Autonomous virtual player in a video game imitating human players : the ORION framework

Cédric Buche
Lab-STICC, ENIB
Email: buche@enib.fr

Cindy Even
Virtualys and Lab-STICC, ENIB
Email: even@enib.fr

Julien Soler
Virtualys
Email: julien.soler@virtualys.com

***Abstract*—This paper introduces the design of autonomous virtual player based on imitation learning using human behavior observations. The ORION model provides both data mining techniques allowing the extraction of knowledge and behavior models allowing the control of the autonomous behaviors. ORION is also an operational tool allowing the representation, transformation, visualization and prediction of data. We illustrate the use of our model by detailing the implementation of a virtual player for the video game Unreal Tournament 3. Thanks to ORION, data from low level behaviors were collected through three scenarios performed by human players : movement, long range aiming and close combat. Behaviors can then be learned from the obtained data-sets after transformations and application of data mining techniques. ORION allows us to build a complete behavior using an extension of a Behavior Tree integrating *ad hoc* features in order to manage aspects of behavior that we have not been able to learn automatically.**

***Keywords*-Data Mining; Artificial Intelligence; Video Games; Imitation learning ; Behavior trees.**

I. INTRODUCTION

This research focuses on the design of skillful and believable non-Player Characters (NPCs) using imitation learning techniques. It consists in extracting knowledge from data produced by human players to reproduce their behaviors. The datasets obtained can be of various types and forms and can be abstract or have strong semantics. The *data semantics* is generally not taken into account in different data processing and representation tools such as ELKI [1] WEKA [2], GGobi [3] or Orange Canvas [4]. Without semantics, once feature extraction is performed, the data become abstract. Investigating why a particular data is misclassified, for example, becomes extremely complex. Even if these tools allow to visualize data through scatter plots, histograms and other diagrams, in order to perform an analysis it is essential to visualize the data according to their semantics [5]. Existing tools suffer from *limited data visualization* possibility, providing numeric and categorical data only and never make the link with behavioral model. The ORION tackles those issues. The ORION model proposes a generic approach to represent data-sets. It also offer the possibility to perform some transformations on these data-sets and to visualize them. In addition, the ORION model makes the *link between data and behavioral models*. Our model can be used in any game as long as data from its environment and controls are accessible. In section II, we present the ORION model. The work-flow is presented in section II-A, followed by a detailed description using UML diagrams as a generic data mining model in section II-B. In section III, we illustrate the use of the ORION tool to control a virtual player in the game Unreal Tournament 3. We conclude this paper in section IV.

II. ORION

A. ORION *Workflow*

Taking the example of a 3D video game, data are usually strongly associated to concrete concepts such as position, speed, orientations, hit points, etc. Using ORION, first, the analyst adds semantics to the data. Then, she can transform and visualize them according to their semantics to ease the exploratory analysis. She can also train predictive model with the data. Next, she uses the behavioral model to build a behavior using information extracted previously. ORION's behavior model is based on a paradigm commonly used in the video game's industry giving access to the traditional tools used for IA.

B. ORION *Model*

1) ORION *Structural Model:*

Data Representation: In our model (see Figure 1), we preserve the semantics of the data via a generic approach: each attribute has a type. An interface `Type` must be implemented for each type of data to be processed. The ORION model provides basic types (reals, integers, enumerations) but also vectors, images, etc. This list can be extended by implementing the `Type` interface. These implementations are then available in our tool via the reflection mechanism of the language (here, Java). The components that will transform or display the data therefore have access to the associated semantics. `Dataset` stores information on the data set : its name, if it is or not a time series, and in that case its sampling frequency, etc. It is composed of `VariableSpec` representing the specification of each attribute of the data-set. The `VariableSet` is a list containing the values referenced by `VariableSpec`.

Data Transformation: In order to perform data transformations, the generic `DataTransform` interface must be implemented (Figure 2). It has access to the `Dataset` and a list of `VariableSpec` to process and must implement the `DataTransform.transform()` method. Allowed transformations can vary significantly : attributes modification, addition of attributes, etc. Various sub-interfaces (`ClusteringTransform`, `DataTransformer`, `SummarizeTransform`, etc.) are available to have access on the type of transformation performed. Most of the traditional transformation algorithms are implemented in ORION but other ones can easily be added by implementing the `DataTransform` interface.

Data Visualization: Every element in charge of data visualization implements the `DataViewer` interface (see Figure 3). This interface is configurable by the enumeration `DisplayRange` to define the data to be represented (the current datum, the current selection or all data).

Data Prediction: To achieve data prediction and improve the learning functions, ORION offers a level of abstraction for all methods of classification and regression. Figure 4 illustrates this model. Regression and classification algorithms implemented in ORION include classical algorithms and can be extended by implementing the `Classification` or `Regressor` interface.

2) ORION *Behavioral Model:* ORION's behavioral model is an extension of behavior trees (BT). We extended the model of BT since it is endowed with many qualities to achieve AI in video games. First, without losing the potential of BT to add *ad hoc* code features, we use a `VariableSet` as an execution context. It contains, at each tick, data collected by the agent and allows the different nodes to change this context (add or change data). The data being typed, each behavior can control its consistency and performs introspection on the data.

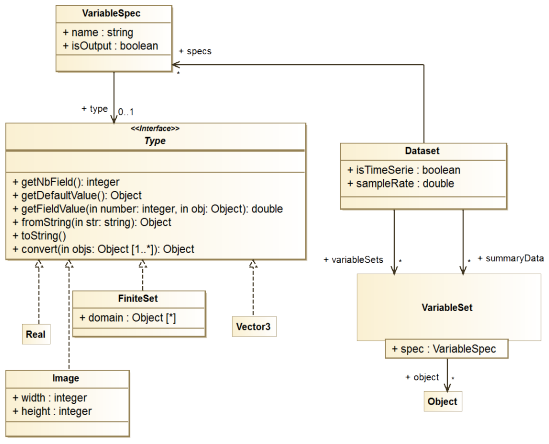

Figure 1. ORION data model

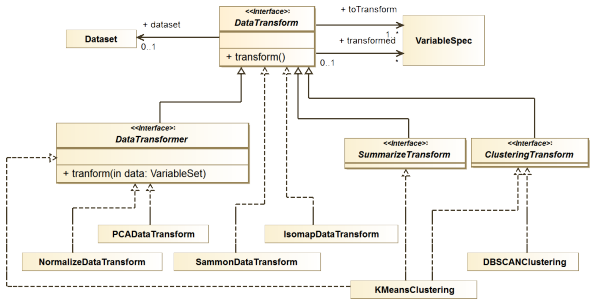

Figure 2. ORION data transformation model

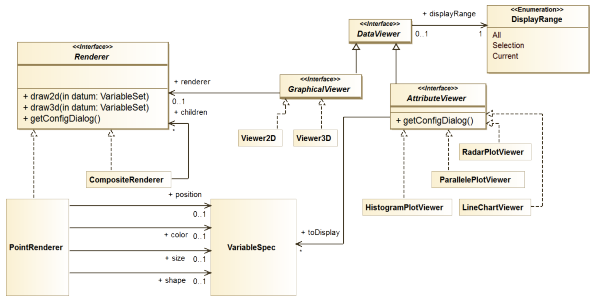

Figure 3. ORION Data visualization model

Data Mining Behaviors: Our objective is to add to BT the possibility of using learned behaviors from a data-set. We propose to add node types to the BT model. First, a particular type of node composed of a `FunctionLearner` to predict an output variable based on input variables is added. This is an action node that changes the context by adding the predicted value in it. We also add a new type of composite node to choose which child node to run through the prediction of a classifier. Finally, we add an action node composed of a `DataTransformer`, to transform the data. Figure 5 presents our model of BT.

Online and Offline Learning: The construction of BT is possible both online and offline. The offline construction is simply enabled by our implementation of ORION, manually constructing the tree and setting the node parameters.But online training requires to integrate other mechanisms. Most data mining techniques we studied previously are not iterative so it is not possible to start learning at each tick. We must therefore offer a flexible method to trigger a learning algorithm, following an event in the game or after a number of tick for example. We decide to add to ORION some nodes dedicated to online learning. These nodes allow to store online data within a data-set and to launch the training over those online data-sets (Figure 6).

III. APPLICATION

To imitate human behavior, we first need to collect data from human players operating in the game environment and to extract knowledge from these data. The use of ORION

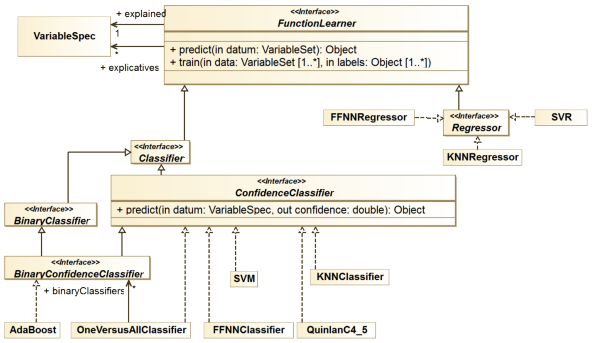

Figure 4. ORION data prediction model

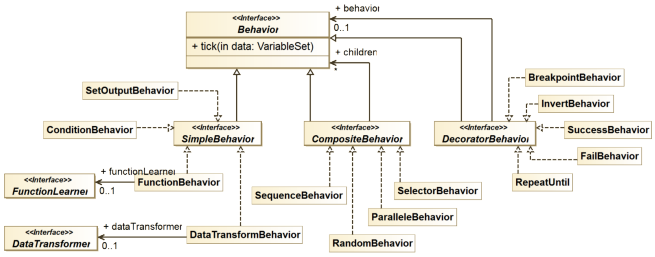

Figure 5. ORION data mining behaviors

in the design of a Bot for the game Unreal Tournament 3 (UT3) is described in this section. This game is a First Person Shooter (FPS) video game. The player incarnates a character with a variety of futuristic weapons, evolving in an arena with other players. The arena contains items that players can collect.

A. Human Players Tracking

1) Raw Data: When observing a player evolving in the game with the GameBots mod, many information about him and his environment are available. Raw data (table I) are processed with ORION as explained later.

Player position	Enemy positions*
Player orientation	Enemy orientations*
Player velocity	Enemy velocities*
Current weapon name	Number of seen items
Life points	Item positions*
Remaining ammunition	Number of seen navigation points
Shoot (is the player shooting?)	Navigation point positions*
Number of enemies visible	(* : the three closest)

Table I
DATA COLLECTED

2) Data Collection: In order to create data-sets containing only sequences where a player is performing a specific task, we conducted scripted gaming sessions with these three scenarios :

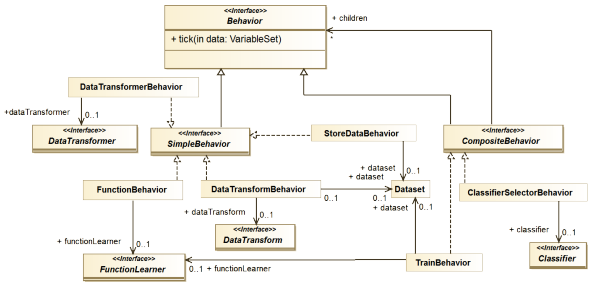

Figure 6. ORION online learning model

Movement: The player must simply navigate in the environment. No enemy is present. The obtained data will allow us to learn how to move in the environment.

Long range aiming: The player cannot move. He has infinite ammunition. He is alone in a large room with a single enemy and only intended to shoot it.

Close Combat: The player is in a small room with an enemy and must dodge attacks while firing at the enemy.

B. Data Transformation and Visualization

With ORION, exploratory analysis is facilitated by the addition of semantics. In UT3, most data are spatial locations of elements in the game. ORION implements several types associated with locations in three-dimensional space and allows their visualization. Using these features, we can rebuild the game play in ORION.Spatial location from raw data does not allow an effective reproduction of the observed behavior. The position of the player, should not be used as input data of a learning algorithm since the learned behavior would only be able to reproduce actions at specific locations where they were observed. Using ORION, the data related to location can be transformed to change the coordinates from the environment to the local coordinates of the player.

C. Data Prediction

After collecting data with each scenario, the behaviors need to be replicated individually. ORION allows the integration of various algorithms : KNN for navigation, FFNN for aiming and IOHMM for close combat.

1) Movement: For the movements of the Bot we use the CHAMELEON model [6] and the Stable Growing Neural Gas (SGNG) [7] algorithm which suggests to learn a navigation graph with a vector quantization. However, it is not sufficient to reproduce the movements of a real player. Players with a minimum of experience have a natural tendency to strafe (move sideways) when reaching the corner of a hallway which maximizes their field of vision. To learn this behavior we submit to the SGNG, besides the player's position, his speed and direction. One step of the algorithm is to submit a data to the network and to find the two closest data. When the data represents a position in space, the use of Euclidean distance is totally justified. But when

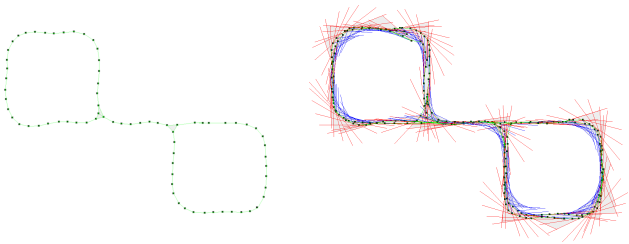

Figure 7. SGNG Results on UT3 (Left: CHAMELEON, Right: ORION)

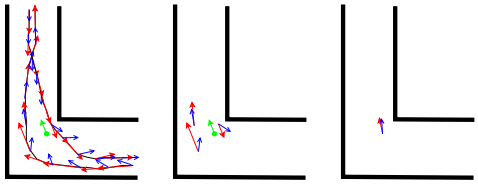

Figure 8. Movement regression principle in UT3

data additionally contains speed and direction, the choice is more complex. The orientation is indicated in our data by a unit vector, therefore the use of a simple Euclidean distance would tend to just ignore it from the positions. We perform a re-scaling by feeding the SGNG with the following distance:

$$Distance((\vec{p1},\vec{o1},\vec{v1}),(\vec{p2},\vec{o2},\vec{v2})) = d(\vec{p1},\vec{p2}) + \alpha d(\vec{o1},\vec{o2}) + \beta d(\vec{v1},\vec{v2}) \quad (1)$$

where d is the regular Euclidean distance and α and β are scaling factors. To choose the scale parameter related to the speed, we start from the observation that a player rarely moves backwards when simply navigating (but does, when fighting). The direction of the velocity vector is strongly correlated with the orientation. The norm of the velocity vector is also relatively constant when the player is moving (the character control being done by keyboard, which only contains binary switches). So there seems to be no need to take into account the speed when calculating the distance and we therefore choose to set β to zero. On the contrary, the orientation is essential. We expect the vector quantization performed by the SGNG to be able to contain two data having the same position but opposite orientations for example. We choose the parameter α so that criterion is met. Figure 7 illustrates the result. On the left, the SGNG has learned the navigation graph (CHAMELEON version). On the right, the velocity vectors are shown in red and the direction vectors are shown in blue ($\alpha = 250$). We distinguish the difference between the orientation and velocity vectors in the turns, illustrating the strafing performed by the player. To reproduce a more realistic navigation behavior, we use the vector quantization as explained above with the algorithm K-Nearest Neighbors (KNN). In figure 8, a part of the network is displayed in (a). The point and the green arrow correspond respectively to the current position of the agent and the direction it aims in. As for the construction of the network with the SGNG, the use of KNN requires the use of a distance. We use the same distance 1. Knowing the position and the desired direction of the agent, we try to get the speed and direction that the agent should adopt. We must use the velocity vector (indicating the desired direction) and not the orientation, to compute this distance : $\alpha = 0$. To set the parameter β, we apply this requirement : two data should be far away if their velocities are in opposite directions, even if their position is really close. In figure 8, the image (b) illustrates a possible choice of three nearest data ($k = 3$). The image (c) shows the regression then performed by weighting the closest data by the inverse of the distance to the data submitted to KNN.

2) Long Range Aiming: Aiming is one of the most important elements of FPS games. For ease we chose to reproduce the aim of an immobile player. Experienced player uses, at least partially, the movement for aiming but beginners and average players mainly perform static aiming which is what we decide to focus on. The inputs and outputs involved when performing this behavior are rather obvious. The outputs are necessarily rotational speeds (in yaw and pitch) and firing. As inputs, the position of the enemy is essential and it is likely that its speed is necessary too. By performing an exploratory analysis, we found no clear correlation between the enemy orientation and outputs. So we do not use this information as input to learn this behavior. But we found the current weapon is very important as an input. Some weapons are used by continuously pressing the fire button and others by discrete firing. In addition, the velocity of projectiles is different from weapon to weapon. Therefore, the influence of enemy speed is not the same from one weapon to another. We identify, regardless of the weapon used, the following correlations between:
- the Y position of the enemy in the player's coordinate system (right or left) and the yaw rotation speed.
- the Z position of the enemy in the player's coordinate system (top or bottom) and the pitch rotation speed.
- the Y enemy speed in the player's coordinate system (right or left) and the yaw rotation speed.

The differences in behavior according to the weapon suggest to process different training sessions for each of them. We learn these behaviors with regression algorithms. We conducted this learning through Feed Forward Neural Network (FFNN), KNN and Support Vector Machine (SVM) algorithms. The quantitative results appeared to be quite similar from one algorithm to another but the resulting behaviors were not really correlated to the Mean Squared Error (MSE). To compare the artificial behaviors to those of the human player, we reproduce them in our test case

scenario. When collecting the data for the weapon called LinkGun, the human player scored 50 points and was killed 10 times (so the bot scores 10 points), winning the game. We have reproduced this scenario using our behavior instead of the player's. The best behavior, obtained with a multilayer perceptron, killed the bot 32 times and was eliminated 50 times, losing the game. Our behavior is therefore less efficient than the player.

3) Close Combat: Close combat is common in UT3 and some weapons are more appropriate for this kind of confrontation. By analyzing the data from our scenario in which the player is in contact with an enemy in a small room, it was difficult to find relevant information that could be learned.Changes in direction and jumps are common, without being obviously correlated with the slightest stimuli. The player, however, constantly tries to face the enemy. Strafing and backward runs are often used rather than the forward movement. The standard regression algorithms that we used for the aiming behavior did not provide convincing results. This is expected since the exploratory analysis shows that there is no obvious dependencies between inputs and outputs. These algorithms are used to approximate a mathematical function that, for a given input always provides the same output. However, in our data set, this is clearly not the case. Therefore, we used an Input-Output Hidden Markov Model [8]. A probabilistic model is indeed more suited to this type of data because it provides a certain unpredictability.

D. Behavioral Model

We have shown how we reproduce three low level actions: shoot the enemy from afar, fighting hand-to-hand and navigating in the environment. Now we have to offer a complete behavior, using these actions in the behavioral model of ORION. The overall behavior is implemented with a Behavior Tree (BT) that we extended in order to be able to use data mining techniques. Consult [9] for a complete description of the model. Figure 9 shows a simplified version of the BT implementing the overall behavior. In this figure, the green boxes represent reference decorator nodes. They are used to name a sub-tree in order to reference that sub-tree elsewhere in the BT. They are only present here for the sake of clarity. The behavior starts by transforming the input data (discretization and change of basis). The implementation details of this node are not shown in the figure. Then, the selected node is in charge of the choice of the low level behavior to execute. The implementation of the nodes CloseCombat and AimAndShoot is quite simple. First we test if an enemy is present (and close enough in the case of CloseCombat), following this we select the nearest enemy (by adding a variable in the execution context), we then perform the regression with the algorithm that has been trained beforehand, and finally we delete the variables that could be used by the Navigation behavior from the execution context. If no enemy is present, the Navigation behavior is executed. This behavior consist in choosing a destination (if none is already selected) and adding it in the execution context. Then, if needed, it calculates the path to get to this destination and selects the appropriate navigation point. Finally it performs regression movement with KNN. The BT is searched depth-first on each tick.

E. Results

Thanks to ORION tools, we tracked a player for some minutes (for each scenarios as explained above) in order to learn the low level behaviors.

1) Subjective analysis: Observations from the resulting behavior are the following:

Movement: we noticed a significant improvement with the possibility to obtain movements that reproduce the *strafing* behavior of human players at the turning points of corridors.

Long range aiming: we noticed that the *aiming is different* for each weapon, in order to have a satisfying result we need to perform a training session for each of them.

Close combat: The result was not entirely satisfying. The behavior of human player is *unstructured and unpredictable* which complicate its reproduction.

The resulting behaviors are promising since by watching the bot, it looks like a bot controlled by a human. In order to validate our proposition we performed a formal evaluation described in the next section.

2) Objective analysis: In order to evaluate in more detail the believability of the learned movements of the bot, we carried out a study. The study consisted of two rounds of gameplay, followed by a survey. For the first round, participants played a three minutes training match against a native bot of the game to become familiar with the game and its controls. Then, players played a five minutes match against the trained bot. Finally, players were asked to answer several Likert-style questions about the the believability of this bot's movements. A four-level Likert scale (1:Strongly disagree, 2:Disagree, 3:Agree, 4:Strongly agree) was used and the questions were the followings :

1) Its movements resemble those of a human player.
2) It rotates in a human-like fashion.
3) It looks around in a human-like fashion.
4) It avoids walls in a human-like fashion.
5) The path it takes seems to be that of a human player.

Seventeen people volunteered to participate in our study. Among them, 59% agreed (picked the level 3 or 4 on the Likert scale) with the question 1), 53% with the questions 2), 3) and 5), and 47% with the question 4). These results are encouraging since for all elements of movements, to the exception of wall avoidance, more than half of the participants found them believable. This allows us to point out the elements to be considered with greater importance when learning.

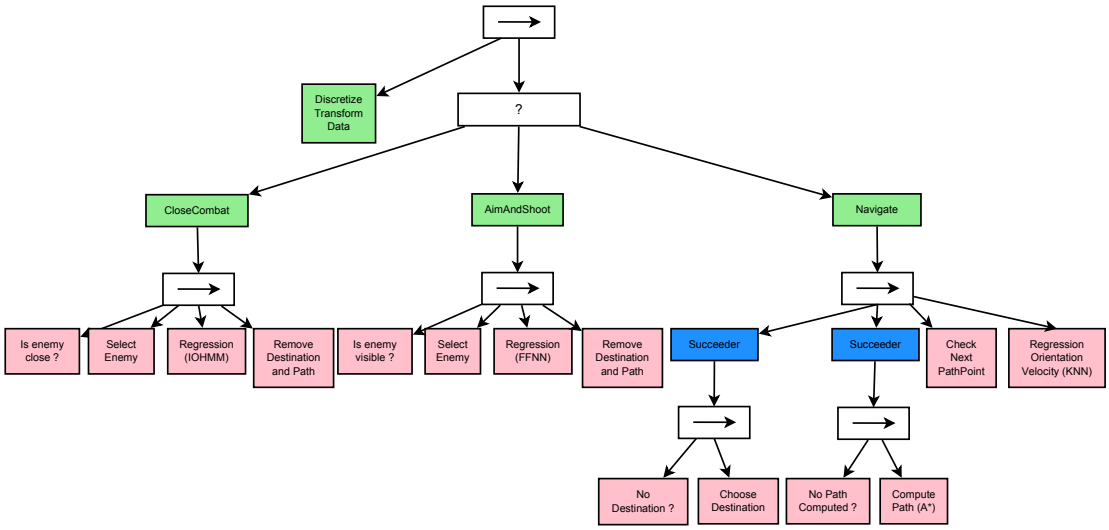

Figure 9. ORION BT of our UT3 Agent

IV. CONCLUSION AND FUTURE WORKS

The ORION model formalizes various techniques coming from research in data mining for and proposes to gather them together according to their use (data clustering, vector quantization, extraction of characteristics or prediction). Moreover, ORION permits to associate semantics to the data, enabling a better understanding of the results of data mining algorithms we want to use. In addition, ORION also provides a powerful data visualization solution. Finally, ORION also offers a behavioral model which extend the BT model. We add to this model the ability to use data mining techniques to implement complex behaviors. This model allows both online and offline learning. We illustrated the use of our model to produce AI in UT3 game using machine learning techniques. We have shown how the exploratory data analysis can provide help to make the choice of learning techniques to use. We also showed how some of the supervised and unsupervised learning algorithms can be used as part of the creation of an AI in video games. Many issues still need to be resolved before our behavioral model can automatically and completely learn both credible and effective behaviors without human intervention. The first opportunity for improvement is probably automatic identification of low-level behavior in the traces of the player. Temporal clustering is a difficult task but work done as part of the video segmentation, such as that of [10], seems to provide good results. While we have not been able to effectively adapt these techniques to our situations, it nevertheless seems to us that some results are encouraging and that it is worth pursuing. The addition of a reinforcement learning mechanism to our model seems to be a possible avenue for improvement. Taking into account the performance of the behavior seems a very useful source of information in order to obtain more convincing behavior. Finally, an automatic construction of BT could be seen as one of the last steps required for creating a behavior by using traces and without human intervention.

REFERENCES

[1] E. Achtert, H.-P. Kriegel, and A. Zimek, "ELKI: a software system for evaluation of subspace clustering algorithms," in *Scientific and Statistical Database Management*, 2008, pp. 580–585.

[2] M. Hall, H. National, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten, "The WEKA Data Mining Software : An Update," *SIGKDD Explorations*, vol. 11, no. 1, pp. 10–18, 2009.

[3] D. F. Swayne, A. Buja, and D. T. Lang, "Exploratory Visual Analysis of Graphs in GGobi," in *COMPSTAT*, no. Dsc, 2004, pp. 477–488.

[4] J. Demšar, T. Curk, A. Erjavec, v. Gorup, T. Hočevar, M. Milutinovič, M. Možina, M. Polajnar, M. Toplak, A. Starič, M. Štajdohar, L. Umek, L. Žagar, J. Žbontar, M. Žitnik, and B. Zupan, "Orange: data mining toolbox in python," *The Journal of Machine Learning Research*, vol. 14, no. 1, pp. 2349–2353, 2013.

[5] C. Vondrick, A. Khosla, T. Malisiewicz, and A. Torralba, "HOGgles: Visualizing Object Detection Features," *2013 IEEE International Conference on Computer Vision*, pp. 1–8, Dec. 2013.

[6] F. Tencé, L. Gaubert, P. De Loor, and C. Buche, "CHAMELEON: A Learning Virtual Bot For Believable Behaviors In Video Game," in *GAME'ON*, 2012, pp. 64–70.

[7] F. Tencé, L. Gaubert, J. Soler, P. De Loor, and C. Buche, "Stable growing neural gas: A topology learning algorithm based on player tracking in video games," *Applied Soft Computing*, vol. 13, no. 10, pp. 4174–4184, 2013.

[8] Y. Bengio and P. Frasconi, "An input output HMM architecture," *Advances in neural information processing systems*, pp. 427–434, 1995.

[9] J. Soler, "Orion, a generic model for data mining: Application to video games," Ph.D. dissertation, UBO, 2015.

[10] M. H. Nguyen, "Segment-based SVMs for Time Series Analysis," Ph.D. dissertation, Carnegie Mellon University, 2012.