Entertainment Computing 33 (2020) 100336

Contents lists available at ScienceDirect

Entertainment Computing

journal homepage: www.elsevier.com/locate/entcom

Smart Reckoning: Reducing the traffic of online multiplayer games using machine learning for movement prediction

Elias P. Duarte Jr.[a,*], Aurora T.R. Pozo[b], Pamela Beltrani[c]

[a] Federal University of Parana, Curitiba, Brazil
[b] Computer Science Department at Federal University of Parana, Brazil
[c] Computer Games at Positivo University, Parana, Brazil

ARTICLE INFO

Keywords:
Massive multiplayer online games
Machine learning
Movement prediction

ABSTRACT

Massively Multiplayer Online Game (MMOG) players maintain consistent views of the positions of each other by periodically exchanging messages. Besides the fact that these messages can suffer delays that cause rendering inconsistencies, they also represent an overhead on the network. This overhead can be significant, as the number of MMOG players is very large, but reducing the number of messages is not trivial. The classic strategy to predict movement avoiding message exchange is based on the Dead Reckoning algorithm, which has several limitations. Other strategies have been proposed more recently that improve the results, but rely on expert knowledge. In this work we propose Smart Reckoning, a movement prediction strategy based on machine learning. The strategy consists of two phases. In the first phase, a learning model classifies whether the classical Dead Reckoning algorithm is able to predict the new avatar position correctly or not. In case the conclusion is negative, another learning model is used to predict the new direction. The proposed strategy was applied to the World of Warcraft game. The learning models were implemented with the Weka tool using real game trace data, and results are presented for the accuracy of multiple algorithms.

1. Introduction

Players of Massively Multiplayer Online Games (MMOGs) have to maintain a consistent view of the game world, so they can decide on actions that make sense globally. In particular, each player has to keep track of the locations of the avatars of all other players, as they move around the game world. To maintain a consistent view, players exchange messages carrying game state information across the network. The messages are transmitted on a periodic basis; the frequency depends on each game requirements. However, if messages carrying updates on avatar positions are lost and/or delayed, players may experience jumps between consecutive frames or may even execute inconsistent actions [1–3]. As current online games can have up to millions of concurrent players [4–6], messages can represent a significant overhead on the network.

Movement prediction strategies can be employed in order to avoid exchanging a message to inform the new position of an avatar. To understand how a message can be avoided by employing a prediction algorithm, consider a player (called *predictor*) that needs the new position of the avatar of another player (called *predicted*). Both players start with the previous position of the predicted avatar, and both run the prediction algorithm. After the predicted player makes a movement and executes the prediction algorithm, it can check if the result of the algorithm is correct or not: has the current position been correctly predicted? Only in case the prediction is *not* correct a message is generated and sent to inform the predictor about the actual new position. Otherwise if the algorithm predicts the new position correctly within some predefined bounds, no message is needed.

The most common strategy to predict movement is the classical Dead Reckoning algorithm [7], which is based on Newton's laws [8]. Dead Reckoning assumes that movement only occurs on a straight line, thus it implicitly assumes that the player does not change direction. This assumption is clearly unrealistc, especially in current games in which avatars may change directions at will. Other more complex strategies have been proposed [9–11], some of which build models that are based on the points of interest of the players. Although, these new strategies present better results they are not easy to use and depend on expert knowledge.

In this work, we present Smart Reckoning, a new strategy based on machine learning to predict movement in online multiplayer games. The proposed strategy consists of two phases. In the first phase a learning model classifies whether the player continues on a straight line

* Corresponding author.
E-mail address: elias@inf.ufpr.br (E.P. Duarte).

https://doi.org/10.1016/j.entcom.2019.100336
Received 25 April 2019; Received in revised form 22 September 2019; Accepted 3 December 2019
Available online 12 December 2019
1875-9521/

and does not change direction, or if it is being affected by surrounding points of interest. If the conclusion is that the direction is not going to change, Dead Reckoning can be applied to obtain a correct new position. Otherwise, the algorithm enters the second phase, in which another learning model is used to make a prediction of the new direction the player will take, and compute the new position. The proposed strategy is generic, in the sense that knowledge is learned from the data.

Smart Reckoning was implemented and we executed experiments for predicting movement in the World of Warcraft (WoW) [12] game. The learning models were built on a new dataset based on two well-know WoW public traces [13,14]. This new dataset has information about the field of vision of each player, including which other players and NPCs (Non-Playable Characters) are visible at each instant on time.

We use the Weka toolkit to built multiple versions of the learning models, and after preliminary empirical results, the following classical machine learning algorithms were chosen: Local Weighted Learning [15], Bootstrap Aggregating [16], Multilayer Perceptron [17] and Reduced Error Pruning Tree [18]. Experiments were executed, and the best results were obtained with the Bagging algorithm which presented an accuracy of 81.10% in the first phase and 73.37% in the second phase; overall the accuracy of the Bagging algorithm is 78.76% combining the two phases. Considering all four machine learning algorithms the average accuracy is 76.60% for the first phase and 51.02% for the second phase; overall the accuracy is 66.41% combining the two phases. In comparison, the accuracy of the Dead Reckoning algorithm was 44.54%. Perhaps, AntReckoning [11] is the related work that is closest to our approach, although it not based on machine learning, but on Ant Colony Optimization (ACO). Like our approach, the predictions of AntReckoning are much better than those of Dead Reckoning. However AntReckoning presents a challenge: a specialist much configure the multiple parameters, and this can be a formidable task, depending on the complexity and richness of the game. Using machine learning those parameters are simply “learned”. We believe our results show that it is very promising to use machine learning to predict movement thus avoid exchanging messages and reducing MMOG bandwidth consumption.

The rest of this paper is organized as follows. Section 2 gives definitions and describes related work in the field of MMOG movement prediction. Section 3 gives a brief overview of machine learning and the algorithms we employed. In Section 4 Smart Reckoning is described, including the implementation, an overview of the World of Warcraft game for which the experiments were executed, and the real traces employed. Results obtained are described in Section 6. Finally the conclusions follow in Section 7.

2. Movement prediction techniques: definitions & related work

One of the biggest challenges of MMOGs is the need to guarantee a consistent game view for the multiple players. For example, consider two players A and B. Consider that player A’s avatar is stationary while player B’s avatar moves on a straight line or changes the direction. At this moment, player A needs to update B’s position in the game world. The simplest way to accomplish that is to have Player B send a message to player A to inform about the change which must be “seen” by player A. If movements are very frequent, then the number of messages exchanged is also very frequent; depending on the number of players, they need to exchange a potentially huge number of messages. Techniques for predicting movement can be used in order to avoid at least some of those messages.

In this section we initially present the classical Dead Reckoning algorithm and several variants that have been proposed. Next, we present other relevant work for keeping the consistency across multiple players of MMOGs. The problem is then considered in terms of saving bandwidth, i.e. reducing the number of messages exchanged among players. Then we describe work that have a focus on dynamically redimensioning the area of interest of the players to reduce the required bandwidth without violating the consistency. Finally we make a reference to a survey on MMOGs published in 2012, before most of the other work described in this section.

2.1. Dead reckoning

Dead Reckoning is one of the earliest and most popular techniques for movement prediction. The Dead Reckoning algorithm estimates the new position of a given player using traditional physics movement equations and receiving as input the last known position of the moving object [7]. Consider two players: one has just moved and the other must update the new position. If Dead Reckoning is employed, both players execute the algorithm and a message is sent only if the difference between the estimated new position and the actual new position is larger than a pre-defined threshold. Besides the last known position (x_t) at time instant t, Dead Reckoning also uses the velocity $vt = \dot{x}_t$ and the acceleration $a_t = \ddot{x}_t$ at time t. Other data can be used to help estimate the forces that are applied to the moving entity. For instance, for objects moving in a three dimensional space the orientation, angular velocity and the angular acceleration are also needed. With these variables the trajectory of a moving object can be computed using traditional physics movement laws.

Basically all movement predictors employ the same pattern. As a player moves, it also computes the estimation that other players will make. If the real position differs from the estimation by a larger margin than that defined by a threshold, then the player sends a message to the other players informing the new position. Thus all players keep the positions and compute estimates for all other players in the so called Area of Interest (AoI), which surrounds the player and is defined and dimensioned by each game. As mentioned above, Dead Reckoning uses physical laws. As several games relax physical laws, so that for instance drastic changes of movement are allowed, it is frequently not possible to use Dead Reckoning. Even when it can be used, some authors argue that it presents notoriously imprecise results in practice [1]. Thus several other approaches including variants of Dead Reckoning have been proposed.

Pantel and Wolf were some of the first to evaluate the feasibility of using Dead Reckoning in network-based multiplayer games [19]. The motivation for using predictions based on Dead Reckoning is to avoid network transmission delays may lead to inconsistency and other problems. They evaluate the accuracy of predictions in different types of games, including racing, sports, and action games.

McCoy and others [9] proposed the Neuro Reckoning algorithm to improve the prediction quality of Dead Reckoning. They note that Dead Reckoning trades accuracy for low computational complexity. Neuro Reckoning is actually based on a bank of neural network predictors trained to estimate changes of the speed within a relatively short time interval. After an error is detected, instead of sending the current speed, predictive information is sent that helps improve the prediction of the remote entity state. Simulation results show that Neuro Reckoning presents low computational cost and improves predictions.

AntReckoning [11] is a technique proposed to improve Dead Reckoning. AntReckoning was inspired on Ant Colony Optmization (ACO) and uses player interests to predict movement. In a way it is possible to say that AntReckoning adds player interest to the Dead Reckoning movement equations. The strategy defines a framework based on pheromones to model the players’ temporal and spatial interests. To incorporate the interests a model was created with both attraction and repulsion forces. The intensity of the applied forces is proportional to the attractiveness of the Points of Interest (PoI). PoI attractiveness must be pre-configured, as they vary from one game to another and can be any game object, including other players. Some objects attract the players, and the attraction is stronger if the object is in some way valuable or the player needs it urgently. In the same way dangerous objects are repulsive to the player.

AntReckoning models these PoIs as ants that release pheromones to

model player interests. Over time the pheromones propagate across the game world and the concentration decays with time. AntReckoning organizes the game world in cells. Let C be the size of a cell. Pheromone management and the corresponding force of attraction depend from the granularity of the cell. For each avatar P, client Q executes the Dead Reckoning algorithm. Q calculates the concentration of the pheromone on each cell and the respective forces that result. For the sake of scalability, only the cells for a given region around P are considered, which is called the attraction zone and is denoted by R. Note that pheromones propagate in R even when the corresponding PoI is not in R. AntReckoning is used in very much the same way as Dead Reckoning: each player computes the estimation for every movement and only send a message if the real position differs from the estimation by a margin larger than a threshold η.

Dong [20] focuses on Real-time Online Interactive Applications (ROIA) as a whole and not only on games, but also presents a version of Dead Reckoning. The problem tackled is the fact that the classical Dead Reckoning algorithm does not take into consideration the user's goal under the current system state, and thus this Dead Reckoning version incorporates target prediction.

In [21] the authors evaluate different movement prediction methods for a two-dimensional racing game. The main purpose is to evaluate how far inconsistencies due to network delays can be avoided. A testing framework was implemented. Experiments were executed in which data from real players was first collected and then evaluated offline as another step. One of the conclusions is the poor performance of Dead Reckoning, but the method the authors call "input prediction" gave the best precision. This method is a variation of Dead Reckoning proposed in [19] that takes into account user commands (not only the position) to predict the next movement.

Jaya, Liu and Chen [10] also propose a variation of Dead Reckoning integrating interest management. The result is a multi-threshold Dead Reckoning algorithm that employs Zone Based Interest Management (ZBIM) and is more flexible than the single threshold traditional Dead Reckoning approach. Experiments were executed using The Open Racing Car Simulator (TORCS) and the conclusion is that the solution improves Dead Reckoning without adding significant computational overhead.

We note that recently there has been an increased interest in Dead Reckoning for predicting car movement, both in the context of vehicular networks [22] and car simulators [23].

2.2. Game consistency

In [24] the authors also evaluate the consistency of network-based multiplayer games. They define a metric to quantify what they call the "time–space inconsistency", which is computed assuming that Dead Reckoning is used and also taking into consideration parameters such as clock asynchrony, transmission delays, human factors, among others. They conclude that the diverse parameters are interrelated in terms of their impact on time–space inconsistency. A ping pong game was employed to check how to fine-tune the game to improve the consistency.

Bondarenko [25] classifies players interactions as either short or long-range, and evaluate the latency sensitivity of each category. Experimental results show that short-range players can tolerate considerably lower latency levels than long-range interactions.

2.3. Bandwidth usage

In [26] the authors executed WoW traces to check the game requirements in terms of network bandwidth. They evaluated how a pure P2P publish-subscribe strategy would compare with an equivalent client–server solution. The conclusion is that client–server solutions result in lower game latencies. They then propose the usage of message aggregation can have a significant impact by reducing the required bandwidth of both the client–server and P2P solutions.

DynFilter [27] is a message processing middleware that limits the number of messages exchanged by remote game entities to inform state updates, so that the bandwidth effectively used stays within a predefined quota. The motivation for the development of the middleware is the fact that server-side bandwidth provisioning is critical for MMOGs. Requirements are difficult to estimate due to factors such as the highly variable number of players and the collective behaviors of large numbers of players. If bandwidth provisioning fails, players can experience problems even disconnections. Experiments show that by DynFilter succeeds in keeping the bandwidth requirements within certain limits while being able to deliver relevant state update messages.

A related problem is the delivery of state update messages in large-scale P2P virtual environments, which is done in a distributed fashion and involving a varying number of participants. In [28] this problem is investigated under a new approach based on a model that takes into the account what the authors call "continuous events". A continuous event results in a series of updates that can be pre-computed. The purpose is to avoid sending some update messages given a series of events. The model is defined, algorithms for continuous event management are presented, and simulation results confirm the reduction of update messages.

2.4. Dynamic area of interest

In [29] Shen and others show empirically that although a single Area of Interest (AoI) is enough for single-avatar virtual environments, it is not enough for games with multiple avatars (MAVEs Multi-Avatar Virtual Environments). To solve the problem they propose a new concept, the Area of Simulation (AoS) which combines and extends both AoI and the popular strategy for synchronizing the game world called EBLS (Event-Based Lockstep Simulation). AoS uses both event-based and update-based models to manage multiple areas of interest, which is more than traditional AoI, but less than EBLS which considers the whole game world. Results confirm the scalability and effectiveness of employing AoS.

In [30] the focus is on combat games, in particular on interest management during combats, which are highly interactive and fast-paced. The focus is on adjusting the AoI of the players depending on the actions they execute. The purpose is to maximize the utility of the trade-off between consistency and performance.

The focus of Yahavi and others in [31] is on NPCs: they argue actually that NPCs can be a problem as they are easily recognized by players reducing the sense of immersion and limiting character interactions. They propose a categorization of NPCs and metrics for quantifying NPC performance in terms of movement, interactions, their influence on decision-making schemes. Then they propose an influence map based on pheromones that can improve the game experience, by giving a summary of the events in the game world which can be effectively used in the decision processes related to NPCs. Experimental results are presented for the Quake III game.

2.5. A survey on MMOGs

Finally, also available is a survey of MMOGs with a focus on P2P techniques and architectures [32]. The authors give an overview of strategies used by MMOGs to maintain state information and guarantee consistency. The techniques they identify includes super peer storage, overlay storage, hybrid storage, and distance-based storage. The techniques are compared in terms of scalability, fairness, reliability, responsiveness, and security. The survey was published in 2012, before several of the other related work described above.

As can be seen in Table 1, related work can be classified into five groups: Dead Reckoning and Variations; Game Consistency; Bandwidth Usage; Dynamic Area of Interest; plus other early approaches that have been described in the 2012 survey [32]. Furthermore, only one of those

Table 1
Related work: a summary.

Dead Reckoning and Variations
Dead Reckoning [7]: Specification of the original algorithm
Pantel and Wolf [19]: Evaluates Dead Reckoning in games
McCoy et al. [9]: Improves Dead Reckoning with Neuro Reckoning
Yahyavi et al. [11]: Proposes AntReckoning
Dong [20]: Proposes a version of Dead Reckoning with target prediction
Larsson [21]: Evaluates Dead Reckoning in terms of network delay
Jaya et al. [10]: Proposes a version of Dead Reckoning with ZBIM
Balico et al. [22]: Applies Dead Reckoning for Vehicular Networks
Chen and Liu [23]: Applies Dead Reckoning for Car Simulators
Game Consistency
Zhou et al. [24]: Defines the time–space inconsistency metric
Bondarenko [25]: Evaluates short- and long-range interactions
Bandwidth Usage
Miller and Crowcroft [26]: Message agreggation
Gascon et al. [27]: Dynamic limits for sending messages
Heger et al. [28]: Model to avoid messages for continuous events
Dynamic AoI
Shen et al. [29]: Defines AoS to manage multiple AoI
Wang et al. [30]: Adjusts AoI according to players' actions
Yahyavi et al. [31]: Predictions that consider NPCs
A 2012 Survey
Gilmore and Engelbrecht [32]: A survey that describes other early approaches

groups presents contributions that are comparable to ours, that is the first group: Dead Reckoning and Variations. Within this group we chose two works to be compared with Smart Reckoning: the original Dead Reckoning and AntReckoning [31], which is the closest to ours, as it also predicts avatar movement taking into account the AoI, NPCs, and avatar behavior. Actually, the precision of Smart Reckoning in comparison with Dead Reckoning is similar to the improvement provided by AntReckoning: 30%. The advantage of our approach is that AntReckoning requires a specialist to fine tune multiple parameters that are employed in their model; in our case Smart Reckoning learns those parameters by itself.

3. Machine learning techniques

Machine learning is the field concerned with the study of mechanisms to automatically induce knowledge from data and thus learning to solve problems [33,34]. Machine learning is a branch of Artificial Intelligence which is right in the intersection between Computer Science and Statistics.

Machine learning tasks are typically classified into two broad categories according to the available information or data: unsupervised and supervised learning. To understand these concepts, we first define that the data is composed by samples, which are by themselves described by a set of attributes. Unsupervised learning detects relationships among the examples, e.g., the detection of similar groups of examples. Clustering can be considered the most important unsupervised learning task. Clustering techniques explore similarities between patterns, grouping the similar units into categories or groups.

The distance (in terms of a given metric) is often used as a measure of the similarity. In this way, the shorter the distance, the more similar the samples are. Furthermore, the association task is an example of unsupervised learning that searches to identify relationships among the attributes that characterize the samples.

Classification and prediction are called supervised learning techniques, as they are based on a training phase. Our contribution is based on these techniques. The training data consist of pairs of inputs (vectors) and an specification of the desired outputs. The classification task produces a model based on the data, which is used to classify unseen examples according to its input (attributes). A prediction task, as the name implies, is used to predict the value of a variable in the future. It is possible to use classification algorithms to predict the future value of discrete variables.

Table 2
Example of a confusion matrix.

	Classified : Negative	Classified : Positive
Reality : Negative	TN = a	FP = b
Reality : Positive	FN = c	TP = d

There is a great variety of classification algorithms, including: linear regression, logistic regression, decision trees, support vector matrices, the naive Bayes algorithm, random forests, among many others. It is a common practice to evaluate multiple different algorithms to choose the best for a specific problem.

The evaluation of a classification algorithm is usually done using a confusion matrix [34]. A confusion matrix can be used to check whether the results obtained with a machine learning algorithm are trustworthy. The matrix has information about how inputs were classified and shows if predictions were correct. To build a confusion matrix, the real classification must be available. It is then possible to compare the obtained classification with the real classification. Each row of the matrix represents the predictions while each column represents the real classifications (or vice versa). The evaluation of the confusion matrix is done after the end of the training phase, and with new data – called the test data set – which consists of data that was not used before. Table 2 illustrates a simple example in which a sample can be classified as either positive or negative. The confusion matrix clearly shows whether predictions are a – True Negatives (TN); b – False Positives (FP); c – False Negatives (FN); or d – True Positives (TP).

In this work we employed the Weka toolkit, version 3.8.2 [35]. After an initial step in which virtually all algorithms available were tried, four algorithms were selected because of their potential: Reduced Error Pruning Tree (REPTree) [17], Local Weighted Learning (LWL), Bootstrap Agregating (Bagging) and the Multilayer Perceptron neural network [16]. Each of these algorithms is briefly described next.

The REPtree algorithm builds a decision (or a regression) tree. Decision Tree learning (DT) is a method for approximating valued target functions, in which the learned function is represented by a Decision Tree. Learned trees can also be represented as sets of if-then rules to improve human readability. These learning methods are among the most popular of inductive inference algorithms and have been successfully applied to a broad range of learning tasks [36]. Sample classification is done by DT by searching down the tree from the root to some leaf node that corresponds to the classification of the sample. Each node of the tree specifies a test of some attribute of the sample, and each branch descending from that node corresponds to one of the possible values for this attribute. An instance is classified by starting at the root node of the tree, testing the attribute specified by this node, then moving down the tree branch corresponding to the value of the attribute for the sample. This process is then repeated for other subtrees [36]. Most learning algorithms based on Decision Trees employ variations of this top-down, greedy search through the space of possible Decision Trees. The REPtree algorithm builds a DT using information gain (or variance) and employs reduced-error pruning with backfitting [35].

The LWL algorithm employs a form of lazy memory-based learning and focuses on locally weighted linear regression [37]. Lazy learning methods defer processing of training data until a query needs to be answered. This usually involves storing the training data in memory, and finding the relevant data to answer a particular query. LWL finds a set of nearest neighbors and uses them to locally build a linear regression model.

Bagging (Bootstrap Agregating) is a machine learning ensemble meta-algorithm [38] designed to improve the stability and accuracy of machine learning algorithms. Boosting is a way of combining many weak classifiers to produce a powerful "committee", it works by

sequentially applying each classification algorithm for multiple versions of training data and takes a weighted majority vote from the classifiers employed. At each iteration the weights are computed according to the error (or loss) for each example of the learning algorithm. Initially, all the weights are the same, but at each round, the weights of the samples that are misclassified are increased so that weak learners are forced to put an extra effort on these samples during the training phase. Although this strategy is usually applied with algorithms based on decision trees, it can be used with any type of learning algorithms.

Finally, Multilayer Perceptron 39 are mathematic models inspired by the neural structure of intelligent organisms that acquire knowledge by experience. The network is composed by processing units, neurons that are linked by communication paths. This connection is associated with a weight value called the *synaptic strength*. In a neural network, the knowledge is distributed across the network, and stored at the synapses of each neuron. During the learning phase, synaptic weights, and threshold values are adjusted until they yield the desired outputs. In the end the network can be used to solve the problem.

The user has to choose several parameters, including the network topology, the number of layers, the number of neurons of each layer, and the functions associated with the neurons. Other parameters are related to the learning algorithm: a termination criterion, the learning rate, and the initial weight values. To start the process, a set of training patterns (input, and desired outputs) is necessary.

A typical topology for structuring the neurons is a multi-layer neural network 39. The output layer is the layer from which the final response is obtained. Intermediate layers are called hidden layers because their outputs are not readily observable. In this kind of neural network, the information flows from the input layer to the output layer without return cycles (feed forward topology).

There are different learning algorithms; the most well known is the back-propagation learning algorithm 39. This algorithm works in two major steps: 1) the input is presented to the input layer, and propagated until it reaches the output layer; and 2) the output is compared to the desired one, and the error is computed. The error is then used to update the weights, first of the output layer. Then the algorithm continues computing the error, and computing new weight values, moving layer by layer backwards toward the input. These steps are repeated for each available input, and are called an “epoch”. Several epochs may be necessary to eventually reach a steady state and obtain the solution.

4. The Proposed Strategy: Smart Reckoning

In this section we first present the Smart Reckoning strategy to predict avatar movement using machine learning. Next, we describe its application to a very popular game: World of Warcraft (WoW)[1] that has sold over 100 millions copies and is played in 224 countries and territories [12].

4.1. Description

Smart Reckoning is based on machine learning and only requires game data to learn and take decisions. Game data is usually available as traces which are simply collected as regular players play the game. In a way, a game trace tells the story of how the game was played. Before a trace is used it is of course cleaned and put in a format that can be understood. Fig. 1 shows that the learning models use the game traces as input to learn gameplay patterns.

Avatar movement depends on its current position and what is in it area of interest. In the proposed strategy each machine learning algorithm is trained with input data obtained from the real traces that must include (i) the avatar position and direction and (ii) the set of Non-Playable-Characters (NPCs) and other avatars in the current area of interest, to produce as output (iii) the next position and direction of the avatar. As we were running the first trials we observed that when the avatar moved on a straight line, Dead Reckoning alone resulted in good predictions. Instead when the avatar changed the direction, Dead Reckoning fails completely, actually in this case it does not make sense to run Dead Reckoning. These factors were decisive in the design of the proposed strategy, as described next.

Smart Reckoning consists of two phases and is invoked every time a player needs to render the avatar of another player. In the first phase, a machine learning algorithm determines whether the Dead Reckoning algorithm will correctly predict the new position of the avatar. Dead Reckoning assumes that a player moves on a straight line and uses physical laws to predict the next position. If the machine learning algorithm decides that Dead Reckoning will fail in its prediction, the second phase is executed. In this case the assumption is that the avatar will change its direction. The new direction is decided by the player given her individual game strategy. Thus it may depend on the NPCs in the area of interest, other avatars, among other factors. In this way in the second phase a machine learning algorithm is used to determine the new angle corresponding to the change of direction of the avatar.

Fig. 2 shows that the information about the current game environment is used as input to the proposed strategy. A particular player position and information about its environment in the game is taken as input to Phase 1 which decides if Dead Reckoning is enough to compute the new position of this player. In other words, in this phase the main question is whether the player is likely to continue moving on a straight line or will change the direction. If the conclusion is that the player will continue on a straight line then Dead Reckoning is used to compute the new position. Otherwise the second phase is triggered, in which another learning model is used to compute the new direction, i.e. the angle of the player’s next move. Whatever the decision (the player will move on a straight line or will change direction) the final output is the new position predicted for the player.

It is important to highlight that the proposed strategy uses game traces to learn the models but, after the models are incorporated into the game no more training is needed. In other words, we are using and off-line approach.

4.2. Application to WoW

World of Warcraft (WoW) is a Massively Multiplayer Online Role Playing Game (RPG). The RPG game genre has some unique characteristics: it is necessary to measure quantitatively each player’s progress, there is a strong social interaction among players, it is possible to personalize the avatars and the system’s architecture is focused on allowing a huge number of players. In the World of Warcraft game, the players can observe their progress along the time as they change levels, acquire equipment, become richer, and conquer achievements by executing certain tasks. In order for a player to progress, the corresponding avatar engages in battles with monsters or are sent on quests by NPCs. Both actions can be executed either by a single player or in groups, and rewards (also called experience points) are received as a result.

Within the World of Warcraft game, we chose the city of Ironforge as we could find reliable public game traces collected in this city. Players can choose one of two factions in the game: the Alliance and the Horde. Ironforge is one of the capitals of the Alliance. Depending on the server to which the player is connected, it is possible to engage in Player-versus-Player (PvP) battles, which are between players of different factions. As Ironforge is a capital of the Alliance, it is highly unlikely to be visited by avatars of the Horde, so aggressive actions are uncommon there. This is enforced by the fact that NPCs of this city actually attack avatars of the Horde. Ironforge is particularly important for being the initial city for every dwarf avatar, and because of this fact it is common to see both high and low level players there. An avatar of a

[1] https://worldofwarcraft.com

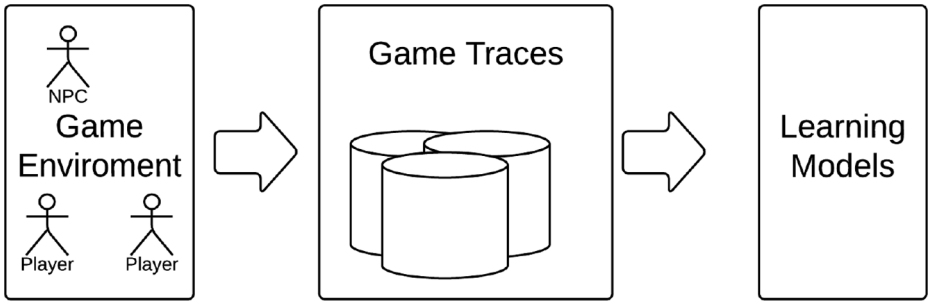

Fig. 1. Game traces are obtained from monitoring and employed by the learning models.

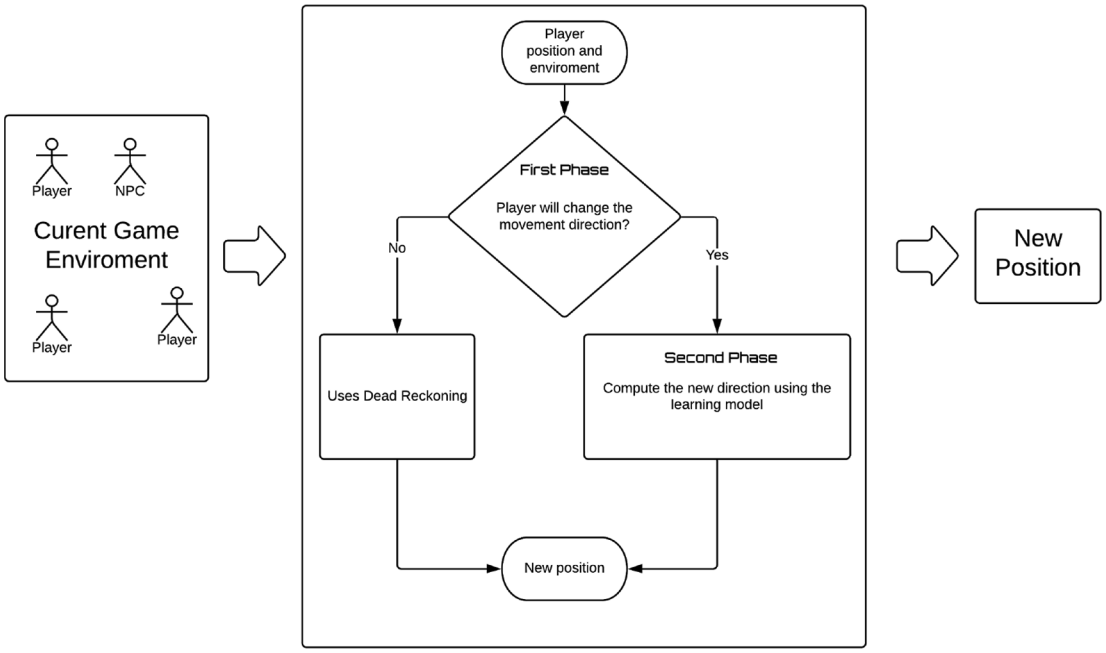

Fig. 2. Smart Reckoning consists of two phases.

high level player (above level 20) can ride mounts that are much faster than a simple walking avatar, and above level 60 the mounts can even fly. A NPC of this city can perform the following roles: Auctioneer, Banker, Battle Master, Collector, Innkeeper, Merchant, among others. Each individual player may or may not be interested in any of this NPCs depending of its own role, for example a Battle Master is not likely to be interested in a Merchant selling sewing material, or an avatar of a high level player will not be interested in quests for beginners.

Although WoW is a three-dimensional game, we employed the Thottbot Position System [14] that maps the position of every element to a two-dimensional space ignoring the height. Thottbot is one of multiple plugins/extensions that have been created by the WoW community to improve/personalize the game experience. Thottbot was developed to collect statistical data about the game, including for example data on how long elements stayed intact, the positions of NPC's, the positions of enemies, among many others. Samples of data collected across the world were sent to a central server. In 2010 Thottbot was discontinued and incorporated into the WowHead database [40], which is one of the traces we employed in this work. The *position* of a player is defined on a two-dimensional space by tuple (x,y), where $0 \leqslant x, y \leqslant 100$, the superior left corner represents (0,0). The Area of Interest is defined as a circle with radius $d = 100$ game units and the center on the corresponding avatar [13]. The *field of vision* of a player is defined taking into account the player position, the direction the player is looking to, the maximum angle within which an avatar or NPC is visible is defined as $\alpha = 80°$, which is similar to human vision.

In order to train and test the learning models we employed data collected from real players available in two plublic traces: WowPosition [13] and WowHead.[2] Actually the dataset we employed consists of a combination of what is available in these two traces. An entry of the dataset consists of the following fields, which are described next: x, y, *AngleOfView*, *DeadReckoning*, *OldMovementAngle*, *MovementAngle*, and data about avatars and NPCs that are within the field of vision of the player. The tuple (x, y) corresponds to the position of avatar. The *AngleOfView* indicates the direction to which avatar is looking. The *DeadReckoning* field indicates whether the classical algorithm predicted the last movement correctly or not; this was an important field for training the new learning algorithms in the first phase of the proposed strategy. The *OldMovementAngle* and *MovementAngle* fields describe the last angles the player has taken and the respective timestamps. These fields were important to train the algorithms in the second phase of the new strategy. Basically, the algorithms of the first phase we trained ignoring the *OldMovementAngle* and *MovementAngle* fields and in the second phase ignored the *DeadReckoning* field.

The data about other avatars and NPCs within the field of vision are: their position, the *distance* to those entities and the angle between the player's avatar and each entity. The maximum number of entities N in the field of vision is limited; in this way if there are more than N entities, the closer ones are taken into consideration and the others are ignored. In order to determine whether an avatar or NPC is within the

[2] http://www.wowhead.com.

field of vision it is enough to compute the an angle between the two taking into account the position and the angle corresponding to the direction of the avatar (*AngleOfView*). The WoWPosition trace employs the original WoW coordinates, while WoWHead which is important as it features NPCs as well as more game information, uses the Thottbot Position System. The main reason that we built our dataset with the Thottbot Position System is that all the information about NPCs is available in this system.

5. Experimental results

In this section we present experiments that were executed to evaluate Smart Reckoning and in particular to answer the question of how much machine learning improves the prediction in comparison with Dead Reckoning. The implementation was based on the Weka toolkit[3]. After a round of preliminary investigations, in which virtually all the algorithms available in Weka were checked, the four algorithms described in the previous section stood out for the good results they presented. These were the algorithms we thus employed to predict gamer movement: Reduced Error Pruning Tree (REPTree), Local Weighted Learning (LWL), Bootstrap Aggregating (Bagging) and Multilayer Perceptron (MLP). As mentioned before, the REP-Tree algorithm is based on decision trees; the LWL algorithm is based on a weighted voting strategy. The Bagging algorithm generates multiple versions of a classifier using multiple versions of the original training set. Finally, the Multilayer Perceptron is a neural network.

As mentioned before, the experiments were executed for the World of Warcraft (WoW) game, using the two public traces described in the previous section that were available for the Ironforge city of is game: WoWPosition and WoWHead. These traces track real gamers for several hours. The WoWPosition trace only shows avatar positions, while WoWHead includes more game information such as NPC location, enemies, maps, among others. Our dataset shows for each player and at each time instant which NPCs and avatars are within the AoI. The WoWPosition trace adopts an area of interest that is a circle with radius of 100 game units (referred to as d). The angle that restricts the vision of each player α was set to 80 degrees, which is similar to that of humans. Actually two datasets were created, the first using data from the first hour and the second using data from the eighth hour. Table 3 shows a description of the two datasets (Dsets). The table shows number of entries in each dataset, each entry corresponds to the position of one avatar in the game. The table also shows the accuracy of Dead Reckoning (DR Accuracy) and the number of distinct avatars in the dataset (#Avtrs). The last two fields indicate the initial and final timestamps (Init Tstamp & Final Tstamp) for each dataset. Furthermore, the value of visible players and NPCs, N, was set to 7.

The average accuracy of Dead Reckoning considering the two datasets was 57.89%, this corresponds to the number of messages that were not transmitted among the players to inform a new position. The accuracy for the first dataset was 61.17% while the accuracy for the second dataset dropped to 44.54%. Note that the number of distinct avatars also drops from 164 to 43, thus there are less players that exchange messages in the second dataset.

Next we present results of three experiments. All experiments were executed after training the algorithms with the first dataset shown in Table 3, all the results we present next were obtained by running the test dataset.

The first experiment was executed to evaluate the use of the learning models to determine whether Dead Reckoning will succeed in predicting the next movement or not. The second experiment was executed to evaluate the prediction of the angle when the avatar changes the direction after Dead Reckoning is predicted to fail. The third experiment was executed to investigate the precision of the strategy whenever the algorithm computes the angle in which the direction changes (independent of the prediction about Dead Reckoning). Last, we present the final results for all algorithms.

Table 3
The new generated database.

Dset	#Entries	DR Accuracy	#Avtrs	Init Tstamp	Final Tstamp
1	19870	61.17%	164	63445137060	63445140659
2	4885	44.54%	43	63445165860	63445169459

5.1. The first experiment: predicting dead reckoning success

The purpose of the first experiment was to evaluate the accuracy of using machine learning to decide whether Dead Reckoning will succeed or fail in its prediction. Thus, given an avatar at a certain position and time instant, the output of the machine learning algorithm was “yes” if it considered that Dead Reckoning would correctly determine the next position, and “no” otherwise. This is actually the first step of the proposed strategy as described in Section 3. If the machine learning algorithm predicts that Dead Reckoning is not going to succeed, then the second phase of our strategy is triggered, which predicts the angle at which the avatar will make the next movement. The accuracy reflects the percentage of times that the learning model was correct, and four cases are possible. A true positive (TP) corresponds to a case in which the machine learning algorithm predicted that Dead Reckoning would succeed and it did succeed, while in a true negative (TN) the machine learning prediction was that Dead Reckoning would not succeed, and exactly that happened. The false negative refers to the case in which the machine learning prediction is that Dead Reckoning is going to fail, but it actually succeeds in computing the next movement correctly. Finally the false positive refers to the case in which machine learning predicts that Dead Reckoning will succeed, but it fails.

The results are presented in Table 4. For this table we show the percentages obtained for TP, FP, TN, FN, as well as the overall accuracy. Thus for example, consider all situations for which Dead Reckoning did succeed. Each algorithm predicted some of those situations correctly (TP) but for others it was incorrect (FN). Let TP′, TN′, FP′ and FN′ be the numbers of cases. Thus in the table TP = TP′/ (TP′ + FN′); FN = FN′/ (TP′+FN′); FP = FP′(FP′ + TN′); TN = TN′/(TN′ + FP′). It is possible to notice that the overall average accuracy (TP + TN) of all algorithms is 76.60%. Actually the Bagging algorithm reached an average accuracy of 81.10%, the predictions of the success of Dead Reckoning were even higher for Bagging, 83.17%. The average percentage of false positives is 26.59%, i.e. the machine learning algorithms predict that Dead Reckoning will fail, but it succeeds. The effect of a false positive is that the second phase of the algorithm is triggered, it was not necessary to do so. The average percentage of false negatives was 20.70%, comparing with true negatives (73.4%) it is possible to conclude that the algorithms have a quite high rate of success in the predictions both of the success and failure of Dead Reckoning.

5.2. The second experiment: predicting the change of direction

The second experiment was also executed to investigate the precision of the learning models to predict the angle of a change of direction. As in this case Dead Reckoning fails, the training and testing datasets are those indicated in Table 3, except that all entries for which Dead Reckoning succeeded were removed. We assume that the prediction is correct when the error corresponding to the difference of the real angle and the computed angle is less than 15°.

The results are shown in Table 5. Considering the overall average for all algorithms, the percentage of correct predictions (Correct Preds) is 56.5%. Again the Bagging algorithm achieved the best results, with a percentage of 83.78% of correct predictions for the new movement

[3] https://www.cs.waikato.ac.nz/ml/weka/

Table 4
Predicting whether Dead Reckoning will succeed or not.

Algorithm	TP	FP	TN	FN	Accuracy
Multilayer Perceptron	81.35%	22.98%	77.02%	18.65%	79.42%
Bagging	83.17%	21.43%	78.57%	16.83%	81.10%
LWL	69.92%	39.82%	61.18%	30.08%	65.85%
REPTree	82.74%	23.16%	76.84%	17.26%	80.04%
Overall Average	79.49%	26.59%	73.40%	20.70%	76.60%

Table 5
Results for the precision of angle prediction.

Algorithm	Error	Correlat Coeff	Correct Preds	Wrong Preds
MLP	33.44%	0.81	1067(49.04%)	1109(50.96%)
Bagging	15.35%	0.92	1823(83.78%)	353(16.22%)
LWL	52.50%	0.79	230 (10.56%)	1946(89.44%)
REPTree	17.20%	0.91	1799(82.67%)	377(17.33)%
Average	29.62%	0.85	56.50%	43.48%

angle, and the error is in average 15.35% while the average error for all algorithms is 29.62%.

5.3. Third experiment: precision of the second phase independent of the first phase

The second experiment was also executed to investigate the precision of the learning models to predict the angle of a change of direction, but in all those cases in which the angle is computed. Thus false negatives of the first phase are included: the prediction had been that Dead Reckoning would fail but it succeeded. Thus the datasets for traning and testing each algorithm were built from the original datasets indicated in Table 3 with TP and FN entries alone.

Table 6 shows that the overall average accuracy for all algorithms was equal to 48,97%. In comparison with the previous experiment, it is possible to observe that the average accuracy for the same algorithms was 43.38%, thus an improvement of 5.49%. The conclusion is that even when the algorithms predict that Dead Reckoning will fail but it does not, the final outcome is often correct. As expected the average error of this experiment is of course larger than that of the previous (34.88% and 29.62%, respectively). This is expected as we are also computing the error of the false negatives.

5.4. Final results for the two phases

The final results indicating how much bandwidth each algorithm saves is presented in Table 7. The complete datasets in Table 3 were used. The best algorithm in this case is Bagging which saves 78.76% of the bandwidth that is required without any movement predictor. REPtree saves 76.78% of the bandwidth. MLP has lower accuracy (66.16%) and the worst is LWL which has an accuracy of 43.78% that is even lower than the original Dead Reckoning algorithm (44.54%).

In comparison with AntReckoning which improves the accuracy of Dead Reckoning by 30% [31]. Our results for two algorithms are better than that (Bagging and REPtree). However, as mentioned before the great advantage of using our approach is that no specialist is required to configure multiple parameters as is the case of Ant Reckoning.

Table 6
Results of the third experiment.

Algorithm	#Entries	Error	CC	Correct Preds	Wrong Preds
MLP	2167	36.75%	0.76	1021 (47.12%)	1146 (52.88%)
Bagging	2193	21.86%	0.87	1609 (73.37%)	584 (26.63%)
LWL	2272	55.79%	0.67	312 (13.73%)	1960 (86.26%)
REPTree	2237	25.12%	0.85	1563 (69.87%)	674 (30.13%)
Average	2717.25	34.88%	0.78	51.02%	48.97%

Table 7
Final results for the two phases.

Algorithm	Accuracy
MLP	66.16%
Bagging	78.76%
LWL	43.78%
REPTree	76.78%
Original Dead Reckoning	44.54%

As a final comment, we measured the time required both to train our models and run a test. The time taken to train the neural network which was the slowest was 1160.5 s. Then the time taken to run the tests on 4885 instances was 0.48 s, thus the time to run model to make a decision with Smart Reckoning embedded in the game is 0.48/4885 = 0.000098 s, i.e. 98 microseconds. Note that we are assuming that both neural networks are executed in parallel.

6. Conclusions

This work presented Smart Reckoning, a new strategy based on machine learning for avatar movement prediction to reduce the network bandwidth consumption of MMOGs. The problem is very relevant, as millions of concurrent players can be playing the game and exchanging messages informing new position updates. Smart Reckoning consists of two phases: in the first phase a learning model is employed to predict whether Dead Reckoning will succeed or fail in determining the next movement. The second phase is based on the assumption that if Dead Reckoning fails, the avatar will change direction, and a learning model is employed to compute the angle of this change of direction. To validate the proposed strategy we employed the World of Warcrat game, and public traces of real players were employed for training and testing the learning models. Four machine learning algorithms were employed: Reduced Error Pruning Tree (REPTree), Local Weighted Learning (LWL), Bootstrap Aggregating (Bagging) and Multilayer Perceptron. Results obtained with the Weka toolkit show an overall average accuracy of 76.60% for the first phase and 51.502% for the second phase. The Bagging algorithm was the best and its accuracy in first phase was of 81.10% and 73.37% for the second phase. The accuracy effectively represents the number of messages that are saved by the algorithms. In comparison with related work, we note that as AntReckoning improves the accuracy of Dead Reckoning by 30% our results are similar, but the best algorithms performed better than AntReckoning. However, the great advantage of using Smart Reckoning is that no specialist is required to configure multiple parameters as is the case of Ant Reckoning: using Machine Learning, the parameters are automatically learned.

Future work includes adding more player information to the dataset to check whether the results can be improved even further. In addition, we are looking forward to seeing results of the proposed strategy applied to other MMOGs including other genres of online games. Finally there are many different open research questions on the application of machine learning for predicting avatar movement. For instance, a relevant point for future investigation is why the accuracy of the different algorithms vary so widely in this context. Last, more work is also needed to apply the proposed strategies make predictions on line.

Declaration of Competing Interest

None.

Acknowledgments

This work was partially supported by Brazilian Education Ministry – CAPES and Brazilian Research Council – CNPq Grant 311451/2016–0.

References

[1] M. Claypool, K. Claypool, Latency can kill: precision and deadline in online games, Proceedings of the First Annual ACM SIGMM Conference on Multimedia Systems, 2010, pp. 215–222.

[2] C.-Y. Huang, C.-H. Hsu, D.-Y. Chen, K.-T. Chen, Quantifying user satisfaction in mobile cloud games, in: Proceedings of Workshop on Mobile Video Delivery, MoViD'14, ACM, New York, NY, USA, 2013, pp. 4:1–4:6. https://doi.org/10.1145/2579465.2579468. http://doi.acm.org/10.1145/2579465.2579468.

[3] S. Schmidt, S. Zadtootaghaj, S. Möller, Towards the delay sensitivity of games: there is more than genres, Quality of Multimedia Experience (QoMEX), 2017 Ninth International Conference on, IEEE, 2017, pp. 1–6.

[4] A. Yahyavi, B. Kemme, Peer-to-peer architectures for massively multiplayer online games: a survey, ACM Comput. Surv. (CSUR) 46 (1) (2013) 9.

[5] A. El Rhalibi, D. Al-Jumeily, Dynamic area of interest management for massively multiplayer online games using opnet, in: Developments in eSystems Engineering (DeSE), 2017 10th International Conference on, IEEE, 2017, pp. 50–55.

[6] D. De Felice, M. Granato, L.A. Ripamonti, M. Trubian, D. Gadia, D. Maggiorini, Effect of different looting systems on the behavior of players in a mmog: simulation with real data, eHealth 360, Springer, 2017, pp. 110–118.

[7] IEEE-SA, Ieee standard for distributed interactive simulation - application protocols, IEEE Std 1278.1-1995 (1996) i–doi:10.1109/IEEESTD.1996.80831.

[8] I. Newton, Philosophiae naturalis principia mathematica, Vol. 1, G. Brookman, 1833.

[9] A. Mccoy, T. Ward, S. Mcloone, D. Delaney, Multistep-ahead neural-network predictors for network traffic reduction in distributed interactive applications, ACM Trans. Model. Comput. Simul. (TOMACS) 17 (4) (2007) 16.

[10] I. Jaya, E.S. Liu, Y. Chen, Combining interest management and dead reckoning: a hybrid approach for efficient data distribution in multiplayer online games, Distributed Simulation and Real Time Applications (DS-RT), 2016 IEEE/ACM 20th International Symposium on, IEEE, 2016, pp. 92–99.

[11] A. Yahyavi, K. Huguenin, B. Kemme, Interest modeling in games: the case of dead reckoning, Multimedia Syst. 19 (3) (2013) 255–270.

[12] Blizzard, Azeroth by numbers, 2013. <http://media.wow-europe.com/infographic/en/world-of-warcraft-infographic.html>.

[13] S. Shen, N. Brouwers, A. Iosup, D. Epema, Characterization of human mobility in networked virtual environments, in: Proceedings of Network and Operating System Support on Digital Audio and Video Workshop, ACM, 2014, p. 13.

[14] WoWHead, It's flat! using wowhead maps, 2010. URL http://www.wowhead.com/news=63952/its-flat-using-wow head-maps.

[15] A. Skoglund, M.L. Course, Locally weighted learning for control, Artif. Intell. Rev. 11 (1997) 11–73.

[16] B.D. Ripley, Pattern recognition via neural networks, a volume of Oxford Graduate Lectures on Neural Networks, title to be decided. Oxford University Press [See <http://www.stats.ox.ac.uk/ripley/papers.html>].

[17] J.R. Quinlan, Simplifying decision trees, Int. J. Man-Mach. Stud. 27 (3) (1987) 221–234.

[18] D. Opitz, R. Maclin, Popular ensemble methods: an empirical study, J. Artif. Intell. Res. 11 (1999) 169–198.

[19] L. Pantel, L.C. Wolf, On the suitability of dead reckoning schemes for games, in: Proceedings of the 1st Workshop on Network and System Support for Games, NetGames '02, 2002.

[20] L. Dong, A target-predicting dr algorithm in roia cloud platform, in: P2P, Parallel, Grid, Cloud and Internet Computing (3PGCIC), 2013 Eighth International Conference on, IEEE, 2013, pp. 25–28.

[21] E. Larsson, Movement prediction algorithms for high latency games: A testing framework for 2d racing games, Ph.D. thesis, Blekinge Institute of Technology, Sweden, Undergraduate Dissertation, 2016.

[22] L.N. Balico, A.A.F. Loureiro, E.F. Nakamura, R.S. Barreto, R.W. Pazzi, H.A.B.F. Oliveira, Localization prediction in vehicular ad hoc networks, IEEE Commun. Surv. Tutor. 20 (4) (2018) 2784–2803.

[23] Y. Chen, E.S. Liu, Comparing dead reckoning algorithms for distributed car simulations, Proceedings of the 2018 ACM SIGSIM Conference on Principles of Advanced Discrete Simulation, SIGSIM-PADS '18, 2018, pp. 105–111.

[24] S. Zhou, W. Cai, B.-S. Lee, S.J. Turner, Time-space consistency in large-scale distributed virtual environments, ACM Trans. Model. Comput. Simul. 14 (1) (2004) 31–47.

[25] O. Bondarenko, The influence of latency on short-and long-range player interactions in a virtual environment, Master's thesis University of Oslo, 2012.

[26] J.L. Miller, J. Crowcroft, The near-term feasibility of p2p mmog's, in: Network and Systems Support for Games (NetGames), 2010 9th Annual Workshop on, IEEE, 2010, pp. 1–6.

[27] J. Gascon-Samson, J. Kienzle, B. Kemme, Dynfilter: Limiting bandwidth of online games using adaptive pub/sub message filtering, in: International Workshop on Network and Systems Support for Games (NetGames 2015), 2015, pp. 1–6.

[28] F. Heger, G. Schiele, R. Süselbeck, L. Itzel, C. Becker, Scalability in peer-to-peer-based mmves: The continuous events approach, in: Consumer Communications and Networking Conference (CCNC), 2012 IEEE, IEEE, 2012, pp. 629–633.

[29] S. Shen, S.-Y. Hu, A. Iosup, D. Epema, Area of simulation: mechanism and architecture for multi-avatar virtual environments, ACM Trans. Multimedia Comput., Commun., Appl. (TOMM) 12 (1) (2015) 8.

[30] J.Y. Wang, K. Zhang, H.-A. Jacobsen, Combat state-aware interest management for online games, Proceedings of the 18th ACM/IFIP/USENIX Middleware Conference: Posters and Demos, ACM, 2017, pp. 17–18.

[31] A. Yahyavi, J. Tremblay, C. Verbrugge, B. Kemme, Towards the design of a human-like fps npc using pheromone maps, in: Games Innovation Conference (IGIC), 2013 IEEE International, IEEE, 2013, pp. 275–282.

[32] J.S. Gilmore, H.A. Engelbrecht, A survey of state persistency in peer-to-peer massively multiplayer online games, IEEE Trans. Parallel Distrib. Syst.

[33] T.M. Mitchell, Machine Learning, first ed., McGraw-Hill Inc, New York, NY, USA, 1997.

[34] M.I. Jordan, T.M. Mitchell, Machine learning: trends, perspectives, and prospects, Science 349 (6245) (2015) 255–260.

[35] I.H. Witten, E. Frank, M.A. Hall, C.J. Pal, Data Mining, Fourth Edition: Practical Machine Learning Tools and Techniques, fourth ed., Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2016.

[36] T.M. Mitchell, Machine Learning, McGraw-Hill, 1997.

[37] C.G. Atkeson, A.W. Moore, S. Schaal, Locally Weighted Learning for Control, Springer, Netherlands, Dordrecht, 1997, pp. 75–113.

[38] R.E. Schapire, The Boosting Approach to Machine Learning: An Overview, Springer, New York, New York, NY, 2003, pp. 149–171.

[39] S. Kartalopoulos, Understanding Neural Networks and Fuzzy Logic: Basic Concepts and Applications, IEEE Press, 1996.

[40] WowHead, Thottbot merged with wowhead framework, 2010. <http://www.wowhead.com/news=175371/thottbot-merged-with-wowhead-framework>.

Elias P. Duarte Jr. is a Full Professor at Federal University of Parana, Curitiba, Brazil, where he is the leader of the Computer Networks and Distributed Systems Lab (LaRSis). His research interests include Computer Networks and Distributed Systems, their Dependability, Management, and Algorithms. He has published nearly 200 peer-reviewer papers and has supervised more than 120 students both on the graduate and undergraduate levels. Prof. Duarte is currently Associate Editor of the IEEE Transactions on Dependable and Secure Computing, and has served as chair of more than 20 conferences and workshops in his fields of interest. He received a Ph.D. degree in Computer Science from Tokyo Institute of Technology, Japan, 1997, M.Sc. degree in Telecommunications from the Polytechnical University of Madrid, Spain, 1991, and both BSc and MSc degrees in Computer Science from Federal University of Minas Gerais, Brazil, 1987 and 1991, respectively. He chaired the Special Interest Group on Fault Tolerant Computing of the Brazilian Computing Society (2005-2007); the Graduate Program in Computer Science of UFPR (2006-2008); and the Brazilian National Laboratory on Computer Networks (2012-2016). He is a member of the Brazilian Computing Society and a Senior Member of the IEEE.

Aurora T. R. Pozo is Full Professor of Computer Science Department at Federal University of Parana, Brazil, and chair of the Bio-inspired Computation Laboratory (C-Bio). She received a Ph.D. in electrical engineering from the Federal University of Santa Catarina, Brazil. She received a M.S. in electrical engineering from Federal University of Santa Catarina, Brazil, in 1991. Prof. Aurora's research interests are in evolutionary computation, data mining and complex problems. She has served on several Editorial Boards, and chaired and served on the TPC of several conferences and workshops in her fields of interest. Prof. Aurora has published more than 100 peer-reviewed papers, and has supervised more than 50 students, including graduate and undergraduate levels. She is a member of the Brazilian Computer, the IEEE and ACM Society.

Pamela Beltrani is an Assistant Professor of Computer Games at Positivo University, Parana, Brazil. She received an M. Sc. in Computer Science from Federal University of Parana (2015) and a B. Sc. in Computer Science from the Catholic University of Parana-PUCPR (2012). Her research interests are on Computer Game Technology, in particular Massively Online Multiplayer Games (MMOGs).