UNIVERSITY OF NAPLES FEDERICO II

Department of Electrical Engineering and Information Technology

C.d.L. COMPUTER SCIENCE

Dynamic difficulty adjustment between Game design and Game development

Relatore
Prof. Marco Faella
Correlatore
Prof. Walter Balzano
Tutor
Dott. Carlo Cuomo

Candidato
Davide De Vita
N97000301

Anno Accademico 2020/2021

Alle maestre Daniela, Marilina e Rosa

che fin da piccolo mi avete insegnato che

intelligenza è: trovare soluzione

Index

Introduction

Starting as experiments in ‘50s, videogames quickly evolved in the last seventy years; for the first half of this period, they were still pure scientific innovation and research topic[B-1], and only around 1980 they finally became popular to the crowds. Since then, their growth has been irrepressible; “games” became a topic in various fields: from science to economics and design; it did also spread across the media, in movies, books, sitcoms and, lately, in some parts of the world it is a sport (e-sport) with tournaments and prizes worth millions of dollars.

During this unbridled evolution, the gaming industry affected and invested into the computer industry and more in general, in technology; nowadays it is possible to access to a wide variety of gaming graphic cards, computers, mice and keyboards, and a new and improved model is developed every month.

But what perhaps is still unknown to the world is the work and effort behind the creation of a game, or a videogame in particular; originally they were developed by mathematicians or physicists who were mostly responsible of any aspect: from the lore to the levels and characters to graphic itself. Nowadays, there are specialized figures for any of these aspects; teams of Game Designers are appointed to come up with new ideas and to make their products thrilling and fascinating for their users, basing their work on detailed analysis.

Game Design grew to be a science, and it studies the core aspects of the creation of a game idea (not necessarily a videogame). The Game Designers will have to take into account the expectations of their audience, and to this end they will have to choose carefully the ideal target audience, highlighting age, gender and experience if necessary; they will also have to make sure that the idea is innovative and interesting, they will have to involve the player and stimulate him/her through music, images, game strategies and emotions. Alongside the designers there are other figures specialised in the design of levels, characters, music and the interaction between the player and the game, as well as marketing and communication experts. It is for this reason that the quality of games has been rising faster and faster over the last twenty years.

Among the different and complicated aspects that have contributed to the growth of the quality of video games, a place of honour must be given to the search for “realism”; greater realism means greater involvement for the player and, in general, greater overall appreciation. The path towards realism started with graphical progress, the graphical impact is in fact the cause of the first impression obtained when approaching a new game; it should be noted that 'realism' does not necessarily mean the resemblance to reality outside the game, but rather the credibility that the events we witness in the game, under the premises and rules of the game world, are possible; and it is this aspect that allows players to immerse themselves and let themselves be surrounded by the world they enter.

Entertainment is but one aspect; games are now moving into reality, and the invisible boundaries separating games and reality are now becoming increasingly obscure[B-2]. In the same way that the graphics have an initial impact on the reality of the game, the intelligence of the characters, friends and foes, that we meet constitutes the next; and among the most obvious features in the eyes of the player is certainly the intelligence of their opponent. In its nature of scientific origin, artificial intelligence has seen as its main objective the achievement of ever greater goals, of ever greater 'intelligence'; however, precisely in the field of videogames, this approach is not optimal. The drawback arises in the ability of the computer to consider, in the same amount of time, too many more scenarios than the player itself; thus, providing an unfair challenge.

As mentioned, in the past, games were a product of science and there was a sense of competition between the player and the programmer; the latter tended, as the game progressed, to put the player in front of more and more complex challenges rather than more and more stimulating ones. What was not considered then was that although for advanced players a game experience is more satisfying when it is difficult to defeat[B-3], beginners tend to enjoy the game more when it is challenging yet beatable[B-4]. This was the big change in the gaming experience: with the help of teams specialised in different areas, games gradually became a path around the player, and not a road the player had to walk down.

Although in some respects this has made recent games easier on average, receiving criticism from gamers of the previous generation, this approach has led the world of video games to a much wider audience; from a small circle of mainly adolescent males to a current audience almost entirely disengaged from gender and age.

One of the fundamental pillars of this process of centralising the experience around the player is the difficulty of the game. The original approach of the games considered an "intrinsic difficulty" of the game itself, which made the challenge proposed to the player "to overcome to the difficulty encountered"; the great merit of this approach is the similarity of the game experience felt by all the players; similarly, the enormous defect that makes it an incomplete approach is the lack or overabundance of experience on the part of the player, who thus often risks being better than the game requires, finding it uninspiring, or not good enough, finding the game frustrating.

All these aspects will be addressed in detail later. The aim of this dissertation is to analyse and study one of the approaches proposed in literature in order to correctly balance the difficulty of the game. This approach is called '***Dynamic Difficulty Adjustment (DDA)***' and consists of observing the player's ability, with the aim of altering the difficulty and intelligence of the world around him. *Dynamic Difficulty Adjustment* (*DDA*) or *Dynamic Game Balancing* (*DGB*) is the process of automatically changing parameters, game scenario and real-time game behaviour[B-5]. The simplified idea is to identify where the player ranks on a scale of experience and set the game around that level, always guaranteeing the right level of challenge and satisfaction, potentially for everyone.

There are different approaches and different ways to quantify the skill, or more properly the experience and level, of a player; and still others to alter their playing experience. It is intuitive to think of making opponents trivially 'stronger', but this often offers poor quality results; whereas it may be much smarter to make the opponent more 'aggressive' or more 'reactive' so as to not only increase the penalty for a wrong move, but the overall difficulty of the challenge; even to the point of changing it. This is just one example of the changes that can be made to the system, in particular a change to the system outside the player (the opponent); but it is also possible to favour or disfavour the player from within, by altering the quality and quantity of supports and power-ups encountered.

Various considerations and analyses on the role, objectives and solutions of adaptive or non-adaptive difficulty in games will be proposed later in this thesis. The study that we propose to carry out starts from the analysis of the difficulty in Game Design, of the flow and of the solutions or palliatives proposed in the literature; afterwards, some of the most known Computer Science strategies suitable for the implementation of the adaptive difficulty will be exposed, and some of them will be compared for their most suitable use and context.

Chapter I

The issue of difficulty

The brain desires to be stimulated; it wants to be challenged and, more than that, it wants to succeed and to derive that sense of satisfaction in having overcome an obstacle. But more than it wants to succeed, it fears failure and the resulting frustration; Goethe said: *"We are used to see that Man mocks what he never comprehends."*[1], and although this may sound like an interesting literary aphorism, it perfectly sums up the behaviour of the human mind.

It looks for patterns in what it encounters, and when it cannot understand or make a pattern its own, it scorns and scoffs. One could open an endless dialogue on how this behaviour is at the root of so many problems and closed-mindedness, from the most innocuous disdain for video game culture to the most severe xenophobia and more; for the purposes of this thesis, Raph Koster's definition of what people call 'noise' when listening to new music is interesting: *"Noise is every pattern we do not understand"*[B-6].

The mind does not immediately despise every new thing it encounters, on the contrary it can sometimes become fascinated and intrigued by a new challenge; after a few attempts to understand a pattern there are two possible scenarios:

1. The pattern begins to make sense; the brain feels it understands and, more importantly and stimulatingly, can assimilate the pattern. Once learned, it gets satisfaction from recognising it later and knowing how to deal with it.
2. The brain repeatedly fails to make sense of this pattern and begins to shut itself off from it; it is easier to think that the pattern is wrong and not the mind that has failed to understand something. And here begins the contempt.

The first scenario is the most interesting one; once a pattern is recognised, it will become increasingly easier and mechanical (automatic) to behave in its presence.

[1] *Faust*, Johann Wolfgang von Goethe. 1808.

The brain craves this simplicity; to a large extent we depend on it and make great use of it in our lives: driving, cooking, or walking to a known area; these are all actions that we complete following a routine that requires no mental effort. However, the more familiar a pattern is, the less stimulating it is.

In a way, the mind behaves as if it wants to consume its own game, even though it knows that once completed (once the mechanics are assimilated) the game will never be interesting again. *"Every man destroys what he loves"*[2], as Wilde said in his time; it seems illogical, but the mind rushes to the finish line before realising that once it has crossed it cannot relive the experience.

The fascination of video games, and games in general, is precisely in this clean, abstract and iconic sequence of patterns and schemes ready to be observed and addressed by the brain. The player voluntarily puts himself in front of challenges that can sometimes be nerve-wracking and require commitment; a brilliant definition of 'game' was given by Bernard Suits: *"The voluntary attempt to overcome unnecessary obstacles"* [B-7].

It becomes 'game', in the common sense of the word, when these patterns become challenges. The player is confronted with the need to understand the obstacle or goal (or rather the developers' way of achieving that goal) in front of them; and then to become an expert in that scheme in order to overcome increasingly complex, and therefore more challenging, versions of it.

This is the role of difficulty in games, to lead the player to engage but never to give up, to stimulate but never mortify him. In reality, such a defined vision of difficulty is quite recent, since, as mentioned above, there is nowadays such care and attention in the development of game contents; whereas in the early days, video games were mostly developed by mathematicians and physicists.

This was the time when the difficulty presented to the player was to all intents and purposes a challenge; the aim of which was to achieve a certain degree of confidence with the new scheme proposed. These games were often characterised by a strong geometry or use of dimensions, from the older grid-based games to the

[2] *The Ballad of Reading Gaol*, Oscar Wilde, 1898

more 'recent' titles such as ***Kula World***[L-21] whose extensive use of three-dimensionality and rotations provided a very interesting and demanding challenge, but that felt too frustrating for many players. Despite being criticized for its lack of ‘replay value’[S-1] shortly after its publication, the game is, twenty years later, considered among the historical titles for PlayStation and one of the hardest to complete[S-2].

Nowadays, the gaming experience is no longer only about the challenge it offers but about the experience the user has; the player is no longer the adventurer who delves into the plot and the developer's idea, but it is the developer, through the game, who takes the player through the plot and the challenges it has to offer. This is probably the biggest change brought about by the discipline of Game Design. But with the vast expansion of the circle of players, calibrating the challenge has become an extremely delicate task.

For starters, the **difficulty setting** can be ***static*** or ***dynamic***, and this concept draws the first cutting plane in this thesis. A *static* difficulty is a setting of challenge level that does (not) and cannot be altered (by some metrics) during the game; this does not mean that the challenge of the first minutes of game will be the same as the challenge encountered later on, but that the curve of increasing effort required is drawn at the beginning or manually set (details on chapter 2). On the other hand, a *dynamic* approach consists in a difficulty curve that evolves as the game proceeds, according to some metrics.

In a study conducted in 2009, the Ph.Ds. Hagelbäck and Johansson stated: *“It is often difficult to find a static difficulty level that suit the preferences of many players. The risk with static levels is that one difficulty setting is way too easy for a player while the next level is too hard. For example, Warcraft III*[L-46] *has been criticised in forums and reviews for being too difficult in skirmish games against computer players”* [B-8]. This intuition (widely shared) is the idea that leads many researches towards ad hoc difficulty settings.

Regardless of all the talk about the human mind, it is quite obvious to say that "playing is fun". This is as simple as it is only potentially true; it would be more correct to say, with due clarification, that “playing is enjoyable". The necessary clarification is that it is good to keep in mind that everyone has a different way of perceiving something as enjoyable; and that this same difference is changeable over time based on the current mood and need of each individual.

In addition to the character difference inherent in each individual and the variability of his or her emotional state, there is also the immense variety of possible ways of playing, the combinatorial possibilities of which offer an almost exponential domain of combinations.

With the expansion of the number of players in the world, the 'types' of players have also grown in number, and their differentiation can take place according to different criteria. In the following I propose to provide three different points of view which should be taken into account in order to understand on the appropriate calibration of the gaming experience provided:

- The different pleasures of the game,
- The different types of players,
- The different types of games.

I.1 Game’s stimuli

For the first category, a taxonomy widely appreciated and useful as a starting point is Marc LeBlanc's *Taxonomy of Play Pleasures*[B-9], that lists what he defined the ‘eight primary pleasures’.

- ***Sensation:*** the pleasure of the senses. Visual, auditive and tactile alike (typical of miniatures and controller vibration).
- ***Fantasy:*** the pleasure of imagining. The possibility to bend the rules of reality and to open the mind to whatever it is portrayed.
- ***Narrative:*** the pleasure of living a story. Just like with books or movies, in games the player has the possibility to actively live the vicissitudes and struggles of the characters.
- ***Challenge:*** the pleasure of putting yourself to the test. This pleasure can also be self-sufficient for some titles and is in different forms present in practically every game. Challenge is where difficulty plays its role; and

being nested deep within the game essence itself require particular handling care.

- ***Companionship:*** the pleasure derived from any form of social interaction: cooperation, competition, community, fairs, conventions, forums and also the basic chatting with friends about a game you both know; in general, any activities related to it even while you are not actively playing.
- ***Expression:*** the pleasure of self-expression and even self-discovery. Lately emerged, bringing a large proportion of users with more artistic traits closer to games; it comes from the possibility of creating and customising. It became even central to several immensely different games such as ***Lego***[L-23], ***The Sims***[L-43], ***Minecraft***[L-29] and ***Animal Crossing***[L-2]; and it's characterized by immense re-playability.
- ***Discovery:*** the pleasure of dwelling in uncharted territory. Generic pleasure derived by finding something, either by searching it or not; this could be a lore fragment, a mechanic, a portion of the world or a secret (easter egg).
- ***Submission (or Evasion):*** the pleasure of abandoning reality and letting oneself go by immersing in the game. Internally, this experience presents two profoundly different modes of achievement, closely related to the player himself: escape as "*non-thinking*", typical of casual games and escape as *"distraction"* or *"focusing on something else"*, typical of puzzle or logic based games.

The author itself specifies that this is not to be read as a check list, the taxonomy is a general guide useful to navigate in the players' experience. Some of these can be core pleasures of their games, such as *Companionship* for team-based games, *Narration* for lore-oriented games, *Challenge* for logic puzzles and *Expression* as mentioned before; while other are considered accessories: *Sensation*, in example, cannot make a bad game into a good one, but can make a good game into a better one[B-10].

LeBlanc's list, although considered a standard, still has its shortcomings. Jesse Schell, another most important name in game design, proposes and reported several noteworthy additions, including: ***Chance, Triumph over Adversity, Cleanliness***

(cleanliness of a scenario)***, Humour, Excitement, Surprise and Pleasure in the joy or misfortune of others*** and many others.

A very common and often taken-for-granted pleasure is the pleasure of ***Completion***: the drive to get to the end, to finish what one has started. Although it could be assimilated in *Cleanliness*, *Completion* involves many analyses and has brought out certain particular types of players and is the driving force that leads players to finish a title, a level or a game.

It is perhaps the most common sensation for all players and the simplest; and it stems from the same nature, already mentioned, of the human mind to want to face a scheme, learn it, assimilate it until it becomes automatic. Eventually the mind will tend to ruin its own game when it has nothing (new) to offer, but until then the desire to continue represents a common and widespread primordial pleasure.

The game’s stimuli are not to be underrated, as they are the legacy of the game itself; many games gained a place of honour in the history of games thanks to the stimuli they offered. The timeless ***Pac-Man***[L-33] is worldwide known as a classic of the arcades and so is the iconic ‘waka-waka’ sound and the undrawn link between the edges of the map, to the point that it is generally called *Pac-Man effect* even in other contexts.

On the same extent, ***MediEvil***[L-27], while started as a small independent project[S-3] grew and became a ‘real lost gem’[S-4] and a model for many Action RPG to come. The gothic, macabre scenario and the journey to redemption merged with the funny facial mimic of the almost-speechless protagonist are considered an inspiration for many titles to come, even very recent, such as ***Little Nightmares***[L-25] and ***Hollow Knight***[L-18].

I.2 Games and gamers

In contrast to the pleasures derived from the gaming experience, the categorisation of games and players is much more intertwined. In a broader view, gamers also differentiate themselves according to the types of games they enjoy. In 1996, Richard Bartle proposed a two-axis taxonomy to place players of multiplayer games, originally for MMORPGs[3] and MUDs[4], but now also used for single player experiences.

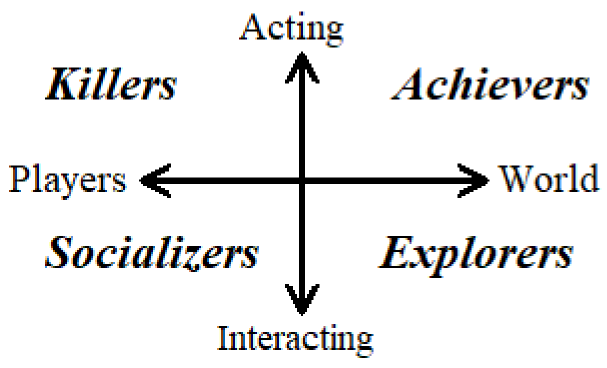


Figure 1 Bartle's taxonomy

With the support of this simple scheme Bartle summarised the behaviour of the players with two questions:

- What do they do? Do they act or interact?
- Towards whom/what? The game world or other players?

The four possible combinations reflect sufficiently comprehensively the typical users of role-playing games, and with some flexibility it is possible to generalise.

Less concise is the division with respect to the types of games sought by different users; beyond the division between board games and video games (in which we are most interested for the purposes of the thesis) several interesting division planes are:

- Single player or multiplayer
- Challenging or relaxing
- Adventure or matches

[3] **M**assive **M**ultiplayer **O**nline **R**ole **P**lay **G**ame.
[4] **M**ulti **U**ser **D**ungeon.

Many others would obviously follow, but it is useful to spend a few lines on the variants mentioned above. Internally multiplayers are further divided into competitive and co-operative; in the former there is usually little room for adaptive difficulty, while in the latter it is possible though difficult to calibrate correctly. Competitive multiplayer, if they want to balance the challenge, usually go for a dynamic difficulty approach, usually giving better resources or environments to the weaker player.

The main problem with adaptive difficulty in a co-operative multiplayer context is that the challenge is unique for the whole group (party), which implies that an overall level of this will be calculated, usually the average or the sum of the individuals. This will inevitably result in an easier challenge for the stronger player and a more complex one for the weaker one, whereas the ideal goal would be the exact opposite.

Similarly, if the experience sought is a relaxing one, there will be little application for dynamic difficulty systems. Take into consideration the class of players of *casual games*: titles typically for mobile devices consisting of short and/or interruptible games in which the game dynamics soon become almost a reflex.

Players in this category are not looking for a complex and challenging experience, often they are not even interested in the narrative events or in the emotional involvement with the game itself; the pleasures they try to stimulate are mainly escapism (as already mentioned), cleanliness and completion.

Here adaptive difficulty finds little application, but the analysis of game difficulty is still crucial. This type of gamer does not want to engage, wants a few mechanics with many variations; take the popular ***Candy Crush***[L-4] as an example.

Finally, games based on an entire adventure to be completed provide a more suitable environment to observe the player over time and understand their level. The same could be said for games based on matches, but these are often not played by the same user, leading to the risk of biased analysis; however, the addition of personal accounts to the console is opening up dynamic difficulty to this category as well.

In the following, I present four examples of titles that either present or not (but looks like they do) some form of dynamic difficulty in different contexts, analysing their choices and motivations in order to understand the classic design issues in the development and especially in the calibration of the difficulty of these games.

I.2.1 Examples and non-examples of dynamic difficulty

An early example of the problem of dynamic difficulty calibration in multiplayer contexts is ***Monster Hunter: World***[L-30]¸ in which players hunt against monsters that are usually much bigger than their avatars; although it was designed for party hunts, the game is also perfectly playable in *single player* as the creatures' statistics vary.

The system originally featured only two different difficulties: single-player and multiplayer; the modifications between these are as follows:

- Health increased to 180%.
- Damage increased by 8%.
- Frequency of status alterations increased.

Here, the main difference was the health, which was almost double the amount for single players; considering that parties could be up to 4 players, the experience for couples was the hardest overall.

The great flaw of this dynamic difficulty system was, ironically, the lack of interactive dynamism throughout the game; it is common in games like these for some players to be forced to disconnect, leaving the party with a few missing players. Apart from the strategic damage caused by the absence of a member, the game did not alter the creature's statistics even if only one participant remained. This was later corrected in an update that also introduced a third difficulty level specifically for pairs.

This first example of dynamic difficulty, as can be seen, does not depend in any way on the experience of the players; in fact, this issue is common to the very category of multiplayer titles, possibly even non-digital ones. Even establishing ad hoc statistics for each possible cardinality is not enough and creates the need to add some form of support for less numerous groups; an inherent example is the board game ***Descent: Journeys in the Dark (2nd edition)*** [L-10] in which 2-player parties can make use of an additional offensive action compared to the standard rules, or enjoy a light healing to compensate for the impossibility of creating a complete group strategy due to the scarce presence of players.

While the term *dynamic* used for *Monster Hunter World* is a bit stretched but accepted, the same cannot be told for *Descent*. The main difference is that, in the first, the number of players can vary during a game session; while in the latter the number of players is set at the beginning of a *mission* (technically even at the beginning of the whole *campaign*[5])

These first two titles were used to introduce to the problem of hypothetical dynamic difficulty in cooperative multiplayer context; although *Monster Hunter World* is poor in term of *dynamic difficulty*, its example is significative due to the scarce applicability to the genre.

The following example allows instead for a peculiar consideration, offering an insight into the problem of dynamic difficulty in the context of competitive multiplayer games. As mentioned above, these games too offer little room for difficulty balancing since the difficulty depends mainly on the experience of the opponent.

However, an interesting idea is the strategy implemented in ***Wii Sports Resort***[L-46], particularly in the *Bowling: Spin Control* mode. This involves a 10-pin bowling game, classically consisting of 10 rounds, but with obstacles (static or dynamic) placed in the lane. The peculiarity of this mode is that the game evaluates, for each player, the skill of the last shot to modify the difficulty of the following. With the exception of shots 1 and 10, each shot has three levels of difficulty: *Beginner*, *Intermediate* and *Expert*. (Shot 1 is standard, shot 10 is clean of any obstacle).

It is peculiar that each player is given a different difficulty despite all being rated equally in the leaderboard; theoretically speaking this makes the game unequal

[5] In tabletop RPG games, often a group of players (called *party*) play together a succession of mission that could last months or years, without changing character. From the beginning to the end, this continuity of *party* and mission is called a *campaign*.

between players but at the same time allows less experienced players to be able to compete with teammates of different level; similarly, it provides more experienced players a consistent challenge even if the opponent is not up to scratch. This is, however, almost inevitable in a competitive multiplayer context.

The last example departs from the previous ones in terms of context and age; but instead takes up the concept of challenging games structured in levels (an honest middle ground between the overall experience of an adventure and the single independently completed matches).

Crash Bandicoot[L-7], becoming a classic almost immediately due to its success, introduced an adaptive difficulty mechanism already from the second chapter of the saga. The main reason for this was already then the varied experience level of the players who played the title, which was in fact often played by a more adult audience but less agile and responsive with the joystick (PlayStation controller).

The main cause of game over in the game is the life of the protagonist, who with a single damage suffered dies if deprived of power-ups. These power-ups (called Aku-Aku in the game) can be accumulated up to 3 times and each one protects against a single damage source except falling (which causes instantaneous death). The game developers themselves commented as follows:

"We had realized that if a novice player died a lot of times, we could give them an Aku Aku at the start of a round and they had a better chance to progress. And we figured out that if you died a lot when running from the boulder, we could just slow the boulder down a little each time. If you died too much a fruit crate would suddenly become a continue point. <u>Eventually everyone succeeded at Crash.</u> Our mantra became help weaker players without changing the game for the better players. We called all this DDA, Dynamic Difficulty Adjustment, and at the time the extent to which we did it was pretty novel. […].

Good player, bad player, everyone loved Crash games. <u>They never realized it is because they were all playing a slightly different game, balanced for their specific needs.</u>"[S-5]

The above comment exhaustively sums up the idea and the type of support (*Comfort Policy*, which will be analysed later); the only unnamed aspects of Crash Bandicoot's DDA system are two, both of which are extremely interesting as considerations: Firstly, the game tends to remember which levels the player had difficulty in so that the latter can be prevented from having to die again several times before the level settles down to a suitable difficulty; this of course also entails the need to be flexible with respect to the user's latest performance (which will probably be improved) to avoid falling into the opposite mistake. Secondly, no reference has been made to the special and secret levels, which in the game are absolved of such shrewdness because it is assumed that the player, who has put in enough effort to find these levels, deserves and craves a challenge at the fullest.

Chapter II

The game design perspective

What has been said so far concerns a general view of the gaming experience; as far as difficulty is concerned, and how the gaming experience varies according to it, a few more words are in order.

From a more technical point of view, the difficulty is analysed and studied in Game Design, and then actually implemented in Game Development. The designers' task is to choose what kind of game experience they want the players to have, but this choice is made only after an analysis of the "typical player" they want to approach; however, a not (entirely) predictable measure is the player's skill.

Three main factors should be taken into account in such an analysis:

1. **Experience with this title:** The player could have never played this specific title before, so it <u>cannot be assumed that they are good at it</u> to begin with; or, they have already played it and would like to replay it.
2. **Experience with the game genre:** The player may have played a similar genre before: a prequel to the title, another RPG or FPS; and therefore has experience and dexterity with the mechanics, or, they have played different genres and it is the first time they are approaching this particular genre.
3. **Experience with games:** This may be the first game the user actually plays and may therefore feel "inept" from the start, or, he hasn't touched a game in years.

It follows that the game, properly balanced, should be challenging and difficult enough for anyone who approaches it. In particular, the first point suggests that the game should be replayable, and in this perspective the difficulty is central (otherwise there is a risk of giving a repetitive and, soon, boring experience). The second and third points instead make it clear that the game should be easily understandable for the inexperienced, and, from a balancing point of view, the difficulty curve should start very low in these cases.

II.1 The *Flow*

The optimal gaming experience is different for each player; this difference is influenced by several factors including the 'skill' of the player himself. Although it may slightly vary, the perfect difficulty of a game with respect to a player approximates around their level of experience (with the game itself or with similar mechanics).

Given this, we can start thinking about the very concepts of 'player experience' (understood as 'skill') and 'game difficulty'. Both are values that, although we do not have a defined unit of measurement to quantify them, we can imagine as ordered values; presumably in numerical terms they would be non-negative values and (theoretically) without an upper limit.

With these assumptions it is straightforward to represent these two values on two orthogonal axes and to study the effects of possible combinations.

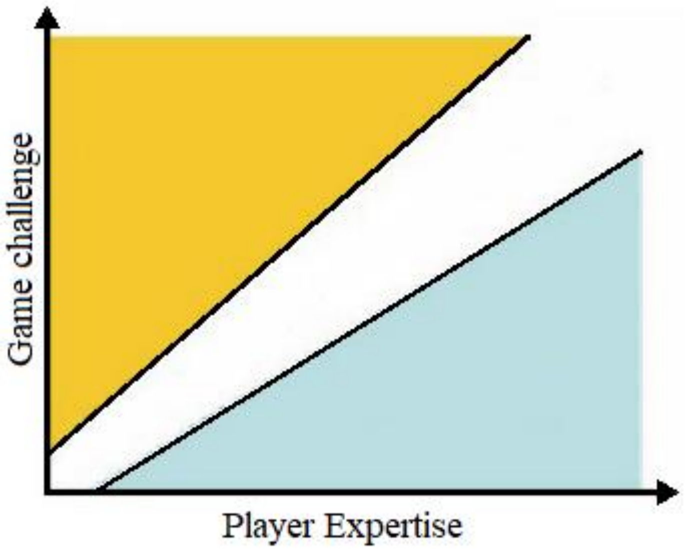


Figure 2 Game challenge and player expertise chart

In this first version, the proposed graph identifies three zones:

- At the top, the difficulty of the challenge is greater than the player's experience. This usually results in a more stressful gaming experience.
- In the middle, a neutral zone in which difficulty and skill are balanced. This is the zone in which most users find the game enjoyable.

- At the bottom, a zone in which the player's skill outweighs the game's difficulty. Here the player no longer perceives the challenge offered; initially the game becomes relaxing in these terms, only to become boring.

Interestingly, the player runs as far to the 'right' as possible, trying to increase his skill and overcome the difficulty of the game; this appears to be the user's goal to 'win'. But the reward beyond the threshold of game difficulty is actually unpleasant and often leads to abandoning the game with the classic thought "it has nothing more to offer me".

Obviously, the player's skill is bound to grow and that is why, as the game progresses, the challenge grows more and more. This is also the reason why, at the end of the main story of the game, you will encounter optional missions that are considerably more difficult than even the final mission of the game.

By analysing these phenomena, game designers have realised that the optimal condition in which to place a player is the *Flow*. This concept was conceived by the Hungarian psychologist Mihály Csíkszentmihályi in 1975[B-11] in relation to a state of involvement and focus on a goal for a user, with a sense of gratification not only in achieving this but also in just carrying out the task.

Flow was born as a psychological concept but soon proved suitable for various other concepts such as sport, research and, more recently, games. Csíkszentmihályi himself, in an interview in 2002, defined it as: "the feeling of complete and energised concentration in an activity, with a high level of enjoyment and satisfaction".

From a gaming perspective, flow can usually be achieved in that central area and is usually represented by a wavy line. Looking graphically at the resulting graph, the choice of the wavy line expresses an additional piece of information: the flow in the gaming experience is not an immobile state, but is reached through the continuous alternation of a slightly higher difficulty than one's own abilities and a sense of relaxation and satisfaction in having overcome this difficulty.

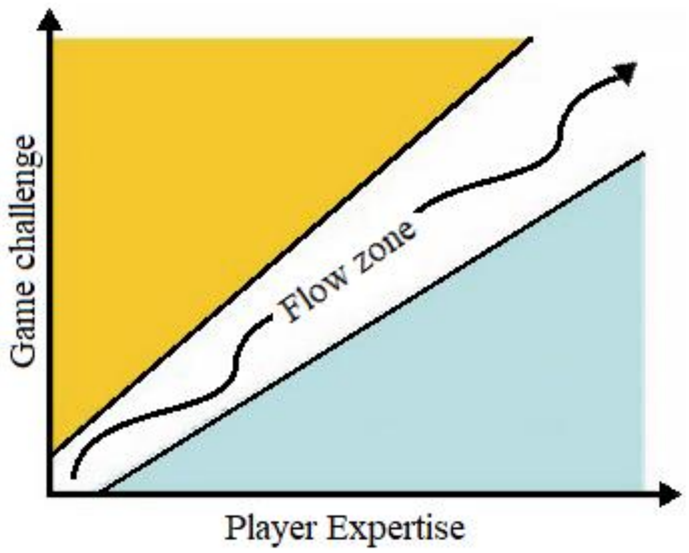


Figure 3 Flow zone chart

This graph offers two different visions: on the one hand, the intuitive correlation between the level of challenge and the skill of the player; on the other hand, one can imagine the increase of these values as a progression over time. When a player approaches a new game, his skill is close to zero (some mechanics similar to other games may start the player with an advantage, as in FPS); as time goes by he will become more and more expert and that is why the challenge gradually increases over time, accompanying the player in an improving path towards the complete dominance of the game itself.

In this second perspective, the Flow Zone graph also represents the increase in difficulty as the game progresses.

II.1.1 Alternative interpretations of the flow

Flow, its understanding and research, have become cornerstones of game design and different scholars have presented a slightly different view of it over the years. It is important to understand that two arbitrary users find the game experience optimal for them in different ratios of skill and difficulty; but there are other factors that can further increase these differences.

The first interesting example of an alternative view of flow was proposed by Mihály Csíkszentmihályi himself:

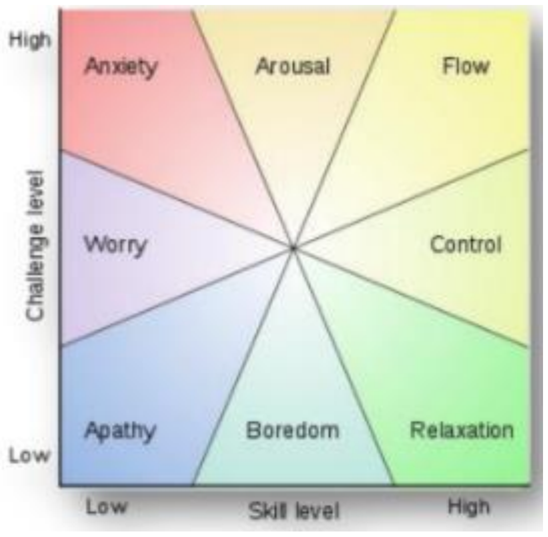


Figure 4 Flow zones proposed by M. Csíkszentmihályi

Its graphical presentation divides each of the two axes into three zones: low, medium and high; and shows the user's reaction according to the correlation of these. The graph shows that the flow is only reached when the level of challenge is high but so is the gaming experience; while when both are low the user is in a state of apathy. The most undesirable are, however, the zones characterized by the greatest difference among the axis, that would eventually lead to Frustration or Disinterest.

Other Flow Zone studies have focused not on the emotional state of the player, but on the type of experience sought. The reason designers actively focus on keeping players in the Zone is because it makes these want to keep playing.

Creating an environment that is stimulating, challenging, non-monotonous, rewarding and interesting is the key to laying the foundation for the desire to play again, recommending it to friends or rating it positively; or playing other games from the same saga or development house.

An individual's flow zone depends as mentioned on the game difficulty and experience level, but also on other factors that differ from individual to individual. Balancing a game in such a way as to keep each type of player in his or her zone is practically a utopia and this is where adaptive difficulty meets flow studies, presenting itself as a possible solution for all these dilemmas.

One of the most dedicated designers to the application of the flow to games, Jenova Chen, stated: *"Designing a video game is very much about how to keep the player in the Flow and eventually be able to finish the game. Therefore, the game system needs to maintain different players' experiences inside the Flow Zone. [...]. Unfortunately, like fingerprints, different people have different skills and Flow Zones. A well-designed game might keep normal players in Flow, but will not be as effective for hardcore or novice players."* [B-12]

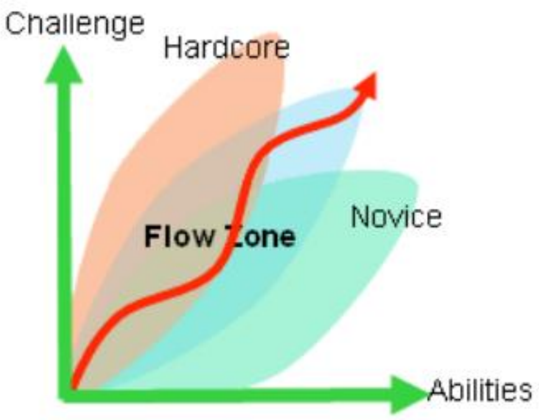


Figure 1. Different Flow Zone in accord to players' expertise (J. Chen)

In his thesis, Chen also discusses Adaptive Difficulty and proposes a system based on explicitly asking the user for feedback. In this attempt, he implemented ***traffic light***: a simple game with minimal interaction. The game objective is to click as late as possible before the traffic light turns red; achieving a positive score two out of three times, would allow to move to the next round. Between each round the player is asked whether he/she prefers to play faster, slower or at the same speed.

His study offers very important results: first, the game flow was extended by an average of 1-2 minutes to 5-12 minutes instead and given the simplicity of the game this improvement is quite significative; however, the explicit and periodic adjustment question broke the player's Flow and immersion. It started offering the player a sense of control, but eventually reduces the player's involvement.

In his thesis, Chen proposed another game that was meant to work on the players' flow zone merging the dynamism in the gameplay itself. This second approach shall be further treated more in details in paragraph II.2.2.

A further difference in individual flow zones is the cultural difference in two different parts of the world. In East Asian countries, for example, thanks to the more widespread video game culture, a variant of the flow was born, slangily called "Oriental Flow" or "Arcade Flow", technically analogous to the classic version but more like an exponential function in form.

The idea is (as in fact happens in Arcade games) that the difficulty increases exponentially with time, requiring a considerable level of concentration and skill. The game is presented as almost or actually impossible, and the goal is just to last as long as possible. A classic example that has been successful all over the world is the famous ***Tetris***[L-38] whose speed of play quickly became unbearable, and which still allows for different variants[L-39] and numerous tournaments of players so skilled that they can play at impressive speeds.

Another mutation of the concept of flow over time is due to the increasing popularity of gaming entertainment. As it opens to a wider audience, expectations tend to increase and, needless to say, tolerance decreases.

Although it has never been defined as a precise area of the graph, the Flow zone is often described (and therefore interpreted) as uniform throughout the personal growth of the player... but this is not really the case.

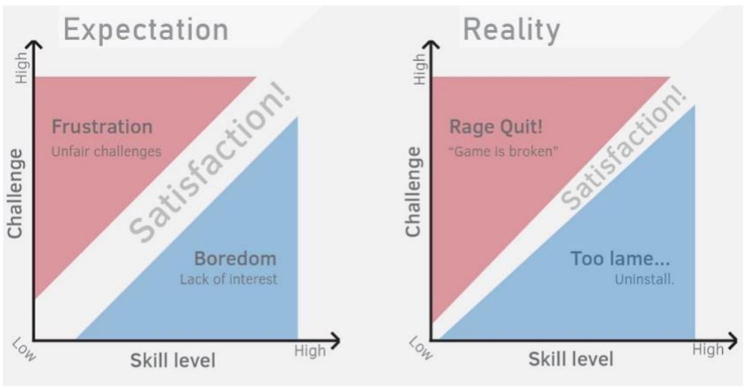


Figure 5 Satisfaction zone. Expected and reality

As can be seen from the images shown, the early stages of a game, when confidence with the mechanics is still low, are critical as it will be extremely easy to disappoint the player's expectations, while as skill increases, one will become (vaguely) more flexible.

Even more innovative and modern is the *Perceived Challenge Flow*, a study which starts from the more psychological foundations of this topic and which proposes and analyses a balancing system based on the perception of the user rather than on the actual difficulty.

In his work on the *Perceived Sense of Challenge (PSC)* [B-13], Shikhar Juyal, analyses this particular possibility in a specific phase of Game Design: the Level Design, i.e. the planning and design of a level (or area/zone). The insight, which proceeds along the path traced by Csíkszentmihályi, is in Juyal opinion:

"*PSC as a concept tells us to prioritize the feeling of* ***'It felt very hard and challenging'*** *over* ***'It was very hard and challenging.'***".

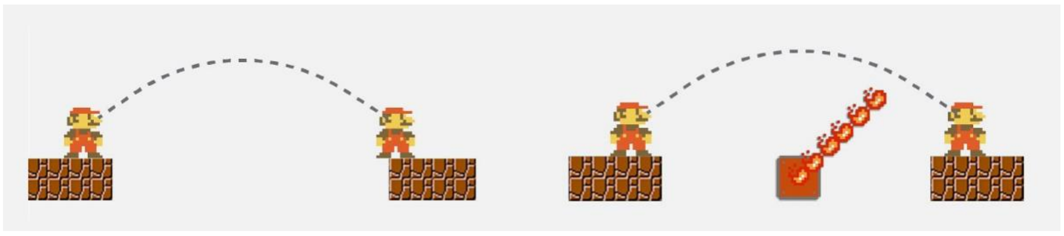


Figure 6 A simple jump in Super Mario (left) and the same with a fake obstacle (right)

The subtle difference between the two statements suggests that gratification in overcoming the obstacle does not derive from how hard the obstacle actually is to overcome, but from how hard it was perceived to be. This idea (greatly simplified here) opens the way to an interesting perspective; an idea to make the player feel that the difficulty is increasing without really changing it, and thus making them perceive a sense of improvement (and consequent gratification) when in fact the difficulty has not changed at all.

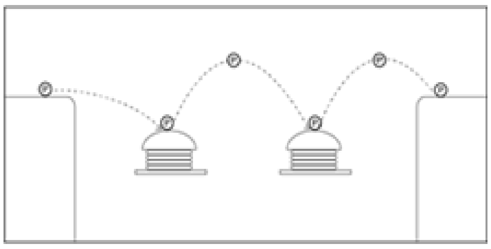


Figure 7 Perceived sense of challenge. Img. 1: Simple Jumps

The introduction of 'dummy obstacles' adds an extra pseudo-level of difficulty that can be used in level design and guarantees a high reusability of patterns.

While the addition of an obstacle, like the one portrayed in Figure ??, requires the extra effort to synchronize the jump with the vertical movement of the obstacle; the obstacle shown in Figure ?? is only aesthetic, and doesn't actually require to synchronize the movement; but eventually the player will feel the needing to concentrate on the obstacle too

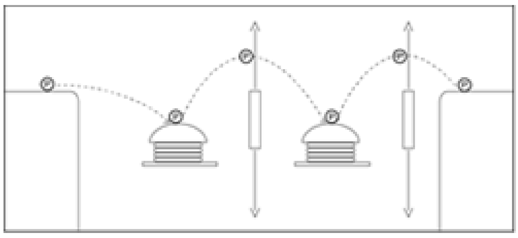


Figure 8 Perceived sense of challenge. Img. 1: Obstacle Jumps

The intermediate grade given by the PSC is an invaluable element for flow control, providing a (technically unjustified) great sense of satisfaction, personal growth and confidence in one's own abilities. Hitting any obstacle here would be a different way to fail, but not a different reason, quoting Juval itself, "*[the] player would never have reached the second bouncing obstacle and would have crashed nonetheless*".

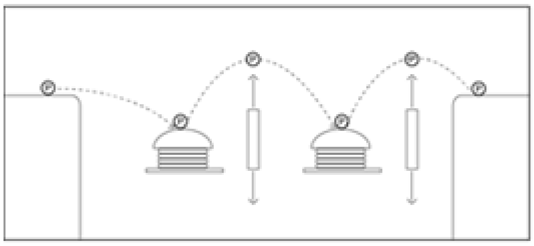


Figure 9 Perceived sense of challenge. Img. 1: Dummy Obstacles Jumps

The time needed for the user to improve and the impatience of the user to be 'good at it' can be stemmed and combined. Adaptive Difficulty Models can take this factor into account to gradually 'degrade' an obstacle to a purely perceived one; intuitively, this was also the idea (without the psychological underpinnings behind it) of the slowing down of the boulder in the previously introduced Crash Bandicoot levels: a danger that gradually became only of script and anxiety, but devoid (or almost) of actual danger.

II.2 Difficulty in game design

After all these considerations, it is clear that balancing the difficulty of the game is a key issue in its development. The issues regarding perfect balancing are varied and only a few have already been introduced.

First of all, the central issue is the different skills of the players and the different individual perfect game experience, as discussed so far; the difficulty must not only be fair but must also grow in step with the player's experience in order to avoid the latter getting bored or frustrated. Moreover, as Jenova Chen's study has shown, it is important that the player remains 'immersed' in the game for an optimal experience; any periodic feedback, or passing through the typical settings screen, causes a detachment from the game world that 'breaks the magic' built around the player immersed in the story or adventure.

The issue becomes even more complicated when the type of game and the number of players start to be taken into account; also the type of involvement and entrainment in the game varies if the main focus is on the storyline or puzzle solving, or even challenges. The game also needs to be properly calibrated not only in terms of difficulty, but also in terms of a number of other features that in turn influence the gaming experience.

To approach balancing in games appropriately, it was decided to outline the important aspects that identify the different types of challenge and balancing. This topic in Game Design is called 'balance'. Again, the classification that is given here was proposed by Schell.

II.2.1 Balance in games

Among the different aspects to be properly balanced, the flow appears under the aspect of 'challenge and satisfaction'; next to it, Schell mentions:

Equity: That is, balance between the potentials of the players and possibly the AI. It is important to understand that it is not necessarily, nor fundamental, that the potentials are the same but that they are appropriately balanced with respect to the challenge to be proposed. This phenomenon is typical of *MOBA*[6] games, such as the notorious ***League of Legends***[L-22] in which every playable champion has a set of different skills and statistics.

[6] **M**ultiplayer **O**nline **B**attle **A**rena.

An example of intentional inequity in a multi-player context has already been presented with *Wii Sport Resort*[L-46], while between player and AI a good example is ***Starcraft II***[L-35] where, at high difficulties, the opponent controlled by the system is deliberately in possession of more information than the player.

Rewards and punishments: important to make the player feel rewarded for their efforts and punished for failures, but without overdoing it and risking the player giving up. It is an extremely sensitive aspect because intuitively, the player who has successfully accomplished his/her task needs less facilities than the one who has failed, but at the same time it is the one who will be rewarded and vice versa.

An example of a game with problems in balancing *rewards and punishments* is the aforementioned *Descent*[L-10] (tabletop RPG) in which players often have to decide between following the mission's storyline or exploring in search of side events (often being able to do only one of the two due to limited amount of rounds); avoiding the latter usually provide immense benefits for the opposing faction for the rest of the game (punishment). Since both the main and secondary storylines rarely offer any real reward at the end, players often find themselves losing deliberately to reduce the damage of an overly punitive system.

Some DDA systems operates on this aspect to affect the game experience without directly act on the player or the world.

Freedom or Guided Path: both of choices and of the world exploration. Although, the guided path offers more control to the designers, it is less 'realistic' and popularly less appreciated by the players; recently, more and more titles adopt the open-world philosophy and are heading down the path of freedom of choice, a most iconic example is ***Life is strange***[L-24] in which every chapter starts with the following sentence: *'Life Is Strange is a story based game that features player choice, the consequences of all your in game actions and decisions will impact the past, present and future. Choose wisely…'*; or of both world and decisions, as in ***Cyberpunk 2077***[L-9].

Freedom, moreover, can also be intended as accessing to something or some place that you don't need in that moment, anticipating events; this could be even a issue in event-based such as the extremely recent ***At dead of night***[L-3], in which an early run for all the artifacts could significantly simplify the game, allowing the player to avoid running around for them when they are needed (extremely convenient because running around is the ‘dangerous part’ of the game).

Determinism and randomness: that is, 'how much randomness should be present and how influential should it be?' Randomness opens the game to an interesting possibility of balancing between different players, bringing the more experienced closer in potential with the beginners; reducing the so formally called ***skill gap***. Intuitively it presents itself as punitive to more experienced and skilled players, thwarting possible strategies; and as an opportunity for beginners.

Nevertheless, the randomness potential does not end here: in ***Undertale***[L-44], for example, at the beginning of every *run*[7] a random number between 1 and 100 is extracted; this goes under the name of ***fun*** and decides which (if any) easter eggs will happen (or could happen) during the playthrough. *Theorizers*[8] will have to play the game many and many times and, even more so, will have to communicate with each other (mostly on forums) to assemble the whole picture (Companionship stimulus).

Same happens with ***Five nights at Freddy’s***[L-14] saga, where, after a death-screen, occasionally a short animated sequence, a sound or a minigame that could unravel something about the obscure lore; but this is completely random and again will require coordination between players and a lot of attempts, giving the game a high replayability over the years and a huge meta-game potential.

Other concepts would follow, but what is most important to specify is that: it is not necessary to find a balance between these aspects by equilibrating both sides; but to choose how much and what to unbalance to obtain the desired result. After all, a perfectly neutral and balanced game is not even that intriguing in the long run.

[7] Different playthrough of the game, from start to finish.
[8] Players dedicated to unravel the secrets and the hidden lore in games

II.2.2 Traditional approaches to difficulty setting

Over the years, new methods have been invented to create systems that are not only of the right difficulty, but that can somehow accommodate a wider range of players.

The evolution of the solutions adopted over the years shows an increasing interest on the part of designers and developers in centring the difficulty range around the general player, and not to bring the player into the already existing difficulty zone; and, even more, to focus the experience around the player and not to make the player venture into the designed experience.

Parallel to these objectives, another important factor not purely of design was taken into account: replayability. As already repeated previously in this paper, the mind tends to lose interest in a game (in a broad sense) once it has been completed; this causes many titles to be played only once and then turn out to be monotonous and no longer stimulating (especially in the case of plot-based games). Difficulty settings have had to take this important issue into account and have presented interesting strategies, making games re-playable decades after their release. Many of these solutions had the common idea of unlocking new contents only at the end of a story line, or different possible storylines.

A first, rudimentary but classic solution could be called the **game's intrinsic difficulty** (or **implicit difficulty**); this approach involves the development of an almost standard game experience among all players. It is the responsibility and interest of the user to be, or become, at least as good as the game requires. This design choice can easily be found in many older or historical titles, such as the classic ***Super Mario***[L-36] which, both in the original versions and in modern reinterpretations, does not have several levels of difficulty but only one.

The experience derived from this design choice is usually homogeneous among players and is perceived as 'challenging' by the average. The first major problem is the inflexible difficulty perceived by the group of users who cannot get through the game, who in the worst cases tend to abandon the game. The second problem, which is just as serious, concerns the players who do manage to get through the game, but tend to get bored when their skill exceeds the required.

The inherent difficulty shows no effort towards replayability of the title, since between one game and another the experience will be practically the same. Historical titles (which implement this difficulty) that are replayed years later have usually offered remakes of the titles or have become so classic that they are reused out of sheer passion and dedication to the title.

Historically, some titles have not abandoned this approach, but have added secondary objectives at the end of the main story; this choice was made to justify a slight simplification of the average difficulty of the game (to facilitate the less experienced) while still creating much more demanding challenges (for the more experienced). Subsequent proposals in the literature almost always evolve in this direction and try to overcome problems concerning this first approach.

The natural evolution of this strategy is called **multiple-choice difficulty level**; this involves explicitly asking the user for the desired difficulty level for the whole game (or for the level in some cases). Although, unloading the choice of the appropriate difficulty onto the player is contraindicated, this solution is still adopted by a wide range of games, particularly for consoles, nowadays.

However, the potential it presents also turns out to be the main problem: <u>the player cannot know a priori how versed or skilled he is in the game before he starts playing</u>; typically, this decision occurs in an early phase of the game: before a new game starts or after a brief tutorial/introduction. From the most restrictive to the least, these games also can provide the possibility to change the difficulty during the game on specific occasions:

- Only at the beginning
- Before the beginning of every level/chapter
- Whenever a responsible NPC or point of interest is accessible
- Always (from the settings menu)

From the experience point of view, it would be foolish to change the difficulty during the game and would not only make the experience too fictitious but would also ruin all the efforts made during the development phase to create the right challenges that would allow the player to overcome the difficulties at certain points and to grow gradually.

With the emergence of console accounts, trophies (or achievements) linked to each game were also introduced: small challenges for the player that sometimes go beyond simply finishing the game, but encourage the player to discover secrets, perform side quests and replay the title. Among them, many games based on the multiple-choice difficulty tend to include trophies related to the completion of the story beyond a certain threshold of difficulty; this modern development has provided a new reason for this approach and, as detailed below, a new route for replayability. However, these trophies were not appreciated by the crowd and this strategy is being abandoned, example of this issue is ***The last of us***[L-40] which implemented such trophies in the first chapter of the saga, but removed them in the second one[L-41].

At the same time, however, it has made the possibility of changing difficulty during the course of the game even more forbidden, although some titles allow the level to be decreased in order to prevent the user from feeling frustrated when faced with a challenge beyond their abilities. The difficulty of completing the game is usually calculated as the lowest of the difficulties adopted (the last in the case of decreasing modifications only).

The aforementioned Jenova Chen, proposed in 2006 ***flOw***[L-15]: an experiment game to better study Flow using choice-driven difficulty: his idea was to implement a game based on layers of depth; the deeper you go, the higher the difficulty. The microorganism (the game's protagonist) can actively change layer by ingesting a red organism (going deeper) or a blue one (going higher); thanks to this system, the player can select the difficulty that suits him best.

flOw uses minimal control to open the door for casual gamers and non-gamers, but still leaves space for hardcore gamers to master it. It offers a wide range of gameplay from simply swimming around and eating to strategically evolving and intensive fighting. Players can customize their Flow experience naturally through the core gameplay, swimming and eating. By swimming closer to or farther away from other organisms, and eating different types of food, players subconsciously balanced their Flow experience[B-12].

With this second experiment, Chen was considerably more successful as the user's choice of difficulty became the game's own mechanics, not detaching the player from the game's reality (the main problem with the first experiment). The most used word to describe the game by the fans was ‘addictive’ and every designer could consider this a clear win. Thanks to his experiment, many games have made the difficulty part of the mechanics itself, thus bypassing historical problems (more on this later).

A natural, but complex, evolution of this approach is ***Dynamic Difficulty*** (and more recently ***Adaptive Difficulty***); which involves considering how the player is doing in order to dynamically alter the world and the challenge around it and thus provide an experience at their level.

Robin Hunicke, professor in Game Design, introduced the problem saying: *“Game developers iteratively refine game systems based on play testing feedback—tweaking them until the game is balanced.”* [B-14], this was (and sadly still is) the standard approach to difficulty design and balancing; she studied and experimented widely on DDA affirming the necessity to automize this process in regard with the player’s actual skill. Her work will be furtherly analysed more in detail.

Philosophically, this approach is perfect, and it also allows (if masterfully calibrated) to accompany the player in their growth, always placing the difficulty just above the player's capabilities. The player will still experience the ups and downs of the difficulty but will never find themself stuck in front of a mountain too High.

Obviously, the gaming experience is neither numerically quantifiable nor perfectly analysable; ergo what is used are heuristics, estimates and 'how they are doing' parameters, and since these are not absolute there is not (and perhaps never will be) a single, exact way to deal with them. The estimates themselves vary from game to game and genre to genre; they can usually be time spent, health lost, items used and countless others.

For the complexity of the purpose it would be necessary to analyse and compute all these parameters continuously, but this is intuitively too computationally burdensome, and would inhibit the development towards more complex artificial intelligence engines.

The solutions proposed in the literature are numerous and varied depending on the genre, the desired experience and the computational effort that can be afforded. Historically, a common solution consists in computing the progress between one 'phase' and another of the game (between two levels, between two zones, at each game over, ...).

There are, however, examples of statistics calculated continuously or on-demand for dynamic challenge adaptation, but this is mainly when both the statistics to be calculated and the adaptations to be made are (computationally) inexpensive. A popular example of this choice is given by the entire series of ***Mario Kart***[L-26] titles, again an example of dynamic difficulty induced inequality in a competitive multiplayer context; here the dynamic adaptation is provided by the objects obtainable from the classic crates: depending on the ranking position occupied at the time of crate collection, different objects are available.

The worse the position on the leaderboard, the worse the player's performance, the better and more significant the objects he or she can have access to; conversely, higher positions usually have access to weaker objects and are often helpless in the face of threats from opposing objects. Here the challenge is continually adjusted; but, as can be seen, the adjustment is not at all onerous, guarantees the desired fluidity and reduces the skill gap without being too unfair for the more skilled players.

The same title also features another dynamic difficulty mechanism via the other type of 'unfair inequity': that between players and AI. By continually assessing the ease with which players compete, the system will tend to introduce spontaneous errors in the NPCs if the player is experiencing difficulties, such as bumps against the edges, falling off the track, ...; vice versa, if players proceed at a fast pace by detaching the NPCs in the race, the latter will get a small increase in speed in order to make the challenge even more stimulating.

Although presented as an idealistically perfect idea, adaptive difficulty has not been appreciated by the public in its most innovative versions. Despite the efforts and innovation of the development and design teams, an interesting aspect that escaped the eyes of the innovators (and proved fatal) was the <u>feeling of being judged felt by users</u>.

The last, typical, approach is a huge step back in this direction and presents itself as another direction walkable from the implicit difficulty. The idea is to explicitly give the player a score that he can always see during the game, a **Completion rating** (or **completion grading**, or **performance grading/rating**); at the end of it, the game will keep track of the best score (highest mark) of the player in such level. The most classic application of this approach is the **3-stars grading**, that spread across the mobile games.

The reason behind the success of this method is in the user's implicit decision between 'completing the game' or 'achieving 3 stars on all levels'. This is beautifully thought because the level does not change, nor do the mechanics of the game; is the challenge that require always more and more accuracy by the user to achieve the highest grade.

<u>And it is not necessary</u>, if a user does not care of the challenge or of the marks he can easily just go on and keep playing. While, on the other hand, the perfectionist player will want all of the stars and will chase them in order to feel that sense of <u>*Cleanliness and Completion*</u>.

This method allows also to a huge variety; stars (or what behaves as such) can be achieved in different ways that merges them with the gameplay itself:

- thanks to a score such as in ***Angry Birds***[L-1], where every destroyed object or defeated pig gives an amount of score; only after certain thresholds a *star* is achieved.
- trough objectives as in ***Clash Royale***[L-6], in which every destroyed enemy tower grants you a *crown*.

- a mixture of the two as in ***Clash of Clans***[L-5], in which you obtain a *star* for every 50% of the buildings destroyed and an additional star when the *Town Hall* is destroyed.
- or even be directly collected during the game as in ***Cut the rope***[L-8], where *stars* are in strategic positions in the level that will require some extra care and timing to be collected.

The Completion grading system has nowadays been widely implemented; even games that still use the more classical multiple-choice approach are borrowing the ‘saving the best result’ pattern to benefit of its effects. Despite the undeniable design success, it is still not seen as the future of difficulty design but only as an improvement step and as a solution for casual games.

II.2.3 Dynamic Difficulty Adjustment

After all the words spent to introduce pros and cons of the other typical difficulty settings seems redundant to explain why the ***DDA*** (***Dynamic Difficulty Adjustment***) seemed to be the road to the future in game design. Both for its theoretically perfect idea and the fascinating and elusive concept, it interested many; in the first decade of the 21st century many studies were made first to try to see if adaptive difficulty could provide an overall better experience and later to improve such adaptive algorithms.

The problem now was how to make a game ‘balanceable’ on runtime. Minor adjustment such as the different power-ups in *Mario Kart* were not enough for the purpose; now the change had to be of the whole experience and as it was soon obvious, incrementing the enemies dealt damage (or any statistic for that matter) was a naïve solution.

This last, although naïve, solution gave actually good results in the popular title ***Final Fantasy VIII***[L-12]; but still did not provide a harder challenge for hardcore players, only a challenge ‘statically’ set in accordance to the party current level. As in many RPG, the numerical *level* value often defines the statistics of enemies and characters alike; and usually balance is designed in function of this value. While designing *Final Fantasy VIII,* the *Square Soft* had already experience in RPG games and level balancing; in previous titles of the series they noticed that the 'intended challenge' was when the enemies were a few levels higher than the player, but being the enemies level static this did not always occur.

The innovative idea was then not to decide for each enemy its level, but how far its level should be from the player's. Thanks to this strategy they managed to have the player always fighting enemies that were always stronger (or weaker) than him in the same amount across different runs. The only required system now was the one that could decide a unit's statistics based on the *level* value; this is a common design problem in games and the two most used solutions are:

- **Look-up tables**: handwritten in development and adjusted through testing. Every different unit (ally or enemy) will have one that will associate for every statistic the correspondent value for each level. Gives a hard direct control on balance and allows unregular scaling.
- **Percentage increase**: which only require the base value for each statistic (for every unit) and the percentage (which is usually constant across everything). Thanks to the uniform scaling factor, this method makes easier to balance across different characters for MOBA-like games. The (simplified) formula would be something like: $stat_{lvl} = stat_1(1+p)^{lvl-1}$, where $stat_x$ is the statistic's value at level x and p is the percentage increment greater than zero (usually between zero and 1) .

This approach gave *Square Soft* what they were seeking: a 'dynamically standard' game difficult overhead, and it also prevented a common 'cheat-ish' strategy in RPGs: *farming* (or *levelling*). This is a common strategy in all level-based (level of experience intended) games: if the game is too hard, go back and level-up and then try again… eventually you shall out-level[9] the challenge and will be able to proceed. By removing this possibility, *Final Fantasy VIII* required players to actually be strategically good; and while this makes the game more interesting, it could be frustrating for those players that, after repetitive failures, start hating the gameplay and refuse to try and change their approach; as poor as it is, this possible behaviour

[9] Slang gaming verb that means: "to be so much higher a higher level (than something) that it doesn't represent a threat anymore".

should be taken into account by designers. Luckily, the *Final Fantasy* public used to be very committed at the time, so the issue was less significative; nevertheless, the eighth chapter of the saga appears to be among the less played of the golden age (from Final Fantasy VI to X) by following generations.

Despite the criticism received, the opposite solution given by the absence of level scaling, still nowadays is subject to heavy critics. Generally, both the presence and the absence is subject to issues that impact over the game experience:

- Level scaling: prevents the mechanism of *farming*, prohibiting less hardcore players to ease the challenge and enjoy the lore without struggling on a difficult section.
- No Level scaling: The continuous killing of random enemies, or the completion of side quest, may cause the lore missions to become ridiculously easy. A modern example of this is in ***Nier: Automata***[L-32], where even having a few levels more than the enemies might cause the player to *one-shot*[10] every enemy encountered.

Following *Final Fantasy VIII*'s steps, many level-based games (mainly RPGs) implemented adjustment to this strategy that now goes under the name of ***level scaling*** (or just ***scaling***). This can be found applied to the whole game (as it was in *Final Fantasy VIII*) or to parts of it as a smoothing measure; from the least invasive to the most:

- No *level-scaling*
- Only main quest
- Main quests and Side quests
- Only after completion[11]
- Whole game

In the meantime, the survival genre proposed a more elaborate and less direct approach, that could alter the gameplay itself as a side effect; instead of changing

[10] Gaming term: mean to kill off a unit in one hit. Figuratively can be used to say that a unit can be killed effortlessly.
[11] Many games provide the possibility to *continue* a completed game, usually returning to the place immediately before the ending.

the characteristics of a unit it was considered the possibility of altering the surroundings of the world.

Whoever played a survival game knows that increasing the game difficulty does not only involve stronger enemies (with which would be possible to deal) but mainly cause supplies to become scarcer. In any survival title, objects like ammos and healing factor are crucial and it is already key factor of the playing strategy to ration them properly; should these become scarcer, the game would not be only straightforwardly harder, but the attitude of the player must adapt to overcome this dearth.

The general improvement idea for DDA was therefore in this direction, hitting the player not directly through enemies but sneakily on his inventory. A formal approach was proposed in 2005 by Robin Hunicke that performed a study on players' experience when dealing with DDA in a FPS context[B-14]. While starring an innovative adaptive state-based AI system called *Hamlet* (that will be later treated in detail) she presented the concept of ***policies***.

In her idea, different *policies* could be used to define the approach to adaptation depending on the desired player experience (*aesthetic goal*). She names three different control policies, explicating that it is nowhere near an exhaustive list:

- ***Comfort Policy***: to keep the player feeling challenged but safe; being generous on their inventory so that he could rely on a gadget or healing device.
- ***Discomfort Policy***: to challenge the player by limiting the supplies and driving up demands after certain conditions.
- ***Training Policy***: to stimulate the personal strategic growth of the player; starting as a *comfort* and gradually degrading to a *discomfort*.

Due to the silent (because of the critics received) implementing of DDA strategies in games, it is hard to tell if any game actually implemented the *Hamlet* system (beside Hunicke's project). However, the analysis of the effect of impacting the player's inventory over the gameplay is observable. A clear example of that is in ***Final Fantasy XV***[L-13]: where, differently from the previous titles of the saga, healing items could be used at anytime, even if characters were knocked-off. This made the player almost invincible, and caused them to radically change their game style only in the zones where items were disabled. Although it helped making the game more approached by new players, *Square Enix* tried to correct this in the following release of the saga[L-11].

Another approach, a bit more advanced and rarer in literature, could be to change the behaviour of the enemies, not their statistics; modifying the skill set or combos. For example, it could be interesting to design boss enemies with different fight states that not only vary during the fight according to their health (as typically happens in ***Kingdom Hearts***[L-20]) but also to provide same fight states that can only be unlocked over or beneath a certain performing grade for the player, making the boss fight <u>different</u> and more (or less) challenging.

The inconvenient with DDA, was that players did not perceive the adaptive difficulty as a way to provide the perfect experience for them, but felt 'judged' and 'cheated', as if a victory would not be result of their skills but as a guided outcome.

Despite the advantages that adaptive difficulty brought, players generally did not appreciate the feeling of being evaluated (especially when evaluated negatively). While dissatisfaction with the rating and adaptive mechanisms already existed among users, the situation escalated (and gained popularity) following the publication of the official guide to the popular title ***Resident Evil 4***[L-34] in 2006.

When players acknowledged that their victory may not be all due to their skill indignation spread; quoting Mark Brown: "*Many hardcore players are too proud to accept the helping hand and would rather beat their head against the wall in frustration than suffer the indignity of getting a free pass*".

Although *Capcom* never officially stated the presence of DDA in *Resident Evil 4*, after the tip from the official strategy guide it was hard to hide it; many players uploaded their playthrough videos on YouTube and clearly showed the lack of enemies and abundance of ammo in a game poorly started.

Despite the accusations, most player had (and still have) no idea on how does the DDA work and why; the few that actually appreciated it were mostly novice that found themselves capable of learning how to play at their own rhythm and players that actually found interesting the idea and <u>admitted that they did not notice it</u>.

The rage against DDA persisted and persists still; many games now prefer to conceal a DDA if implemented, fearing the public's reaction. It is very easy to find forums with questions like 'Does this game have DDA?', followed briefly by 'No, thankfully.. another game would be ruined otherwise' (this happens even on forums of games that actually have DDA).

This event brought the attention of research on another topic: 'Do people realize if difficulty is dynamically adjusted? How they react when they think it is?'. Of course, even in previous researches players did not know if the game they were playing was actually adjusted or not; so, their results can still be useful to analyse this issue. Again, Robin Hunicke studied this phenomenon in her research and affirmed:

"Conventional wisdom suggests that while players enjoy unpredictability or novelty during gameplay experiences, they will feel 'cheated' if games are adjusted during or across play sessions. In order for adjustment to be effective, it must be performed without disrupting or degrading the core player experience." [B-14].

About the testing and the results she witnessed, she decided not to share the evaluation goals with the testing subjects, but she left the 'Dynamic Game Adjustment' in the title. Her intentions were not only to see if her DDA system would improve the gaming experience, but also how people would react during and after the game session; after dividing the subjects in half these are her observation about **adjustment perception**:

"Overall, there was only a small correlation (0.3) between the perception and actuality of game adjustment. Subjects often perceived adjustments that were not there, and several discounted adjustment when in fact they had been receiving help throughout the game. [...] Of all the subjects tested, only two remarked that adjusting a game's difficulty would lessen their enjoyment. Of these two trials, one was adjusted and the other was not—but neither reported it."

Confirming the theory, that would alter the DDA design in years to come, that many players despise the adaptive difficulty concept without even understanding it, blaming it even when it is not implemented because of the mere idea of it.

Proceeding from these steps, Hagelbäck and Johansson decided to focus his interests on the satisfaction and the overall experience given by an even match and a winning match. His experiment, managed to test different players against 5 bots:

- **Bot A** and **Bot B**: were static and respectively medium and low difficulty. They should have provided an average experience and an easy experience.
- **Bot C**: was adaptive and would slowly turn to the player's level. This was to provide an even match where would have been hard to notice the adaptation.
- **Bot D**: was adaptive as **C** but after certain events (that could cause anxiety in the player) would quickly drop to easier to almost always let win. The idea behind **Bot D** was to provide a sense of triumph over a difficulty, the player should feel the exhilaration of striking back when falling but was intentionally implemented as a quick drop of difficulty.
- **Bot E**: was adaptive that could quickly adapt, especially in early game phase.

Results obtained showed, as expected, that adaptive bots (C and E) were the most enjoyable, while Bots A and B were considered unbalanced. Test subjects obviously did not know what they were playing against, nor that some of the bots had adaptive difficulty. However, an interesting insight was about bot D: it was enjoyable in his first match, but would immediately be perceived as unvarying and not stimulating; that overturn of the events in end game was pleasant when unexpected, but as soon as it became routine players would find it unchallenging. Quoting the author:

"The adaptive Bot D that drops the performance in the end of a game was perceived as boring, easy to win against and with low variation. It is interesting to see that this bot was considered least varied even if it uses the same adaptive technique as Bot C until the performance drops and tanks in practice are almost idle. This might be because participants have a tendency to mostly remember the last part of the

game where the AI is really dumb, and therefore consider the bot as too easy and boring."[B-8]

These, and many more experiments have been made in the last decade and a half, and they all showed similar results; DDA improves the gaming experience, allowing unexperienced players to avoid the **frustration zone** especially critical in an early phase and make the whole experience way more enjoyable, and providing hardcore players an always demanding challenge that requires some focus even when the game is repeated and known by heart.

But players are proud. Novices will feel fed and not worthy of finishing the game without the game allowing them to. Hardcore players will mainly feel cheated after every win they achieve, because it will be in their mind that it was not because they overcame the required skillset threshold but because the threshold moved. These are the players that mostly hate the mere idea of DDA.

But should they not know about adaptation happening, they will be both very pleased with themselves: Novices will be proud of having grown and hardcores will feel that they learned after every mistake. The solutions seemed clear but not right: Dynamic Difficulty Adjustment should be concealed.

And this is the principal road taken lately, it is in fact hard to actually know whether a game implements DDA or not because developers never state it; the sole information about improvements and new techniques of adaptation suggests that it is not dead (as commonly believed). Many titles are recently showing tracks of analysing the player expertise and are mainly going down the road of explicit telling by either fusing it with the gameplay or use it for small, playful and light modifications. Summing up, DDA in recent years usually is:

- **Truthfully absent**, either because the game experience does not require or benefit from it.
- **Completely concealed**, to give all the advantages of Dynamic Difficulty Adjustment without making them (the players) feel patronized.

- **Carefully merged with game mechanics**, as the same designer of *Resident Evil 4* later did with ***God Hand***[L-16] and, even more so in the exhaustive example in the following paragraph.
- **Explicitly but playfully and lightly used**, as in ***Metal Gear Solid 5***[L-28] where if the player dies too often, will be given the opportunity to wear a *goofy chicken hat* that reduces the challenge but makes him look like a complete dork.

This last option is the closest to the fifth and untaken road of explicitly asking the player to use or not DDA; option that is unmentioned in literature but could be the path in this decade, giving the player explicit control and consensus on the application of DDA and, perhaps, proving that it actually helps the experience.

II.2.4 A game with all the approaches

Back in 2012, the science of game design was going through an important period of transition: adaptive difficulty rose and fell, and few were now trying to reintroduce it merging it into the game mechanics; modern mobile games were spreading on play store and app store, and the completion grading method with them.

This brought console games to an ugly choice in difficulty setting, by either following the path of the mobile gaming, or taking a step back and returning to the multiple-choice level of difficulty; in this confused period *Project Sora* proposed a title that blended the 3 main traditional approaches to difficulty setting. Third of the *Kid Icarus* saga, ***Kid Icarus: Uprising***[L-19] was a great success and also a great game design experiment, receiving only small criticism (that are actually useful in a game design analysis).

The designers and developers decided to walk through the path of the '*explicit adjustment'* by giving unmistakable knowledge of the adjustment being made and by also starring it into a trailer.

Basically, the game presents itself as a *multiple-choice* difficulty setting title; allowing the player to spend *hearts* (in-game currency) to *'bet'* on the difficulty of the upcoming level. It is a ‘bet’ because the higher the difficulty of the level, the better the rewards will be upon completing it.

However, selecting a high difficulty and try winning it is not enough; upon every game-over the difficulty adjusts itself monotonously decreasing; this is the *dynamic difficulty* in this game. Finally, the game remembers and shows the player the highest difficulty grade on which he succeeded in each level, featuring this way the *completion grading*.

The selected grade of difficulty highly affects different aspects of the gameplay: from the basic number of enemies to the number of bullets shot by each of them, and the velocity of both enemies and attacks, from the damage dealt by enemies to their level or health points. Everything beautifully calibrated to provide an experience from pathetically easy (called ‘*Effortless’* in the game itself) to almost humanly unbearable (‘*Nothing harder’*). Whenever the difficulty drops due to a game over the *hearts* spent are not refunded, and this puts the player in a risky situation while betting because he will have a limited number of occasions.

One of the few things criticized of this game by the designers was the too numerical difficulty grading; this in fact can go from a minimum of 1.0 to a maximum of 9.0 (with each intermediate step of 0.1). This precision of the number that goes to the first decimal digit was accused of breaking the immersion of the game and being too dispersive, offering actually 81 different levels of difficulty for each level.

It was suspected to inherit the same problem encountered by Jenova Chen in his first game experiment, but the public responded well to the design and was not bothered by the decimals, perhaps thanks to the light drawings and funny animations.

Kid Icarus: Uprising was quite a success both from the player viewpoint and from a game designer one, offering both an interesting fusion of known difficulty setting patterns and useful analysis for future designers. The *Fiend’s Cauldron* model (that was the bet system in the game) was in fact later recycled for another *Nintendo* title in 2014[L-37].

II.2.5 A summary taxonomy for difficulty setting

These traditional approaches do not constitute a partition of all the possible settings; there are games that place themselves in intersections and provide various approaches. For starters, the *performance grading* is a strategy appliable with any other as an extra feature, while the difficulty could either be static or dynamic, but never both (being conceptually complementary).

To try to provide an easier navigation among these concepts, it could be helpful a summary taxonomy that could graphically describe the relations between the known strategies.

Keeping in mind that there is no uniquely correct way to proceed, a smart first bipartition (in accordance with the topic of this thesis) could be between ***static*** and ***dynamic difficulty***. Formally, we will define (in this work) a setting as belonging in one or the other as follows:

- ***Dynamic***: if at any point of the game, the game changes its difficulty according to its metrics, or purposes to do so.
- ***Static***: if the difficulty, when changeable, can only be altered by the player directly(in the allowed moments).

Given the definitions, this first differentiation provides a strong separation between these two, intending them complementary; for clarity, any hypothetical mixed approach of player's consensus, average setting, and resettable difficulty would still to be considered as belonging in the *dynamic* subset (details later). From here on, these two sets (and the elements contained) will be referred as ***Statics*** and ***Dynamics*** for conciseness.

The second cutting plane that provides specializations without intersection, would be between the *intrinsic* and the *multiple-choice difficulty*. For *Statics*, this division is the same explained in the previous paragraph; so, a game belongs in each if:

- ***Intrinsic difficulty***: The game's difficulty curve is drawn by the developers and unmodifiable.

- ***Multiple-choice difficulty***: The player can, in some moments, set the game’s difficulty.

While for *Dynamics* this difference is to be interpreted as the starting difficulty. Specifically, what would be *intrinsic difficulty* in the *Statics* here would be ***prefixed start***; games belonging to this class would have all the players start with the same challenge that would later change for each of them; *Wii Sports Resort’s Bowling: Spin Control* would belong here. Should be pointed out that games that allow change in the difficulty setting (like moving from *Difficult* to *Normal*) are not to be considered as *Dynamics* but as *Statics* with multiple-choice; the reason behind this is that the difficulty is not dynamically changed but statically from a menu, like a challenge reset.

The same logic for *Multiple-choice difficulty* is more complicated; a generic formal definition of this subset could be: dynamic difficulty settings that consider the player’s decision for the adjustment. This subset, called *Player driven* from now on, includes (but could not be limited to):

- ***Start indicative***: where the player selects the desired starting difficulty, in some moments as it was in *Statics*, then the difficulty moves from there
- ***Range indicative***: where the player selects the desired indicative difficulty; the game would eventually change it but without moving too far from it.
- ***Explicit player input***: when the player explicitly declares the changes he desires.

To ease the growth of the taxonomy we will assume that this last differentiation is summed up to the possibility to stray from the indicated difficulty.

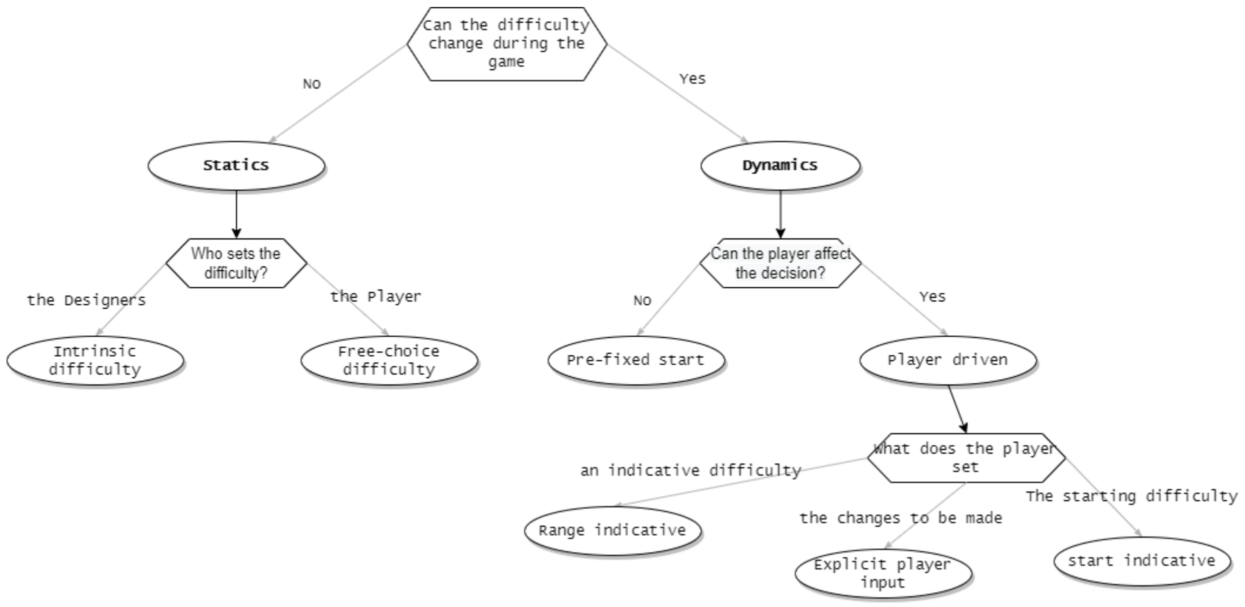


Figure 10 A summary taxonomy for challenge setting in games

So far, the differentiations made could be graphically represented as a tree of possibilities (see figure above); from these leaves further, the properties become partial and overlappable; and so better describable with Venn diagrams. A small, indicative set of the possible additional features/subsets contains:

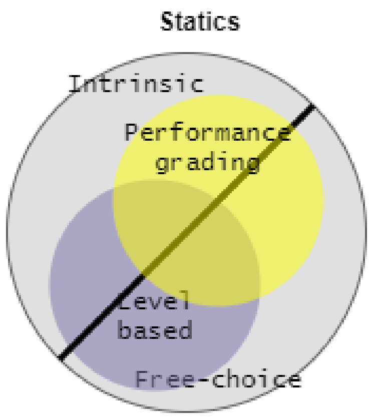


Figure 11 Internal division of the Statics

- **Performance rated**: as explained in the traditional approaches, any difficulty setting could hypothetically implement such system to grant the aforementioned benefits of it.
- **Level based**: in which characters and NPCs have a numerical value (*level*) that indicates their strength and statistics.

    - **Level scaling**: a *dynamic* specialization of *Level based* that provides the level of other units in the world to change according to some metrics relative to the player's level.
- **Adaptive**: a *dynamic* solution that, according to various metrics, changes the difficulty of the challenge without being (entirely) predictable by the player. This is an actual subclass of the whole *Dynamics* concept, being it a way of being dynamic.

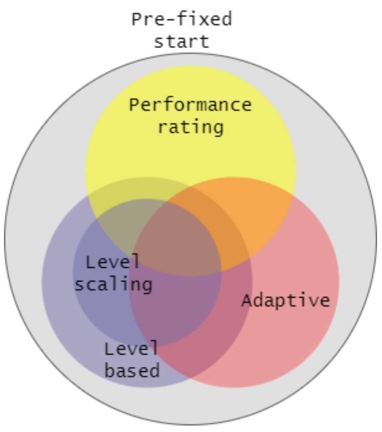


Figure 12 Internal division of the Pre-fixed start

While venturing in the *Player driven* set, many possibilities could be implemented and mixed up; acknowledging that many more could be added, two most interesting would be:

- **Consensus required**: when, before the difficulty is changed, the player is not only informed but has to agree to that change. This could either be in the settings menu as an enablable feature or right before it is changed (i.e. after a death screen)
- **Resettable difficulty**: when, the player can eventually reset the difficulty to a specified value/setting regardless of the modifications made.

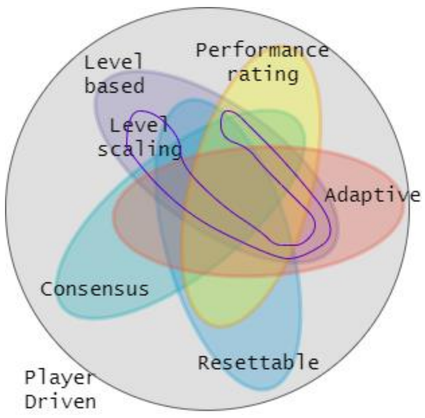


Figure 13 Internal division of the Player driven

These last two are actually implicitly always verified in the *Explicit player input* category, given the nature of it.

An unmentioned traditional approach is the *Score based*; many games implement a scoring system and this feature is either central, as in arcade games, or totally ignored, as in *Super Mario*. *Score based* games are what, in years, evolved in *Performance Rating*. Given the effect over experience, the latter can be seen as an expanding (generalizing) evolution that encapsuled the first.

Games often offer different approaches at the same time, and the customizable settings could allow a single title to move from an intersection to another. The many games referenced so far could be placed in this taxonomy as follows:

Monster Hunter World (MHW): belongs to *Dynamics* only in multiplayer, for this reason single player and multiplayer should be classified separately. In both cases, the player has no power over the change of difficulty and, finally, the game does not give a score of completion of the missions, but there is a concept of character level as well as different levels of strength of monsters.

Follows that, in single player, MHW takes place in the *implicit difficulty* subset, level based but not graded. In multiplayer, will go in *prefixed start*, level-based but not level scaled nor adaptive.

Wii Sports Resort (WSR): as specified earlier, the *Spin Control* mode of *Bowling* presents dynamic difficulty in the peculiar multiplayer competitive context. Here again, given that the player has no control over the changing, the subset will be *prefixed start*. However, there is no level concept here and instead, there is the score concept. Other sports in the game provide a little level scaling mechanism in the single player mode, providing a harder challenge when winning games.

Crash Bandicoot 2 (CB2): Differently, CB2 does not present a multiplayer feature and is way older than the previous titles; while the difficulty is still *prefixed start* here we are firstly introduced with an adaptive mechanism: boulder slowing down, checkpoints spawning and a single or double *Aku-Aku* on start. Furthermore, the *relic hunt* mode provided after a level is completed once, disables the adaptive feature while adding a completion grading based on the time required to complete the level. The existence of three levels of completion (*Sapphire, Gold* and *Platinum*) provides an effect similar to the 3-star grading.

Jenova Chen’s experiments: **Traffic light** and **FlOw**, both have similar traits: The difficulty is dynamically changed thanks to either explicit question to the user (*traffic light*) or explicit possibility in game to move from an easier zone to an harder and vice versa (*flOw*). Moreover, neither of them is adaptive (in a strict sense) nor portray a level feature or a completion grading.

Mario Kart: is another multiplayer dynamic system; by both slowing and accelerating NPCs and giving different power-ups according to the position it provides another example of *adaptive prefixed start* setting. There is no level concept (apart from the ranking in an online session that is non inherent) but the very nature of the race competition provides elements of performance rating.

Final Fantasy VIII: as specified when introduced, is a clear example of *Level scaling*; more specifically, it belongs to the *prefixed start difficulty* category and does not present any other sub feature.

Resident Evil 4: offers instead a selection of 3 (4 in Japan) different challenge settings, but then adjusts to a more suitable difficulty while playing. This puts this title under the *Player driven* category. It is unclear if there is the possibility to stray to a complete different difficulty setting, because he DDA was never officially stated; however, there is no consensus nor resettable feature.

Finally, **Kid Icarus Uprising**, provides a multiple-choice over the initial difficulty of a level that would then change after each game over. This modification is both resettable (restarting the level) and requires consent (although it forces a restart if the player doesn't want to reduce the challenge); and as specified the completion grading is merged with the gameplay.

II.2.6 The players' perspective

Of course, apart from all the design's concepts, the players are those who will actually deal with the results of all these efforts; therefore, how they feel about adaptive difficulty is an important matter. As anticipated, many players from the old school generation to the newbies, show plain dislike for the mere concept of dynamic difficulty, comparing it to a cheat and an insult.

Regardless of the understanding of design concept by them, it is still important to understand what causes this distress and how to handle it. For starters, probably the most common comment about adaptive difficulty among players, is that the game is interpreted as a **challenge**, therefore, should the challenge be reduced when the player struggles, any win would not and should not be satisfactory because the game forced it. Of course, many games do not do that, but this sentiment is understandable; if a game allows an easy win the player is not overcoming a true obstacle[B-7], thus losing the desire (and pleasure) of dealing with it.

The truth, from the other side, is that players are more willing to quit playing than to lower the difficulty level, considering it a (shameful) defeat declaration even more than quitting; but this is a complete loss for the game both economically and conceptually. The DDA in these scenarios is a perfect solution, because it gives the struggling players more time to grow and become stronger without needing to give up the experience; it helps the player by training them until they finally reach the average level. Of course, to do so and not justify the players' fear, a game should never become effortless.

Another rage element is the overruling suffered: when the game eases the challenge without asking it to the player or even without informing it explicitly, it plainly ignores the desire of the player over the experience he/she wants. Surely, a solution would be a notification or confirmation by the player but this approach is rarely used because it is feared that it would ruin the experience: let us imagine a game that after a certain number of loss notifies or asks the player if he/she wants to automatically adapt the challenge level, the player would feel immediately judged and even criticized by the game and would most likely stubbornly refuse to prove the game and the developers wrong. Should, in the end, the player lose, he/she will feel humiliated by the game and feel negatively about it; should the player win instead, without needing to change the difficulty, it would be a confirmation that the adaptive difficulty was unneeded and unwanted and will be not used in future as well.

Probably this is what led the game companies to hide the dynamic difficulty and never state if it is implemented or not, therefore never giving certainty to the players' doubts; and for the same reasons the games that do state the presence of adaptive difficulty, merge it with the gameplay (as successfully done in *Kid Icarus*[L-19]).

The proposed taxonomy, especially the consensus and resettable features, were hypothesized starting from these concepts, and propose themselves as probable approaches for the next years to come; as maybe a more familiarity with the adaptive difficulty features could lower that fear and hatred still existing towards it.

It is worth mentioning that most of the source of these comments and ideas are from players “helped” by the DDA rather than from players challenged by it; thus giving ground to the assumption that most of these ideals are born from pride of unexperienced or novice players that find themselves praised for their victory in new games, ignoring that many new games have lowered their standard difficulty to accommodate the new generation audience; while the adaptive difficulty could justify a harder average game probably still unreachable by most of the players.

Chapter III
The game development perspective

Game development is the other face of the coin; while designers study how the players should feel while playing, which state should the player be in in a particular moment and everything else specified in the former chapters; the game developers have the task to find and implement solutions to these ideas.

It is the developers' job to create the functions and routines that make all the designers' conjectures and schematics even possible. In the latest years, developers have been appointed to also create the bridges between the code and the designers.

The basic idea is that while the developers can code and edit the game statistics, the designers should be the ones that can interpret the effects of a major or minor statistic value, as well as the maps designs and the desired NPC behaviour. But designers do not necessarily 'speak code'.

That is why recently, while developers started to understand more about design, designers were given tools to edit game features without touching the code; and so, thanks to data driven characters, maps, statistics and NPCs' behaviours, balancing and fixing game features became not only easier but even more precise.

On the other hand, game developers were given the ungrateful task to implement always more generic, more accessible and more efficient codes to ensure top notch quality games. While game design is not a technical field of computer science, game development delves deep in all the aspects of it; from software engineering standards and models to local and online persistency issues, from code efficiency and performance to dealing with computationally complex problems.

In a broader sense, game development is everything that concerns the actual implementation of the game, starting from the models and ideas of game design. For what concerns balance in games, developers came out with a wide range of ideas and solutions that provided formulas easy and efficient to implement.

Percentage increase of statistics according to level is a clear example of that: it is space inexpensive (opposed to *stats look-up tables*) and provides a regular increase with interesting insights; mainly, stats that start high grow faster. This apparently naïve peculiarity is key in enunciating characteristic differences among units in game.

$$stat_{\boldsymbol{L}} = stat_{\boldsymbol{1}}\,(1.0 + percentage)^{\boldsymbol{L-1}}$$

For example, a *tank*[12] unit will start with a higher value of health than the average; this means that at higher levels its health will be grown way more than others', thus underlining its peculiarity. This mechanism is known among gamers as *equilibrating* and *breaking* a character, and both go under the balancing procedure; where, by "equilibrating" it's meant to have all stats near average (many games provide at least one equilibrated character), while with "breaking" it's indicated the voluntary difference between stats in order to expose strengths and weaknesses.

In a very development optic, balancing is not even static: it is not (only) done during development and test phase and then considered complete, but always more games are now adopting the *patching* paradigm, which consists in periodical updates to game mechanics, stats and even contents. This feature is most typical in online games and multiplayer given the massive variety of players and executions (runs).

A brilliant Game Development approach to patching mechanism is portrayed by *League of Legends*[L-22] with the implementation of the **Public Beta Environment (PBE)**: this allowed developers to test their patches, before the release, on an significative sample of players. Every player could join the PBE in which there would be no effect over win and loss, any bug or excessive strength or weakness of a feature could be spotted, the players' reaction to any update could be predicted and eventually corrected and, in meta-game, the community could get familiar with any new major update before its official release.

All these are development solutions, environment and strategies very useful in development; but they are still approaches to balance the game overall, not for the specific player. The adaptive strategies in DDA usually requires, in their more

[12] A unit characterized by a high amount of health and defences.

elaborate shades, competences in algorithmics, machine learning and computational complexity.

III.1 Modern DDA approaches

In 2018, a research[B-15] conducted by Mohammad Zohaib showed that the researches about DDA gained interest and popularity in the last decade, and also provided an interesting overview of DDA methods approached and their peculiarity.

Probabilistic Methods were a popular and widely approached solution. Starting from the idea of an optimization problem, dynamic programming algorithms were implemented; later other methods such as *genetic algorithms*, *Hidden Markov Models*[B-16], *Bayesian optimization*[B-17] and *orthogonal coevolution* were proposed and tested. The results obtained were interesting, but the methodologies were complex and not designer-friendly; this means that any modification in calibrating these DDA systems required specialized figures.

Nevertheless, these approaches opened the path for more advanced ideas; for example, the Dynamic Scripting is an evolution of the *fuzzy rulebase*[B-18] proposed in 2008 in China, which proposed a set of rules that could be active or disabled, and a system that enabled or disabled these in order to maintain a certain win rate, in the quoted research case specified by the player itself.

Single and Multilayered Perceptrons: In a succession of studies carried on by Georgios Yannakakis and Julian Togelius; the idea of the entertainment value of a given game is an unknown function of player and game features which an ANN could learn[B-19]. Initially, a simple single-neuron (perceptron) is utilized for learning the relation between features and the value of the investigated emotional preference of a game (for interpretability of results) [B-20].

Continuing from this study they built a model to generate levels for platform games customized to suit real-time player experience automatically in real-time, predicting emotions, to some extent, from controllable features[B-21].

Dynamic Scripting: It operates many (manually designed) rulebases in the game, running one for every opponent type. On creation of a new opponent, script rules are selected from the base depending on the weight value allotted to them. In this approach, learning takes place progressively. On completing an encounter, the rule weights used in the encounter are treated depending on their effect on the result. The rules leading to success have their weights increased, while those leading to failure have their weights decreased; remaining rules are adjusted accordingly so that the sum of all the rulebase weights remains constant.

Authors' experimentation[B-22] showed that this method is very effective (in battle scenarios) and the three variations proposed could not only improve variability, but also performed even better in a real application[L-31]. Moreover, the dynamic scripting approach meets the standard requirements for AI in games, being fast, cost efficient, not deterministic and extremely customizable.

Hamlet: As already introduced, the Hamlet system operates on the player's inventory. It uses metrics for monitoring incoming game information as the players advance through the game world and tries to predict the future state of the player from this information. Whenever an unwanted, but preventable, state is predicted, Hamlet steps in and tweaks game settings as required. Essentially, it tries to anticipate when the player is struggling repeatedly and nearing a state where his existing resources can no longer meet the requirements. When such struggle is detected, the system intervenes to assist the player to continue the game. Although generalizable to any game genre with a heavy inventory use, this system was designed for FPS games[B-9].

Self organizing system (SOS): a SOS is a set of entities that obtains global system behaviour optimization via local interactions without a centralized control[B-23]. These, combined with ANNs can provide a dynamic adjustment of NPCs as proposed in a research[B-24] that implemented an offline learning of a neural network through a combination of Cooperative coevolution algorithm and genetic algorithm; this phase meant to teach the ANN to chase the player. The adaptive feature came with the online learning that combined Interactive Evolutionary Computation and Simulated Annealing to select the worst behaving AI and replacing it with a 'more fit'. Evaluation of fitting for any NPC's AI (NN), is weighted on the proximity of it to the player (the game on which it was implemented was *Pac-man* so the efficacy of a NPCs was mainly determined by the capability to catch it). SOSs provided a very adaptive approach to a multiple NPC context, portraying a global adaptation through modification of the single parts without centralized directives.

Upper Confidence bound for Trees (UCT) to train ANN: While UCT is a popular solution for finite-space round-based games (notorious for its application in Chess and Go), a study[B-25] conducted in 2010 decided to use UCT created data to train an ANN using again *Pac-man* as test-bed. This title was the perfect subject thanks to its finite-space nature; however, regardless of the good results obtained, UCT strongly depends on the computation time allowed to it, so it is applicable to games only when computation time can be somehow tweaked or afforded, otherwise the ANN was proved to behave poorly.

Reinforcement Learning: Reinforcement learning is a computational approach to automate goal-directed learning. Essentially, it is the process of mapping the states of an agent to a set of actions. In each step, an RL agent determines and executes the best action on the environment that maximizes long-term rewards. In the difficulty adjustment problem, the actions that an RL agent can do to adjust the difficulty is a change in game properties associated with difficulty parameters (e.g., changing the speed of moving charter in the gaming environment) [B-26].

Reinforcement Learning, alongside with the Dynamic Scripting idea, were inspiring for the design of the proposed adaptive system of this dissertation; and the learning process was in fact achieved through a variant of this type of learning.

Affective Modelling Using EEG: a new research branch studied how to use information from physiological signals, i.e. trough Electroencephalography or fNIRS brain sensing. Thanks to this information it was possible to deduce better when a player was feeling bored or frustrated and adapt the game consequentially.

All these methods proved to have quite good results in their own experiments, but what emerges from analysing the lot is that DDA systems often (or practically always) require <u>domain specific knowledge</u>. Moreover, the game genre also determines whether an approach is effective or even appliable. Based on the feature that is desired to adjust and to the typology of the game AI and the game genre itself, an approach could be more indicated or less.

III.2 DDA sensitive features

As these approaches show, challenge-altering mechanics almost always require *game-specific knowledge*; therefore, it is usually developed after the core idea of the game itself. These strategies must rely on gathering specific information in order to analyse the player's expertise or struggles; as well as tuning specific values to apport the desired adaptation.

The work of selecting which aspects to track and which features to be modifiable requires a deep knowledge of the game and, especially, of its *dynamics*: for clarification, in Game Design, a *game mechanic* is a feature implemented in the game; while a *game dynamic* is a pattern, or logic emerging from mechanics. In example, the ability to jump backward after an attack is a *mechanic*, the usual chasing of a fleeing enemy (*kiting*) is a *dynamic*.

It is possible to mainly categorize those features that affect or are affected by DDA under three definitions:

- **Observed Parameters**
- **Objective Functions**
- **Controllable Features**

For clarification's sake, these names are not commonly considered as absolute; thus, it is possible to find them also as *observed features* or *controllable parameters*.

These 3 DDA elements reflect the idea of ideal challenge intrinsic in the flow zone chart. In its most basic version, the flow zone chart presents only a *player's skill axis* (the *x-axis*) and a *game challenge axis* (the *y-axis*); the flow zone is hereby described as the locus of the points for which the challenge is ideal for the player expertise being both stimulating and overcomable. Despite all the conclusions already discussed in game design, in game development these two axis are actually:

- ➢ an observable and incontrollable parameter, the player’s expertise, which is unknown to the designer/developer and to the system alike; but which can be estimated from the in-game behaviour and with heuristics.
- ➢ An indirectly settable parameter, the game challenge, on which the designer/developer “has control”; but which the ideal value is to find in order to meet the flow zone.

The flow zone, in this scenario, is another unknown value due to the different nature of each player; and is therefore another parameter that requires to be estimated (it would be ideal to map every player into a category for which the perfect experience is known, even just among *challenging experience*, *relaxing experience* and *neutral experience*).

[INSERT IMAGE]

Moreover, as specified, the challenge parameter is only indirectly changeable; due to the presence of various elements that compound it as such, the challenge can only be adjusted by altering the environment surrounding the player experience (controllable features). Many, more elaborate, adaptive systems also try to identify which controllable feature is more responsible for the distance between the optimal and current setting and/or to (try to) predict which change in the feature space would cause the most valuable adaptation.

This is the idea of DDA interpreted in game development: identify the x-axis position of the player expertise <u>and</u> the function that associates to each expertise range the idoneous challenge interval in order to have an indicative target of the optimal challenge desired. Of course, the flow zone chart is a bidimensional simplified representation of an idea; down in a real application the x-axis would be a multidimensional skillset array that identifies the expertise of the player in all the possible aspect of the given game, e.g.: reflexes, coordination, pattern knowledge, strategy, resource economy, response under pressure, …; in the same way, the challenge level of the game in a given moment would be multidimensional too: velocity of threats, damage dealing indicator, survivability, strategy, team coordination and harmony, events and trigger thresholds, and so on.. .

Ideally, the DDA should only be the first step to player personalized game contents; with the following being ***adaptive training***. Adaptive training means developing DDA systems in order not only to match the perfect player game experience, but also to adjust optimally to help the player grow, to make this last learn at the exact pace needed. Of course, should this be achieved in Game Design and Development, it could be extended to other branches like medical rehabilitation, gamification, instruction, etc… .

Formally, in Artificial Intelligence, this elements in analysis are referred as *task environment*; and are acronymically summarized as **PEAS** (**P**erformance, **E**nvironment, **A**ctuators, **S**ensors). An example to understand the meaning of each concept is usually the *taxi driver problem*: a fully automated taxi is currently somewhat beyond the capabilities of existing technology[B-27], it serves however an excellent example to understand the following schema.

Agent type	**P**erformance measure	**E**nvironment	**A**ctuators	**S**ensors
Taxi driver	Safe, fast, legal, comfortable trip, maximize profits	Roads, other traffic, pedestrians, customers	Steering, accelerator, brake, signal, horn, display	Cameras, sonar, speedometer, GPS, odometer, accelerometer, engine sensors, keyboard

In our specific context: **P**erformance would be the evaluation of the *objective function*, **E**nvironment would be the *observed parameters* and **A**ctuators would be the *controllable variables*. The **S**ensors depends on the game itself and could be speedometer, barometer and so on; but could even not be present in completely digital game (the game itself is the sensor).

III.2.1 Observed parameters

These are the input for a DDA engine; and constitute the main actors of the player status interpretation. Understanding correctly which behaviour, variation and numbers actually represent whether a user is struggling or enjoying the experience is key in the whole DDA engine development.

The selection of the observed parameters requires a detailed study, that should also combine with the work on the *objective functions*; Such selection specifies what is the focus (of the designers and developers alike) when observing, trough the game, their players. While there is no correct and wrong approach, the ideal selection should make it possible to hypothesize over the current state of frustration and enjoyment of the player, but also to identify the aspects that make the user struggle the most.

Taking inspiration from the researches of Yannakakis and Togelius, a major difference among observable parameters is between ***Environment parameters***, ***Player (***or ***Gameplay) Parameters*** and ***Game-through***[13] ***parameters***.

All three are important and dense of information, and their combination is therefore crucial to assemble a complete picture. The first, are used to give context to the information gathered and are hence essential to avoid dealing in absolutes; *Environment parameters* are all those that do not depend on the user interaction, but only by configuration of a map, the placement of units, the size of obstacles and so on.

The second are, on the other hand, specific of the user playing and its performance: variation of statistics over time, gathering of collectibles (objects), direction switches, precision, dexterity and so on. These are fundamental to understand (or to try to) whether is enjoying the challenge or struggling, if is he having fun, what triggers the greatest challenge for him and everything that is related to his playing.

Player parameters are actually a specialization of the ***Agent parameters***, when they refer only to the player as agent; but many systems evaluate the behaviour of the NPCs as different agents and their gameplay statistics are collected too. We will refer to these as ***AI agent parameters***.

The last, the *game-through parameters,* are somewhere in between: these are those observable parameters that depends on both the players' interaction and NPCs' decisions but do not characterize <u>their</u> behaviour as such, they offer an image of the

[13] We shall define ***game-through*** as the evolving succession and the course of the game and game events.

common game situation instead. Examples of these could be: distances between units or teams, entropy of map coverage, etc… .

Player parameters are what designers and developers would observe of the players' behaviour if they could constantly look at their users. The same recording of player parameters could, and should, be interpreted completely differently in different scenarios; as well as environment ones alone do not give enough intel about the player specific useful adaptation. Different games, genres, DDA strategies and, most importantly, *objective functions* could require different kind of parameters, and use them very differently as well. These could comprehend measurements of time, actions (button clicked or in-game actions), events, statistical values and even stress level.

III.2.2 Objective functions

Adaptation, in its most elaborate attempts, is treated as a research optimization problem; and as such requires the formulation of a well designed objective function: a value to maximize (or minimize) under the given constraints. In the DDA context, it is usually a score of how well the system is performing or how much does it fit to the player expertise.

Designing the objective function directly condition the 'nature', the behaviour and the focus of the DDA engine. It is the central key step in the whole dynamic adjustment: *parameters* are chosen in order to provide the information required by it, while the apported changings in *controllable features* are always in the attempt of improve the objective function in its next evaluation and/or dictated by the current value of it.

Such functions can either be very general (in concept), as could be understanding and predicting the emotional state of the player; or very domain specific, as the fitness of a strategy in attempt to achieve a goal. Either way, domain knowledge is necessary to correctly evaluate (and design) which and how aspects can affect the game experience.

To be meticulous, these functions are not even always treated as *objective functions*, but even as scoring or indicative evaluation, necessary to understand the effect of the actual setup of the game and where would be effective to alter such setting.

Given that examples of objective functions are various, different and highly game related, they will be presented together with an overview of the previously named strategies in a few pages.

III.2.3 Controllable features

Controlled features are the last step of the adaptation process; these are all the game parameters that are editable at runtime. It is important to notice that making a feature parametric and editable at runtime requires an extra programming (and development) effort.

Controlled features can be as various as the permitted variations: it could be the speed of a certain enemy or obstacle, the range of sight of an observer, the statistics of certain units or the number of them in a moment; or it could be something non-numerical such as the set of spells available, the algorithm selected for a behaviour, the priority ranking of a situational handler and so on.

Just like the previous aspects, the selection of controllable features requires a certain game specific insight, as well as a detailed analysis of the game dynamics and heavy experimentation. Such an attention and an effort is justified by the importance of these values as such: any modification of which would affectively alter the challenge (and sometimes the dynamics) of the game itself.

However, adaptation is not just about changing these features' values until they meet the perfect combination: it is fundamental to understand the value of the objective function in relation to the observed parameters in order to decide what and how to alter a feature; but is also just as important to predict how the hypothesized changing would later affect the objective function evaluation.

Concluding, the feature adjustment, along with the objective function evaluation and the parameters observation, can take place in different moments according to the DDA design and the development team idea:

- It can be *continuously* or *periodical on time*; meaning that after each prefixed Δ_t the features' values are corrected according to the observation made (that could take place continuously at a different pace). In this consideration, *continuously* is interpreted as runtime, that is having a Δ_t equal to one frame.
- It can be *events triggered*; when an action or a situation can trigger the system start. This can be the gathering of an item, the changing of zone or the trespass of a threshold; with due schematization, different subtypes could be *on phases* (levels, zones, thresholds), *on encounters*, *on respawn* and so on.

And many others according to the design needs.

On an ending note, it is common use to include the controllable features in the observable parameters as well.

III.2.4 The DDA analysis

For the purposes of this thesis, before selecting a testbed game and a strategy of interest, the literature of recent state of the art DDA strategies was consulted and from it, all the conclusions mentioned above. An analysis of these aspects in the strategies already existing in literature followed in order to establish an iter in a methodological approach to DDA.

The purpose of this effort was to practice and test a DDA analysis posteriori of its implementation and design, helping to set the methodology base and an expectation of goals for the subsequent analysis of feasible testbed games. The conclusions of it are summarized as follows.

Modelling player experience (Yannakakis, Togleius, …):

- The *observable parameters*' set comprehended: <u>*gameplay features*</u> related to jumps, time, items, deaths, kills and misc.; and all the controllable ones (*environmental*).

- The *objective function* was in this case an indicative function (more than an optimization problem) and it consisted in the output of the ANN interpreting the current emotional state of the player.
- The *controllable features* were in this case all environmental, as expected given the level auto-generation nature of the research in the first place. These included number, width and spatial diversity of gaps displaced, and number of required direction switches. The adaptation takes place to the level generation (*event triggered*) being this the scope of the research itself.

These last were kept as essential as possible in order to extend the results at the platform genre instead of the *Super Mario* title alone. Moreover, the usage of a well-known game is highly recommended in this researches because it allows (and justifies) the assumption that most players (testers) already know the basics of the game itself and therefore their behaviour is not influenced by the initial attempt to understand the game.

Adaptive Game AI with Dynamic Scripting (Spronck, Ponsen, Sprinkhuizen-Kuyper, Postma):

- The *observable parameters* were directly correlated with the outcome of the encounter: win or loss (of the AI, not of the player), remaining health of the agent and of the allies on the same team, damage dealt and survival capability. These could be classified as *game-through* and *AI agent parameters*.
- The objective of the system was, quoting the authors themselves, *to generate scripts in a way that the number of wins of the dynamic team is about equal to the number of losses at all times*[B-19]. The *objective (indicative) function* was, in this case, a ***fitness function*** for each agent; this did actually take into account the behaviour as team <u>and</u> the behaviour as single unit.
- The *controllable features* were, given the structure of the system, the weights of the rules that could be extracted; and, as the research affirms, this adaptation is made at each (enemy) unit respawn (*event triggered*).

Originally, this approach was built with the idea of generating winning strategies; the adaptivity came from the selection of three different ‘enhancements’ to the dynamic scripting technique that would allow to not always prefer the best performing scripts.

AI for Dynamic Difficulty Adjustment in Games (Hunicke and Chapman):

- ➢ The *observable parameters* were in this case correlated with the inventory state and immediate future: average damage, inventory status and shortfall probability; but also, overall statistics such as: location and progress of the player, deaths counter, encounter repetitions and number of system adjustment interventions. (prevalently all *player parameters*)
- ➢ The objective of the system development was, again quoting the authors themselves, *to determine when a player is flailing*[B.14]. However, it is not explicit in the research compendium which is the formal *objective function*; despite this, it seems that the adjustment policies are triggered by the prediction of recurrent inventory shortfalls.
- ➢ The *controllable features* divided in two typologies of adjustments: *reactive actions* and *proactive actions*. Reactive actions will affect elements in-play (i.e. units that already engaged the player or that have been spotted) *manipulating the accuracy or damage of attacks, strength of weapons, level of health, and so on*[B-14]. Proactive actions will affect the game backstage (unit still not spawned or unseen), *this includes changes to the <u>type</u>, spawning order, health, accuracy, damage and packing properties of such entities*[B-14]; but, also, environmental features such as available ammos and health items, and invisible power ups like player's ammo damage.

A last, most important example (for its diversity and game testbed) is **Dynamic Difficulty Adjustment in Games by Using an Interactive Self-Organizing Architecture** by Adeleh Ebrahimi and Mohammad-R. Akbarzadeh-T:

- ➢ The *observable parameters* were in this case differentiated according to their usage. The offline learning required only the distance variation between Pac-man and the ghost in a given Δ_t (*game-through parameters*); while the online learning considered <u>also</u> Pac-man behavioural components such as number of wall hits, direction switches and key pressed (*player parameter*). The distance of a ghost from Pac-man and from its nearest ghost constitute also the input of its ANN (The authors stated that there could be more inputs, which would improve the efficacy, but did not revealed which ones).

- The *objective functions* are here considered ***fitness functions***, and are two different ones according to the phase of development: offline and online. For the offline learning the fitness function is simply the *chasing ability* of the ghost; in easier words: the variation between the initial distance from the prey to the final distance (after a pre-set Δ_t). The online learning is more elaborate and considers a mechanism called ***Fitness sharing***, a function evaluated for each ghost and dependent on two factors: The $\boldsymbol{\mathit{Fitness}_{\mathit{Total}}}$ value, which is common to all ghosts and dependent only on the Pac-man behavioural components; and a ghost specific parameter that gives major importance to ghosts closer to Pac-man.
- The *controllable features* are, given the structure of the system, the weights of the ANN. <u>These are not adjusted in this phase</u> but selected from a pool of chromosomes precedingly sorted by fitness (aggressivity) in offline phase; only after the next chromosome is selected, the mutation operator is applied to it and then the weights are overwritten (*periodically on time*).

III.3 AI in Game development

Despite all the considerations on adjustment strategies, system objectives and feasible algorithms a few words on what artificial intelligence in a game context are mandatory.

For starters, it is important to specify that every computer directed unit, namely an NPC, is governed by an AI system; it could be a simple *decision tree* with few actions or a most advanced solution, every solution to simulate the decision making process of a unit in game is considered artificial intelligence. The objective is the make-believe that that unit is in some way *thinking* of its own, in its own somehow realistic way.

While this could be expected from every (living) creature met in the game reality. The credibility of a game was entrusted in the early days almost exclusively to graphics, then to the physics engine, then to cover stories, the focus on artificial intelligence is relatively recent. And is on *credibility* that the attention should be focused; AI in games should not be developed to grow always better but to be always credible.

It would in fact be unrealistic if any gargantuan club-armed troll should prove himself a major tactician, as well as if any unit does not stutter when shot. The natural evolution of credibility is evolution itself; unit should learn and grow used to the player's strategies, but at the same time should not be too efficient in preventing these after they have seen these once.

Apart from this design concept (the realistic behaviour of NPCs), games have long been seen as the perfect test-bed for artificial intelligence (AI) methods, and are also becoming an increasingly important application area. Game AI is a broad field, covering everything from the challenge of making super-human AI for difficult games such as ***Go***[14] or *StarCraft*[L-35], to creative applications such as the automated generation of novel games[B-28]. Games have also been ground for experimenting on important computability problems.

Beyond this, another major difference between AI in game development and in computer science is the time cost tolerance. Usually (not always), the thoughtfulness on time complexity for AI in computer science regards mainly the asymptotic complexity; should an AI take a few seconds more or a few seconds less is tolerated in sight of performance seeking. That is not the case in game development however.

To be more accurate, there are various applications of AI that are not tolerant of small-time delays; game AI is among these. Its outcome is often required in short time, between a frame and another; that is why, given the unnecessity of quality over time, quicker but more approximate solutions are preferred in this context.

[14] **Go**, Strategic board game

Formally, the constraints required from adaptive AI were listed by Spronck et al. in the introduction to their work[B-22]. Summarily, they divided such constraints in two categories: computational and functional; respectively, the first are:

- Speed: since it takes place during game-play and must not disrupt the game experience.
- Effectiveness: as, not inferior to manually designed, otherwise it is a pointless complication.
- Robustness: must not override the game experience and must be coherent with the randomness of the specific game.
- Efficiency: must learn as much as possible from the smallest amount of samples and experiences.

While the functional are:

- Clarity: because both game developer and designers must have the possibility to read and interpret the learning results. It should not be forgotten that games are an artistic work, and no algorithm could give the human intuition.
- Variety: by giving different solutions and avoiding predictability. Many hand-designed game suffered from extreme predictability which tackled the entertainment.
- Consistency: Despite keeping the extremely important randomness of a game, its result should be averagely consistent; it is unacceptable, for an adaptive system, to behave in completely different ways towards two users playing similarly (once complicating and the other simplifying, in example).
- Scalability: must scale to the skill level of the human player. Although, this seems a bit obvious given the context, therefore it would be more interesting to consider the *Scalability* as the capacity to extend the possibility of the system without needing a complete rebuild of the system.

Chapter IV

A DDA system for *maze chaser* agents

Once the research for state of the art and most recent studies on DDA for games was completed, it was decided to approach the design and implementation of a new DDA system.

While studying the different approaches of the last decade, a flaw was spotted in most sophisticated systems: although the adaptation was successful, the AI often resulted <u>denaturalised</u>. The behaviour of the game towards the player (most especially of enemies and NPCs) is usually carefully tuned in its most little details; game designers spend a lot of time to ensure that the <u>*behaviour*</u> of a unit is reflected in its skills, abilities and statistics; and while the scientific approaches so far did beautifully, mathematically speaking, they lacked in preserving the uniqueness and the self of the game behaviour.

When designing the approach to follow in this research, the *self* of the AI was considered important to save and preserve in order to only enrich the game experience.

IV.1 The testbed game

In pursuing this purpose, it was considered most interesting to try an approach different from those found in literature. The usual modus operandi of such implementation is to start with an innovative idea for a DDA system, and then apply it to a game/genre. Instead, in this dissertation it was decided to study the ***D**ynamic **G**ame **B**alancing* as a concept, exploring various strategies and ideas in order to widen the inherent knowledge. The purpose of this choice was to allow to understand the game design and development concept of DGB, therefore becoming more able to analyse a game and make informed decision about the suitability of a DDA system, and its consequences.

With this in mind, various games were analysed; in the end, the notorious ***Pac-man***[L-33] was chosen. Among the many reasons in choosing it as candidate, it is interesting to name a few:

- ***Pac-man* is a classic** and the *flagship* of its genre (maze chase). Its notoriety goes beyond age and geographical difference, and this allows to a crucial assumption in these researches: the players are already familiar with the game.
- ***Pac-man* is essential** in the inputs of the player. Despite the controversial appreciation of countless mechanics, most games find their balance and success by offering few mechanics but many dynamics. In this Pac-man excels: the player can only decide the direction toward which Pac-man is moving.
- ***Pac-man* is addictive**. As many arcade games, the player does often play more than a simple game (match), which allows multiple observations.
- ***Pac-man* ghosts have different behaviours**. This gives the (desired) possibility to focus this work on an adaptive system that does not denaturalise the uniqueness of the NPCs.

IV.1.1 Original ghosts' behaviour

In *Pac-Man* the enemies that would chase the player in the maze are the ghosts; the designers of the classic arcade version managed to create NPCs that would be extremely smart with minimal characteristics.

Studying the strategies of the classic version is fundamental to both: understanding the decisions made to the original balancing, strengths and weaknesses of the units; and keeping in mind what is the nature of the NPC. Moreover, the original design required minimal computation given the architecture on which it was implemented; for this reason, the decision process of the ghosts is incredibly simple and presents little peculiarity unknown to most players. For the aforementioned reasons, a detailed inspection of the original Pac-man ghosts' behaviour is required.

The ghosts are not free to move; they can only try to go to their ***target tile***. This simple concept has however constraints:

- Ghosts do not compute a path. This is an extremely ignored features of these characters; they do not compute a complete path, they only choose their next direction based on a greedy heuristic: the Euclidean distance from their ***target tile***.
- Should two or more options have the same distance from the target tile, a tie-break rule is applied: the next direction is chosen following this priority order: Up, Left, Down, Right.
- Ghosts cannot effectuate 180° turn (except under specific conditions); this means that on entering a tile, they cannot choose as next tile the one that they just left; so it would be more precise to state that ghosts cannot willingly effectuate 180° turns. Moreover, this implies that on entering a *corridor*, a ghost won't actually have to decide a new route until it reaches the next intersection.
- Ghosts ignore the toroidal connection of the maze; they can still use them, but mostly they go there by accident and not as a shortcut strategy.
- There are certain tiles in which ghost cannot take un up-turn (except under specific conditions). These tiles are just outside the *ghost house* (where ghost start) and where Pac-man starts. Knowledge of this comports an immense advantage for players that can easily shake off his/her chasers by forcing them to take a longer route.

The ghosts are always in one of 4 states (technically 5), these are: **Scatter, Chase, Frightened, Eaten, (House)**. These states are mainly responsible for the selection of the ghosts' ***target tile***; this is intuitively the tile they are trying to get to, however it does not require to be an actual tile in the maze: the *target tile* is actually a pair of coordinates that ghost use to compute the distance.

The fifth state, **House**, is not actually in any documentation and only represents the initial condition of the ghosts that roam around their 3-tiles house until a certain amount of time (depending on both level and individual ghost) is passed. Blinky, the red one, is the only one that always starts already outside of the house; it follows Pinky, then Inky and finally Clyde. Only this state and *Eaten* state are ghost-specific, while all the others are shared between all the ghosts (technically *Frightened* too can be seen as individual, but more on that later).

In **Scatter** mode, ghosts are spreading. This is the state they usually are in when they leave the *ghost house*; it does only actually set a prefixed *target tile* for each ghost forcing them to roam around one of the four corners of the maze. This tile is, surprisingly, not in the maze for any of them, but they are specifically placed to obligate a specific loop for every ghost.

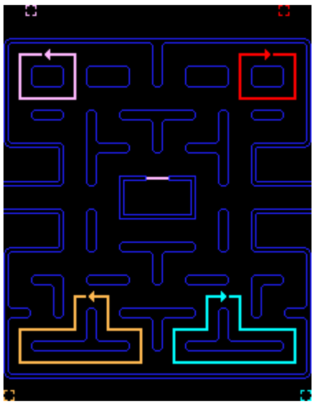


Figure 14 Pac-Man's ghosts scatter tiles and routes

In **Chase** mode, ghosts try to catch Pac-man. This is the state they spend most of the time in because the game alternates periodically few seconds of Scatter and around 20 seconds of chase in each level; however, after 3 times the *Chase* state becomes permanent. Differently from how many think that the ghosts behave, in this state they do not directly go after Pacman; they only select a target tile that is correlated somehow with him (Pacman). This behaviour is what makes each of them unique and different and so they will be explained in detail later on.

Frightened mode is user-activated when Pac-man collects a *super-pellet* (or *Energizer*). For the duration of the effect, ghosts will turn blue and will seem to run away from Pacman. In this state, ghosts are no threat to the player: not only they cannot eat Pacman, but they get actually eaten by him should he catch them. Curiously, they do not really escape from him: they will just start moving randomly (always according to the 180° constraint); this state is the only case in which ghosts can take those special upturns mentioned before. Frightened mode lasts always less on higher levels, until around level 16 they do not even enter this state anymore. Although it is a shared condition by both ghosts and Pacman itself (that obtains a speed bonus), this state can be seen as individual by ghosts because, should they get eaten and recover, they will go back to their previous state; therefore it is possible to have some ghosts in Frightened state while others in Eaten, Scatter or Chase.

Eaten, intuitively, is the state the ghosts are in when Pacman eats them in Frightened mode. Once eaten, ghosts will run quickly to their home and will return in *Scatter* or *Chase* mode. Curiously, a ghost in this state will not become frightened if another super pellet is eaten; nor instantly and not on respawn.

One last, most important, peculiarity before we dwell in each ghost specific behaviour is the transition between states. Whenever a transition occurs (both timer-dictated or user-activated) every ghost will automatically turn 180°; this is the only case in which they turn this way. The reason behind it is quite brilliant: in *Chase* state ghosts are considered to be surrounding Pacman or closing on him, while in all the others they are moving away; rotating their directions on state transitions causes a visual effect of escape (if they were in *Chase* mode for example), or of focus on Pac-man if they were scattering. The only exception to this is when they exit *Frightened* mode, in which they will not turn; there is no documentation about this exception, but probably it is to avoid that on high levels, when ghosts are too close, even eating an energizer could cause a death sentence.

The various states also alter both Pacman's and ghosts' speed; the details about it will be specified further when analysing the DDA features for Pac-man.

IV.1.1.1 The chaser ghost

The red ghost is called *Blinky*, and the game describes its personality as **shadow** in English, while in Japanese is referred to as 追いかけ (*oikake*), which translates as "pursuer" or "chaser". In both language the personality description is very accurate, since Blinky's *Chase* target tile is Pacman himself. This causes him to almost always follow Pacman from behind as his shadow. Blinky, as mentioned, is the only ghost that starts outside of the *Ghost House*.

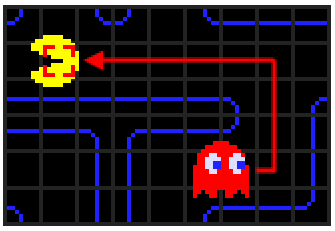


Figure 15 Blinky's chase target

In *Scatter* mode Blinky roams around the up-right corner of the map; however, Blinky has an exception to all the other ghosts: twice per level, based on the number of *pellets (dots)* left in the maze, he will obtain a power-up. When in this enhanced state, Blinky is generally referred to as "*Cruise Elroy*", though the origin of this term seems to be unknown.

The first power-up occurs on 20 dots left in level 1 and on 120 from level 19 on (there are 244 total dots in the classic maze). It will improve Blinky’s speed increasing it by 5%, matching Pacman’s speed. Moreover, in *Scatter* state he will target the same tile he would in *Chase* state.

The second power-up occurs on 10 dots left in level 1 and on 60 from level 19 on, and will increase Blinky’s speed of an additional 5%, thus becoming faster than Pacman.

IV.1.1.2 The ambusher ghost

The pink ghost is called *Pinky* and his personality is described as **speedy**. This is a considerable departure from his Japanese personality description, which is 待ち伏せ (*machibuse*) which translates as "**ambusher**". The English adaptation of the name is misleading because every ghost has the same speed in the game; except, as seen, for Blinky in *Cruise Elroy* mode. However, the “speedy” description probably refers to his attitude to close Pacman frontally. Pinky will join the game short after Blinky in the first level and will immediately instead later on.

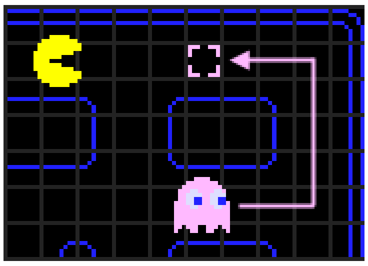


Figure 16 Pinky's chase target

In *Scatter* state, Pinky will roam around the top-left corner of the maze; while in *Chase* state he will target the tile that is 4 tiles ahead from where Pac-man is, in direction of where he is looking; this usually causes Pinky to *ambush* the player cutting his road.

However, Pinky also has two “flaws”. The first, most important is a bug in the original 16-bit architecture, that was preserved in future implementation as a reference to the history of the game: on computing the tile that is 4 tiles ahead of Pac-man when he is looking upward, an overflow error occurs that causes the carry of a multiplication to corrupt the *x* value of the target

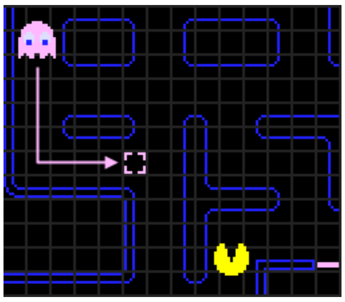


Figure 17 Pinky's chase target overflow bug

tile, causing Pinky's target not to be 4 tiles upward, but 4 tiles upward and 4 on the left[B-29].

The second flaw is logical instead; Pinky will tend to lose at *games of chicken* against Pacman, causing him to turn before he hits him if he can. The reason beneath this bizarre behaviour is that, on coming too close to Pacman, the target tile will be behind Pinky himself. Given that he cannot turn 180° he will take the first left/right turn he finds in order to reach the tile behind himself. Of course, this is possible only if Pinky can take such a turn; nevertheless, most experienced players use this peculiarity to shake Pinky off when he is chasing from behind: facing backward just before Pinky reached an intersection, causing him to take a turn and then continuing the game.

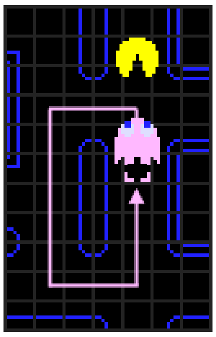


Figure 18 Pinky turning before hitting Pacman

IV.1.1.3 The unpredictable ghost

The blue ghost is nicknamed *Inky* and he is the most difficult to predict during the game. The personality is described as *bashful* in English, which is similar to the Japanese 気紛れ（*kimagure*), "whimsical". He is the third ghost to leave the house, and on the first level he will join after 30 dots are eaten.

The reason why Inky is so hard to predict is because he is the only ghost that uses another ghost's position as factor, Blinky's position to be specific. While on *Scatter* mode he behaves like the others, roaming around the bottom right corner, in *Chase* state he will try to close Pacman between himself and Blinky.

His strategy is to consider 2 tiles ahead from Pacman (in the direction he is looking) and imagining the vector from Blinky to such tile, he doubles it; there is where he will go to. This makes him go very erratically when Blinky is far away from Pacman (or when he is *Eaten*) but will also cause a frustrating surrounding strategy when he is near.

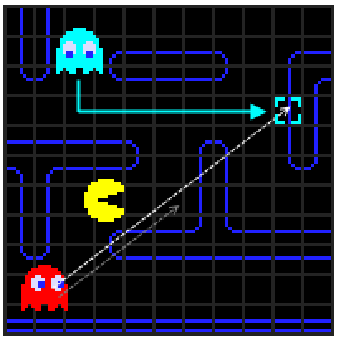


Figure 19 Inky's chase target

Inky, of course, also suffers the same overflow bug of Pinky; but given the shortest distance from the player (2 tiles instead of 4) it is less severe. However, the doubling of the Blinky-tile vector is what causes most of his wandering unpredictably; this means that when Blinky is far away from Pac-man, Inky will also go further away.

IV.1.1.4 The poking ghost

The fourth classic ghost, the orange ghost, is called Clyde and does not exit at all in the first level until over a third of the dots have been eaten. With a name that sticks out from the others, it seems clear that he is the odd one out[15]. His personality is *pokey* in English, which is different but appropriate nonetheless to the Japanese version お惚け (*otoboke*): *feigning ignorance*.

He is the last to leave the ghost house and in *Scatter* state he presides the bottom left corner. When in *Chase* state, he appears to be minding his own business, carelessly poking around Pacman from time to time, and then leaving.

The truth is that Clyde bases his *Chase* target tile on his proximity to Pacman: should he be further than 8 tiles from him, he will target Pacman's own position, just like Blinky; otherwise, he will target the same tile he targets on *Scatter* state, thus returning to his corner.

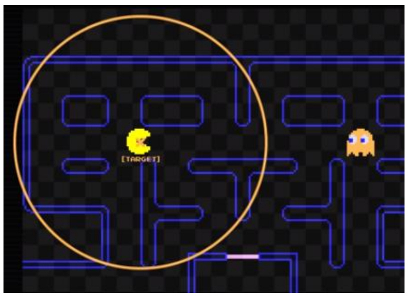


Figure 20 Clyde's chase target

Of course, this means that when he is approximately at 8 tiles of distance he will start alternating its targets and, moreover, that he cannot catch Pacman because he would be too close. However, this implementation works extremely well with the game's constraint that obliges the ghost to complete the corridor they are in before they turn, given the impossibility of turning 180° (the change of target tile does not count as state transition).

This constraint makes Clyde move way further than 8 tiles from Pac-man and also come nearer. Clyde is rarely a treat in per se to the player (unless he is collecting

[15] **Pac-Man Ghosts AI Explained**. *Retro Games Mechanics Explained*: YouTube channel. 2019 www.youtube.com/watch?v=ataGotQ7ir8

dots in his corner), but mostly is a stress factor, being somewhere around the player and not exactly knowing if he is closing in next or straying away.

IV.1.2 Pac-man DDA analysis

As specified, it was desired to design a system that would not directly change the difficulty, making the game naively easier or harder when needed, but a system that would change the behaviour of the game, making the agents more or less aggressive, preserving their identity and nature. When dealing with Pac-Man's ghosts, it was essential to understand how each ghost behaves to create variants (from now on called ***behaviours***) that would have the same nature.

Aside from the ghosts' behaviours, another active factor in a Pac-Man game is the set of constraints that allow or deny certain features, the *180 degrees turn* constraint in example. Moreover, each level present different settings for speed, triggers and timers.

About the speed, this was perhaps the parameter that would vary the most among the different levels. In its original implementation, the *Speed* was a constant common to all *agents* (Pacman and ghosts); however, the specific agent's speed is multiplied by a constant depending on different conditions specified in an environmental table:

PacMan	**Ghosts**
<u>Slows</u> on collecting dots	<u>Heavily slows</u> in tunnel
<u>Accelerates</u> when energized	<u>Slows</u> when frightened
	<u>Hastes</u> when eaten

These all can be (and are) *controllable feature* of the game and were divided in 5 groups: 4 ghosts + the Environment, where this last concern all those features acting on the game itself, the board or that are in general not specific for a single agent. The idea was to evaluate the effectiveness of each behaviour and to select a set of strategy that would match the player expertise.

The observable parameters instead are mainly dependent of the position of the agents in the maze, the completion of a game and the duration of it. Although there could be a huge variety of observables, it is important to focus on the information these could bring and how could this be useful to adaptation.

In example: a standard Pac-Man game usually lasts less than two minutes; given that adaptation would be desired to occur during the game, it is required a short period of observation and adaptation. However, a too brief period would result in discontinuity in agents' behaviour, leading to a dizzy and unsatisfying AI implementation.

A balance was found in observation periods in the range 15 to 30 seconds; although 30 seconds would include both Scatter and Chase phases in observation, it would result in only 2 to 3 adaptations per game (on average).

The observation will be discussed in detail further, but it is interesting to analyse already some of the parameters observed and the reason behind their selection. Distancing from Ebrahimi and Akbarzadeh-T research on Pac-man, that considered the variation of distance from the beginning of the observation to the end as its only parameter to evaluate a ghost effectiveness, it was decided to consider multiple parameters to obtain a more complete insight.

- **The average of the minimum ghost distance** (in the last observation period) is observed to understand the amount of threat caused by the agents to the player.
- **The number of dots eaten** is observed to consider the completion ratio of the game.
- **The dispersion of the dots in the maze** is observed to consider how much the player was able to be modular in cleaning the maze; it was observed that at highest levels, players used to clean the maze in parts, leaving small clusters of dots grouped close to one another.
- **The number of energizers eaten** is observed to consider the safe time-intervals left to the player.
- **The number of lives left** is observed to consider how much the player is close to a Game Over.
- **The time of the observation** is observed to give consequentiality to the observations.

While not all of these were kept during the development, the number of energizer has been reinserted after an insight offered by another game analysis. Released in the same period of Pac-Man, ***Gradius***[L-16] already portrayed a system of Adaptive difficulty; the game is an old-style arcade horizontal-scrolling shooter in which the player has to survive the hordes of enemy spaceships and obstacles coming onto them. This essential gameplay provided a peculiarity called '*power-meter*': basically, upon killing off an entire group of enemies (or an elite one), a power-up capsule would spawn; the player could collect these capsules to increase its power-meter; upon activating the power-up, the game would give the one corresponding to the number of capsules collected (obviously increasingly powerful) and then would reset the power-meter to zero.

The adaptivity in this game was that achieving a high tier power-up, would make the challenge of the following section of the game harder: enemies would be more numerous, they would shot more frequently and could even move faster. The reason behind the correlation between the power-ups and the challenge setting (instead of using the score for example) was that to achieve a high tier power-up, the player must have been able to clear off entire groups of enemies multiple times without needing to activate it. This strategy made the early levels of the game more stimulating for more hardcore players (without impacting the game for less experienced ones) and forced also to use such power-ups instead of storing them.

In Pac-Man, the opposite consideration is doable: power-ups are not achieved but always available in the maze; when a player collects an Energizer, he/she is in troubles and needs to spend it to survive (or to feel safe). Therefore, it is reasonable to consider a player that does not need to spend energizers to survive as a more experienced player, that could use a more demanding challenge.

IV.2 The proposed system: Design

The DDA idea of this research is to substitute a subset of agents' behaviour periodically during the game in order to make the player's ability and the game expected threat as similar as possible.

The system so designed requires 3 main components:

- A set of *agents' behaviours' families*, each graded with a score of effectiveness.
- A *user-performance grading evaluator*.
- An *adaptive sub-system* that works through the previous two.

This last, will be the feature responsible for matching the observed (evaluated) user score with the setup expected threat level, based on the scores of the agents' behaviours' families.

By *family of agents' behaviours*, we refer to a set of **interchangeable** variants of the same behaviour (that would therefore affect the same game dynamic). From each family, an element must be chosen during the game setup; and thus, the representation of the setup itself can be the sequence of behaviours from each family.

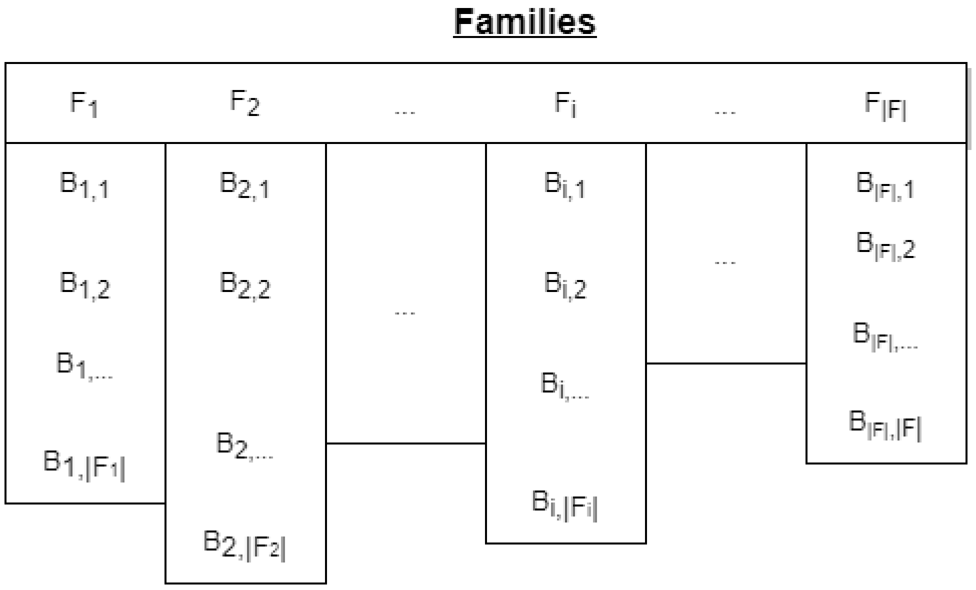


Figure 21 Families and behaviours array

For what concerns the scores, of both the user and the behaviours, designing a system that could evaluate such scores is a harder quest. For starters, it would be way too stretched to handwrite a function that would intuitively assign a lower score on few lives left and higher when few dots are left to collect; this function would be naïve to predict and blind to the correlation unspotted during the design.

A predictable user-scoring function could also make the system vulnerable to "forced adaptation"; that is when a player plays bad on purpose to force the game to become easier (when there is DDA), and then make it through easily. Such strategies are most used during *speed-runs*[16] in order to achieve better times, as happened with Resident Evil 4[L-34] once the DDA system was exploited.

To avoid such complications and to ensure more genuine scores, a solution is to rely on **Reinforcement Learning**, more precisely on **Temporal Difference Learning (TD-Learning)**. This form of unsupervised learning, better detailed in a following sub-paragraph, generates a score for each state; and can therefore be used in both our scenarios.

However, this introduces the need to formalize a concept of ***state***. By its most generic definition, a state is a representation of all the variables in a certain timestamp; however, even in a "simple game" as Pac-Man, this would lead to a huge state dimension, requiring to consider:

- The <u>position</u> and <u>number</u> of every **Dot** and **Energizer** in the maze
- The <u>position</u> and <u>direction</u> of **Pac-Man**
- The <u>position</u>, <u>direction</u> and <u>state</u> of every **Ghost**
- The <u>remaining time</u> of the Energizer effect (**Fright time timer**)
- The <u>timer</u> of the current **state** (Scatter/Chase)
- The <u>number</u> of **lives left**

Plus, the eventual setup configuration (which is here assumed to always be standard).

Aside from the more concise state representation to be found (in accordance with the design requirements) there lies another issue driven from the choice of applying Reinforcement Learning: a game simulation to be run multiple times. This is required because, on its most general concept, Reinforcement Learning is achieved through observation of outcomes of many and many executions.

[16] An attempt to complete a game as quickly as possible, usually timed with a chronometer.

The necessity of a game simulation sets the ground for the necessity of a sample of Pacman players; however, using real players would be insanely expensive in both terms of money, effort and time; consequently, a set of Pacman Player Simulator Bots, or **PacBots**, is required; and it is most important that these bots show different skill levels and reflect gradual ability (or disability) with dealing with in-game mechanics, in example: strong in escaping but poor in collecting Dots.

Summing up, the three main requirements specified at the beginning of this paragraph are resolved in a list of smaller requirements to be fulfilled:

- A set of **PacBots**, with different in-game behaviour
- A set of **behaviour variations** for different dynamics of the ghost, providing (supposedly) higher and lower levels of threat to the player
- A concise representation of ***User state***, to be observed during the game and used as domain of the *user score evaluation function*.
- A concise representation of ***Agent state***, to be observed during the game and used as domain of the *agent behaviour evaluation function*.
- A ***user score evaluation function***, that for each instance of the *user state* gives a score in a given interval
- An ***agent behaviour evaluation function***, that for each instance of the *agent state* gives the score of every behaviour variant in a given interval
- A ***setup adapter*** that given the state representations, substitutes a subset of behaviours with a new subset that should balance the expected threat and the observed *user score*.

For the way it was ideated, this system would place itself under the Adaptive subset of the Pre-fixed Start element of the summary taxonomy proposed (Chapter II.2.5A summary taxonomy for difficulty setting). This because the user has no voice in the dynamic difficulty setting and is not alerted, nor can he fully predict how the challenge will be adjusted.

IV.2.1 Reinforcement Learning

Reinforcement Learning is the study of 'how agents can learn what to do in the absence of labelled examples of what to do'; basically, it differs from a classic supervised learning method for its lack of information about what is supposed to be known. This learning method is extremely useful to learn from zero (for example learning to play a new game) and has shown great results through the years.

Differently from the most notorious Artificial neural networks, Reinforcement Learning learns through on-events feedbacks that basically just tell whether the sequence of choices made brought to a desirable or undesirable event; these feedbacks derive from the concept of *reinforcement* studied in animal psychology for almost 60 years and take therefore the name of ***reinforcements*** (or ***rewards***) themselves.

The idea of reinforcement learning is to use observed rewards to learn an optimal (or nearly optimal) policy for the environment, where we assume prior knowledge of neither model of the environment or the reward function. Reinforcement can occur with different periods according to the application's specifics: it occurs on checkmates in chess, or periodically at the end of a hand in tennis.

A major differentiation in Reinforcement Learning methods is between ***passive learning***, where the agent's policy is fixed and the task is to learn the utilities of states (or state–action pairs) and that could also involve learning a model of the environment; and ***active learning***, where the agent must also learn what to do. [B-27]

A passive Reinforcement Learning agent (passive RL-agent, from now on) has a fixed policy, or behaviour, and its goal is to learn how good the policy is, without knowing how the transitions between states work (*transition model*); in specific, it tries to learn a ***utility function U(s)*** that gives, for each state, a score of 'how good it is to find oneself in such state'.

Given that this learning technique observes a sequence of state transitions culminating in an observable outcome, it is fundamental to understand that the utilities of the states are not independent, and therefore obey to the *Bellman's equation*:

$$U(\boldsymbol{s}) = R(\boldsymbol{s}) + \gamma \sum_{\boldsymbol{s}'} P(\boldsymbol{s}'|\boldsymbol{s})U(\boldsymbol{s}')$$

Which means that the utility of each state equals its own reward plus the expected utility of its successor states; where *R(**s**)* is the reward observed for being in state ***s,*** $\gamma \in]0; 1[$ is a discount factor and ***s'*** is a state such that there exists the transition *s→s'*.

Of course, in a real application the transition model is unknown, and should be computed on the fly using the frequency of observation. Such technique goes by the name ***Adaptive Dynamic Programming (ADP)***; where, upon reaching a new state *s*, ADP adjusts the state's utility to agree with all of the successors that might occur, weighted by their probabilities. The active version of this algorithm shows clearly the typical characteristics of dynamic programming, selecting the action that maximizes the expected utility over all the reachable states.

A variant of ADP is ***Temporal Difference Learning*** (***TD Learning***). It does not learn as fast as ADP and exhibits a higher variability, but it requires much less computation per observation and does not require a transition model. TD Learning uses the observed transitions to adjust the utilities only of the observed states so that they agree with the Bellman's equation. Without considering every possible upcoming state from the current state ***s***, TD Learning does not compute directly the *utility value* ***U(s)*** but it observes the contribute that the transition *s→s'* would cause to *U(s)* without considering its probability (because unknown). This value is commonly called *TD-target*

$$TD - target_{\boldsymbol{s} \to \boldsymbol{s'}} = R(\boldsymbol{s}) + \gamma\, U(\boldsymbol{s'})$$

Learning is then achieved through *bootstrapping*: a portion (α) of the previous utility estimation *U(s)* gets adjusted (replaced) with the observed TD target. The adjustment will be, after any transition observation *s→s'*:

$$U(s) \leftarrow U(s) + \alpha(\, R(s) + \gamma U(s') - U(s)\,)$$

where the portion of adjustment α is a value between 0 and 1 called *learning rate.* Although equivalent, the equation can be written as follows to emphasize the adjustment apported by the TD target.

$$U(s) \leftarrow U(s) - \alpha U(s) + \alpha\, TD - target_{\boldsymbol{s} \to \boldsymbol{s'}}$$

Or

$$U(s) \leftarrow (1-\alpha)U(s) + \alpha\, TD - target_{s\rightarrow s'}$$

The difference between the TD target and the previous utility estimation *U(s)* is also referred to as ***TD error***, this to symbolize that when it tends to zero, the learning is converging.

TD Learning does not need to know the probabilities of state transitions and therefore weights every observation equally; this might seem dangerous when a rare observation with an outlier score occurs, but, then again, these transitions occur rarely; so the average value of *U(s)* will converge properly.

A crucial matter in bootstrapping is the correct use of the *learning parameter* α; should a constant value be used, the last observation would weight more than all the previous one. The most desirable behaviour for the alpha parameter would be for it to soften the adjustment on a state when it has been already adjusted plenty of times. Intuitively, the learning rate should be a function of the number of observation of a state $\boldsymbol{N_s}$, and it should range from 1 inclusive on its first adjustment (so to overwrite the default starting value) to 0 non inclusive, and slowly diminishing as N_s gets higher. A function of this such written that would behave similarly to a hyperbole.

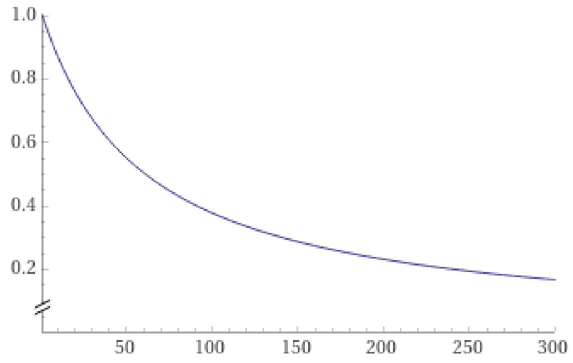


Figure 22 alpha function chart

$$\alpha(N_s) = \frac{k}{k-1+N_s} \text{ with } k \in \mathbb{N}^+$$

Where k is a hyperparameter that makes the convergence faster the smaller it is; while the higher it is, the slower the learning decelerates. Should it be possible to predict the average number of observations for each state n_s and the desired minimum learning rate (namely, on the n_sth encounter) α^*, it would be possible to set a good first estimation of the k hyperparameter as follows:

$$k = \frac{\alpha^*}{1-\alpha^*}(n_s - 1)$$

An active learning agent, on the other hand, must also learn to decide what actions to take; a state transition in this scenario should then also consider the action taken a that triggered said transition $s \overset{a}{\rightarrow} s'$. Consequently, Bellman's equation would have to be rewritten taking into account both the effect of the action over the transitions and the agent's ability to choose an action.

$$U(\boldsymbol{s}) = R(\boldsymbol{s}) + \gamma \max_{a} \sum_{\boldsymbol{s'}} P(\boldsymbol{s'}|\boldsymbol{s},\boldsymbol{a})U(\boldsymbol{s'})$$

In our case, we are not actually interested in active learning because we assume periodical changes of the policies; so we would most likely want to know the utilities of finding in any state for any given policy (***behaviour***).

IV.2.2 State abstraction

As anticipated, an abstraction of the concept of state is required because a complete state model would be computationally unsustainable. Moreover, the smaller the state abstraction is, the least number of states exists; therefore, we need balance between density of information of the state and number of states to consider.

To begin, it was decided to consider two different state abstractions: a **user state** and a **behaviour state**. The player state is necessary to evaluate the *user score*, so it should contain dimensions about the expertise of the player; the behaviour state, instead, is required to evaluate the *behaviours' score*, so it contains information about the pressure exerted on the player by the current setup.

About the *user state*, we define it as a quadruple: $s_{user} = (p,\ d,\ e,\ l) \in P \times D \times E \times L$; where:

- $\boldsymbol{p} \in \boldsymbol{P}$. is a temporal *phase* dimension.
- $\boldsymbol{d} \in \boldsymbol{D}$. is an interval of the number of collected *dots*
- $\boldsymbol{e} \in \boldsymbol{E}$. is the number of collected *energizers*
- $\boldsymbol{l} \in \boldsymbol{L}$. is the remaining number of *lives left*

and

- ***P*** **= {0s-15s; 15s-30s; 30s-45s; 45s-1m; 1m+ }**.

A 15 seconds difference between phases is chosen analogue to the observation period; it was initially set to 30, and then 20, seconds because it was supposed that more time was required to make a significative development in the game. However, this was proved wrong in experiments, and it was also observed that a game almost never went over the 1 minute of duration, therefore the initial partition of time that went until the second minute was corrected obtaining both better results and shorter times.

- ***D = { 0-75; 75-150; 150-175; 175-200; 200-225; 225+ }***.

  Considering that in the classic Pacman maze there were 244 dots (energizers included) it was originally decided to divide the intervals every 50 dots. However, experiments showed that in the first seconds of the game dots are quickly collected, resulting in 1 observation on a thousand of less than 50 dots collected in the first phase (first 15 seconds).

  This is actually understandable for two reasons: first, at the beginning of the game the maze is full of dots and Pacman is literally surrounded by them, causing every movement to collect them. Second, in the first seconds of the game the ghosts are in *Scatter* mode, so they do not actually bother the player; moreover, the first two ghost to leave the house are those who roam the upper side of the maze, leaving the lower side (where the player starts) completely unguarded.

  Originally the abstraction of the ***D*** interval was *{ 0-75; 75-150; 150-200; 200+ }*, but it was observed that the last hundred of dots are collected with much more struggle and the new more dense format offered far better results.

  The number of collected dots is an important parameter that shows how much the player is close to finish the game (to win).

- ***E = { 0; 1; 2; 3; 4 }***

  In the classical maze there are 4 energizers, so it was decided to consider all the 5 possible occurrences. The number of collected energizers indicates how many safe windows are left to the player.

- ***L* = { 1; 2; 3 }**

  The number of lives left was a must, it clearly shows how close the player is to losing a game, and it is a easily interpretable parameter. Given that in the original game the player starts with 3 lives the number was kept. However, a major difference from the classic game, is that we did not keep the playing score that was originally used to grant the player extra lives.

For a total of 450 possible *user states*.

On the other hand, about the *behaviour state*, we define it as a triple: $s_{behaviour}^{b} = (p,\ d,\ l) \in P \times D \times L$; where:

- ***b*** ∈ ***B***. is the evaluated *behaviour* (check on ***IV.3.2 Behaviours***).
- ***p*** ∈ ***P***. is a temporal *phase* dimension.
- ***d*** ∈ ***D***. is an interval of the number of collected *dots*
- ***l*** ∈ ***L***. is the number of *lives left*

and

- ***P* = {0s-15s; 15s-30s; 30s-45s; 45s-1m; 1m+ }**.
- ***D = { 0-75; 75-150; 150-175; 175-200; 200-225; 225+ }***.
- ***L = {1, 2, 3}***

As it emerged from the tests, although running a considerable number, many of the states of both kinds did never occur. This is understandable for (nearly) implausible states such as ***s***$s_{user} =$ *(1m45s+; 0-75; 4; 3),* but would also occur on other states that would just happen to occur rarely; for this reason it was decided to set a minimum number of observations for any state, those that would not occur enough times would be simulated starting games in non-beginning situation. We refer to this approach as: **Rare states simulation**, and we will discuss it more further on.

Originally, the states' abstractions evaluated also the dispersion of the dots and the average minimum distance from the ghost (respectively the user state and the behaviour state), however the latter could not be simulated and was therefore hard to evaluate without corrupting the learning process; the former, on the other hand, could be simulated and showed interesting results too, but the effort required was far greater than it was worth it, so it was removed to save computation time. The removal of these two dimension allowed the reduction of the observation time from 20 seconds to 15, apporting great results and still reducing the computation time and learning table size.

For clarity sake, we will refer to the intervals here introduced as *dimensions* from now on.

IV.3 The proposed system: Development

The main protagonists of the observable effect of this research are the implemented *Behaviours' Families*: that is, the collection of variants of strategies that the game can choose to use against a player, grouped by their *Family*.

The decision to orientate this DDA research on variations of (hand-coded) strategies is what differentiates this work from the studies of the last decade; in fact, in these researches, the adaptation was usually brought by automatic calibration of statistics (creating an automatic *balancing*) or by NPCs learning how to play at different levels of aggressivity. The problems of these approaches are: first, a non-natural behaviour of the NPCs, that will learn how to play without any temperamental indole that created the uniqueness and the personality of an NPC; and second, removes the artistic touch of the designers that can decide how to balance either not only changing the numerical values or altering them purposely to ensure an experience.

Of course, even in these scenarios, adaptive difficulty is a fascinating achievement, nevertheless it is interesting to explore another path. The idea in this dissertation was to find a compromise between the artistic ability of the designer and the numerical tuning of adaptive difficulty; the strategies variants are created by designer with complete freedom and not automatically created (as in many studies with Reinforcement Learning and Artificial Neural Networks), however, what the system does autonomously is to evaluate such strategies and compare the player evaluation with the average of the game setup (computed from the evaluation of the strategies currently in use). This also leaves room for potential patches, new strategies and tuning.

Of course, it is consequently important to ensure both a good model of scores evaluation and a creative and efficient (on different levels) variety of behaviours for each family. During this phase, the experimentation can outline which parameters of each strategy are more sensitive to alteration, which PacBots behave too good or too poorly to be realistic and all the other aspects typical of a game balancing design.

IV.3.1 PacBots

About the PacBots design, that will be responsible of a reliable game simulation, a few words should be spent. For starters, to achieve a consistent variety of PacBots, it was decided to design them in *components*; each that will affect a different attitude of the bot.

For modularity, it was decided to design the PacBot as a player that continuously decides the next direction; essentially like the human player that, in each tile, can only decide to change the direction.

To achieve this, PacBots are compound of only three elements:

- A *Wander Direction Getter* ***W***
- An *Escape Direction Getter* ***E***
- An *alert threshold* $\boldsymbol{\tau}$

The ‘Direction Getter’ modules are functions that, given the configuration of the game board, return a direction $\boldsymbol{d} \in \boldsymbol{D}, \textbf{with } \boldsymbol{D} = \{\boldsymbol{Up}, \boldsymbol{Left}, \boldsymbol{Down}, \boldsymbol{Right}\}$. Therefore, the behaviour of a PacBot is (with due simplifications) a continuous update of the *direction* field as follows:

$$next\_direction = \begin{cases} \boldsymbol{W}(), & min\_dist > \boldsymbol{\tau} \\ \boldsymbol{E}(), & else \end{cases}$$

with

$$min\_dist = \min_{ghost \in \{\boldsymbol{Blinky}, \boldsymbol{Pinky}, \boldsymbol{Inky}, \boldsymbol{Clyde}\}} (\, distance(\boldsymbol{PacMan}, ghost)\,)$$

Despite this solution being effective, an upgrade was ideated to erase the shaking problem: on escaping from ghosts the PacBots will find themselves usually jumping closer and further than τ repeatedly in subsequent frames. Although this rarely ends up in Pacman getting caught, it usually reduces the effect of the escape module; therefore, a solution was implemented that uses two different threshold values: τ_{min} and τ_{max}. These will be used according to the last getter invoked: should it be the *Wander* then the PacBot would consider himself not in danger and will use τ_{min}, should it have been *Escape* then he will consider himself out of immediate danger, but not safe yet, therefore he will use τ_{max}.

$$next\_direction = \begin{cases} \boldsymbol{W}();\ \tau = \tau_{min}, & min\_dist > \tau \\ \boldsymbol{E}();\ \tau = \tau_{max}, & else \end{cases}$$

This strategy is called 'Hysteresis' (from ancient Greek *ὑστέρησις*: delay) and it is the dependence of the state of a system on its history. Given that $\tau_{min} < \tau_{max}$, τ_{max}'s value has been initialized with a positive offset from τ_{min} in the PacBot constructor.

Wander Direction Getters should work to collect the remaining dots and, therefore, complete the game. As such, three different solutions were designed.

- ***Random Turns***: The PacBot would decide a new direction only on intersections (crossroads); this will be selected randomly among the available ones but will automatically exclude the one from where it is coming.
  This last constraint was implemented to avoid the "roaming around the start tile" behaviour that was frequent. Moreover, it was decided to compute a new direction only on crossroads, differently from the *Random Bot* portrayed in Ebrahimi and Akbarzadeh-T's research on Self-organized-systems, because it showed almost no movement and was too unnatural. This component has proven to behave poorly, as expected, presenting an almost 0% win rate, and was therefore not used in scores evaluation.
- ***Pellet Gatherer (k)***: This second is a parametric evaluator in the *k* value that corresponds to the "look ahead distance". As the previous one, it evaluates a new direction only on crossroads; every available direction (i.e. not a wall) gets a score directly proportional to the number of dots encountered in *k* steps in that direction. These scores determine then the probability of being

selected.

When exploring any *n* steps ahead in a given direction, three rules are applied. First, every reachable tile is explored; that means that, on encountering a crossroad, all turns are taken. Second, the direction from where you are coming is not considered, because it would only compute more than once the same tile. Third, a closer dot weighs more than a far one. Finally, when all the scores are collected, the new direction is selected using the *softmax function* to generate a probability in]0; 1[for each (available) direction. For the same reason discussed with the previous variant, the direction from where the bot is coming gets its score halved before the softmax evaluation. The *k* values that showed best results were 4, 7 and 10; although, the smaller *k* the more probable the bot would roam randomly when far from a dot (frequent in endgame), a higher *k* value would result in the bot considering a too dispersive gathering solution, with a heavy loss of effectiveness.

A minor optimization has been to increment the *k* value after every search that found no dots in proximity, and resetting it when one is found.

- ***Pellet BFS***: This solution was the obvious best behaving; this bot would consider the shortest path to a dot using the BFS algorithm. Although the maze should actually be represented as weighted (given that Pacman slows under certain conditions), it is important to notice that such slowing and hasting factor are actually the dots themselves; therefore, a BFS implementation fits perfectly for the situation. In order to avoid excessive re-computation of the same instance, the path found is saved as a queue of 'directions to take'; consequently, the path is computed only if the queue is empty or the previous state was not Wander (because during the Escape the position might have changed), under all the other circumstances a new direction is selected upon entering any new tile of the maze. This *Wander direction getter* provided an almost 70% win rate against the standard setup of the game.
- ***Score Based (k)***: This last was inspired again by Ebrahimi and Akbarzadeh-T's research that required too the development of PacBots; one of these was

the *Cost Based* which evaluated each tile adjacent to Pacman with a cost. The *Score Based Wander* (and *Escape*) variants proposed are based on a similar idea: the adjacent tiles are given a score that gets lower the riskier it gets, from the lowest to the highest these are: *Ghost*, *Near Ghost*, *Wall*(unused), *Empty, Tunnel*, *Dot*, *Energizer*. This direction getter also works looking *k* steps ahead and weighting the probability of taking any direction according to the score of such.

The idea of this variant is to avoid the extreme behaviour of the PacBot that completely ignore the dots while escaping and completely ignore the ghosts while wandering (typical of the other combinations). Despite the expected useful idea, this variant showed abysmal results and was therefore removed from the combinations.

Escape Direction Getters, on the other hand, should work to evade from nearby ghosts. Their behaviour was designed to consider, for each available direction, a score directly proportional to the minimum distance to a ghost; the direction with the best (highest) score is then selected.

An improvement to this strategy was to implement a look-ahead parameter. In the same fashion used by the *PelletGatherer* wander direction getter, the escape direction getter would consider, for each direction, every tile reachable in *k* steps (ignoring turning backs). Again, the scores will be weighted according to the imminence of threat: closer tiles have higher scores.

About the Escape Direction Getters, what really varies from one another is the concept of distance between tiles.

- ***Square Euclidean Distance***: Square of the classic n-dimensional (here 2-dimensional) Euclidean distance; is the length of a line segment from a tile to another. The distance is squared because the square root application to a formula is an unnecessary computational effort given that is monotonous.
- ***Toroidal Squared Euclidean Distance***: Same as the *Squared Euclidean Distance*, but this one considers the toroidal connection between the opposite sides of the maze. Although it would seem more accurate, the presence, in the

classic maze, of a single toroidal connection makes this extra care almost worthless if not, in some cases, deleterious.

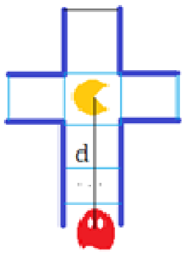
- ***Manhattan Product Distance***: this distance is considered to encourage turns over straight routes, in opposition to the habit of the Euclidean distance to keep the chaser directly behind. To do so, a variation of the *Manhattan Distance* is used:

$$dist_{Manhattan\ Product}(\boldsymbol{src}, \boldsymbol{dest}) = (1 + |\boldsymbol{src.x} - \boldsymbol{dest.x}|) * (1 + |\boldsymbol{src.y} - \boldsymbol{dest.y}|)$$

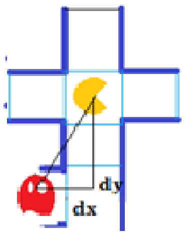
  This formula is maximized when the difference among the axis is the same (fixated the *Manhattan Distance*); therefore, when the chaser is right behind, this distance will cause the PacBot to take turns.

- ***Manhattan Distance***: The classic *Manhattan distance* was not implemented due to its poor effectiveness. Moreover, as could be easily proved, it would give the same score to each direction except the one toward the ghost in the first scenario of the following image, and would behave exactly like the Squared Euclidean in the second one.

- ***Manhattan Product + Squared Euclidean*** and ***Manhattan Product + Toroidal Squared Euclidean***: These two variants, as the name specifies consider the distance as the sum of the *Manhattan Product* and the (*Toroidal*) *Squared Euclidean Distance*. As the following image shows, this distance behaves exactly like the Squared (Toroidal) Euclidean; therefore it was decided to remove it in order to diminish the number of different bots.



	$Eucl^2$	MaxProd	$Eucl^2$+MaxProd
Up	d^2+2d+1 △	$d+2$	d^2+3d+3 △
Left	$d^2 \quad +1$	$2d+2$ △	d^2+2d+3
Down	d^2-2d+1	d	$d^2 \quad -d+1$
Right	$d^2 \quad +1$	$2d+2$ △	d^2+2d+3



	$Eucl^2$	MaxProd	$Eucl^2$+MaxProd
Up	$d_x^2+d_y^2+2d_y+1$ △	$d_xd_y+2d_x+d_y+2$ △	$d_x^2+d_y^2+d_xd_y+2d_x+3d_y+3$ △
Left	$d_x^2+d_y^2-2d_x+1$	$d_xd_y+d_x$	$d_x^2+d_y^2+d_xd_y-d_x \quad +1$
Down	$d_x^2+d_y^2-2d_y+1$	$d_xd_y \quad +d_y$	$d_x^2+d_y^2+d_xd_y \quad -d_y+1$
Right	$d_x^2+d_y^2+2d_x+1$ △	$d_xd_y+d_x+2d_y+2$ △	$d_x^2+d_y^2+d_xd_y+3d_x+2d_y+3$ △

Up	$d_y > d_x$	Up	$d_x > d_y$	Up	$d_y > d_x$
Right	else	Right	else	Right	else

Figure 23 Ghosts' distance functions effects

- ***BFS***: This is the most accurate and represents the length of the shortest path from a tile to another. However, this is also the most effortful and could cause a significant slow down in simulations; therefore, an optimization was made that would store the distance matrix and the exploring queue for each ghost, this way the lookahead parameter barely affects performance anymore. As it could be expected, this last distance provides the best results.
- ***Score Based (k)***: This last is identical to the homonymous in the Wander section. Thanks to the existence of this strategy in both wander and escape direction getters there will be PacBots combination that still take into account the dots while escaping and vice versa. The only difference with the Wander direction getter is that this does not reduce the probability of taking a 180° turn and the other does. Differently from its Wander counterpart, this variant behaved averagely, showing statistics similar to the Squared Euclidean Distance.

Last, the *alert threshold* τ has shown extremely poor results on values higher than 5; although unexpected, it was observed that a minor variation in this hyper-parameter would provoke a major change in the win rate; for these reasons, the *alert threshold* values selected were 2, 2.5 and 3, while the τ_{max} thresholds were 1, 2 and 3; although extremely close to one another, the results can vary significantly among these.

IV.3.2 Families and behaviours

About the behaviours, they are grouped in 14 families, although it is always possible to add more. Every family can have a different number of behaviour variants and this too are eligible for new adds or removal. The families could be imagined in 5 groups, based on the agent they affect, but this schematisation is only intuitive and has no visibility in neither design nor implementation.

The first macro-group collects those families that concern all ghosts at once or the environment of the game and are therefore called *Environmental*. These are:

$\boldsymbol{Ghost\ Speed}: \{-5\%;\ -7.5\%;\ -2.5\%;\ 0\%;\ +2.5\%;\ +5\%\}$

> As observed by the original level design of Pac-Man, the ghost speed is one of the most drastic parameter that would change from one level to another. The values it can assume are to be summed to the fixated parameter: ***Pacman Speed: 0.8*** (which was excluded from the adaptation because it felt weird that the player speed would change in the middle of a game); <u>all the other speeds are obtained from applying a formula to the previous two and are therefore passively adjusted</u>.
>
> The original range of this parameter was from -5% to +5%, because it considered no difficulties beneath the level 1 of the game; For this reason, a -7.5% was added. The -2.5% and +2.5% variants were inexistent in the original setup of the levels but were added to allow a more gradual variation. This family is one of the most effective for heavy adaptation.

Just like variations of the Pacman Speed value were discarded from the controllable features because it was weird to alter the player speed directly, so it was thought with the global parameter SPEED. The idea behind the possibility of including it among the families (of controllable features) came observing that many unexperienced players could not process a strategy fast enough, and most often would decide to take a turn too late. The introduction of a global speed family could partially mend this issue; however, it was both weird during the game and was harder to simulate its effect with Pacbots, given that they do not have a counterpart of the human reflexes. The family was therefore discarded.

$\boldsymbol{Up - turns\ allowed}$: $\{No;\ Yes\}$

As anticipated, in the classic PacMan game there was a constraint for the ghosts that forbid them to take upturns in certain specific tiles; this family concerns that constraint allowing it to be active as default or to disable it and to allow those turns. Surprisingly, although no PacBot used this knowledge in their favour, disactivating this constraint provided a major diminishment of the win rate.

$\boldsymbol{Ghost\ direction\ evaluator}$: $\{Euclidean2;\ Manhattan\}$

As in the original game, Ghost only decide their next target-tile; but the whole path is decided upon encountering crossways, when ghosts check which direction minimizes the distance from theirs target. While the standard uses the Euclidean distance (implemented as Euclidean Squared distance), a variant with the more naïve Manhattan distance is possible.

This weakening option is quite effective because it alters all the ghosts at once; it was conjectured to implement a Toroidal Euclidean Distance and a BFS variant that would be more aggressive; however, the setup was short of weakening families and behaviours therefore, given that these both would be more aggressive, they were discarded.

$\boldsymbol{Fright\ time}$: $\{5s;\ 5.5s;\ 4.5s;\ 4s\}$

This parameter most distinguish among different levels of the classic implementation of Pac-Man; although it was hypothesized that this could be a too much drastic change in the middle of a game, it was decided to experiment how it would behave in a real application and the results were surprisingly good. It still makes a great difference when selected for adaptation, but also shows averagely longer games on smaller values, which is interesting.

The second macro-group concerns the red ghost *Blinky* and contains variants of the Chase and Scatter functions (as all ghosts will) but also the variants concerning the *Cruise Elroy* mode:

$\boldsymbol{Elroy\ 1\ dots\ left\ activation}$: $\{20;\ 30;\ 50;\ 70;\ 100\}$

Blinky is the most annoying adversary in PacMan, due to his nature of chasing restlessly the player. Such aggressive chase behaviour is given by its furious phase called *Cruise Elroy* in which he will behave as in chase mode even during the Scatter phase. As expected, anticipating the activation of the first level of Cruise Elroy a great amount of threat was added; moreover, the behaviours in this family are extremely related to the DotsCollected parameter in the state abstraction and grant a spread variety of levels of difficulty.

$\boldsymbol{Elroy\ always\ chases\ activation} : \{1;\ \ 2;\ \ 3\}$

This family sets the Cruise Elroy level on which Blinky starts chasing even in Scatter phase. Alterations on this family cause a major change in the game dynamics and usually occur when the user is extremely struggling.

$\boldsymbol{Blinky\ Chase\ tile\ getter} : \{Blinky\ std\ chase,$
$Blinky\ Chase\ Roam(2),\quad Blinky\ Chase\ Roam(4),$
$Blinky\ Chase\ Roam(5)\}$

Being Blinky's chase behaviour extremely simple, and extremely aggressive, a single variant was implemented: *Blinky Chase Roam(k)*; which makes him chase not the exact position of Pacman, but a random position in a square around Pacman of k tiles in all directions.

$\boldsymbol{Blinky\ Scatter\ tile\ getter}: \{Blinky\ std\ scatter,$
$Blinky\ Random\ Scatter\ (Up\_Right),$
$Blinky\ Random\ Scatter\ (Out\ House\ NE),$
$Blinky\ Random\ Scatter\ (Around\ House)\}$

About the scatter tile getter functions, all the ghosts will show the same: the standard, as it used to be in the original game, and the three random. This three are called random because they select a random tile in a given zone, and they always are: Their whole quadrant, the zone outside of the house in direction of their quadrant, and the rectangular loop around the house.

Apart from *Around House* which behaves a little worse than the average, all these variants produce minor, but noticeable, difficulty adjustment.

The third macro-group concerns the pink ghost *Pinky*:

$\boldsymbol{Pinky\ Chase\ tile\ getter}:\ \{Pinky\ std\ chase,\quad Pinky\ std\ chase(2),$
$Pinky\ std\ chase(6),\ \ Pinky\ std\ chase(8),$
$Pinky\ Chase\ Bug\ fix(4),\quad Pinky\ Chase\ Bug\ fix(6),$
$Pinky\ Chase\ Next\ intersection\}$

Pinky’s chase behaviour caused him to ambush the player from the front, chasing a few steps ahead of him; the variants proposed were three (plus the parameter tuning):

The standard chase is kept with a variation of ± 2 on the chase ahead parameter. It was seen that on 6, Pinky was marginally less effective, and on 8 it was more significantly less effective (but still not a dummy obstacle).

The bug fix chase is the same of the standard but provides correction of the upward bug portrayed by both Pinky and Inky (see ***IV.1.1 Original Ghosts Behaviour***). Observed that with a chase ahead parameter of 2 it behaved almost exactly as with 4 it was removed. With 6 and 8 however, it showed a diminishing in aggressivity.

The *Next intersection Chase* is, on the other hand, the most effective and aggressive: it makes Pinky chase the next intersection that Pacman is going to reach, given the direction that he is going to, regardless of its distance. Should Pinky already be in such intersection, he starts walking the corridor towards Pacman, granting, almost for sure, a life lost. This last, although extremely aggressive, does not change the idea and the temperament of the ghost but makes it smarter and way meaner.

$\boldsymbol{Pinky\ Scatter\ tile\ getter}: \{Pinky\ std\ scatter,$
$Pinky\ Random\ Scatter\ (Up\_Left),$
$Pinky\ Random\ Scatter\ (Out\ House\ NW),$
$Pinky\ Random\ Scatter\ (Around\ House)\}$

The fourth macro-group concerns the blue ghost *Inky*:

$\boldsymbol{Inky\ Chase\ tile\ getter}:\ \{Inky\ std\ chase,\quad Inky\ std\ chase(5),$
$Inky\ std\ chase(6),\quad Inky\ Chase\ Bug\ fix(0),$
$Inky\ Chase\ Bug\ fix(6),\quad Inky\ chase\ clipped(2.5),$
$Inky\ chase\ clipped(4),\quad Inky\ chase\ clipped(6)\}$

Inky’s chase behaviour is more difficult to custom, especially given its correlation with Blinky; nevertheless 3 different variants are implemented.

The standard chase is kept alongside with a version showing 5 and 6 as the offset parameter.

The bug fix chase is the same of the standard but provides correction of the upward bug portrayed by both Pinky and Inky (see ***IV.1.1 Original Ghosts Behaviour***), as above. Although, with zero as parameter, it does not really fix any bug it is kept in this version because it computes faster than its counterpart; with 6 as parameter however, it is a little less precise than the original, but still a bit more than the standard with 6 as parameter.

The *Clipped Chase* is, on the other hand, more innovative and better behaving: it makes Inky behave exactly as in the standard version, but the reflected vector is now forced to be shorter than the given parameter, making Inky less incline to go astray. It is observed to behave good with a maximum reflection length of 4, and still nicely with 2.5, however it degenerates to Pinky’s behaviour as the parameter goes to zero and as standard Inky as it grows.

$\boldsymbol{Inky\ Scatter\ tile\ getter}: \{Inky\ std\ scatter,$
$Inky\ Random\ Scatter\ (Down\_Right),$
$Inky\ Random\ Scatter\ (Out\ House\ SE),$
$Inky\ Random\ Scatter\ (Around\ House)\}$

There was supposed to be another family concerning Inky, which could made him refer not necessarily to Blinky, but was discarded because the propositions were to erratical.

The fifth and last macro-group concerns the orange ghost *Clyde*:

$$\boldsymbol{Clyde\ Chase\ tile\ getter}: \{Clyde\ std\ chase,\quad Clyde\ std\ chase(12), \\ Clyde\ std\ chase(5,3),\quad Clyde\ std\ chase(12,3), \\ Clyde\ closest\ dot\ chase(8),\quad Clyde\ closest\ dot\ chase(12), \\ Clyde\ closest\ dot\ chase(8,3), Clyde\ closest\ dot\ chase(5,3), \\ Clyde\ closest\ energizer\ chase(5), \\ Clyde\ closest\ energizer\ chase(12)\ \}$$

Clyde, as anticipated, behaves weirdly to the eyes of the player: he chases Pacman exactly like Blinky but upon being too close (8 tiles in the standard version) he would run of as in Scatter. The implemented variants kept this attitude by providing 3 small modifications that would make him more, or less, annoying and still keeping him weird.

The standard chase is kept with modification to the minimum distance parameter, and with the employment of a new parameter that will make Clyde chase a few steps ahead of Pacman; this way he is even less predictable, keeping chasing Pacman when behind him or going astray even if further than expected if in front of him.

The *Closest dot Chase* shows the same two parameters indicated above and will behave as said; however, instead of selecting it scatter tile when too close, he will roam around the closest dot to Pacman, making him extremely annoying.

The *Closest energizer Chase* will behave as the previous one with the exception that will roam around the closer energizer instead (if no energizer is left he will chase the closest dot). This difference is of great importance because upon escaping from the ghosts, a player will be most interested in using an energizer to shove them off, but Clyde will make it harder (more aggressive); however, the closest energizer might be far from the rumble of Pacman and its chasers, causing Clyde to become a problem less to the player in struggle (less aggressive). Finally, this strategy is easier to deal with when there are still energizers left, thus rewarding a conscientious economy from the player.

$\boldsymbol{Clyde\ Scatter\ tile\ getter}$: $\{Clyde\ std\ scatter,$

$Clyde\ Random\ Scatter\ (Down\_Left),$

$Clyde\ Random\ Scatter\ (Out\ House\ SW),$

$Clyde\ Random\ Scatter\ (Around\ House)\}$

IV.3.3 Rare states simulation

Given the shape of these states' abstractions and the natural evolution of the game, an issue occurs while attempting to learn the scores. As it is noticeable, although the states are treated as any possible combination of the p, d, l (and eventually e) dimensions, it is intuitive that some of these are rare or even impossible; example given: in the first 15 seconds of the game it is not possible to collect 150 dots (or more), therefore the only feasible pairs of *phase* $\boldsymbol{p}$ and *dots* $\boldsymbol{d}$ intervals for $\boldsymbol{p} = \boldsymbol{0}$ are (p:0, d:0) and (p:0, d:1); same goes for many other combinations including energizers and lives left too.

As anticipated, the proposed solution to this problem is to simulate such states after the standard learning is complete; the idea is to split the learning in two phases: a standard one, that simply executes a huge number of games; and a simulation one, that runs the game not from the beginning but from a artificial start.

This is doable because the TD-Learning algorithm uses only the following state to alter the value of a state.

The issue here is that these states are rare and as such should be treated, they should not affect the other results heavily because their contribute should be weighted by their probability; being this last unknown we must find a way around this inconvenient.

A natural solution comes with the learning rate α behaviour: given that it decreases as the number of encounters grows, it is rightful to assume that any other state encountered from a rare one is either rare or has been encountered enough times to be affected minimally by a simulated encounter. The other solution is to decrease the k hyperparameter used to compute α, this way any new encounter will affect less than it would have done.

With the rare state simulation strategy comes another consideration that is important to make. To achieve such simulation the system loops over all the possible states, that is every possible configuration of the *p, d, l* (and eventually *e*) dimensions, and this is naively done by nesting loops for each of these; however it is important to wonder whether the order in which the states are simulated is important or not (affects somehow the learning or not).

To answer this, we must consider that the dimensions of the states' abstractions, although treated as independent during the transitions, can only move in one direction (each):

- The temporal phase dimension ***P*** <u>always increases by 1</u> from an observation to another, and there is no other way it can behave (it cannot increase by 2 or more between two observations and it cannot stay put or decrease). Therefore, it starts as 0 and periodically goes up.
- The dots collection dimension ***D*** <u>can never increase</u> because, once collected, a dot will be always collected; it can however remain the same from an observation to another or even increase of more than one step at a time. It starts as 0 and eventually grows.
- The energizer collection dimension ***E*** <u>can never increase</u> just like the dots collection one (being *energizers* dots in the first place); it can in fact remain the same from an observation to another or even increase of more than one step at a time too. Exactly like the previous, it starts as 0 and eventually grows.
- The lives left dimension ***L*** <u>can only decrease or remain constant</u> instead. It starts as 2 and on every life lost it decreases until it reaches 0 (0 is not game over yet). It is not rare that from an observation to another this value decreases of more than one unit.

The starting state is then (-, 0, 0, 2), where '-' implies the 0 seconds time phase, preceding the p←0.

Having all the dimension a direction it is understandable that the order in which they are simulated could indeed affect the results obtained. In details, there are two approaches to simulation:

- From the first value for each dimension.
- From the last value for each dimension.

There would be no particular reason to mix the approach starting from the first value for some dimension, and from the last for the others.

These two different approaches have slightly (but interesting) different outcomes:

- Starting from the first value will require less simulations, or at least it is reasonable to suppose that it would; this because by starting from an 'early' state, the simulation could run into other states that did not reach the minimum thresholds of encounters, thus needing fewer simulation by encountering more rare states in one simulation. The reasonability of expecting such phenomenon is understandable with the following example: starting from an impossible state having $p=0$ and $d=3$ it is possible to reach the (otherwise impossible) states having $p=1$ and $d=4$ or $d=5$.
- Starting from the last value will produce better results, or at least it is reasonable to suppose that it would here too; this because these states will converge to their 'true score' before any state could transit towards them, therefore on updating these last the algorithm could use more precise values as expected of the following states. This is even more desirable given that the first states are the most crucial and that the last states (especially over the temporal dimension) are almost never encountered by players.

Upon considering these correlations, it lies on the designer insight to consider more valuable a shorter learning process duration or more accurate values.

IV.3.4 The Learning results

Upon understanding how the automatic learning should work, how the states are to be abstracted and which shall be the behavioural variants we consider that would enrich the game experience and quality, the learning process started. The quest was to learn, for each state (of both user-state and strategy-states) a score indicative of its value; for the user state, such score would have to be interpreted as 'how good it is for the user to find himself in such situation?', while for the strategies: 'how effective is this strategy under these circumstances?'.

About the user score learning, formally its objective is to learn how to approximate an unknown function from $P \times D \times E \times L$ to $[0;\ 1]$. As machine learning usually works, the approach would be to run an enormous amount of simulation to allow the scores to converge to their true values; however, how many times it is not known. Moreover, it is not granted (on the contrary, it is very improbable) that in any reasonable prefixed number of simulations all the states occur with the same frequency, or all of them enough times to let the learning converge.

A solution to this, as anticipated, is to simulate those state that did not occur enough times; nevertheless, it is still to decide a few things:

1. How many 'standard' simulation are to be run (before simulating)?
2. How many encounters of each state at least have to happen?
3. How should the TD-Learning hyperparameters be set?

Not existing exact solution to these questions, these hyperparameters are to be opportunely tuned by observing different results obtained.

In the end, a number of 100 000 standard games were played, with a minimum of 500 encounters per state. The alpha function was called with $k = 100$, meaning that on the 500th encounter of a state around 15% of the value is corrected; while, during the simulation of rare states, given that these occurrences are forced to happen, the alpha function is weakened with $k = 50$, meaning that on the 500th encounter of a state less than 10% of the value is corrected; while the gamma value is set to 0.95. Finally, the reward of being in a non-terminal state is 0.06, symbolizing the praise of not having died (interpreting the PacMan arcade fashion as a game of survivance); there is no reward for being in a terminal state because this would have to be different according to the epilogue (Win/Loss), and are implemented instead as states for which the utility is known: 1 for a 'win' terminal state, and 0 for a 'loss' terminal state

After many attempts, this configuration seemed well enough calibrated, and gave the better results. Being the user scores 450, it would be overly verbose to either report all of them or explain each of them; nevertheless, an extract of these is shown to better explain how these values are to be read.

Phase	Dots	Energizers	Lives left	Score
15-30s	75-150 dots	1 energizer	1 life left	0.163
			2 lives left	0.4
			3 lives left	0.63
		2 energizers	1 life left	0.061
			2 lives left	0.361
			3 lives left	0.559
		3 energizers	1 life left	0.051
			2 lives left	0.343
			3 lives left	0.436
		…		
15-30s	175-200 dots	2 energizers	1 life left	0.357
			2 lives left	0.73
			3 lives left	0.848
		…		
30-45s	75-150 dots	2 energizers	1 life left	0.084
			2 lives left	0.314
			3 lives left	0.534

In this extract, 5 triplets were taken to show how the scores were affected by the variation of each parameter. Each triplet shows the same configuration if not for the number of lives left; as was expectable, the score drastically falls as the number of deaths grows, producing a range wide averagely 0.5.

The first 3 triplets show the same configuration except for the number of energizers already taken; despite collecting energizers is mandatory to win the game, as this value grows the score goes down a little. This happens because upon consuming an energizer there is a short period of impossibility to lose, therefore, the more the player collects these early in the game, the lesser safe periods remain to him. This parameter becomes less and less significative as the time period progresses.

The fourth triplet shows the changing caused by the number of collected dots (from the second triplet, which coincides on everything else). As one could imagine, this parameter mimics the completion ratio of the game, by letting the score grow significantly as it grows. Moreover, as this parameter approaches its max (the end of the game) the scores become more tolerant of the changings in other values, because a victory is close enough to be reachable in few seconds, regardless of the time gone so far, energizers and (a little) even of lives left

The last group of three is used to show the difference caused by the time period parameter; this last shows large diminishing of the score on consecutive values; especially when the dots collected did not grow by the same pace.

About the strategy score, on the other hand, despite the considerations made are still valid, the number of states is enormously bigger; this happens even if the state abstraction is smaller (not considering the energizers left) because all the states are to be considered for each behavioural variant. Being 67 the different behaviour variants, and given that those that compound the standard version of the level will would have the same score (in this implementation) and are therefore learnt only once for all, the possible states configurations are to be considered for 54 different setups; leading to 4860 different scores.

Before delving in the results obtained through reinforcement learning, the hyperparameters (as seen for the user scores) are set as follows: an amount of 10 000 standard games were played <u>for each setup</u>, with a minimum of 500 encounters per state (which was initially 300 but resulted too erratic). The alpha function and gamma value are set as in the user score learning, while the reward for being in a non-terminal state was set to -0.06 (opposite to the User score learning); the reward for a non-terminal state is opposite to the one for the user score learning as well: 0 for a ‘win’ terminal state, and 1 for a ‘loss’ terminal state (keeping in mind that a win for Pacman is a loss for the ghosts and vice versa).

On studying the learning results for the behaviours, three different scores’ sets are shown: The first, is an experiment made where the strategy states were freed of the lives left value; the second, is the result of the learning described and used in the end; and lastly the third, is the result obtained in the stateless version.

The following, the first, was considered among the experiments because it was firstly hypothesized that no strategy depends on the number of lives left for its effectiveness. The results obtained were, however, disappointing due to the imprecision of the reset function used to simulate the rare states; this issue is present in the second (and used) version too and is caused by the lack of information about the number of lives left and energizer collected necessary to set the game in a generic instant. In both cases, these values are randomly set with a uniform distribution.

The solution of considering both these values in the strategy state abstraction was discarded due to the opposite problem that caused such score only to reflect the user score.

Behaviour	**Phase**	**Dots**	**Score**
Elroy 1 dots activation: **30**	15-30 sec	0-75 dots	0.835
		75-150 dots	0.522
		150-200 dots	0.347
		200+ dots	0.292
…			
Elroy 1 dots activation: **70**	15-30 sec	0-75 dots	0.796
		75-150 dots	0.535
		150-200 dots	0.4
		200+ dots	0.507

The 2 quartets reported show the variation over dots of 2 different strategies in the same time interval; these two concern the same family, the activation of the first level of Cruise Elroy, which influences heavily the game difficulty; what is interesting to notice is how in the first 75 dots collected the first strategy is even more effective than the second, despite both being inactive at this time; also, in the last dots interval the second strategy score grows instead of decreasing. These unexpected scores are effects of the imprecision derived from the reset function; in the first case due to the incomplete knowledge of energizers and lives left, while in the second scenario due to the restart in the same position with Cruise Elroy already active.

Behaviour	Phase	Dots	Lives left	Score
Elroy 1 dots activation: **30**	15-30 sec	175-200 dots	0 lives left	0.787
			1 life left	0.389
			2 lives left	0.328
		200-225 dots	0 lives left	0.986
			1 life left	0.895
			2 lives left	0.807
		225+ dots	0 lives left	0.985
			1 life left	0.878
			2 lives left	0.792
…				
Elroy 1 dots activation: **70**	15-30 sec	175-200 dots	0 lives left	0.975
			1 life left	0.529
			2 lives left	0.491
		200-225 dots	0 lives left	0.983
			1 life left	0.95
			2 lives left	0.834
		225+ dots	0 lives left	0.99
			1 life left	0.928
			2 lives left	0.852

The same strategies are shown in the table above; here it is noticeable how more coherent the scores are. The scores of the first strategy are generically lower of those of second in the same states, while both decrease as the number of dots collected grows (because the player is closer to the victory); however, we can observe how slower is this decreasing in the second strategy because the activation of Cruise Elroy befalls on the 3rd interval instead of the 4th (and last). Nonetheless, it is plain how these scores are heavily affected by the user score and mostly they show an averagely global difference among the same states of different strategy; this behaviour gave the final resolution to the stateless implementation that follows.

Behaviour	Score
Ghost Speed: **+0%**	0.6534
Ghost Speed: **+5%**	0.8149
Fright time: **5.5 sec**	0.3815
Fright time: **4 sec**	0.6222
Elroy 1 dots activation: **20 dots**	0.4154
Elroy 1 dots activation: **100 dots**	0.7147

IV.3.5 The Adaptation process

Once completed the offline learning process, the development team should be in possess of the two tables (as files of some form); these are easily interpretable during a game as the user and strategy states are easily computable while the game is running.

It was decided to implement the adaptation process as: *"The decision process to decide (1) which family should undergo a substitution, and (2) what behaviour should substitute the currently in use"*. Basically, this process mimics the 1-flip schema typical of Genetic algorithms mutations and optimization problems neighbourhood.

The current **setup** can in fact be imagined as a Vector:

$$(\boldsymbol{b_1};\ \boldsymbol{b_2};\ \dots;\ \boldsymbol{b_i};\ \dots;\ \boldsymbol{b_{|F|}})\ where\ \begin{matrix}\boldsymbol{F} \text{ is the collection of Families} \\ \forall i\ \ b_i \in F_i\end{matrix}$$

For reasons of performance, b_i is not the behaviour itself but the index from *0* to $|F_i|$ representing that the b_ith variant of the ith family. Remembering that one (and only one) variant is always selected for each family, the starting setup is represented by the all-zeros vector mapping the setup of the original game, here rewritten for clarity.

$$Ghost\ Speed{:}\ -\mathbf{5}\%$$
$$Up - turns\ constraint{:}\ \boldsymbol{active}$$
$$Ghost\ distance\ evaluator{:}\ \boldsymbol{Euclidean2}$$
$$Fright\ Time{:}\ \mathbf{5}\ \boldsymbol{seconds}$$
$$Elroy\ 1\ activation{:}\ \mathbf{20}\ \boldsymbol{dots\ left}$$
$$Elroy\ alway\ chases{:}\ \boldsymbol{level}\ \mathbf{1}$$
$$Blinky\ chase\ tile\ getter{:}\ \boldsymbol{Standard}$$
$$Blinky\ escape\ tile\ getter{:}\ \boldsymbol{Standard}$$
$$Pinky\ chase\ tile\ getter{:}\ \boldsymbol{Standard}$$
$$Pinky\ escape\ tile\ getter{:}\ \boldsymbol{Standard}$$
$$Inky\ chase\ tile\ getter{:}\ \boldsymbol{Standard}$$
$$Inky\ escape\ tile\ getter{:}\ \boldsymbol{Standard}$$
$$Clyde\ chase\ tile\ getter{:}\ \boldsymbol{Standard}$$
$$Clyde\ escape\ tile\ getter{:}\ \boldsymbol{Standard}$$

Finally, before delving in the specifics of the adaptation decision, the process has been designed as independent 1-flip change; this means that should a more drastic change be desired by the designers, the decision could (and should) just be executed more than once. Given the independence of the decision from previous ones, it could happen that on consecutive evaluations the same family is affected (thus nullifying the previous change); nevertheless, this is not only tolerated but appreciated too; but the reason will be discussed after the explanation of the process itself.

Algorithm 1 Adaptation Process

$uState = (p, d, e, l)$ is the user state
$sState = (p, d, l)$ is the strategies state
$uTable$ is a table that for each uState gives a score $\in [0; 1]$
$setup = (b_1, b_2, ..., b_{|F|})$ is the current setup vector

```
1:  procedure ADAPTATION($uState, sState$)
2:      $uScore \leftarrow uTable[uState]$
3:      $setupScore = \frac{1}{|F|} \sum_{f \in F} f.B[setup[f]].score(sState)$
4:      $softmaxDenom\_F \leftarrow 0$
5:      for $f \in F$ do
6:          $softmaxDenom\_B \leftarrow 0$
7:          for $b \in f.B$ do
8:              $b.softmaxScore \leftarrow e^{\lambda(1-|setupScoreIfSubstitute(f,b,sState)-uScore|)}$
9:              $softmaxDenom\_B \mathrel{+}= b.softmaxScore$
10:         $expected\_f \leftarrow 0$
11:         for $b \in f.B$ do
12:             $b.probability \leftarrow \frac{b.softmaxScore}{softmaxDenom\_B}$
13:             $expected\_f \mathrel{+}= b.score(sState) \cdot b.probability$
14:         $f.softmaxScore \leftarrow e^{\lambda(1-|setupExpectedScoreIfSubstitute(f,sState,expected\_f)-uScore|)}$
15:         $softmaxDenom\_F \mathrel{+}= f.softmaxScore$
16:     for $f \in F$ do
17:         $f.probability \leftarrow \frac{f.softmaxScore}{softmaxDenom\_F}$
18:     $f' \leftarrow randomFamily(F)$
19:     $b' \leftarrow randomBehaviour(f')$
20:     $setup[f'] \leftarrow b'$
```

Let us explore the details of what is happening here: this procedure does, as anticipated, a 1-flip switch; choses one of the families and, consequently, one of its behaviours to substitute to the previously in use. In order to do so, it considers how good would it be to select each family as substitution target and how good would it be to select each behaviour of each family; this last actually influences the utility of selecting such family.

For starters, both the *user score* and the *setup score* are evaluated. While the user score is simply read from the *user table* in correspondence of the *user state*, the setup table is considered as the average of the scores of all the currently in use behaviours. That is, given that each family has a behaviour implemented, the sum for each family of the score (given the *strategy state*) of its implemented behaviour.

$$setupScore \leftarrow \frac{\sum_{\boldsymbol{f} \in F} \boldsymbol{f}.B[setup[\boldsymbol{f}]].score(sState)}{|F|}$$

Where *f.B* denotes the set of behaviours of the family *f*, and the *score*(*sState*) function accesses the learnt score table of a behaviour, given the observed state.

Unless specific policies are implemented, all the behaviours are eligible for substitution, as all families are; the probability of selection is directly proportional to the utility of making the swap through the use of a softmax evaluation.

The concept of 'utility of making a substitution' is the key of this decision. Essentially, what would be most desirable is to find a setup *setup'* that exhibits a setup-score closer to the user-score than the current setup; given that we consider 1-flip changes in behaviours we are looking for a substitution *b→b'* such that this difference is reduced; so we start computing how would the setup score vary should any behaviour b of any family f be used instead of the currently in use (given the observed strategy state).

$$setupScore_{\boldsymbol{b}} \leftarrow \frac{setupScore \cdot |F| - f.B[setup[f]].score(sState) + \boldsymbol{b}.score(sState)}{|F|}$$

Intuitively, the new setup score coincides with the previous when the behaviour in exam is the one in use. Nevertheless, it is important to consider those in use too, because the best option for a family could already be the most fit.

The difference between the new setup score and the user score gives a metric of which behaviour is most fit to be in use, with a behaviour being most appropriate the closer it is to zero. Because we want the most desirable to be the most probable, their score is reversed by subtracting it to the maximum achievable: 1. Thus, each behaviour's softmax numerator shall be:

$$b.softmaxScore \leftarrow e^{1-|setupScore_b - userScore|}$$

and

$$b.probability \leftarrow \frac{b.softmaxScore}{\sum_{b' \in f.B} b'.softmaxScore}$$

Known the probability of each behaviour of each family, it possible to compute the expected score of a family, should it be selected for substitution. Intuitively, that would be the sum of all its behaviours multiplied its probability.

$$E[f] \leftarrow \sum_{b \in f.B} b.score(sScore) \cdot b.probability$$

By the same logic used for behaviours, we can know how the selection of a specific family for substitution would affect the setup score, and we can therefore compute a metric of which would bring it closer to the user score.

$$setupScore_f \leftarrow \frac{setupScore \cdot |F| - f.B[setup[f]].score(sState) + E[\boldsymbol{f}]}{|F|}$$

$$f.softmaxScore \leftarrow e^{1-|setupScore_b - userScore|}$$

and

$$f.probability \leftarrow \frac{f.softmaxScore}{\sum_{f' \in F} f'.softmaxScore}$$

It is important to notice that it could occur a substitution with a worse fit behaviour (or family); however, a repeated execution of this procedure more than once per observation (in our case 15 seconds) has extremely high possibility of correcting eventual worsening substitutions, making the setup score eventually converge to the user score; in our implementation, the number of adaptation per observation was set to 5, although 3 worked fairly well too. This possibility is lowered by the improvement described followingly.

IV.3.5.1 The stateless behaviour's score variant

Observing many games, it was possible to notice that the setup score was almost always equal to: $1 - user\_score$; upon studying more in detail this phenomenon it was concluded that the behaviour's scores in most of the states did mimic the user score in that same state.

This was hypothesized during the design phase, specifically, during the states' abstractions design; the concern, back then, was that a strategy did not change its effectiveness according to the time interval or remaining lives, and only a few were related to the number of collected dots (e.g. Elroy's activation, etc…).

Following this theory, it was ascertained that the difference between the win ratio of two strategies was reflected in the difference between the scores of such behaviours in almost all the states; given this, a variant of the behaviours' scores evaluation was developed that would not require a state, but that would keep a single value (for each behaviour) derived by its win ratio in learning phase.

The adaptive algorithm portrayed above was not modified nor affected by this possible variation because the Behaviour class was turned into an abstract class with two specializations: a Tabular one (with a score for each state) and a Single-value one. Having more variants of the system was considered an added value to this research and in testing phase both were used with the respective names: ***adaptive T*** (**T**abular) and ***adaptive SBS*** (**S**tateless **B**ehaviours' **S**core).

IV.4 Improvements and thoughts

Aside from the algorithm and the learning system, improvements were considered that could drive the adaptation and reduce the risk of making bad substitution or, more generally, to reduce the probability of selecting a worse substitution over a good one.

The two improvements described are namely: ***the policies*** and the ***difference enhancement***. While the first filters out behaviours alternatives and families that do not satisfy certain requirements (and that would therefore behave poorly), the second one accentuates the difference between variants that would numerically look too similar.

IV.4.1 The policies

The idea of the policies concept comes from the study of Robin Hunicke already introduced[B-14]; she introduced this term as the paradigm that the adaptive process should follow to reach an aesthetic goal. While she proposed a few, namely: *Comfort, Discomfort and Training Policy*, she also asserted that many more would be necessary.

Starting from this thought, the conceptual idea of 'policy of the adaptive process' has been imagined as something that would drive the adaptation in its decision by keeping a general idea of the target experience desired.

In the system proposed, we considered the eventual policies as filters that would ensure that the alternatives are only among those who can grant such experience.

Formally, the policies could be ranges $[min;\ max] \subseteq [0;\ 1]$ for the behaviours' score and families' expected score upon substitution.

A standard, non-filtering, solution would be a **generic policy** obtained by the range: $[0;\ 1]$ and would behave as if no policy is currently in use, thus permitting every possible solution to be selected.

A **comfort policy**, recycling Hunicke's term, could ensure that, despite the substitution, the new solution will not be too hard by applying a filter: $[0;\ userScore + \varepsilon]$. Opposite to that a **discomfort policy** could ensure a not too easy experience with a filter: $[userScore - \varepsilon;\ 1]$. While a final combination of the two given by the range $[userScore - \varepsilon;\ userScore + \varepsilon]$ could avoid unbalanced situation for which a family has a very aggressive behaviour and another an extremely easy; this could be a **balanced policy**.

Policies could also consider other parameters and become even more interesting, for example, taking inspiration by the Reinforcement Learning exploration versus exploitation balance, policies could take into account the number of times the player has encountered each behaviour. A **confidence policy** would therefore promote behaviours already encountered, while a **training policy** would do the opposite.

Upon testing, it was seen that the use of policies would really narrow down the variants considered providing better adaptations. Two more policies have been designed in this phase that did actually improve greatly the quality of the game: **Converging transition policy StoU** and **BtoU**.

The first, considers only those behaviour variants that are scored between the currently in use and the target score (user score); so, should the userScore be higher than the inUseVariantScore, the range would be: $[\,inUseVariantScore;\, userScore + \varepsilon\,]$; the other way around would be $[\,userScore - \varepsilon;\ inUseVariantScore]$ otherwise. The latter, on the other hand, considers only those variants that are ranked between the setupScore and the target score (user score), but excluding both the extremes. so, should the userScore be higher than the setupScore, the range would be: $]\,setupScore;\ userScore + \varepsilon\,[$; the other way around would be $]\,userScore - \varepsilon;\ setupScore\,[$ otherwise. They behave similarly and both give good results; they were called *Converging* because they only consider those variants with a score between the starting point and the target; *BtoU* stands for ***B**ehaviour **to** **U**ser*, while *StoU* stands for ***S**etup **to** **U**ser*.

The last policy implemented is an improvement of the previous one and was designed upon observing how the algorithm behaved in a real application; the problem observed with the *Converging policies* is that they not only allow, but also encourage heavy transitions. In example, starting with a low score setup (i.e. 0.2) while the user score is high (i.e. 0.8), all the behaviours with a score from 0.2 to 0.8+ε will be considered; therefore, due to the effect that a 0.8 scored behaviour would have on the setup score, most likely a high score behaviour will be selected.

But this would allow extremely harsh transitions, as was often observed from *Ghost speed -5%* to *Ghost speed +5%*. The adopted solution was called ***Smooth transition policy***, and this also had the variants ***BtoU*** and ***StoU***; these would consider only those behaviours whose score was contained between the currently-in-use behaviour score/setup score and the average from such value and the user score.

Summing up, the policies implemented portrayed the following intervals:

Generic	$[\mathbf{0};\ \mathbf{1}]$
Comfort	$[0;\ userScore + \varepsilon]$
Discomfort	$[userScore - \varepsilon;\ 1]$
Balanced	$[userScore - \varepsilon;\ userScore + \varepsilon]$
Converging (StoU)	$[\ min(setup, user - \varepsilon);$ $max(setup, user + \varepsilon)\ ]$
Converging (BtoU)	$[\ min(currBehaviour, user - \varepsilon);$ $max(currBehaviour, user + \varepsilon)\ ]$
Smooth (StoU)	$[\ min(setup, \frac{setup + user}{2} - \varepsilon);$ $max(setup, \frac{setup + user}{2} + \varepsilon)\ ]$
Smooth (BtoU)	$[\ min(setup, \frac{currBehaviour + user}{2} - \varepsilon);$

	$max(setup, \frac{currBehaviour + user}{2} + \varepsilon)$]

Confidence and *Training* policies are not in the table because they would require an alteration of the scores before filtering.

The final configuration of the game, also used in the questionnaire, portrayed the *Smooth transition policy BtoU* policy with ε set to 0.0225 for the Adaptive Tabular version and portrayed the *Smooth transition policy BtoU* policy with ε set to 0.0875 for the Adaptive Stateless Behaviours' Score version; this difference was dictated by the higher difference among scores of the second version.

IV.4.2 The difference enhancement

This step follows the filtering provided by the policies and simply drives the softmax function's result in providing a fair difference between options. It basically just consists in multiplying the exponent in the softmax numerator computing by a scalar $\lambda \geq 1$.

$$b.softmaxScore \leftarrow e^{\lambda(1-|setupScore_b - userScore|)}$$

$$f.softmaxScore \leftarrow e^{\lambda(1-|setupScore_f - userScore|)}$$

To better show the effects of this simple multiplication I am providing a few examples:

Let us notice that the exponent is maximized when the difference between the new setup score and the user score ($|setupScore_b - userScore|$) is 0; thus the exponent ranges from 0 (when the new setup score is completely different from the user score) and 1 (when they are the same). Let us suppose, that a certain family in exam has 3 behavioural variants and that they would make the new setup score differ from the user score respectively of 1, 0 and 0.5.

To give visualization of the possibility of these scores' differences I shall provide an example of the three (for simplicity I will refer to a *family's score* as the score of the active behaviour of such family):

Scenario	**1**	**2**	**3**
Setup	All families have score 0 except for one having b_{score}	All families have score 0 except for one having b_{score}	|F|-1 families have score 0, the others have 1
User Score	1	0	0
Substitution	$b_{score} \to 0$	$b_{score} \to 0$	$1 \to 0$

Interpreting this cases: the first scenario is a worsening substitution, reducing the setup score to the minimum possible (0) when the user score is the highest (1); the second scenario is the best possible, making both the scores coincide; the last scenario portraits nearly half the families to a 0 score, and the substitution in exam would bring another family from 1 to 0, as desirable, (Let us assume that such combination is possible for due clarity). The following table shall express how the exponent and the probability varies between these three options.

variants	b_0	b_1	b_2
$newSetupScore - userScore$	1	0	0.5
Softmax score	$e^0 = 1$	$e^1 = e$	$e^{0.5} = \sqrt{e}$
Probability	~18%	~51%	~31%

It is noticeable that the first variant (b_0), even being the worst possible choice still has almost a chance on five to be selected, while the second one (b_1) that would be the best possible choice has only half the chances. The rub is that e^0 is not so far from e^1; although there would be the options of changing the base e (with a larger value like 10), enlarging the exponent proved to be a better solution. As evidence of this a new table with $\lambda = 3$ follows.

variants	b_0	b_1	b_2
$newSetupScore - userScore$	1	0	0.5
Softmax score	$e^0 = 1$	e^3	$e^{1.5} = e\sqrt{e}$
Probability	~4%	~78.5%	~17.5%

This solution becomes less efficient as the number of options grows, but still remains effective; and this consideration makes the synergy between the filtering of the policies and the difference enhancement even more desirable.

A last example is shown with more realistic differences between new setup score and user score, and with 3 different λ values, for readability already sorted from the worst behavioural variant to the best.

Variants. ($\lambda = 1$)	b_0	b_1	b_2	b_3	b_4
$newSetupScore - userScore$	0.4	0.3	0.25	0.2	0.15
Softmax score	$e^{0.6}$	$e^{0.7}$	$e^{0.75}$	$e^{0.8}$	$e^{0.85}$
Probability	~17.3%	~19.1%	~20.1%	~21.2%	~22.2%
Variants. ($\lambda = 3$)	b_0	b_1	b_2	b_3	b_4
Softmax score	$e^{1.8}$	$e^{2.1}$	$e^{2.25}$	$e^{2.4}$	$e^{2.55}$
Probability	~12.7%	~17.2%	~20%	~23.2%	~27%
Variants. ($\lambda = 5$)	b_0	b_1	b_2	b_3	b_4
Softmax score	e^{3}	$e^{3.5}$	$e^{3.75}$	e^{4}	$e^{4.25}$
Probability	~9.1%	~15%	~19.3%	~24.8%	~31.8%

IV.5 Design and Development alternatives and generalization

Although designed and implemented for PacMan, this adaptive idea is only based on a set of variants for behavioural components and could therefore be applied to a generic game in which it is desired to leave the peculiarity of an NPC attitude, or of an experience, and still tuning the challenge accordingly to the player shown expertise. Moreover, the generic idea of pure learning, by which an AI learns entirely how to play, could still be used as one of the variants.

There were many propositions during the design and the study of the literature, that were considered not perfect for the game decided as test bed but that still looked interesting enough to be considered in a generalization to another game/genre.

First, the families of behaviours could be easily portioned based on the agent they affect (as was shown in the inherent analysis); however, while in this research the adaptation was process performed by an overlooking centralized unit above all the agents, Ebrahimi and Akbarzadeh-T's research on Self-organized-systems[B-24] has shown the benefits of an decentralized system, where every agent could be in charge of examining itself and eventually making a modification in its setup. The modifications would not be communicated to the other and therefore they could all change something or none could.

This would be too heavy on a game like PacMan but would make much more sense in an open environment filled with adversaries and NPCs; and would be an interesting approach for both MMO and Action RPGs.

Another interesting idea that could easily be added to such system is a correction for the states' scores based on the experience of the player. This idea is only appliable and makes sense to those game that are played for longer periods and provide an adaptation not only based on the offline gathered data (as in our example are the PacBots' results), but also by the effects of such states on the specific player that is playing, making the adaptation much more oriented towards the real target.

The score of a certain behaviour, either in a certain state or absolute (Tabular or Single-value Behaviour), would then become a compound of the score learnt offline and the score learnt online; where this last would initially be null or zero. These scores would have to be combined weighting their experience, or how we trust those value to be accurate; an example could be the number of times they were encountered, with the online encounters weighting more.

$$\text{(Single Value) } score_b = \frac{f(N_b^{offline})\, score_b^{offline} + \omega\, f(N_b^{online})\, score_b^{online}}{f(N_b^{offline}) + \omega\, f(N_b^{online})}$$

$$\text{(Tabular) } score_{b,s} = \frac{f(N_{b,s}^{offline})\, score_{b,s}^{offline} + \omega\, f(N_{b,s}^{online})\, score_{b,s}^{online}}{f(N_{b.s}^{offline}) + \omega\, f(N_{b,s}^{online})}$$

Where, N_b and $N_{b,\,s}$ are the number of observation namely for the behaviour *b* and, eventually, the state *s*; the function *f()* is a monotonous crescent function that returns the weights the number of observation, and $\omega > 1$ is a scalar that gives more importance to the online observations. It would be important to give higher value to these last because they would be specific of the player and not generic as the offlines.

Of course, this would require something like an account for the player but could also open the door to a still more advanced player expertise mapping that could be cross-games and even cross-platforms.

Finally, a much more simple and achievable alternative is the ***geometric progression of the scores evaluation***.

IV.5.1 The Geometric progression of the scores evaluation

This strategy, although simple in concept, could help dealing with the dizzy adaptation problem: the system correcting excessively in one direction and, consecutively, excessively in the opposite one to balance. This would be most useful in any adaptive difficulty application, and was thought as alternative to the **Adaptive SBS** variant as correction of the **Adaptive T** weaknesses.

The idea came by observing that a score (user score or setup score) in a given observation is treated as completely independent from its previous value. For clarity:

- Let us call $\boldsymbol{\sigma_t}$ the observed score at time ***t***. That is, the score evaluated by observing the corresponding state abstraction.
- Let us call $\mathbf{S_t}$ the score given to the adaptation process at time ***t***. In the implementation proposed so far $\boldsymbol{\sigma_t}$ and $\mathbf{S_t}$ always coincide.
- We consider **t** a natural timestamp index from 0 to …

So far, we always considered $\forall \mathbf{t} \ \ \mathbf{S_t} = \boldsymbol{\sigma_t}$; however, this made the system completely oblivious of the history of the game. Moreover, it made the issue of getting two completely different scores in consecutive observations much more severe. The adaptive process, in fact, considered only the instantaneous score to process its adaptation, many times ignoring that a player had played good or bad until then.

Treating the scores as geometric progression would mean to consider the previous scores too during the adaptation. In particular, two options were considered implementable.

The first considers keeping a portion $\varphi \in]0;\ 1[$ of the previous score in the current one, therefore we will have:

$$\hat{S}_{\mathbf{0}} = \sigma_{\mathbf{0}}$$

$$\hat{S}_{\mathbf{1}} = \varphi\, \sigma_0 + \sigma_{\mathbf{1}} = \varphi\, \hat{S}_0 + \sigma_{\mathbf{1}}$$

$$\hat{S}_{\mathbf{2}} = \varphi^{\mathbf{2}}\, \sigma_0 + \varphi\, \sigma_1 + \sigma_{\mathbf{2}} = \varphi\, \hat{S}_1 + \sigma_{\mathbf{2}}$$

$$\dots$$

$$\hat{S}_{\boldsymbol{i}} = \sum_{j=0}^{\boldsymbol{i}} \varphi^{i-j}\, \sigma_{\mathrm{j}} = \varphi\, \hat{S}_{\boldsymbol{i}-1} + \sigma_{\boldsymbol{i}}$$

The ^ symbol above the scores given to the adaptation process S_i was put because, as it has been written, these values would overflow the range considered for them (in our scenario [0; 1]). It follows that these temporaneous scores are to be weighted according to their contribute.

$$S_{\boldsymbol{i}} = \frac{\hat{S}_i}{\Phi_i}\text{; where } \Phi_{\boldsymbol{i}} = \sum_{j=0}^{\boldsymbol{i}} \varphi^{\boldsymbol{j}} = \Phi_{\boldsymbol{i}-1} + \varphi^{\boldsymbol{i}}$$

As **t** grows, the early values of S_i will become less significative. Computationally speaking, this approach does not require to store every S_i value, but only the last $\hat{S}$, the last Φ and the *t* index.

The second approach is more similar to the learning rate contribute shown in the TD-Learning algorithm, but has similar effects.

Considering a portion $\boldsymbol{\rho} \in]0;\ 1[$, possibly closer to 1 than 0, of the current score reading and the remaining 1-ρ of the previous score, it does not need to be weighted before being given to the adaptive process.

$$S_{\mathbf{0}} = \sigma_{\mathbf{0}}$$

$$S_{\mathbf{1}} = (1-\rho)\sigma_0 + \rho\sigma_{\mathbf{1}} = (1-\rho)S_{\mathbf{0}} + \rho\sigma_{\mathbf{1}}$$

$$S_{\mathbf{2}} = (1-\rho)^2\sigma_0 + \rho(1-\rho)\sigma_{\mathbf{1}} + \rho\sigma_{\mathbf{2}} = (1-\rho)S_{\mathbf{1}} + \rho\sigma_{\mathbf{2}}$$

$$\dots$$

$$S_i = \sum_{j=0}^{i} \rho^{i-j}(1-\rho)^j \sigma_j = (1-\rho)S_{i-1} + \rho\sigma_i$$

Here again, computationally speaking, it is not required to store every encountered value, only the last one.

It should be observed that these approaches can be reduced to each other with the right parameters, and the only true difference among them is in how they are computed in code.

This approach, in both its versions, would also give the opportunity to compute an ending score for the player and the setup.

Chapter V
Implementation Architecture

Although it is not the focus of this dissertation, a few words should be spent on the code architecture that has been responsible for learning, testing and game simulation. The whole architecture is object oriented and implemented in Java; there has been no need for a Database, although there are two plain text files that are read on need.

As advised in the Game Development sector, elegance of the code has been in part sacrificed for computation speed; that means that many fields were kept *public* instead of creating getters and setters, and many were kept static even they should not logically be. A fair economy in variables has been made, both recycling variables and objects instead of creating new ones and preferring to pay space complexity over time complexity.

Before delving in a few of the classes that were implemented, a brief overlook of the responsibility and purpose of each module could help. For starters, the classes could be categorized in different groups according to their use:

- **The simulation classes**: core of the whole simulation and responsible for most of the communications among groups and initialization.
- **The Agents' classes**: responsible for every decision and state regarding the agents; including both agents as such and sub-modules.
- **The Learning classes**: in charge of everything that concerns the gathering and interpretation of observable data to achieve learning; they are mostly autonomous from the rest and are used in specific executions (runs) to create the scores tables.
- **The DDA classes**: responsible of adaptation runs and features. Thanks to modularity they are really just heavily associated with the core simulation classes.

- **The utility classes**: small data structures and collection of static and logging procedures widely and sparsely used in every classes. Given their small relevance, there won't be a section regarding them.

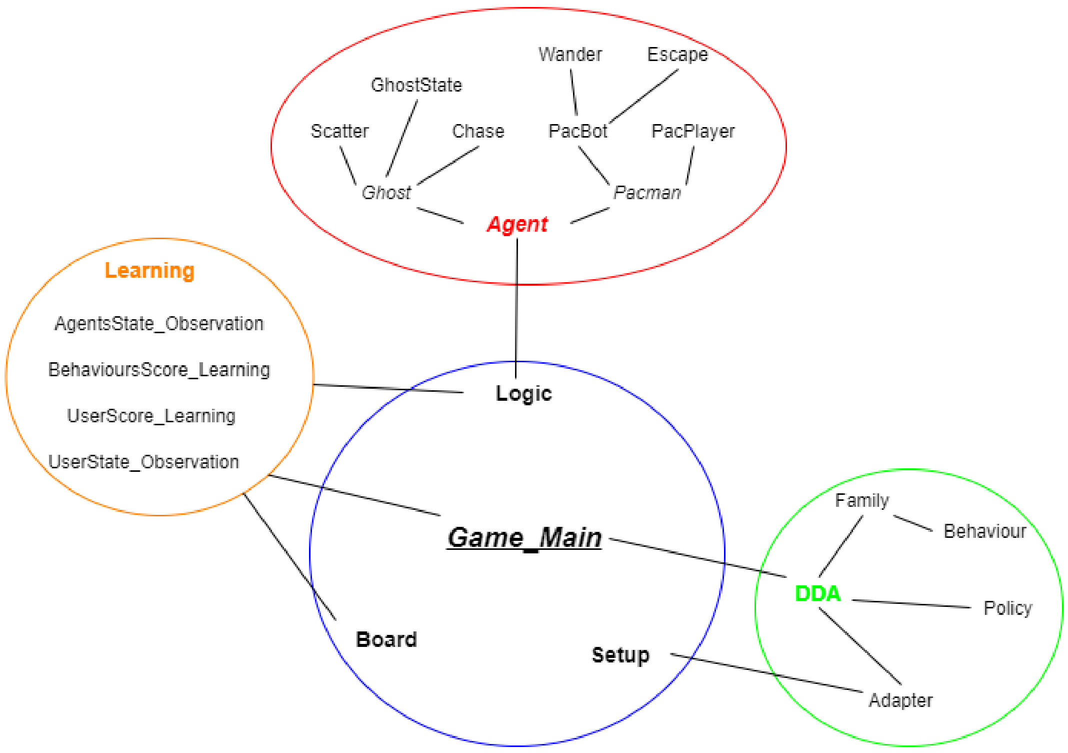


Figure 24 Implementation's subsystems

V.1 The simulation classes

The simulation classes are responsible for the game as it is and for the differentiation between each kind of run/test. These are in the centre of the whole architecture and, in the centre of this group, there is the ***Game_Main*** class.

Game_Main works as a controller between the different modules. It contains the main function that, based on a set of hyperparameters, executes a different run; it also is responsible for the harmonic communication between other classes in its set and other modules. To understand its responsibilities, the run modes are listed and explained:

- *Playable:* The classic Pac-Man game executed with no extra feature.
- *User-score Learning:* A number of games are played and their result are observed to train the *user-performance grading evaluator*.

- *Behaviours-score Learning:* A number of games are played and their result are observed to learn the effectiveness of a certain behaviour.
- *Adaptive:* The classic Pac-Man game is executed with the DDA module active.

While the ***Playable run*** requires no extra feature it is the base from which the other runs are implemented. The main class creates an instance of ***Logic***, a ***Board*** and a ***View*** (which is considered among the *Utilities*); the *Logic* represents a game execution and is responsible for the update of the characters and the game over conditions, it also contains a *Setup* class that contains details of all the thresholds, strategies and callback functions that create the diversity of each level. The *Board* represents the maze and the items in it, it is responsible for reading the maze from an integer matrix and associates to each tile the item in it (*Dot, Energizer, Empty*) and the graph connections with the adjacent tiles; it is also responsible for the maze reset. Finally, the *View* is responsible for the graphical representation of the board and the characters.

The ***User-score Learning*** and the ***Behaviours-score Learning*** (and the unmentioned **Both Learning**) cooperate with the Learning module. The number of games played are specified by two static parameters kept in *Game_Logic*: the first, blankly states how many games must be played; the second, dictates a minimum number of encounters for each state abstraction. Should any state, at the end of the games played, not have been encountered enough times, a game run is simulated from the state conditions specified (the *Board* class is responsible for these simulated resets).

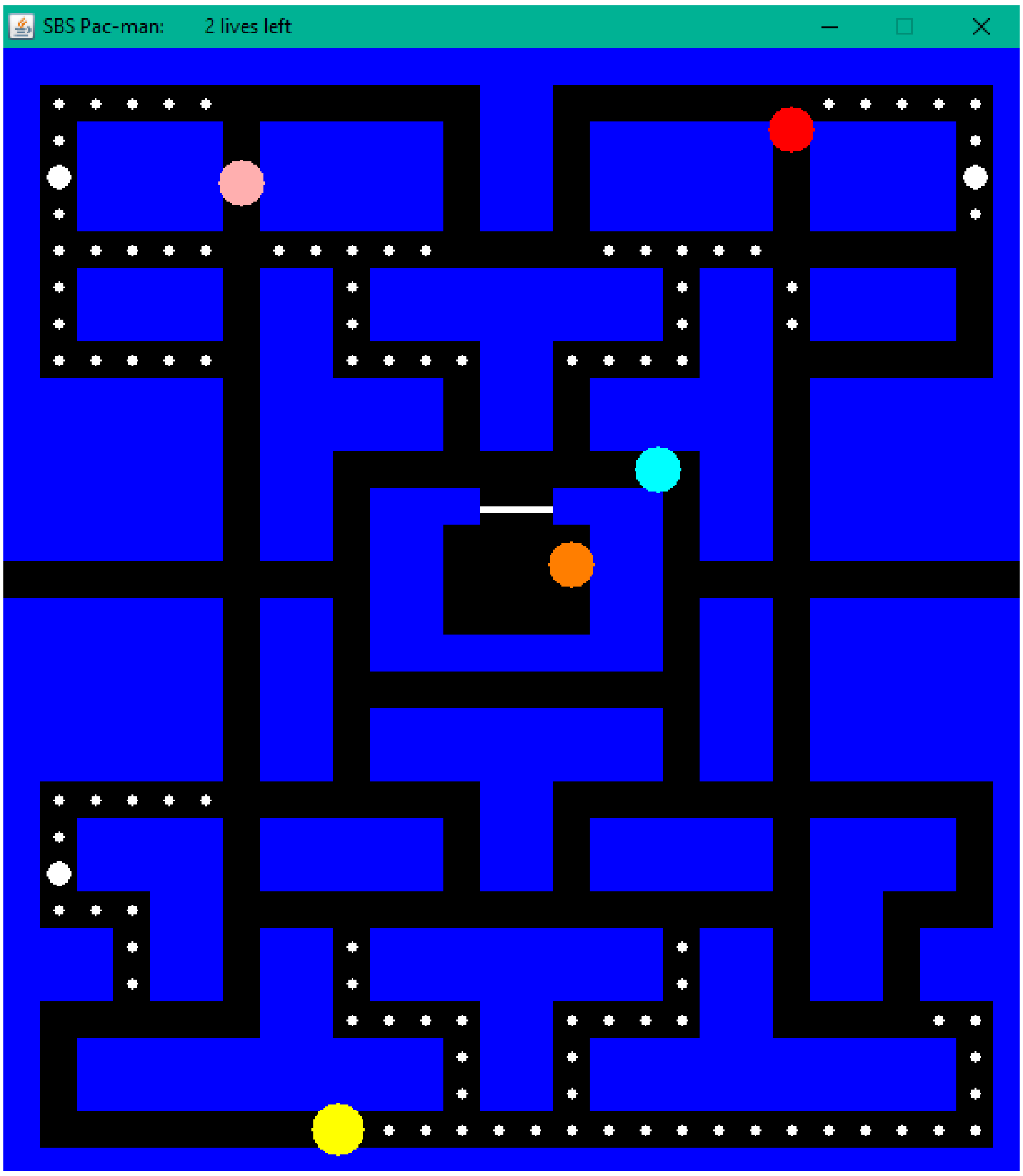


Figure 25 A screenshot of the implemented game

Core of the learning process are the PacBots that simulates various levels of expertise of a Pacman player; a collection of PacBots is created before the tests start and on each game the bot changes. In order to evaluate how the bots behaved, statistics for each PacBot variant are kept and updated through the tests. An inconvenient could occur, during these simulated games, that causes a game to never end; this could happen when the ghosts and the bot are in certain positions that will cause the bot to escape in a certain loop and the ghosts to follow such loop forever. To solve this problem, should game last more than 300 simulated seconds, it is aborted and ignored.

The ***Adaptive run*** behaves exactly like the Playable, but also communicates with the *DDA* module that periodically changes the game setup. Just like the tests run, this requires the *Logic* class to collect and return the state observations.

A note about the *Board* class, it contains both a matrix and a backup matrix of ***Tile*** objects; these represent, as anticipated, both the *Collectible* in it (Dot, Energizer, Empty) and the connections: *Path*. About the *Path*, it is represented as an enumeration compound of:

Vertical corridor	Horizontal corridor	Corner Up-Left
Corner Down-Left	Corner Down-Right	Corner Up-Right
Ghost Door	Cross	No Way
Vertical + Right	Vertical + Left	Horizontal + Down
Horizontal + Up	Horizontal + Up Exception	

V.1.1 The problem of discrete movement

The way it is described, every agent is in a tile of the board and moves from a tile to another adjacent to it; however, the speed can have different values according to the agent and other factors such as states, dots and phases. Considering the movement between tiles is discrete, a way to make it “dense” is required; moreover, this would also be necessary to graphically show the movement as continuous and not as jumpy from a tile to another.

The adopted solution was to consider each tile as a square of a given size, as the length from side to side; with this implementation any movement would be internal to a tile, and only after it exceeds its boundaries would move from tile to tile.

Nevertheless, the size of the tiles (expressed in *units*) and the speed (now considered as *units per second*) have to be opportunely tuned to avoid discretising problems. Because, although this method creates an internal dimension of the tiles, this is still discrete and undergoes the same issues. A logic solution would be to consider the speed as *percentage of tile per second* and treat the position of an agent as the percentage of the tile already walked, however this would have been much more complicate to treat and tune and was therefore discarded as solution.

The initial values of *TILE_SIZE* and *SPEED* (remembering that the speed is a shared value for all the agents that will have different percentage of it as personal speed) were set respectively to 22 and 200 but were successively changed due to a major issue caused by the combination of the two (and of the set FPS value: 20). As we know, the designers decided to set Pacman's speed to 0.8 and ghosts' to 0.75; however, when walking on a dot, Pacman's speed would slow to 0.71, thus becoming slower; this is an important detail because otherwise, no ghost would be a treat from behind.

Given that every cycle of the game loop is delayed being in accord with the desired FPS rate of 20 Frames per second (0,05 seconds), let us run some calculus to see the effects of these values:

Agent	**Personal Speed**	**Units per frame (discrete)**
Pacman on an empty tile	0.8	8 units
Ghost (normally)	0.75	7,5 units → 7 units
Pacman on a dot	0.71	7,1 units → 7 units

Table 1 SPEED=200; TILE_SIZE=20

As emerges from the table above, although Pacman should be slower than the ghosts when collecting dots, he appears to run at the same speed; therefore, they are not able to reach him from behind, but he can gain distance when not on a dot.

The values have been adjusted to 66 (TILE_SIZE) and 660 (SPEED), almost maintaining the ratio of the previous configuration but solving the problem:

Agent	Personal Speed	Units per frame (discrete)
Pacman on an empty tile	0.8	26,4 units → 26 units
Ghost (normally)	0.75	24,75 units → 24 units
Pacman on a dot	0.71	23,43 units → 23 units

Table 2 SPEED=660; TILE_SIZE=66

Enlarging the internal size of a tile also exploited another hidden issue: the way it was implemented, an agent could take (when possible) a turn according to the open directions in its tile; however, this caused taking 90° turns a cheat to run way faster than designed. It was in fact possible to take a 90° turn on whenever entering a tile, without needing to reach the middle of it, thus allowing players (and ghosts) to move many tiles in a few frames; a skilled exploiting player could easily take advantage of this to outrun ghosts even when Pacman should have been slower than them.

V.2 The Agents' classes

Agents, and agents related, classes are a major portion of the code and are responsible for every decision related to the 5 agents of the game: Pacman and the 4 ghosts.

These characters have some elements in common (in the **Agent** class), the most important of which is an *update* method that is called on every frame by the *Logic* class; this method internally calls two subroutines: *updateDirection* and *updatePosition*; but these, although very similar, are implemented separately for ghosts and Pacman.

The **Ghost** class, intuitively representing a ghost character, decides the following direction to take according to the **GhostPersonalState** (Scatter, Chase, Frightened, Eaten, House) and, in case of one of the first two, based on two objects of the class **TileGetter** contained in the Setup of the game.

Tile-getters exist variated for Scatter and Chase states and represents the different Behaviours proposed per each ghost; these can be narrowed down to a method called *get[Escape/Scatter]Direction*.

The **Pacman** class is abstract and only implements the *updatePosition* method, while still laying the responsibility for the direction choosing on its two subclasses: **PacBot** and **Player_Pacman**. This last is extremely simple and only interprets the arrow keys translating them in desired direction, converted into effective directions when possible.

PacBots, on the other hand, are more complex and look more like Ghosts: they too have two getters, *Wander* and *Escape*, that are invoked according to an alert parameter (as specified in *IV.3.1*), but in their case these getters return the following direction to take.

On updating, the *Logic* class invokes the update method of each agent and subsequently checks for ending or life-lost conditions through a key subroutine called *consequences*.

V.3 The Learning classes

The Learning module is compound of two pairs, distinctively one for each learning type: user and behaviours; each pair is made of a state observation class and a learning class.

The state observation is the abstraction discussed in *IV.2.2* and contains methods to log such observations and value fields that permit to update, read and reset statistical values such as minimum, maximum, average and last entry. Moreover, these also have a method to retrieve not the value but the index of the interval in which the value belongs.

Such indexes are necessary both for Adaptation and Learning, because they are both based on n-dimensional vector and ach state observation (with these indexes) represents a set of coordinates in this vector. At the end of the learning procedure, in each coordinate will be stored a score.

The learning class, responsible for such scores, proceeds to fulfil the Reinforcement Learning algorithm explained in *IV.2.1*; Results are then showed in a regular fashion that can be easily reshaped.

V.4 The DDA classes

The DDA module is the final step of this system; it has a core class, the ***DDA_Utils***, that communicates with the Game_Main, and manages its own module. This is summoned before the game starts, to load from files the user-score and behaviours-score values, and, subsequently, is periodically invoked to commit an adaptation process; where adaptation processes are a sequence of *n* (specified by an hyperparameter) 1-flip adaptations; this is fulfilled by the main method of the class: *adapt* and by its *n* inner calls to *_adapt*.

This class is responsible for creating the abstraction of the families and of the respective behaviours. These will be represented with a vector of objects of the class **Family**, each of which holds numerical fields used in the *_adapt* method to compute softmax numerators and probabilities as specified in *IV.3.5* and a collection of objects of the **Behaviour** class. Internally, these last too internally keep such values, but also have either a n-dimensional vector representing the behaviour score evaluation in each state (**Behaviour_tabular**) or a single value (**Behaviour_singleValue**) as anticipated in *IV.3.5.1*. Just for readability, each behaviour also contains a String value for their name.

The constructor of a *Family* object also requires an **Adapter** object, which is essentially a concretization of the Adapter functional interface, whose only method provides a change in the setup of the current game.

In the end, the DDA_Utils also invokes a **Policy** object: an interface that, with two Boolean functions decides whether a behaviour, or a family, is to be considered among the available for adaptation according to what said in *IV.4.1*.

Chapter VI

Testing and results evaluation

To conclude the research, it was decided to test, observe and evaluate the game behaviour with different players. Although a full-scale testing plan, with an online playable version and questionnaire, would have been optimal, it was decided to settle with a local analysis over a few dozens of players with various, but mostly non null, experience with games and keyboard controls in general.

The test-bed game was ideal as first approach to the DDA analysis and implementation but proved too short and sudden, as dynamics, to allow a smooth and precise adaptation. It would have been more fit if the player could play many levels until the loss of all the lives (as in the classic arcade version); in that case, adaptation in subsequent levels could use the scores read from the previous ones, allowing a slower and still effective adaptation.

The adaptation system implemented, being closer to the Dynamic Scripting than to any other known method in literature, gives its full potential in long observable environments and on multiple encounters, which is not the case for Pac-Man. Nevertheless, the results were interesting and the adaptation observable even in short games.

Mostly, it is noticeable how the game interacts differently against an expert player than it does against a novice; and a few peculiarities, important for the design aspect of the topic, are analysable, such as:

- The game does not grant a free win: As one of the most criticized fear of the players towards the Dynamic Difficulty Adjustment, we managed to keep Pac-Man a challenge that even when it adapts to a struggling player does not annihilates the challenge.
- On the same extent, the game is not impossible at its hardest configuration, nor is this configuration sudden: although less feared, it would violate many

concepts of challenge in game design as well as an easy win; the game still has to be playable.

- The challenge changes, but the experience does not: this was probably the most carefully approached objective of the thesis; the challenge gets harder but the NPCs do not change the experience they generate. Moreover, players who know the classic version by heart will recognize the differences and will experience a new challenge in the same walls they know of.

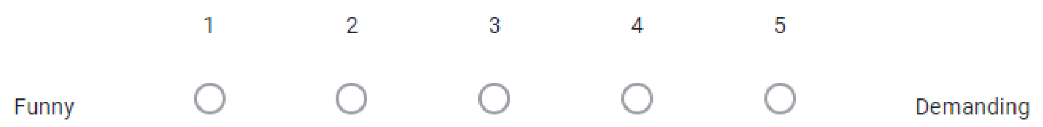
Unfortunately, there are also two observed downsides:

- The adaptation is noticeable. It could be set to a smoother adaptation still and better hidden, but the short nature of the matches requires hard changes to be effective. Nevertheless, should this approach be export in another, more fit, environment this could be fixed.
- The adaptations stack. This would not be too much of a problem if it was not for the danger that, in a competitive scenario, some players could use this peculiarity as a strategy: they could initially behave as inexpert, collecting few dots and maybe losing an early life, thus willingly lowering the challenge and then hasting to completion not giving the game the time to re-adapt, therefore exploiting this feature as a weakness.

VI.1 The testing plan

Despite all this, the test set up aimed to verify if the game was still appreciable, if it was correctly difficult and if it was not too much erratic; as well as with which frequency, the adaptation was perceived.

Given the inexperience in this area, the testing plan was widely inspired by the Hagelback and Johansson's paper[B-8] on player experience in presence of runtime dynamic difficulty scaling. After playing a certain number of games, players were asked to answer a setoff simple question where they had to place over an axis from an adjective to another where did the specified game place itself



	1	2	3	4	5	
Funny	○	○	○	○	○	Demanding

The adjectives are divided in three sets internally divided in positives and negatives, 6 clusters in total, each containing 4 adjectives. Each cluster is paired with every other one except the opposite, therefore every adjective is used only once.

The clusters were meant to evaluate the aforementioned experience characteristic: *Appreciability*, *Difficulty* and *Variability*. For the last two clusters, two adjectives were chosen to sound like good qualities and the two others to seem unwanted qualities; these two inner categories are always matched with the same category in order to never let the user have to decide between something that sounds good and something that sounds bad, thus hopefully concealing any desire of the results. Of course, this is inapplicable for the Appreciability cluster (being the whole concept natively good), therefore in those couples containing an Appreciability adjective the good/bad meaning was chosen based on the cluster of the Appreciability itself (positive or negative).

Appreciability + abbr. **A$^+$**			
Funny	*Pleasant*	*Entertaining*	*Nice*

Appreciability - abbr. **A$^-$**			
Boring	*Tedious*	*Unnerving*	*Annoying*

Difficulty + abbr. **D$^+$**			
Good way		*Bad way*	
Challenging	*Demanding*	*Stressful*	*Tiresome*

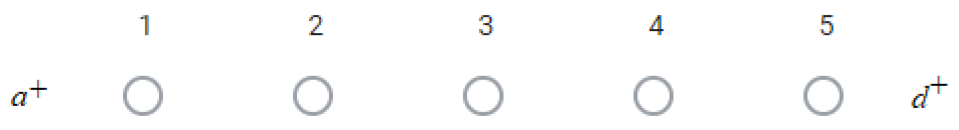
Difficulty - abbr. **D$^-$**			
Good way		*Bad way*	
Easy	*Simple*	*Trivial*	*Effortless*

Variability + abbr. **V**$^{+}$			
Good way		*Bad way*	
Unpredictable	*Variegated*	*Irregular*	*Erratic*

Variability - abbr. **V**$^{-}$			
Good way		*Bad way*	
Regular	*Coherent*	*Monotonous*	*Foregone*

Each answer in the questionnaire, being a point on the axis, is easily converted to a score based on how much such adjective is prevalent; this score is summed for each adjective of the same cluster and, finally, the cumulative score of each cluster is diminished by the score of its opposite, thus generating a score for the 3 original aspects we were interested in.

In example, given the pair (a^{+}, d^{+}) with $a^{+} \in A^{+}$ and $d^{+} \in D^{+}$ any placement on the axis would have the following meaning:



1: $a^{+} \gg d^{+}$

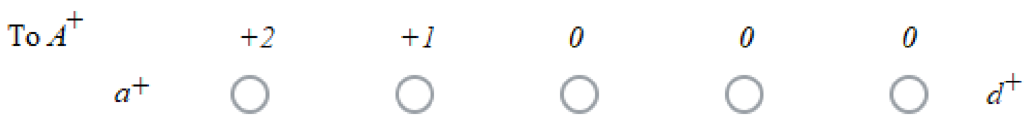
2: $a^{+} > d^{+}$

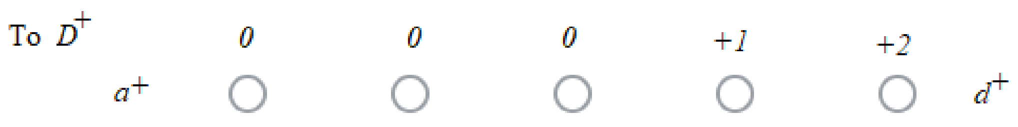
3: $a^{+} = d^{+}$

4: $a^{+} < d^{+}$

5: $a^{+} \ll d^{+}$

And would contribute to the sum respectively to A^{+} and D^{+} as follows:





The subjects will play three games, one for each version among **not adaptive**, **adaptive T** and **adaptive SBS** (see *IV.3.5.1 The stateless behaviour's score variant*); and will have to answer 12 questions: 12 pair comparisons as explained above.

The decided adjective pairs are the following:

Funny – Challenging	$A^+ – D^+$
Erratic – Boring	$V^+ – A^-$
Effortless – Monotonous	$D^- – V^-$
Nice - Unpredictable	$A^+ – V^+$
Tiresome – Tedious	$D^+ – A^-$
Simple – Variegated	$D^- – V^+$
Coherent – Entertaining	$V^- – A^+$
Annoying – Trivial	$A^- – D^-$
Demanding – Regular	$D^+ – V^-$
Pleasant – Easy	$A^+ – D^-$
Foregone – Unnerving	$V^- – A^-$
Stressful – Irregular	$D^+ – V^+$

To better analyse these results, before asking about the three games the candidates were asked 3 questions concerning respectively: age, experience with videogames and experience with Pac-Man. These were used to group the results and to study eventual correlations.

Although none of the candidates knew which version of the game they were playing (or in which order), some (60%) knew the topic of the dissertation and of the implementation. For this reason, the questionnaire results were evaluated considering this hypothetical bias too.

At the end of the three games (and of the respective section in the questionnaire), the candidate will be asked whether he/she noticed or not Adaptive difficulty in any of the game (with a brief explanation of what it is). Originally, this question was asked after the 12 pairs for each game, but was subsequently removed to avoid that the user, after being asked in the first game, would focus on noticing an eventual adaptation more than he/she would have otherwise.

VI.2 The evaluation

The evaluation of the questionnaire results aimed to quantify averagely the appreciation, perceived difficulty and observed variation among the three different variants of the game (again: **not adaptive**, **adaptive T** and **adaptive SBS**). However, these values were expected to vary consistently from player to player, providing dizzy and only superficial interpretations. Therefore, these scores were also computed grouping the answers by age, experience with games and experience with Pac-Man in particular. It was also considered to calculate a ratio of perceiving the adaptive difficulty according to such categories and according to the knowledge of the topic of the research itself.

After compiling an evaluation scheme, as summarized above, the issue of lack of entries occurred. Due to restrictions the number of questionnaire answers collected was smaller than expected, and therefore many values were to approximative to be considered reliable.

For this reason, variants of evaluation plan were created; in particular, three modifications were designed that would allow a fairer evaluation even on few data, and that could be stacked together, thus providing 8 evaluation plan variants (as the number of combinations obtained with 3 binary modifiers).

The base of all these plans is the **Original plan**; this divides the user personal categories as follows:

Age				
Less than 20	From 20 to 29	From 30 to 39	From 40 to 49	50 or more

Experience with games			
None	Poor	Average	High

Experience with Pac-Man			
None	Poor	Average	High

Starting from this, the three modifiers can be applied either one at a time or in groups, thus forming the other seven variants. The specified modifiers are:

- ***Binary experience (BE)***: that unites Average-High and None-Poor under the same groups, thus making the experience bias less dense but also needy of a many samples.

Age		
25 years or less	From 26 to 40	More than 40 years

- ***25-40 age thresholds (AT*** as **A**ge **T**hresholds***)***: that divides the age intervals differently. This specific partition was designed to divide the first half of the *20-29 years* interval from the second because it was observed a major difference between the two. This partition also grouped together less populated intervals.

Experience with games	
None or Poor	Average or High

Experience with Pac-Man	
None or Poor	Average or High

- ***Mutations of borderliners (MoB)***: this last, hypothesize to insert mutations of existing records to enrich the population. These mutations are chosen among those that are borderline between two categories (in example: a player that is 29 years old and is soon to become 30, or a player that is between average and high expertise). This technique, of course, unbalances the global data, giving more weight to some of the evaluation (occurring more than once).

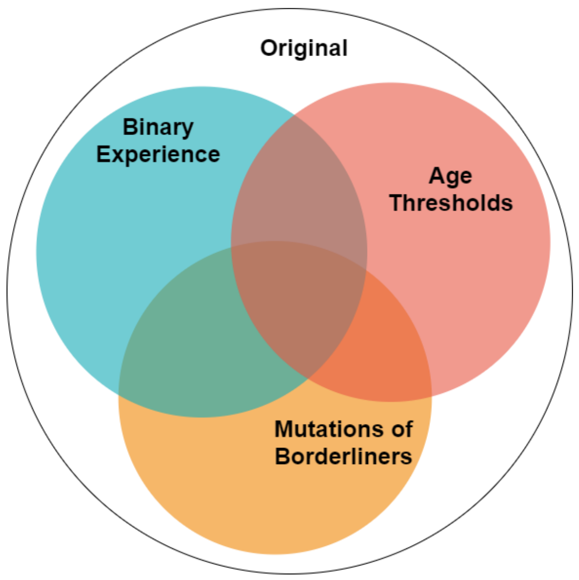


Figure 26 Evaluation possibilities

This last, Mutation of Borderliners, was discarded after a reasonable amount of evaluation was collected.

While observing the results, it soon became clear that a considerable amount of the candidates (45%) classified themselves as *Average* or *High* on videogame experience and *Poor* on PacMan experience. Among them, of course, many were actually extremely capable, perhaps because their expertise with games in general overcame their lack of experience with PacMan; while the others were visibly less able to play. These two inner groups showed completely opposite results while belonging to the same macro groups during the evaluation, and thus creating dizzy results.

It was also observed that many good, or even very good, players were humble about themselves (probably) and therefore were modest about their expertise in the questionnaire; while, on the other hand, many less experienced players signed themselves above their effective ability. This attitude too of course, caused many contradictions; because it was expected that the game would behave almost diametrically differently with experienced and not experiences players.

The solution to this problem was to categorize the groups of players according to something impartial and uncontrollable by the candidates: whether they won or not in the Standard (non-adaptive) game. This was intuitively the best approach because it was the only game identical (in setup) for everyone, and could give an indicative information about the skills of the player: were they enough to win the Standard level or were the not? Given that the win-rate of the Standard game was around 50% this seemed fair enough.

Following this idea, every statistic has been computed dividing the candidates set among winners and losers of the standard game; we will refer to this evidence as: ***WoS*** (**W**inners **o**n **S**tandard) and ***LoS*** (**L**osers **o**n **S**tandard).

VI.3 The results

Before talking about the achieved results, let us introduce what was most desirable and what was expected. Remembering that the questionnaire gave values of *Appreciation*, *Difficulty*, *Variability* and *Perception of adaptation* for each game, let us imagine how these parameters should be in the best (most successful) and expected results.

Regarding the most optimistic outcome, it would be a great success if the ***Appreciation*** was observed to be <u>higher in the adaptive versions rather than in the Standard one</u>, meaning that users averagely appreciated more a dynamic setting of challenge than a static one; this goes regardless of age, expertise and knowledge of the research topic.

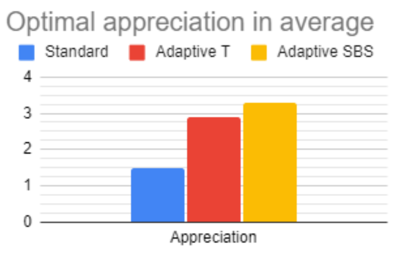


Figure 27 Optimal appreciation in average among all the candidates

About the perceived ***Difficulty***, the matter is a little bit more complex; it would be perfect if less experienced players, felt the standard version harder than the adaptive, meaning that the game successfully simplified the game for them. On the other hand, more skilled players should have their adaptive games harder than the standard that should be beneath their abilities.

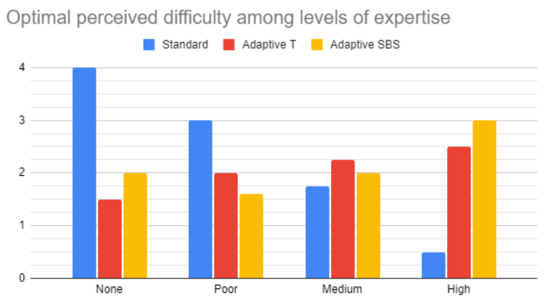


Figure 28 Optimal difficulty based on the declared gaming expertise

Concerning the ***Variability*** parameter, it is not vital but it could be better if the adaptive games did not prove too much more erratic than the standard one; failing in this aspect would not have been a failure for the research itself, but would show that the adaptation is not smooth as the Game Design would want it to be.

Finally, the ***Perception of Adaptation*** would be perfect if generally low, especially for those who were not aware of the topic.

Despite these being the best scenarios for the collected parameters, there are certain design and development choices made that could alter these results; therefore, before analysing the collected results, it could be useful to try and predict what we expect to see.

Regarding the expected values of ***Appreciation***, it was expected to depend on the outcome of the standard game:

- Players who lost the Standard game: we expect them to appreciate more the adaptive ones, especially those who considered themselves more experienced with games (or with PacMan) who could feel frustrated of losing the standard game.
- Low expertise players who won the Standard game: we expect them to appreciate similarly all the games, because the adaptation should either simplify the game (a little) or slightly complicate it; the players should therefore still be able to win without facing a challenge too hard for them.

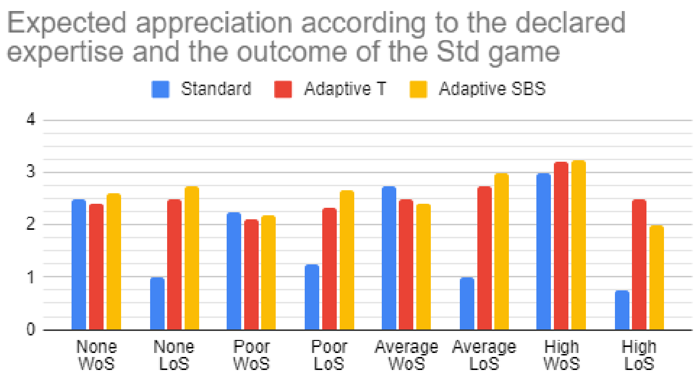


Figure 29 Expected appreciation according to the declared expertise and the outcome of the Standard game

- More experienced players who won the standard game: we expect them to either prefer significantly more the Standard version or the adaptive one; this because the adaptive games should be noticeably harder and could therefore become <u>stimulating</u> or <u>frustrating</u>.

About the perceived ***Difficulty***, we expect a behaviour similar to the best and the expected appreciation; we imagine players who lost the standard version to feel the adaptive games easier and vice versa. Upon introducing the win-on-standard division, we expect the difficulty to be lowered for the *LoS* (from Standard to Adaptive) and higher for the *WoS*, regardless of the declared expertise if not for some occurrence.

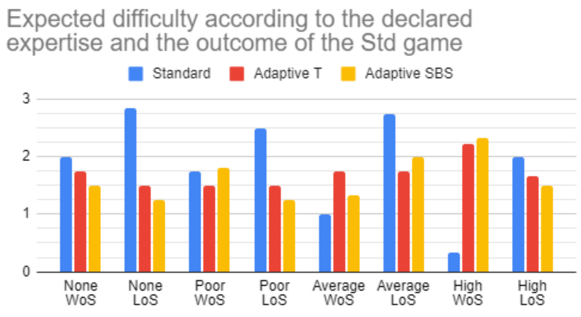


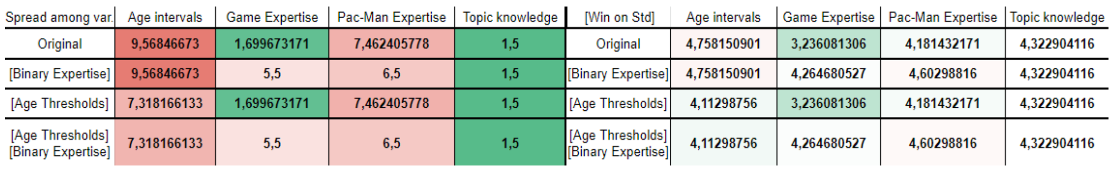
Figure 30 Expected difficulty according to the declared expertise and the outcome of the Standard game

Concerning the ***Variability*** parameter, it is only expected to be higher for the Adaptive T game due to its less-controlled nature.

Finally, the ***Perception of Adaptation*** is expected to be more present for those who knew the topic of the research, and more precise for more experienced players.

With that being said, let us examine what has been collected so far.

To decide which inner partition of the candidates will be considered (among *Binary Expertise* and *Age Threshold* variants), it was considered how spread the occurrences were internally. Given that the questionnaires gathered were not satisfyingly numerous, it was hypothesized that the application of both the modifiers could help making each subgroup more numerous and, therefore, its result more precise. However, it was observed that the spread of the variants created with the *Binary Expertise* modifier was higher than the ones created without it. The situation observed was as follows:

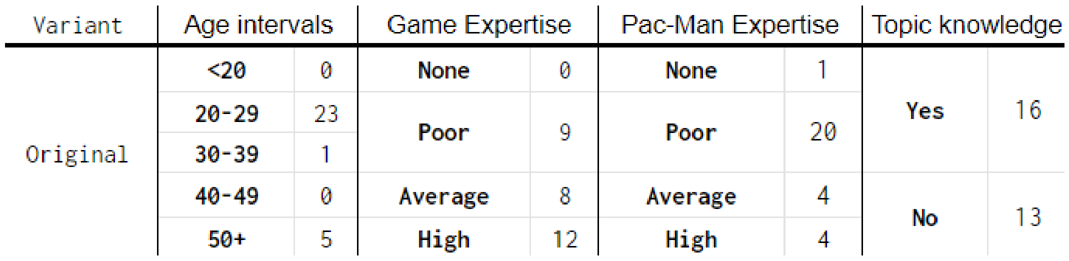


Spread among var.	Age intervals	Game Expertise	Pac-Man Expertise	Topic knowledge	[Win on Std]	Age intervals	Game Expertise	Pac-Man Expertise	Topic knowledge
Original	9,56846673	1,699673171	7,462405778	1,5	Original	4,758150901	3,236081306	4,181432171	4,322904116
[Binary Expertise]	9,56846673	5,5	6,5	1,5	[Binary Expertise]	4,758150901	4,264680527	4,60298816	4,322904116
[Age Thresholds]	7,318166133	1,699673171	7,462405778	1,5	[Age Thresholds]	4,11298756	3,236081306	4,181432171	4,322904116
[Age Thresholds] [Binary Expertise]	7,318166133	5,5	6,5	1,5	[Age Thresholds] [Binary Expertise]	4,11298756	4,264680527	4,60298816	4,322904116

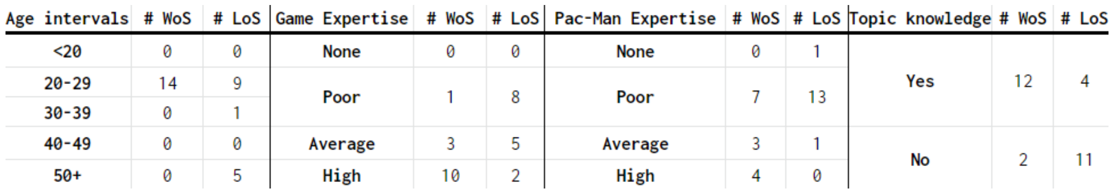
The closer a tile gets to red, the higher its inner spread is; meaning that the less valuable the observation will be. As observed in the image, the most valuable solution would be applying the *Age Threshold* modifier only; nevertheless, the age division is the less interesting of all, therefore, despite the proposed modifiers, it was proposed to use the original questionnaire evaluation in the analysis.

VI.3.1 The observed results

At the end of the evaluation, only 30 questionnaires were collected; the most of which were of candidates between 20 and 29 years old. Internally, they were divided as follows:



Variant	Age intervals		Game Expertise		Pac-Man Expertise		Topic knowledge	
Original	<20	0	None	0	None	1	Yes	16
	20-29	23	Poor	9	Poor	20		
	30-39	1						
	40-49	0	Average	8	Average	4	No	13
	50+	5	High	12	High	4		



Age intervals	# WoS	# LoS	Game Expertise	# WoS	# LoS	Pac-Man Expertise	# WoS	# LoS	Topic knowledge	# WoS	# LoS
<20	0	0	None	0	0	None	0	1	Yes	12	4
20-29	14	9	Poor	1	8	Poor	7	13			
30-39	0	1									
40-49	0	0	Average	3	5	Average	3	1	No	2	11
50+	0	5	High	10	2	High	4	0			

Given that no observation was collected for '*Game Expertise: None*' and that the age range has been considered irrelevant we can conclude that the '*Game Expertise*' subdivision is equally distributed enough; while, on the other hand, the '*Pac-Man Expertise*' was presented a clear majority of *Poor* candidates compared to the few of all the other categories.

The result evaluation will be schematised following the main observable outcomes: *Appreciation, Difficulty, Variation, Win rate, Perception*.

For starters, let us consider these values in absolute (in average of all candidates), to have a first idea of how they are presented.

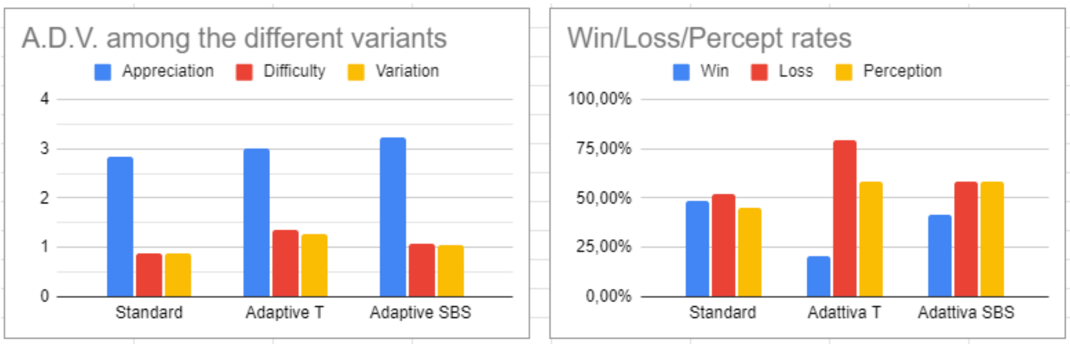


Figure 31 Average statistics. Appreciation, difficulty and variability (left) and win, loss and perception rate (right)

As it is observable, the **appreciation** is averagely similar among the three versions of the game, although slightly higher in the adaptive ones; in all three cases the value is medium-high, which is a gratifying outcome.

The perceived **difficulty** on the other hand is averagely low, but higher in the adaptive versions; this was not expected because the win rate is not generally high, and it rather suggests that the system may be over-punitive; on the other hand, it also implies that the game does not become excessively easy and does therefore not grant a free win, which is reassuring.

The **variation** of the games, which means whether or not the game seemed prone to change many times during the same game, was also averagely low, being as expected minimally higher in adaptive versions (given that they do in fact change). In any case, it shows not high values which could indicate that the game applies too sudden and dizzy correction.

Above any wishful expectation, the *Standard* game shows an almost 50% **win rate**; sadly, however, the *Adaptive T* version drops to an alarming 20%, while the *Adaptive SBS* only to 42%. There was no expectation and no optimal value for the win rate, but still such a severe drop, as observed for the *Adaptive T* version, was undesirable; studying the issue, it turned out that this version happened to harden the game even in situation that did not require it (mistaking), although it recovered in the following adaptation it was often too late because the player had already lost one or more life.

Regarding the **perception** of adaptation, as expected but not as wished, it was observed a higher perception of adaptivity in the adaptive versions rather than the standard one. As said, this was expected because the implementation made forced adaptation many times during the short game for research purposes; nevertheless, it seemed to not disturb the game experience. Moreover, the standard (not adaptive) version itself was perceived as adaptive in 45% of the games, against the only slightly higher 58% of the adaptives.

For each of these parameters, it is interesting to see how they were obtained internally according to the outcome of the Standard game and, eventually, other divisions such as expertise with games in general or with PacMan specifically.

As the following image shows, in average the values studied behave as expected: the **appreciation** is even slightly higher in the standard game than in the adaptives for those players who won the standard game; while it is more significantly higher for the adaptive ones than the standard for the others. The **difficulty**, on the other hand, behaves in the best possible scenario: showing that the games becomes harder for WoS players and easier for the LoS ones. Finally, the **variation** values show that LoS players are much less aware of this feature than the, presumably more experienced WoS ones; nevertheless, the values for the adaptive games is always, even if slightly, higher than in the standard game, but low nonetheless.

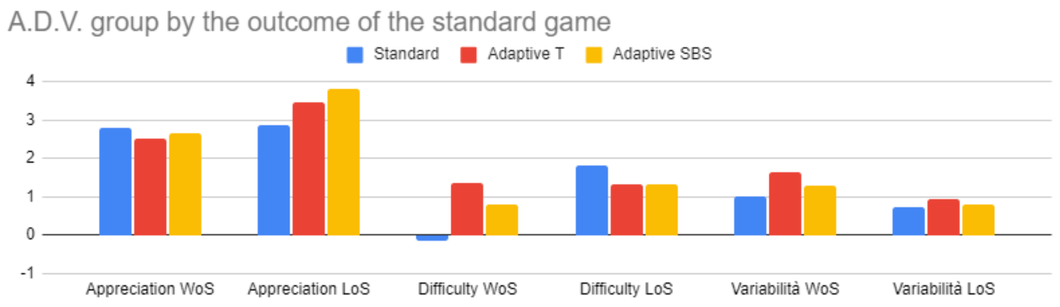


Figure 32 Appreciation (blue), Difficulty (Red) and Variation (Yellow) perceived by candidates, based on the outcome of their Standard game

Delving more in details, it is noticeable that there is only one entry of significantly higher **appreciation** for the Standard game rather than the adaptives: players with average PacMan experience that won the Standard game. Apart from it, we can observe that WoS players do not dislike the adaptive games and even like it more sometimes, while LoS players show a plain preference for adaptivity.

Regarding the **difficulty** perceived, it appears to behave as wished: becoming more complex for players who won the Standard game and easier for those who lost it. The only exception to this is for the players with poor game experience, for which the adaptation looks less precise, despite the high values of appreciation portrayed by the same.

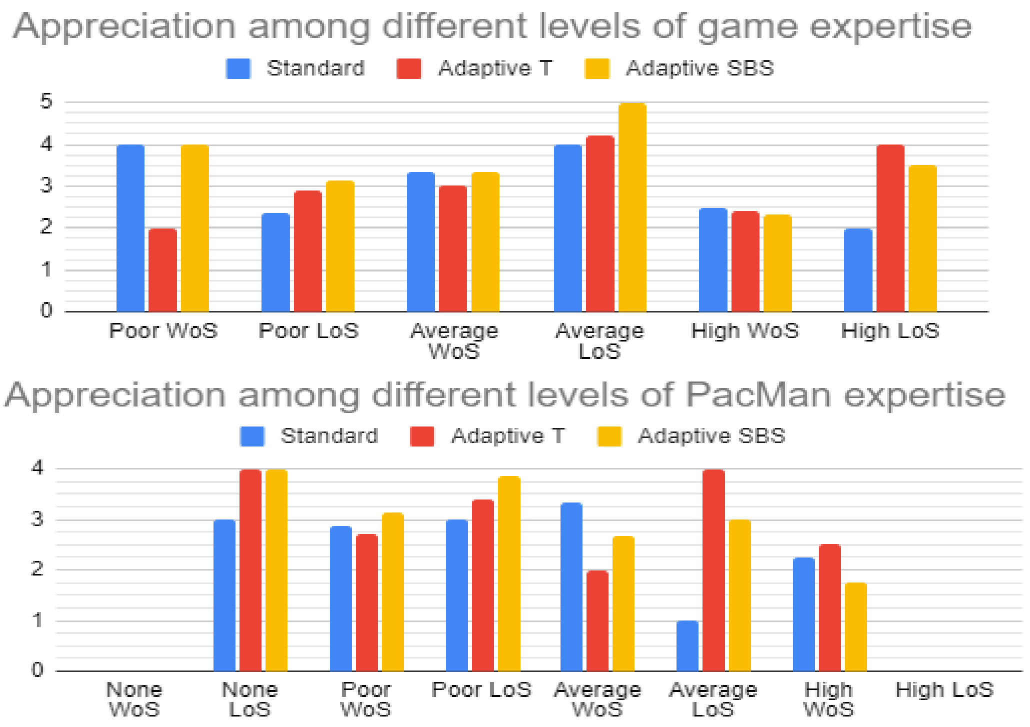


Figure 33 Appreciation perceived according to the outcome of the Standard game. Group by levels of game expertise (Up) and Pac-Man experience (Down)

Another less precise example, is for the players with high Pac-Man experience (all WoS because they it was never observed a Loss in Standard game for them), for which the difficulty looks averagely the same. We have no explanation for this, especially because from the game logs it resulted heavy hardenings; we only hypothesize that this fuzzy behaviour is fault of the poor variety of the sample: in fact, it was hard to find a player with high expertise on a game that is 40 years old, and was therefore asked the only one found to play multiple questionnaires in different days.

It is also observable that difference of difficulty registered between the modalities are given by players that signed themselves as more experienced but lost on the standard game, which proves that using an impartial parameter such as the victory or loss on the Standard game was indeed justified.

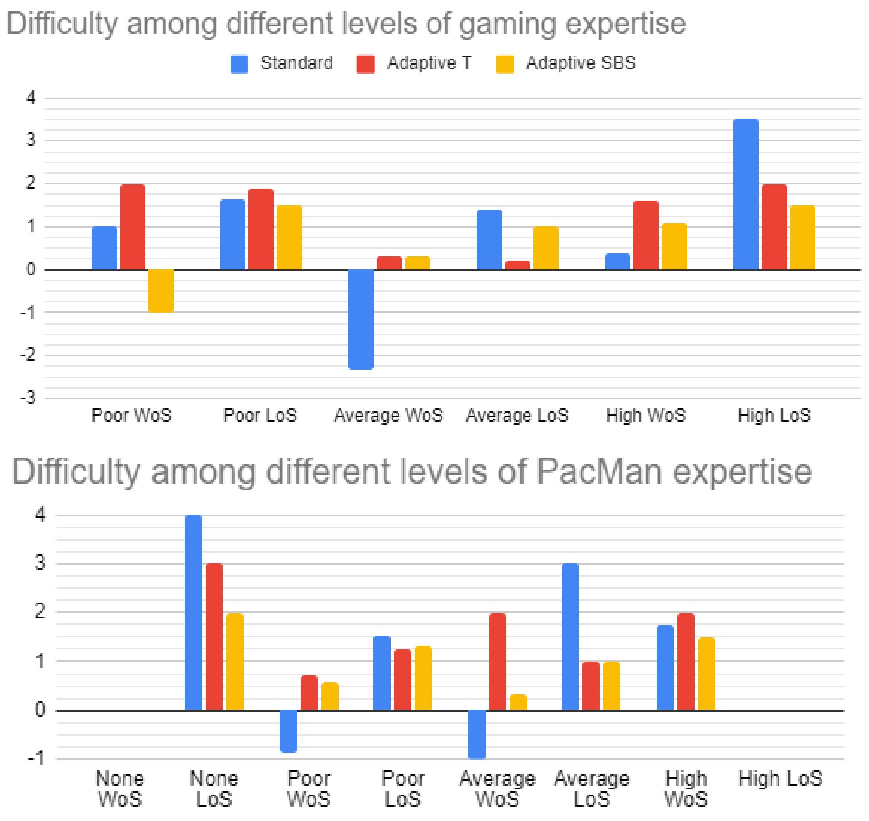
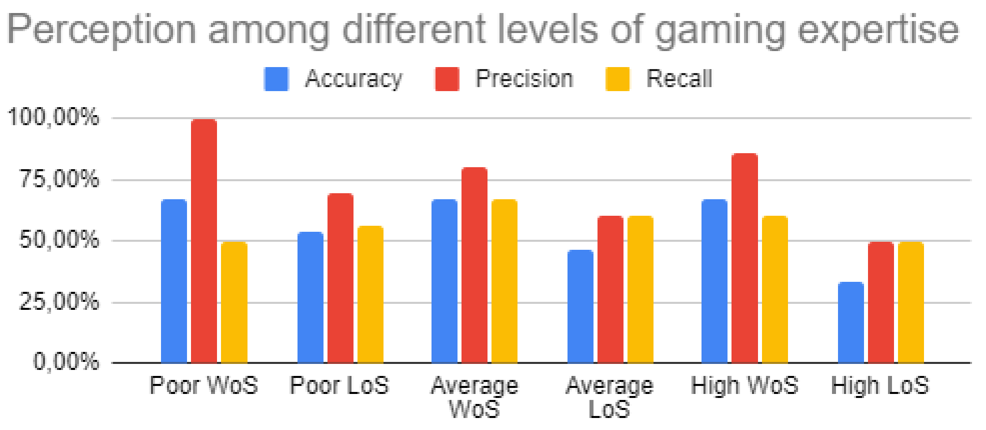


Figure 34 Difficulty perceived according to the outcome of the Standard game. Group by levels of game expertise (Up) and Pac-Man experience (Down)

Finally, concerning the perception of adaptation of the games, we can observe how more aware of the adaptation WoS players are; on the other hand does not seem to be any correlation between (declared) game expertise and correct spotting of adaptation, it actually decreases instead for LoS players as the expertise goes up. A little more sense make the PacMan experience parameter, for which we see a correlation with precision and accuracy.



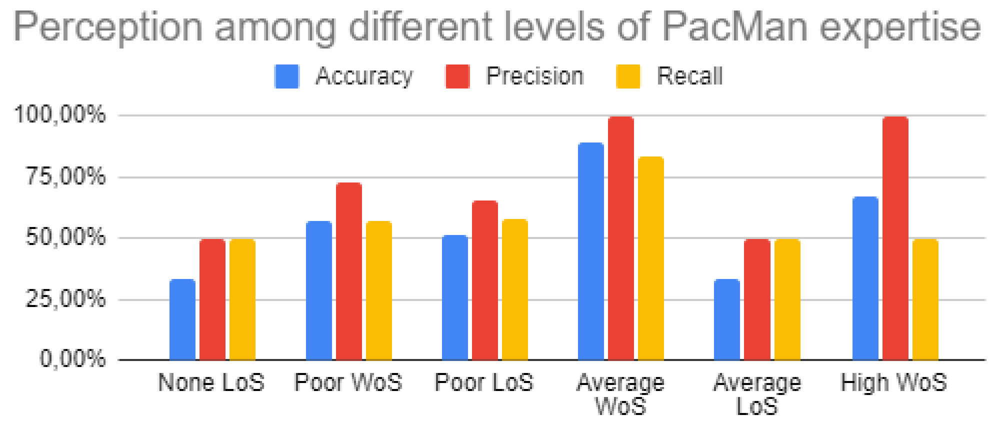
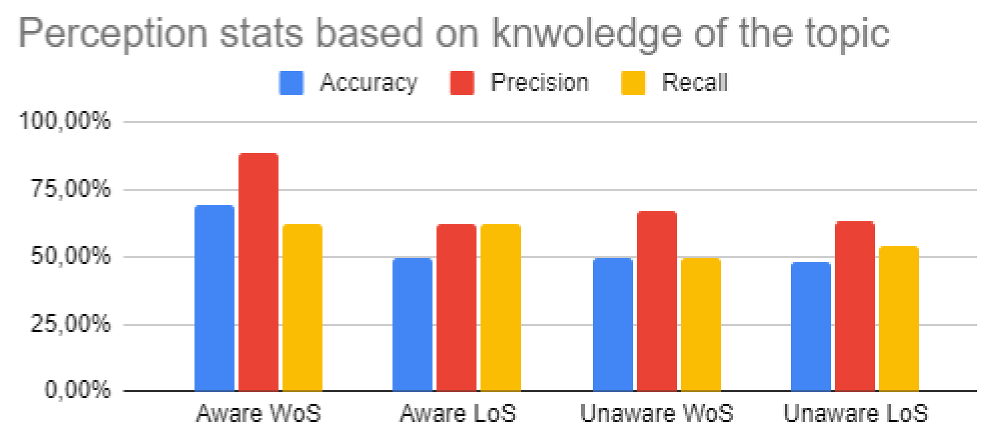


Figure 35Awareness of perception statistics according to the outcome of the Standard game. Group by levels of game expertise (Up) and Pac-Man experience (Down) . Accuracy (Blue), Precision (Red) and Recall (Yellow)

What really makes a great difference was the knowledge or not of the research topic: candidates unaware of the research topic affirmed that they perceived adaptation in all the game variants with the same frequency, even slightly higher for the Standard one. While on the other hand players aware of the topic behaved diametrically the opposite based on the outcome of the Standard game. WoS misplaced the Standard game as adaptive only 16% of the time, while noticing that the game was adaptive much more often in the *Adaptive T* version (75%) and *Adaptive SBS* (50%); LoS instead appeared to be eager to say that they noticed the adaptation, in this way they never misjudged an *Adaptive SBS* game (100%) but perceived the Standard as adaptive as well 75% of the times.



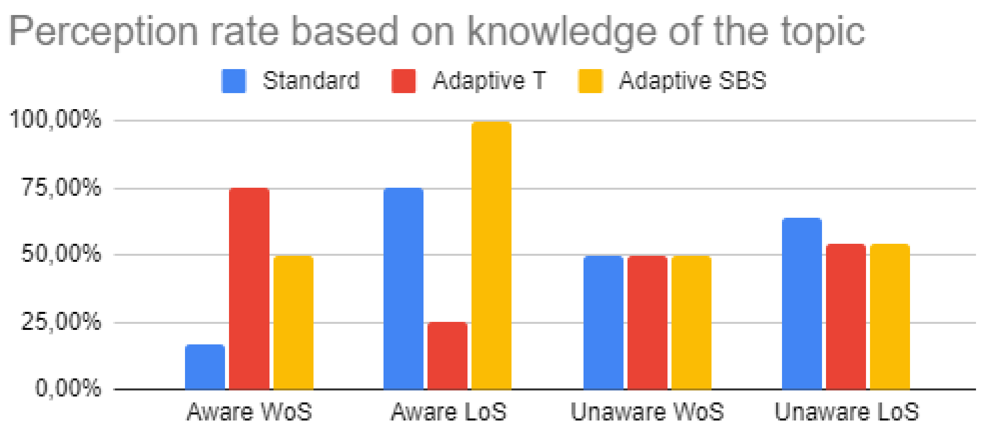


Figure 36 Perception of Adaptation statistics (Up) and rate (Down)

VI.4 Conclusions

Given the small sample, it is hard to draw generalizable conclusions on both the efficacy and the applicability of the system and the Adaptive difficulty in general. However, it is remarkable that some results are already observable even with such small numbers. In particular, the appreciation and difficulty values are extremely satisfying considering the essentiality of the game itself.

What could be concluded is that the **Dynamic Difficulty Adjustment** can greatly improve the game experience, even in short periods; but should be used for much longer games and in longer periods of observation. The 15 seconds period set in our experiment is in fact a weakness for the entire system, making it both more erratic, more spottable and less accurate.

About the adaptation process in per se, it is actually completely independent from the game it is implemented on and is unaware of everything beyond the setup, families and behaviours, which are very generalizable concepts. The process was widely inspired by the *Dynamic Scripting* methodology which has however been seen in use only in fighting games or strategic RPGs; but this research can show that its idea is appliable to other games and genres.

The implementation provided all the families considered in any adaptation, but this is easily modifiable for more complex games; and would actually be a great idea to do so because it would open the door to much more customisable systems.

Regarding the two adaptive variants, ***T*** and ***SBS***: the former did not show the problems expected so heavily; they both behaved almost the same, one behaving better in some scenarios and the other in others. Further works could exploit more about the *Adaptive T* weaknesses, but with the results gathered it is impossible to declare one better than the other.

The *Adaptive T* games adapted faster, and this could either be good or bad according to the case; although it was not perceived as globally harder, it also portrayed an extremely lower win rate than the other versions. The *Adaptive SBS* version, on the other hand, adapted more smoothly, but often too slowly; moreover, its ‘controlled’ drove the adaptation towards the idea of the designer instead of the acquired knowledge; which is not often a good effect in many computer science application, but could be in a more artistic one such as the Game Design.

Bibliography

In apparition order:

[1] **The priesthood at play: computer games in the 1950s**; *Alexander Smith*. 2014.

[2] **Digital games and escapism**; *G. Calleja*. 2010.

[3] **RTS games as testbed for real-time Al research**; *M. Buro and T. Furtak*. 2003.

[4] **The illusion of intelligence. Al game programming wisdom**; *B. Scott*. 2002.

[5] **Dynamic balancing in ARPG games**; *R. Bednarski, M. Słonski*. 2018

Faust, *Johann Wolfgang von Goethe*. 1808.

[6] **A theory of fun for Game Design**; *Raph Koster*. 2014

[7] **The Grasshopper: Games, Life and Utopia**. *Bernard Suits*. 1978

[8] **Measuring player experience on runtime dynamic difficulty scaling in an RTS game**, *Johan Hagelback and Stefan J. Johansson*. 2009.

[9] **MDA: A Formal Approach to Game Design and Game Research**, Robin Hunicke, Marc LeBlanc, *Robert Zubek*. 2004

[10] **The art of Game Design**. *Jesse Schell*. 2008.

[11] **Flow: The psychology of optimal experience**. *M. Csíkszentmihályi*. 1990.

[12] **Flow in games**, *Jenova Chen*. 2006.

[13] **Perceived Sense of Challenge (PSC)**, *Shikhar Juyal*, yet to publish.

[14] **AI for Dynamic Difficulty Adjustment in Games**. *Robin Hunicke, Vernell Chapman*. 2004

[15] **Dynamic Difficulty Adjustment (DDA) in Computer Games: A Review**; *Mohammad Zohaib*. 2018.

[16] **Modelling Individual Differences in Game Behaviour using HMM**; *S. Bunian, A. Canossa, R. Colvin and M. Seif El-Nasr*. 2018

[17] **Designing Engaging Games Using Bayesian Optimization**; *M.M. Khajah, B.D. Roads, R.V. Lindsey , Y. Liu and M.C. Mozer*. 2016.

[18] **A Fuzzy Approach to Generating Adaptive Opponents in the Dead End Game**, *H. Hsieh, A. Ling and L. Wang*. 2008

[19] **Game and Player Feature Selection for Entertainment Capture**; *Georgios N. Yannakakis and John Hallam*. 2007

[20] **Modeling Player Experience in Super Mario Bros; Chris Pedersen**; *Julian Togelius and Georgios N. Yannakakis*. 2009

[21] **Towards Automatic Personalized Content Generation for Platform Games**; *Noor Shaker, Georgios Yannakakis and Julian Togelius*. 2010

[22] **Adaptive Game AI with Dynamic Scripting**; *P. Spronck, M. Ponsen, I. Sprinkhuizen-Kuyper, and E. Postma*. YEAR

[23] **Self-organizing networked systems for technical applications: a discussion on open issues**; *W. Elmenreich and H. de Meer*. 2008.

[24] **Dynamic Difficulty Adjustment in Games by Using an Interactive Self-Organizing Architecture**; *A. Ebrahimi, Mohammad-R. Akbarzadeh-T*; 2014.

[25] **To create DDA by the approach of ANN from UCT-created data**; *X. Li, S. He, Y. Dong*. 2010

[26] **MPRL: Multiple-Periodic Reinforcement Learning for Difficulty Adjustment in Rehabilitation Games**; *Yoones A. Sekhavat*. 2017

[27] **Artificial Intelligence: A Modern Approach 3rd ed**; *Russel and Norvig*. 1994.

[28] **Artificial Intelligence and Games**; *Georgios N. Yannakakis, Julian Togelius*. 2018

[29] **Why do Pinky and Inky have different behaviours when Pac-Man is facing up?**; *Don Hodges*. 2008.

Sitography

In apparition order:

[1] Edge (magazine), July 1998.

[2] theretroperspective.com. 2018. Consulted on March 18th. 2020

[3] Chris Sorrell, original designer of MediEvil. Interview for PlayStationBlog in 2012.

[4] Behind the classics (series), PlayStationBlog.

[5] all-things-andy-gavin.com, *Making Crash Bandicoot (blog)*, Jason Rubin and Andy Gavin. 2011.

Ludography

In alphabetical order:

[1] **Angry Birds (series)**, *Rovio*. puzzle causal strategy shooter originally for mobile devices. 2009.

[2] **Animal Crossing (series)**, *Nintendo*. Social simulation game for Nintendo's consoles. 2001 (first).

[3] **At dead of night**, *Baggy Cat*. Horror point and click. 2020.

[4] **Candy Crush**, *King*. Puzzle match-three game for Browser and mobile phones. 2012.

[5] **Clash of Clans**, *Supercell*. strategy game for mobile devices. 2012.

[6] **Clash Royale**, *Supercell*. RTS for mobile devices. 2016.

[7] **Crash Bandicoot 2: Cortex strikes back**, *Naughty Dog*. Platform for PlayStation. 1997.

[8] **Cut the rope (series)**, *ZeptoLab*. Puzzle game for mobile devices. 2010.

[9] **Cyberpunk 2077**, *CD Projekt Red*. Action RPG game for PS4, PS5, Xbox One and PC. 2020.

[10] **Descent: Journeys in the Dark**, *Fantasy Flight Games*. RPG boardgame. 2005 (2012 the second edition).

[11] **Final Fantasy VII Remake**, *Square Enix*. RPG for PlayStation 4 and PlayStation 5. 2020.

[12] **Final Fantasy VIII**, *Square (*now *Square Enix)*. RPG originally for PlayStation. 1999.

[13] **Final Fantasy XV**, *Square Enix*. RPG for PlayStation 4, Xbox One, PC, Google Stadia. 2016.

[14] **Five nights at Freddy's (series)**, *Scott Cawthon*. Survival horror originally for PC. 2014 (first).

[15] **flOw**, *Jenova Chen and Nicholas Clark*. Microorganism simulator. 2006.

[16] **God Hand**, *Capcom*. Action beat 'em up game for PS2. 2006.

[17] **Gradius**, *Konami*. Arcade horizontal-scrolling shooter. 1985.

[18] **Hollow Knight**, *Team Cherry*. Action adventure 2D. 2017.

[19] **Kid Icarus: Uprising**, *Project Sora*. Action shoot 'em up. 2012.

[20] **Kingdom Hearts (series)**, *Square soft (*then *Square Enix)*. Action RPG. 2002 (first).

[21] **Kula World**, *Game Design Sweden AB*. Puzzle game for PlayStation. 1998.

[22] **League of Legends**, *Riot games*. MMORPG for PC. 2009.

[23] **Lego**, *The LEGO group*. Construction toys. 1949.

[24] **Life is strange (series)**, *Dontnod Entertainment*. Graphic Adventure for PS4, Xbox One and PC. 2015 (first).

[25] **Little Nightmares**, *Tarsier Studios*. Horror adventure puzzle platform. 2017.

[26] **Mario Kart (series)**, *Nintendo*. Go-kart-style racing game. 1992 (first)

[27] **MediEvil**, *SCE Cambridge Studio*. Action-adventure hack and slash game originally for Playstation. 1998.

[28] **Metal Gear Solid V: The Phantom Pain**, *Kojima*. Stealth action adventure. 2015.

[29] **Minecraft**, *Mojang*. Sandbox game originally for PC. 2011.

[30] **Monster Hunter: World**, *Capcom*, Action RPG for PS4 and Xbox One (and later for PC). 2018.

[31] **Neverwinter Nights**, *BioWare*. RPG for PC, 2002. Republished for consoles in 2019.

[32] **Nier: Automata**, *Platinum Games*. Action RPG for PS4 and Xbox one, 2017.

[33] **Pac-man**, *Namco*. Maze arcade game. 1980 (first)

[34] **Resident Evil 4**, *Capcom*. Survival horror originally for PS2. 2005.

[35] **Starcraft II: Wings of liberty**, *Blizzard Entertainment*. RTS for PC. 2010.

[36] **Super Mario (series)**, *Nintendo*. Platform games, since 1985.

[37] **Super Smash Bros. for Nintendo 3DS and Wii U,** *Project Sora* and *Bandai Namco*. Fighting for Nintendo 3DS and Wii U. 2014.

[38] **Tetris**, *Aleksej Leonidovič Pažitnov*. tile-matching Arcade. 1984.

[39] **Tetris 99**, *Arika*. Tile-matching online competitive for Nintendo Switch. 2019

[40] **The last of us**, *Naughty Dog*. PS3 and PS4. 2013.

[41] **The last of us Part II**, *Naughty Dog*. PS4. 2020.

[42] **The legend of Zelda (series)**, *Nintendo*. Action adventure game for Nintendo's consoles. 1986 (first).

[43] **The Sims**, *Maxis*. Life simulation game originally for PC. 2000.

[44] **Undertale**, *Toby Fox*. RPG originally for PC. 2015.

[45] **Warcraft III: Reign of chaos**, *Blizzard Entertainment*. RTS for PC. 2002.

[46] **Wii Sport Resort**, *Nintendo*. Sport simulation for Wii. 2009.

[47] **World of Warcraft**, *Blizzard Entertainment*. MMORPG game for PC. 2005.

Ringraziamenti

Prima di tutto, grazie a te, lettore; che hai letto fin qui pieno di determinazione e non hai saltato a piè pari cento e rotte pagine scritte versando sangue, sudore e lacrime per mesi andando dritto ai ringraziamenti scritti in una mattinata di totale delirio. A te, che probabilmente non ti sei neanche accorto che il resto delle pagine non è in italiano; a te, che probabilmente non sai neanche di cosa parla la tesi; e più di tutti a te, che ancora dici che faccio Ingegneria Informatica o che (dato che sono Informatico) ti posso aggiustare il Fax o recuperare la password di Instagram. Tu, lettore, che sei qui a festeggiare con me questo giorno, ti ringrazio.

Ed ora, passiamo all' annovero dei santi che stai aspettando; vi riesco quasi ad immaginare dietro il computer mentre scrivo, che se dimentico qualcuno non vivrò per vedere un altro giorno. Nessuna pressione eh.

Alla mia famiglia (mi sento più scontato del MD a cominciare con la famiglia), che con tutti i difetti, la testardaggine e il mio intramontabile rancoreggiare, siamo un teatrino di improbabile convivenza. Ma in fondo, sono fatto della vostra stessa pasta, ahimé. Grazie perché avete plasmato in gran parte quello che sono, nel bene e nel male; grazie perché avete iniziato alcune mie passioni, dalla più condivisa del cibo alla più personale dei giochi. Grazie perché accettate (quando state di genio) il mio modo di essere me senza cercare di cambiarmi (o fallendo gloriosamente nel tentativo). Siete un guaio, come lo sono io, ma vi voglio bene anche per questo.

Eh va bene, hai vinto; pongo l'accento su di te: MVM (perché così mi hai detto di citarti eh) che mi hai dato la morte per tre mesi non circa l'andamento della tesi, ma solo circa se ti avessi messa nei ringraziamenti o meno. Però si, dai, un ringraziamento per essere l'infelice cavia di ogni abominio io abbia fatto sorgere da quel relitto di computer te lo meriti; a te che hai testato (seppur fallendo sempre miseramente) i miei giochi a oltranza, a te che mi hai ascoltato con pazienza ripetere per ore senza capirci mai una parola, ma più di tutto a te che mi svegli ogni mattina con canzoni sempre più improbabili. Un grazie a te, mamma, che sai essere tanto comprensiva quando vuoi tu, che sei unica al mondo (ringraziando il cielo) e così folle da essere arrivata al punto di scrivermi sul braccio pur di entrare nei ringraziamenti.. Ah, se non ci fossi bisognerebbe inventarti.. però con pareeeecchie modifiche.

Grazie nuovamente a voi, maestre Daniela, Marilina e Rosa; perché i vostri insegnamenti sono ancora i più preziosi anche a vent’anni di distanza. Grazie per tutta la pazienza e la passione con la quale avete gettato le basi di ciò che sono oggi; grazie di aver valorizzato i miei successi senza mai mortificare le mie lacune. Grazie per non avermi mai demoralizzato per la mia grafia, ma per avermi incoraggiato a migliorare e a guardare la sostanza dietro l’apparenza. Grazie per avermi insegnato ad essere curioso invece di schernire le cose che non comprendo. Grazie, ancora, per avermi ripetuto ogni giorno che la vera intelligenza è: trovare soluzione.

Al professor Marco Faella, che ha seguito lo sviluppo di questa tesi dai suoi albori e mi ha incoraggiato ogni giorno a perseguire lo studio di un argomento ancora poco trattato, che ha gettato le basi per questa tesi dalla triennale con il libro sulla programmazione Java alla magistrale col corso sul Game Design e Development;
al Professor Walter Balzano, per i preziosi consigli circa la presentazione e circa gli ulteriori scenari applicativi possibili per la difficoltà adattiva;
al Dott. Carlo Cuomo, che ha fornito un prezioso supporto esterno circa gli argomenti di Game Design e la strutturazione del questionario;
ed a *CodeThisLab s.r.l.* per aver fornito con tanta gentilezza i codici e gli applicativi di numerosi giochi che si sono rivelati preziosi per l’accrescimento personale e lo sviluppo di questo lavoro. Grazie per la disponibilità e il supporto mostrati in questi ultimi mesi, fondamentali per quest’ultimo passo di questa laurea magistrale.

A Francesco, Luigi, Giovanni, Rita, Catello e tutti voi che avete reso gli anni della triennale i momenti di studio più belli della mia vita. Grazie per aver condiviso con me le ansie e le paure di ogni giorno, per le mangiate in facoltà come se fosse sempre una festa, per tutti i giochi tra un teorema ed un progetto, per tutte le cacce ai bug, per le marce al patibolo ad ogni esame e le attese dei risultati, per il tifo agli orali e per essere unici e speciali così come siete. Grazie, per avermi spronato a dare sempre il meglio, tutti insieme, ed aver condiviso con me questo viaggio, impegnativo e difficile, ma che assieme a voi rifarei senza esitare.

A Stefano e Maurizio, al mio fianco in ogni giorno di questa magistrale; grazie per aver condiviso con me quei barlumi di delirio mentre Gödel straziava le nostre menti, per gli sguardi di palese agonia durante lezioni particolarmente pesanti e per le risate soffocate nei momenti meno opportuni; grazie per essere stati vicini anche a distanza in questo ultimo anno; e grazie per la vostra serietà e diligenza, affiancate alla simpatia e a quel pizzico sempre presente di follia.

A Dada, la mia Allys; a te che sei la migliore amica che ho sempre aspettato, a te che mi guardi attraverso come fossi di vetro e mi leggi in ogni mio piccolo gesto. Potrei scrivere per ore di tutto ciò per cui ti sono grato, ma dovrò scegliere con cura le cose più importanti. Grazie di tutte le piccole cose che condividiamo, dai giochi alla scrittura e a tanti modi di essere; grazie della bellissima influenza che hai su di me, con i tuoi valori a cui sei così salda e con il tuo modo di essere, unico e tuo, che mi sprona ad essere sereno nell'essere me stesso; grazie delle interminabili chiacchierate a cuore aperto, grazie per aiutarmi a tenere a bada tutte le mie paure e paranoie; ma più di tutto, grazie di questa bellissima amicizia che condividiamo, grazie di conoscermi così bene e di tenerci a me; e grazie, dal più profondo del cuore, di ogni momento di quotidiana e semplice felicità.

A Dani, fratello più che amico; grazie per tutte le risate e i ricordi che custodirò per sempre gelosamente nel cuore; grazie per l'entusiasmo che mi trasmetti in tutti gli interessi e le passioni in comune; grazie per tutti gli sguardi di tacita intesa, ma ancora di più per quei preziosi momenti in cui parliamo con facilità e complicità. Grazie a te, che da solo redimi in parte quella brutta categoria degli ingegneri! Grazie di essere allo stesso tempo la persona più intelligente che io conosca e il più irrecuperabile degli idioti! Ma più di tutto, grazie di essere diventato il mio migliore amico essendo semplicemente come sei, e facendomi essere me stesso in tua compagnia senza neppure provarci.

A Igi, mio più caro e vecchio amico, che sei davvero il migliore specchio per osservare i miei cambiamenti negli anni. A te che hai scritto assieme a me metà e più delle cicatrici sul mio corpo; a te, mio folle compagno di avventure e di esperimenti, grazie di essere cresciuto assieme a me in questi quasi vent'anni ormai, grazie di avermi aiutato a maturare tanto ogni giorno e di farmi sentire sempre come il bambino di 6 anni che ti ha conosciuto quando sono in tua compagnia. Grazie della preziosa complicità che abbiamo, grazie di questa bellissima amicizia che va avanti da che io ho memoria, grazie della inimitabile competizione che c'è tra di noi, così genuina e così pura, che è per me un tesoro di inestimabile valore. Grazie Igi, di questa vita come mio alleato e come mio avversario, come ragazzino barricato dietro un astuccio e come ragazzo cresciuto e in gamba, come mio amico e compagno di giochi quando eravamo piccoli e come mio più caro amico e per sempre compagno di giochi per tutti gli anni che verranno.

A Vivi, dolce come un fiore e altrettanto forte che potresti crescere su qualsiasi terreno; grazie per ogni momento di genuina amicizia, grazie per tutte le ricette di buonumore che abbiamo fatto e per tutte quelle che faremo ancora, grazie a te che condividi e risvegli in me la passione sopita per la corsa, grazie per i meravigliosi compleanni che abbiamo festeggiato assieme; ma più di tutto, grazie dell' intesa in tutti gli sguardi che condividiamo, grazie delle risate e dei ricordi che hai intrecciato in me in questi anni, e grazie per tutti i ricordi che ancora dobbiamo creare.

A Carla, a te che sei così semplicemente e genuinamente te, come i tuoi disegni, così speciale ed inimitabile senza neanche farlo apposta. Grazie dei preziosissimi ricordi che porto nel cuore. Grazie delle mille e una partite a Mario Kart assieme, delle chiacchierate serali e di condividere con me i tuoi interessi. Ma ancor di più grazie di aver dato vita e colore alla mia fantasia, di avermi dato modo di ammirare il mio primo gioco così come lo immaginavo nella mia mente; non avrei desiderato che nessun altro, se non te, animasse quella goccia e quella fiamma. A te, easter egg nel mio primissimo programma e del mio primissimo gioco, grazie di essere così come sei: una goccia di pura luce nella vita di chi ti sta attorno.

A voi tutti, amici miei, preziosissimi tasselli nella mia memoria, tante volte tanti ormai nella mia storia che non vi conto più. Grazie a voi di essermi accanto in ogni giorno, miei compagni cuochi e alabardieri, pirati, inventori e compagni delle primissime guide; grazie a voi che mi seguite in avventure folli, che ridete con me quando mi faccio male come mio solito e lenite le mie ferite con dolcezza, voi compagni di pizze mangiate come barboni e voi che mi raccontate scorci di quotidianità anche se siete lontani, voi cacciatori di tramonti, voi compagni di film visti assieme, voi che mi sfidate per gioco e mi spronate a essere sempre migliore nella vita, voi compagni di canti stonati, voi bevitori longevi di sangria e insostituibili volti di feste, mangiate, falò, viaggi e avventure. Grazie ad ognuno di voi che asseconda le mie citazioni ad ogni minuto, che conosce il mio folle nascondere easter egg in ogni minima cosa; ma più di tutto che aggiunge ogni giorno un ricordo al mosaico che porterò dentro di me per tutta la vita.

Grazie di aver condiviso assieme le battaglie della vita, con voi nel cuore non ho bisogno di alcuna arma: voi, amici miei, siete la mia forza; e io mi impegnerò ogni giorno per essere la vostra.