Artificial Intelligence in the Game Design Process 2: Papers from the 2013 AIIDE Workshop (WS-13-20)

# A Generic Method for Classification of Player Behavior

**Marlon Etheredge, Ricardo Lopes, Rafael Bidarra**
Computer Graphics and Visualization Group, Delft University of Technology, The Netherlands
Marlon.Etheredge@gmail.com, R.Lopes@tudelft.nl, R.Bidarra@tudelft.nl

**Abstract**

Player classification allows for considerable improvements on both game analytics and game adaptivity. With this paper we aim at reversing the *ad-hoc* tendency in player classification methods, by proposing an approach to player classification that can be integrated across different games and genres and is particularly suited to be used by game designers. This paper describes our generic method of interaction-based player classification, which consists of three components: (i) intercepting player interactions, (ii) finding player types through fuzzy cluster analysis and (iii) classification using Hidden Markov Models (HMM). To showcase our method we developed Blindmaze, a simple web-based hidden maze game publicly available, featuring a bounded set of interactions. All data collected from a game is interaction-based, requiring minimal implementation effort from the game developers. It is concluded that our method makes player classification even more available by making it generic and reusable across different games.

## Introduction

The ability to track player's behavior can greatly aid game development. It not only allows you to assess whether games were experienced as intended by designers, but it can also serve as a basis for adaptive games (Lopes and Bidarra 2011). Recently much research has been done in game adaptivity, *i.e.* adjusting the properties (*e.g.* of the player or non-playing characters), content (objects, and even complete worlds) or other aspects of a game to the player behavior. This has brought to game development the challenge of both modeling the player behavior and adapting content.

Using game analytics in game development has thus evolved beyond its initial goal of assessing behavior and performance towards better game design. It is also needed to model player behavior on-the-fly for a variety of games which can be made adaptive. Furthermore, there is an inherent need to still include game designers in this process (Lopes, Tutenel, and Bidarra 2012). For these adaptive games to work, game design must shift from designers authoring *static* games with fixed content to designers authoring *dynamic* systems with player modeling and automatic generation. This includes, among other challenges, the inclusion of player modeling techniques in the game development cycles of different games, and its use by game designers.

In this paper, we investigate a method towards this goal. We propose a generic interaction-based system for automatic player behavior classification. This method, that uses fuzzy cluster analysis and Hidden Markov Models, can be applied to analyze player behavior for whatever purposes, including to model players in adaptive games by classifying their behavior. Our main contribution lies on the generic nature of our method, as it can be applied by game designers to many different games and genres.

Following this introduction, we will first examine previous work related to player modeling. Second, we will introduce various concepts that are relevant for our method. Third, we introduce the various components that make up our method. Fourth, we will introduce Blindmaze, an experiment that was executed to showcase our method. Finally, a discussion and conclusion are included before the introduction of our future work.

### Other related work

Various methods of player modeling have been introduced recently. From methods relying on neural network implementations as seen in (Henderson and Bhatti 2001), as well as other AI techniques as seen in (Kirman and Lawson 2009), (Geisler 2002), (Davidson et al. 2000), (Henderson and Bhatti 2001), (Norling 2003). While these methods provide solutions to the problem of player modeling, the introduction of these methods within games might be a complex and non-trivial task. Data that is used to train is complex and deciding what data to use within learning from player data is an error-prone task (Harrison and Roberts 2011). This complexity induces the ad-hoc tendency seen in most player classification methods, since specific methods are introduced that do not focus on generic player behavior modeling (Charles and Black 2004)(Spronck and den Teuling 2010).

Player modeling techniques using AI principles like neural networks or evolutionary processes are typically slow, in contrast to simple operations on simple data collected from a game. Other methods aim at modeling experience in sense of emotions (Pedersen, Togelius, and Yannakakis

Copyright © 2013, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

2010), instead of real play styles. Our method relies on finding play styles, i.e. classes as those described in (Smith et al. 2011). For that, we use FLAME fuzzy cluster analysis (Fu and Medico 2007) designed for DNA analysis inspired by (Kerr, Chung, and Iseli 2011). The clusters found are used to train Hidden Markov Models using the forward-backward Baum-Welch algorithm (Welch 2003). For the actual classification component of the system, we rely on the Viterbi algorithm (Viterbi 1967) for finding the most likely state (or in our case, style) that fits a player. This structure is self-supporting, the only input required is player data and the actual application to be determined by the game developers.

Earlier, Matsumoto and Thawonmas introduced a similar method of player classification using Hidden Markov Models (Matsumoto and Thawonmas 2004) and applied it to a simple MMOG. This method features similar classification, but it predefines player types: game developers still need to pre-specify all types of players a game will attract. Since their approach requires such assumptions, it is less applicable as a design tool for use in the development of different games.

## Behavior and Interactions

Defining behavior becomes highly important when doing behavior-based player classification. We define player behavior as a sequence of time-bound actions performed by the player. Actions are activities performed by a player, and they can be either transitive (i.e. relating to an object, *e.g.* shoot a hand gun) or not (*e.g.* jump). For simplicity, in this paper, we will use the terms action and interaction interchageably.

It is very interesting to identify tendencies and preferences in player behavior. From our definition, this means to single out individual actions from a behavior sequence. For tracking this, we introduce a scoring system for every individual action, where the cumulated scores will characterize the 'popularity' of actions within the player's behavior.

From our definition, tracking player behavior means intercepting the actions that are performed by a player. Actions do not require further information other than the action that was performed (eventually its score and object) and a timestamp marking the moment at which the action was performed. Since a simple registry of actions can be constructed from these fields, introducing this interception of actions in a game will be a simple and trivial task; merely a method for intercepting game actions needs to be introduced.

Most interesting in our definition of behavior is that player behavioral profiles can be derived from it. Our aim is that, when tracking multiple players, certain groups of players (profiles or classes) will naturally emerge from the data. In a sense, this is a dynamic variant of Bartle's types (Bartle 1996); for example, in a real-time strategy game, we would find players who enjoy building and defend, players who enjoy creating large army's and fight, and various players types in between these styles. From our method, these player profiles are the intersection of all cumulated scores of all collected player action data.

With our method, player behavioral profiles are flexible enough to support both absolute and relative player profiling. With absolute profiling, players are classified with exactly one profile. With relative profile, players have a varied behavior which is shares high cumulated scores from more than one profile. Player profiles and their classification are explained in detail below (section Cluster Analysis).

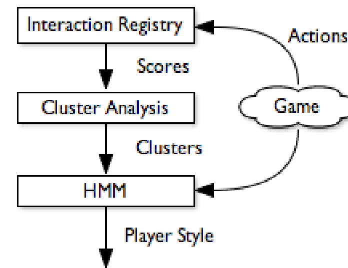


**Figure 1: An overview of the various components**

Defining behavior solely in terms of in-game actions might seem limited at first. One might advocate that a case where we have a 'shoot' action and two players of divergent skill level, the 'shoot' action does not provide equal information about both players. Imagine a case with two different players: the first one is always able to 'shoot' head shots directly, and the second always needs 10 or more shots to kill an enemy. In this case, the 'shoot' actions of the first player are less frequent than the 'shoot' actions of the second one. Since we track sequences of actions, including the 'shoot' and 'kill' actions, we can simply recognize how many 'shoot' actions lead to 'kill' in both cases. Thus, sequences of actions, like the number of 'shoot' leading to 'kill' actions, can be enough to provide accurate classification.

Benefits in using this definition of behavior are twofold. On one side, the inputs and outputs of the system are easily understandable by designers. Actions and interactions are natural design terms for them, rich enough to support intuitive clustering of actions into player classes. Additionally, using actions and objects is a powerful mechanism, generic by nature, since these are concepts which are recurrent in the majority of games.

## Interaction Registry

As seen in Figure 1, our generic player behavior classification approach consists of three components: Interaction registry, cluster analysis and HMM. Naturally, this generic method will be useful when these components are integrated with an external game.

The interaction registry serves as the data storage for separate actions provided by the game and is responsible for maintaining cumulative score values per interaction. Prior to the calculation of cumulative scores, scores per action need to be defined, *i.e.* some sort of scoring logic needs to be implemented. Figure 2 is an example of action scoring. A decay, starting to decrease the action score after 15 seconds, was implemented. This decay can, for example, be used to set the importance of frequent and new interactions. The score of older actions that were not recently performed will slowly decrease exponentially. The decay strength $F$ determines the falloff for the time based decay.

```
struct Action {
    Timestamp        := T
    InteractionIndex := I
    Value            := V
    Score            := 0.0

    void SetValue() {
        S := log(Value);

        dT := Time - Timestamp;
        if (seconds > 15)
        {
            X = seconds - 1;
            S := S * exp(F * X * X);
        }

        Score := S
    }
}
```

**Figure 2: Score update routine**

Furthermore, like stated earlier, a registered action should have a timestamp (initialized by a time $T$) and an interaction index (initialized by an integer $I$), later used to identify the action. A value $V$ could be provided to give certain actions a higher initial value over other actions. Finally, the initial score will be set to zero and later updated by the *SetValue* routine. Since the properties of an Action object are limited to these four fields, we are able to maintain scalability even when processing a large amount of Action or large sequences of Actions.

Games are integrated with our method via the interaction registry. This is done through the registration of actions/interactions and its cumulative scores. For active polling of the current action and registration of all the player's actions, the interaction registry routine *UpdateScoreForIncrement* (Figure 3) needs to be invoked.

Typically, a game update routine should call the *UpdateScoreForIncrement* routine, registering a new action with the interaction registry. Every update tick, the complete set of actions is iterated and the new score values are defined. The cumulated session values per action type are then written to a designated field in the session set. In Figure 3, $Session$ is a set containing our cumulated sessions values. $Actions$ is a set containing the complete sequence of actions throughout the session as $Action$ instances (2). At the end of a game, the $Analyzation.Session$ set will thus contain the cumulated session values over the complete player session and will be processed later on.

The session data that is collected from a play session is stored in a matrix file to be processed by the cluster analysis component. Figure 4 gives an example of such a matrix showing all action scores of five play sessions. Example actions are shown for moving right, left, down, up, making a stop, and walking back a path that has already been visited (loop). We can clearly see that the 'loop' action was only performed by two players in these five sessions. The 'loop'

```
struct Analyzation {
    Session := {}
    Actions := {}

    void New() {
        Session := {0, 0, 0, 0, 0, 0}
        Actions := {}
    }

    void UpdateScoreForIncrement() {
        E = new Action(I)
        Actions.push(E)

        for each (Action in Actions) {
            Action.SetValue()
        }

        Session = {0, 0, 0, 0, 0, 0}
        for each (Action in Actions) {
            N = Action.InteractionIndex
            Session[N] += Action.Score
        }
    }
}
```

**Figure 3: The polling and cumulative scoring routine**

| Right | Left | Down | Up | Stop | Loop |
|---|---|---|---|---|---|
| 39.50 | 11.09 | 35.35 | 13.86 | 4.15 | 0 |
| 22.18 | 6.93 | 16.63 | 9.70 | 7.62 | 40.89 |
| 30.49 | 13.86 | 23.56 | 14.55 | 14.55 | 0 |
| 13.16 | 21.48 | 22.87 | 13.16 | 20.10 | 0 |
| 30.49 | 11.09 | 24.95 | 12.47 | 11.09 | 14.55 |

**Figure 4: Matrix showing results of five play sessions**

action for the second play session (second row) has the highest score value of all actions during these five sessions.

## Cluster Analysis

The cluster analysis component is responsible for player classification, *i.e.* identifying and defining the player behavior profiles. The data collected by the interaction registry and stored in the matrix is processed by the FLAME fuzzy cluster analysis component. This component determines cross sections of actions, by clustering player session data for all players.

Figs. 6 and 5 show two examples of clustering graphs containing approximately 190 analyzed play sessions, for a matrix like the one in Figure 4. Each color represents a different cluster and each dot is the cumulated score value of one play session. The clusters identified by the cluster analysis correspond to play styles, *i.e.* player classes or profiles. Typically, a game designer inspects and interacts with these graphs to identify the clusters that have high score values for certain actions. Designers are responsible for finding clusters which emerge from the data and interpreting their meaning, *i.e.* the common gameplay features they refer to. When looking solely at Figure 6, we can clearly see a blue

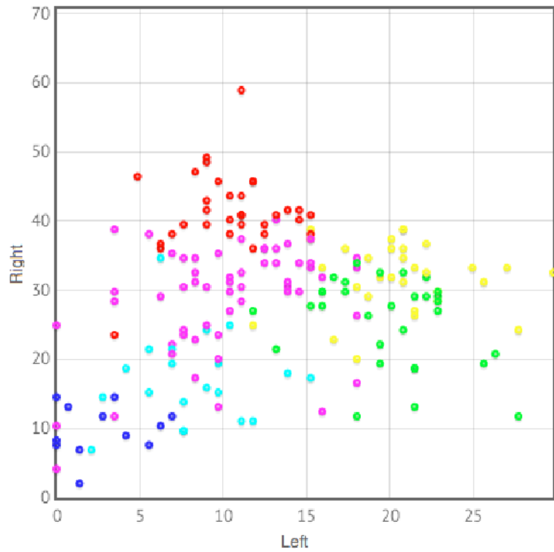


**Figure 5: Graph showing the clusters found with the 'left' action on the x-axis and the 'right' action on the y-axis**

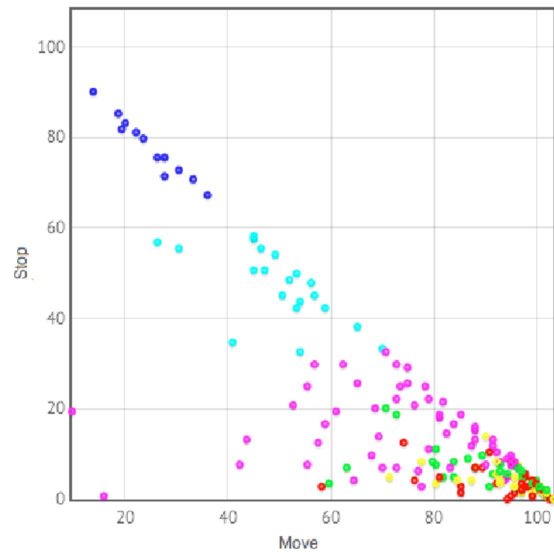


**Figure 6: Graph showing the clusters found with the 'move' action on the x-axis and the 'stop' action on the y-axis**

cluster containing "*doubtful*" players, i.e. reading high values on the stop action. The linearity of the graph is obvious, since it is impossible to perform both stop and move at the same time. When looking further along the x-axis, yielding an increase of the move score, we can see an intermediate light blue cluster and a variety of clusters in the higher move action score region. This higher move action score region starts with the lower purple cluster and includes green, yellow and red clusters, the latter almost never performing the stop action.

The fuzziness of the cluster analysis allows us to have intersections between clusters. A play session could not only be a member of one specific cluster, but also possibly contribute to the value of multiple clusters.

With this analysis, player classes emerge from the clustered data. This means there are no pre-specified assumptions about the player classes which a specific game will attract. This not only makes this method generic and re-usable across different games, but it allows designers to build on top of it. The player data should be typically collected for a game in development, during testing. This way, designers can use the clusters as a basis to create adjustments to the game rules, properties or other game content. In an adaptive game these adjustments would occur on-line, dependent on a player model which uses the same player classes. However, designers can now be responsible for creating these dependencies between adjustments and player classes, by using cluster analysis as its basis and justification. As an example one could think of an adventure game featuring a rope gun used to bridge large distances. A game designer would possibly feel the need to adjust the rope length property of the rope gun for more adventurous player classes derived from our method.

## Hidden Markov Models

The final component of our method is responsible for fitting player classes. In essence, it is a simple player modeling method which can be used in combination with interaction registry and cluster analysis. Typically, this component is not used in the design stage, but during gameplay, for an adaptive game. Its goal is to classify new players, as they play, in order to find an appropriate and dependent game adjustment, as explained above.

The actual classification of players is performed by using Hidden Markov Models. The Hidden Markov Models are trained using the same data of the cluster analysis and the forward-backward Baumm-Welch algorithm, feeding observations in the following format:

$$O = 3,3,3,3,3,3,3,3,1,1,1,1,5,5,3,3$$

Each value in the set of observations $O$ is the interaction index we earlier defined for the registered action in the interaction registry.

Training the Hidden Markov Model with these observations leaves us with a model that we can then use to search the most likely state given a new set of observations. Since states are player classes and symbols are player actions, this method will predict a player class for a set of new observations.

## Blindmaze

For showcasing the interaction registry and cluster analysis components, we executed a simple experiment where we integrated these components with a new game. We developed Blindmaze, a web-based maze game[1] illustrated in Figure 7.

[1]http://photon.marlonetheredge.name/blindmaze/index.html

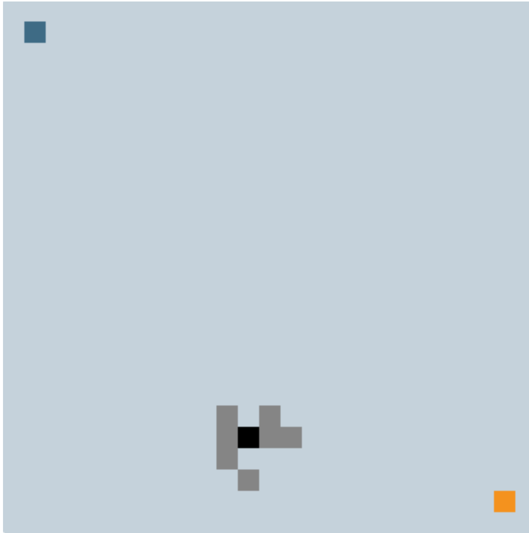

**Figure 7: An overview of the Blindmaze game**

In this game, players need to find a path from start to finish while they are not able to see the complete maze, but only 12 blocks, 2 in each adjacent direction, 1 diagonal. Our aim was that this restriction would encourage players to choose a certain play style, *e.g.* players that have a preference for moving to the left, or players with different abilities in memorizing and getting an overview of the maze (since visual information is lacking).

For tracking a player's behavior we keep track of the actions left, right, up, down, stop, loop, as explained for Figure 4. The current action a player performs is stored, and every 100 ms this action is polled and stored in a separate registry. This registry is continuously updated to determine cumulative scores per player per action.

In Blindmaze's web-page, several visualization tools are available for further exploration. Since scored actions are collected individually, a live interaction graph is presented to the player, as seen in Figure 8. Scores are presented on the y-axis and each interaction has it's own index on the x-axis. These indexes are:

0. Right: The player moves to the right
1. Left: The player moves to the left
2. Down: The player moves down
3. Up: The player moves up
4. Stop: The player stands still
5. Loop: The player visits an already visited cell

Statistics, as seen in Figure 8, are presented to the player and updated in real-time. This gives players (and whoever else is observing gameplay) direct insights in his action score values without having to read a table like Figure 4. Seen in this graph is a high reading of the stop action (4) and low score values on the left (1), up (3) and loop (5) actions. Both right (0) and down (2) actions read intermediate score values, but are high compared to the low score value actions.

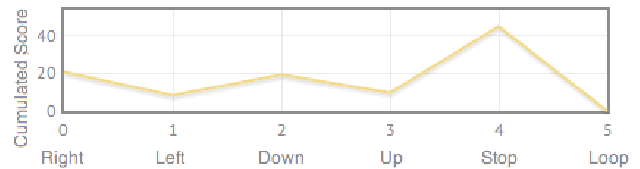


**Figure 8: The real-time statistics presented to the player**

Additionally, all cluster analysis is being performed in the background and presented in Blindmaze's web page. The graphs from Figs. 5 and 6 were taken from the latest dataset of all play sessions. Visitors can interact with the graph clusters, selecting which two actions to plot.

The memory usage of Blindmaze for one player session proves to be 10464 bytes for the set of performed actions and 48 bytes for the set of cumulated session scores (over 6 interactions).

## Discussion

Since interactions featured in Blindmaze are very simple, pinpointing differences between clusters is simple. We found six clusters on six interactions. The most appealing differences in clusters are the differences between stopping and moving in Figure 6, where we can see the differences between uncertain players standing still for long periods of time (one cluster), intermediate players that tend to stop more often than moving (one cluster). And four clusters of faster players, varying between moving most of the time, or almost never standing still. When joining the findings of Figure 6 with the findings of Figure 5 that set out left and right actions, we logically state that for each frequent moving cluster there is one preferring left over right and vice versa. Non-frequent moving clusters or intermediate clusters tend to use the left and right actions less often, explicable, since the reading of the stop action is much higher with respect to the decay decreasing score values for non-frequent interactions.

Implementation of our method within Blindmaze is a trivial task. Like described, the sole task for a game developer when integrating our method is catching interactions. Within Blindmaze this means storing the current interaction a player is performing (e.g. player is moving to the left) and feeding this action to the interaction registry on a set interval. The interaction registry in our case is implemented in Blindmaze itself. For general-purpose use of our method, presenting game developers with a software library encapsulating our method should ease implementation of our method even more.

Apart from the simple interactions featured in Blindmaze, the generic principle of our method allows it to be implemented in more (complex) games featuring a large set of interactions. One can think of the implementation within a first-person shooter where one interaction per weapon, tool or item may be intercepted by our method. Imagine this first-person shooter to have a variety of weapons, tools and items, each featuring some action routine triggered upon performing the primary action of the object. When intercepting the

action routine on these objects, we could distinguish between players using certain types of weapons (e.g. players having a preference for long-range weapons, sniper rifles, or high-explosive weapons, rocket launchers), tools (e.g. players favoring a wrench to repair vehicles over a med-pack to heal wounded players) or items (e.g. a player prone to finding keycards to open locked doors). Like with Blindmaze, the score values per interaction will embody preferences for certain interactions, or groups of interactions. Practicalities of our method are only dependent of the interactions game developers integrated within a game.

Other than game development, use within other branches of software development may be figured. As an example we could think of intercepting actions within a more traditional application like a web store. Actions in this case might be viewing a product, adding a product to a shopping cart and checking out a product. Individual score values per product per action will represent the likings of a customer for a certain product and might lead to the classification of groups of customers favoring similar products.

## Conclusions and Future Work

Player classification has allowed us to greatly improve on both game analytics and game adaptivity. With our method, we aim to make player classification even more available, by making it generic and re-usable across different games. We conclude that we can achieve this by following two main design principles. The first one defines player behavior as a sequence of game actions. With this sort of behavior, player classification can be applied across different game domains, since game actions are an ubiquitous concept across existing games. The second principle offers game designers action-based cluster analysis. With no clusters (player classes) pre-specified, it is up to designers to identify them. The first principle is a key cornerstone of the second one. Using behavior defined as game actions to plot and identify clusters (classes) allows both: (i) that designers can relate to the data and actually find meaning in clusters, and (ii) that clustering analysis can be applied across different games.

With our case study, we concluded that our method provides a generic and simple way of finding existing styles of play within a game. Finding such play styles was possible by identifying the gameplay meaning of the output of the cluster analysis. We concluded that directly using actions as behavior allows a clear interpretation of the processed clusters into meaningful insights on gameplay. Furthermore, from the Blindmaze case, we believe the interaction-based nature of our method allows game developers to easily integrate this method with their games.

We are presently working on a new case study, introducing our method in a game featuring a wider set of (more complex) interactions. We will introduce the notion of layered interactions, where we will be able to share similar interactions over multiple differing objects and thus grouping similar objects (i.e. weapons). Our aim is to integrate this method within a larger approach to adaptive games. We aim to use actions and interactions, specifically its properties, not only as the basis for identifying play styles, but also as the adjustments to be done in-game, in an adaptive fashion.

## References


Bartle, R. 1996. Hearts, Clubs, Diamonds, Spades: Players Who Suit MUDs. *The Journal of Virtual Environments* 1(1).

Charles, D., and Black, M. 2004. Dynamic player modeling: A framework for player-centered digital games. In *Proc. of the International Conference on Computer Games: Artificial Intelligence, Design and Education*, 29–35.

Davidson, A.; Billings, D.; Schaeffer, J.; and Szafron, D. 2000. Improved opponent modeling in poker. In *International Conference on Artificial Intelligence, ICAI00*, 1467–1473.

Fu, L., and Medico, E. 2007. Flame, a novel fuzzy clustering method for the analysis of dna microarray data. *BMC bioinformatics* 8(1):3.

Geisler, B. 2002. *An empirical study of machine learning algorithms applied to modeling player behavior in a first person shooter video game*. Ph.D. Dissertation, Citeseer.

Harrison, B., and Roberts, D. L. 2011. Using sequential observations to model and predict player behavior. In *Proceedings of the 6th International Conference on Foundations of Digital Games*, 91–98. ACM.

Henderson, T., and Bhatti, S. 2001. Modelling user behaviour in networked games. In *Proceedings of ACM Multimedia01*, 212–220. ACM Press.

Kerr, D.; Chung, G. K. W. K.; and Iseli, M. R. 2011. The feasibility of using cluster analysis to examine log data from educational video games. Technical report, Los Angeles: University of California, National Center for.

Kirman, B., and Lawson, S. 2009. Hardcore classification: Identifying play styles in social games using network analysis. In *Proceedings of the 8th International Conference on Entertainment Computing*, ICEC '09, 246–251. Berlin, Heidelberg: Springer-Verlag.

Lopes, R., and Bidarra, R. 2011. Adaptivity challenges in games and simulations: a survey. *Computational Intelligence and AI in Games, IEEE Transactions on* 3(2):85–99.

Lopes, R.; Tutenel, T.; and Bidarra, R. 2012. Using gameplay semantics to procedurally generate player-matching game worlds. In *The Third Workshop on Procedural Content Generation in Games*.

Matsumoto, Y., and Thawonmas, R. 2004. Mmog player classification using hidden markov models. In Rauterberg, M., ed., *Entertainment Computing ICEC 2004*, volume 3166 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg. 429–434.

Norling, E. 2003. Capturing the quake player: using a bdi agent to model human behaviour. In *Proceedings of the second international joint conference on Autonomous agents and multiagent systems*, 1080–1081. ACM.

Pedersen, C.; Togelius, J.; and Yannakakis, G. N. 2010. Modeling player experience for content creation. *Computational Intelligence and AI in Games, IEEE Transactions on* 2(1):54–67.

Smith, A. M.; Lewis, C.; Hullet, K.; and Sullivan, A. 2011. An inclusive view of player modeling. In *Proceedings of*

*the 6th International Conference on Foundations of Digital Games*, FDG '11, 301–303. New York, NY, USA: ACM.

Spronck, P., and den Teuling, F. 2010. Player modeling in civilization iv. In Youngblood, G. M., and Bulitko, V., eds., *AIIDE*. The AAAI Press.

Viterbi, A. 1967. Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. *Information Theory, IEEE Transactions on* 13(2):260–269.

Welch, L. R. 2003. Hidden Markov Models and the Baum-Welch Algorithm. *IEEE Information Theory Society Newsletter* 53(4).