GAMEPLAY MORPHING

Memòria del Projecte Fi de Carrera
d'E` gi` yeria e` I` formàtica
realitzat per
Roger Pla` es Camprodo`
i dirigit per
E` ric Martí Godia
Bellaterra, 2 de Febrer de 2010

To Andrew, for your help, patience, support, and all those long hours hearing all my crazy ideas.

To everybody who helped me by taking the test, thank you all, this project would be nothing without your help.

INDEX

1. INTRODUCTION

The idea of creating this project comes from the concern as a gamer about the expansion of the videogame industry experienced in the last 5 years. This industry has been in constant growth since it's beginning being able to introduce new customers to every generation, appealing to different targets.

The first videogame was created in 1958, by William Higinbotham, using an oscilloscope and one analog computer. This game was called Tennis for two (Figure 1), showing a simplified tennis court from the side and a gravity-controlled ball.

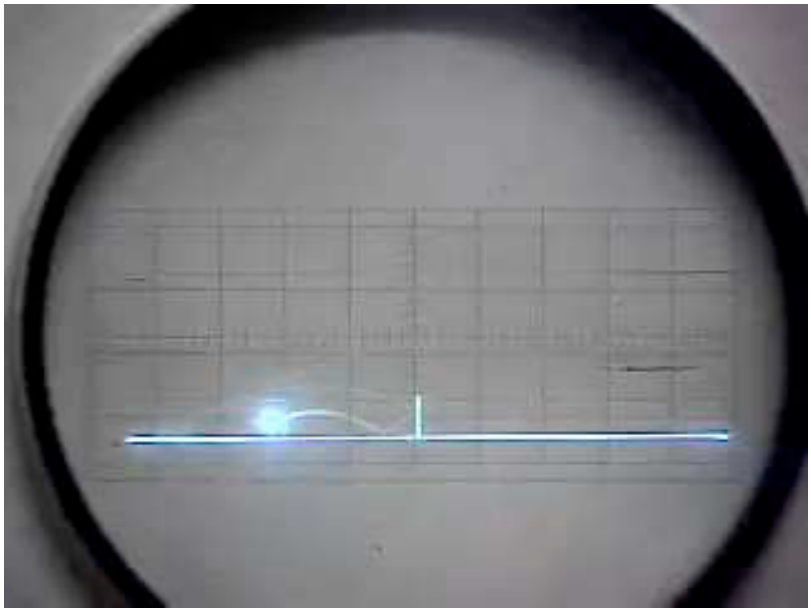

Figure 1 : Tennis for two

After this first videogame, games started to be created using arcade machines, which were then placed in bars and malls. This hardware could not be made personal yet because of the hardware's high price. When in 1980 the first gaming computers appeared, people were able to play the games they had been playing in the arcade machines for the last decade from home. With the introduction of Commodore 64 and Amstrad systems we can declare the start of the videogaming industry as we know it now. When the first videogame consoles appeared in 1984, the industry was able to appeal to more consumers as now, as people played videogames on their tv's.

Up until the year 2000 videogames have been seen mostly as an entertainment for children and nerds, but the expansion of the industry lately towards casual gamers has change the view of videogames in our society.

The first step on this expansion can be related to Sony and it's system, Playstation2 (Figure 2a), when it was launched in 2000. Combining a hardcore software developed by both first and third party studios, Sony established some new franchises directed to the casual gamers, like Buzz, Singstar or Eyetoy … These franchises and others to come were named as social games based on its appeal of being used by groups of people instead of the one or two player games we had known before. These two different software lines helped Sony to distribute an astonishing 120 million consoles breaking the mass market. With the low pricing of computer games and the appeal of the entertainment factor of video games, this made people interested in having a console in their living rooms.

Figure 2a : Playstation 2

Figure 2b : Nintendo DS

Nintendo took the idea of party games and introduced a new way of how to play games, thus trying to appeal to the market Sony had created. With the introduction of the Nintendo DS (Figure 2b) system in 2004, and later the Wii system in 2006's holiday season, the two machines were created with the same goal of an easy user interface that wouldn't scare people who were not used to 8 buttoned controllers.

With the handheld system (Nintendo DS), they had implemented for the first time a touch screen in a videogame system which was also accompanied with the microphone, a dual screen system and the buttons that has been used since 20 years ago. The public's reception to the new system was good but it wasn't until Nintendo did a revision of the system and an

introduction of the "Brain Training" franchise, that sky-rocketed both hardware and software sales to numbers never seen before.

Under the 'protection' of the "Touch" Generations, brand new software was created with the goal to appeal to a market that was never interested in videogames before. (women and senior market)

The introduction of the Wii during E3 2006 (Electronic Entertainment Expo) unveiled the next step of Nintendo, motion system controllers that would make it easier to play anyone, even if they had never played games before. The launch of the Wii system was a success worldwide, Nintendo was selling all it's produced consoles. Due to a stock shortage in the first year of release in all the markets (Countries), they had to increase the production count more than once from 1.2 million to a 1.8 million Wii units produced per month.

The introduction of the motion system controller lead to a new line of games such as Wii sports, Wii fit and Wii music. These games were created to appeal to the widest audience possible, along with the classic company's franchises such as, the well known "Mario", "The Legend of Zelda", Metroid etc.

As Nintendo's software was getting closer to the casual gamers or new gamers, hardcore gamers have been continuously disappointed with the proposals of the company for them as they found the games "too easy to complete" as the games created for the Nintendo system were made to appeal to the widest audience. The industry finds itself with a really fragmented market and most companies release their products to a specific group of players.

1.1 state of art

This section will give a brief description of each step that will need to be followed in order to perform this project. The last section explained the motivation and main goal of this project, so the first thing that will be needed to do in this section will be a research into the state of art of dynamic difficulty adjusting and settable difficulty levels.

1.1.1 dynamic difficulty adjustment

Even if the mainstream games released nowadays still follow the same pattern than the games released fifteen years ago, the discussion about difficulty in games being a dynamic feature has been talked about in the industry for almost a decade. In 2004 Hunicke and Chapman (Robin

Hunicke 2004) proposed a "probabilistic technique that dynamically evaluates the difficulty of given obstacles based en user performance, as the game is running". The term "developed" in Hunicke's paper is 'Dynamic Difficulty Adjustment (DDA)' and is performed by the Hamlet system, a set of libraries embedded in the Half Life[1] game engine.

Defining events into the gameplay and evaluating different game statistics lead the Hamlet system along with the probabilistic formulas to perform a dynamic balancing in terms of the difficulty applied in the game. Depending on a rate of hit/missed shots, lives that were lost or time used in performing a task, the system would recalculate the enemies accuracy and life, and increase players ammunition to balance the difficulty in the game.

This last term leads us to the benefits on balancing games studied in the past. Ernest Adam (Adam 2002) worked on getting positive feedback from balanced games bringing some clues on how to get an ideal game progression that would keep players interested but challenged at the same time. As Figure 3 shows us, the ideal game is the one that starts off balanced but slowly gets unbalanced over time until one player wins, using A and B as the two players that play this game.

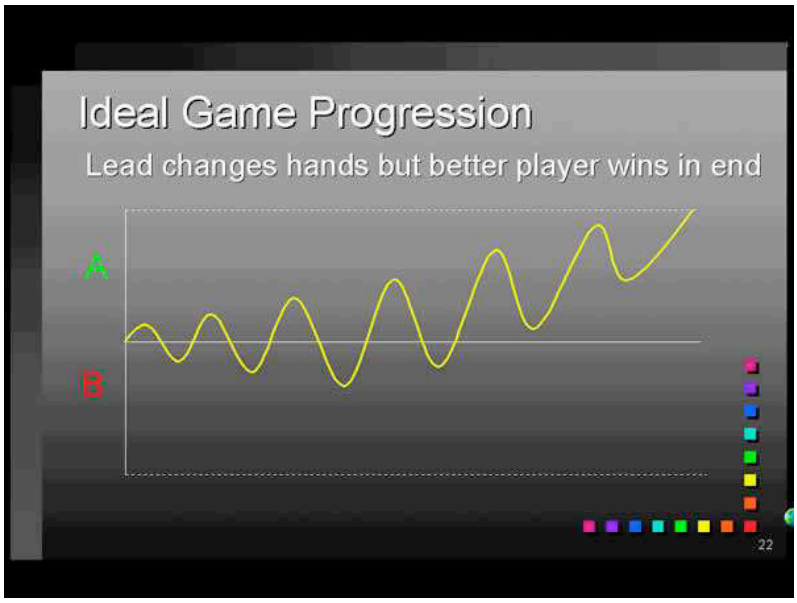

Figure 3 : Ideal Game Progression

The player cannot find the game really easy, leading to a probability of boring the player avoiding a further experience, or cause the player's frustation when the game doesn't give a chance on being beaten. The consumer will most likely spend their time playing a game that offers a challenge but at the same time can be beaten with some effort.

[1] Half Live , 1998 Valve Software

We are going to include player skills at this point of the discussion. The skills needed to complete a game will be increased the more time we spend on the game, making the game easier the better the user gets playing at it. If we want to provide an approximation to the ideal game progression seen before, it is at that point where the balancing has to be performed, reverting the impression of the player that the lead of the game is on the player's side. Raising the difficulty parameters or changing the gameplay at that point, it will become a challenge to the player, who will then need to play better or increase their skills in order access new areas in the game creating a balanced experience that the player will most likely enjoy.

As Figure 4 shows, we have to find a balance between the player's skills and the difficulty of the game, creating a game that is never too easy or too difficult.

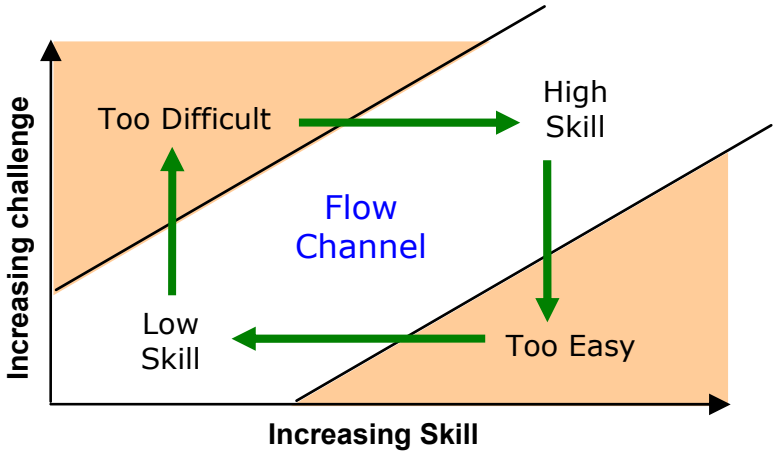

Figure 4 : Game Balance

The Hamlet system and the other approaches seen in this section are biased towards single game DDA (Dynamic Difficulty Adjusting), using the same game data to balance the game's difficulty. This piece of work is biased towards finding a system to create a standard that will allow developers to adjust difficulty levels and gameplay morphing based in the skills of each player.

1.1.2 settable difficulty levels on games

As stated at the beginning of this section, since the start of the industry, the difficulty levels have been taken from the same point view, a set of options to choose from to make the game harder for an expert in whichever genre, or easy for newcomers, known as settable difficulty levels (Figure 5).

When a new game starts, the player chooses which level rather played at, setting the parameters of the gameplay to a specified value.

Nearly all difficulty levels nowadays follow the same pattern, they base themselves on the artificial intelligence of the created sofware and number of elements on screen at once. When a gamer picks a game to play they are offered a range of levels normally between 3 and 5, so they can choose which level is more appropiate to their skills. These levels use the AI of the characters controlled by the CPU as a measure for the difficulty level. As higher the level is chosen, the better AI we will have to front, this system is used in shooters, racing games , fighting games etc..

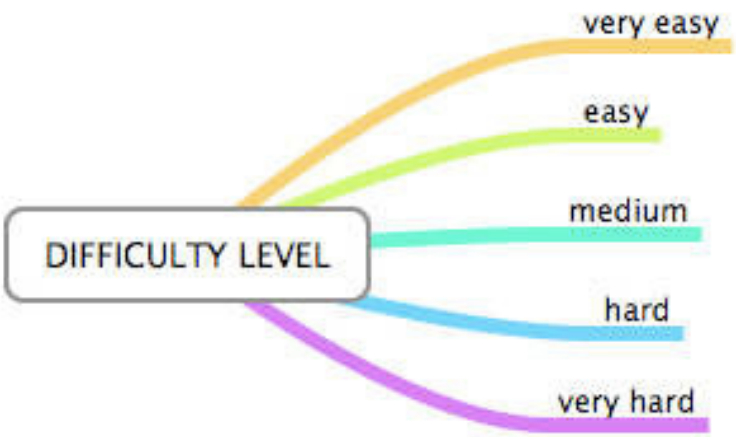

Figure 5 :Standard Difficulty levels

These are not the only approaches to balancing games in the past. Lately Nintendo presented a feature, which approaches the problem from another perspective, that tries to help the user when the game is too hard for them. In June 15th, 2009, the newspaper USA Today (Saltzman 2009) revealed some news on a project Nintendo had been working on that will help gamers in the walkthrough of the game. As it has been reported, if the player finds it hard to overcome a certain point in the game, the game can automatically take over and show the players how it's done, returning the control when any button is pressed. This new feature is avalaible with New Super Mario Bross Wii © and is expected to appear in future games. This demo mode shows the interest of the industry helping players through the gameplay, in any case it doesn't follow this work's direction, based on adjusting difficulty instead of showing the player how it's done.

With this approach the company is helping the users that cannot keep up with the game, but is exchanging the frustation on not being able to beat the game for the frustation of being told what to do, not allowing the satisfaction accomplished of going through without help. This system resolves one of the problems stated in section 1.1, the increasingly unease of core

gamers against the low difficulty in Nintendo's software. A system is yet to be found that will satisfy all users, from casual to hardcore gamers. This is one of the goals of this approach.

There is also disagreement on DDA being a good feature in videogames, as some experts defend the idea of the settable difficulty levels. Ernest Adam dedicates one of his articles discussing DDA (Adam 2008), giving some developping advice on how to integrate DDA into real games. The first fact recommends to increase game difficulty instead of weakening the player taking something the player has already achieved from their arbitrary. It is also recommended to keep DDA optional and to hide it's details avoiding player cheats.

This advice does not need to be followed as they are not mandatory, but all of them have a point on creating a transparent system where the player doesn't feel cheated by the game, and in any case, always make the game harder before we downsize player abilities in the game.

As seen in this section, it's clear that the difficulty levels are a concern to the industry in order to appeal to the maximum amount of people. Having seen different aproaches to the same problem, we will present a new approach in this project.

1.2 the GamePlay Morphing project

According to this, the starting point of our project (GamePlay Morphing, GPM) is to find a way to release videogames that will themselves adapt to the skills of each player giving a challenging experience to all kind of players. The first topic we will need to discuss is how these skills can represented and how we can capture them in an understandable way. After we represent a gamer's skills, we will need a method to obtain the level of these skills and compare them with other players to be able to normalize these abilities.

It's important to remark that these capabilities won't be based only on psychological aptitudes of the player. This project's goal is more interested in the skills a person needs to actually play these games i.e: From the puzzle solving used in most adventure genres nowadays, skills with the buttons as response time of the player or the capability of pressing a sequence of buttons to produce an action in the game amongst other skills that will be discussed in the corresponding session. Some other abstract skills will have to be taken into account too, having to differ between a gamer that finds having some difficulty a challenge and the player that loses interest in playing as soon as they don't know what to do or where to go.

GamePlay morphing is a new approach to the difficulty levels based in different parameters as the ones discussed before. The goal of GPM is to introduce a satisfactory experience to every kind of person that can play a game.

Imagine an adventure game that combines both puzzles, fights and explorations in its gameplay. By the standards of the industry, the player is given a choice to select a difficulty level at the start of the game (easy, normal, hard levels). Based on the level chosen, the player will have to solve from easier to harder puzzles, if the programmers have implemented this, the different enemies will take more or less life everytime they attack or we will have to defeat more or less enemies depending on the level chosen. To summarise, all the parameters will only rise as the level goes upward. (easy to hard)

This project's goal is to find a method where all these parameters can be raised or lowered separately based on the player's level. If the player is good at solving puzzles but finds battling hard or is not interested, we can give a better experience if we increase the level of the puzzles and decrease the number of enemies to defeat. All of these parameters can be re-calculated at some checkpoints of the game to keep adapting the player's skills.

Another example of this adaptation to every player skill can be represented by the fighting genre. Most of the games released in the last 5 years use the same formula, e.g the player has to press a certain combination of buttons as quick as possible to do a move in the game. This formula is good for hardcore gamers but it turns away casual gamers that don't want to learn all the moves and want to just to play occasionally. We could adapt the number of buttons needed to do a certain move based on how often the player plays or even based on the results of the player fighting against the CPU, only concentrating on the skills a player has by pressing a combination of buttons.

These exposed ideas does not mean extra resources or a longer development time, nor does it mean changing the way we create games, we just need to add an easier level in the user interface originally created and switch to it if it's needed.

An application to obtain the player's different skills and evaluate them after will be implemented. One of the main factors will be how many people use this application to be able to have a good range of results to normalize all the data and extract the conclusions of this project.

The project will be divided into five chapters, this being the first chapter. Chapter 2 will be dedicated to define the requirements needed to perform this project. First, we will enumerate skills needed to play videogames and to find a way to represent these skills.

Once we clarify which are the skills to be tested, we will discuss the details in full about how these skills can be used for Gameplay Morphing, giving examples of how existing genres could take advantatge of GPM. The rest of the chapter will be focused on choosing a console to develop the test for and an overview to this console's hardware and the developing environment that will be used to implement the test.

Chapter 3 will discuss the proper implementation process followed to develop the ability test. First we will need to define all the different skill's challenges that will be used in the skill's test. Once we have specified how the test will be performed, we will clarify how to obtain the skill's level of each user, including all the formulas used to calculate each value. The majority of the chapter will be devoted to the implementation of the test for the chosen platform, explaining the developed code if necessary. The last section of chapter 3 will expose some ideas on how to integrate the ability test and the representation of each user skill in real systems, using GPM as a standard for all the games in a console or using GPM as a single game difficulty adjusting.

Chapter 4 will show the obtained results including the test itself, the results people obtained performing it, a proper discussion about these results, considering if they are the expected ones and how they can affect the videogame industry.

Last but not least, chapter 5 will be left to explain the conclusions extracted from the realization of this project, along with the improvements that could be implemented in this project in the future.

2. SYSTEM REQUERIMENTS

This chapter will explain the needs for creating a system that will allow us to observe a player's skills and how to represent them in a graphical way, the GamerShape. It will also be explained how GamePlay morphing can be used in real games in order to improve gaming experience and which hardware and developing environment will be used in order to create a test able to evaluate all types of gaming skills.

2.1 gamershape

In order to study and represent the abilities of the game player we will need to specify which different areas will be tested and how these are going to be represented in an easy way for the videogames players.

As explained in the introduction, these abilities refer to the actual interaction of the player with the controller and the game itself. It is not intended to create a psychological profile of the player in order to adjust the different aspects of the gameplay. We will need to know how each player solves different situations or how good they are interacting with the game. Due to being experienced with playing computer games for a long time and other types of games, the abilities that will be taken have been divided as follows:

Dedication : Fairly the most abstract ability. The point is to track the way each person faces gaming. In the vast amount of players we can find people that experience bordem when they get stuck without knowing what to do next and we obviously can find people that enjoy the challenge of using a lack of the hints option. We can also track in a puzzle solving situation how long it takes for the user to ask for a hint (seen in the "Professor Layton" series for the Nintendo DS system).

Controller : Each player will test their ability by pressing button sequences. When a game requires a 10+ button sequence to perform a special move or combo from a 8+ button controller, the majority of casual players find it to difficult to perform, being left to do simple moves as experienced gamers can perform any move in the game, creating a lack of competivity on the online gaming.

Strategy and Solving : Ability of the player to solve different situations in a game in order to progress further. We will need to track the time used in solving a situation (if it's solved) and the option used to solve the situation (most obvious or not). It will also

be computed as a type of this skill, the ability of the player in strategy games and how good they are at deciphering an AI routine.

Precision : With the most succesful genre in videogames being FPS (First Person Shooter), it is important to test the precision of the player at pointing either with the controller stick or it's pointer to a certain part of the screen and the time it takes to do it, with and without stress of environment.

Timing : The skills that involve time somehow will be included in this category. Reflexes are one of the most important skills needed to play videogames, since racing games, sports games and even shooting games base part of their gameplay on the player's reflexes. Pressing a button at a correct time is another skill mostly used in the music games genre like Guitar Hero, DJ Hero etc..

	Controller	Precision	Timing	Solving	Dedication
Action			X		X
Action-Adventure			X	X	X
Adventure				X	X
Fighting	X		X		X
Life Simulation					X
Music	X		X		X
Racing		X	X		X
Role Playing Games				X	X
Shooter	X	X			X
Sports	X	X			X
Strategy				X	X

Table 1 : Skills needed per genre

Obtaining the results from these five areas, it will be possible to adjust the different parameters of the gameplay to what the player wants or is able to do. By tracking them we can rise the gameplay level as the player rises on their abilities. Table 1 shows which abilities are needed depending on the different genres (Various 2008) in the videogame industry.

As we can see in the table above we will use the dedication to adjust difficulty in all kind of games as dedication involves how the players feel towards gaming. More precised skills are used depending on the genre but it is clear to see that each genre aims towards different skills and that someone that is good at some games does not necessarily mean that they will be good at other games.

Once we have the abilities of each person tested, we need to find a graphical and understandable way to show them, easy to check and compare, bringing even competition to see who's got the best skills. These abilities will be shown as a pentagon which shape determines the player's skills. As seen in some other games such as Pro Evolution Soccer, the pentagon represents the abilities of each soccer player or in Brain Training's case, it shows the results in five different brain areas.

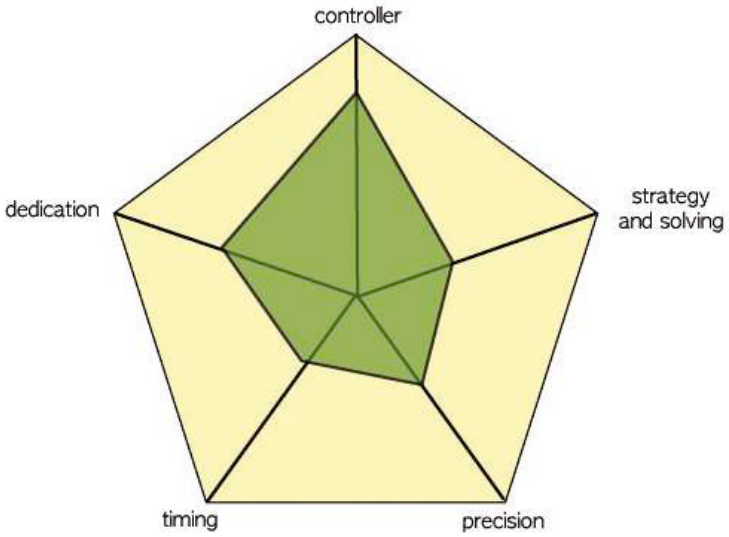

Figure 6 : The GamerShape

As Figure 6 shows, we have created a graphical way to show the skills to the people, under the name of "GamerShape". Each person will have their own GamerShape that will change its form as the person's abilities improve. A cause of a long time without playing will affect these skills and in return are decreased.

Each line between the center and each vertex is used to measure the level of the player in that area on a normalized scale from 0 to 100, 0 being the center of the pentagon and the maximum (100) at the same vertex point.

The method used for the normalization of this abilities will be discussed in further sections along with the test needed to obtain the player skills. The user will need to do some actions and solve different situations to allow the system change the gameplay.

This GamerShape is only the final representation of each player's skills in order to provide a graphical representation instead of providing the user with just a sequence of numbers. As explained briefly above, each ability will have a punctuation in the 0-100 range to create this shape. These values are what we actually need to store not the whole Gamershape image.

It will only be needed to store a five integer position array, from now on will be called **Player Handicap** (Figure 7). It will be discussed if this Player Handicap should be allocated with each user's profile, if it should be shared for all the owned games of the users, or it should be used as a single game handicap having more than one handicap per user.

PLAYER HANDICAP				
Dedication	**Precision**	**Controller**	**Strat. & Solving**	**Timing**
50	30	80	30	25

Figure 7 : Representation of the Player Handicap

2.1.1 extension of the profile

With the last generation of consoles being released, the integration of well-known OS user profile into the gaming industry could allow us more than just storing a friends list, achievements[2] or trophies[3]. With these new all-in-one entertaintment machines that consoles have become, including a hard drive in consoles will become a standard by next generation.

Gameplay morphing could take advantage of these resources to store some data involving each console's user preferences. The "term user preferences" applies in this case to more specific questions about different genres and how the gaming experience can be more satisfactory.

[2] Specific term for special goals that can be completed on Xbox Live-enabled games. Achievement Points are awarded for completing Achievements, which count towards a player's Gamerscore.

[3] Playstation version of Xbox Achievements.

As seen in the last section, there are many genres that would not allow putting all videogames in the same box because between some of these genres the differences in terms of gameplay are quite important. Being able to save all this data would allow developers to be able to change the gameplay of the game before it even starts.

Developers could take advantage knowing these preferences giving a different experience to each player all on one single disc.

2.2 gameplay morphing

This section contains examples on how to implement GamePlay Morphing in real games. When we talk about games and gaming, it is important to differ between single player and multiplayer games.

In a single player game there is only one playable character on screen controlled by the user. There can be other characters that help the player to go through the game, but these characters are controlled by the CPU.

Multiplayer games are the ones where more than 1 playable character is shown on the screen at the same time. Multiplayer games can have cooperative modes and versus modes where players will have to join forces to beat the game in cooperative mode and will have to play against each other in versus modes.

2.2.1 single player

The ultimate outcome for the Gameplay Morphing is for it to become an automatic difficulty level system in charge of the gameplay parameters, but this is not the only outcome we should expect. As seen in the introduction, the growing disagreement of the hardcore sector of the community with games getting easier to appeal the mass market is one of the main reasons this project started. Gameplay Morphing can explore with gameplay beyond setting difficulty parameters to a desired choice.

Based on both skills and preferences of the consumer, developers can serve a personalised game experience to all kind of players, for example we can remove some “scenes” of the game for players that are not interested or allow an experienced player to access optional phases of the game. Each different genre can offer something different for each sector of the market. We will provide some examples on applying gameplay morphing to existing games.

The RPG (Role Playing Game) genre could really take advantage of the system being able to appeal to more consumers. This genre provides the games with the longest average duration, no less than 30 hours in most cases. Turn based battles based on extensive menus, random battle on field and experience based levels can be too much for a newcomer in the genre or for someone more interested in the story line or action/exploration parts of the game. Knowing what each player demands can allow the industry to provide an experience for everyone on a single disc.

The hardcore gamer will have no problem as the game can still be played at 100%, while somebody with less avalaible time or less interest will be able to play a lighter version of the game cutting off the less atractive parts of the game in their opinion.

A game that lasts 40 hours going through all the elements created for it can be reduced to a 20 hour game with no random battles, easier and clearer menu interface (less options displayed) and no optional or secondary content, without leaving the sensation that half the game was not played as the user does not see the chance to do it. We could even ask the player how many hours he will want to play in order to adjust the difficulty so the game can be completed in that time.

If the recently Achievements (Hyman 2007) or Trophies (Entertainment 2008) are used to give the players an extra motivation to re-play the game with more features and the possibility of giving different endings, for each type of gameplay we can extend the game's life in the future. Involving Achievements can be extended to all different genres avalaible today.

The action-adventure genre can also benefit from the gameplay morphing system allowing customers to focus on the gameplay elements more suited in their preferences. Combining combats with puzzle-solving and sometimes statistical character developing, some people can enjoy exploration and puzzle-solving but find caring about statistics tedious. The game could automatically build up these statistics based on where the player is (give the expected level to a character when arriving to a certain point in a game). Different scenarios are found if the consumer is not attracted to puzzle-solving. We can clearly give easier puzzles to be solved or even change the situations to the gamer's preference to allow acces to a new area or item.

Even if the platform genre is not on its peak nowadays, there are still some titles that could take advantage of the gameplay morphing. Recent titles are based on different stages with different goals for each scenario.

These games can be completed (understand completed as seeing the ending and final credits) when 50-60% of the game is achieved, leaving it optional to complete the rest of the game. It may seem irrelevant but not allowing experienced players to access the last stage/s until they achieve 80 or 90% of the game can be an extra motivation for them, while consumers that would complete the game at 50% and not play anymore, still have the option of doing it.

These examples show that most genres can take advantage of gameplay morphing one way or another as seen in both introduction and this section's examples. Implementing these features won't mean a large increase in both resources and time, developers will create the same game just needing to remove different parts of the game as required, there are no additions as the game is made with all that was intended.

2.2.2 multiplayer

Most cooperative and versus multiplayer game modes use absolute metrics[4] in order to give punctuations to set the difficulty level or determine the winner of a task. Thanks to the Gamershape we could also use relative metrics[5] as an optional feature in multiplayer games. The use of relative metrics could be an alternative way to play multiplayer as we would automatically include a handicap for the player with worse skills. This option is specifically aimed towards making friendly[6] multiplayer matches more competitive.

Imagine a group of players with different skill levels the same as families with young members or more experienced users playing along with casual gamers. In these cases we could balance both gameplay or difficulty to match their skills.

For example, in racing games the player with less experience would need less precision with the control of the car in order to balance the race. Most games already incorporate some help giving the player in the last position better power-ups/weapons in order to balance the race but if the problem is due to a low skill level, the player will go back to lower positions sooner or later.

Fighting games could also take advantage of balanced gameplay by allowing fights where the player with lower button skills (if there is a significant difference) can do the same movements

[4] When the outcome of a battle/task is decided on who have better skills.

[5] When the outcome of a battle/task is decided on who performed better in reference to their skills.

[6] As a result of the match no change neither in the status or the number of gamepoints will reflect on the game.

as his oponent with less button combination, instead of just giving him more life that most handicap adjusment systems have already.

When the outcome of a battle or task comes from the number of buttons pressed that a player can do in an interval of time, we could use the relative metrics to determinate the winner. Based on the maximum number of buttons each player can press, the winner will be the one that gave a better performance based on their maximum.

These methods of balancing matches in multiplayer games have to be seen as another way of making games challenging because as we balance the match, the probabilities of having the same winner in different matches decrease allowing more uncertainty on the winner of the match. This would also mean a better approximation to the ideal game progression seen in section 1.3.1, and making more of an enjoyable experience to play multiplayer matches.

2.3 H/W and developing environment

In order to develop a test that evaluates gaming skills, we first need to decide which platform the test will be developed for. Between home console systems and handheld systems, the best option in our case is to develop for a handheld system as it's portablility allows us to develop for it with any computer that includes a gcc compiler and it will be easier to get people to take the test from a handheld system.

The current handheld systems in the market are Playstation Portable and Nintendo DS. Having studied both system's characteristics, the Nintendo DS is the system we have chosen to develop our skill test.

The system's input devices as touch screen or microphone are the perfect way to test skills such as precision or puzzle solving and we still have the input buttons as every other console to test the controller skills for each user.

Also, as it has been stated in section 1.1, Nintendo expanded the videogames market, having surfaced controversy about difficulty of games on their systems, making this the perfect platform and company to develop this project for.

Section 2.3.1 discusses the hardware of the chosen system, the "Nintendo DS". This section specifically aimed to specify the main memory banks the system provides and the different input devices that can be used to interact with the users will also provide a brief history of the system and its different re-designs.

Section 2.3.2 will be dedicated to introduce the developing environment and the libraries that will be used to develop for the Nintendo DS.

2.3.1 nintendo DS hardware

The Nintendo DS's Hardware was developed by Nintendo in 2003 when the success of the company was at the lowest point in it's history. Seeking a revolution in terms of gameplay interaction Nintendo focused their efforts on providing a combination of existing technologies never used in the gaming industry rather than focusing on a powerful hardware that would increment the developing cost and the price of the system. In terms of processing power, the NDS system is similar to the power held by the Nintendo 64 (Figure 8), the Nintendo home system released in 1996 that was quite larger and non portable system.

The Nintendo DS was first released to the American market in November 21, 2004. With measures of 148.7 x 84.7 x 28.9 mm its hardware it's rather powerful for its size. NDS runs on two different ARM processors, a 67.028 MHz ARM946E-S that works as a main CPU and one 33.514 MHz ARM7TDMI coprocessor.

Each one of these processors can be used to create a 2D graphic engine to one of the two NDS's 3 inches LCD. These two screens run at a resolution of 256 x 192 pixels, with the only difference being the built-in touch screen on the bottom LCD.

Figure 8 : Nintendo 64.

There is also the possibility of using a 3D graphic engine, in which case can only be rendered at one screen at a time. Some developers have accomplished rendering 3D graphics on both screens, splitting the graphic power into two less powerfull engines.

In order to decrease the processing power from the core processor, the NDS includes a transform and lighting chip in its hardware. This 3D hardware has a maximum number of 6144 vertices that can be rendered in a frame or single scene.

The system hardware stores it's data on a 4MB RAM commonly known as main memory (see Figure 9 for a memory map of the system). This main memory stores the current processing instructions and data. Both ARM9 and ARM7 processors can access main memory, where as the ARM7 has a higher priority in order to avoid any bus conflicts.

The ARM7 processor also has an exclusive 64KB fast RAM (IWRAM) used to store the ARM7 executable code and data. The bus connection between the ARM7 and the IWRAM is 32bit connection while the rest of the memory banks have a 16 bit connection bus.

The ARM9 processor has both data and instruction cache memories. Since the main memory is cacheable, all the data will be temporarily stored in these caches memory to improve performance.

There is also a nine bank 656 KB video RAM (VRAM). These nine banks can be used for different purposes; as holding the textures of the 3D engine, holding the sprites that are use in the game or holding images that will be directly mapped onto one of the screens. Along with the VRAM there's a possibility to dedicate some of the VRAM memory to the 2D engines, which is caused by the lack of 2D dedicated memory on the NDS system (also known as Virtual video RAM).

Finally we find two banks of fast shared RAM. Each one can be assigned to any of the processors but they are initially mapped to the ARM7, providing along the IWRAM memory, 96KB of RAM memory.

The Nintendo DS system provides the users a Wi-Fi connection to play multiplayer games, both through LAN or the Internet. If the player wants access to the internet, the Wi-Fi hardware allows a connection to any B or G Wi-Fi hotspot, including WEP encryption security.

Last but not least, we find the input devices that provide the user interface. As stated before, the bottom screen includes a touch screen that allows input of one point at a time. There is also a directional pad (Dpad), two shoulder buttons (R- right and L-left) and four buttons known as A, B, X and Y. The NDS has also a standard microphone built-in avalaible for input, allowing input controls by voice commands or by blowing into the microphone.

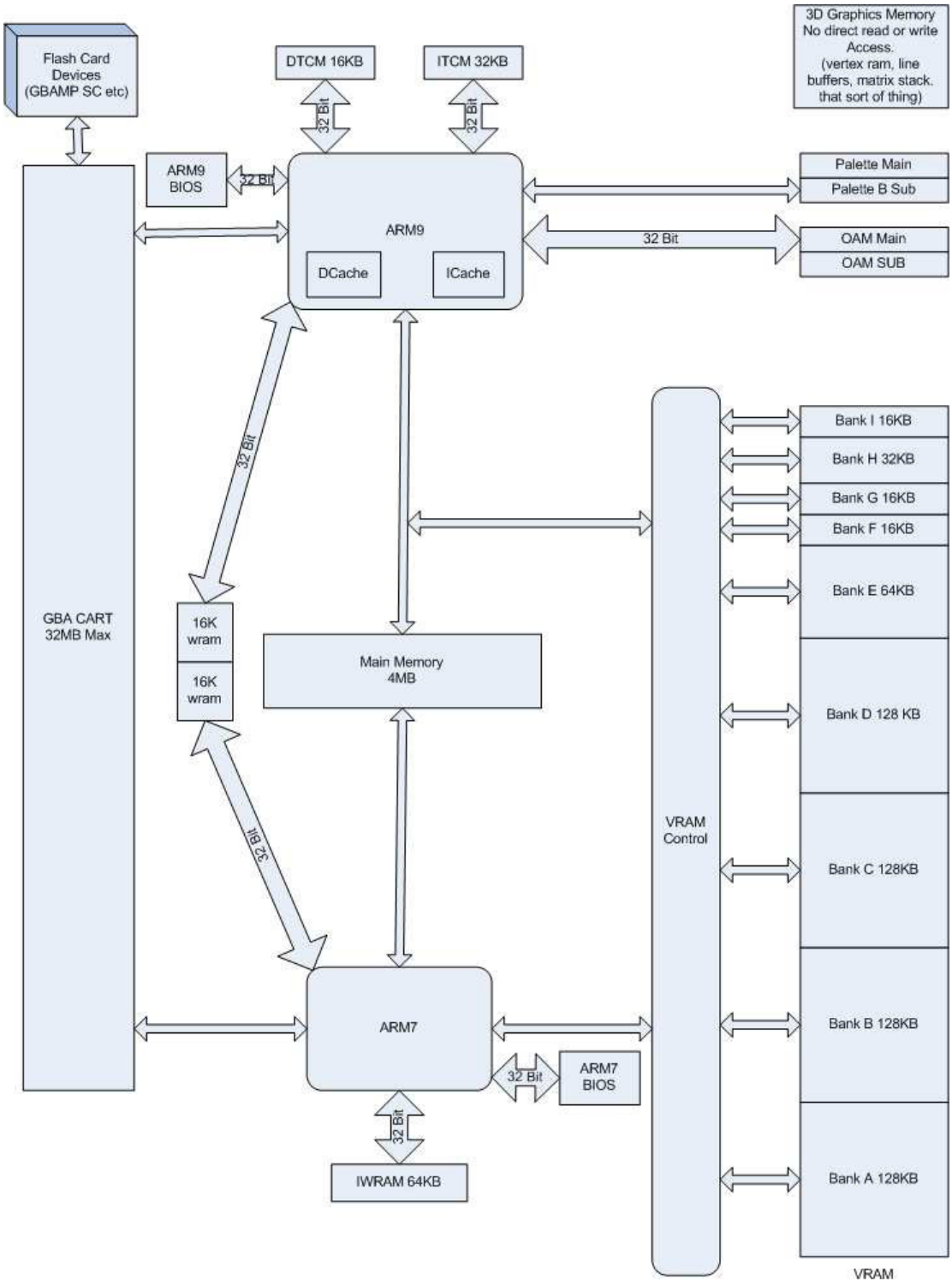

Figure 9 : Nintendo DS Memory Map (dev-scene.com 2006).

Fourteen months after its first release, Nintendo launched the first redesign of their system, the DS Lite. This redesign had the same features and capabilities of the original model, smaller in size being the main atraction (133 x 73.9 x 21.5 mm) and a modern appearance to appeal to a wider audience. On November 1, 2008 a new redesign hit the stores in Japan, called Nintendo DSi system. This third iteration of the system included few hardware changes in comparition with the original design, includind two build-in cameras (one in the interior hinge and one in the external shell). The DSi also included an ARM9E processor running at 133 MHz and an upgrade regarding RAM memory, being now avalaible 16MB of RAM.It also features bigger screens (0.25 inches larger), and a 256MB flash memory to store the photos that can be taken by the camera.

Figure 10 : Nintendo DS , Nintendo DS Lite, Nintendo DSi, Nintendo DS

Nintendo recently launched a new redesign of the system, called DSi XXL, this newer version includes bigger screen as the only modification (new screens ar 0.95 inches larger than DSi screens, and 1.2 inches larger than both DS and DS lite screens). See Figure 10 for the different redesigns of the system.

2.3.2 developing environment

This section will introduce the chosen developing environment, devkitPRO and which of the available libraries have been chosen to implement the source code of the test.

a) devkitPRO

DevkitPro is a collection of development toolchains for the GBA, NDS, GameCube and PSP systems. For the DS you'll need devkitARM which allows the compiling of ARM binaries, and is based on gcc, the gnu compiler collection.

We will only need to install devkitARM which is the toolchain needed to compile ARM assembly code. However, devkitARM also contains the necessary tools to allow DS programming on a higher level language, as C or C++.

It is also important to note that devkitPro is not an Integrated Development Environment (IDE). This environment installs path variables into the Operating Systems and attach these variables to an existing compiler such as gcc or g++. Once the environment is set up, developers can use any existing IDE to compile and make files.

b) libnds & other libraries

In order to develop software for the NDS system, a developing environment is needed. Libnds is the toolchain used for the development of this application. Created by Jason Rogers and Michael Noland (Amero 2008, ch. 4), libnds is a library of register definitions and low level functions in order to provide and open source alternative to the official SDK developed by Nintendo.

Nowadays, libnds is maintained by Dave Murphy through the devkitPro project. DevkitPro offers all the libraries to allow homebrew development for all the consoles, leaving which libraries to install as a developer's choice. In libnds' case, it also includes different tools to allow developers to create data for their projects, providing audio and image converters.

There is another alternative library in order to develop for the NDS system, called PAlib. PAlib is an open source community project with it's goal of creating different basic functions for the DS development. All the documentation of the PAlib project has been translated into seven different languages, including; spanish, french, german and japanese. Although PAlib is not distributed along with devkitPro, it is easy to integrate it's libraries into the environment.

Libnds and PAlib offer general libraries to allow development. There are also other smaller and specialized purpose libraries, as MaxMod(Various) or DSwifi. Maxmod library is a sound solution for the NDS system that allow developers to add MOD, WAV, IT modules to their software, while DSwifi is the library that provides the necessary tools to use the built-in WI-FI in the system. All these specialized libraries can be integrated to devkitPro by just adding the source files to the needed folders.

This project is not focused towards going into full details of the libnds libraries and NDS development tools, using this section to give an overall view on the libraries and its different functions. Table A1 in Appendix A has been extracted from the libnds documentation and a brief description for each source file has been given.

3. IMPLEMENTATION

Chapter 3 contains the process made by us to create a skill test from the different applications used to evaluate these skills until the proper developing process using devkitPRO and libnds.

Section 3.1 is devoted to the design of the skill test, and which challenges will have to be performed by users. Section 3.2 discusses the method created to obtain each Player Handicap from the test's outputs, while section 3.3 will be dedicated to the proper implementation process used to create the test.

3.1 evaluating skills

Having defined the different skills needed to play videogames (see section 2.1), an application is needed to evaluate these skills. This test will be a comprehension of smaller applications where each application evaluates one or two different skills. Each section is devoted to a different skill, precision (3.1.1), timing (3.1.2), controller (3.1.3), strategy and puzzle solving (3.1.4) and finally the dedication (3.1.5).

3.1.1 precision

The first skill to evaluate will be the precision. We will need to evaluate the precision of the player at shooting or touching objects. Taking advantage of the Nintendo DS touch screen, we will define the application named **Shooting Targets** to test the precision.

In this application the user will need to touch with the stylus[7] as many targets as posible during the three rounds the application has. Each round will have 20 sets of targets to be touched. There are three rounds so that each round can have a different difficulty level; easy, medium and hard.

First round will show a single target that appears in a random coordinates of the screen. If the user touches the target, we will update the hit count and create a new target in other coordinates of the screen. If the user misses, the target will dissapear and be replaced by a new target with random coordinates. Each target will disappear on it's own after a second if the user doesn't touch any part of the screen.

The Second round will show two targets at the same time to be touched in a second. Each target will appear randomly on the screen and will have to be touch separately. In this case, we

[7] Pen specially designed for the Nintendo DS to use with the touch screen.

cannot count it as a miss if the user doesn't touch one of the targets, because it can be aiming to the second target. If the user doesn't touch the target but touches a coordinate close enough to the target, we will be suppose that they were aiming at this target and will be counted as a miss.

The number of hits in the second round will be added to the total hits achieved during the first round.

The Third round of the application will be the same as the second round. In this case showing three targets at a time that will dissapear after a second. The same method used in the second round for counting hit and misses will be used in the third one, evaluating if the user's touch coordinates close enough to the target but not the target itself.

The output of the application will be the number of hits achieved by the user during the three rounds, 120 being the maximum number of hits.

TOTAL = (Round 1) 20 hits + (Round 2) 2 * 20 hits + (Round 3) 3 * 20 hits = 120 hits

3.1.2 timing

In order to evaluate the timing skills of each player we will need an application that evaluates the response time of the player. The game Resident Evil 4 (Capcom 2006) introduced the quick time events, where in the middle of a game sequence the player needs to press an explicit button as quick as possible in order to succeed. These quick time events have become common in most adventure and actions games lately and is a good method to process the response time of the player.

We suggest the **Response Time** application for the timing skill.

This application will provide the user with a sequence of buttons that will need to be pressed as quickly as possible and evaluate the response time of the player touching the demanded buttons.

Each step of the sequence will choose randomly which button will be need to be pressed between buttons A, B, X, Y, shoulder buttons R and L which are shown in Figure 11.

To avoid time patterns that will allow the user to know when the image of the button will have to be pressed will be displayed, we will create a random system where each button can appear between 0.5 and 6 seconds after the last displayed button. Once the button is displayed we will check how long it takes the user to press it, considering it as penalty if the button pressed is not the one asked for.

Figure 11 : Nintendo DS buttons' position.

The sequence will have ten buttons to be pressed. The average time needed to press those buttons being the output of the application.

3.1.3 controller

To evaluate the controller ability of each player, we will need more than one application due to the different ways games can be played. For this test we will create two application:

a) Button Tapping

b) Combo Making

a) **Button Tapping**

The first application will evaluate the button tapping[8] skills, needed in adventure, action, shooting and some fighting games. We will evaluate both the time needed to press a button an exact number of times and the amount of times a button is pressed in a interval of time.

The program will have three rounds, the first two dedicated to counting the time needed, and the third round dedicated to the number of times a button is pressed.

[8] Amount of times a button can be pressed in an interval of time.

First round will evalutate the time needed to press a button 10 times while the second round will evaluate the time needed to press a button 20 times. It will be expected that the player won't need a double amount of time due to a higher number of times the button needs to be pressed .

Third round will evaluate a long run of button tapping. The user will have 20 seconds to press a button as many times as possible, aiming at the resistance ability of the player.

The output of the application will be both times needed in first and second rounds, and the amount of times the button is pressed in round 3.

b) **Combo making**

Apart from the button tapping skills the test will also evaluate the ability of each player at quickly pressing a sequence of buttons. These sequences, commonly known as combos, were created for fighting games but ended up spreading to other game genres such as puzzle and sports games.

There will be 10 different sequences to be pressed, starting with the easiest combinations as X +Y or BBB and finishing with 8 button sequences involving both the directional pad, frontal buttons and shoulder buttons. The sequences need to be entered in a short period of time, less than 0.3 seconds between each button of the sequence.

Giving the complexity of some of the combos, players will have two opportunities to enter each sequence in order to ensure that the actual skill is obtained.

The output for this test will be the number of combos correctly entered.

3.1.4 strategy and puzzle solving

As most games require memory to solve some puzzles or remember numbers or codes to go through the game, it is interesting to observe the memory ability each player has. Also the player will be asked to solve two puzzles that are similar to puzzles that can be found in real games. In total there will be three applications that will evaluate this skill:

a) Memory Test.

b) The Door.

c) Hidden Numbers.

a) **Memory Test**

A memory test will be performed giving a random sequence of colours that the player will need to repeat later. The colours chosen are white, red, green, blue and yellow.

The first round of this application will show a sequence of five colours. If the player succeeds, a second round of 8 colours will be displayed. In the case the second round is completed, there will be a third round with a sequence of 11 colours. If the player fails either the first or second round, the memory test will be finished and the counter for the rest of rounds will be 0.

The output of the application will be the number of colours remembered for each round.

b) **The Door**

This first puzzle is called the door, where the player needs to identify a code in order open a door that allows access to a new zone (see Figure 12). Most experienced gamers will find the puzzle simple, as it's been used in a lot of genres and games, but casual gamers would find it more difficult as they are not used to it. There will be also be the option of getting a hint to solve the puzzle, being used to process the dedication of each player.

We will evaluate the time needed to open the door (enter the right sequence) and the use of the hint to solve the puzzle.

Figure 12 : The door puzzle.

c) **Hidden Numbers**

The second puzzle will show and image of a supposed map that could appear in a videogame. Hidden in the image there are four numbers (see Figure 13) that need to be found in order to

solve the puzzle. There is also the possibility to get a hint in order to solve it and the use or non-use of this hint will count towards the dedication computing.

The output of this application will be the same as the first puzzle.

Figure 13 : Hidden numbers puzzle.

3.1.5 dedication

The dedication that each player puts into gaming can be hard to analyse. In the case of this project, dedication will be evaluated based on the use of hints in the different applications to know if the player likes the challenge and solving the problem on their own. If the player uses a hint, it will be considered as the user not enjoying the challenge and prefers easier gameplay.

3.2 obtaining the player handicap

At this point we have defined the skills test that will allow us to generate a player handicap and the gamershape. As stated in section 2.1, each ability will have a value a between 0 and 100 that will come from the test results. In this section we will establish the rules that will be followed to calculate each parameter of the player handicap.

Starting with the precision parameter, it will be totally generated by the output of the targets test. In this case, the maximum value to be scored is 120, so this value will have to be normalized. Once we have the output value we will follow a simple rule of three to obtain the value.

Precision parameter = (output * 100) / 120.

For the controller parameter we have more than one output, the button tapping and the number of combos completed. Giving that the combo making are much more important in videogames than the button tapping (the button tapping is not used as nearly as much as the combos). it has been decided that the combo making will count 70% towards the final value, leaving the other 30% to the button tapping results. These values have been decided by our own experience playing videogames, so it's an empirical decision.

The button tapping results come from three rounds, each round will equally count to the player handicap (10 % each). The minimum possible times for round 1 and round 2 of the button tapping test are 1 and 2.4 seconds, while the maximum achievable value for round 3 will be 160 presses.

The minimum time for round 1 means the user should press the button every 0.1 seconds, while for round 2 the button should be presses once every 0.12 seconds. For the third round we will count as the maximum punctuation 160 presses in 20 seconds (one press each 0.125 seconds).

Every 3 hundreths of a second or presses above the minimum values will have an impact of -1 in the total value for each round.

Button tapping R1 = 100 - ((timeR1 - 1) * 100) / 3.

Button tapping R2 = 100 - ((timeR2 - 2,2) * 100) / 3.

Button tapping R3 = 100 - ((160 - R3output) / 3).

Controller parameter = 0.7*(combo output * 10) + 0.1*(round 1 button tapping) + 0.1*(round 2 button tapping) + 0.1*(round 3 button tapping).

The timing parameter also comes from just one output, the response time application. The output value is the average response time from the ten button sequence the user will have to press.

Robert J. Kosinski has done a research (Kosinski 2009) on reaction time types and times, as single and choice experiments. A single reaction time is the one that comes from only one stimule (seeing how a light turns on), while the multiple choice have more than one stimuli (the same light can turn on in different colours). The normal reaction time for a simple stimuli is an average of 0.2 seconds if we are talking of a visual stimuli (seeing an image). When there's more than one choice we follow Hick's reaction time experiments, he concluded that response time was proportional to log(N), where N is the number of possible stimuli. For a six choice response time the minimum value is 0,6 seconds, and we will use this value as the minimum

achievable value. If the user's response time is 0,6 seconds the timing parameter will be 100, decreasing a point for each hundreth of a second over the minimum value.

Timing parameter = 100 - (output value - 60).

The puzzle solving & strategy parameter comes from the outputs of three different tests. There is a proper puzzle, a visual puzzle and a memory test. The proper puzzle will count 40% towards the puzzle parameter while the visual puzzle and the memory test will count 30 % each.

Both puzzle's minimum solving time are estimated at 15 seconds, every three seconds above that time will have an effect of minus one point in the total value for that puzzle. If the user gets a hint, it will have a penalty of 30 seconds (double the minimum resolve time). In the case of the memory test, each round will count a 10% to the total value. If the round is completed it will give the maximum points, if not, we will give a mark according to the total of colours remembered per round.

Door value = 100 - (solve_time -15 + (hint_value * 30)/3).

Numbers value = 100 - (solve_time - 15 + (hint_value * 30)/3).

Memory R1 = 20*colours_remembered.

Memory R2 = 12.5*colours_remembered.

Memory R3 = 9.1*colours_remembered.

Puzzle parameter = 0.4 * (Door value) + 0.3 * (Numbers value) + 0.1 * (Memory R1) + 0.1 * (Memory R2) + 0.1 * (Memory R3).

The dedication parameter comes from the use the player gives to the hint on both puzzles. If the user decides to not use a hint the maximum value will be given (50) but if they use it, we will use the next formula:

Dedication parameter = 0.5 * (door_time_hint/3) + 0.5 * (numbers_time_hint/3).

If the hint is used in just one puzzle, we will add the maximum value for one puzzle with the result of the formula of the other puzzle.

Once we have obtained the player handicap it will be possible to obtain the GamerShape for each player. For the realization of this project, we will use Microsoft's Excel 2007 to perfom all the needed calculations to obtain each player handicap and use it's graphic's engine to draw each gamershape. Excel 2007 can draw a filled radial graph that is close enough to the GamerShape presented in section 2.1 (see Figure 14).

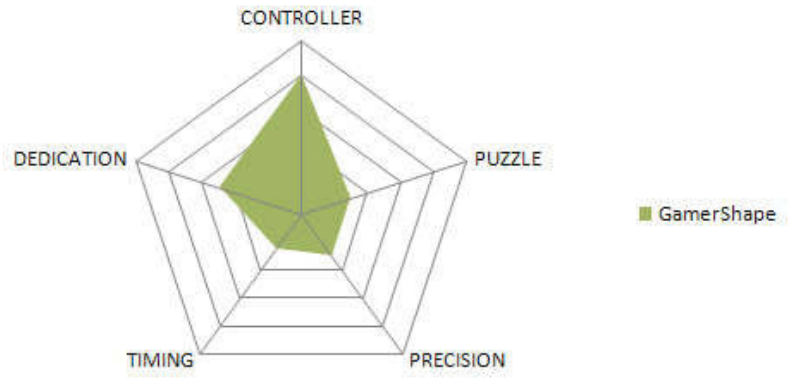

Figure 14 : Radial graphic for the values in figure 7.

3.3 developing process

This section will give us a description of the developing process needed to implement the skills test for the Nintendo DS system. Starting from the environment and libraries set up, to the most important code needed to develop each application.

Section 3.3.1 will show the steps needed to set up the developing environment, while section 3.3.2 will be devoted to explain the developing process and source code of the created application.

3.3.1 setting up the environment

The platform used to develop this program is an Apple MacBook with MacOs 10.6 (Snow Leopard OS).

First we will need to install devkitPRO and the needed libraries to the platform:

1) We will download the needed libraries from devkitPRO web page. The versions used for this project are libnds-1.3.2, devkitARM-r25 and maxmod-1.0.3.

2) Once we have the libraries we will need to create a folder in /usr/local/ named devkitPRO and extract the binary files of the libraries into this new folder, having a new folder for each library.

3) We can also download some code examples to start programming. They must also be located inside the devkitPRO folder.

4) When all the libraries are installed we have to add some variables to our execution environment file, for MacOS .bash_profile file located in the home folder.

```
export DEVKITPRO=/usr/local/devkitPRO
export DEVKITARM=${DEVKITPRO}/devkitARM
```

Now we are ready to start writing some code and compiling it with gcc, having an executable .nds file as the output of the process. The code can be written using any application, in this case Xcode[9] is used to write the C++ code.

3.3.2 developing the skills' test

This section is aimed to explain the developing process followed to create the skills application written in C++. Libnds is a really big library and it is impossible to put all the functions used and the explanation for all of them in this project. For each of the small applications of the test we will be explain their most important feature, these features being; the use of timers, sprites or loading .png files as backgrounds.

Before we start with the proper code of the developed application, we suggest reading the developing basics on libnds that we have written in Appendix B.

The program's main function will need to call each one of the subroutines that perform each small test and retrieve the output data of each routine. This function is mostly written with C/C++ code, with calls to functions and screen printing.

Each subsection will be devoted to one application of the program :

a) button tapping function.

b) the door & hidden numbers functions.

c) the response time function.

d) the shooting targets function.

[9] Xcode is Apple's premiere development environment for Mac OS X.

e) the combo making function.

f) the memory test function.

a) button tapping function

The button tapping function have to count how long it takes for the player to press the same button 10 and 20 times, and the number of times the button is pressed in 20 seconds. Apart from the input handling code (seen in section 3.3.2.1) needed to count the times the button is pressed, the use of timers that can count up to a undredth of a second.

Timers are handled by the DS's interrupt system. We will need to use the macros defined in timers.h to set up the timers frequence and other parameters (see Code 1).

```
// this line sets timer0 to be overflowed 100 times for each second
TIMER0_DATA = TIMER_FREQ_1024(100);
// list of setting for timer0, enable the selected timer, enable a 1024 divider
// the timer will count at 33.514 / 1024 MHz, and enable the generation of an
interrupt when the timer is overflowed
TIMER0_CR = TIMER_ENABLE | TIMER_DIV_1024 | TIMER_IRQ_REQ;
//set which function will be called when the timer is overflowed.
irqSet(IRQ_TIMER0, timerFunction);
```

Code 1 : Setting up timer 0.

The function timerfunction will be called everytime the timer overflows (100 times per second), but we will need to call it from our function to know how many seconds the user needed to press the button. Timerfunction will call contador function with value 0 in order to increase the count of hundreths, and also contador can be called with value 1 when we want to retrieve the numbers of hundreths counted until some point (see Code 2).

```
int contador(int mode)
{
     int mili2;
     if(mode==0)
     {
             mili++;
    }
     if (mode==1) {
             mili2=mili;
             mili=0;
             return mili2;
     }
     return 0;
}

void timerFunction()
{
     int dumb;
     dumb=contador(0);
}
```

Code 2 : Timerfunction and contador

With this code we will only need to enable or disable the timer interrupt when needed (see code 3). Once the player presses A to start counting, we enable the timer interrupt until the number of required times the button needs to be pressed is achieved.

```
while (!(keysDown() & KEY_A)) scanKeys
();

     irqEnable(IRQ_TIMER0);
     while (cont!=(i+1)*10) {
            scanKeys();
            if (keysDown() & KEY_X)
            {
                  cont++;
            }
     }
     irqDisable(IRQ_TIMER0);
```

Code 3: Enabling and disabling timer interrupt.

b) the door & hidden numbers functions

This function will count the seconds needed to open a door using a code that is hidden in an image. These functions use sound effects to let the players know if they are entering the correct numbers or not. Using the maxmod library functions, we can easily load the sound effects and play them when needed.

We need to store our .WAV files into a folder so the library can automatically create the soundbank files used during programming. Once all files are included in our soundbank, we need to initialize it and set up the parameters for each effect (See Code 4).

```
//initialise soundbank file
mmInitDefaultMem((mm_addr)soundbank_bin);
// load sound effects
mmLoadEffect( SFX_THRUST );
//set up effect

mm_sound_effect thrust = {
            { SFX_THRUST } ,         // id
            (int)(1.0f * (1<<10)),  // rate
            0,           // handle
            255,  // volume
            128,         // panning
     };
```

Code 4 : Setting up sound effects.

Once we have loaded and set it up the sound effect, we will just need to call it when we want it to be played (see Code 5)

```
switch (opcio) {
            case 1:
                  combo=1;
                  break;
            case 2:
                  combo=2;
                  break;
            case 3:
                  combo=3;
                  break;
            case 4:
                  combo=4;
                  mmEffectEx(&boom);
                  break;
            default:
            break;
}
```

Code 5 : Playing a sound effect.

Maxmod library also allows developers to play background melodies throught 16 different audio channels, to see more information of the Maxmod library refer to it's documentation(Maxmod 2008).

As stated in section 3.1.5, these functions will also need to track the dedication of each player. The test informs the possibility of getting a hint for solving the puzzle by pressing the Select button. If the player ever presses the button, the hint will be shown and the seconds elapsed since the start of the puzzle will be stored in a variable. This way, we can know how long it took to ask for a hint and calculate the dedication of the player.

c) the response time function

This function shows an on screen image of the button that requires to be pressed and calculates the hundreths of a second the user needed to press it. The images shown on screen are portable network files (png), but first they need to be converted so the DS hardware can manipulate them.

Libnds includes an external library called GRIT, developed by Jasper Vijn (Vijn 2007). This library reads the png files and converts them to binary data of various bitdepths which can be directly put into VRAM, but also can convert the images to tilesets[10].

[10] A 32x32 pixel image is converted to a tileset of 8x8 pixels tiles, so it reduces the overhead in processing power updating the screens. The hardware won't need to update the whole screen if it's not needed, just updating the needed tiles.

Each png file has to be associated with a GRIT file (see Code 6) that includes the rules the library will use to convert the file (see Appendix B for the original GRIT documentation extracted from the project's webpage). The function decompress will allow us to copy the binary data to the desired memory bank when we call it with the source, destination and compression type as parameters.

```
-W3
# disable alpha and set opaque bit for all pixels
-gT!

# use lz77 compression
-gzl

# 16 bit bitmap
-gB16
```

Code 6 : GRIT file used to convert button images.

To determine the user's response time, the timers exposed in section 3.3.3.2 are also used in this application. In order to avoid patterns, each time this application is executed it will generate a random sequence of buttons to be pressed, and each button will appear after a random interval of time (between 0,5 and 6 seconds).

d) the shooting targets function

To be able to show a background and the targets at the same time, we will need to configure the main engine to be able to show sprites. The Nintendo DS handles sprites via the object attribute memory (OAM) and libnds defines two basic structures that each sprite will use to render itself, SpriteEntry and SpriteRotation. The SpriteEntry structure contains the "ObjectAttributes", and they can be set using the top level function oamSet (see Code 7).

```
oamSet(graphics engine,
     oam index (0 to 127),
     coodx, coody,
     priority, lower renders last (on top),
     palette index,
     SpriteSize_64x32,
     SpriteColorFormat_256Color,
     pointer to the loaded graphics,
     sprite rotation data,
     double the size when rotating?,
     hide the sprite?,
     flipx, flipy,
     apply mosaic?
     );
```

Code 7 : oamSet function.

When developers want to work with sprites, the first thing needed is to initialize the object attribute memory at the start of the application using the oamInit function. Once the OAM is set up, we need a memory address where to allocate the sprite graphics data, providing libnds function oamAllocateGfx with the paramaters graphics engine, sprite size and sprite colour format we can store the memory address that the function returns.

To copy the graphics to the obtained memory address we can take advantatge of the direct memory access the system provides. When DMA is activated, the DMA controller takes control over the hardware (halting the CPU), the data is transferred and the control is given back to the CPU.

Transferring the data through DMA is much faster than doing it through a loop in the main code. Libnds library provides developers with a couple of functions that allow DMA data transferring, being used in this application the dmaCopy function. This function needs three parameters, source, destination and lenght of the transferred data, and as seen in section 3.3.3.4, we use GRIT library to obtain the converted data and its length. When the graphics data is finally stored in the obtained memory address, we will use the top level oamSet function to set the parameters as coordinates, the priority for each sprite and update the object attribute memory. Code 8 shows the steps used in the application to load the target sprite.

```
oamInit(&oamMain, SpriteMapping_1D_32, false);
u16* gfx2 = oamAllocateGfx(&oamMain, SpriteSize_32x32,
SpriteColorFormat_256Color);
dmaCopy(//0-3
                  dianaTiles, //grit generated
                  gfx3, //destination (oamAllocateGfx() will
                  dianaTilesLen);//grit generated

oamSet(&oamMain,  //main graphics engine context
     1,           //oam index (0 to 127)
     164, 100,    //x and y pixle location of the sprite
     0,           //priority, lower renders last (on top)
     0,           //this is the palette index
     SpriteSize_32x32,
     SpriteColorFormat_256Color,
     gfx2,        //pointer to the loaded graphics
     -1,          //sprite rotation data
     false,       //double the size when rotating?
     false,       //hide the sprite?
     false, false, //vflip, hflip
     false        //apply mosaic
 );
```

Code 8 : Code needed to load an sprite and show it on screen.

e) the combo making function

This function provides the user with 10 different button series to be entered. To complete a series each button has to be pressed before 0,3 seconds once the user presses the first button of the sequence. This time condition is handled via timer's interrupts, as seen in section 3.3.3.2.

The complexity of this application consists in tracking the pressed button at each step and give an error either the button is not the required one or the time interval elapses.

Code 9 shows the code needed to track the user's input for combo number 6 (right,down,left,A,B,A).

```
//SIXTH COMBO  ->|<- ABA
     combo=0;
     ok=0;
     i=0;
     decompress(combo6Bitmap, video_buffer_main,  LZ77Vram);
     printf("\n\n\t*****COMBO 6***** \n");
     while(ok==0 && i<2)
     {
     while(combo==0)
            {
            scanKeys();
            if (keysDown() & KEY_RIGHT) {
                  cont2=contador3(1);
                  while((cont=contador3(1)) < cont2+30)
                  {
                  scanKeys();
                        if (keysDown() & KEY_DOWN) {
                              cont2=cont;
                              while((cont=contador3(1)) < cont2+30)
                              {
                              scanKeys();
                              if(keysDown() & KEY_LEFT)
                              {
                                    cont2=cont;
                              while((cont=contador3(1)) < cont2+30)
                              {
                              scanKeys();
                              if(keysDown() & KEY_A)
                              {
                                    cont2=cont;
                              while((cont=contador3(1)) < cont2+30)
                              {
                              scanKeys();
                              if(keysDown() & KEY_B)
                              {
                                    cont2=cont;
                              while((cont=contador3(1)) < cont2+30)
                              {
                              scanKeys();
```

```
                              if((keysDown() & KEY_A) && ok==0)
                              {
                                    combo=1;
                                    ok=1;
                                    combocount++;
                                    printf("\n\n\t COMBO 6 OK !");
                              }
                              if (keysDown() & ~KEY_A && combo==0) {
                              combo=1;
                              }
                              }
                              if(combo==0) combo=1;
                              }
                              if (keysDown() & ~KEY_B) {
                              combo=1;
                              }
                              }
                              if(combo==0) combo=1;
                              }
                        if ((keysDown() & ~KEY_A) && (keysDown() & ~KEY_LEFT))
                              {
                              combo=1;
                              }
                              }
                              if(combo==0) combo=1;
                              }
                  if ((keysDown() & ~KEY_LEFT)  && (keysDown() & ~KEY_DOWN)) {
                              combo=1;
                              }
                              }
                              if(combo==0) combo=1;
                  }
                  if ((keysDown() & ~KEY_DOWN)  && (keysDown() & ~KEY_RIGHT)) {
                        combo=1;
                        }}
            if(combo==0) combo=1;
            }}
            if(ok==0) printf("\n\n\t COMBO 6 FALLADO\n");
            if(ok==0 && i==0)
                  {
                  starttime=checktime();
                              while(endtime-starttime!=2)
                              {
                              endtime=checktime();
                              }
                  printf("\n\nSEGUNDA OPORTUNIDAD PARA COMBO 6\n\n");
                  combo=0;
                  }
      i++;
      while((keysDown() & KEY_Y) || (keysDown() & KEY_X) || (keysDown() & KEY_A) ||
(keysDown() & KEY_B)) scanKeys();
      }
```

Code 9 : Combo number 6

There is a second chance to enter each combo if there is an error at the first try, with a wating period of 2 seconds (use of checktime function). At each step of the series, if a button is pressed and is not the required button, the combo variable will be set at 1 while the ok variable will remain at value 0. If at the end of the loop the ok variable is 0 and we are still at the first iteration, the second oportunity will be granted.

f) the memory test function

The memory function is mainly written in C code using some of the libnds' top-level functions to display the backgrounds. This application consists of three rounds but the user will have to complete each round to be able to access the next round, in the case of a fail, the test will end.

Each round will give a sequence of 5, 8 or 11 colours generated randomly. Each sequence will be a combination of 5 different colours (white, green, blue, yellow and red) with the only condition that the same colour cannot appear twice in a row. Each colour of the series will be displayed during 3 seconds. Once the series is finished, the user will have the opportunity to enter a sequence of colours, see Figure 15 and Code 10 for the input sequence handling.

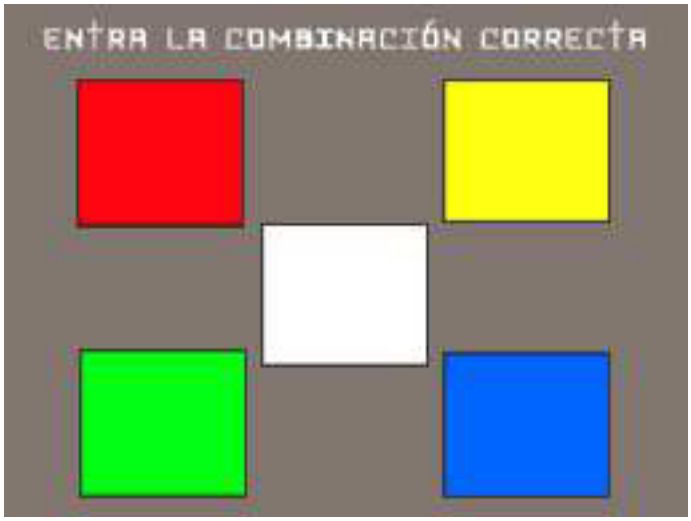

Figure 15 : Input sequence background

Once the image is shown the code checks which pixels are touched on the screen. If the touched coordinates are inside the range of a colour, the colour pressed will be store in the sequence array.

If the touched area is not in the range of any colour, an error message will be displayed on the screen, communicating the invalid entry. When the input sequence is complete the code compares the generated sequence with the input sequence giving access to next round if both sequences are equal.

```
decompress(colorsBitmap, video_buffer_sub,  LZ77Vram);
while (it<n+1) {

            scanKeys();
            if(keysDown() & KEY_TOUCH)
            {

                  updateInput(&touchXY);
                  opcio=6;
if(touchXY.px >= 166 && touchXY.px <= 226 && touchXY.py >= 131 && touchXY.py <= 185) opcio=1;
if(touchXY.px >= 96 && touchXY.px <= 158 && touchXY.py >= 83 && touchXY.py <= 135) opcio=2;
if(touchXY.px >= 29 && touchXY.px <= 90 && touchXY.py >= 129 && touchXY.py <= 184) opcio=3;
if(touchXY.px >= 28 && touchXY.px <= 89 && touchXY.py >= 28 && touchXY.py <= 83) opcio=4;
if(touchXY.px >= 162 && touchXY.px <= 226 && touchXY.py >= 28 && touchXY.py <= 81) opcio=5;

                  switch (opcio) {
                        case 1:
                              input[ronda][it]=0;
                              it++;
                              break;
                        case 2:
                              input[ronda][it]=3;
                              it++;
                              break;
                        case 3:
                              input[ronda][it]=2;
                              it++;
                              break;
                        case 4:
                              input[ronda][it]=1;
                              it++;
                              break;
                        case 5:
                              input[ronda][it]=4;
                              it++;
                              break;
                        default:
                              videoSetModeSub(MODE_5_2D | DISPLAY_BG0_ACTIVE);
                              consoleDemoInit();
                              bgUpdate();
                              iprintf("\n\n\n\tINVALID COLOUR ENTRY\n");
                              iprintf("\n\tINTRODUCE AGAIN THE COLOUR\n");
                              iprintf("\n\t\tPRESS A TO CONTINUE\n");


                              while(!(keysDown() & KEY_A))
                              {
                              scanKeys();

                              }
                              videoSetModeSub(MODE_5_2D | DISPLAY_BG3_ACTIVE);
                              bgUpdate();
                              decompress(colorsBitmap, video_buffer_sub,  LZ77Vram);
                              fora=0;
                              break;
                                    }

      }
      }
```

Code 10: Sequence input handling.

3.4 approach to real systems

Once we have designed and developed the application, we will need to integrate our GamerShape into real systems, making it a standard by introducing it into every users profile, or using it as a single game difficulty adjusting.

3.4.1 making it a standard

This section will discuss the integration of this method on real videogames systems as the Nintendo Wii, Xbox 360 or Playstation 3. With the introduction of Miis and Avatars users have a visual profile to check (see Figure 16 for a simulation of an integration of the GamerShape on Xbox 360), and with the addition of Achievements and Trophies, players usually check their profile. With the introduction of the GamerShape in each user's profile, it would be much easier for every user to access their skills level or being able to change their user preferences as they get more experience.

Figure 16 : Representation of the GamerShape integrated on the XBOX360 dashboard.

Once a new profile is created, the system will ask the user to perform an ability test to create a player handicap that matches user's level and store it's values into the user profile. This will allow the system to recreate the GamerShape at any given time. After that, we will need to ask the user some questions to obtain each user preferences and add them too.

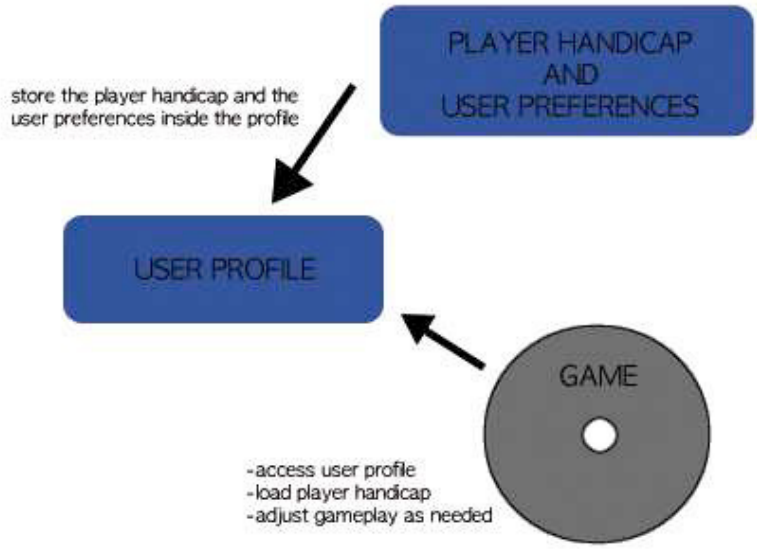

Figure 17 : Integration scheme.

This data will have to be accessible for all the different games played on the console. Once we start playing a game, while loading, access to the profile will be needed in order to adjust the game difficulty and gameplay to the required level (see Figure 17 for an scheme). When the player finally completes the game, based on their performance, an update of the Player Handicap would be needed to approximate the new skills after completing the game. The user can also update his Player Handicap taking the ability test again once every two or three months, if they don't play usually.

Associating the GamerShape and user preferences with the extension of the profile will allow our friends or any player to check our skills, bringing online gaming a step further. Using the internet connection, any player could download their profile to a friend's console in order to adjust the difficulty or gameplay while playing multiplayer. We will not only know the level of a player as a number, we will know which skill each player is good at. As more games include online cooperative modes, players will able to choose team mates that compensate their own skills or players with equal skills so the gameplay doesn't change.

3.4.2 single game gameplay adjusting

With the amount of companies and studios in the videogame industry, people may think it's impossible to reach an agreement on standardizing difficulty levels, the GPM can be used to come to a standard among different studios in a major distributor. If we base the project from this point of view, specially if it's a third party distributor, getting players to play a test for each company or the use the GamerShape to represent their skills wouldn't be the appropiate method to proceed.

The market is not gonna be interested on having a dozen of gamershapes in their profile to compare, less if they have to go through a skill test everytime they purchase a game from a new company.

We must obtain the same results while the process is invisible to the consumer, the goal is to obtain the data from the player and calibrate gameplay based on the skills levels. Even considering saving data, thanks to the large capacity hard drives consoles include lately, this data will need to be actualized at some point or created for the first time. It is in this case where we need to obtain the data without letting the player know when it is done.

This can be achieved taking advantatge of the tutorial. Most games have a tutorial at the start of the game to let players get used to the controllers, training mode if avalaible or even the game walkthrough. Single player modes or campaigns usually include a first "level" or zone used as tutorial to show actions or situations the game is based on. Obtaining different data from these tutorials could allow developers to determine the skills of the gamer playing their own game and adjust the parameters of the gameplay without changing the game experience.

The time needed to finish the tutorial and the simple actions that compound it, the outcome of the different actions as damage received, number of deaths or even shots needed to defeat an enemy are just some examples of how the data can be obtained. If GPM is used, once a game has been finished, if it has been started again or the second time a player does the tutorial, the data results will in most cases improve from the first time it was played, raising the difficulty where needed in this second walkthrough or allow the player visit new areas that in the first time weren't avalaible.

Until this point, GamePlay Morphing can be seen as an automatic difficulty level chosen at the start of the game. GPM's goal is to improve game's experience making the game more challenging as the user gets better at playing at it. We can mark some checkpoints through the game where more data will be obtained and establish the new gameplay scene. For a 20 hour single player game, five checkpoints can be performed to see how the player manages the game. The data stored, along with the game data, can be something similar to the player handicap explained before and be saved on the hard drive.

Single Game dynamic difficulty adjustment has been discussed by a large amount of industry insiders as seen in section 1.1. Single game adjustment is not the main focus of this work as it is more focused at creating a standard in the industry. For more detailed information read Hunicke's (Robin Hunicke 2004; Hunicke 2005) work.

4. RESULTS

This chapter is dedicated to show the obtained results in this project. First, in section 4.1, we will show the skill's test running on a Nintendo DS Lite. Section 4.2 will give a description on the expected results, based on studies in how people that play videogames have better skills. Section 4.3 will finally discuss the results obtained by people actually performing the skills' test.

4.1 the test

Once the test has been developed and tested we can show some images of the test running in an actual Nintendo DS Lite(see Figures 18-24).

Figure 18 : Combo making application.

Figure 18 shows the combo application running in the system. We can see on the top screen the button sequence the user has to press while the bottom screen prints the results for each combo. If the combo is achieved, the application will print an Ok message, while if the combo is a fail, an error message will appear on screen.

Figure 19 : The touching targets application.

Figure 19 shows the first round of the shooting target application. This application displays on the bottom screen a background with the image of a forest and renders the target as a sprite on top of the background.

Figure 20 : Response time application.

Figure 20 shows the response time application. The required button to be pressed appears on the bottom screen and stays there until the user presses it. After a random interval between 0,5 and 6,5 seconds a new button will appear on the screen.

Figure 21 : The door puzzle.

Figure 21 is The Door puzzle. As we can see, the top screen shows the puzzle itself, where the colours that appear in the door are linked to colours of the clock. The bottom screen shows us a keypad where the right combination has to be introduced.

Figure 22 : Hidden numbers puzzle.

The image on the top screen of Figure 22 shows a hypothetical game map with numbers hidden in it. When the user finds the correct combination, it has to be introduced in the bottom screen's keypad.

Figure 23a : Memory test sequence.

Figure 23b : Memory test input.

Figures 23a and 23b show the memory test. In Figure a we can see how the colour sequence is displayed on screen, the whole top screen displays each colour in the sequence, while Figure b, shows the user's sequence input. Once the colour sequence is finished, the top screen becomes black, and an image with all the possible colours appears in the bottom screen.

Figure 24 : Skills' test results.

Figure 24 shows the output of the skills test once is finished. The test will display on screen all the output for each application, so the results can be copied all at once and processed to obtain the Player Handicap and its GamerShape.

4.2 expected results

Studies made to test how videogames can improve human skills (Roach 2003; Delahunty 2007; H 2007), have come to similar conclusions that people that play videogames improve their cognitive skills, reaction time and also improve their pheripherical vision.

The implemented skills test will be performed by both gamers and non-gamers and we can expect different results from both groups. From the skills that will be tested (precision, timing, controller, puzzle and dedication), there are some that will likely be higher in the gamers group.

The controller ability is most likely to be higher in the gamers group because this group is used to playing videogames. With the controller ability there is a remark that has to be made, as this test is only performed in one system, some players could not be used to playing with this system therefore the final value could be different if performed on another system. Players that usually play videogames on the Nintendo DS system will likely have a better performance in the controller test.

The other abilities expected to be performed better by the group of gamers are the timing (that comes from a reaction time test) and the puzzle solving. As the studies mentioned before, the reaction time improves the more hours a person plays videogames. The puzzle solving test is based on puzzles that have similarities with puzzles in real videogames, but it doesn't mean that gamers will perform better as there can be people that don't play videogames that are used to it or like to solve puzzles.

The precision ability doesn't have to be necessarily better in the gamers group as it just involves touching targets on the touch screen and the dedication also depends on each person, if they like challenge or not.

4.3 skills' test results

The test has been finished and debugged and we have asked people to take the test in order to have enough results to evaluate them. The test has been sent to people that own a Nintendo DS. We have asked them to provide us with the results with the condition that the test could only be taken once or the results that were sent back would be from the first take.

Some of the results have been discarded as it was clear they were not possible (maximum results in all the challenges or impossible solving times for both puzzles). To discuss the obtained results in this section it has been decided to only use the results taken from people that did the test infront of us so we can be sure the final results are valid.

The results presented in this section come from 50 different people, divided into 25 people that play videogames usually and 25 people who don't play videogames. It has been decided to use the same amount of people from both groups so we can compare the average GamerShape from each group.

These 50 people are also compose of 33 men and 17 women, although the comparision of the average GamerShape of each group may not be precise, it can be interesting to compare both GamerShapes.

The first GamerShape to be discussed is the one obtained when taking the average Player Handicap values from the 50 people (see Figures 25 and 26).

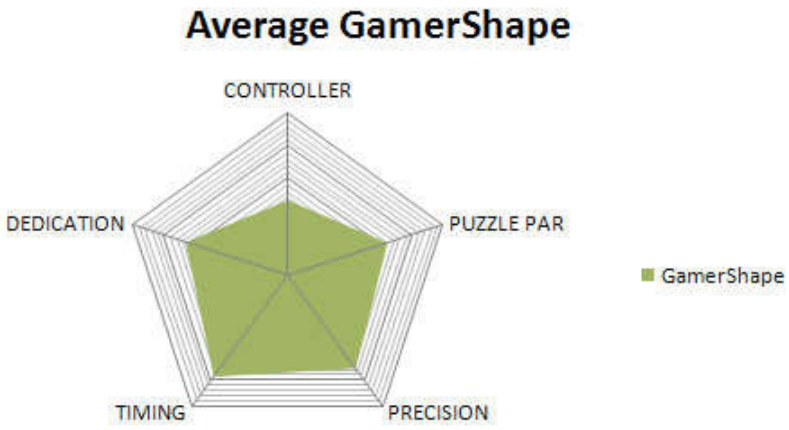

Figure 25 : Average GamerShape.

AVERAGE PLAYER HANDICAP				
Controller	**Puzzle**	**Precision**	**Timing**	**Dedication**
46	64	71	78	66

Figure 26 : Average Player Handicap.

As we can see it's an equilibrated GamerShape with the controller skill having the smaller value. This is expected because the non-gamer group are not used to playing videogames and the gamer group doesn't have to be used to playing with the Nintendo DS buttons, that is why the combo making values can be low.

The other skills have an average value in the range of 66-75% aproximately of the maximum value, which can be considered as good values but with enough room for improvement. The best skill is timing, with a value of 78. We will need to see if this high value comes from the gamers or non-gamers group.

Now we will compare the gamers group's average GamerShape with the non-gamers one. (Figures 27-30).

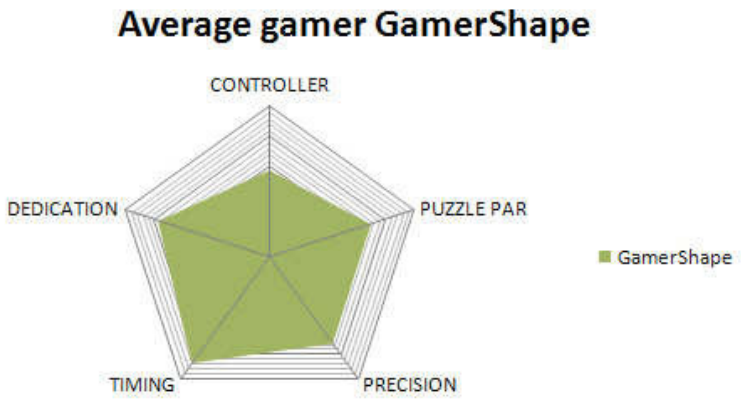

Figure 27 : Average gamer GamerShape.

AVERAGE GAMER PLAYER HANDICAP				
Controller	**Puzzle**	**Precision**	**Timing**	**Dedication**
58	71	71	87	77

Figure 28 : Average gamer Player Handicap.

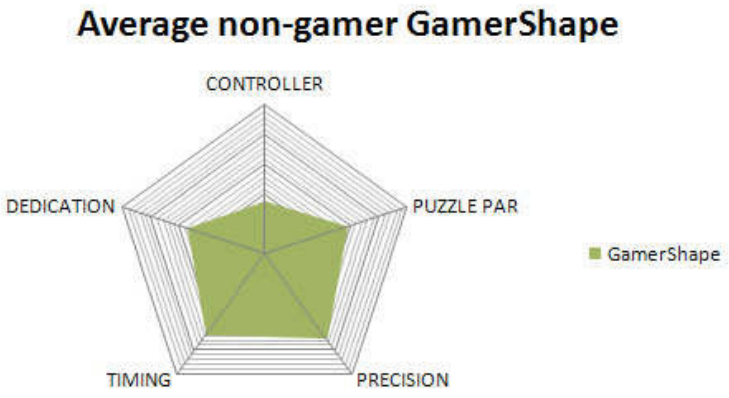

Figure 29 : Average non-gamer GamerShape.

AVERAGE NON-GAMER PLAYER HANDICAP				
Controller	**Puzzle**	**Precision**	**Timing**	**Dedication**
35	58	71	68	55

Figure 30 : Average non-gamer Player Handicap.

The figures in the last page show clearly how the gamer's GamerShape is significantly better than the non-gamers one. Looking at their Player Handicaps, the timing value is specially high for the gamers' group, as we could expect thanks to the studies exposed in section 4.2. All the values are higher in the gamers group except the precision value where both groups achieved the same value. This reinforces the hypothesis on how touching the screen with the stylus wasn't related to playing videogames, as everybody has used a touch screen before.

Gamers were also better at solving the puzzles and remembering colours and used less hints than non-gamers. With these results we can say that non-gamers are less likely to enjoy being stuck while playing and that they play more to have fun. Figure 30 also shows how non-gamers can find it really complicated to play videogames as they are not used to the controllers. Having to play a game with 8 different buttons can be a serious problem making people not enjoy the experience.

Non-gamers could really take advantage of GamePlay morphing with these values, because the games they play would be at their skill's level, for example in precision they wouldn't need an easier game, but both puzzles and controller would be less complicated for them. Also with GPM, if one person of each group played against each other, we could balance the gameplay so it would be an interesting game. If GPM is not applied, the non gamer wouldn't have a chance at winning.

Next we will compare the average men and women's GamerShape, starting with the groups of 33 men and 17 women (Figures 31-34).

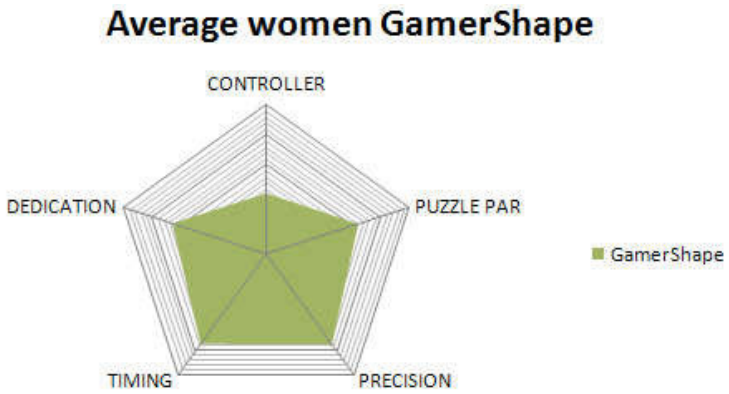

Figure 31 : Average women GamerShape.

AVERAGE WOMEN PLAYER HANDICAP				
Controller	**Puzzle**	**Precision**	**Timing**	**Dedication**
41	65	75	74	66

Figure 32 : Average women Player Handicap.

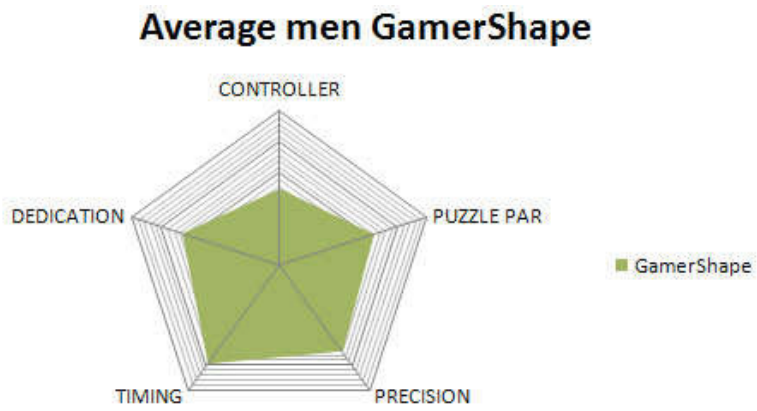

Figure 33 : Average men GamerShape.

AVERAGE MEN PLAYER HANDICAP				
Controller	**Puzzle**	**Precision**	**Timing**	**Dedication**
49	64	69	79	66

Figure 34 : Average men Player Handicap.

We can extract some interesting conclusions from these two GamerShapes, although the groups are not balanced. The 17 women show better precision (6 points above) and they are slighty faster at solving the puzzles, while we can see the dedication doesn't have anything to do with genres as both groups have the same value. The men obtain better results at timing and controller, something that could be expected.

Finally we will show some interesting GamerShapes obtained from the test, and explain how they could take advantge of GPM (see Figures 35-40)

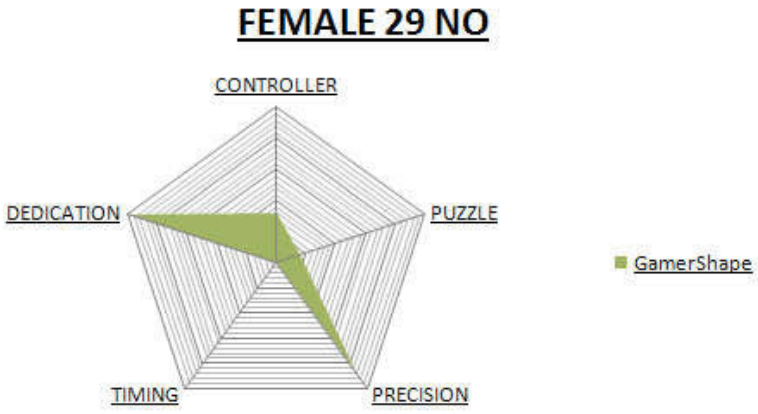

Figure 35 : 29 years old non-gamer female GamerShape.

Figure 35 shows the obtained GamerShape from a 29 year old female who doesn't play videogames. We can clearly see a really good skill precision, and a total dedication, while the other skills are quite poor. With GPM, this person would find a higher difficulty for her skills level, and parts of the game that involve timing or puzzles would be easier compared to more experienced player. The games that need precision, would have a higher difficulty as her level is high.

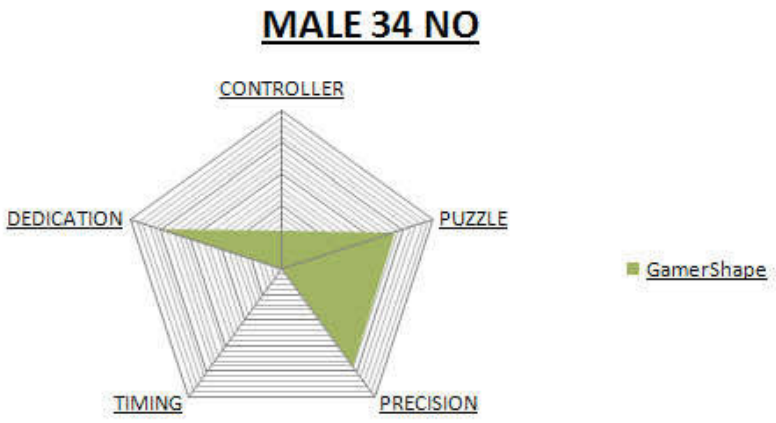

Figure 36 : 34 years old non-gamer male GamerShape.

Figure 36 shows us a similar case with the difference being that this user has a high puzzle skills level. If this user plays an adventure game, the puzzle parts would be difficult, while the fight parts that involve controller skills would be easier. We could also reduce the number of needed buttons to play while the person gets used to playing videogames.

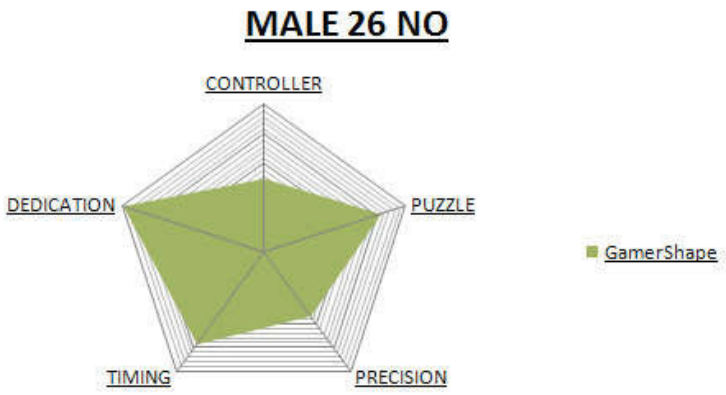

Figure 37 : 26 years old non-gamer male GamerShape.

The GamerShape in Figure 37 shows us a completely different case, a non-gamer with actually good skills for playing videogames. When this person plays videogames, they wouldn't need to be as easier as expected. Precision parts of the gameplay would need to lower their difficulty,

while puzzles and quick events would be more difficult. Also, seeing the dedication, we know this person likes challenges, so GPM would increase the difficulty of the game, in order to provide a personalized experience to this player.

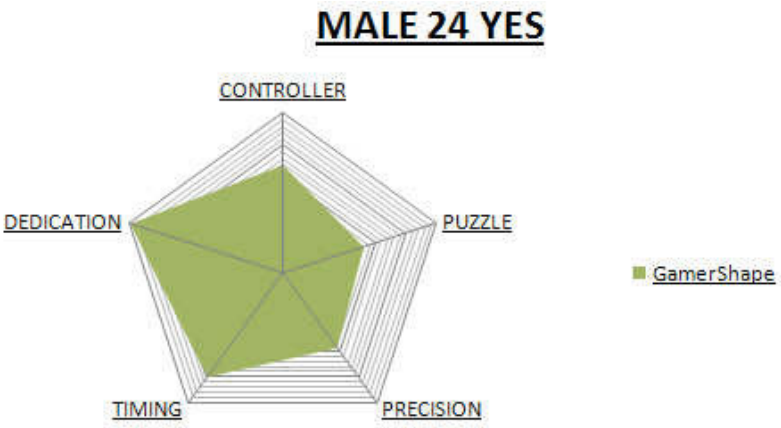

Figure 38 : 24 years old gamer male GamerShape.

Comparing the GamerShape in Figure 38 with the one in Figure 37, we can see a good difference between a gamer user and a non-gamer one. Obviously the gamer will have better controller skills, but in this case, the non-gamer is better at solving puzzles. For the GamerShape in Figure 38, fighting games will have a higher difficulty than puzzle games, and advendure games would adjust each part of the gameplay to the required levels, matching the player's level in each ability. We can see in this GamerShape how a gamer doesn't have to be good at all the abilities, so it doesn't make sense to raise or decrease the difficulty of all the parameters to the same level.

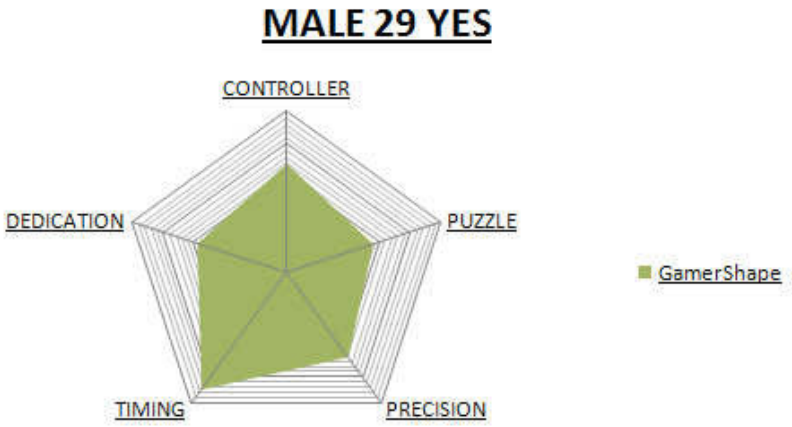

Figure 39 : 29 years old gamer male GamerShape.

The user with the GamerShape in Figure 39 have similar skills to the user who's GamerShape is in Figure 38. The difference in this case is the dedication, being half the dedication in Figure 39. This gamer doesn't really enjoy getting stuck at playing videogames, so the difficulty would be better adjusted to his level, so he could finish the game without complications, enjoying the game more. For this GamerShape, quick events and other timing gameplay parts would have a higher difficulty than the rest of the game.

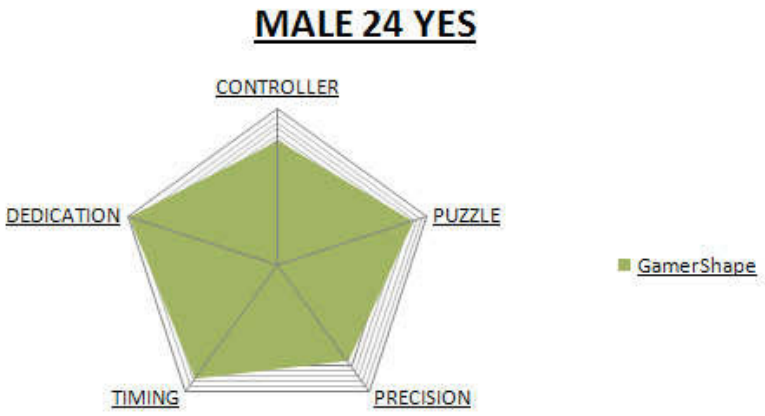

Figure 40 : 24 years old gamer male GamerShape.

Finally, the gamer who's GamerShape is in Figure 40 would have a high difficulty in all the parameters of the game, making the precision games slightly easier. This Gamershape is the best one from all the obtained results.

5. CONCLUSIONS & IMPROVEMENTS

In this project we have introduced the current state of dynamic difficulty adjustment and the different approaches the videogame industry has done to solve this problem. We also have exposed the increasing concern the industry has about appealing to the widest possible audience when releasing a new game.

We have exposed some examples of why GamePlay morphing can be beneficial for the videogame industry by balancing videogames difficulty based on differents parts of gameplay, instead of raising or decreasing all the parameters of the game at the same time.

In order to balance videogames' gameplay and dynamically adjust the difficulty of these games, we have defined the necessary skills required to play videogames and provided a graphical way to show these skills to the console users, the GamerShape and its Player Handicap.

We also have defined techniques to evaluate these skills, designing the ability test that will be used to create each user's GamerShape from the obtained values of this test. An explanation and justification of the output values from this test and the created formulas to obtain the Player Handicap and consequently the GamerShape has also been made.

The ability test has been implemented for the Nintendo DS platform explaining all the steps needed to set up a developing environment. For each application we have exposed the most important feature that has been used and the source code needed to implement this feature.

In order to anticipate which results would be good to obtain from the ability test, we have done a research on the different projects that study how videogames improve human skills. Thanks to this research we have exposed the expected results from this project.

Once the test was implemented, we have asked different people to take it so we could extract the required results for this project. We have presented the obtained results showing different Gamershapes from both groups of people and from a single person.

The results of the ability test match up with the expected results, proving that GamePlay morphing can be a valid solution to balance difficulty in today's videogames.

The hardest part of this project has been the design of the ability test, in order to be able to create a valid test that would correctly evaluate the required skills. Implementing the test hasn't been easy, the most problems occurred when trying to display the sprites on screen.

As for future improvements that could be made for this project, we could implement the ability test for other console systems and find the best way to update the GamerShape of each user:

- Regarding the controller ability, if the test is implemented on home game consoles, their controller implements analogue sticks. We should update the combo application to include button sequences with the analogue sticks. Musical games can be seen as a combination of timing and controller ability as they ask to press a sequence of buttons at a correct rhythm. We could use these musical games to track both timing and controller abilities.
- For the precision skill we should also add the analogue sticks to our test. One way to observe the precision could be making the user change the direction of the camera with the stick and make them look at an object. Tracking the time needed and the number of direction changes achieved to correctly point the object. The wii remote controller includes a pointer that allows people to point directly to the screen, so we should change the target test using the pointer instead of the stylus. In the case of analogue controllers, the stick would be used to point the different targets.
- It's obvious that making each player perform the same test over and over again to update their Player Handicap is not a wise choice. Most people would remember the answers and it would be tedious to perform the same task everytime. There are two options to update the player's handicap and we can use them separately or combine both of them. First option would be creating a new test every two or three months to update the values, these tests wouldn't have to be as big as the first test that each user does when creating a profile, just adding a new challenge for each ability would be enough to see if it has improved. As most videogames give punctuations when the player has finished a game, depending on the difficulty and the scored value, we can easily know if the skill's level is good or bad. If the punctuation is high, we could update the player handicap (a reasonable value) by increasing it, or decrease it if the punctuation is bad.

APPENDIX A

This appendix contains a brief description of each header file that is included on the libnds library. Each file contains the needed definitions to implement some high level functions.

background.h	Nds background defines and functionality
bios.h	Nintendo DS Bios functions
boxtest.h	Box Test Functions
cache.h	ARM9 cache control functions
console.h	Nds stdio support
debug.h	Currently only used to send debug messages to NO$GBA debug window
decompress.h	Wraps the bios decompress functionality into something a bit easier to deal with
dma.h	Wrapper functions for direct memory access hardware
dynamicArray.h	A dynamically resizing array for general use
fifocommon.h	Low level FIFO API
fifomessages.h	Standard fifo messages utilized by libnds
image.h	An image abstraction for working with image data
arm7/input.h	NDS input support
arm9/input.h	NDS input support
interrupts.h	Nds interrupt support
keyboard.h	Nds stdio keyboard integration
linkedlist.h	A simple doubly linked, unsorted list implementation
math.h	Hardware coprocessor instructions
memory.h	Defines for many of the regions of memory on the DS as well as a few control functions for memory bus access
nds.h	Master include file for nds applications
ndsmotion.h	Interface code for the ds motion card, ds motion pak, MK6
ndstypes.h	Custom types employed by libnds
pcx.h	A simple 256 color pcx file loader
postest.h	Position Test Functions.

rumble.h	Nds rumble option pak support
sassert.h	Simple assertion with a message conplies to nop if NDEBUG is defined
sound.h	A simple sound playback library for the DS. Provides functionality for starting and stopping sound effects from the ARM9 side as well as access to PSG and noise hardware. Maxmod should be used in most music and sound effect situations
sprite.h	Nds sprite functionality
system.h	NDS hardware definitions
timers.h	Contains defines and macros for ARM7 and ARM9 timer operation. It also contains a simplified API for timer use including the ability to chain multiple callbacks to a single hardware timer
trig_lut.h	Fixed point trig functions. Angle can be in the range of -32768 to 32767. There are 32768 degrees in the unit circle used by nds. To convert between standard degrees (360 per circle):
video.h	Basic defnitions for controlling the video hardware
videoGL.h	OpenGL (ish) interface to DS 3D hardware

Table A1 : libnds files and brief description.

APPENDIX B

To start writing code for the Nintendo DS system there are some basic topics such as input handling and graphic modes that will be discussed in this section.

B.1 input handling

To be able to interact with the user we will need to know how to recognise the input of the system, either buttons or touch screen.

Libnds defines a register named REG_KEYINPUT (see Figure B1) where the state of each button will be stored in order to provide an easy way to check the current state of each button.

E F	D	C	B	A	9	8	7	6	5	4	3	2	1	0
-	-	Touch	Y	X	L	R	Down	Up	Left	Right	Start	Select	B	A

Figure B1 : Register in memory that holds the current state all keys

This register will contain a 1 in state of rest, and 0 if the key is pressed. Libnds provides top-level functions to help checking button states. The first function is named scanKeys() and has to be called before any key press processing is done. It scans the REG_KEYINPUT register and prepares the data to be used for key state comparison functions. These three functions are keysHeld(), keysDown(), and keysUp(). KeysHeld() function is used to scan which keys are down and have down for a while, keysDown() and keyUp() are used to scan for the keys that have been pressed down or released recently.

Libnds also defines each bit of the register as KEY_A for the A button, KEY_B for the B button and so on. Code B1 and Code B2 gives an exameple of key handling.

```
scanKeys();
if(keysDown() & KEY_A)
{
// If A button have been pressed recently
}
if(keysHeld() & KEY_A)
{
// If A button have been pressed for a while
}
if(keysUp() & KEY_A)
{
// If A button have been released recently
}
```

Code B1 : Example of key handling.

```
while(!(keysDown & KEY_A))
{
scankeys();
}
//Wait until key A is pressed
```

Code B2 : Another example of key handling.

Touch screen's input is a bit more complicated than the button's input as it is not just a bit which status is either on or off. The touch screen have to store the coordinates of the touch so the developer can make interactive objects to the user. Libnds has a built-in struct named touchPosition that holds px and py variables, and integer value between 0 and 256 for px and an interger value between 0 and 192 for py.

The developer will only have to call the function touchRead(touchPosition) to store the new coordinates in the struct (see Code B3).

```
// Create var touchXY type touchPosition
touchPosition touchXY;

scanKeys();
//if the touch screen have been pressed
if(keysDown() & KEY_TOUCH)
{
//read new coordinates
touchRead(touchXY);
}
//New coordinates stored in touchXY.px and
//touchXY.py
```

Code B3 : Example of how to obtain coordinates of the touch screen.

These are just some basic examples to help understand libnds input handling, combining all these different functions developers can create some complicated user interaction.

B.2 graphic modes and backgrounds

Backgrounds are a basic feature on videogames where the user is in control of a character that moves on some type of background. As the Nintendo DS have two screens, there exists two different 2D graphic engines, named main engine and sub engine. Each graphic engine will be assigned to any of the bottom or top screens, leaving it up to the developer's choice (see Code B4).

```
lcdMainOnTop(); // main engine renders graphics in the top screen
lcdMainOnBottom(); // main engine renders graphics on bottom screen
lcdSwap(); //swap the screens the engines were rendering to
```

Code B4 : Existing functions to assign top and bottom screens to main and sub engines.

The main engine has 8 graphic modes avalaible while the sub engine has 6 graphics modes to be used (see Table B1). There are some differences between both engines, the main engine is the only able to render 3D graphics in the background 0 of each mode, and it also allows developers to work with the framebuffer mode.

Main 2D Engine				
Mode	BG0	BG1	BG2	BG3
Mode 0	Text / 3D	Text	Text	Text
Mode 1	Text / 3D	Text	Text	Rotation
Mode 2	Text / 3D	Text	Rotation	Rotation
Mode 3	Text / 3D	Text	Text	Extended
Mode 4	Text / 3D	Text	Rotation	Extended
Mode 5	Text / 3D	Text	Extended	Extended
Mode 6	3D	-	Large Bitmap	-
Frame Buffer	Direct VRAM display as a bitmap			
Sub 2D Engine				
Mode	BG0	BG1	BG2	BG3
Mode 0	Text	Text	Text	Text
Mode 1	Text	Text	Text	Rotation
Mode 2	Text	Text	Rotation	Rotation
Mode 3	Text	Text	Text	Extended
Mode 4	Text	Text	Rotation	Extended
Mode 5	Text	Text	Extended	Extended

Table B1 : Graphic modes.

Each graphic mode has 4 different background layers that are rendered separately where each layer has a priority (0 highes priority and 3 lowest priority) to know which layers will be rendered above the others.

FrameBuffer : Allows developers to manipulate every pixel on it's own. There's a region of memory that gets printed on screen directly. It has 256*192 pixels to manipulate and the colour has to be set using RGB15 (5 bits for red, 5 bits for green and 5 bits for blue).

3D : Background used to render 3D graphics, using videoGL (libnds library similar to OpenGL).

Text: Text backgrounds divide the backgrounds into blocks of 8x8 pixels named Tiles. Supports up to 512x512 pixels backgrounds.

Rotation: Similar to text backgrounds but they allow rotating and scaling these backgrounds.

Extended : Similar to rotation backgrounds allowing developers to scroll and zoom the backgrounds.

LargeBitmap : Allows developers to use 1024 x 512 pixels backgrounds.

Once the developer decides which graphic mode needs to be used for their application, they will also need to know which memory bank to store the graphics in. There are 9 vram banks in total to be used. Each one of them can be used for different purposes (see Table B2).

BANK	CONTROL REGISTER	SIZE	MODES
VRAM VRAM_A	VRAM_A_CR	128 KiB	VRAM_A_LCD VRAM_A_MAIN_BG_0x6000000 = VRAM_A_MAIN_BG VRAM_A_MAIN_BG_0x6020000 VRAM_A_MAIN_BG_0x6040000 VRAM_A_MAIN_BG_0x6060000 VRAM_A_MAIN_SPRITE VRAM_A_TEXTURE_SLOT0 = VRAM_A_TEXTURE VRAM_A_TEXTURE_SLOT1 VRAM_A_TEXTURE_SLOT2 VRAM_A_TEXTURE_SLOT3
VRAM_B	VRAM_B_CR	128 KiB	VRAM_B_LCD VRAM_B_MAIN_BG_0x6000000 VRAM_B_MAIN_BG_0x6020000 = VRAM_B_MAIN_BG VRAM_B_MAIN_BG_0x6040000 VRAM_B_MAIN_BG_0x6060000 VRAM_B_MAIN_SPRITE VRAM_B_TEXTURE_SLOT0 VRAM_B_TEXTURE_SLOT1 = VRAM_B_TEXTURE VRAM_B_TEXTURE_SLOT2 VRAM_B_TEXTURE_SLOT3

VRAM_C	VRAM_C_CR	128 KiB	VRAM_C_LCD VRAM_C_MAIN_BG_0x6000000 VRAM_C_MAIN_BG_0x6020000 VRAM_C_MAIN_BG_0x6040000 = VRAM_C_MAIN_BG VRAM_C_MAIN_BG_0x6060000 VRAM_C_ARM7 VRAM_C_SUB_BG_0x6200000 = VRAM_C_SUB_BG VRAM_C_SUB_BG_0x6220000 VRAM_C_SUB_BG_0x6240000 VRAM_C_SUB_BG_0x6260000 VRAM_C_TEXTURE_SLOT0 VRAM_C_TEXTURE_SLOT1 VRAM_C_TEXTURE_SLOT2 = VRAM_C_TEXTURE VRAM_C_TEXTURE_SLOT3
VRAM_D	VRAM_D_CR	128 KiB	VRAM_D_LCD VRAM_D_MAIN_BG_0x6000000 VRAM_D_MAIN_BG_0x6020000 VRAM_D_MAIN_BG_0x6040000 VRAM_D_MAIN_BG_0x6060000 = VRAM_D_MAIN_BG VRAM_D_ARM7 VRAM_D_SUB_SPRITE VRAM_D_TEXTURE_SLOT0 VRAM_D_TEXTURE_SLOT1 VRAM_D_TEXTURE_SLOT2 VRAM_D_TEXTURE_SLOT3 = VRAM_D_TEXTURE
VRAM_E	VRAM_E_CR	64 KiB	VRAM_E_LCD VRAM_E_MAIN_BG VRAM_E_MAIN_SPRITE VRAM_E_TEX_PALETTE VRAM_E_BG_EXT_PALETTE VRAM_E_OBJ_EXT_PALETTE
VRAM_F	VRAM_F_CR	16 KiB	VRAM_F_LCD VRAM_F_MAIN_BG VRAM_F_MAIN_SPRITE VRAM_F_TEX_PALETTE VRAM_F_BG_EXT_PALETTE VRAM_F_OBJ_EXT_PALETTE
VRAM_G	VRAM_G_CR	16 KiB	VRAM_G_LCD VRAM_G_MAIN_BG VRAM_G_MAIN_SPRITE VRAM_G_TEX_PALETTE VRAM_G_BG_EXT_PALETTE VRAM_G_OBJ_EXT_PALETTE
VRAM_H	VRAM_H_CR	32 KiB	VRAM_H_LCD VRAM_H_SUB_BG VRAM_H_SUB_BG_EXT_PALETTE
VRAM_I	VRAM_I_CR	16 KiB	VRAM_I_LCD VRAM_I_SUB_BG VRAM_I_SUB_SPRITE VRAM_I_SUB_SPRITE_EXT_PALETTE

Table B2 : VRAM banks.

As we can see in Table B2 each memory bank can be used to store backgrounds, sprites, textures for the main or sub engine, depending on the bank. Libnds provides some macros to help configure graphic modes and vram banks.

videoSetMode & videoSetModeSub : used to determine the graphic mode used for each engine and which backgrounds or layers will be enabled (see Code B5).

```
//set video mode to mode 5 with background 3 enabled
     videoSetMode(MODE_5_2D | DISPLAY_BG3_ACTIVE);
//set video mode to mode 5 with both background 2 and 3 enabled
     videoSetModeSub(MODE_5_2D | DISPLAY_BG2_ACTIVE | DISPLAY_BG3_ACTIVE);
```

Code B5: How to set up a graphic mode.

vramSetBankX : macro used to determine which function will have the desired bank and which address will allocate the graphics (see Code B6).

```
   //map vram A to start of main sprite graphics memory
        vramSetBankA(VRAM_A_MAIN_SPRITE_0x06400000);
   //map vram B to start of main background graphics memory
        vramSetBankB(VRAM_B_MAIN_BG);
```

Code B6: How to set up banks to allocate graphics.

This section is just a glimpse of the functions avalaible to developers for configuring and using graphics and input on the DS. There are so many functions and macros defined by libnds to be explained in this project. The reason for this is to teach the basics on how to develop for the DS. For more information refer to libnds documentation(doxygen 2009) and dev-scene tutorials (Hull 2009).

BIBLIOGRAPHY & REFERENCES

Adam, E. (2002). "Balancing Games with Positive Feedback." Gamasutra (Electronic)

Adam, E. (2008). "Difficulty Modes and Dynamic Difficulty Adjustment " Gamasutra (Electronic).

Amero, J. (2008). "Introduction to Nintendo DS Programming".

Capcom. (2006). "Resident Evil Portal." from http://www.residentevil.com/.

David Thomas, K. O., Scott Steinberg (2007). "The videogame style guide and reference manual", Power Play Publishing.

Delahunty, J. (2007) "Video games improve Surgeons' skills."

dev-scene.com (2006). "Nintendo DS Memory Map", dev-scene. 736x996.

doxygen. (2009). "Libnds Documentation." Retrieved 12-01, 2010, from http://www.devkitpro.org/libnds/.

Entertainment, S. C. (2008). "Playstation Network Trophies." from http://www.us.playstation.com/go/trophies/.

H, N. (2007). "Benefits of playing videogames." Helium.

Hull, A. (2009). "NDS Tutorials." Retrieved 12-01, 2010, from http://dev-scene.com/NDS/Tutorials.

Hunicke, R. (2005). "The case for dynamic difficulty adjustment in games."

Hyman, P. (2007) "Microsoft's Achievement Points Yield 'Nerd Cred'." GameDaily.

Kosinski, R. J. (2009). "A Literature Review on Reaction Time Clemson University".

Maxmod. (2008). "Maxmod Programming Reference." Retrieved 12-01, 2010, from http://www.maxmod.org/ref/.

Roach, J. (2003). "Video Games Boost Visual Skills, Study Finds." National Geographic.

Robin Hunicke, V. C. (2004). "AI for Dynamic Difficulty Adjustment in Games."

Saltzman, M. (2009). "Nintendo confirms secret 'Help' feature". USA Today

Various. "Maxmod Sound System." Retrieved 12/20, 2009, from http://www.maxmod.org/.

Various. (2008). "Video game genres." Retrieved 03-01, 2010, from http://en.wikipedia.org/wiki/Video_game_genres.

Vijn, J. (2007). "Coranac ›› Projects." Retrieved 12/01, 2010, from http://www.coranac.com/projects/#grit.