UCL Université catholique de Louvain

École polytechnique de Louvain (EPL)

epl

Machine Learning for Smartphone Games

Dissertation presented by
Antoine VAN MALLEGHEM , Nicolas VAN WALLENDAEL

for obtaining the Master's degree in
Computer Science and Engineering

Supervisor(s)
Pierre DUPONT

Reader(s)
Pierre DUPONT, Vincent BRANDERS , Jérôme PAUL

Academic year 2015-2016

Contents

Chapter 4

The game

This master thesis obviously needs a game. The game choice was based on the reference article [57] for this thesis which was about a Rock-Paper-Scissor AI. This article explained that a truly random game of rock-paper-scissors would result in a statistical tie with each player winning, tying and losing one-third of the time like explained in section 3.6. However, since people are not truly random, a person's tendencies and patterns can be exploited by computers to take advantage.

To describe the chosen game, this chapter will depict what is a game and what makes a game something interesting for us to work on. Followed by the game description itself, how game theory analyses it and its implementation as a smartphone game.

4.1 A game ?

According to Salen & Zimmerman, the formal definition of a game [58] is:

> " A system in which players engage in an artificial conflict, defined by rules, that results in a quantifiable outcome."
>
> Salen & Zimmerman

Combining this statement with the definition given by Wolfgang Kramer in his book [59], a game must contain the following particularities :
Game rules which are the main part of the game and must successfully regulate the game flow and termination conditions.
Goals which are nothing else than the victory condition or the quantifiable outcome.
A game should also include a competition so that there are winners and losers or in other words an artificial conflict.
And one last but important component, chance so that the course of the game is as much as possible never the same.

These elements are the main components of a game. but to chose one, we personally have some more criteria;

1. The best player for the chosen game can't be a random player because the final agent must be smart and consequently not simply play randomly.
2. It mustn't be too difficult to understand and implement. The purpose of this thesis is the machine learning part and not the implementation of the game.

3. It must be playable on a smartphone meaning, intuitive enough to have a simple UI to be played on a mobile.

With all these conditions, we also wanted a game that was close to the original article game. Thus something simple but still interesting.

In the following sections, we will define our choice and describe it as a "game with rules" (according to [59]) and as a "Machine Learning problem" (see section 1.5).

4.2 "Cowboy game"

The "Cowboy game" (found on [60]), also called "Shotgun game" or "007", is similar to "rock-paper-scissors" (a.k.a. Roshambo). Indeed, it is basically a two players simultaneous turn based hand game but conversely to Roshambo, it's an iterative game. Therefore each action played at the previous round is reported, if it has any incidence, to the next round until the victory condition are met. Since the game is simultaneous, it also includes the concept of **chance** because the player can't guess the action chosen by his opponent.

The game has a simple set up. Initially, players have the same status. They are defined by the following characteristics :

- An activable shield with a solidity of 9
- a power of 0
- 1 health point

The **victory condition** is simple : the first player reaching no health point dies and consequently looses the game. The other player, therefore, automatically wins so the game is **competitive**. If both player dies simultaneous, which is possible, a tie is reached. At the beginning, both players have the same chance/propensity to win.

At each iteration of the game, the player can choose between 3 actions (if available), `Reload`, `Shield` or `Shoot` to modify his or its opponent characteristics. Simply speaking the ultimate goal is to `Shoot` directly toward the opponent health point. But to be able to `Shoot`, a player must `Reload` and a player can protect itself from this attack by activating his `Shield` for one turn.

A more precise description of the actions is given here after :

1. Reload : increase your power by 1. The maximum value for this attribute is 10.
2. Shoot : the power of the shoot can be 1,3,5 or 10 but the player must have at least a power of this amounts. After a shoot, the power of the player decreases by the power of the shoot. Each direct hit regardless of the power (without the opponent using the action `Shield`) is deadly.
3. Shield : used to block a possible shoot of the opponent. To be able to block a `Shoot`, the solidity of the shield must be superior or equal to the shooting power. As a result the shield is decreased by the power of the `Shoot`. If the opponent plays `Shoot` but the solidity is strictly less than the power, the bullet hits the health point of the player and he therefore dies/loses. If `Shield` is played versus another action than `Shoot`, nothing happens for this player.

Consequently, a player loses when the opponent shoots while he reloads or when his shield isn't sufficient. Indeed each player has only 1 health point and a `Shoot` has a minimum value of one. If both players shoot at the same time/iteration, the game reach a tie.

If the **game rules** are not clear enough, a summary table can be found in appendices at Table 14.1.

4.3 Cowboy game as a ML problem

As expressed in section 1.5, let's first define the three main notions and observe how they fit the main definition.

In our case, the Task is to create an agent who plays against a human and can learn how to win at the "Cowboy game". It means :

- Learns the good/bad combination of actions (for example, the action "shield" at the beginning of a game is useless because of the null initial value of the player's power)
- Adapt its strategy when it looses or wins
- Finally find an optimal strategy, if it exist.

Ultimately the role of the Task is, at each round, to output the next action the agent should play to maximize its chance to win the game against its human opponent with some Experience if any (from bots or reals humans). This Experience could be acquired either by observing games or by playing against the agent or both accrued.

But how would the quality/performance of this agent be assessed ?
A good agent should achieve a certain win rate. Against bots this win rate should be (much) higher than 50% since by definition they are predictable, whereas against humans who tends to play more intelligently, a win rate near 50% is acceptable. Win rate is computed as follow : +1 for every win, +0 for loses and tie are excluded (see equation 4.1).

$$\text{win rate} = \frac{1 \times wins + 0 \times loses}{\#games - \#ties} \tag{4.1}$$

Indeed, if we include tie as +0.5 for example it has a tendency to smooth to 50% scores which is bad for comparison. It is more interesting to separate win/lose score and tie score to have more granularity. But this has a drawback, matches with a high tie rate can't be reliable due to the reduced (win/lose) sample size used to compute the score.

It's also important to consider on how much game this rate is computed. A player should not play 1000 games such that the agent learns enough to be efficient. As by personal experience a fair amount of game to play is near 80-100 which is equivalent to 10-15 minute game time. This number reflects two aspects. Firstly, less than a quarter of an hour play time seems acceptable. Secondly and maybe more importantly, the game has a high tie rate, approximately 30-40%, and the (bad/good) luck factor must not be neglected. For these two last reasons a number of game below 50, is highly influenced by these factor and may not be representative. In the experience we will present scores at 50, 100 and 500 games to show short terms tendencies that are highly variable in general (at 50), the main score at 100 games and the global tendency at 500.

The game length is also an issue. A classical game length between humans is near 10-15 moves per game. Agents even against bots that tends to have long game length compare to other (with same win rate) should be considered less efficient. Indeed either they don't adapt fast enough or don't really know which is the "good" action to play which could mean that the agent as fallen in a local optimum (can happen against bots for example).

Another significant factor is the capability of the agent to adapt himself to various kind of player and this the most efficiently as possible. The goal of the agent is not to be the perfect hard counter of a specific agent but rather to be able to face various opponent equally. Therefore the mean victory is more important than "local" very high scores. Most of these factors which makes the performance measure can't be really rated with a score and therefore are a more a personal evaluation.

4.4 Game Theory applied to Cowboy

By using the game theory defined by Chapter 3, the "Cowboy Game" has the two common attributes of being a two-person competitive zero sum and non co-operative stochastic game. Indeed, as for Roshambo, zero sum implies that if one player wins the other loses with ties admitted and the non co-operative part comes from the same deduction. The game is also stochastic since it is an iterative game with actions influencing one's utility.

To define the game we also have that each player knows what all the players did in the previous iterations, but not what they're doing in the current iteration especially due to the simultaneity of the game. Thus, we have an imperfect-information game with perfect recall.

Now that the game is defined through his main characteristics, let's study its equilibrium and optimal strategies. To make it easier, instead of analyzing the entire game at once, we will only consider 4 typical game states which can be repeated at each iteration of the same round :

1. no player has power
2. both player don't have have power sufficient to bypass the opponent shield
3. one player has a power sufficient to bypass the opponent shield
4. both player have a power sufficient to bypass the opponent shield

For each case, the game theory concept presented in the previous chapter can be applied. To study each state, the expected utility (EU) of each action compared to the other has to be determined. These will be resumed in Table 4.1 to 4.4.

4.4.1 State 1

The first state is equivalent to the beginning of the game when no player has ever `Reload'ed`. Table 4.1 resume the EU. Indeed if both players `Reload`, it's good for both because they unlock their third action which is desirable. `Reload'ing` while the opponent `Shield` is even better because you unlock the `Shoot` action while the opponent stays in the same state and therefore yield a better score. Both players `Shield'ing` is advantageous for no one.
From these observations, we can output that every pair of actions which contains `Reload` are a Pareto equilibrium. Indeed a player can't switch from one strategy to another to increase his

utility without decreasing someone's else.
Now the Nash equilibrium of this game state and optimum strategy for both players is given by both playing the pure strategy `Reload`. Indeed it is always more interesting to play `Reload` than `Shield` in any case. This respect the equilibrium since no player can benefit by changing strategies while the other players keep theirs unchanged.

	`Reload`	`Shield`
`Reload`	1,1	2,0
`Shield`	0,2	0,0

Table 4.1: No player has realoaded yet.

4.4.2 State 2

The second state is a bit more complex. Now both player can `Shoot` but not kill their opponent when they `Shield`. Therefore `Shoot` on a `Shield` is bad but blocking is good obviously. Whereas `Reload'ing` on a `Shield` is the opposite. The last two new pairs of actions (`Shoot`, `Reload`) and (`Shoot`, `Shoot`) are the win and tie state.
This state and the following being most complex we will only study the Nash equilibrium. Since the game is a finite game it has an equilibrium and one of this three rules must be true for any equilibrium :

- Player 1 plays a pure strategy.
- Player 1 mixes between exactly two pure strategies.
- Player 1 mixes among all three pure strategies.

The same must be true for player 2.

The first statement is not possible since playing only one action is not viable and can be counter-played each time. Playing only two strategies is also not possible. If a player chooses for example to only play a mixed strategy between `Reload` and `Shoot`, the second player can obtain a strictly positive utility by only playing `Shoot`. (In red in Table 4.2). The same yields for a mixed strategy of `Shield` and `Shoot` (in green) and for `Shield` and `Reload` (in blue). For these reason only a combination of the three actions is viable.
The distribution of each strategy can be found with this system based on the table 4.2 :

$$\begin{pmatrix} U_{reload} \\ U_{shield} \\ U_{shoot} \end{pmatrix} = \begin{pmatrix} p_r & p_s & 1-p_r-p_s \end{pmatrix} \begin{pmatrix} 1 & 1 & -10 \\ -1 & 0 & 1 \\ 10 & -1 & 0 \end{pmatrix}$$

with $U_{reload} = U_{shield} = U_{shoot}$

This gives for both players (since the utility matrix is symmetrical),

$$\begin{cases} p_{reload} = 0.083 \\ p_{shield} = 0.827 \\ p_{shoot} = 0.09 \end{cases}$$

This shows that to reach an equilibrium the players plays a lot of shield, because the win utility score is high (10) and to reduce this influence `Shield` is played a lot. It is also a safe action to play due to the high value of one's player shield. Now, if we reduce for example win utility score from 10 to 1, which means that winning is equivalent to reloading for example, the relation gives,

$$\begin{cases} p_{reload} = 0.33 \\ p_{shield} = 0.22 \\ p_{shoot} = 0.45 \end{cases}$$

With these mixed strategy probabilities, `Shoot` is more efficient and this shows how everybody can tweak his strategy depending of his objective. In this game *a priori*, winning is still the best option and therefore yield more utility.

	`Reload`	`Shield`	`Shoot`
`Reload`	1,1	1,-1	-10,10
`Shield`	-1,1	0,0	1,-1
`Shoot`	10,-10	-1,1	0,0

Table 4.2: No player has sufficient power to overcome shield.

4.4.3 State 3

This state is much comparable to the previous one. Only this time, the second player can kill the first player over his shield, which means that `Reaload'ing` more isn't worth it anymore for player 2 and `Shoot` on a `Shield` is now good. Looking for a draw is also valuable for player 1 (but much less than a victory) since he may lose at any moment now.

As before, one equilibrium rule must be true. This time, `Shield` for player one gives always a negative utility which means it will never use it see blue at Table 4.3. For player, it's the action `Reload` (see red). The solution is therefore a mixed strategy of 2 actions this time. Like before the system can be written as,

$$\begin{pmatrix} U_{reload} \\ U_{shoot} \end{pmatrix} = \begin{pmatrix} p_{2s} & 1 - p_{2s} = p_{2kill} \end{pmatrix} \begin{pmatrix} 1 & -10 \\ -1 & 1 \end{pmatrix}$$

$$\text{with } U_{reload} = U_{shoot}$$

This gives for player 2,

$$\begin{cases} p_{2 \text{ shield}} = 0.84 \\ p_{2 \text{ killing shoot}} = 0.16 \end{cases}$$

Now for player 1, the system is,

$$\begin{pmatrix} U_{shield} \\ U_{killingshoot} \end{pmatrix} = \begin{pmatrix} p_{1r} & 1 - p_{1r} = p_{shoot} \end{pmatrix} \begin{pmatrix} -1 & 1 \\ 10 & 0 \end{pmatrix}$$

$$\text{with } U_{reload} = U_{shoot}$$

which gives,

$$\begin{cases} p_{1reload} = 0.08 \\ p_{1shoot} = 0.92 \end{cases}$$

This results show that for this Nash equilibrium player two will tend to protect himself a maximum to place supposedly the killing blow at the best moment, whereas player one wants to kill or trade kill his opponent with much more probabilities.

	`Reload`	`Shield`	`Killing Shoot`
`Reload`	1,0	1,-1	-10,10
`Shield`	-1,0	0,0	-10,10
`Shoot`	10,-10	-1,1	1,0

Table 4.3: One player has sufficient power to overcome shield.

4.4.4 State 4

This last state happens when both players can kill each other with the `Shoot` action regardless of the opponent action. In this case, only winning is worth utility and so does a tie (but less), cfr. Table 4.4.

In this relation every action is a Pareto optimum, since like previously a player can't switch from one strategy to another to increase his utility without decreasing someone's else. Nevertheless any combination of `Reload` and `Shield` are Pareto dominated by both playing `Shoot`. This last pair of action is also the Nash equilibrium. Indeed, once again no player can benefit by changing strategies while the other players keep theirs unchanged in this state.

The best action is simply to `Shoot` with the high probability that both players will die.

	Reload	Shield	Shoot
Reload	0,0	0,0	-10,10
Shield	0,0	0,0	-10,10
Shoot	10,-10	10,-10	1,1

Table 4.4: Both players have sufficient power to overcome shield.

4.4.5 Interpretation

All of the previous values are strictly theoretical meaning that the Nash equilibrium is only valid when both players follows it. Therefore playing exactly as theory suppose is not valid if both player doesn't follow the same rule. For example in Roshambo playing randomly each action one third of the time is the Nash equilibrium but if the opponent only plays says `Scissors`, this last statement doesn't yield anymore.
If one can forecast their opponent action accurately, they may be able to do much better than the Nash equilibrium strategy. Also why won't the other agents use their Nash equilibrium strategies ? Because they may be trying to forecast your actions too.
Thus this part is meant to give a real understanding of what the best reaction should be following theory but in practice this should not be followed blindly.

This conclude the little game analysis of the game. The next section will show how by using the game concept we created a design for everyone to enjoy it on smartphone.

4.5 The mobile application

One of the outcome of this thesis was the production of the game. You learned the basic concepts of the rules earlier. Now is the time to apply them to something more visual and entertaining, an Android application.
This section will summarize how the game was designed and with which tool and will also depict some of the limitations imposed by a gaming framework and smartphone specifically for our game.

4.5.1 The application

To develop this application, since it is first of all game based, we had to select between some known or maybe less know game framework/engine. Some example can be cited; First of all, Unity [1]. This engine gained popularity these last years mostly by "Indie developers", since the engine is free when the user revenue doesn't excess US$100,000. This framework uses C# and is multiplatform. Another mobile (/multiplatform) engine in JAVA is LibGDX[2]. This latter is totally open-source compared to Unity, but is, as for many open-source software, still in alpha/beta phase. These two software described alone the two larges families of engines, which principally holds in commercials and non-commercials ones.
For the sake of simplicity, stability and the quality of the editor, we choose Unity between many

[1] `unity3d.com`
[2] `libgdx.badlogicgames.com`

candidates (for not exhaustively citing all of them).

Unity development is based on a scene approach, meaning that each game screen (different levels, menus, ...) are individually loaded. We made use of two scenes, one for the game, strictly speaking, and another for the main menu. This latter is composed of 3 choices, launch the game, expose the game rules or the credits. A visualization of the menu can be found on Figure 4.1a. As for the game, the gameplay was implicitly dictated by the rules. Therefore we kinda just had to imagine a theme/concept that could encompass the rules. Since the game is designed as a versus fight between an IA and a human, we thought that an alien-learning invasion theme would be appropriate. With that we added a pinch of piracy, because, why not. This lead to the game design shown at Figure 4.1b. The pirate is played by the player and the alien/octopus is played by the AI.
Now, how are the actions portrayed. Conceptually, the shooting power of the players are represented by the unveiling of the canons of each ship (respectively 1,3,5 and 10 as shown on the side of the ships) and by the star shaped number on top of the screen (see Figure 4.1b). The shields are pictured as the whale back water stream for the player and a magnetic field force for the AI (see Figure 4.2a). Shields respective values are also written on the shield shaped number on top of the screen. Reload is designed as the floating "Ammo Crate" for the player and the symbol "Ammo" on the Alien screen. A `+1` is also shown on the top bar (see Figure 4.2b). Finally any shooting action is depict with specific projectiles for each power. To conclude this description of the game design, for readability sake, available actions are highlighted with dashed circles.

4.5.2 The constraints

This game was developed with the intent to demonstrate that we could apply machine learning to smartphones and that this hardware is not a limitation in itself. Nevertheless develop on smartphones poses technical constraints. First of all in our case we used a game engine, which imply that everything is designed with a gaming/visual background idea. When you want to apply machine learning you don't have standard matrix functions for example or libraries specifically design for this kind of job, but you still can code it yourself since the framework stays a C# compiler but a simplified one. Indeed you don't have access to all the function that you could use on a desktop version of .NET.

Another limitation is, since we have a game, a fraction of the resources of the device is used to render graphic on the screen, which decrease computation power and processor time allocation for a ML job. In our case, this has nearly no impact since as you will discover later in the chapter describing our final model (Chapter 12) we don't need much computation between each move of a player.

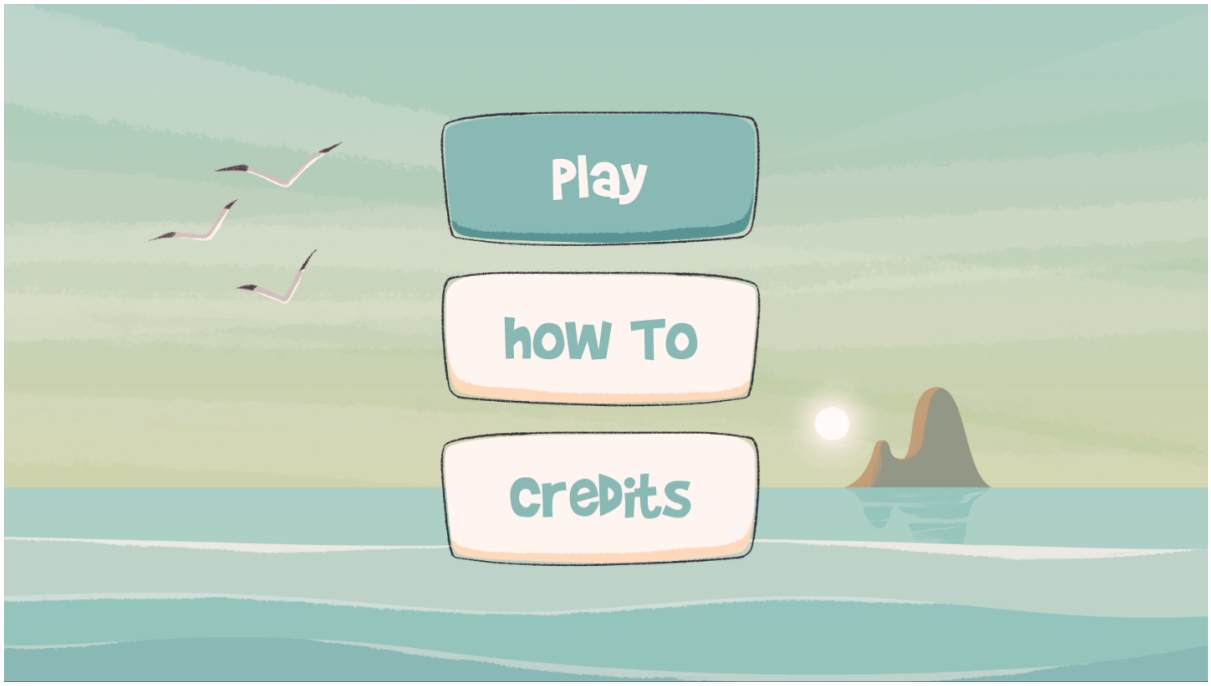
This last concept brings another constraint, responsiveness. You can't expect a player to wait forever between each move to let the AI think, this has a direct implication to the Agent model design.

One more constraint is "backgrounding". Indeed in modern mobile OS, we can have multiple apps opened at the same time. If the main app displayed is too resources consuming, the OS will kill backgrounded application to free up memory and free processor time. For example we made the following test on a Nexus 7 (Android 4 - 2012), launch a Skype video call during the launching of the game (pressing play) causes the OS to close Skype. On newer devices and OS

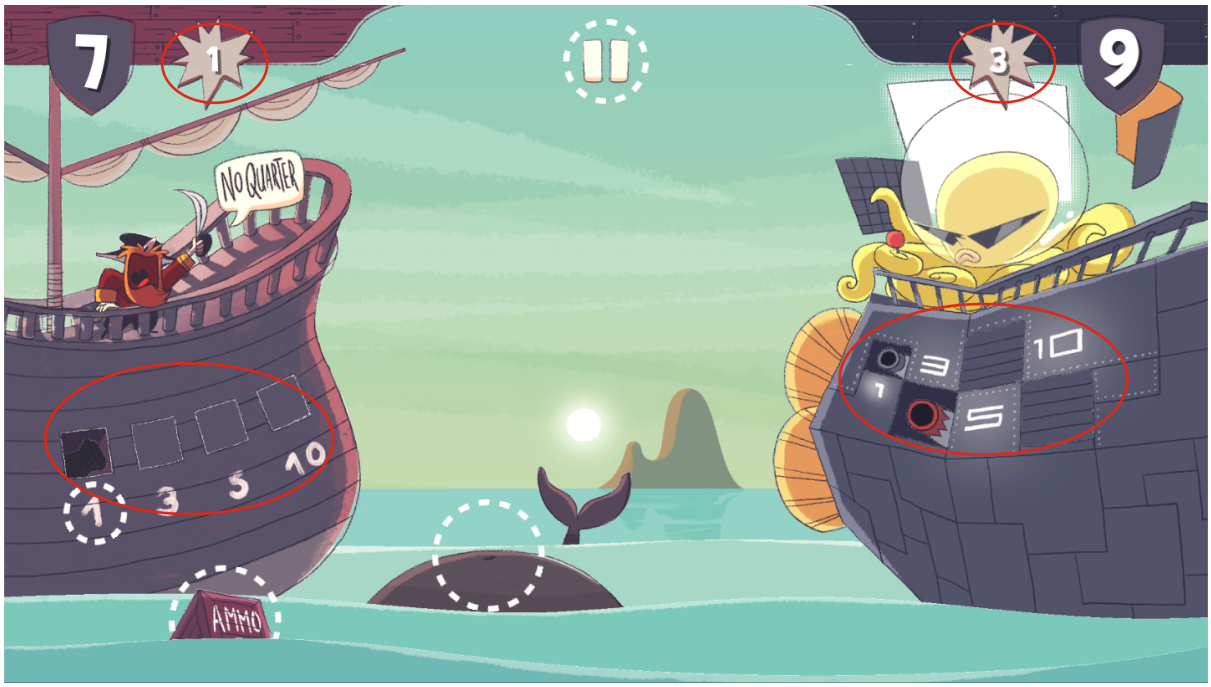
version (Samsung S6 edge - Android 6 - 2015) this doesn't happen anymore.

One last but somewhat important constraint is I/O. On mobile, it seems to be a blackhead. Reading files is less a problem whatsoever, but writing really is, recall section 2.2.2. For android device, many doesn't have dedicated storage and need external SD memory (with possibly bad performance) or have inside storage with really poor capabilities. Once again recent device seems to improve in this matter. With the same devices as mention before (and internal memory), 40mo of raw data takes 35 second on the older device and merely 3.5 on the other. For these reasons saving time are sometimes a bit long on older device. So was the loading part too, since it needed to load all the graphical part of the game and set up the AI, but thanks to some tricks (a main menu for example) we manage to "hide" some of these to make the application look faster.

This concludes some of the highlights of mobile development. This section demonstrated the mobile application, but what also interesting is to know and understand how the AI inside the game was designed. This is the subject of Part II. Therefore let's jump to the conclusion this part and find out what lies ahead.

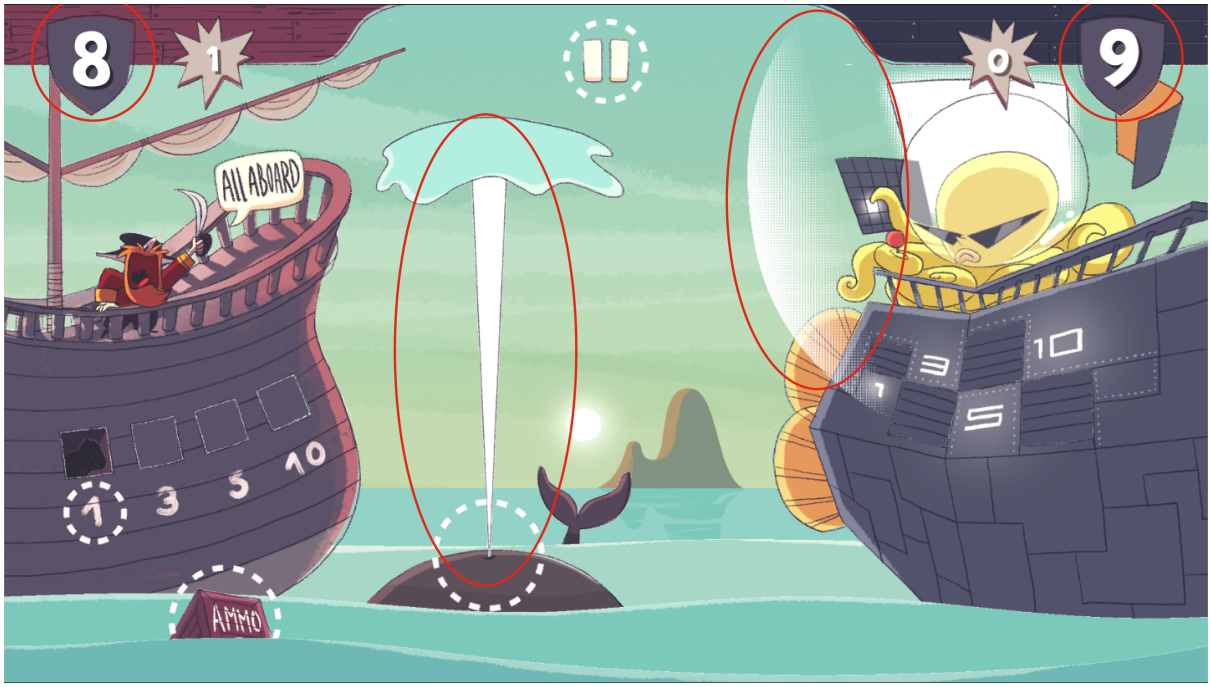


(a) Game : Main Menu

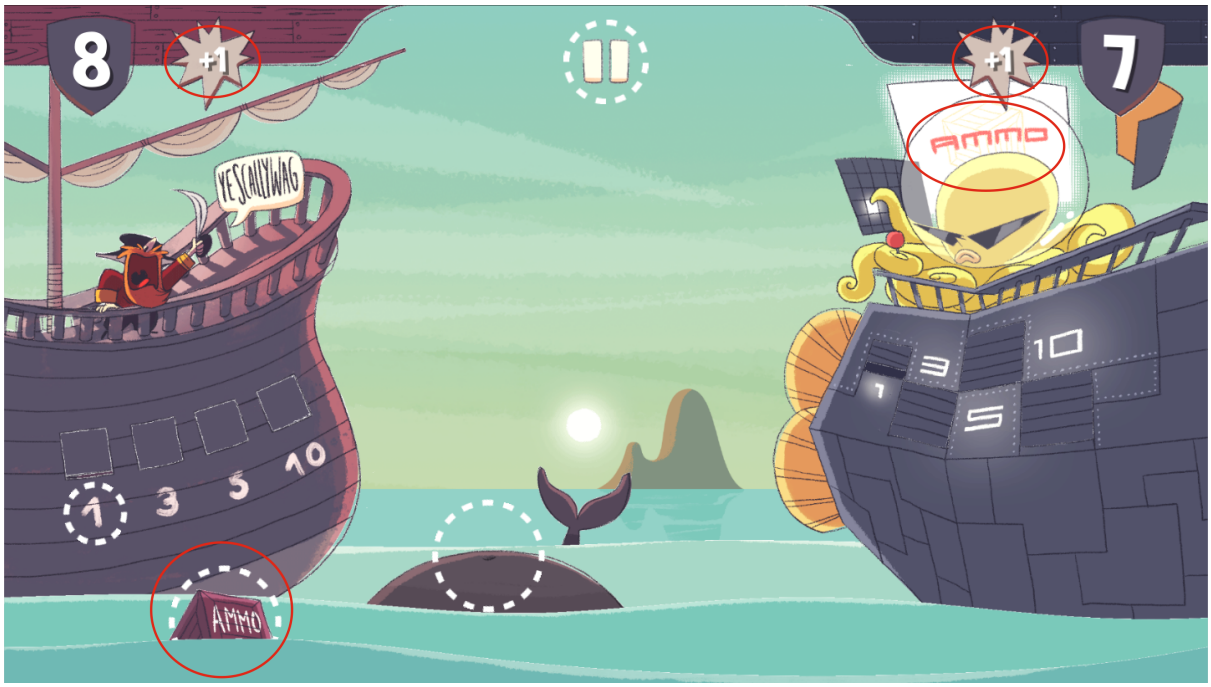


(b) Game : Shoot Power

Figure 4.1: Game Design (1)



(a) Game : Shields



(b) Game : Reload

Figure 4.2: Game Design (2)

Chapter 5

Conclusion

After this first part, the basis of the thesis has been set up. Indeed to be able to develop and analyze the concept of machine learning for smartphone game, we needed to put mobile devices, games and machine learning somehow into perspective.

From these it could be learned that creating a virtual challenger for the human brain has been a long and incredible journey that started with Claude Shannon's belief and continued till today and for many years to come.

Also we stumbled on a new kind of device, the smartphone, and discovered that although it is relatively knew, mobile device posses some interesting capabilities in term of computation. Indeed it is still unable to achieve jobs as easily as a computer but even with its restrictions which can be summarized as slow (hard) memory, limited heap space, restricted background processes, heat threshold, limited battery and the requirement of high optimization, they can still be useful in many cases. Phone also have advantages like, on device data access, latency, nomadism, internet access if necessary or simply said a reduce version of a laptop. The jobs that most successfully suits smartphones are those who benefit a maximum from only RAM access and limited resources from disk or swap. Meaning that either a program has basically these characteristic or has to be optimized to fit these requirement by reducing their complexity or adjusting the computation/loading to only fit in cache if possible. Successful examples like Google's or Qualcomm's already exist but point out the complexity of it as much as the rewards.

Last but not least, the thesis hopped in the game world through game theory. This latter was used to analyze thoroughly the game we choose which is as a reminded the *Cowboy Game*. The outcome of the analyses gave the optimal strategy to follow in 4 main stages of the game and helped to understand the quality of each move depending of the situation. This strategy although optimal is only meaningful if both players use it which is unlikely since the opponent and ourselves may try to forecast the enemy's moves.

This bring us to the continuity of this thesis, be able to predict and respond accurately to the opponent strategy. This motive introduce subtly machine learning as a means to study the adversary and use this knowledge to beat him. We already set the basis by applying the basic notion of ML to the game and clarified how an agent could achieve good performances. The next part will cover machine learning but more specifically a subpart of it, reinforcement learning. The ultimate goal being to create an agent able to compete against a human being. Once achieved this agent purpose is to be implemented in the smartphone game presented before and be playable by everyone equipped with an Android smartphone. This application is also a means

to prove that the method/model is applicable to mobile phone with the additional constraint that a graphical interface must be supported too.

Let's now dive in the second part which will this time apply machine learning to the *Cowboy Game* and thanks to an iterative process reach an agent able to compete against pre-programmed (semi-)static strategies and humans.