ORION : A Generic Model and Tool for Data Mining

Cédric Buche[1], Cindy Even[1,2], and Julien Soler[2]

[1] Lab-STICC CNRS UMR 6285, ENIB
25 rue Claude Chappe, 29280 Plouzané, France
buche@enib.fr;even@enib.fr
[2] Virtualys
41 rue Yves Collet, 29200 Brest, France
julien.soler@virtualys.com

Abstract. This paper focuses on the design of autonomous behaviors based on humans behaviors observation, In this context, the contribution of the ORION model is to gather and to take advantage of two approaches: data mining techniques (to extract knowledge from the human) and behavior models (to control the autonomous behaviors). In this paper, the ORION model is described by UML diagrams. More than a model, ORION is an operational tool allowing to represent, transform, visualize and predict data ; it also integrates operational standard behavioral models. ORION is illustrated to control a bot in the game Unreal Tournament. Thanks to ORION, we can collect data of low level behaviors through three scenarios performed by human players: movement, long range aiming and close combat. We can easily transform the data and use some data mining techniques to learn behaviors from human players observation. 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.

1 Introduction

This research focuses on the implementation of behaviors to provide skillful and believable Non-Player Characters (NPCs). In this context, imitation learning is a very promising way to build such behaviors [6]. Imitation learning consist in extracting knowledge from data produced by human players in order to be able to reproduce their behaviors. Data mining offers a large range of tools that can be very useful to extract important knowledge.

Datasets obtained from human players can be of various types and forms and are used to learn behaviors. They can be time-series or not, contain quantitative and qualitative attributes, be abstract or have strong semantics. No model covers all these cases. The data semantics is generally not taken into account in different data processing and representation tools such as ELKI [1] WEKA [14], GGobi [23] or Orange Canvas [8]. Even if these tools allow to visualize data through scatter plots, histograms and other diagrams, in order to perform an

efficient analysis, it is essential to visualize the data according to their semantics, means make the connection between data and their meaning. The problem is well identified by Vondrick et al [30]. The authors present studies on object detection in an image using feature extraction techniques. Once feature extraction is done, the data becomes abstract. Investigating why a particular data is misclassified, for example, becomes extremely complex. In order to maintain understandability of data throughout the analysis process, the authors propose an image reconstruction from the extracted features. This is indeed a major problem that must be addressed. Visualization and transformations applied to the data should be consistent with their semantics. For example, given that the data represents two or three dimensional vectors, it is relevant to calculate an angle between two of them. The semantics also help the user to choose a more appropriate distance between data. It is certainly better suited to compare two RGB colors using a DeltaE distance rather than an Euclidean distance.

Existing tools suffer from limited data visualization possibility, providing numeric and categorical data only and never make the link with behavioral model. Orion tackles those issues. The Orion model proposes a generic approach to represent datasets. It also offers the possibility to perform some transformations on these datasets and to visualize them. In addition, the Orion model makes the link between data and behavioral models.

In section 2, we present the Orion model. In this section, we present the work-flow (section 2.1) and we detail the model (section 2.2). The model is described using UML diagrams as a generic data mining model (section 2.2) and a behavioral model linked to the data mining model (section 2.2). More than a model, Orion is also available as an operational tool. Orion is then a full software; it can be used as a library if the user need to define new types, algorithms or behaviours. In section 3, we illustrate the use of the Orion tool to control a virtual player in the game Unreal Tournament 3. Section 4 provides conclusion and future works.

A preliminary version of this work has been reported in [7]. This paper is an extended version, including in-depth state of the art, technical details and examples to illustrate the proposed models.

2 Orion

2.1 Orion Workflow

We offer with Orion a complete work-flow divided into two parts: a structural and a behavioral part (see figure 1).

Taking the example of a 3D video game, data are usually strongly associated to concrete concepts such as position, speed, orientations, hit points, etc. Other examples are given in the UCI[3] database where one can find datasets provided as CSV files, representing images of handwritten characters, chess positions, musical notes, etc. In Orion, the first task for the analyst is to add semantics to

[3] https://archive.ics.uci.edu/ml/datasets.html

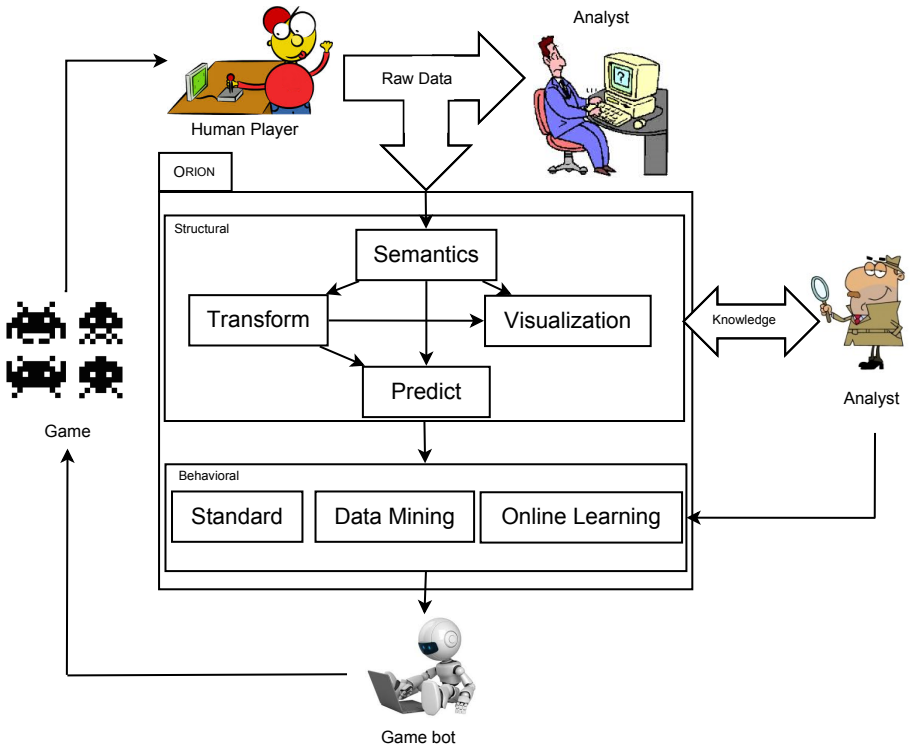


Fig. 1. ORION Workflow

the data. Then, he can transform and visualize them according to their semantics which ease the exploratory analysis. The analyst may find in the data, invariants between players for example. He can also train predictive models with these data. After this analysis step, the analyst uses the behavioral model to build a behavior using information extracted from his preliminary analysis and his predictive models. ORION's behavior model is based on a paradigm commonly used in the video game's industry. Therefore, the analyst has access to traditional tools used for IA (behavioral block). He can easily implement the knowledge identified in the data (to reproduce the invariant observed between the players for example). He can also use the predictive models that he built to reproduce a part of the behavior (like the aiming of an enemy for example).

2.2 Orion Model

Orion Structural Model

Data Representation Traditional data processing and representation tools do not take data semantics into account. [1, 14, 22, 8]. Without semantics, data be-

come abstract after the feature extraction. Visualizing data according to their semantics facilitates their analysis [30].

In our model, 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 [4] of the language (here, Java). The components that will transform or display the data therefore have access to the associated semantics. A `Type` is considered as a possible data aggregation. A three-dimensional vector is, for example, considered as the aggregation of three values. These values are necessarily convertible to real numbers. Thus, an image is composed of the width $\times$ height ($\times$ 3 if it is a color image) real numbers representing the intensity of each pixel. An enumeration is composed of an integer (and thus a real) representing its position in a list of possible values of the enumeration. This allows to process data from any type of component even if no processing or display is able to take into account the semantics of the data. The `Type` interface is also in charge of the textual representation of the data. Thus, a user can edit them with a simple text field. Finally, this interface is also responsible for data conversion to another type. For example it is possible to convert three real values into a `Vector3`. A data set is represented in the ORION model by the `Dataset` class. This class stores information on the data set, like for instance its name, or the fact that it is or not a time series, and in that case its sampling frequency, etc. This class is composed of `VariableSpec` representing the specification of each attribute in the data set and `VariableSet` representing a given data set. The `VariableSpec` consists of the name and type of the attribute, and whether the attribute is an input or an output. The `VariableSet` consist of a list containing their values referenced by the `VariableSpec` of the corresponding attribute. Appendix 1 shows the UML class diagram that implements our data representation. The `Dataset` class is also composed of an attribute `summaryData` of type `VariableSet`. This attribute is mainly used to store the result of vector quantization for example. Also, attributes can be converted from one type to another.

To illustrate data conversion, we present in Figure 2 conversion process of the Optitracks database available on the UCI database . This dataset represents images of handwritten digits. The original image was made up of two colors and had a resolution of 32x32 pixels. Each image was cut into blocks of 4x4 pixels. The dataset consists of 65 attributes. The first 64 attributes represent the number of black pixels in each block of 4 pixels by 4. The 65th represents the class of the data (a digit). After importing the csv le into the tool, the process is made of five steps: (a) selection of the attributes to be converted, (b) specification of the parameters of `VariableSpec` into which the data will be converted (the name of the attribute, input or output), (c) type selection and

[4] Reflection provides information about the class to which an object belongs and also the methods of that class which can be executed by using the object.

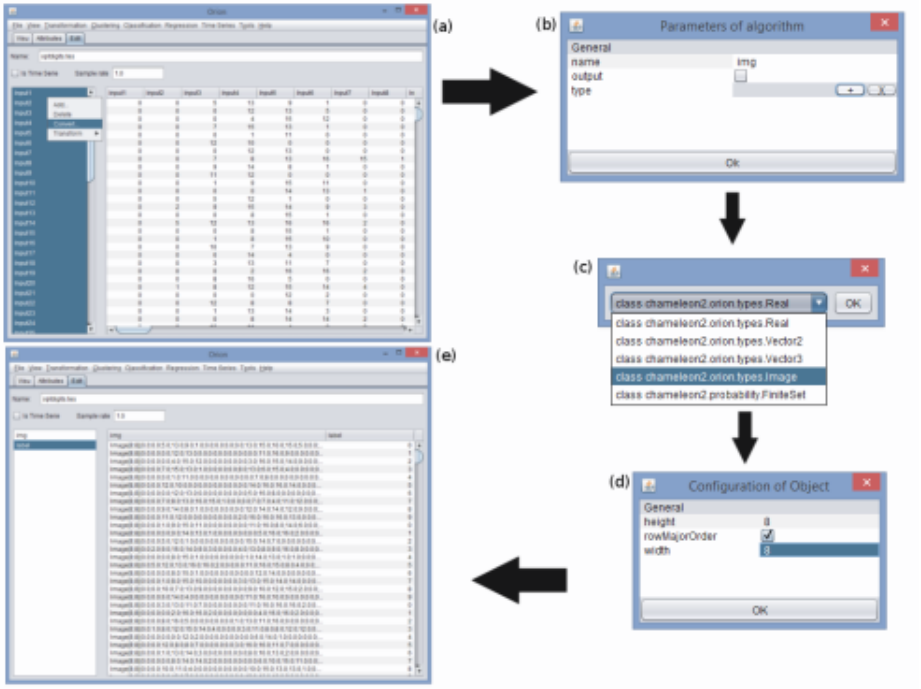


Fig. 2. Type Conversion process for the Optitrack Database in ORION.

optionally (d) configure the parameters of this type (here it is the image type, so we need to specify its dimensions). Step (e) shows the result of the conversion of the first 64 attributes in an attribute (which was the type Real) to a type Image and the 65th `FiniteSet` type.

Data Transformation In order to perform data transformations, the generic `DataTransform` interface must be implemented. The latter simply has access to the `Dataset` and a list of `VariableSpec` to process and must implement the `DataTransform.transform()` method. Allowed transformations can vary significantly. They generally involve attributes modification, addition of attributes or `summaryData`, etc. The downside of this simplicity is that the GUI for manipulating this model has no information on the type of transformation performed by the `DataTransform`. In order to overcome this problem, various sub-interfaces are proposed like `ClusteringTransform`, `DiscretizerTransform`, `VectorQuantization`, etc. These interfaces are only present to classify data transformation methods into several categories. An algorithm can also implement several of these interfaces. Appendix 2 shows the UML class diagram implementing the transformation model of ORION. Data transformation algorithms implemented in ORION include Principal Component Analysis (PCA), Kernel PCA [21], Clas-

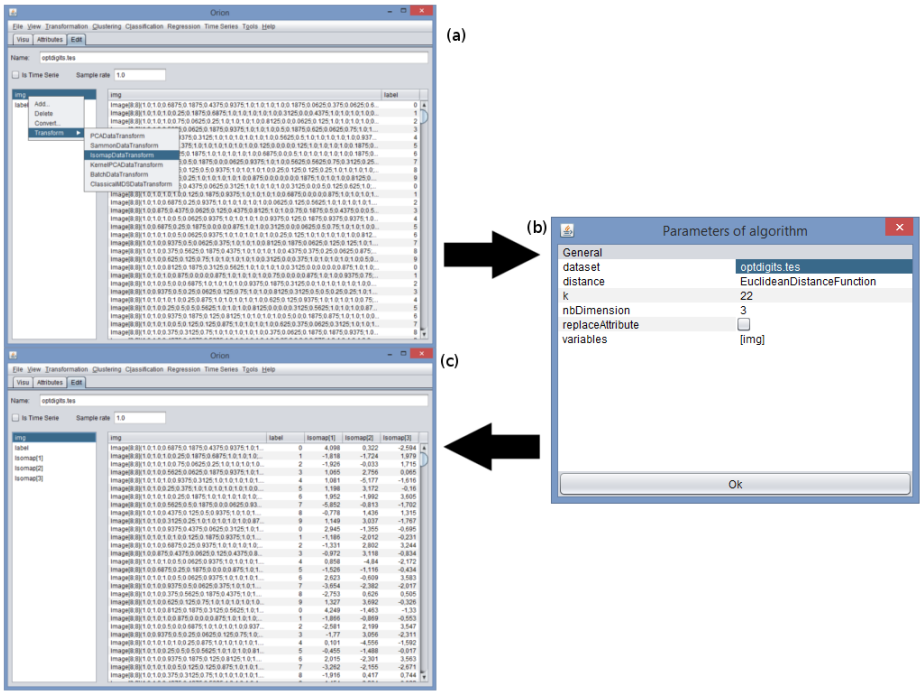


Fig. 3. Transformation process for the optitrack database (Isomap).

sical Multidimensional Scaling [28], Isomap [26], Sammon projection [20], and many clustering algorithms (KMeans, DBSCAN [9], OPTICS [2], Gaussian Mixture Model, Growing neural Gas Networks [12], etc). This list can be extended by implementing the `DataTransform` interface.

Using the example of UCI's Optitracks database, original images used to build the dataset are not available because each image is represented by the sums of black pixels in a block of 4 by 4 pixels. However, it is possible to reconstruct 8x8 grayscale images with this data. To do this, we need to normalize each value between 0 and 1 (1 being white and 0 black). ORION provides a class `BatchDataTransform` to apply an expression to each value of the data set. In addition, it may be useful to produce multidimensional scaling on data. Figure 3 illustrates the multidimensional scaling execution process carried out by the Isomap algorithm through our tool. Data normalization was previously carried out via a similar process.

Data Visualization In the ORION model, every element in charge of data visualization implements the `DataViewer` interface. This interface is configurable by the enumeration `DisplayRange` to define the data to be represented (the current datum, the current selection or all data). Two display types are present: the 2D

and 3D rendering interfaces implements the `GraphicalViewer` interface while the various components displaying data, attributes by attributes, implement the `AttributeViewer` interface. The `GraphicalViewer` interface is implemented by the classes `Viewer2D` and `Viewer3D`. This interface is not intended to be extended by other classes. A `GraphicalViewer` is composed of a `Renderer`. The `Renderer` interface provides methods (`draw2D` and `draw3D`) to draw a datum on the display panel and a method to provide a dialog panel to configure the `Renderer`. For example, the implementation of `PointRenderer` allows to represent data as points. The position of the point can be defined by an attribute of type `Vector2` or `Vector3` or three `Real` attributes. Similarly, its size, shape or color can be specified as fixed, varying with an attribute value or with its position in the time series. This list can be extended by implementing the `Renderer` interface. Figure 4a and 4b show the display of UCI's Optitracks dataset as well as the display of NAO robot's skeleton taken from a goalkeeper agent in the context of the RoboCup 3D Soccer Simulation Competition. The interface `AttributeViewer` consists of a list of attributes to display. ORION implementations of `AttributeRenderer` include Radar and parallele plots, histograms, line charts, etc. This list can be extended by implementing the `AttributeViewer` interface. Appendix 3 shows the UML class diagram that implements the visualization model of ORION.

Prediction Prediction is an important task of our data-based learning approach. In order to achieve prediction and therefore improve learning functions, the ORION model offers a level of abstraction for all methods of classification and regression. Appendix 4 illustrates this model. The `FunctionLearner` interface represents a regression or classification algorithm. This interface is composed of the explanatory variables and the dependent variables. It has the methods `predict` and `train` that are respectively used to predict the dependent variable based on explanatory variables and to train the algorithm based on sample data. The `Classifier` and `Regressor` interfaces extend `FunctionLearner`.

Several interfaces extend `Classifier` to separate different characteristics of these algorithms. Indeed, many classification algorithms are binary classifiers (although most classification methods being initially binary have seen multiclass extensions proposed like for SVM and AdaBoost). In addition, many algorithms do not perform a peremptory classification but instead return a probability of belonging to each class. Interfaces `BinaryClassifier`, `ConfidenceClassifier` and `BinaryConfidenceClassifier`, therefore express these features. Regression and classification algorithms implemented in ORION include KNN, Quinlan C4.5 [18], SVM [29], Feed Forward Neural Networks [4], Hidden Markov Model [31], Naive Bayesian Networks [19], AdaBoost [11], etc. This list can be extended by implementing the `Classification` or `Regressor` interface.

Orion Behavioral Model ORION's behavioral model is an extension of behavior trees (BT) [15]. This section presents the concepts underlying this model and the extensions made in ORION. Finite State Machine (FSM) and BT are the

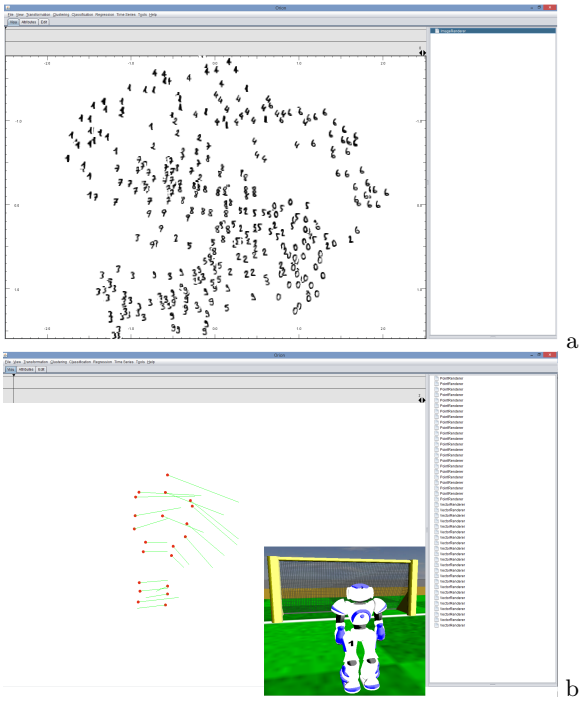


Fig. 4. UCI Optitracks dataset visualization (a). NAO Skeleton dataset visualization with ORION(bones positions and speed). The picture at the bottom right, that is not part of ORION GUI, illustrate NAO within the simulation (b)

architectures the most used in the game industry. FSM has many shortcomings over BT. Transitions from one state to another must be explicit, resulting in excessive complexity when the number of states is large. As a state machine is not a Turing machine, some behaviors are impossible to describe with FSM. FSMs are event oriented (events are needed to cause state changes) as opposed to goal oriented, which makes it difficult to implement goals with priority management among them for example. BT do not have these problems and for these reasons we believe that BT is better suited for the development of an AI solution in video games.

Standard Behavior Trees BTs originate from the computer game industry as a powerful tool to model the behavior of NPC, used by video games such as Halo, Bioshock, and Spore. BT have the ability to create complex tasks composed of simple tasks, without worrying how the simple tasks are implemented. BT model is represented by a tree structure where each node of the tree represents a goal and each child of a node represents a sub goal. When the nodes are executed, they return their current status (Success, Fail or Running). In its most simple implementations, at each time step, the tree is traversed from the

root node spreading deep into the branches to the leaves. The leaves of the trees are mainly of two types: condition and action. They are used to condition the execution of actions or goals. Action nodes perform operations on either the agent environment or its internal state. Composite (Non Leaf) nodes are used to control the execution of their child nodes. There are mainly two types: Sequence and Selector. A Selector node executes its child nodes until one of them returns Success. It allows to reach a goal by testing several solutions. A Sequence node executes its child nodes until they all return Success. It allows to perform a sequence of actions to achieve a goal. Decorator nodes have only one child node. These nodes are used to add execution control features like *while* or *for* statement equivalent.

Orion Behavior Tree Extensions We saw how BT is endowed with many qualities to achieve an AI in video games, therefore we decided to extend this model. First, without losing the potential of BT to add *ad hoc* code features, we use as an execution context of BT, a `VariableSet`. This context 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 assigned to a type, each behavior can control its consistency and performs introspection on these data. We propose to extend BT to be able to use the data mining techniques. We presented earlier, the structural model encompassing the various tasks achievable by data mining techniques. Here we present the integration of the structural model into the behavioral model.

The most important functionality we want to add to BT is the possibility to use behaviors that have been learned from a dataset. To do this, we propose to add node types to the BT model. First, we add a particular type of node composed of a `FunctionLearner` to predict an output variable based on input variables. 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. Appendix 5 presents our model of BT.

We want the construction of BT to be 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 (by associating a `FunctionLearner` previously trained with data to a `FunctionBehavior` for example). But online training requires to integrate other mechanisms. Indeed, many additional problems arise when trying to learn online. 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 ticks for example. BT's formalism allows to simply achieve this kind of behavior. We decide to add to Orion some nodes dedicated to online learning. These nodes allow to store online data within a dataset and to launch the training over those online datasets. Appendix 6 illustrates the implementation of these features in the Orion model.

The typical use of these units in order to perform online learning is shown in figure 5. In this example, the left side of the tree allows the execution of the behavior performed once learning is done and the right part allows directed learning and implement a default behavior if learning has not been achieved. Thus, if learning has not yet been done, the condition node on the left (*Trained?* in the figure) returns `Fail` and the right node is executed. The current context is then stored in a `Dataset`. If the number of data in the `Dataset` is sufficient or an event occurs in the game for example, learning is carried out in several steps. First, the `Dataset` is divided into n sequences by a data processing algorithm (usually a time series clustering algorithm). These sequences belong in this example to two different types. Then, the three `FunctionLearner` nodes associated both with the execution part (the three nodes references *Classifier 1*, *Regression 1* and *Regression 2* in the figure 5) and the Train nodes are trained with the `Dataset`. Finally, the execution context is changed to indicate that learning was achieved (Trained = True). The following tick therefore executes the train behavior (left side of the tree). As long as learning is not finished, the default behavior (implemented with *ad hoc* features for example) is executed.

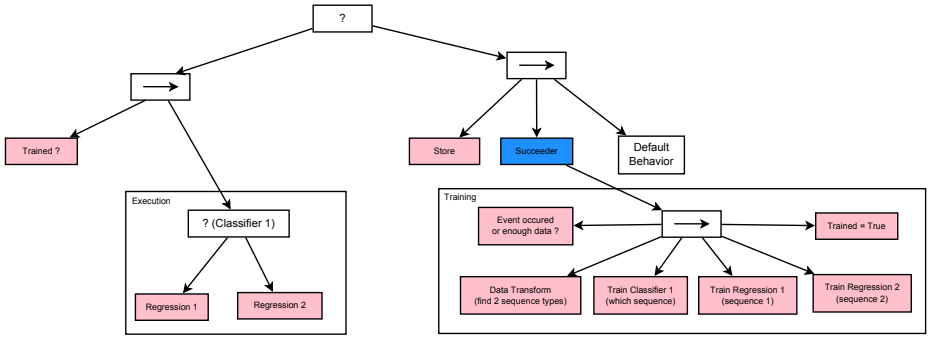


Fig. 5. Typical use of ORION Behavior Trees for online learning

3 Application

Different approaches have been explored by the scientific community to design a solution for imitation learning of behaviors. Tencé et al. [24] suggested a probabilistic approach. Connectionists [27] and memory based [5, 32] approaches have also been considered. In the video game Black & White (Lionhead Studio, 2001), a decision tree [10] was used to implement a creature of the game. The problem with most of those techniques is that everything is encapsulated in a black box, data from human tracking are not analyzed before entering the imitation learning algorithms. Thus, important data are missed and insignificant data are

considered. In order 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 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 developed by Epic Games and released in 2007. In this 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. They may be weapons, ammunition, shields, first aid kits etc.

First we describe how by tracking a human player we can collect data (section 3.1) that can be easily visualized and transformed with ORION (section 3.2). We continue by explaining how by predicting the data with ORION we can learn low level behaviors (section 3.3). Next, we show how our behavioral model uses the learned behaviors and how we implement features of the Bot behavior which we have not been able to learn automatically (section 3.4). Finally, we provide results (section 3.5).

An overview of different components of the ORION model used in this application is presented in Figure 6.

3.1 Human Players Tracking

Raw Data When observing a player evolving in the game with GameBots, we have access to much information about him and his environment. We choose to collect this unprocessed information in a CSV file. Raw data are processed with ORION as explained later. The information collected is summarized in table 1.

Player position
Player orientation (pitch and yaw, roll is always 0)
Player velocity
Current weapon name
Life points
Remaining ammunition
Shoot (is the player shooting?)
Number of enemies visible
Enemy positions (the 3 closest)
Enemy orientations (the 3 closest)
Enemy velocities (the 3 closest)
Number of seen items
Item positions (the 3 closest)
Number of seen navigation points
Navigation point positions (the 3 closest)

Table 1. Data collected

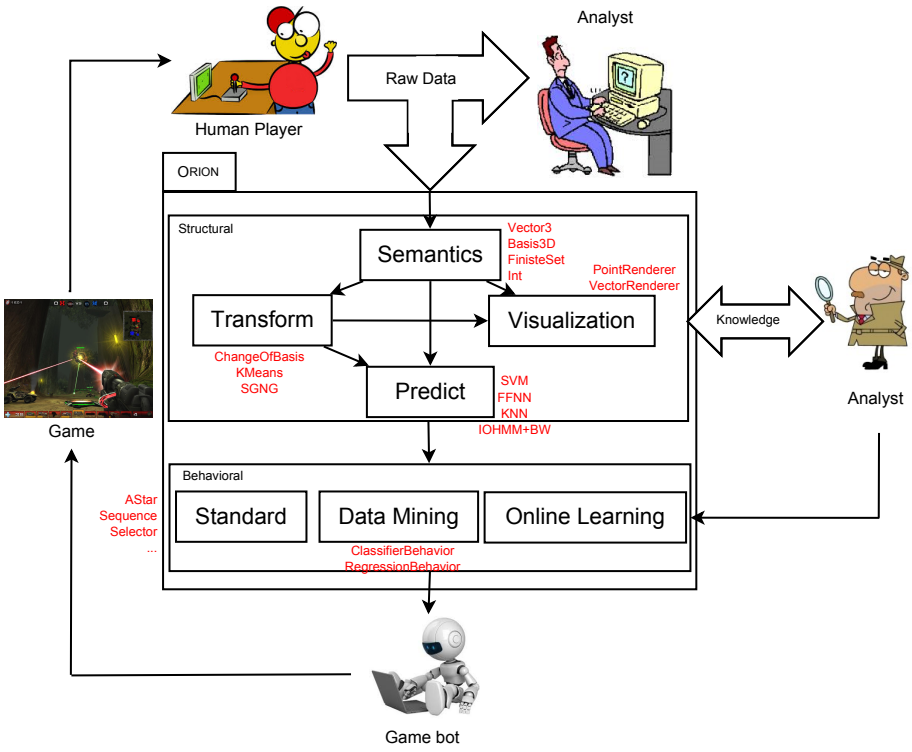


Fig. 6. ORION Workflow for UT3

Data Collection In order to create datasets containing only sequences where a player is performing a specific task, we conducted scripted gaming sessions. A human player performs the tasks defined in our scenarios and game traces are recovered. Three scenarios are proposed:

1. 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.
2. 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.
3. Close Combat: The player is in a small room with an enemy and must dodge attacks while firing at the enemy.

3.2 Data Transformation and Visualization

Raw data does not allow to easily perform analysis. As ORION adds semantics to the data, we make use of it to facilitate an exploratory analysis. In UT3, most

data are spatial locations of elements in the game (players, items, etc). ORION implements several types associated with locations in three-dimensional space and allows their visualization. Using these features, we are able to rebuild the game play in ORION as shown in figure 7.

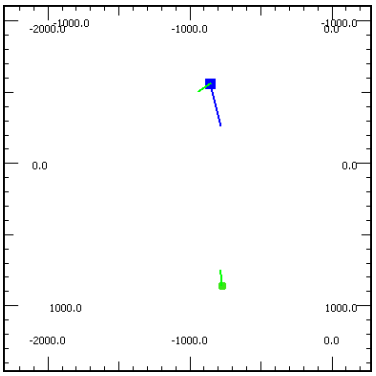


Fig. 7. UT3 game reconstruction in ORION

Spatial location from raw data does not allow us to effectively reproduce the observed behavior. The position of the player, for example, should not be used as input data of a learning algorithm. Doing so, the learned behavior would only be able to reproduce actions at the specific locations where they were observed. Therefore, we need to transform all data related to location in the environment to the local coordinates of the player. A type implemented in ORION is used to represent a position and orientation in space. This type is implemented through a matrix of homogeneous coordinates and corresponds, from a mathematical point of view, to a 3D orthonormal basis. Spatial transformations become possible in ORION when data are of this type. Such transformations are mainly changes of coordinates. We use this feature of ORION to express all the data related to a location in the coordinates system of the player.

Finally, we wish to use algorithms that accept only discrete data as inputs. Therefore we perform a discretization of data using the K-Means algorithm [5] [16].

3.3 Prediction

Earlier in this article we presented a data collection of low level behavior samples through scenarios previously performed by human players. The problem now is to replicate these behaviors individually. Thanks to ORION, we integrated easily

[5] Data discretization is a pre-processing method that reduces the number of values for a given continuous variable by dividing its range into a finite set of disjoint intervals, and then relates these intervals with meaningful labels

various algorithms : KNN for navigation, FFNN for aiming and IOHMM for close combat.

Movement For the movements of the Bot in the environment we use the CHAMELEON model proposed by Tencé et al. [24] that suggests to learn the environment with a vector quantization method called Growing Neural Gas (GNG) [13]. The learning environment algorithm rebuilds the navigation graph automatically. This information allows to know which places are reachable in the environment and to compute a path. The model has been modified to be able to learn continuously on a player without growing indefinitely but being able to extend itself if the teacher begins to use a new part of the environment. This model is called Stable Growing Neural Gas (SGNG) [25].

However, a navigation graph is not sufficient to reproduce the movements of a real player in this environment. A human player with a minimum of experience in the game has, for example, a natural tendency to strafe (move sideways) when reaching the corner of a hallway which maximizes his field of vision and allows him to quickly see if an enemy is present in the corridor. To overcome this issue we propose to submit to the SGNG, besides the player's position, his speed and direction. However, this creates an additional difficulty: one of the steps 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, using the Euclidean distance to find the closest points is totally justified. But when data additionally contains speed and direction, choosing a distance to obtain the desired result becomes more difficult. 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 propose to perform a re-scaling by feeding the SGNG with the following distance:

$$\text{Distance}((\boldsymbol{p1}, \boldsymbol{o1}, \boldsymbol{v1}), (\boldsymbol{p2}, \boldsymbol{o2}, \boldsymbol{v2})) = d(\boldsymbol{p1}, \boldsymbol{p2}) + \alpha d(\boldsymbol{o1}, \boldsymbol{o2}) + \beta d(\boldsymbol{v1}, \boldsymbol{v2})$$

where d is the regular Euclidean distance and α and β are scaling factors. This re-scaling is very simple to perform thanks to the data transformation features of ORION.

In order 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 8 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.

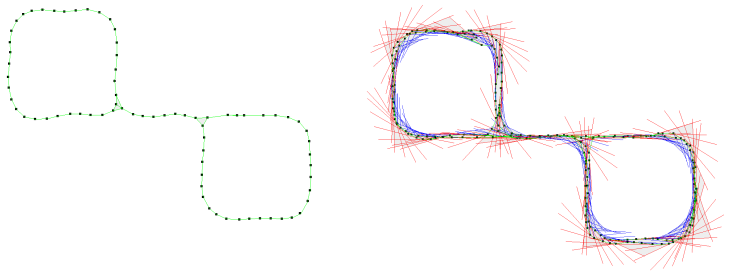


Fig. 8. SGNG Results on UT3 (Left: CHAMELEON, Right: ORION)

To reproduce a more realistic navigation behavior, we use the vector quantization as explained above with the algorithm K-Nearest Neighbors (KNN). Figure 9 illustrates the principle. In this figure, 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. Once again we need to perform a data scaling and decide to use the same distance 3.3. However, the parameters α and β must be selected differently: knowing the position and the desired direction of the agent, we try to get the speed and direction that the agent should adopt. So this time, we must use the velocity vector (indicating the desired direction) and not the orientation, to compute this distance. Therefore, we choose $\alpha = 0$. In order to set the parameter β, we apply the following requirement: two data should be far away if their velocities are in opposite directions, even if their position is really close.

In figure 9, 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.

Long Range Aiming Aiming is one of the most important elements of FPS games. There are many techniques dedicated for that, and experienced players use advanced ones. The performance of the players in this activity largely determines their overall performance in the game. We chose to make it easier to reproduce this behavior, by learning it from scripted game play in which the player does not move and faces a single enemy. But an experienced player uses, at least partially, the movement for aiming. It is a very commonly seen technique to strafe from side to side without moving the mouse to refine the target. So we will not be able to reproduce the behavior of an experienced player. However,

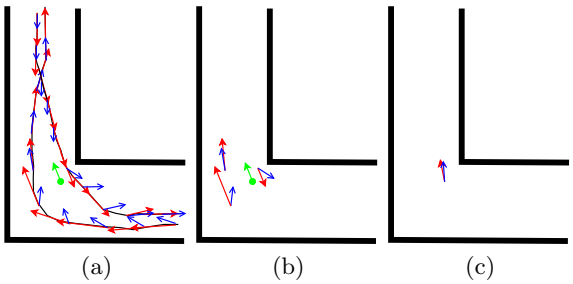


Fig. 9. Movement regression principle in UT3

beginners and average players mainly perform a static aiming. So, we focus on reproducing the aiming behavior of such players.

In order to better choose a method to reproduce this behavior, we conduct an exploratory analysis. 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.

Close Combat Close combat is quite common in UT3. The arenas are often made of tortuous corridors, where one often encounters enemies at turning points. Some weapons are more appropriate for this kind of confrontation. Reflexes are necessary but an unpredictable behavior is the key to make the task of the enemy harder.

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 in the context of the reproduction of this behavior. 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.

We tried to reproduce this behavior using standard regression algorithms that we have used for the aiming behavior. However, we have not been able to obtain convincing results with this method. This is not surprising because the exploratory analysis shows that there was 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 (IOHMM) [3] to reproduce this behavior. A probabilistic model is indeed more suited to this type of data because it provides a certain unpredictability. We used a network very similar to the CHAMELEON model, applying the recommendations on variable dependencies.

3.4 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 ([removed for blind review]) for a complete description of the model.

Figure 10 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

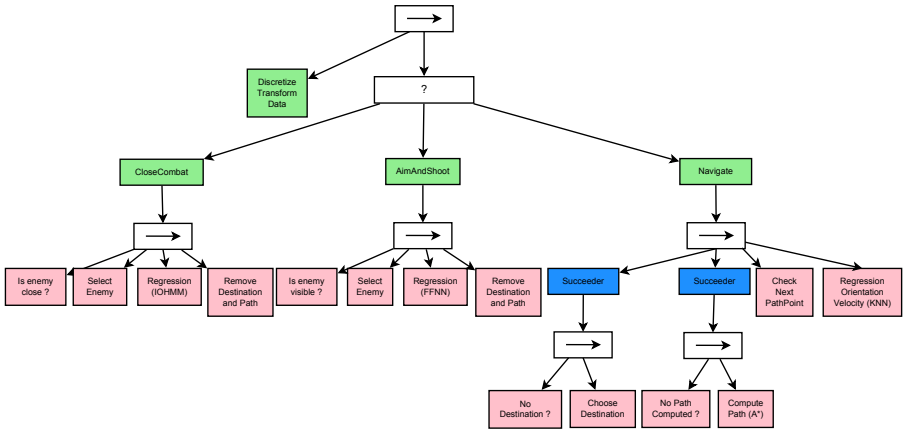


Fig. 10. ORION BT of our UT3 Agent

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.

3.5 Results

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

Subjective analysis Our 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.

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. Participants do not have background knowledge about UT3 or UT-like games. 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.

4 Conclusion and Future Works

We presented in this paper the Orion model. Orion permits to associate semantics to the data. This semantics is essential to enable a better understanding of the results of data mining algorithms we want to use. Orion also provides a powerful data visualization solution. Traditional charts for univariate or multivariate analysis of the data set (histogram, radar and parallel plot, etc.) are implemented and can be extended. A generic representation model allows the data to be displayed in 2 or 3 dimensions. Data visualization primitives such as points, vectors, images, texts or graphs are proposed and the model allows extensions. This allows us to display, in a generic tool, most of the data available on the UCI database.

The Orion model formalizes various techniques coming from research in data mining and proposes to gather them together according to their use. Our tool, therefore, proposes to perform tasks as diverse as data clustering, vector quantization, extraction of characteristics or prediction. The implementation of our model provides a generic approach for data analysis and data mining in the manner of WEKA or ELKI. The main contributions of our tool in relation to the latter are the management of semantics and the greater possibility of visualization. The number of available algorithms is smaller than the ones provided with WEKA but may easily be extended.

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. Unsupervised learning, for example, helped us to automatically reconstruct the environment and the information on how the players move in it. Supervised learning has allowed us to reproduce relatively simple actions that can be used as primitives to implement more elaborate behaviors. While the behaviors conducted with our model are neither more efficient nor more believable than behavior implemented with far more widely used methods in this industry, their implementation shows that learning methods from the world of research can help the design of AI in video games, easing the designer's workload and providing it with relevant information on human player behavior.

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 [17], 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.

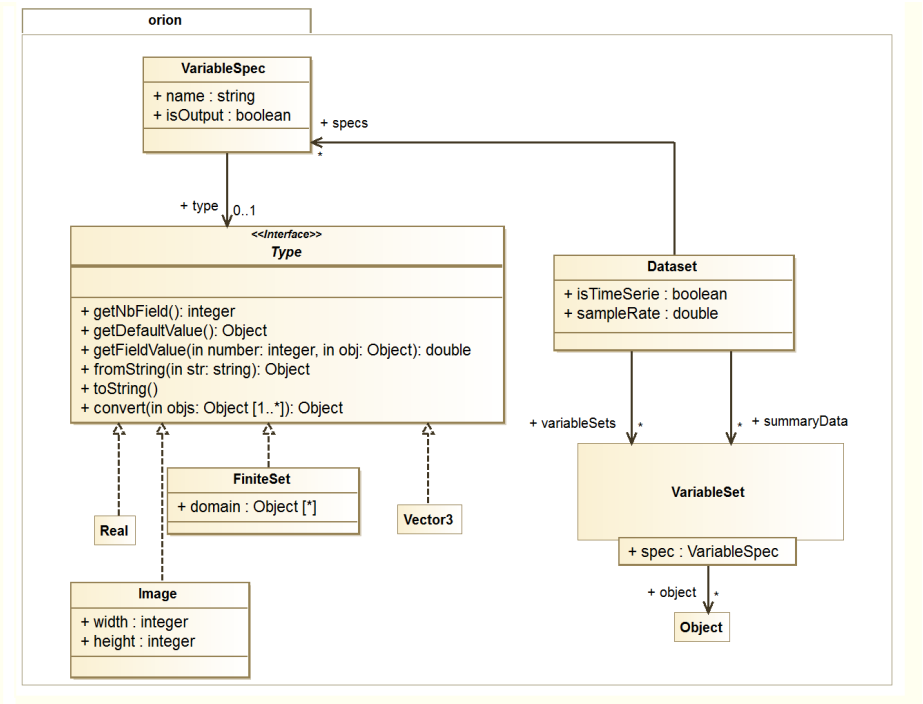
References

1. Achtert, E., Kriegel, H.P., Zimek, A.: ELKI: a software system for evaluation of subspace clustering algorithms. In: Scientific and Statistical Database Management. pp. 580–585 (2008)
2. Ankerst, M., Breunig, M.M., Kriegel, H.P., Sander, J.: OPTICS: ordering points to identify the clustering structure. ACM SIGMOD Record (1999)
3. Bengio, Y., Frasconi, P.: An input output HMM architecture. Advances in neural information processing systems pp. 427–434 (1995)
4. Bengio, Y., Frasconi, P.: Input-output HMMs for sequence processing. Neural Networks, IEEE Transactions on **7**(5), 1231–1249 (1996)
5. Bentivegna, D.C., Atkeson, C.G., Cheng, G.: Learning from observation and practice using primitives. In: AAAI 2004 Fall Symposium on Real-life Reinforcement Learning (2004)
6. Buche, C.: Adaptive behaviors for virtual entities in participatory virtual environments. Habilitation à diriger des recherches, Université de Bretagne Occidentale - Brest (2012)

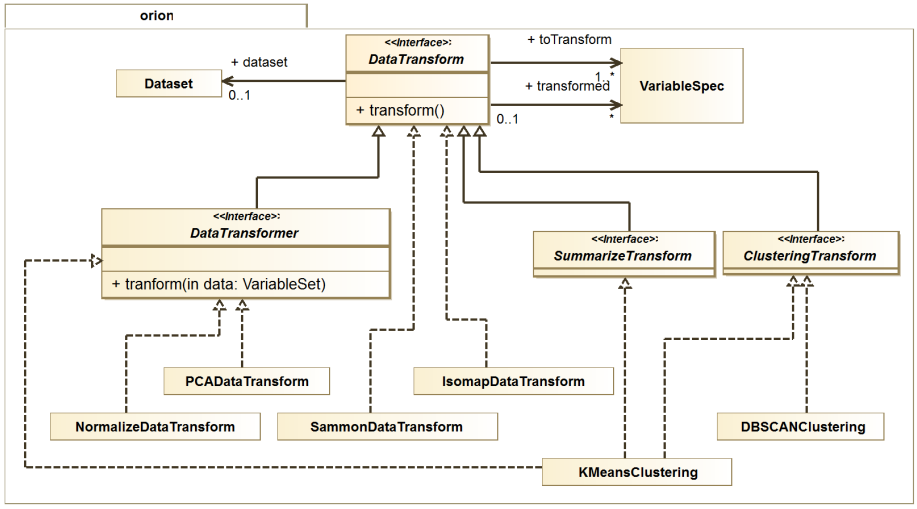
7. Buche, C., Even, C., Soler, J.: Autonomous virtual player in a video game imitating human players: the orion framework. In: International Conference on Cyberworlds. pp. 108–113. IEEE (2018)
8. Demšar, J., Curk, T., Erjavec, A., Gorup, v., Hočevar, T., Milutinovič, M., Možina, M., Polajnar, M., Toplak, M., Starič, A., Štajdohar, M., Umek, L., Žagar, L., Žbontar, J., Žitnik, M., Zupan, B.: Orange: data mining toolbox in python. The Journal of Machine Learning Research **14**(1), 2349–2353 (2013)
9. Ester, M., Kriegel, H.P., Sander, J., Xu, X.: A density-based algorithm for discovering clusters in large spatial databases with noise. In: Proceedings of the 2nd International Conference on Knowledge Discovery and Data mining. pp. 226–231 (1996)
10. Evans, R.: The Use of AI Techniques in Black & White (2001)
11. Freund, Y., Schapire, R.E.: A Decision-Theoretic Generalization of On-Line Learning and an Application to Boosting. Journal of Computer and System Sciences **55**(1), 119–139 (Aug 1997)
12. Fritzke, B.: A growing neural gas network learns topologies. Advances in neural information processing systems **7**, 625–632 (1995)
13. Fritzke, B.: Growing grid - A self-organizing network with constant neighborhood range and adaptation strength. Neural Processing Letters **2**(5), 9–13 (1995)
14. Hall, M., National, H., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., Witten, I.H.: The WEKA Data Mining Software : An Update. SIGKDD Explorations **11**(1), 10–18 (2009)
15. Lim, C.U., Baumgarten, R., Colton, S.: Evolving behaviour trees for the commercial game DEFCON. Applications of Evolutionary Computation pp. 100–110 (2010)
16. Lloyd, S.P.: Least squares quantization in PCM. IEEE Transactions on Information Theory **28**(2), 129–137 (Mar 1982)
17. Nguyen, M.H.: Segment-based SVMs for Time Series Analysis. Ph.D. thesis, Carnegie Mellon University (2012)
18. Quinlan, J.R.: Improved use of continuous attributes in C4.5. Journal of Artificial Intelligence Research **4**(1), 77–90 (1996)
19. Russel, S., Norvig, P.: Artificial Intelligence: A Modern Approach. Prentice Hall, 3rd editio edn. (2009)
20. Sammon, J.W.: A Nonlinear Mapping for Data Structure Analysis. IEEE Transactions on Computers **C-18**(5), 401–409 (May 1969)
21. Schölkopf, B., Smola, A., Müller, K.R.: Nonlinear component analysis as a kernel eigenvalue problem. Neural computation **10**(5), 1299–1319 (1998)
22. Swayne, D.F., Buja, A.: Exploratory visual analysis of graphs in GGobi. In: COMPSTAT 2004 – Proceedings in Computational Statistics. pp. 477–488 (2004)
23. Swayne, D.F., Buja, A., Lang, D.T.: Exploratory Visual Analysis of Graphs in GGobi. In: COMPSTAT. pp. 477–488. No. Dsc (2004)
24. Tencé, F., Gaubert, L., De Loor, P., Buche, C.: CHAMELEON: A Learning Virtual Bot For Believable Behaviors In Video Game. In: International Conference on Intelligent Games and Simulation (GAMEON'12). pp. 64–70 (2012)
25. Tencé, F., Gaubert, L., Soler, J., De Loor, P., Buche, C.: Stable growing neural gas: A topology learning algorithm based on player tracking in video games. Applied Soft Computing **13**(10), 4174–4184 (Oct 2013)
26. Tenenbaum, J.B., de Silva, V., Langford, J.C.: A global geometric framework for nonlinear dimensionality reduction. Science (New York, N.Y.) **290**(5500), 2319–23 (Dec 2000)

27. Thurau, C., Sagerer, G., Bauckhage, C.: Imitation learning at all levels of game-AI. In: Proceedings of the international conference on computer games, artificial intelligence, design and education. pp. 402–408 (2004)
28. Torgerson, W.S.: Multidimensional scaling: I. Theory and method. Psychometrika **17**(4), 401–419 (1952)
29. Vapnik, V., Golowich, S.E., Smola, A.: Support vector method for function approximation, regression estimation, and signal processing. In: Advances in Neural Information Processing Systems 9. pp. 281–287 (1996)
30. Vondrick, C., Khosla, A., Malisiewicz, T., Torralba, A.: HOGgles: Visualizing Object Detection Features. 2013 IEEE International Conference on Computer Vision pp. 1–8 (Dec 2013)
31. Welch, L.R.: Hidden Markov Models and the Baum-Welch Algorithm (2003)
32. Yamamoto, K., Mizuno, S., Chu, C., Thawonmas, R.: Deduction of Fighting-Game Countermeasures Using the k-Nearest Neighbor Algorithm and a Game Simulator. Ice.Ci.Ritsumei.Ac.Jp (April), 0–4 (2014)

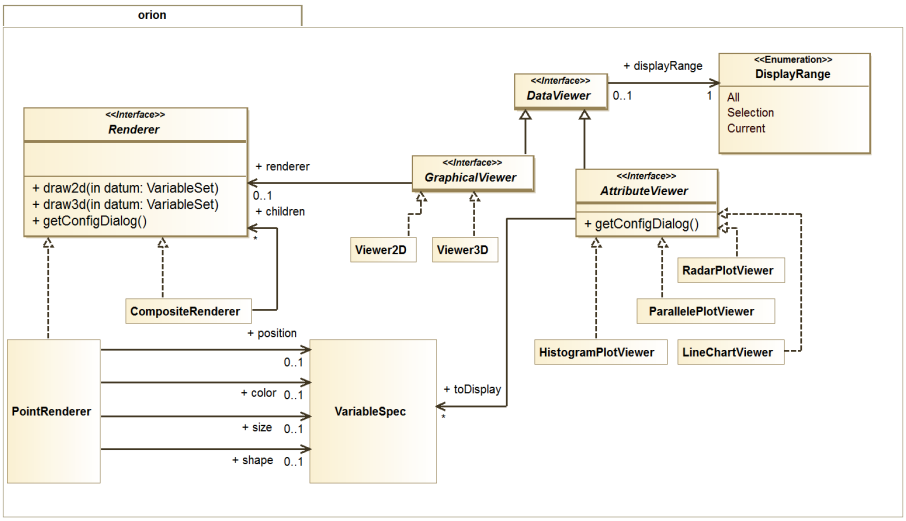
5 Appendix



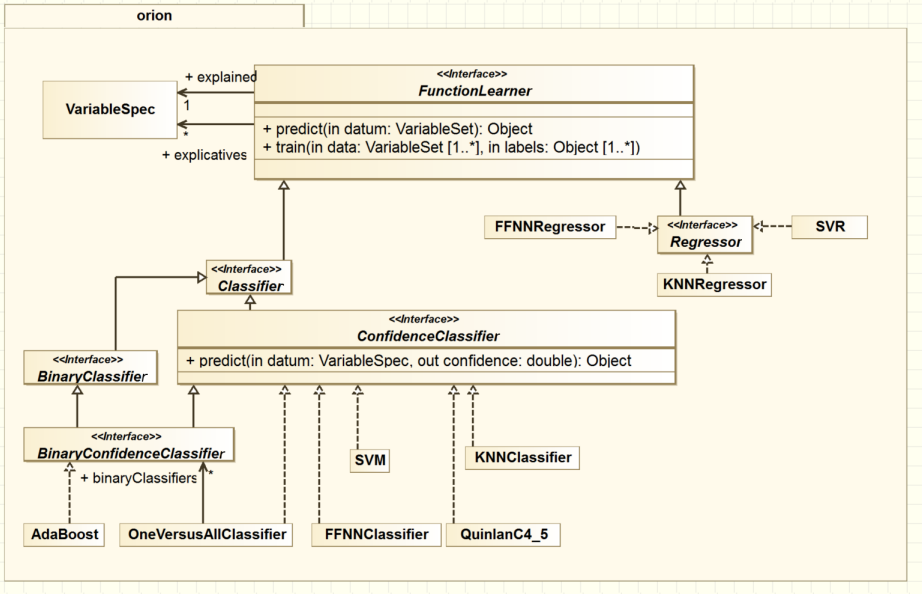
Appendix 1. ORION data model



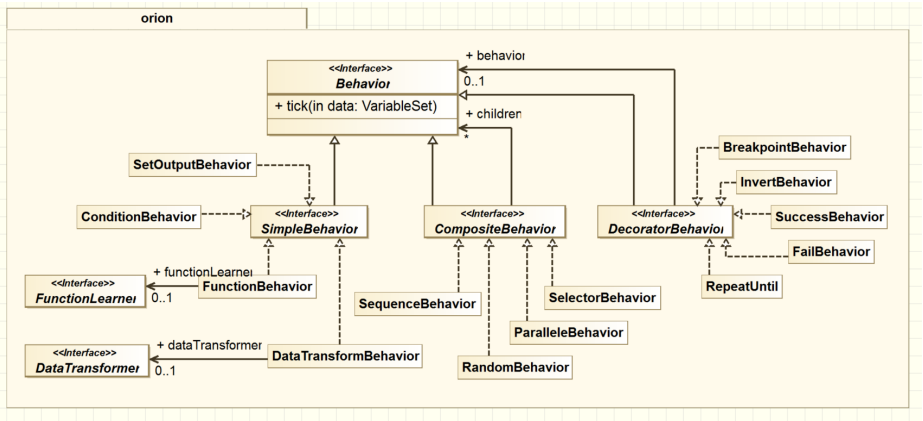
Appendix 2. ORION data transformation model



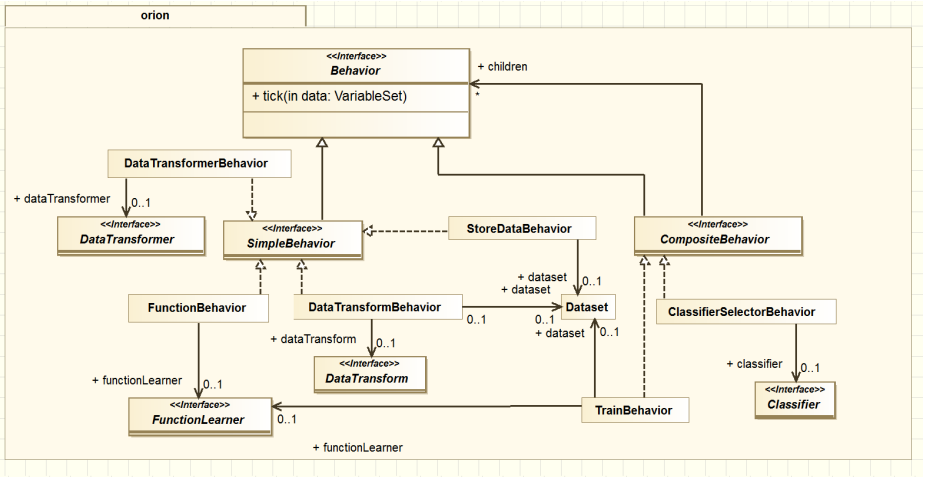
Appendix 3. ORION Data visualization model



Appendix 4. ORION prediction model



Appendix 5. ORION data mining behaviors



Appendix 6. ORION online learning model