Applying Hidden Markov Model for Dynamic Game Balancing

Marcelo Zamith
Instituto Multidisciplinar
Universidade Federal Rural do Rio de Janeiro
Rio de Janeiro, Brazil
mzamith@ufrrj.br

Jose Ricardo da Silva Junior
Computing Department
Instituto Federal do Rio de Janeiro
Rio de Janeiro, Brazil
jose.junior@ifrj.edu.br

Esteban W. G. Clua
Instituto de Computação
Universidade Federal Fluminense
Niteroi, Brazil
esteban@ic.uff.br

Mark Joselli
Escola Politécnica
Pontifícia Universidade Católica do Paraná
Curitiba, Brazil
mark.joselli@pucpr.br

***Abstract*—In Artificial Intelligence (AI) field, Machine Learning (ML) techniques present an interesting approach for games, where it allows some sort of adaptation along the game session. This adaptation can make games more attractive, avoiding that Non-Player-Characters (NPC) present too easy or hard patterns during the game. In both cases, the player may be frustrated due to undesired experience. Although ML techniques are appealing to be used in games, some games characteristics are hard to model. Besides, there are techniques that require a wide variety of observations, which implies two hard barriers for game application: the first is the power processing to compute a huge amount of data in games, considering the real-time characteristic of this kind of application. The second threat is related to the vast majority of games' attributes that must be described in the model. This work proposes a novel approach using ML technique based on Hidden Markov Model (HMM) for game balancing process. HMM is a powerful technique which can be used to learn patterns based on a strong co-relational between an observation and an unknown variable (the hidden part). Our proposed approach learns the player's pattern based on temporal frame observation by co-relating his/her actions (movements) with game events (NPC destruction). The temporal frame observation approach allows the game to learn about player's pattern even if a different person plays it. After the learning process, the following step is to use the knowledge pattern to adapt the game according to the current player, which normally involves making the game harder for a certain period of time. During this time, another pattern may arise, subjected to be learned. In order to validate the presented approach, a Space Invaders clone has been built, allowing to observe that 54% of participants had more fun while playing it with ML activated in relation to a base version that did not take into account dynamic difficult balancing.**

***Index Terms*—artificial intelligence, machine learning, hidden markov models, games**

I. Introduction

Balancing a digital game is one of the most challenging and time consuming task during game development. Due to the subjectivity attached to this process, the game difficult may be pleasant for some players while unpleasant for others due to each individual perception. Besides, there is no mathematical model that can be used for balancing a game, and thus this process is manually tweaked. The most used approach for such process is by using Beta testing [1]. The Beta test phase is a valuable source of information for the game designer, which can use such data in order to balance a game. Normally, these beta testers are volunteers who are recruited to play a Beta version of the game, that is, an early, pre-release version. Besides that, there are some tools and techniques aimed for supporting game designers while balancing a game. One successfully method is the Game Analytics [2], used to understand players's behaviours as well as problems in the game. Game Analytics tools have been used to understand cause and effects relationships in the game. Additionally, Kohwalter et al. [3] present a framework for collecting gameplay session data and map it in terms of data provenance [4]. The framework produces a provenance graph used for further analysis about why the context of actions have been performed by the player.

However, it is important to state that normally the parameters set during a game balancing is target to a group of players, or player profiles. Normally these profiles either use "easy" and "hard", or "beginner" and "expert". In this case, a desired profile must be chosen before the game actually starts. Unfortunately, even players with similar skill levels may find some aspects of the game more difficult for them individually. Besides that, according to Black and Hickey [5], player's profile can change in a progressive move or immediate change. The former is referred as evolutionary adaptation while the latter is referred as revolutionary adaptation. In this case, the statically game difficult parameters setup previously may not accommodate all the players individually, causing boredom or frustration if a game is too easy or too difficult, respectively.

The solution for such problem includes monitoring the players' actions and their performance in the game which are referred in the literature as "Dynamic Game Balancing (DGB)" [6] and "Dynamic Difficult Adjustment (DDA)" [7]. Such approaches are becoming a trend for games [8], and have been used for a while in games such as Mario Kart [9]. In order to perform this, several approaches for collecting and understanding players' behavior on the game have been proposed, including player's emotional state [10], static level design based on player's real world data (instead of arbitrary data) [11], artificial intelligence [12], [13], and data provenance [3], [12], [14], [15]. However, the use of DDA is controversial among players and developers, especially in competitive games, where the player's skills are tested. Instead, there are clearly benefits for its use in some kind of games, such as educational [16]. Besides that, it is important to state that the usage of DDA is more effective when the player does not have knowledge of its existence in the game [17].

In this paper we propose a novel approach for dynamic difficult adjustments through machine learning technique based on Hidden Markov Model (HMM). In order to validate our approach, a Space Invaders clone has been built. In the game, we model the enemies as the hidden layer and give a probability for such enemies being destroyed based on the players' movement. In this case, the game level of difficult will be self adjusted based on the player's behavior during the game. In order to validate our approach, we conducted an experiment with 11 participants, where they played each version of the game with HMM enabled and not enabled. Our results show that all the participants find the version with HMM enabled most challenging. Besides that, 54.5% of the participants had more fun with this version.

This paper is organized as follow. In Section II we presented the related work. Section III presents a brief explanation about the HMM method, while in Section IV we describe our proposed approach. Section V presents the methodology used for validating our approach. Finally, the results and discussion are presented in Section VI, while Section VII presents the conclusion and future work.

II. Related Work

In games, several artificial intelligence techniques have been developed in order to make computer challenging. Samuel [18], [19] proposed a learning technique using the game checkers. It is an important contributions in modern search-based games (with alpha-beta cutoffs and quiescence search) and learning techniques for automatically improving the program's performance over time.

Collecting data becomes crucial for performing dynamic difficult adjustment, specially aimed to analyzing the game flow. There are three basic requirements that must be satisfied [6]: the game must identify and adapt to the player's level; the game must track the evolution and regressions of player's performance; and the game must remain believable.

Chandler and Noriega [20] present a framework used to evaluate success and failure in modern games based on collected data. In the paper, they claim that a game must be challenging, and neither too hard nor too easy, suggesting that the game has to adapt its difficulty automatically based on the player's skills. In the same direction, Zuin and Macedo [21] used the HMM approach for detecting infinite combos in fighting games and thus giving the player's chance to recover from this sequence of attacks.

Tijs et al. [10] propose performing dynamic difficult balancing by using currently player's emotional stage during gameplay. The main drawback of this approach is the break in the flow state along the game [22], as the player needs to answer questions from time to time to adjust the difficult level. Vasconcelos de Medeiros and Vasconcelo de Medeiros [11], on the other hand, used offline collected player's data for procedural level design balancing. Unfortunately, due to their static approach, variation in the player's skills is not take into consideration. Hunicke [7] implements a FPS based game that uses a probabilistic approach in order to identify when the user is having troubles in the game. In order to answer for such trouble, game mechanics characteristics such as weapons or munitions spawn rate are changed in order to assist the player.

When considering artificial intelligence based approach, Prez et al [13] use fuzzy and probabilistic causal relationships through cognitive maps for dynamically increase or decrease the spawn rate of obstacles in the game as well as power ups. Olesen et al. [12] use Neuro Evolution of Augmenting Topologies in real time for adjusting game challenge. Using an in house game, they analyzed the keys that contribute in the level of difficult and proposed a model to take those keys into account.

In [23] an automatic game balancing with reinforcement learning is presented in a fighting game. It uses the Q-Learning [24], a popular reinforcement learning algorithm, in order to adjust the agents behavior. Some works [25], [26] use intelligent agents based on genetic algorithms for such difficulty balancing. One drawback of such approach is that it needs an offline training in order to act correctly. Additionally, such approach does not consider past history data while performing such adjustments in difficulty along sessions.

In [27] a machine learning technique named Hidden Markov Model was used to predict players combos in fight games. The authors aimed to an automated solution to one of the main flaws in fighting games, specifically infinite or unfair combos that make the game be easy. This work proposed the used of Hidden Markov Models to predict if a subset of player commands may result in a combo.

Matsumoto and Thawonmas [28] discussed the use of Hidden Markov Models on video games to classify players in Massive Multi-player Online Games. This classification is based on player's actions recorded in log files during the online matches. The results obtained in this work show that Hidden Markov Models have satisfactory recognition performance, especially for player's classification among different types, but having similar action frequencies.

III. Hidden Markov Model

Hidden Markov Model (HMM) is a Machine Learning technique [29], [30] which works on a double stochastic process:

i an unobservable stochastic process that is well-known as hidden part and, ii another stochastic process that produces an observable sequence.

Due to this, HMM infers the hidden part based (item i) on the co-relation of observable stochastic process (item ii).

Since HMM uses Markov Chain in the hidden part, we brief explain the Markov Chain before discussing HMM in more detail. Markov Chain may be defined as a series of random variables $x^1, x^2, x^3, \dots, x^n$ such that the following conditional independence property holds for $n \in 1, 2, 3, \dots, N$

$$p(x^{n+1}|x^1, x^2, x^3, \dots, x^n) = p(x^{n+1}|x^n). \quad (1)$$

This probabilities' condition may be represented by graph chain, where the states are random variables and the edges are the probabilities. The probabilities also represent the transition of one state to the other and, in some cases, there is a probability to keep it in the same sate. Fig. 1 depicts a Markov Chain with two variables (states) and their conditional probabilities.

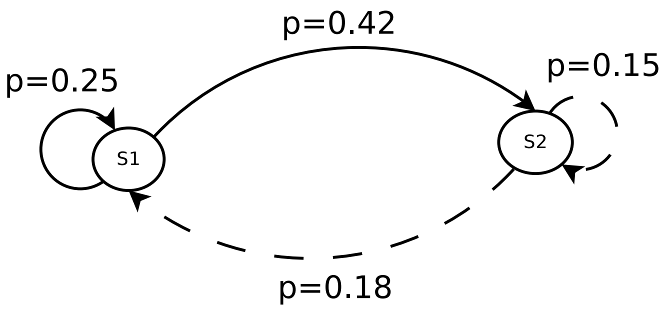

Fig. 1. Markov Chain with two variables.

In Fig. 1, the full line represents the probabilities of variable x^1, as long as the dashed line is the probabilities of the other variable. Both of them are independents and they obey to the following standard stochastic constraints:

$$a_{i,j} \geq 0 \quad (2)$$

$$\sum_{j=1}^{N} a_{i,j} = 1 \quad (3)$$

The Markov Chain can also be represented by matrix A (4), where each matrix coefficient is the transition probability.

$$A = a_{i,j} = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \quad (4)$$

In several problems, there are co-related events, eventually one of them is hard to track or observe as long as the other is not. In this case, we can built a Markov Chain through an observable events so that we can infer the co-related and untracked event. Thus, HMM arises as very useful tool which allows us to infer the Markov Chain and figure out the most likely sequence (states of Markov Chain) through other observable events that are co-related to the Markov Chain. Indeed, the HMM is an extension of discrete Markov process [31] where Markov process is unknown with another stochastic process co-related that produces an observable sequence.

We can understand the HMM concept through a simple example called: "Urns and Balls" [32] [1]. In this example, we assume that there are N urns in a room and each urn has a larger number of colored balls. We also assume that there are M distinct colors of the balls. All the urns are hidden by curtain and a person takes a random color ball from random urn, showing only us the ball so we can record the color. So, the physic dynamic of example consists in a person in the room who choose a random urn. Once the urn is selected, the person takes up a random ball of this urn, recording the color of the ball as the observation. Then, the ball is put back in the selected urn. And this same process is repeated by random selection of a new urn and a new random ball is picked up from this and recorded the ball's color as the observation. Repeating this selecting ball process for a finite time, HMM is able to build a HMM Model that allows us to predict the most sequence of urns that will be selected.

The previous example give us a good idea of HMM and its application. Besides, HMM defines its notation and elements as following:

- N is the number of states in the model, referring to states of the hidden part of the model.
- S denotes the states, where $S = \{S_0, S_1, \dots, S_{N-1}\}$.
- M represents the number of distinct observation symbols per state, the number of observation symbols.
- The transition probability distribution is defined in matrix $A = a_{i,j}$.
- The observation symbol probability in state j is given by vector $B = b_j(k)$.
- The initial distribution represented by vector π.
- The length of observation is given by T.
- Observation sequence $O = \{O_1, O_2, \dots, O_T\}$.
- $V = \{0, 1, \dots, M-1\}$ is the set of possible observation.
- Distinct Markov state is defined by $Q = \{q_0, q_1, \dots, q_{N-1}\}$

Fig. 2 depicts a general scheme of Hidden Markov Model, where X are the states of the hidden part (inside the dash box) which represents a Markov Chain. B refers to the co-relation of each state with a observation O.

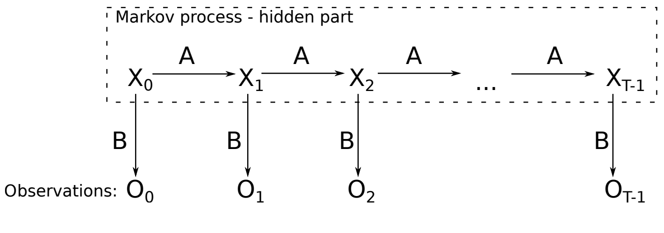

Fig. 2. Hidden Markov Model.

HMM is able to solve three fundamental problems:

i) The first problem determines the likelihood of observation O, given a known model $\lambda = (A, B, \pi)$. In this case, the

[1]The urns and balls example was introduced by Jack Ferguson in lectures on HMM theory.

model finds out an observation probability. The model parameters (matrix A, vectors B and π) have been previously defined.

ii) The second problem seeks the optimal state sequence of the hidden part (Markov process), given the model $\lambda = (A, B, \pi)$. In this case, similar to the first problem, the models parameters have already been defined.

iii) The third problem discover the model $\lambda = (A, B, \pi)$ based on an observation sequence O. In other words, given an observation sequence O and the dimensions N and M, the HMM find a model $\lambda = (A, B, \pi)$ so that it can maximize the probability of O. The third problem is also called as the training method.

In order to identify or learn pattern, we have to discovery the model $\lambda = (A, B, \pi)$, solving item iii, following the step that defines the likely states sequence of hidden part, which is the second problem (ii). Therefore, the former employs Baum-Welch Algorithm [33] to find out the model and the latter uses Viterbi Algorithm [34] to build the most likely state sequence of HMM hidden part.

Viterbi Algorithm solves item ii considering Markov property of the HMM, the joint probability of the most likely path that ends at state s in certain time instant, i.e., the highest probability path is the desired solution. The Algorithm selects the most likely terminal state, following a backtracking procedure interactive in an effort to build the joint probability of the most likely path and find observations.

As there is at least a path from the initial to final state, the Algorithm converge. For each step of backtrack interactive procedure, a part of path is selected until the initial state. If there no path from initial state to any final state, then this is not a valid path. Finally, the path length is given by the length of observation sequence.

Baum-Welch Algorithm [33] is a special case of the proposed work by B. Benyacoub et al. [35] and consist in learning step. The Algorithm starts initial HMM λ and an observation history O and finds a new HMM λ' that can explain the observations better than previous model, $P(O|\lambda') \geq P(O|\lambda)$. Aiming to the Algorithm tries to maximize $P(O|\lambda)$, the model λ' shares the same observation of λ.

Baum-Welch Algorithm is also interactive procedure and the first step is to define the initial condition $\pi_s^{'} = P(X_1 = S|O, \lambda)$, following the build matrix A, where each coefficient is given by:

$$a_{i,j} = \frac{E[S_{i,j}]}{E[S_i]} \tag{5}$$

where i and j are states and $E[S_{i,j}]$ represents the expected transition probability of state i to j. While $E[S_i]$ is the expected probability of state i;

Additionally, a matrix of entries is also build, as follow:

$$M_{s,m}^{'} = \frac{E[K_{s,m}]}{E[K_m]} \tag{6}$$

where $E[K_{s,m}]$ is the expected number of times that state s occurred in m, as long as $E[K_m]$ is the number of times that state s occurred. Finally, the Algorithm converges when $P(O|\lambda^{'}) \geq P(O|\lambda)$ is achieved.

IV. Proposed Approach

In order to demonstrate the usage of HMM for dynamic difficulty adjustment in games, we built an in-house Space Invaders clone with HMM module that is responsible for predicting the most likely player movement. The game consists in defending a base using a cannon, which can move sideways. The main objective of the game is to destroy enemies by shooting them, while avoiding enemies' shots.

With this in mind, we can see each enemy as a state of Markov Chain with a probability p to be destroyed. However, these probabilities rely on player skill in controlling the cannon and its position as well as the enemy position. In this case, each time step may be presented as a set of probabilities that each enemy can be destroyed. In such a way, the enemies can be modeled as Markov Chain that composes the hidden part and the player movement is another stochastic process that produces an observable sequence. Finally, the latter is co-related to the former based on the fact that player movement implies enemy destruction.

The aim of HMM module is to learn the player style for every n player movements. Once identified the player's style, the game adapts itself, adjusting the enemies defense strategy by operating over their shooting mechanics. The machine learning procedure in games is composed by two steps: 1) learning the player style in order to figure out the most likely enemies spaceship; and 2) counter-attack.

The learning consists in detecting the player's style, which includes their movement and shooting patterns. Following, it is necessary to make the enemies counter-attack, where some enemies will active a shield and increase the shoot frequency. Based on the described process, the HMM module uses a sequence of the player's movement to train the model $\lambda = (A, B, \pi)$, where the matrix A (Markov Chain) represents the hidden part composed by the probabilities of one enemy gets destroyed given any other that were destroyed in the previously (i.e., each enemy spaceship is a state of Markov Chain). Matrix B co-relations the hidden part with observations and vector π, which presents the initial probabilities.

During update, the game loop actives the learning process for a batch of n player movements by invoking the HMM module, so that it can learn based on the player movements. During this process, the player movement is gathered as the observation. In other words, the game observation O is given by player's movement. Here, we consider three possible movements: Left (0), Right (1) and Stopped (2). Therefore, the HMM module build a new model λ for each 100 observations, i.e., 100 player moments. Besides, the HMM module calculates the next likely moment as fast as possible, it takes less than $\frac{1}{60}$, otherwise, the game may lost its interactivity. Indeed, the length of the observation sequence should be great enough to represent the player pattern, but it must not affect the game interactivity since it is computed in only one game frame.

We could initialize matrix A and π with any random value due to training procedure and re-define these matrices in according to the observation and the matrix B. However, matrix B describes the co-relation of player spaceship movement and each enemy spaceship. In this case, we considered two variables to define those probabilities: the distance and the angle between player spaceship of each enemy spaceship at the time step where HMM model trains the model λ.

Additionally, we included both the angle of shooting (90º degree in relation to x-axis) as well as the distance in order to calculate the probability of any enemy spaceship to be hit. Enemies next to the player in an angle close to zero degrees are more likely to be destroyed. On the other hand, enemies far from the player in a wider angle have less chance to be killed, as can be seen in Fig. 3.

In order to build probabilities (matrix B), we work on inverse of distance and angle. Equation 7 presents the first part of this relation, where $d_{p,i}$ denotes the distance between i^{th}-enemy and player spaceship in p movement (Left, Right and Stopped). $\theta_{p,i}$ is the angle between the same spaceship.

$$\omega_{i,p} = \frac{1}{d_{i,p} \times \theta_{i,p}} \tag{7}$$

We build the probabilities normalizing each $\omega_{i,p}$ in relation to p movement (Equation 8). The sum of probabilities should be 1 (Equation 9).

$$\Omega_{i,p} = \frac{\omega_{i,p}}{\sum_{p=0}^{n-1} \omega_{i,p}} \tag{8}$$

$$\sum_{i=0}^{n-1} \Omega_{i,p} = 1 \tag{9}$$

In matrix B, columns represents the player's movement, whereas the rows are enemies spaceship. Given Equations 8 and 9, we can summarize Matrix B as following:

$$B = \begin{bmatrix} \Omega_{0,0} & \Omega_{1,1} & \Omega_{1,2} \\ \Omega_{1,0} & \Omega_{1,1} & \Omega_{1,2} \\ \vdots & \vdots & \vdots \\ \Omega_{n-1,0} & \Omega_{n-1,1} & \Omega_{n-1,2} \end{bmatrix} \tag{10}$$

The following step after defining matrix B is to train HMM model λ using Baum Welch Learner algorithm [36]. HMM model also provides the sequence of the most likely states. In case of our game, this sequence refers the sequence comprising the most likely enemies that can be destroyed.

Finally, the following step is to perform the game's difficulty adaptation. Hence we figure out which enemies have more likely to be destroyed, these enemies and all other in the same columns turn on their force shields and increase the frequency of shoots for 10 seconds, as Fig. 4 depicts. After that, the force shields are disable and the shoots became normal (Fig. 5).

Once selecting the most likely enemies to be destroyed, the sequence is disposed and the HMM model trains another model (λ), considering a new observation. This approach allows the game to learn the player pattern and acts accordingly. As it is performed in real time after 100 observations, the game will be adapted to consider this new observed pattern.

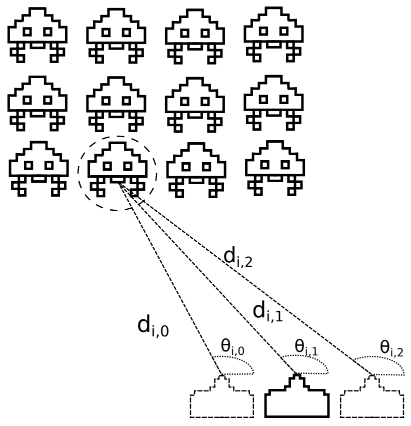

Fig. 3. Influence of the enemies' distance and angle of shooting.

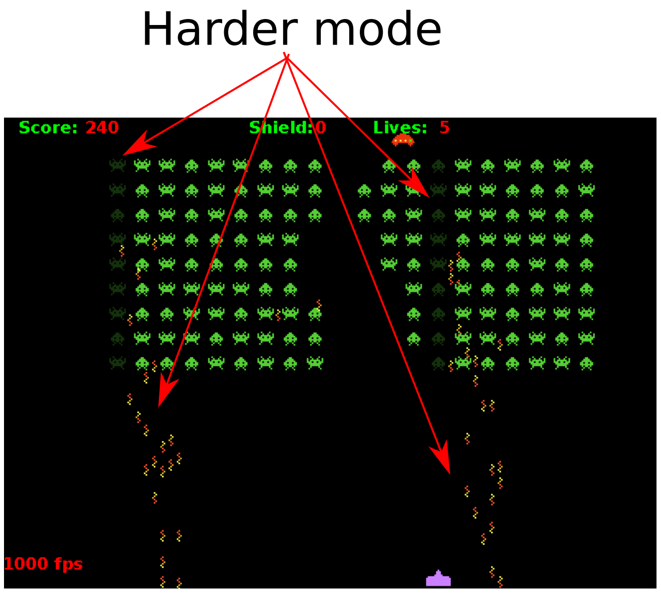

Fig. 4. Harder mode enabled.

V. Methodology

In order to validate the presented approach, a study has been performed to investigate the machine learning implementation. This investigation involved participants who played the in-house game with HMM enabled and disabled. For a didactic approach, the former is named by **Space-Invader-HMM** while the latter is just called **Space-Invader**. While playing, participants where not informed which version of the game they were playing. In fact, they did not even know the objective of the experiment before playing the game.

We carried out the tests with a total of 11 different volunteers (players) recruited from different classes in the University. Volunteers selection was diverse across age and academic career. All the participants had some experience with games, 3 had low experience, 5 mid level experience and 3 had high experience level with games.

Before the experiment actually starts, initial instructions

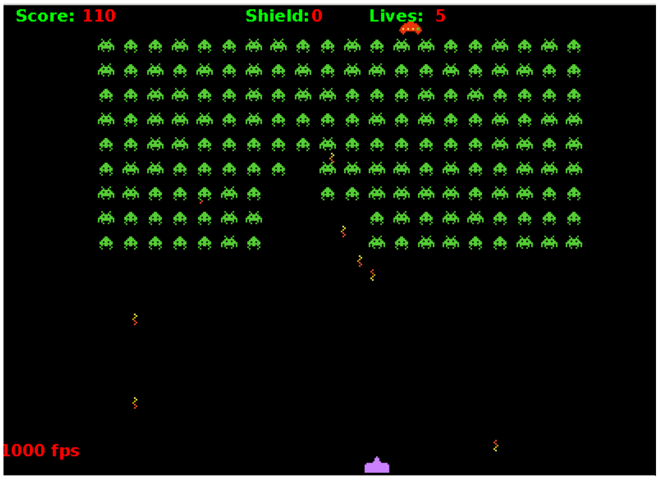

Fig. 5. Normal game mode.

are given to each participant about the game mechanics. No information about the DDA have been given to the participant in order to avoid bias in the result [17]. Following, we asked the player for playing each version of the game (**Space-Invader** and **Space-Invader-HMM**) once, for five minutes, in order to collect game statistics. All these tests have been performed using a notebooks with a keyboard to control the player spaceship. The player starts with 5 lives, infinity shoots and 180 enemies to kill. There are also a bonus enemy, which appear from time to time in the top of the screen that gives a bonus point in case the player kill it. Besides that, player's shoots can neutralize an enemy's shoot, which can be a defense strategy for the player.

During the experiment, statistical information were gathered for both version of the game, as following:

- **Player score:** the game is by considering the number of enemies killed;
- **Player Shoots:** number of shots the player made;
- **Number of enemies killed:** increased by killing enemies;
- **Number of player's lives :** every time the player get hit by colliding with an enemy or its bullet the player looses a life. The player start with 5 lives and if this number reaches 0 the game ends.

Finally, after the tests, the participants were asked to fill up a brief interview in order to evaluate the their experience. The following characteristics were addressed:

- **Difficult factor:** perception of the game difficult for both versions of the game, using a 5-point Likert scale ranging from 0 (very easy) to 5 (very difficult);
- **Most fun version:** the participant's perception about which version he/she played was most fun;
- **Most challenge version:** the participant's perception about which version he/she found more difficult.

VI. Results and Discussions

In this section, we present our results by using the HMM module. Subsection VI-A presents the performance evaluation of HMM while Section VI-B presents participants' evaluation of the HMM approach.

A. *HMM Performance*

In this section we aim to evaluated the performance of HMM module in order to analyze the feasibility of our approach in commercial games. In doing so, we measured each processing step which compose the HMM module:

i) **Initialization**: this step allocates and initializes all the HMM modules variables: A matrix and B and π vectors. Due to the fact that HMM module will learn with player's movement, A matrix and π vector can be initialized with any random value. On the other hand, B vector is calculated by Equation 7 considering each enemy presented in the game. In other words, in the beginning of game, we have 180 enemies in order to compute their probabilities.

ii) **Training**: in this step, as discussed in Section IV, the HMM module applies Baum-Welch Algorithm so that it can figure out A matrix and π vector, building the $\lambda = (A, B, \pi)$.

iii) **Most likely state sequence**: The last step uses the Viterbi Algorithm to build the most likely state sequence of HMM hidden part.

We leaded the experiment using a notebook Intel i7-4510U CPU @ 2.00GHz with two physical cores and hyperthreading enabled, totaling four virtual cores. It comprises 8 Gbytes of RAM memory and runs CentOS 7 as OS. The prototype is implmented in Java and the JVM version was Oracle J2SE build 1.8.0_144$-b$01. The game loop used is a single-threaded uncoupled model [37].

While the game is running, the HMM module performs the previous described steps for a collection of 100 player's movements. Thus, we also leaded tests to measure the computational processing of each step of HMM module and to define the asymptotic curve of HMM module in function of enemies.

Fig. 6 and 7 illustrate the power processing demand of HMM module. Remark that the training step spends almost 90% of computation time, following by the most likely states sequences and initialization requires almost nothing representing 1%.

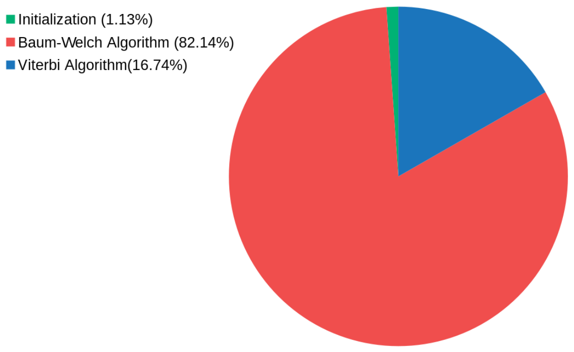

Fig. 6. Elapsed time in percent of each HMM module.

In fact, the HMM module may take too much time (Fig. 7). In accordance to Cutting et al. [38], the Baum-Welch as Viterbi present complexity $O(TN^2)$, where T is the observation length and N is the number of hidden states. Therefore, Fig. 7 reinforces both algorithms, presenting exponential growth of computational time in function of states length.

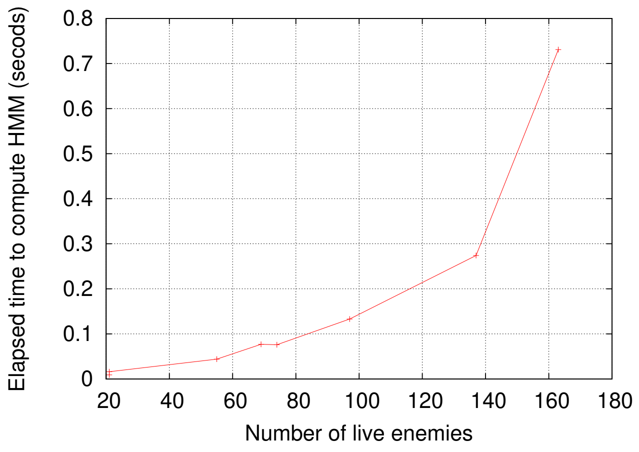

Fig. 7. Asymptotic curve based on the number of enemies.

According to Fig. 7, it is possible to observe that our game, running with HMM module activated, does not impact on the game interaction. However, due to complexity of HMM algorithms, the game invective may becomes compromised if there are too many states ($N > 180$ enemies).

B. Participant's Evaluation

Table VI-B shows the *Player Score*, *Player Shoots* (P_S, and *Killed Enemies*(K_E) for each participant in both versions of the game. Besides that, it shows the accuracy, defined here as $\frac{K_E}{P_S}$, becoming an important metric to distinguish how skilled a participant is. According to the result, is is possible to see that all the participants were capable of finishing the game (killing all the 180 enemies) in the **Space-Invader** version. On the other hand, the majority of the participants had problems while playing **Space-Invader-HMM**, as just one participant (participant K) was able to finish the game. Also, we can observe that the number of players' shoots on the **Space-Invader-HMM** version is much higher than the **Space-Invader**, which represents an increase of 36.86% considering the mean among all players. One fact to such an increase is the need to use the shoot as a defense to protect her spaceship.

Additionally, it is possible to see a decrease about 42.86% in the mean for the accuracy from **Space-Invader** to **Space-Invader-HMM**. This is another fact that can be explained by the increase in shots fired by players, as participants missed more bullets in the latter version of the game. One interesting fact is that all participants dropped their accuracy when HMM was enabled (being participant A the one that dropped more), except for participant D, which presented an slightly increase in his accuracy. One reason for such result could be a failure in the HMM to learn his pattern while playing.

When looking to the participant's answer for choosing which version had been more challenge, all of them answered it was the **Space-Invader-HMM** version. Besides that, participants classified the **Space-Invader** either very easy (81.8%) or easy (18.2%), leading to a possibly disengagement in the game after some time. On the other hand, **Space-Invader-HMM** has been scored as very difficult (27.3%), difficult (63.6%), or normal (9.1%) by the participants. However, this does not necessary indicates participants got frustrated by the challenges presented in **Space-Invader-HMM** version. For instance, participant *D* says: *"The second version [**Space-Invader-HMM**] provides new mechanics (enemies) and irregular patterns"*. It is important to state that the same participant increased his accuracy in the **Space-Invader-HMM** version. Additionally, participant *H* (the second highest accuracy drop) says: *The shots [from the enemies] constrained my movements, requiring different strategy to beat them.* In fact, according to Fig. 8, the majority of the participants (54.5%) had more fun with the second version (**Space-Invader-HMM**), while 34.6% had fun with both. Finally, when asked for additional comments in the end of the experiment, participant *D* says: *"The second version [**Space-Invader-HMM**] is more dynamic; it is interesting to learn enemies new mechanics just by observation, requiring time... The first version does not present a challenge, being boring..*

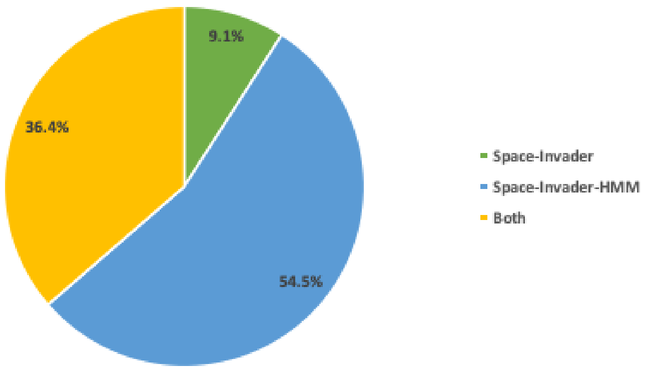

Fig. 8. The version that participants had more fun.

VII. Conclusion and Future Works

Game balancing is one of the critical aspect of the game, and also one of the more difficulty for achieve in game development. The usage of machine learning could help the development of this task, allowing developers to focus in others tasks of the game. At the same time, the majority of the games' level of difficult are based on a group of players, without considering that players evolves differently.

In this paper we presented a machine learning applied for game dynamic difficult adjustment, using a real time Hidden Markov Model. In order to validate our approach, a 2D shooting game has been developed and tested with volunteers. According to our results, the version that did not use our approach was classified as easier for the player, presenting

	Space-Invader					Space-Invader-HMM				
User	Player Score	Player Shoots	Enemies Killed	Player Killed	Accuracy	Player Score	Player Shoots	Enemies Killed	Player Killed	Accuracy
A	2250	310	180	0	0.58	2550	821	175	5	0.21
B	2100	283	180	0	0.64	2370	583	162	5	0.28
C	2150	263	180	0	0.68	2180	300	143	5	0.48
D	2350	645	180	1	0.28	2100	485	165	5	0.34
E	1850	272	180	0	0.66	2010	604	144	5	0.24
F	2150	323	180	3	0.56	1070	301	128	5	0.43
G	2000	379	180	0	0.47	2350	826	160	5	0.19
H	2050	279	180	2	0.65	1900	450	130	5	0.29
I	1950	261	180	0	0.69	820	128	77	5	0.60
J	1950	438	180	1	0.41	1840	566	134	5	0.24
K	2150	348	180	0	0.52	2670	956	180	4	0.19
Mean	**2086.36**	**345.55**	**180.00**	**0.64**	**0.56**	**1978.18**	**547.27**	**145.27**	**4.91**	**0.32**

TABLE I
RESULTS OF THE STATISTICS COLLECTED IN THE GAME PLAY.

almost no challenge. On the other hand, the game version using HMM could be finished just by one participant. Even increasing the challenge level, the majority of the participants find the HMM version more fun (90.90%).

Also, for future topics, it is included to porting the machine learning to others game types, allowing more games to have an adapted game balancing with machine learning. Besides that, an independent HMM module that can be adjusted by processing different parameters for performing DDA without requiring any kind of programming knowledge is an interesting topic for further research.

Additionally, another future work may be to develop a parallel approach of the HMM module in order to compute the model concurrently with the game, synchronizing only when necessary. With that, great domains (several states) will not affect the game interactivity.

ACKNOWLEDGMENT

The authors would like to thank CAPES, NVidia, CNPq, and FAPERJ for the financial support.

REFERENCES

[1] J. P. Davis, K. Steury, and R. Pagulayan, *A survey method for assessing perceptions of a game: The consumer playtest in game design*. Game Studies 5, 2005.

[2] M. Wulff, M. Hansen, and C. Thurau, "Gameanalytics for game developers know the facts improve and monetize," ¡http://www.gameanalytics.com/¿, accessed: 2017-01-20.

[3] T. Kohwalter, E. Clua, and L. Murta, "Provenance in games," in *Proceedings of the 2012 Brazilian Symposium on Computer Games and Digital Entertainment*, ser. SBGAMES '12, 2012, pp. 162–171.

[4] J. Freire, D. Koop, E. Santos, and C. T. Silva, "Provenance for computational tasks: A survey," *Computing in Science and Engg.*, vol. 10, no. 3, pp. 11–21, May 2008. [Online]. Available: http://dx.doi.org/10.1109/MCSE.2008.79

[5] M. Black and R. J. Hickey, "Maintaining the performance of a learned classifier under concept drift," *Intell. Data Anal.*, vol. 3, no. 6, pp. 453–474, Nov. 1999. [Online]. Available: http://dx.doi.org/10.1016/S1088-467X(99)00033-5

[6] G. Andrade, G. Ramalho, A. S. Gomes, and V. Corruble, "Dynamic game balancing: An evaluation of user satisfaction," in *Proceedings of the Second AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, ser. AIIDE'06. AAAI Press, 2006, pp. 3–8. [Online]. Available: http://dl.acm.org/citation.cfm?id=3023108.3023110

[7] R. Hunicke, "The case for dynamic difficulty adjustment in games," in *Proceedings of the 2005 ACM SIGCHI International Conference on Advances in Computer Entertainment Technology*, ser. ACE '05. New York, NY, USA: ACM, 2005, pp. 429–433. [Online]. Available: http://doi.acm.org/10.1145/1178477.1178573

[8] R. Lopes and R. Bidarra, "Adaptivity challenges in games and simulations: A survey," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 3, no. 2, pp. 85–99, June 2011.

[9] A. Saltsman, "Game changers: Dynamic difficulty," ¡http://www.gameanalytics.com/¿, 2009, accessed: 2017-01-20.

[10] T. J. Tijs, D. Brokken, and W. A. Ijsselsteijn, "Dynamic game balancing by recognizing affect," in *Proceedings of the 2Nd International Conference on Fun and Games*. Berlin, Heidelberg: Springer-Verlag, 2008, pp. 88–93. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-88322-7_9

[11] R. J. V. d. Medeiros and T. F. V. d. Medeiros, "Procedural level balancing in runner games," in *2014 Brazilian Symposium on Computer Games and Digital Entertainment*, Nov 2014, pp. 109–114.

[12] J. K. Olesen, G. N. Yannakakis, and J. Hallam, "Real-time challenge balance in an rts game using rtneat," in *2008 IEEE Symposium On Computational Intelligence and Games*, Dec 2008, pp. 87–94.

[13] L. J. F. Pérez, L. A. R. Calla, L. Valente, A. A. Montenegro, and E. W. G. Clua, "Dynamic game difficulty balancing in real time using evolutionary fuzzy cognitive maps," in *2015 14th Brazilian Symposium on Computer Games and Digital Entertainment (SBGames)*, Nov 2015, pp. 24–32.

[14] L. B. Jacob, T. C. Kohwalter, A. F. V. Machado, E. W. G. Clua, and D. d. Oliveira, "A non-intrusive approach for 2d platform game design analysis based on provenance data extracted from game streaming," in *Proceedings of the 2014 Brazilian Symposium on Computer Games and Digital Entertainment*, ser. SBGAMES '14. Washington, DC, USA: IEEE Computer Society, 2014, pp. 41–50. [Online]. Available: http://dx.doi.org/10.1109/SBGAMES.2014.33

[15] F. M. Figueira, L. Nascimento, J. da Silva Junior, T. Kohwalter, L. Murta, and E. Clua, "Bing: A framework for dynamic game balancing using provenance," in *2018 17th Brazilian Symposium on Computer Games and Digital Entertainment (SBGames)*, 2018, pp. 57–5709.

[16] C. R. Beal, J. Beck, D. Westbrook, M. Atkin, and P. Cohen, "Intelligent Modeling of the User in Interactive Entertainment," in *AAAI Spring Symposium on Artificial Intelligence and Interactive Entertainment*, 2002, pp. 8–12. [Online]. Available: http://info200.infc.ulst.ac.uk/˜darryl/Papers/Digra05/digra05.pdf

[17] D. Charles, M. Mcneill, M. Mcalister, M. Black, A. Moore, K. Stringer, J. Kücklich, and A. Kerr, "Player-Centred Game Design: Player Modelling and Adaptive Digital Games," in *Digital Games Research Association 2005 Conference: Changing Views - Worlds in Play*, 2005. [Online]. Available: http://info200.infc.ulst.ac.uk/˜darryl/Papers/Digra05/digra05.pdf

[18] A. L. Samuel, "Some studies in machine learning using the game of checkers," *IBM Journal of research and development*, vol. 44, no. 1.2, pp. 206–226, 1959.

[19] ——, "Some studies in machine learning using the game of checkers.

ii—recent progress," *IBM Journal of research and development*, vol. 11, no. 6, pp. 601–617, 1967.

[20] C. Chandler and L. Noriega, "Games analysis – how to stop history repeating itself," in *WSEAS International Conference on Multimedia, Internet & Video Technologies*, 2006, pp. 47–52.

[21] G. L. Zuin and Y. P. A. Macedo, "Attempting to discover infinite combos in fighting games using hidden markov models," in *2015 14th Brazilian Symposium on Computer Games and Digital Entertainment (SBGames)*, Nov 2015, pp. 80–88.

[22] J. Schell, *The Art of Game Design: A Book of Lenses*. Morgan Kaufmann Publishers, 2008.

[23] G. Andrade, G. Ramalho, H. Santana, and V. Corruble, "Automatic computer game balancing: A reinforcement learning approach," in *Proceedings of the Fourth International Joint Conference on Autonomous Agents and Multiagent Systems*, ser. AAMAS '05. New York, NY, USA: ACM, 2005, pp. 1111–1112. [Online]. Available: http://doi.acm.org/10.1145/1082473.1082648

[24] C. J. Watkins and P. Dayan, "Q-learning," *Machine learning*, vol. 8, no. 3-4, pp. 279–292, 1992.

[25] P. Demasi and J. d. O. Adriano, "On-line coevolution for action games," *International Journal of Intelligent Games & Simulation*, vol. 2, no. 2, 2003.

[26] R. P. Wiegand, W. C. Liles, and K. A. De Jong, "Analyzing cooperative coevolution with evolutionary game theory," in *Evolutionary Computation, 2002. CEC'02. Proceedings of the 2002 Congress on*, vol. 2. IEEE, 2002, pp. 1600–1605.

[27] G. L. Zuin and Y. P. Macedo, "Attempting to discover infinite combos in fighting games using hidden markov models," in *Computer Games and Digital Entertainment (SBGames), 2015 14th Brazilian Symposium on*. IEEE, 2015, pp. 80–88.

[28] Y. Matsumoto and R. Thawonmas, "Mmog player classification using hidden markov models," *Entertainment Computing–ICEC 2004*, pp. xv–xx, 2004.

[29] S. Russell, P. Norvig, and A. Intelligence, "A modern approach," *Artificial Intelligence. Prentice-Hall, Egnlewood Cliffs*, vol. 25, p. 27, 1995.

[30] C. M. Bishop, *Pattern recognition and machine learning*. springer, 2006.

[31] L. R. Rabiner, "A tutorial on hidden markov models and selected applications in speech recognition," *Proceedings of the IEEE*, vol. 77, no. 2, pp. 257–286, 1989.

[32] S. S. Joshi and V. V. Phoha, "Investigating hidden markov models capabilities in anomaly detection," in *Proceedings of the 43rd annual Southeast regional conference-Volume 1*. ACM, 2005, pp. 98–103.

[33] L. R. Welch, "Hidden markov models and the baum-welch algorithm," *IEEE Information Theory Society Newsletter*, vol. 53, no. 4, pp. 10–13, 2003.

[34] G. D. Forney, "The viterbi algorithm," *Proceedings of the IEEE*, vol. 61, no. 3, pp. 268–278, 1973.

[35] B. Benyacoub, I. ElMoudden, S. ElBernoussi, A. Zoglat, and M. Ouzineb, "Initial model selection for the baum-welch algorithm applied to credit scoring," in *Modelling, Computation and Optimization in Information Systems and Management Sciences*. Springer, 2015, pp. 359–368.

[36] Z. Ghahramani and M. I. Jordan, "Factorial hidden markov models," in *Advances in Neural Information Processing Systems*, 1996, pp. 472–478.

[37] A. LaMothe, *Tricks of the 3D game programming gurus: advanced 3D graphics and rasterization*. Sams, 2003, vol. 2.

[38] D. Cutting, J. Kupiec, J. Pedersen, and P. Sibun, "A practical part-of-speech tagger," in *Proceedings of the third conference on Applied natural language processing*. Association for Computational Linguistics, 1992, pp. 133–140.