, which is the component taking care of emulating a child playing the game; following that, a brief explanation of the environment connecting the Child Simulator with the `PDEP_Evaluator` will be carried out; finally, this chapter will find its closure with an overall discussion of how the AI behaves when in contact with the Child Simulator.

6.1 Simulating the Behavior of a Child

Section 5.1 already contains a thorough explanation of the contributions of Learning to Focus on Number in the design of the AI. In particular, the section defines the trial's ND-NND space in the context of the application and proposes a model for the player, whose main principles are briefly recalled in the following:

1. The player is a classifier that takes trials in the ND-NND space as input and outputs a label for each of them, that can be either "right" or "left" and represents the player's perception of which fence contains more animals.

2. This classifier is characterized by a decision boundary in the form of a line passing through the ND-NND origin, and labels the input trials based on the sign of their projection with respect to the boundary's normal vector $\vec{n}$. Moreover, the boundary is defined by its angle with respect to the NND axis, named α, that can assume a value in the range $[0, 90]°$.

3. Together with the boundary, the player's model is also characterized by a noise in their representation of the trials within the ND-NND space. The noise is assumed to have a Gaussian distribution, centered at the input trial and with a uniform standard deviation along both axes, named σ.

4. With the passage of time and education, which are represented internally as the number of mini-game iterations done by the player, both of these parameters are supposed to decrease with a trend that is assumed to be linear by the AI. The magnitude of such decrease is named **learning rate**, which is

distinct for both α and σ, and is defined the slope of the line representing the parameter over the course of the aforementioned iterations.

Keeping in mind that all these principles are coherent with the theory of Learning to Focus on Number, they can be used to come up with a component that **simulates exactly this player model**: if the actual children do follow the sharpening and filtering hypotheses when playing the game, then a simulator implementing these behaviors can be taken as a reasonably good component for testing the AI's capabilities.

Therefore, the **Child Simulator** was developed by following these assumptions to the letter. In detail:

- At instantiation time, the Child Simulator is provided with four fundamental parameters: the **starting** α, the **starting** σ and the **learning rates** related to both of them.

- In its initialization method, the objects taking care of the parameter's improvements are instantiated, while the α parameter is processed in order to extract a matrix, that will be referred to as M. By naming the unit vector in the second quadrant forming an α angle with the NND axis as $\vec{b}$, and by defining its perpendicular vector in the first quadrant as $\vec{n}$, M represents the transformation matrix mapping trials from the ND-NND space to the new set of coordinates whose axes are defined by the vectors $\{\vec{n}, \vec{b}, \}$

- Whenever the Simulator is asked to play an iteration of the mini-game, the entry in the Lookup Table corresponding to the proposed trial has to be provided as input.

- After that is done, the Child Simulator extracts the trial's ND and NND values from the entry and stores them into a vector. Moreover, it applies the noise related to σ to such vector, thus obtaining an array representing the **noised trial**: for simplicity, this is implemented with the addition of an another vector, whose two values are extracted from the same normal distribution $\mathcal{N}(0, \sigma)$.

- Then, the vector representing the noised trial undergoes a dot product with respect to M: from a mathematical point of view, the second value of the resulting vector represents the component with respect to $\vec{n}$, i.e. the signed distance d of the noised trial with respect to the boundary.

- Finally, the Child Simulator returns its prediction depending on d: if positive, it returns the label corresponding to a "right" prediction; if negative, the "left" label is returned. Once that is done, the simulator performs a change in the α

and σ parameters according to their learning rates and recomputes M, after which it is ready for a new mini-game iteration.

In figure 6.1 are shown examples of predictions performed by simulators defined by different α and σ parameters. As for the improvement of parameters, an example is shown in figure 6.2

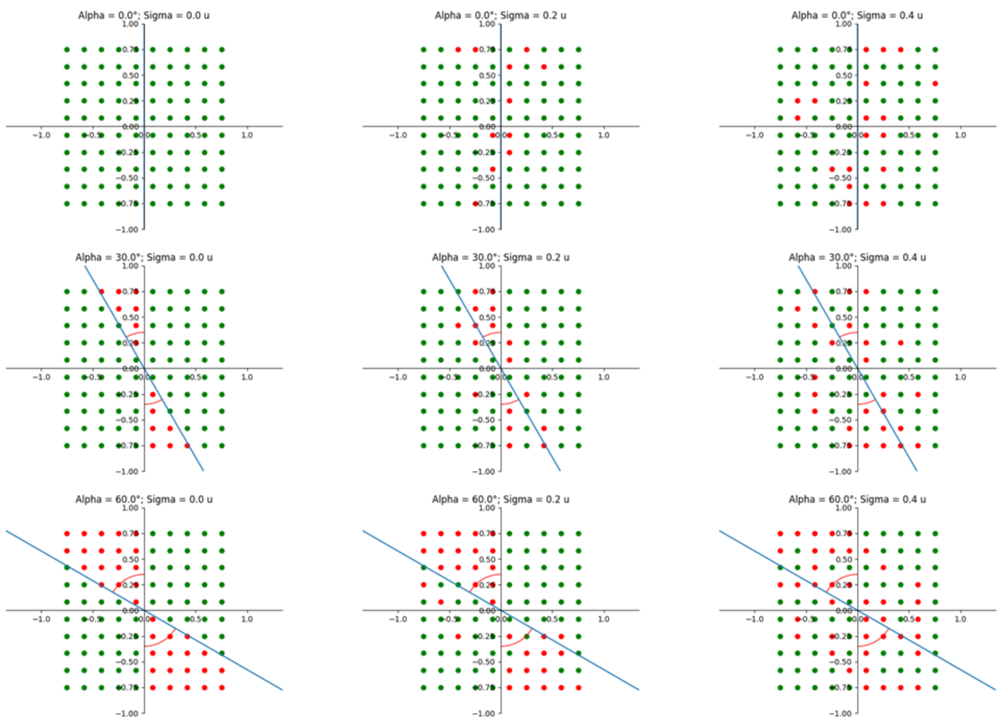


Figure 6.1: Simulator's responses with varying α and σ. Green dots represent trials with correct answers, while red dots represent wrong answers

.

According to the explained procedure, the Child Simulator takes as input a whole Lookup Table entry, comprised of the two fences' $\{number, ISA, FA\}$, but only makes use of the ND and NND values provided: this is because, for now, the AI does not see beyond the trial's ND-NND values. As a possible future development, the Child Simulator can also be made to model the **time required for answering**, which will be dependent on other details besides the ND-NND values, and will be possibly used by the AI to regulate the Perceived Difficulty of the proposed trials.

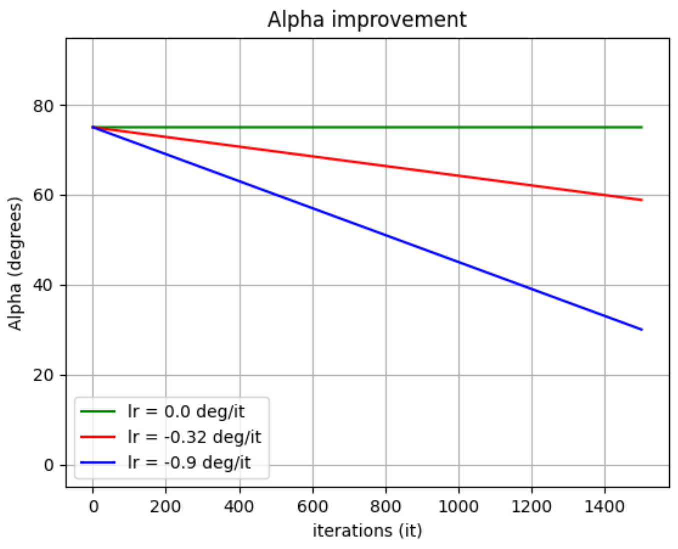


Figure 6.2: The plot of the α parameter for three Child Simulators with different learning rates. As in the internal AI's representation, the learning rate is represented in terms of degrees per performed iteration, and is mathematically equal to the slope of the line in the degree-iteration space for the attained values of α

.

6.2 A Tool to Evaluate the AI

Having defined the Child Simulator, it is now time to describe the environment in which it can interact with the `PDEP_Evaluator`. As will be shown in the following, such simulation environment was designed in order to properly assess the AI's performances, both by controlling the interactions between the two components and by producing all required data regarding each performed simulation.

6.2.1 General Aspects

The Number Farm, as recalled multiple times, is a game that is meant to be played for long period of times. As in the case of Calcularis, in fact, each player is supposed to exercise on a daily basis through the application for several weeks or months; differently from such game however, The Number Farm is a much smaller game and is meant to be played for about 4-5 minutes per day, since it only contains a single mini-game for now.

This means that, in order to extensively assess the behavior of the AI, the simulation environment has to condense the equivalent of several weeks of play-time in a much more reasonable fraction of time, so as to allow the developer to test the

AI under several conditions. This calls for the need to detach the `PDEP_Evaluator`'s code from the client side, and even from the rest of the server code, as attempting to access it indirectly from the application's regular working mode would increase significantly the time taken to perform each simulation.

The candidate's predecessors had already developed a first simulation environment doing something similar, although it was merely done to visualize trials that were proposed in a completely random way. As such implementation was even older than the first version of the SimpleEvaluator, the candidate decided to implement a new simulation environment from scratch that was tailored to work around the `PDEP_Evaluator`.

As such, the **Simulations Runner** was developed: this component, accessible from a dedicated script (simulate_main.py), takes as input more than 25 parameters and contains a method that manages simulations of playthroughs in their entirety, which are referred to as **"Player Cycles"**. Among these parameters, the most important are:

1. The starting α and σ for the Child Simulator that will be instantiated for the simulation.
2. The learning rates for both the aforementioned parameters.
3. The length of the simulation expressed in **days**.
4. The number of trials, and therefore of mini-game iterations, played by the Child Simulator each day.
5. The name of the simulation and of the folder in the local file system where the simulation's results will be stored.
6. The difficulty calibration that, as stated in section 5.2, can be regulated so that the player achieves an overall accuracy either equal to 70% or 50%

As the actual passing of time is not relevant for the AI, the number of trials per day is generally kept at a fixed value of 30, which stands for a daily play time close to the desired 4 minutes, while the actual amount of mini-game iterations is controlled through the number of days of the simulation. For the sake of simplicity, it is therefore possible to represent each simulation from the set of its main properties: in principle, the notation $\{\alpha_0, \sigma_0, lr_\alpha, lr_\sigma, N_{days}\}$ can be used to quickly define the main parameters of each described simulation; however, in the case where the two learning rates are equivalent in their normalized form, an even more compact notation can be derived where lr_α and lr_σ are condensed into a single parameter $\tilde{lr_n}$ representing the normalized **daily** learning rate of both α and σ. As the latter situation will occur much more often in the rest of this thesis, its related compact notation will be used to quickly indicate the main characteristic of the various performed simulations. It's worth noting that the difficulty calibration is not considered as a main parameter, as it will be be kept as to achieve an accuracy equal to 70% unless stated otherwise.

When required, the Simulations Runner also allows to access low-level properties of the `PDEP_Evaluator`'s components (ex. the Difficulty Component), in order to better understand the performances of each them taken individually: In those cases, the objective of the simulation will be described explicitly.

Finally, the Simulations Runner also contains the procedures related to the ablations studies that allow the Player Estimator to work under optimal conditions, as they use the results of several simulations in order to arrive at their results. As this section will focus solely on the simulations themselves, the discussion of the ablation studies is postponed to appendix A.

6.2.2 Collected Information

As stated before, the Simulations Runner takes as input also the name of the folder where the simulations' results will be stored. In fact, the mole of data produced by each simulation is considerably large, and is not suitable for just being displayed at run-time. This subsection is dedicated to describing all the information collected by the simulations, that were selected in order to provide the developer with all relevant data without the need to re-run the related player cycles every time.

The Proposed Trials

Every certain amount of simulation days, the Simulations Runner takes a snapshot of the trials proposed during said day, and saves a plot of them as shown in figure 6.3.

These plots provide a graphical representation of the trials proposed within the simulation, as well as how close the estimated values are to the Child Simulator's ones. Moreover, the Simulations Runner also produces a 3D representation of both these informations, as shown in figure 6.4

The Parameters over Time

All the parameters related to the Child Simulator and to the AI's estimations are also shown in suitable plots, where the actual simulator's values are compared with respect to the ones produced by the **first pass estimation** and by the **second pass estimation**. Plots are generated in two versions, one with the single gaming-iteration as the x axis (figure 6.5), and one for better readability where the x axis represents the days of the simulation (figure 6.6). In the latter case, the plotted curves are computed as the average value found within the day.

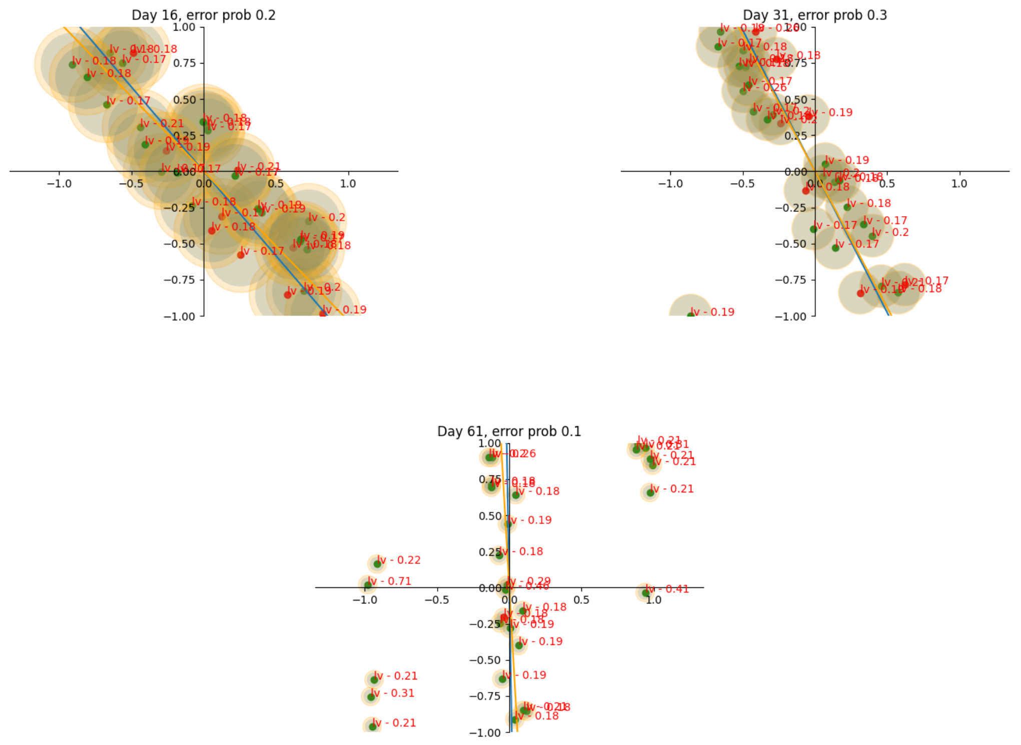


Figure 6.3: Snapshots of the trials proposed at several days for a simulation of the kind $\{55°, 0.3, -0.01, 65\}$. Red dots represent trials labelled incorrectly by the Child Simulator, while the green ones represent the correct answers. The figure shows also the decision boundary and the σ noise of both the Child Simulator (in blue) and the AI's estimation (orange). Both of the displayed values together with the target error probability in the title are snapshotted at the beginning of the day, so the trials displayed may have been generated according to different combination of these values. The red annotations for each trial instead display value for the Optimality Score computed at trial generation time by the Trial Proposer

All the values required to generate these plots are also saved as .npy files, so that additional studies can be done with the actual data at hand (ex. comparisons between several simulations).

Accuracy and Others

As it's vital to get a concrete assessment of the Trial Proposer's performances, plots of the accuracy are also generated by the Simulations Runner, showing both the

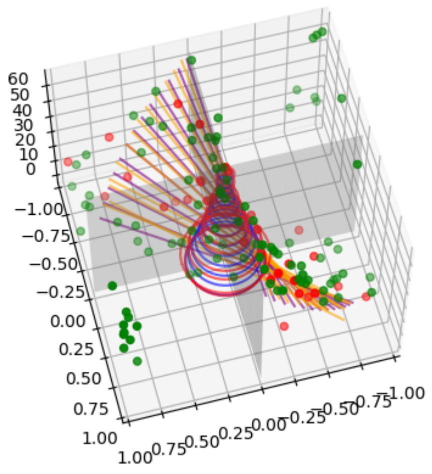


Figure 6.4: 3D plot summarizing the same simulation of figure 6.3. The plot represents a subset of the proposed trials and of the estimated and actual values for the α and σ parameters over time (represented in the z axis). The plot can be viewed at run-time from different angles

.

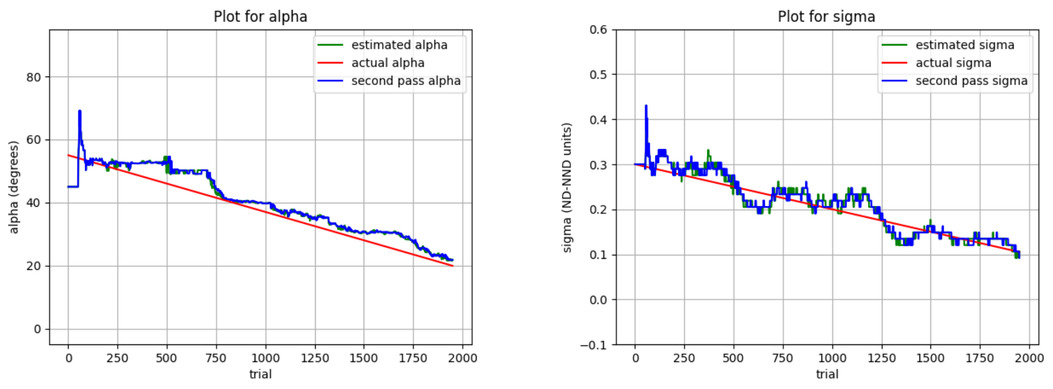


Figure 6.5: Plots of the simulator's α and σ along with the first and second pass estimation values with iteration granularity. In the figure, the simulation was defined by the set $\{55°, 0.3, -0.006, 65\}$

accuracy attained by the Child Simulator within the day as well as the cumulative accuracy measured since the start of the player cycle. Two examples are shown in

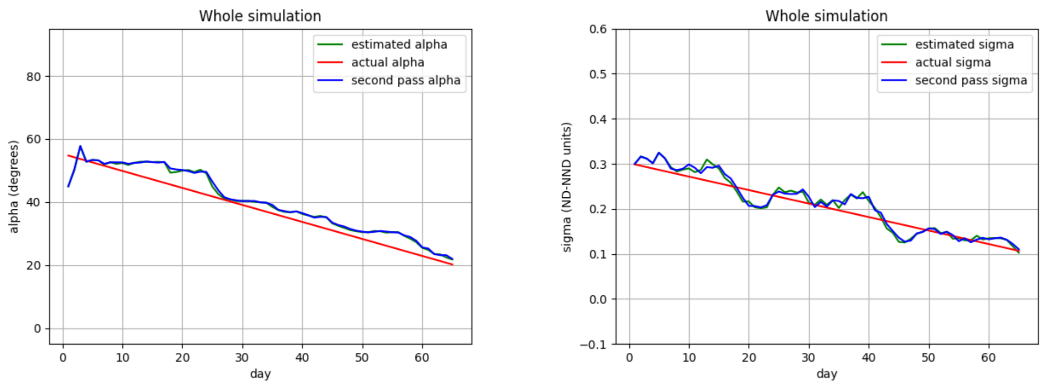


Figure 6.6: Plots for the same simulation of figure 6.5, but plotted against each simulation's day

figure 6.7

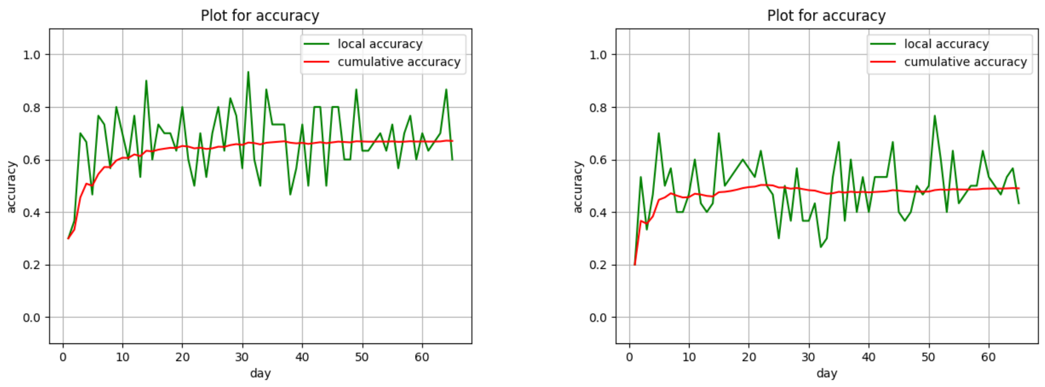


Figure 6.7: Accuracy plots for two simulations with different difficulty calibration. On the left is a simulation calibrated to achieve 70% accuracy, while on the right is shown the plot for a simulation with a sightly higher difficulty aimed at reaching 50% accuracy. It's worth noting that the local accuracy (i.e. the accuracy achieved during the day) is a rather inconsistent value: this comes from a design choice, as the Difficulty Controller changes the target error probability in a way that doesn't average within the single day to the overall desired value.

Finally, the simulation environment also produces tables summarizing the most important aspects of the simulations, that are stored directly in LaTeXformat. As shown in table 6.1, the tables include several aggregated statistics that are computed on a monthly basis as well as for the whole simulation. These tables allow for easy comparisons of the two estimation modes, namely the first pass estimation and the second pass estimation.

Month	FP Avg Dist	FP Dist Std	FP Max Dist	SP Avg Dist	SP Dist Std	SP Max Dist
1	8.49	15.62	73.47	3.95	7.66	35
2	3.63	2.4	16.21	0.53	0.24	1.77
3	4.11	2.42	10.07	0.29	0.36	1.7
4	3.3	2.96	11.8	1.14	0.89	2.77
5	5.16	2.98	15.57	1.84	0.5	2.81
6	4.4	2.85	14.15	1.11	0.5	2.81
All	4.85	7.07	73.47	1.48	3.39	35

Table 6.1: Example of a produced table for the estimation of alpha. All values are expressed in degrees, while "FP" and "SP" stand for first pass estimation and second pass estimation values respectively. For each of the two, the distance of the estimated values with respect to the simulator's ones is measured as the absolute value of their difference, and is analized in terms of average value, standard deviation and maximum value.

Now that the simulation environment and the Child Simulator have been discussed, it's now time to put them to use together with the `PDEP_Evaluator`. In the next section, a thorough discussion of the AI's performances will be therefore carried out, all while analyzing its performances with respect to different Child Simulators.

6.3 Results and Discussion

While the design principles of the `PDEP_Evaluator` have been explained in chapter 5, no statements about its actual performances have been made. Having described of the tools used for assessing the AI, the rest of this chapter will be dedicated to a complete analysis of the `PDEP_Evaluator` in terms of its ability to propose balanced trials and of the accuracy in its estimated values. The assessment will be carried out by analyzing the `PDEP_Evaluator`'s main components singularly as well as the AI as a whole, all done through the use of the Simulations Runner with varying configurations regarding the Child Simulator's starting α, σ and its learning rates.

6.3.1 The consistency of the Trial Proposer

The most important components of the `PDEP_Evaluator` are definitely the Trial Proposer and the Player Estimator, which have already been stated to be interdependent. However, even if the Trial Proposer cannot perform its task without the estimated α and σ values, it's rather easy to detach this component from the rest of the AI: all that needs to be done by the Simulations Runner is providing it with "fake" values for all its inputs, which allows to assess the quality of its proposed trials regardless of the rest of the AI.

The main idea behind the following analysis therefore consists in providing the Trial Proposer with the Child Simulator's real α and σ and in keeping fixed the value for the **target error probability for the whole player cycle**: by doing so, it is possible to understand whether the Trial Proposer can keep proposing trials so that the Child Simulator **fails in accordance to the selected probability**. In order to not introduce any influence from other parts of the AI, the trials will also be forwarded to the Child Simulator without passing through the Trial Adapter, as this would introduce limitations on the final generated trial.

The performed simulations will not be required to be particularly long, and as the Trial Proposer does not store any information about previous mini-game iterations, the learning rates of the Child Simulator can also be kept at 0 for simplicity. The set of simulations performed for this analysis will therefore be of the kind $\{\alpha_0, \sigma_0, 0.0, 30\}$, and will be performed by exploring different values for the target error probability.

Figure 6.8 shows the accuracy plots from four simulations characterized by different α_0 and σ_0 and by keeping the target error probability fixed at 0.1.

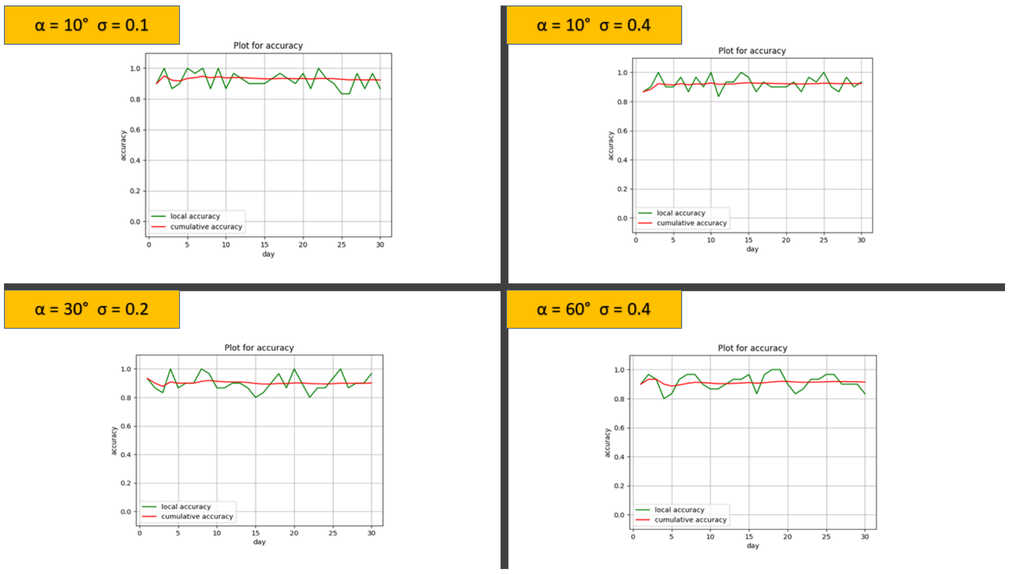


Figure 6.8: Accuracy plots for different simulation with target error probability fixed at 0.1

.

In reality, many more simulations were performed, but regardless of the different starting parameters, they all demonstrated the same trend: the Trial Proposer is fully capable of generating trials with a 10% error probability, which leads to an overall accuracy of the Child Simulator that very quickly converges to 90%. A similar trend is shown for simulations that have a fixed target error probability equal to 0.3%, as shown by figure 6.9.

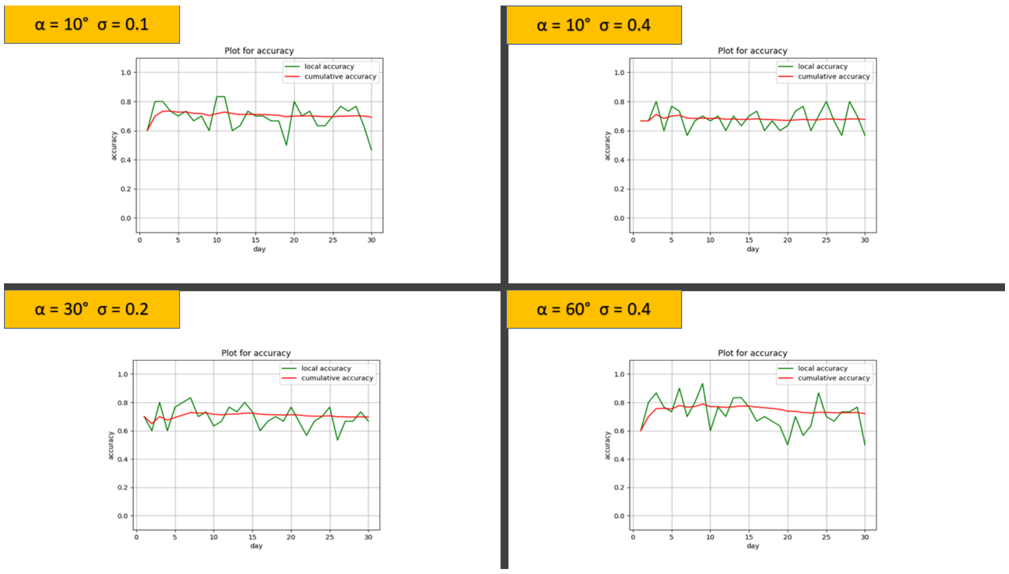


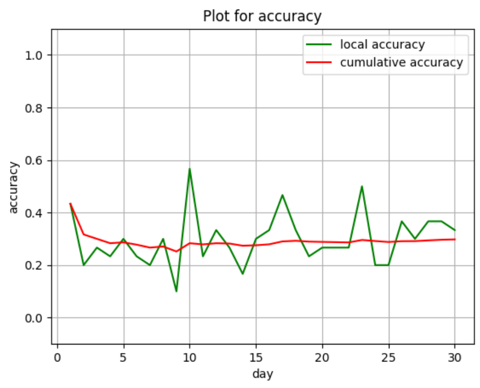
Figure 6.9: Accuracy plots for different simulation with target error probability fixed at 0.3

.

Even in this case, the Trial Proposer has no trouble with calibrating the difficulty of trials so as to achieve the desired overall accuracy: no matter the degree of the σ noise of the Child Simulator or the inclination of its decision boundary, the overall accuracy is made to converge to 70%, as expected from the design.

Things do slightly change when the target error probability is instead fixed to a value greater than 0.5. When the starting α is large enough, the Trial Proposer shows no sign of problems; when instead it becomes too small, the proposed trials do not reflect the desired error probability, as shown in figure 6.10.

In both the two figures, the target error probability is kept fixed at 0.8; however the Trial Proposer can't seem to achieve the desired accuracy in those situations, since the accuracy converges to 30% for the simulation with $\sigma_0 = 0.1$, and to 40% for the one with $\sigma_0 = 0.4$. This, however, **doesn't happen because of flaws in the design**: the issue doesn't stem from the fact that the Trial Proposer can't find suitable trials within the ND-NND space, but it is instead **due to the lack of such trials to begin with**. Trials with an Error Probability greater than 50% are in fact, mathematically, bound to be located in the subspace between



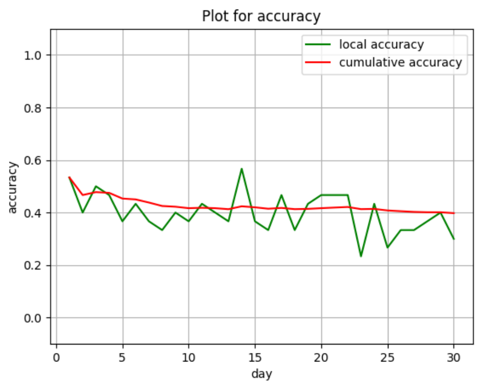


Figure 6.10: Accuracy plots for two simulations showing the Trial Proposer's limitations. The simulation on the left has parameters $\{10°, 0.1, 0.0, 30\}$, while the simulation on the right is characterized by the set $\{10°, 0.4, 0.0, 30\}$.

the player's decision boundary and the NND axis. With the player improving in terms of its α parameter, this space becomes increasingly smaller, and since σ also defines a certain minimum distance from the boundary to achieve the desired Error Probability, the proposal of very difficult trials **becomes mathematically unfeasible under the displayed circumstances**. For simplicity, this concept is given a graphical explanation in figure 6.11.

The only way to fix this problem is to program the Trial Proposer so that the search space is wider than the currently used $[-1,1]$ bounds of the normalized ND-NND space; however, another problem would arise in that case, which is the lack of highly unbalanced trials in the NND dimension inside of the Lookup Table: this is due to client limitations, so such problem cannot be entirely solved without performing changes to the client's code.

Other than this hard-to-solve issue, it is rather safe to say that, as long as it receives accurate estimations for the player, the Trial Proposer **can perform its task rather well for a wide range of parameters for the Child Simulator**. This can be attained in practice only if the Player Estimator is sufficiently good at performing its task, which is the main topic of the next subsection.

6.3.2 The AI's estimations

The other components that requires an in depth discussion is the Player Estimator, as its proper functioning determines not only the generation of balanced trials but also the actual assessment of the player's abilities over time.

Separating this component from the rest of the AI is, however, not as simple as

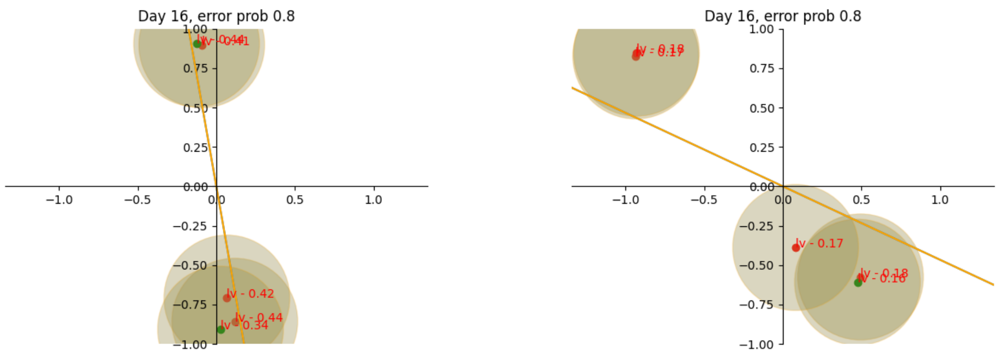


Figure 6.11: Generated Trials from two simulations with fixed target error probability equal to 0.8 that differ in terms of the α angle. From the figure on the right, it is clear that trials with such Error Probability are characterized by a non negligible distance with respect to the decision boundary; in the figure on the left, it can be seen that such distance cannot be guaranteed without crossing the NND axis, which in turns changes the actual correct answer for the trial and overthrows the related Error Probability. The result is that the trials are still generated within the correct part of the ND-NND space, but are associated with an Error Probability that is much closer to 50% than it is to 80%

in the case of the Trial Proposer: this component is in fact designed to receive as input large amounts of data that is produced by the Trial Proposer itself; moreover, providing it data that was not directly generated from the rest of the AI might lead to misleading results in terms of its performances. Therefore, the analysis carried out in this subsection will require the **full functionality** of the `PDEP_Evaluator`'s pipeline, minus the Trial Adapter as it's not strictly necessary outside of the client interaction.

Another difference with respect to the Trial Proposer lies in the fact that the Player Estimator also makes use of data generated in several different iterations: for this reason, an analysis has to be carried out also with configurations for the Child Simulator with non-zero learning rates. As for the target of the analysis, the aim is not to actually confirm whether the desired overall accuracy is being achieved, but rather to understand if the AI **can reasonably estimate the player's ability in terms of its α and σ parameters**. In order to get a numerical representation of that, a measurement will be consistently reported based on the **average distance of the estimated value with respect to the actual simulator's one**, which represents the average error in percentage after being normalized with respect to the maximum values of either α or σ.

The first pass estimation

As a first step, it's worth analyzing the results of the **first pass estimation** in simulations where the Child Simulator's learning rates are set to zero. For this purpose several simulations characterized by different starting parameters have been performed.

As shown by figures 6.12 and 6.13, the Player Estimator is fairly able to estimate the Child Simulator's parameters regardless of the α angle when the value for σ is kept low.

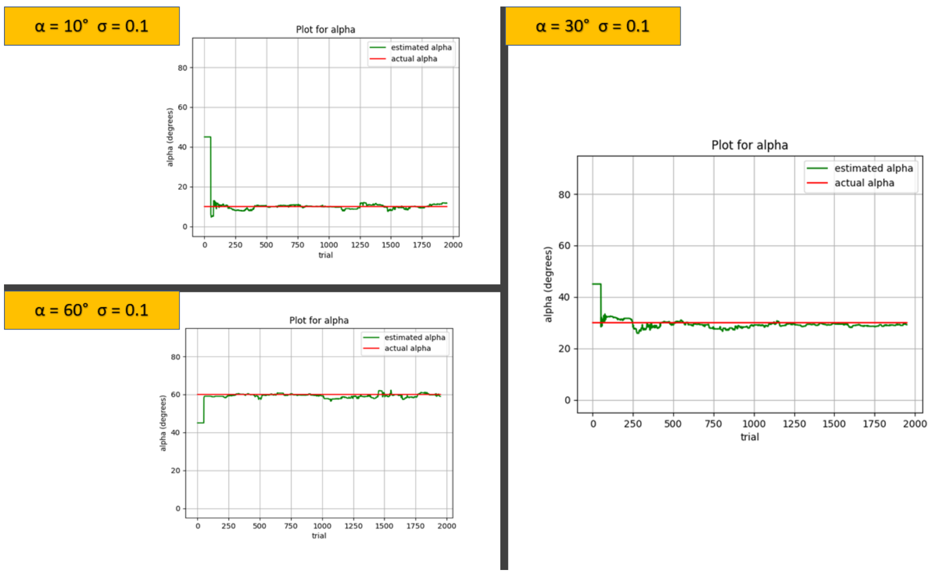


Figure 6.12: Plots for the first pass estimation of α versus the actual Child Simulator's value for simulations with different α_0. In all the simulations, the value of σ_0 is equal to 0.1, with both learning rates being set to zero

.

More in detail, it can be stated that simulations with different values for α_0 **did not lead to significant changes in terms of performances**. In fact, regardless of the chosen value for α_0, all the simulations led to both the estimated parameters having an average distance from the real value lower than 3%. This can be considered as a very good result, since this measurement includes also the first part of the simulation where the Player Estimator is simply returning default values to the Trial Proposer. When the σ noise increases however, the estimation for both parameters becomes more inconsistent, as shown in figure 6.14.

In such cases, the average error in the estimations reaches 5.7% and 6.6% for the estimation of the α and σ parameters respectively, which although is not particularly bad as a result, it's still considerably worse than in the previous simulations. In any case, this outcome is rather expected, as the Child Simulator is characterized

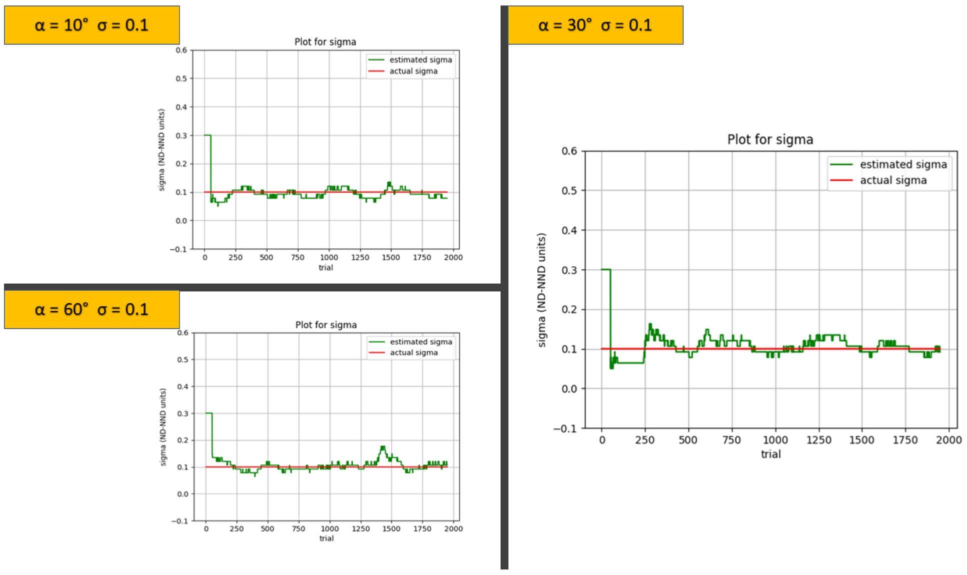


Figure 6.13: Sigma first pass estimation for the same simulations shown in figure 6.12. In all simulations the σ parameter was kept constantly equal to 0.1, and the AI was able to find such value regardless of the different configurations chosen for α_0

.

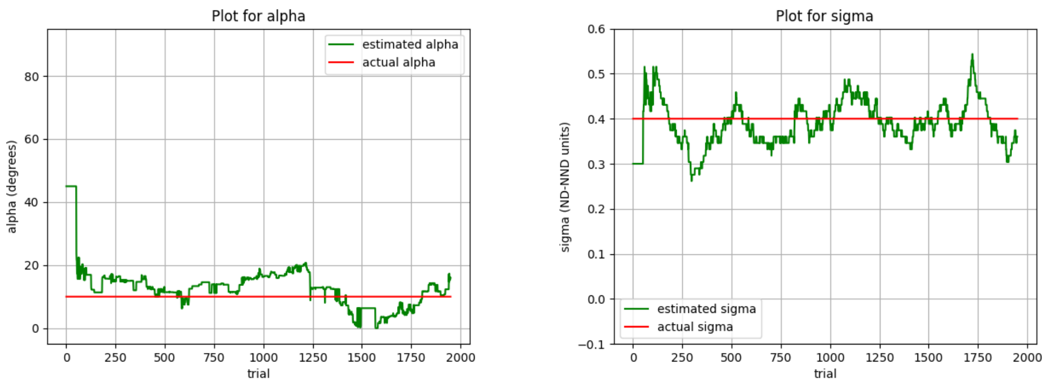


Figure 6.14: Plots for first pass estimation for both α and σ for a simulation characterized by the set $\{10°, 0.4, 0.0, 65\}$. As σ_0 is higher than before, both estimations appear to be less consistent in time

by a higher σ noise which leads to the generated trial data to be much more noisy: generally, the presence of a high noise within a dataset is not something that can be easily overcome without introducing large amounts of data, **which explains why**

the first pass mode that uses only 180 trials obtains these sub-optimal estimations.

The results of the second pass estimation

It is now time to discuss the performances attained by the second pass estimation, which in the rest of this thesis is considered to be performed at the **end of the player cycle**. As it's designed to adapt also to situations like the one shown in figure 6.14, it is worth comparing the results reported in the previous paragraphs with the ones produced by the second pass estimation. Both these results are compared in figure 6.15, where the displayed simulation was run with the same configuration as before.

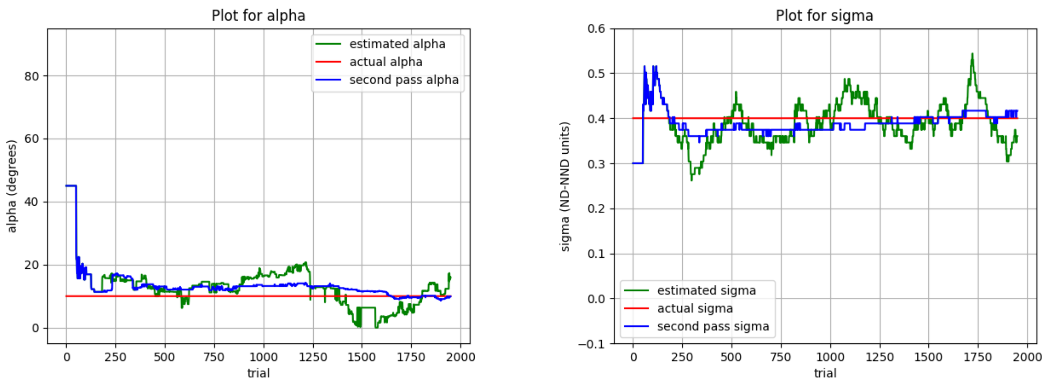


Figure 6.15: Plots for the same simulation of 6.14, with the addition of the second pass estimation's result

In all cases the Player Estimator is able to understand that, since the player's parameters are observed to not change significantly in time, it is possible **to leverage also from older data in order to perform the second pass estimation**. As should be expected, the second pass estimations therefore shows estimated values that are much closer to the actual Child Simulator's ones, reaching estimation errors below 1.5% for simulations with low σ, and bringing the error down to less than 3.5% for the high σ simulations shown in the figure.

Performances with an improving player

Some more discussions have to be made about simulations that are characterized by non-zero learning rates. For this purpose, a set of simulations of the kind $\{55°, 0.3, \tilde{lr}_s, 65\}$ will be analyzed, where maximum value considered for $\tilde{lr}_s$ results in the Child Simulator parameter's convergence before the end of the player cycle.

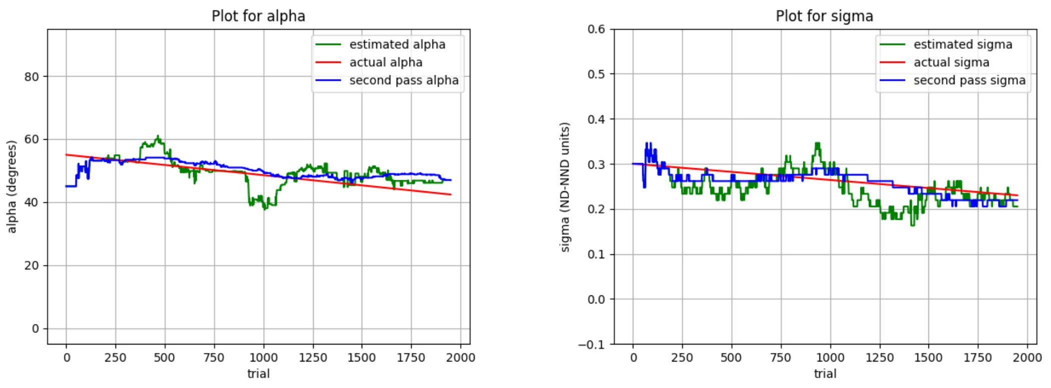


Figure 6.16: Plots of the estimated values for a simulations with $lr_n = -0.0022$

As shown in figure 6.16, the presence of a small normalized learning rate for both parameters does not represent a significant problem for the AI. While the average error is still slightly higher than simulations with null learning rates, the average error for the first pass estimations is in fact within reasonable values, being equal to 3.4% for α and to 4.6% for σ. Also in this case, **the second pass estimation improves greatly both results**, as instead both the errors go as low as 2.5%.

Even when the slope reaches high magnitudes, the Player Estimator can still manage to provide quite good estimations, as shown in figure 6.17

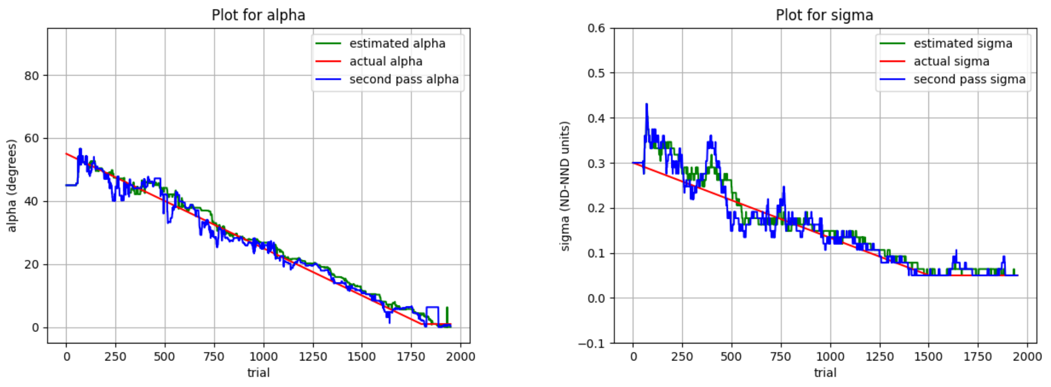


Figure 6.17: Plots of the estimated values for a simulations with $lr_n = -0.01$

In this case, the first pass estimation and the second pass one attain very similar results, as in the latter the Player Estimator understands that, in order to not introduce outdated data, **it is better to use a limited amount of trial data**

similarly to the first pass estimation. Moreover, although the first part of the simulation appears to be more noisy, the estimations of both α and σ reach quite good results, as they attain an error of around 2.2% for the estimation of α and of 3.5% in the case of σ.

6.3.3 General Considerations

In general, the `PDEP_Evaluator`'s components have shown to be able to perform their tasks reasonably well: in fact, the Trial Proposer can in principle follow the instructions provided by the Difficulty Controller by generating very balanced trials; on the other hand, the Player Estimator is able to provide reasonably good estimates to the Trial Proposer, with an average error that is around 3% for several different Child Simulator configurations. As a closure to this section, it is worth making considerations about the `PDEP_Evaluator` pipeline as a whole, by analyzing both the estimations and the achieved accuracy from different simulations.

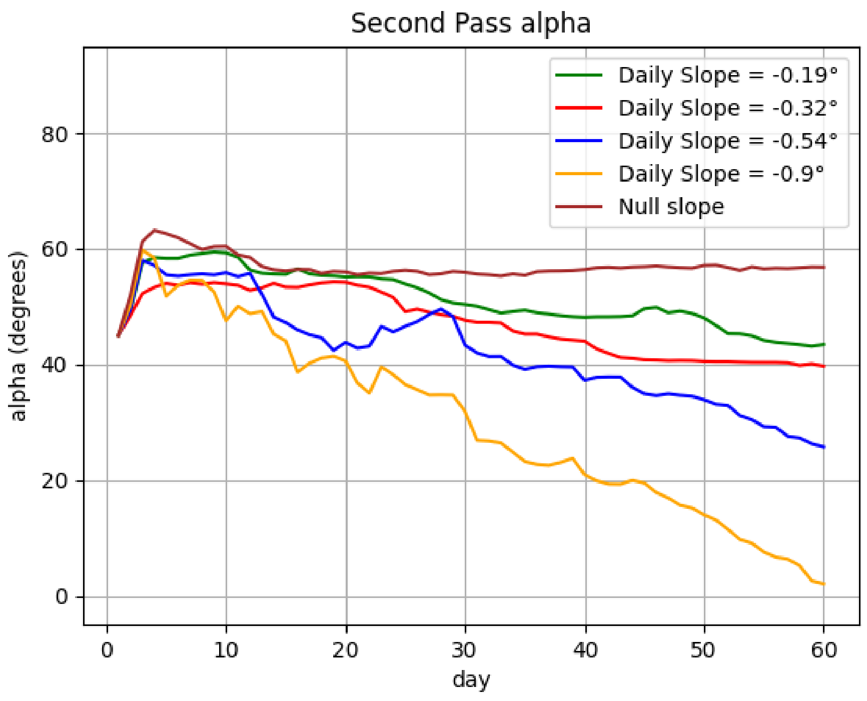


Figure 6.18: Plots for the second pass estimation of α for several simulations with varying learning rates. The learning rate (named "Slope" in the figure) is expressed as the daily decrease of the Child Simulator's α in degrees

.

As shown in figure 6.18, the AI is able to **distinguish fairly well the abilities of players with different learning rates**, as the plot shows estimated values that decrease proportionally to each simulation's learning rate.

The same goes for the estimation of the σ parameter, although the AI demonstrates greater difficulty in some situations: in fact, when the learning rate is rather high and the Child Simulator's σ is still high enough to influence estimations performed on a small dataset, the σ estimations appears to be more noisy. The fact that, as shown in figure 6.19, the estimation for σ is not always as consistent over time as the one for α is **however a rather expected result**. In fact, the latter is done according to well established procedures in the field of machine learning (i.e. SVM) while the former is performed through an **ad hoc algorithm designed by the candidate**. In addition to that, the σ estimation is also **reliant on the result of the one performed for** α: if noise is present in the estimated boundary during the current iteration, such noise will be propagated directly to the estimated σ value, thus enforcing the fact that the σ estimation **can't achieve better results than the** α **one by design**.

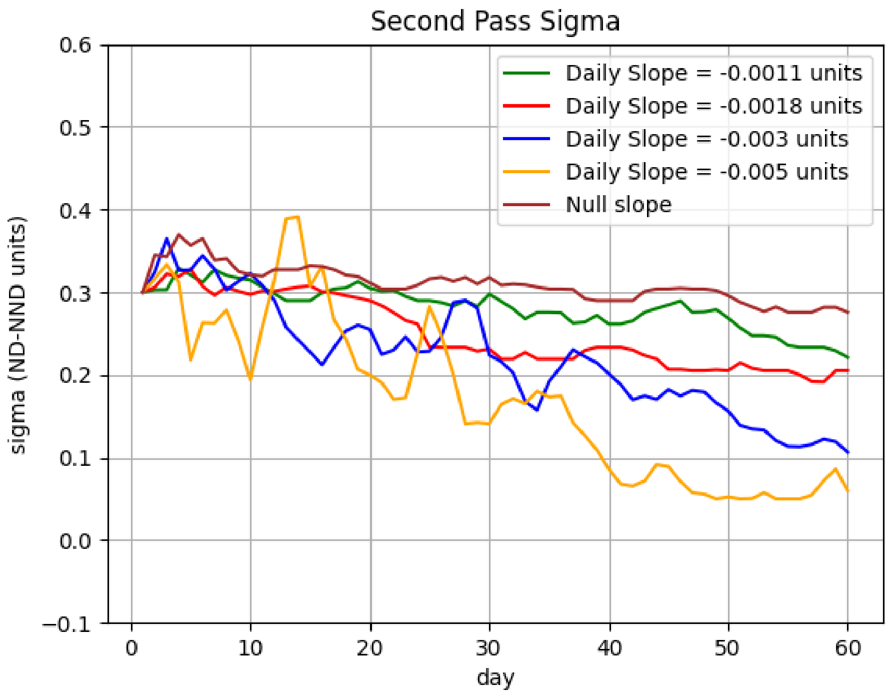


Figure 6.19: Plots for the second pass estimation of σ for several simulations with varying learning rates. The simulations from which this plot is generated are the same used for making the plot shown in figure 6.18

.

Having said that, the slightly sub-optimal results obtained in such cases should not be seen as a major problem. In fact, the simulations show remarkable results in terms of the simulator's accuracy, as shown in figure 6.20: no matter what learning rate for both parameters is given to the Child Simulator, **the AI is still able to make the player answer correctly according to the desired probability since very early stages of the simulation**.

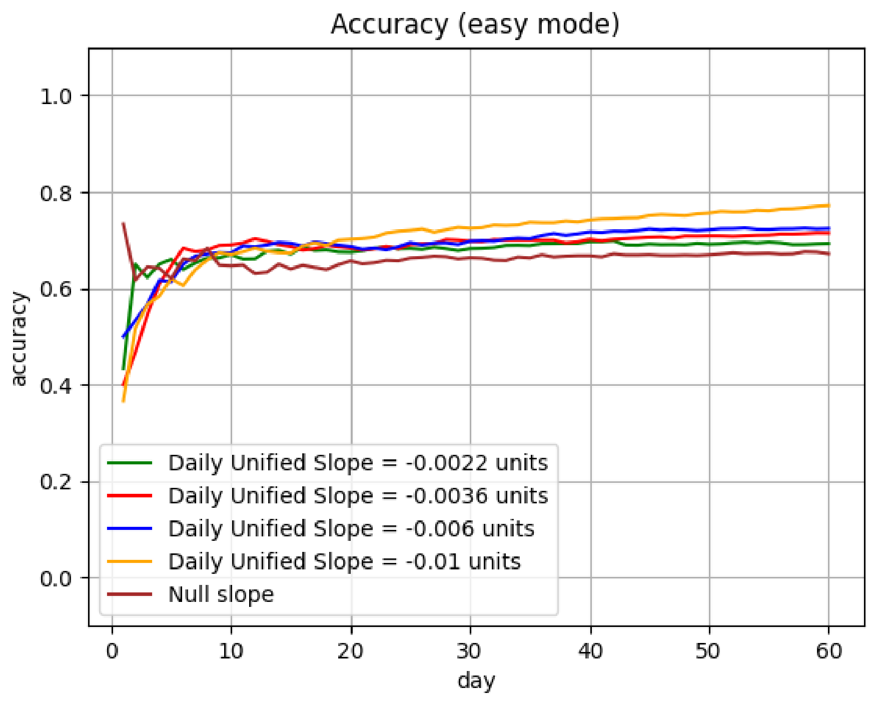


Figure 6.20: Cumulative accuracy achieved by the Child Simulator according to different learning rate. The simulations are the same mentioned in figure 6.18 and 6.19, and all show an overall accuracy quite close to the desired value of 70%

.

Even in the last days of high learning rate simulations, the lack of very hard trials within the ND-NND space is still not enough to significantly influence the AI's propositions, as the presence of this problem only brings to a change of about 7% in the final overall accuracy at the end of the simulation with $lr_s = -0.01$.

As a closure to this chapter, the accuracy plot for a similar set of simulations to the ones shown so far is displayed in figure 6.21, with the difficulty being calibrated as to achieve an overall accuracy of 50%: as the more balanced target error probabilities provided by the Difficulty Controller **allow the proposed trials to span more uniformly the whole ND-NND space**, the aforementioned problem appears visibly only in the latest days of the simulations, even if reaching an overall lower accuracy should be mathematically more difficult for the AI.

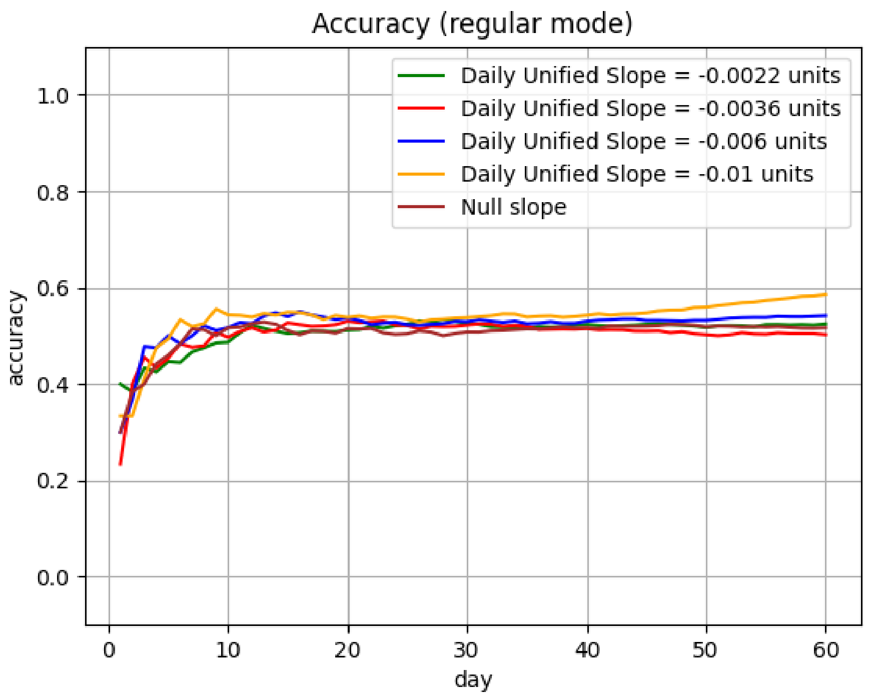


Figure 6.21: Cumulative accuracy plot for the same simulations mentioned in figure 6.20 but with different difficulty calibration.

.

Chapter 7

Conclusions and Future Works

With the conclusion of this thesis, The Number Farm has almost reached its second year of development: since its start, several students belonging to both Politecnico di Torino and University of Essex have brought their own contributions, allowing it to grow little by little into what it is today.

This thesis' work is no exception, as the game has finally reached several of its main objectives ever since its first phases of development. As a closure to this work, it is therefore wise to report the main additions that were brought to the game ever since the candidate's arrival in the project.

By going chronologically, the first main feature brought by the candidate consisted in the implementation of the **remote connection between the client and the server**. The Number Farm has always been designed to be playable from a mobile device, and now that it is possible to run the server on its separate machine, the rest of the project's contributors were able to create builds for the client side that could run on Android devices. This was followed by a proper handling of multiple clients from the server's code, which makes The Number Farm a step closer to **becoming a fully operational game**.

The second contribution is also the most important brought by this thesis: the development of an AI that can adapt to the player's ability. This game has in fact been intended to be based on Artificial Intelligence ever since its conception, and with the introduction of the `PDEP_Evaluator`, it is no understatement to say that this goal has finally been achieved. While borrowing some ideas from other previous applications, the AI has finally come to a state where it can **provide a balanced training experience to its players**: by using well-established techniques in the field of machine learning as well as some others developed purposefully for this project, the AI is able to understand the player's ability according to the sharpening and filtering hypotheses described in Learning to Focus on Number; starting from this knowledge, it is additionally capable of proposing numerical comparison trials that are tailored around the player, by following a criterion designed around reaching an overall satisfying accuracy while also avoiding excessive exertion to produce an answer. As it's also one of the main points of the filtering and sharpening hypotheses, the `PDEP_Evaluator` was also designed to follow and estimate how each player improves with time and practice, which can open new paths in the study of Dyscalculia and on how it can be treated.

Finally, the addition of the **Simulations Runner** also introduces a long-awaited feature in the project. In fact, the **Child Simulator** was immediately identified as one of the key points of this thesis, as the implementation of a component able to simulate the behavior of a player has always been considered necessary to continue with the development of the AI. Moreover, the candidate developed it in accordance to the filtering and sharpening hypotheses, which is the main distinctive trait of The Number Farm. This component, together with the whole simulation environment, allows to **properly study the behavior of the AI under several different conditions**: thanks to them, the development of the `PDEP_Evaluator` could be accompanied by a complete testing procedure, which led to the satisfying demonstrations of its capabilities discussed in the last part of this thesis.

That being said, the development of The Number Farm still hasn't reached its end. As already mentioned in the rest of this thesis, there are still some aspects regarding the AI that still need improvements, namely:

- A more concrete definition for the Perceived Difficulty. As discussed in the previous chapters, it would be best if this metric could be inferred from real-life data, by possibly relating it to the time taken by the player to answer to the proposed trials.
- A proper modelling of such response time inside of the Child Simulator, which would in turn allow to test the optimization for the aforementioned Perceived Difficulty.
- Some possible tweaks to its design, after a proper assessment has been done by testing the application with real children on a sufficiently long period of time.

Other than these aspects, it is rather safe to say that the number comparison mini-game is very close to its final state. As at its core it is meant to be a broad and entertaining game, a possible direction for future developments of The Number Farm may be found in the implementation of other mini-games, as done by other previous applications. Dyscalculia is in fact an issue affecting several areas of mathematical development in children, so taking care of some other of its aspects, such as arithmetic operations or symbolic representation of number, can be another step forward in making The Number Farm a valuable application in the field of software-based training programs. Moreover, the impact that this would have on the overall user experience should not be underestimated. The hope is, after all, to alleviate the issues caused by Dyscalculia, and providing an entertaining and engaging game can surely improve the success of The Number Farm while also motivating children to play and possibly improve their mathematical abilities.

Appendix A

Ablation Studies for the AI

In the field of Artificial Intelligence and machine learning, the term "hyper-parameter" is widely used to indicate additional inputs required by algorithms and models other than the actual input data. As these hyper-parameters affect the quality of the end result, a primary study has to be generally conducted, with the aim of finding the optimal hyper-parameters maximizing the performances for the specific task at hand.

The proposed project, as made clear in section 5.5, is no stranger to such concepts, as the Player Estimator performs two different tasks that require an optimization of their related hyper-parameter. This appendix is therefore dedicated to the two ablation studies related to such procedures, namely:

- The selection of the optimal C during the estimation of the α parameter, as it is required in order to run the SVM algorithm.
- The optimal number of trials z used during the second pass estimation given the estimated learning rates of the player.

A.1 Finding the optimal C

In general, SVMs are not used exactly as they were within the project. In the field of machine learning, they are in fact trained in order to generate a model for classification or regression by using a pre-determined training set, so as to generalize its labelling patterns and reproduce them in real use-case scenarios where the label is not provided. As they also require an hyper-parameter **C** acting as a regularization parameter (i.e. control the effects of over-fitting), an ablation study is generally performed in order to find its best value. This is done by training several models with varying value for C, and by analyzing them through the use of a validation dataset: simply, the various models are used to label the validation dataset, and the one achieving the best results also tells which is the optimal C parameter for the task at hand.

In the proposed project however, the AI uses SVMs to estimate α in a way that differs to the aforementioned approach regarding two aspects:

1. The "training dataset" D is not a pre-determined collection of data, but it is an ever changing set produced according to the design of the Trial Proposer and of the Difficulty Controller.

2. The performance attained by the procedure is not measured by how well the resulting classifier can reproduce the labelling pattern, but rather as how close the estimated values are to the Child Simulator's ones. As such, together with the fact that the implemented SVM uses a high-bias linear kernel, no validation dataset is required as the performances can be immediately measured from the resulting classifier alone.

So, how should the Player Estimator choose the proper value for C at each of its iterations?

As a starting point, it is possible to make an important assumption by looking at how the dataset D is generated. In fact, such dataset is generated by the Trial Proposer by receiving the α and σ parameters estimated in previous iterations together with the Difficulty Controller's target error probability and target perceived difficulty: given that the Difficulty Controller provides all possible input parameters in the span of few iterations, it can be assumed that large datasets (i.e. with size greater than 180) are distributed within the ND-NND space **depending on the α and σ provided as input to the Trial Proposer**. In other words the idea is that, by feeding the same α and σ to the Trial Proposer for a large number of iterations, it is possible to produce a large **training dataset**. Moreover, this dataset is, in principle, similar to the datasets **produced during normal use when the same α and σ are being estimated**: by finding the value of C leading to the best estimations for this training dataset, it is likely that such value will also be the optimal one during the normal use-case.

Starting from this assumption, the candidate came up with an ablation procedure that is described in the following steps:

1. At first, a set of configurations $\{P_0, P_1, ...P_n\}$ is generated. Each configuration P_i represent a set of parameters $\{\alpha_i, \sigma_i\}$, with each configuration differing with respect to all the others for at least one of these two parameters. Together with that, a set of candidate C values $\{C_0, C_1, ...C_k\}$ are generated, logarithmically spanning the range of $[0.001, 1000]$.

2. At the beginning of the ith iteration of the procedure, the configuration P_i is selected.

3. By selecting a C_j value among the candidate values, a certain number M of simulations was run by using C_j as the hyper-parameter for the SVM. Given the current configuration P_i, each of these M simulations were run with the set of parameters $\{\alpha_i, \sigma_i, 0.0, 80\}$, following the notation introduced in section 6.2.1.

4. For each of the M simulations, the average error of the first pass estimation is measured with respect to both the Child Simulator's normalized α and σ values. Just like the metric defined in section 6.3.2, such error is computed as the normalized average distance of the estimated value with respect to the Child Simulator's one.

5. All errors related to each of the M simulations are summed: the resulting value will be named E_{ij}, and it represent a measure of the overall error attained in both the α and σ estimations while using configuration P_i and C_j as the SVM's hyper-parameter.

6. Step 3 to 5 are repeated, until the set $\{E_{i0}, E_{i1}, ...E_{ik}\}$ has been computed.

7. Out of this set, the minimum value E_{io} is extracted: the C value identified by the index o is therefore selected as the optimal C for configuration P_i.

8. Step 2 to 7 are then repeated for each configuration, until the optimal C value has been found for all of them.

At the end of this ablation procedure, the pair $\{P_i, C_i\}$ representing the optimal C value for the ith configuration has been generated for all the studied configurations, which is graphically shown in figure A.1

As the several configurations have been generated in order to reasonably span the whole space of possible values that can be estimated by the AI, the procedure to extract the optimal C during normal operation is rather simple: first, the α and σ values estimated during the previous iteration are collected and normalized, thus producing the pair $\{\tilde{\alpha}, \tilde{\sigma}\}$; then, the euclidean distance of this pair is computed with respect to each $\tilde{P_i}$ configuration, which is the set of configurations defined in the ablation study after undergoing normalization; finally, the normalized configuration that is the closest with the pair $\{\tilde{\alpha}, \tilde{\sigma}\}$ is selected, and the optimal value of C related to it is chosen as the C to be used in the estimation of the current iteration.

There is a drawback in this procedure however: although the Trial Proposer is assumed to produce similar datasets when the same α and σ are provided as input over several iterations, the same can't be said when the difficulty calibration performed by the Difficulty Controller is changed. In fact, by varying the overall target accuracy that has to be achieved by the player, the way the generated trials are distributed in space does change as well. This means that the ablation

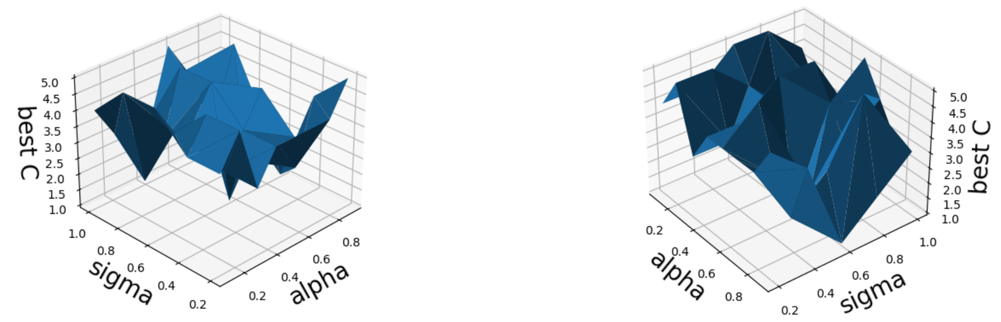


Figure A.1: 3D plot for the ablation study of the C hyper-parameter. The x and y axes represent the normalized values of α and σ defined by each configuration, while in the Z axis is reported the exponent related to the power of 10 that leads to the optimal result when used as the C hyper-parameter

procedure has to be carried out for each available difficulty calibration, which is part of the reason why only two of them are currently provided by the AI.

A.2 Finding the optimal z

When the second pass estimation was explained in subsection 5.5.3, the computation of the mapping between the steepest estimated slope and the optimal amount of trials z for the estimation was only briefly mentioned. As this appendix is dedicated to talk about these ablation procedures, it is now time to discuss how such mapping was found by the candidate.

The objective is to come up with pairs $\{\tilde{S}_i, Z_i\}$, where $\tilde{S}_i$ is a normalized slope representing the decrease of either the normalized α or σ value after each mini-game iteration, and Z_i represents the optimal amount of trials to use for estimating the parameters of a player improving according to such slope. Moreover, as the goal is basically to understand how "fast" the proposed data becomes too old to be used, each $\tilde{S}_i$ is considered to be negative: in the case that the AI finds a player that is getting worse over time with a positive slope, the mapping is done with respect to the closest $\tilde{S}_i$ in terms of its absolute value. The overall procedure is rather similar to the one for the C hyper-parameter, and it is summarized in the following steps:

1. Before starting the procedure, two arrays are instantiated: the first one is composed of the set of $\{\tilde{S}_0, \tilde{S}_1, ...\tilde{S}_n\}$, and represent a series of possible values for the aforementioned normalized slope; the second is instead a set of

candidate z values $\{Z_0, Z_1, ...Z_k\}$. Moreover, the former is generating in order to span logarithmically the range $[3.3^{-6}, 3.3^{-4}]$, while the latter uniformly spans the range $[100, 1800]$.

2. At the beginning of the ith iteration of the procedure, the slope $\tilde{S}_i$ is selected and a certain amount M of simulations is performed with parameters $\{80°, 0.4, \tilde{S}_i * 30, 80\}$. Note that the $\tilde{S}_i$ is being multiplied by 30 (i.e. the amount of trials performed per day of simulations) before being used as a simulation parameter. This is because, as defined in section 6.2.1, that value should represent the normalized **daily** learning rate for both α and σ, while $\tilde{S}_i$ represent the learning rate over the single trial proposition.

3. For each of the M simulations, both the generated trials and the result of the first pass estimations are collected.

4. By selecting a Z_j value among the ones defined in step 1, the second pass estimation is performed for each of the M simulations. For each of them, the related first pass estimation data and the generated trials are provided as input, while the amount of trials used for the estimation is set to be equal to Z_j.

5. From the results of all these second pass estimations, a measure of the average error E_{ij} is extracted, in a similar way to how it was computed at step 4 and 5 of the ablation study for parameter C. E_{ij} therefore represents a measure of the average error attained by using Z_j as the z value for the second pass estimation while having a player that improves according to $\tilde{S}_i$.

6. Step 4 and 5 are repeated until the set $\{E_{i0}, E_{i1}, ...E_{ik}\}$ is computed.

7. Out of this set, the minimum value E_{io} is extracted, and the corresponding Z_o is found as the optimal z to be used when the slope $\tilde{S}_i$ is encountered.

8. Steps 2 to 7 are repeated for each of the candidate slopes defined in step 1, until the optimal z value has been found for each of them.

It's worth noting that the ablation procedure does not take into account various configuration of the starting parameters for the Child Simulator, as they are not supposed to play a role in the usability over time of the proposed trials. Instead, values close to the maximum ones were used as initialization for the Child Simulator, as to allow the simulation to finish without either α or σ reaching the minimum possible value regardless of the considered learning rate. The set of pairs $\{\tilde{S}_i, Z_i\}$ generated by this procedure is shown in figure A.2

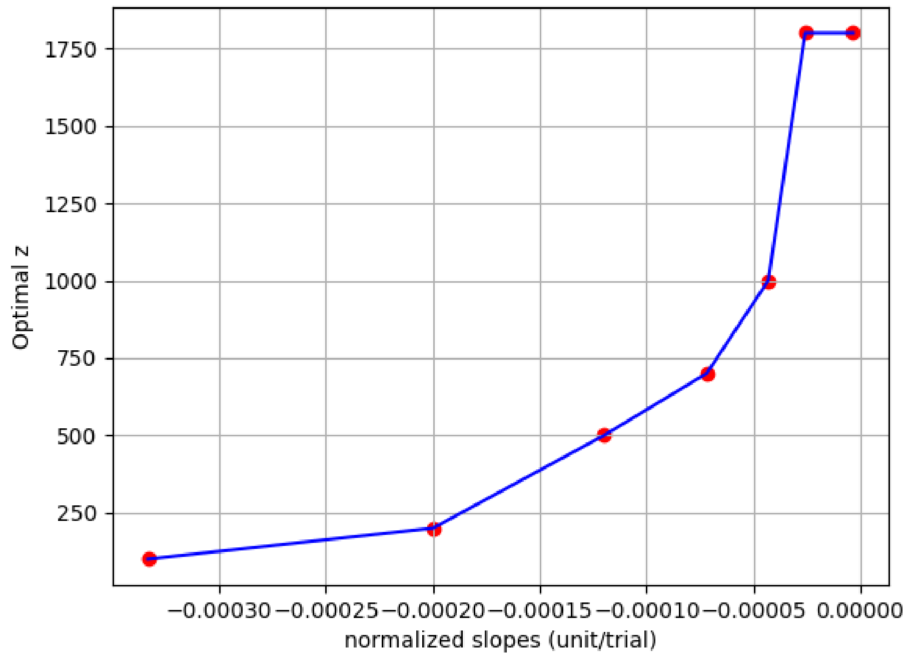


Figure A.2: The optimal value of z for each of the candidate slopes considered within the ablation study. Note that the x axis represents the loss of either the normalized α or σ parameter per each mini-game iteration: as should be expected, higher slopes in absolute value limit the amount of usable trials for the estimation, while lower slopes allow the AI to leverage from older data

.

Similarly to the case of the C hyper-parameter, the optimal z is selected during normal operation by taking the estimated learning rates for both α and σ, normalizing them and selecting the most restrictive (i.e. the highest in absolute value). After that, the closest $\tilde{S}_i$ with respect to this value among the ones considered in the ablation study is selected, and its related value for z is also provided as the optimal z for the current iteration of the second pass estimation.

Appendix B

Computation of ND-NND variable

During the course of this thesis, the concept of the ND and NND values for trials related to number comparison tasks has been widely used. These two values are in fact vital to describe the trials according to the filtering and sharpening hypothesis; however, the actual computation for these two values from the actual trial's parameters has only been mentioned, especially when it comes to the NND value.

Since such computation is not extremely relevant for the `PDEP_Evaluator` and it was already partially implemented before the candidate's arrival, its discussion has been relegated to this appendix. Before continuing, two different versions for these computations can be identified:

- The computation according to the paper Learning to Focus on Number.
- The computation adapted to this project.

Since the second version is derived from the first one, this appendix will first treat the computation as described in the paper, and only later the aspects regarding the version used in the server's code will be explained.

B.1 ND and NND within Learning to Focus on Number

In section 2.2 the theoretical concepts behind numerical comparison tasks have been described in their broadest form. As explained by the authors, the trials related to such tasks are defined by two sets of elements, each of which is characterized by a set of dimensions that can both represent numerical and non-numerical features of said trial. For completeness, these dimensions are briefly recalled in the following together with their abbreviation:

- **N**: the number of elements in the set.
- **ISA**: the area occupied by each single element.
- **TSA**: the total area occupied by all elements.
- **FA**: the area of the disk where the elements are located.
- **Spar**: the average distance between each element.

Of the aforementioned dimensions, only the first one is related to numerical features of the trial, while the remaining four are considered to be as **non-numerical dimensions co-related to the numerical one**.

The N dimension already allows to compute the ND value for the given trial, as it is simply computed as the logarithmic ratio of the N value for the right set over the N of the set displayed in the left.

Regarding the NND however, the procedures requires some more steps. In fact, at first all four non-numerical dimensions have to be reduced into a single dimension, that will be used to summarize the non-numerical features of each of the two sets of elements.

To do so, the authors of the paper preformed the following two steps:

1. First, the four non-numerical dimension were transformed into their component orthogonal to the N dimension. This was done by subtracting from each dimension its component parallel with respect to number.
2. From the new set of non-numerical dimensions produced in this way, a dimensionality reduction in the form of PCA was performed starting from the dataset used within the experiments, out of which only one dimension was extracted.

The dimension obtained in this way would therefore be orthogonal with respect to the N dimension while being the most describing of the four non-numerical features of the trials. According to the authors, this component explained 98.9% of the variance of the 4 different non-numerical dimensions within the used dataset, and since the PCA procedure found a set of weights equal to $\{0.557, 0.487, 0.473, 0.467\}$ which are related respectively to the Spar, ISA, TSA and FA, they could also conclude that all original non-numerical dimensions were being loaded equally into the found summarizing dimension.

By naming this dimension as SD, it is finally possible to define the NND as, simlarly to the ND value, the logarithmic ratio of the SD of the set shown in the right over the SD of the one on the left.

B.2 The ND and NND within the project

As described in chapter 5, the AI makes decisions and estimations within the ND-NND space, as it's how the trials are represented by its most relevant components, namely the Trial Proposer and the Player Estimator.

The way these two values are computed is irrelevant when it comes to these two components, as they are designed to function regardless of the meaning of the ND-NND space. However, when the server code has to interact with the client side, the **Trial Adapter** has to translate the trials generated by the Trial proposer into the set of parameters used within the client: as already explained, this is done by choosing a trial within the Lookup Table, according to its vicinity in the ND-NND space to the input trial.

The Lookup Table is therefore required to contain the ND and NND value for each one of its entries; however, the non-numerical dimensions used to describe trials in the Lookup Table are slightly different from the ones used in Learning to Focus on Number, due to the need of adapting them to the client side parameters. More in detail, each fence of each trial within the Lookup Table is represented by the triplet of dimensions defined by $\{N, FA, ISA\}$, each of which having the same meaning as the corresponding ones described in the previous section.

Despite the relationship of these three dimensions defines a set of constraints when it comes to the trials displayable within the clinet, it can be observed that both the FA and ISA are **already orthogonal with respect to N by definition**. Therefore, in the case of the project, the summarizing dimension SD was simply computed as the first dimension found by the PCA applied to the set of possible combinations of FA and ISA that could be found within the Lookup Table. This procedure is performed for both the Lookup Tables used within the application, and just like in the paper, the PCA found similar weights for both the FA and ISA, meaning that also in this case both dimensions contribute equally to the computation of the SD.

As for the actual ND and NND values, the same procedures used in the paper are followed, starting from the N and SD dimensions suitably defined in the context of the project.

Bibliography

[1] G. R. Price and D. Ansari. «Dyscalculia: Characteristics, Causes, and Treatments». In: (2013). DOI: `http://dx.doi.org/10.5038/1936-4660.6.1.2` (cit. on p. 1).

[2] L. Attout, E. Salmon, and S. Majerus. «Working Memory for Serial Order Is Dysfunctional in Adults With a History of Developmental Dyscalculia: Evidence From Behavioral and Neuroimaging Data». In: (2015). DOI: `https://doi.org/10.1080/87565641.2015.1036993` (cit. on p. 1).

[3] K. Landerand A. Bevan and B. Butterworth. «Developmental dyscalculia and basic numerical capacities: a study of 8–9-year-old students». In: (2004). DOI: `https://doi.org/10.1016/j.cognition.2003.11.004` (cit. on p. 1).

[4] R. S. Shalev. «Developmental Dyscalculia». In: (2016). DOI: `https://doi.org/10.1177/0883073804019010060` (cit. on p. 1).

[5] E. Callaway. «Dyscalculia: Number games». In: (2013). DOI: `https://doi.org/10.1038/493150a` (cit. on p. 1).

[6] A. J. Wilson, S. K. Revkin, D. Cohen, L. Cohen, and S. Dehaene. «An open trial assessment of 'The Number Race', an adaptive computer game for remediation of dyscalculia». In: (2006). DOI: `https://doi.org/10.1186/1744-9081-2-20` (cit. on pp. 2, 17).

[7] J. Halberda and L. Feigenson. «Developmental change in the acuity of the 'number sense': The approximate number system in 3-, 4-, 5-, and 6-year-olds and adults». In: (2008). DOI: `https://doi.org/10.1037/a0012682` (cit. on p. 2).

[8] M. Piazza, V. De Feo, S. Panzeri, and S. Dehaene. «Learning to focus on number». In: (2018). DOI: `https://doi.org/10.1016/j.cognition.2018.07.011` (cit. on p. 2).

[9] S. Dehaene. «Précis of The Number Sense». In: (2002). DOI: `https://doi.org/10.1111/1468-0017.00154` (cit. on p. 4).

[10] B. Fischer, C. Gebhardt, and K. Hartnegg. «Subitizing and visual counting in children with problems in acquiring basic arithmetic skills». In: (2008) (cit. on p. 5).

[11] M. G. Von Aster and R. S. Shalev. «Number development and developmental dyscalculia». In: (2007). DOI: `https://doi.org/10.1111/j.1469-8749.2007.00868.x` (cit. on p. 6).

[12] P. J. Dinkel, K. Willmes, H. Krinzinger, K. Konrad, and J. w. Koten. «Diagnosing Developmental Dyscalculia on the Basis of Reliable Single Case FMRI Methods: Promises and Limitations». In: (2013). DOI: `https://doi.org/10.1371/journal.pone.0083722` (cit. on p. 6).

[13] V. Izard, C. Sann, E. S. Spelke, and A. Streri. «Newborn infants perceive abstract numbers». In: (2009). DOI: `https://doi.org/10.1073/pnas.0812142106` (cit. on p. 7).

[14] M. Piazza, P. Pica, V. Izard, E. S. Spelke, and S. Dehaene. «Education Enhances the Acuity of the Nonverbal Approximate Number System». In: (2013). DOI: `https://doi.org/10.1177/09567976124640` (cit. on pp. 7, 11).

[15] J. Nys, P. Ventura, T. Fernandes, L. Querido, J. Leybaert, and A. Content. «Does math education modify the approximate number system? A comparison of schooled and unschooled adults». In: (2013). DOI: `https://doi.org/10.1016/j.tine.2013.01.001` (cit. on p. 7).

[16] T. Gebuis and B. Reynvoet. «The interplay between nonsymbolic number and its continuous visual properties.» In: (2012). DOI: `https://doi.org/10.1037/a0026218` (cit. on p. 7).

[17] S. Dehaene. «Symbols and quantities in parietal cortex: elements of a mathematical theory of number representation and manipulation». In: (1993). DOI: `https://doi.org/10.1093/acprof:oso/9780199231447.003.0024` (cit. on p. 8).

[18] A. Starr, N. K. DeWind, and E. M. Brannon. «The contributions of numerical acuity and non-numerical stimulus features to the development of the number sense and symbolic math achievement». In: (2017). DOI: `https://doi.org/10.1016/j.cognition.2017.07.004` (cit. on p. 8).

[19] M. Piazza, A. Facoetti, A. N. Trussardi, I. Berteletti, S. Conte, D. Lucangeli, S. Dehaene, and M. Zorzi. «Developmental trajectory of number acuity reveals a severe impairment in developmental dyscalculia». In: (2010). DOI: `https://doi.org/10.1016/j.cognition.2010.03.012` (cit. on p. 11).

[20] A. J. Wilson, S. Dehaene, P. Pinel, S. K. Revkin, L. Cohen, and D. Cohen. «Principles underlying the design of 'The Number Race', an adaptive computer game for remediation of dyscalculia». In: (2006). DOI: `https://doi.org/10.1186/1744-9081-2-19` (cit. on p. 14).

[21] T. Käser, G. M. Baschera, J. Kohn, K. Kucian, V. Richtmann, U. Grond, M. Gross, and M. Von Aster. «Design and evaluation of the computer-based training program Calcularis for enhancing numerical cognition». In: (2013). DOI: `https://doi.org/10.3389/fpsyg.2013.00489` (cit. on p. 18).

[22] K. Kucian, U. Gronda, S. Rotzer, B. Henzi, C. Schönmann, F. Plangger, M. Gälli, E. Martin, and M. Von Aster. «Mental number line training in children with developmental dyscalculia». In: (2011). DOI: `https://doi.org/10.1016/j.neuroimage.2011.01.070` (cit. on p. 18).

[23] T. Käser, A. G. Busetto, G. M. Baschera, J. Kohn, K. Kucian, M. Von Aster, and M. Gross. «Modelling and Optimizing the Process of Learning Mathematics». In: (2011). DOI: `https://doi.org/10.1007/978-3-642-30950-2_50` (cit. on p. 21).

[24] L. A. Outhwaite, M. Faulder, A.Gulliford, and N. J. Pitchford. «Raising Early Achievement in Math With Interactive Apps: A Randomized Control Trial». In: (2019). DOI: `http://dx.doi.org/10.1037/edu0000286` (cit. on p. 25).

[25] G. Brugo. «Development of an AI Game for Children with Dyscalculia». In: (2022) (cit. on pp. 26, 29, 35).

[26] I. Orefice. «Design of an AI-Based Gane for Prevention of Dyscalculia». In: (2022) (cit. on pp. 26, 35).

[27] F. Hurn. «Development of an AI game-app for dyscalculic children». In: (2021) (cit. on p. 35).

[28] R. Rana. «Prevention of Dyscalculia at an early Stage using a Smart game». In: (2022) (cit. on p. 36).

[29] Kudirat Ajibola Sofola. «Preventing Dyscalculia using an AI based game». In: (2022) (cit. on p. 37).

[30] S. Katoch, S. S. Chauhan, and V. Kumar. «A review on genetic algorithm: past, present, and future». In: (2020). DOI: `https://doi.org/10.1007/s11042-020-10139-6` (cit. on p. 54).