Expert Systems with Applications 41 (2014) 6029–6038

Contents lists available at ScienceDirect

Expert Systems with Applications

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

Fuzzy Tactics: A scripting game that leverages fuzzy logic as an engaging game mechanic

Michele Pirovano [a,b], Pier Luca Lanzi [a,*]

[a] Dipartimento di Elettronica, Informazione e Bioingegneria Politecnico di Milano, Milano, Italy
[b] Department of Computer Science, University of Milano, Milano, Italy

ARTICLE INFO

Keywords:
Video games
Fuzzy systems
Game mechanics
Scripting games

ABSTRACT

Artificial intelligence (AI) plays a major role in modern video games by making them feel both more realistic and more fun to play. Game intelligence usually works alongside the game logic, in the background, invisible to the players who enjoy the resulting character behaviors, the adaptive gameplay, and the procedurally generated content. However, artificial intelligence can also have a central role and become a major component of the overall gameplay (as for instance in the video game Black & White).

In this paper, we define the genre of *scripting video games* and introduce *Fuzzy Tactics*, a video game we developed that has an innovative gameplay based on fuzzy logic and uses fuzzy rules as its core game mechanic and user interaction mechanism. In *Fuzzy Tactics*, players lead their troops into battle by specifying a set of fuzzy rules that determines the battle behavior of the units. Fuzzy logic is the only mean that players have to interact with the game and to command to their troops. Thus, it becomes the main game mechanic that allows us to (i) extend the depth of the game, (ii) keep the interaction intuitive, while also (iii) increasing the replayability and the educational value of the game.

1. Introduction

Artificial intelligence in video games aims at enhancing players' experience in various ways (Millington, 2006; Buckland, 2004); for instance, by providing intelligent behaviors for non-player characters, by implementing adaptive gameplay, by generating high-quality content (e.g. missions, meshes, textures), by controlling complex animations, by implementing tactical and strategic planning, and by supporting on-line learning. Noticeably, artificial intelligence is typically invisible to the players who become aware of its presence only when it behaves badly (as demonstrated by the huge amount of YouTube videos showing examples of bad artificial intelligence[1]).

Artificial intelligence can, in few rare cases, play a more central role and become a mean to introduce *innovative* game mechanics that are built around specific techniques. For instance, the games from the Creatures series (Gameware Development, 1996) make the interaction with the learning mechanism of the underlying neural networks the main focus of the player. The creatures remember facts and adapt to the environment, thus looking intelligent. The award-winning game Black & White (Lionhead Studios, 2001) leverages reinforcement learning to support the interaction with the player's giant pet-avatar as the main core of the gameplay. Most of the gameplay in Black & White (Lionhead Studios, 2001) concerns teaching what is good and what is bad to the pet, a novel mechanic enabled by the AI. In Galactic Arms Race (Hastings, Guha, & Stanley, 2009), the players' weapon preferences form the selection mechanism of a distributed genetic algorithm that evolves the dynamics of the particle weapons of spaceships. The players can experience the weapons' evolution based on their choices. In all these games, the underlying artificial intelligence is the main element that permeates the whole game and also the biggest selling point. The use of AI to support gameplay mechanics thus enables the exploration of new design solutions. It is however difficult to develop compelling gameplay around a specific artificial intelligence technique since it would both require a significant amount of resources for experiments and an in-depth knowledge of the technique as well as of game design; accordingly, only a few successful games based on this idea have been built so far (Gameware Development, 1996; Hastings et al., 2009; Lionhead Studios, 2001).

In this paper, we define the genre of scripting video games and introduce *Fuzzy Tactics*, a tactical role-playing game that makes the artificial intelligence, a fuzzy system in this case, the central element that supports gameplay. In *Fuzzy Tactics*, players lead their

* Corresponding author. Tel.: +39 0223993472; fax: +39 0223993411.
E-mail addresses: michele.pirovano@polimi.it (M. Pirovano), pierluca.lanzi@polimi.it (P.L. Lanzi).

[1] http://www.youtube.com/watch?v=tFxbakAamsc.

http://dx.doi.org/10.1016/j.eswa.2014.02.052
0957-4174/

troops into battle by specifying a set of fuzzy rules that determines the behavior of the units. In our game, the interaction with the underlying fuzzy system *is the main game mechanic* and thus it gives artificial intelligence a primary role by requiring players to script behaviors to be able to play. The use of fuzzy logic in the main game mechanics enables us to (i) extend the depth of the game further than would be manageable, (ii) allow more compelling gameplay to emerge, while also (iii) keeping the interaction intuitive thanks to natural language and (iv) increase the educational value of the game (which at the end can also be viewed as a joyful way to learn something about fuzzy logic).

The paper is organized as follows. In Section 2 we discuss the pros and cons of applying fuzzy logic to games. In Section 3, we briefly overview the use of fuzzy logic in commercial games and in games developed in academia; we also discuss the video games that offer mechanics related to *Fuzzy Tactics*. In Section 4, we present our approach to the use of fuzzy logic in games and show how *Fuzzy Tactics* exploits this technique to implement innovative mechanics. In Section 5, we describe the game mechanics of *Fuzzy Tactics* while in Section 6 we present examples of gameplay scenarios showing the peculiar features of our game. In Section 7, we draw some conclusions and outline directions for future research.

2. Fuzzy logic in games

Fuzzy logic has been officially introduced in game development in 1996 in the Game Developer Magazine[2] by O'Brien (1996) and since then it has been listed as a major technique for game artificial intelligence design by several reference sources. The major textbooks on game AI devote entire chapters to it (Bourg & Seemann, 2004; Buckland, 2004; Millington, 2006) and several introductory articles show how to apply fuzzy logic to games (e.g., McCuskey, 2000). Zarozinski (2002) stretches as far as suggesting that fuzzy logic finds its way in almost every game.

2.1. The benefits of using fuzzy logic in games

Fuzzy logic has several advantages over other artificial intelligence techniques as a mean to introduce advanced behaviors in games. Firstly, fuzzy logic needs no prerequisites apart from basic knowledge of Boolean logic and therefore it is a good candidate to add advanced AI to any game with relatively little effort. In addition, because of its linguistic nature, domain experts can specify fuzzy rules even if they have no knowledge nor understanding of the underlying technology so as to implement human experts' strategies (McCuskey, 2000). This can be very useful in sports and war simulation games. Thirdly, the input–output mappings of fuzzy rules are typically non-linear thus it is generally easy to implement complex behaviors without the need to define mathematical models that may be tedious or impossible to obtain (Gabriyel Wong, 2006); at the same time, such non-linearity is exploited to decrease the predictability of the controlled agents. Accordingly, game designers can use fuzzy logic to implement complex game intelligence without the need for a programmer to assist them. Moreover, while traditional decision making approaches can result in unnatural and unrealistic sudden switch of action policies, fuzzy logic produces smoother changes, although Millington (2006) suggests that such gradual transitions may be over-kill for most current games. Another benefit of fuzzy logic lies in its intrinsic non-sequential representation of knowledge. In fact, since fuzzy rules can be activated in any order, designers can easily add or remove rules without worrying about their activation sequence. Fuzzy logic has also a low computational cost (Li, Musilek, & Wyard-Scott, 2004) which makes it an ideal solution due to the low resources available to game AI developers and to the real-time constraints they are often required to abide to; this benefit is one of the main reasons why fuzzy logic is favored in the gaming industry. Finally, fuzzy systems are also good candidates for in-game adaptation and learning.

To aid game developers in their implementation of a fast-performance fuzzy system, the Free Fuzzy Logic Library[3] (FFLL) has been created and its use is widely encouraged in the literature (e.g., Zarozinski, 2001).

2.2. The pitfalls of fuzzy logic in games

There are relatively few drawbacks for using fuzzy logic in games. Fuzzy systems typically work better if a domain expert is available to specify what the input and output variables are, as well as to sketch the rules that represent existing relationships; if an expert is not available, it might be hard to come up with an adequate rule set and a significant amount of tuning might be needed to implement satisfactory behaviors. This may be a drawback for games based on the simulation of uncommon situations, where domain experts may not even exist (for instance, when designing the game AI for spaceship battles). In addition, if not carefully designed, the development of a fuzzy rule set may result in a large amount of (possibly redundant) rules that will be tested at each time step, dramatically increasing the computational cost. To limit this issue, several improvements have been suggested (Alexander, 2002). For instance, single-state outputs can be enforced to avoid unnecessary computations, hierarchical behaviors can be introduced to resolve groups of rules at once and parallel and independent behavior layers with different evaluation frequencies can be used.

3. Related work

Fuzzy systems have been often applied to video games to tackle several tasks e.g., to implement game intelligence (Johnson & Wiles, 2001), for graphics (Hsu, Kao, & Wu, 2009), and to support design (Lo & Wen, 2010). In this section, we overview games that are related to Fuzzy Tactics either for their use of fuzzy logic or for their game mechanics, in order to provide a frame of reference for the subsequent discussion.

3.1. Fuzzy logic in commercial games

Although fuzzy logic is well known in the game AI literature (Millington, 2006), it is hard to find mentions of actual commercial games that leverage fuzzy logic. This suggests two alternative conclusions: either fuzzy logic has become such a wide-spread technology that it is deemed not worth mentioning as a bullet point as compared to other more exotic techniques, or the technique, while well known in theory, is not actually used so thoroughly. Nonetheless, an older comprehensive list of games using fuzzy logic techniques can be found among the titles listed in Woodcock (2000). Unreal (Epic Games, 1998) is one of the most famous first person shooter in videogame history and has been reported to use fuzzy state machines (FuSMs) to control the behavior of enemy aliens; when published, the game was praised for its believable game intelligence (Johnson & Wiles, 2001; Woodcock, 1999). Civilization: Call to Power (Activision, 1999) (Fig. 1(a)), turn-based strategy game that is a spin-off of the well-known franchise, uses fuzzy state machines (FuSMs) to set priorities for

[2] Previously at http://www.gdmag.com now available at http://www.gamasutra.com/topic/game-developer.

[3] http://ffll.sourceforge.net/.

Fig. 1. Games that make use of fuzzy logic. Top left: Civilization: Call to Power. Top right: The Sims. Bottom left: Close Combat 2. Bottom right: S.W.A.T. 2.

strategic level intelligence, allowing personality traits to be defined for the different civilization leaders (Johnson & Wiles, 2001). The worldwide top-selling game The Sims (Maxis, 2000) (Fig. 1(b)) uses fuzzy state machines to determine what objects a character can interact with based on their properties and the characters personality traits. Close Combat (Atomic Games, 1996) and its sequel Close Combat 2 (Atomic Games, 1998) (Fig. 1(c)) use a Fuzzy State Machine that weights hundreds of variables to determine the probability to select actions (Merrick & Maher, 2009; Sweetser & Wiles, 2002). Enemy Nations (Windward Studios, 1997) features enemies that employ finite state machines, fuzzy state systems, and a database of goals and tasks (Woodcock, 2000). S.W.A.T. 2 (Yosemite Entertainment, 1998) (Fig. 1(d)) is a real-time tactics game that has been reported to make extensive use of fuzzy logic to enable the non-player characters to behave spontaneously based on their defined personalities and abilities (Johnson & Wiles, 2001; Sweetser & Wiles, 2002).

3.2. Fuzzy logic in game research

Fuzzy logic has been widely used in commercial games with success, however most of the times its application is restricted to simple inference engines or fuzzy state machines. In contrast, fuzzy logic has been broadly applied in academia to tackle a wide variety of tasks related to game research. For instance, it has been used for the design of the behavior of the enemy ghosts in a Pac-Man clone (Namco, 1980; Shaout, King, & Reisner, 2006); however, heavy tuning was needed to achieve a reasonable behavior. Fuzzy Q-learning, borrowed from the fields of robotics, was used in a Ms. Pac-Man clone (DeLooze & Viner, 2009; Midway, 1982). Li et al. (2004) mention fuzzy control as a practical method for generating subtle behavior and use it in a Belief-Desire-Intention (BDI) framework as part of decision making for a BattleCity (Namco, 1995) clone. Ho and Garibaldi (2008a) introduced the concept of Context-Dependent fuzzy system, in which the membership functions of the fuzzy variables are not fixed but change according to the context. The proposed approach was applied to design a controller for the car racing competition held at FuzzIEEE 2007. In Ho and Garibaldi (2008b), the same authors, present an improved version of the controller for the 2007 CIG Simulated Car Racing Competition. The controller consisted in a two-layers architecture that combines a high-level path planner with a low-level execution controller based on fuzzy logic.

In Perez, Recio, and Saez (2009), Perez et al. presented a driver based on a fuzzy controller for the 2008 CIG Simulated Car Racing Competition. First, they designed the rules and the fuzzy sets of a base driver. Then, they applied a genetic algorithm to optimize the parameters of the fuzzy sets. Onieva, Pelta, Alonso, Milanes, and Perez (2009) developed a modular architecture in which the general driving was implemented by a fuzzy system controlling the target speed. To this date, the tuned fuzzy controller still outperforms other controllers in the same competition.

In Onieva, Cardamone, Loiacono, and Lanzi (2010), we presented an initial study of blocking in car racing games based on our experience in the organization of the Simulated Car Racing Competition (Loiacono et al., 2008, 2010;) we showed that even the most competitive controller can fail to overtake even the most basic blocking strategies on very simple straight track sections; we also showed that a simple fuzzy controller could tackle blocking behaviors that more advance drivers failed to manage. The original study was later extended in Cardamone, Lanzi, Loiacono, and Onieva (2013), where we presented a more detailed experimental analysis. Fujii, Nakashima, and Ishibuchi (2008) also applied fuzzy systems to simulated car racing. In this case, rules are generated

from a set of training patterns; the study compares two methods for generating such training patterns and two representations of the sensory information (third person vs. egocentric).

The application of fuzzy logic as an alternative of the original AI of commercial games has been widely explored as a way to have a industry validation of the approach (and thus a greater impact on the game industry). This however requires the game's AI to be extensible and the commercial game to be open-sourced, features found only in a handful of games. The code of Quake III Arena (Id Software, 1999), a major commercial success in the game industry, is available as open source and the game has thus been the target of much research. Fuzzy-logic controlled bots have been released for the game (van Waveren, 2001), with weapon and item selection controlled by fuzzy decision making. The fuzzy bots have shown interesting performance and they are now taken as the basis for comparison when more advanced techniques are used (Prieditis & Dalal, 2006; Westra & Dignum, 2009). Pinto and Alvares (2006) model fuzzy sensors as input to extended behavior networks in Unreal Tournament (Epic, 1999), another famous first person shooter, while Acampora (2010) uses timed automata and fuzzy controllers to model emotions for bots in a Unreal Tournament 2004 (Epic, 2003) match.

Additional uses of fuzzy logic in games regard the classification of player feedback and learning from the player. In these contexts, fuzzy logic is used as an effective way to model the player's reasoning. El-Nasr, Yen, and Ioerger (2000) report that fuzzy logic has provided better means of modeling emotions due to its qualitative and quantitative expressiveness. Levillain, Orero, and Rifqi (2010) use fuzzy decision trees to categorize the emotional feedback of players during gameplay. Gabriyel Wong (2006) uses fuzzy control to manage the complexity of a scene by classifying 3D models according to their level of detail. Ohsone and Onisawa (2008) use fuzzy decision trees to determine the best response of their virtual opponent.

Fuzzy systems are good candidates for learning and fuzzy logic has been used to model rules that are then evolved using the player's actions as input to an evolutionary algorithm (Avery & Michalewicz, 2008). In Ishibuchi, Sakamoto, and Nakashima (2003), the authors extract data from the iterative execution of games and then learn fuzzy rules for the classifications of player actions.

3.3. Scripting as a game mechanic

In *Fuzzy Tactics*, the players indirectly control their units through scripting a set of fuzzy rules that defines units behavior beforehand; then, they sit back, relax, and watch their troops go to war. In the game literature, there is no mention of similar mechanics. In this paper, we introduce the notion of *scripting games* to identify those games in which the players cannot directly interact with the game world to complete the game objective, but they have to script the behavior of one or more agents to complete the tasks. Scripting games have usually two distinct phases: (i) a planning phase, during which players program their agents, (ii) a simulation phase, when they experience the results of their planning. Based on the results, players can either progress to the next level/battle or need to refine their plan.

Scripting games have their roots in the early programming games like Darwin (Vyssotsky, Morris-Sr, & McIlroy, 1961), Core Wars (Jones & Dewdney, 1980), and their descendants such as RoboCode (see http://corewar.co.uk/ for more programming games). They are not a popular game genre probably due to their appeal to the niche of strategic-type players, their lack of immediate feedback, and to the intrinsic complexity of the scripting mechanics. Nonetheless, various attempts have been made to increase the appeal of this genre, accordingly we can find games of this genre on all major gaming platforms. Cargo Bot (Viana, 2012) is a scripting game published for the Apple iPad in which players have to program a manipulator to complete puzzles. In Light Bot (CoolioNiato, 2008), players must program a small robot to move around the game grid and activate all the required blocks. SpaceChem (Zachtronics Industries, 2011) is a good example of a successful scripting game, set in a chemistry plant, in which players build machines that separate and merge chemical elements to meet production requirements.

All games mentioned so far belongs to the puzzle genre, but scripting games cover a wider range of genres such as tactical and strategic games. The Carnage Heart (Artdink, 1997) series for the Sony PlayStation systems requires the player to equip and program robots for an upcoming duel. Gratitious Space Battles (Positech Games, 2012) is a successful independent scripting game in which fleets of spaceships fight against each other; the player is allowed to customize and program the spaceships, although using only simple orders. The games of the Dominions (Illwinter Design, 2007) series make great use of battle scripting: players can issue simple orders to their units before a battle begins and they have no control on their behavior during the actual battle. Frozen Synapse (Mode 7, 2011) is another recent example of independent successful game that borrows features from scripting games, in a lesser degree. In Frozen Synapse (Mode 7, 2011), the player is asked to control a handful of units to complete tactical objectives. At each turn, which consists of a few seconds, the player can visually program the behavior of each unit, then the turn is played.

A few open-source games are available online that require actual programming skills such as Real Time Battle (Real Time Battle, 2006) and RoboCode (Nelson, 1980), both requiring the players to program battle robots. These games are obviously targeting programmers (or programming students) and their commercial value is hard to estimate.

4. Fuzzy logic as a game mechanic

Scripting games, like all programming systems, implement a trade-off between language simplicity, representation power, and programming flexibility. To be fun and accessible to non-programmers, a scripting game needs a very simple programming language. Most games accomplish this through the creation of visual operational languages, such as in the case of Light Bot (CoolioNiato, 2008) or SpaceChem (Zachtronics Industries, 2011). Other games, such as Gratitious Space Battles (Positech Games, 2012) or Dominions (Illwinter Design, 2007), simply let players specify *only* high level behavioral rules, thus limiting their flexibility. These approaches are typically easier to understand for non-programmers, but they can only represent very basic behaviors. Accordingly, they are way too simple to implement advanced tactical maneuvering. In fact, their use is usually limited to simple puzzle games and their extension to wargames has not been successful yet.

In *Fuzzy Tactics*, we propose a solution to this issue that leverages the intrinsic simplicity of fuzzy logic and uses it to support the main game mechanic for a tactical role-playing game with indirect control. Our aim is to create complex and deep gameplay for scripting games, favored by tactical-minded players, while keeping the gameplay accessible and intuitive for most people. Accordingly, we developed *Fuzzy Tactics*, a game that requires players to lead their troops into battle by specifying the behavior of single units and teams. In our game, units are controlled by a fuzzy system and the players can define their behavior using a rich visual language. The players goal is to learn to command their troops using the fuzzy rules that control each unit, becoming increasingly skilled in defining the rules as the game progresses. Through trial

and error and the experience shared with other players, players improve their skills and reach mastery.

We chose fuzzy logic since, because of its linguistic nature, it allows players to easily write complex strategies that their units will execute. Players command the troops by writing readable (fuzzy) rules that encode complex strategic decisions with ease using an intuitive graphical interface that resembles visual scripting environments (see Section 5). Players can thus easily express complex notions while the underlying fuzzy logic emulates the reasoning.

Compared to other games that use fuzzy logic in the design and development phase, in *Fuzzy Tactics* we take a step further and *give the players the role of domain expert/designer* whose knowledge is leveraged by the underlying fuzzy system. By using fuzzy logic as a major element of gameplay we also gain additional benefits. Since the underlying system is a black box, players can specify the rules without the need to delve into technicalities which might distract them from the real goal, that is to have fun playing a tactical warfare game; instead, players only need to specify unit commands that are typical of strategic and tactical scenarios. The intrinsic non-linearity of fuzzy inference systems also makes the creation of unpredictable and complex behavior easier, so as to increase replayability and gameplay variety. In addition, the low computational cost of the fuzzy reasoning systems, coupled with the solutions for managing a large number of rules, allows us to deploy and control a large number of units on the battlefield. Finally, the affinity of fuzzy systems to learning algorithms can be leveraged to create suitable computer-controlled enemies for the game and also aid in balancing.

5. Fuzzy Tactics: the game

Fuzzy Tactics is a scripting video game we developed whose genre lies at the intersection of strategy games, tactics games and tactical role-playing games (see Fig. 2(a) and (b)). The player commands an army and has to defeat enemies in battles, which take place on a battlefield represented by a hexagonal grid. The game generates a wide possibility space through a plethora of units to choose from, different weapons to equip, skills to use and abilities to learn; all of these elements affect in their own way the result of a battle, promoting emergent gameplay (Juul, 2002). The players are required to exercise their strategic skills to manage the army's resources and their tactical maneuvering skills to win each battle, thus focusing on two of the five fundamental mechanics types as listed by Adams (2009). The current game prototype presents the battles as a series of separated events; we later plan to link the battles with a plot and a layer of strategic turn-based gameplay.

In contrast to most games in the tactical/strategic genres, in our game the player has no control over the units during battle, and this peculiarity of the gameplay makes *Fuzzy Tactics* a *scripting game* (see Section 3.3). The player can equip units, train them and arrange their formation before the battle begins, but when the fight starts the player can only sit back and watch the clash from afar. The player can create orders for the units to follow, with each unit having its own list of orders. These orders allow the player to plan what actions the unit will perform during the fight, basically programming its behavior. Thanks to the use of fuzzy logic in this process, issuing orders to the units becomes as natural as explaining them what to do using written language. For example, the unit may be given the order to attack with a ranged weapon when it is far enough from the enemy unit, or to flee when the enemy gets dangerously close.

The battle takes place in real time, but all units are actually synchronized and reason/act in discrete time steps. At each time step, each unit analyzes the battlefield's state to determine, based on its orders, the action to take. To do so, the fuzzy reasoning system of each unit (basically, its brain) checks its rule base (its orders) against the current state of the battle, that is determined by the values of the fuzzy variables it can access. Input fuzzy variables are automatically created for each variable parameter of a unit (maximum health, current health, strength, intelligence, speed, etc.) and for each variable that represents a relationship between two different units (for example, distance); each unit has access to all the fuzzy variables of all units on the battlefield. To each input fuzzy variable, we assigned seven trapezoid fuzzy membership sets that we labeled *very-small*, *small*, *medium-small*, *medium*, *medium-large*, *large*, and *very-large*, covering the whole range of the variable (see Fig. 3). Output fuzzy variables represent the preference for actions to be taken and for what to target, with one output variable created for each action-target pair. The output of the fuzzy inference engine thus determines the action the unit takes (flee, attack, move, etc.) and which unity to target (enemy, ally, itself) based on the current battle state. When all units have chosen their action, a single battle step is simulated and the battle advances with all units acting simultaneously. Since the fuzzy reasoning system of each unit can efficiently choose the action to take, there is no delay between one battle step and another, resulting in a continuous battle flow.

A single battle can end in three ways: (i) either the player's army is defeated and thus the player loses; (ii) the enemy army is defeated and the player wins; or (iii) too much time has passed without a clear winner and the battle ends with a draw.

5.1. Order Creation System

Before the battle begins, players can setup their army through the *army setup* interface. In the current version of the game, units have fixed but different equipment, statistics and skills, while their placement on the battle grid and their orders can be configured by the player.

Players can easily specify the orders for each one of their units using a graphical interface we created (dubbed *Order Creation System* or OCS) that is linked to the fuzzy reasoning system of the units. To make the system as straightforward as possible while still retaining its flexibility, we allow the players to define orders using predefined building blocks on a three-layer hierarchy. The first layer is the order-list layer: here, the player can create, delete, rename or choose to modify orders in a sequence that determines their priority and thus the correspondent fuzzy rule's weight (see Fig. 4(a)). A library of orders is also available and can be used to add pre-defined orders to the current unit. The second layer concerns the details of a specific order and can be accessed when an order is selected with a click in the first layer. The player can add, delete and modify a set of *conditions* independently (the antecedents of the fuzzy rules) and a set of *consequences* (the consequents of the fuzzy rules) (see Fig. 4(a)). Conditions and consequences are linked with an *and* clause, and they are collectively referred to as *prepositions*. The third and final layer concerns the details of the fuzzy rules' prepositions and can be accessed when one of them is selected with a click in the second layer. The player can build prepositions from a sequence of atomic building blocks that correspond to subjects, variables, verbs, label modifiers and labels. By combining the building blocks, a preposition is created much in the same way as a person would write it down using the English grammar structure, making the creation process natural. Fig. 4(b) shows the interface of the Order Creation System where a visual explanation of the preposition is shown on the right to further help the player. Subjects refer to the units in the battle (enemy, allied, self), variables to the fuzzy variables tied to the units (health, distance, strength, etc.), verbs to the actions that

Fig. 2. Battles in the world of Fuzzy Tactics. On the top, we see a barrage of arrows hitting the unfortunate frog-men. On the bottom, we see crocodile-men and frog-men fighting with arrows.

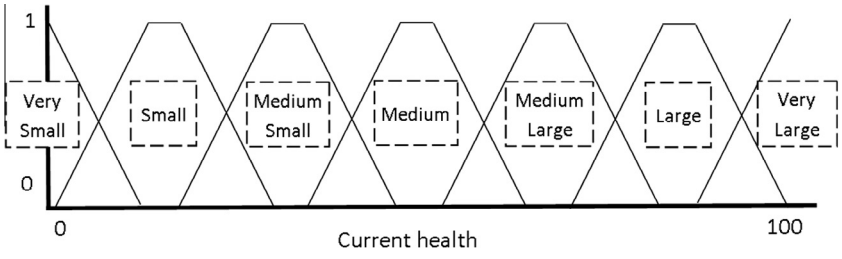

Fig. 3. A fuzzy variable related to the current health parameter of a unit (from 0% to 100%). We can see the seven fuzzy membership sets that cover the whole range.

the units can perform (attack, move, flee, etc.), and labels and label modifiers correspond to either one of the seven defined fuzzy set labels or special labels that define set compounds. To make the editing even simpler, we constrained the possible sequences of blocks to forbid infeasible or meaningless prepositions using the preposition's type (condition or consequence) and the blocks already used in its construction.

Conditions are created in the form "if *subject*'s *variable* is *label*", with extensions for label modifiers and relative variables. Example conditions are "if my health is large" or "if an ally's distance to the enemy is very large." Consequences are created in the form "then *verb target*", with examples such as "then attack the enemy" or "then cast the 'cure' spell on the ally." Before a battle starts, all the orders are converted to actual fuzzy rules based on the specific battle's setup: generic subjects such as *enemy* or *ally* are converted to the actual unit IDs, and the resulting rule set is used.

5.2. Rule creation example

As an example of gameplay, we illustrate the creation of rules for one unit. Suppose we want to create the rules for a melee-type unit and that we we want each warrior to behave as follows: when an enemy is far, the warrior should get closer to the enemy; when an enemy is near, the warrior attacks.

Fig. 4(a) shows the interface used for creating such orders. In the order-list layer (i), we add two orders and we assign them the name *Get closer* and *Attack closest*. The first order (*Get closer*) is highlighted because it has been selected and as such the order

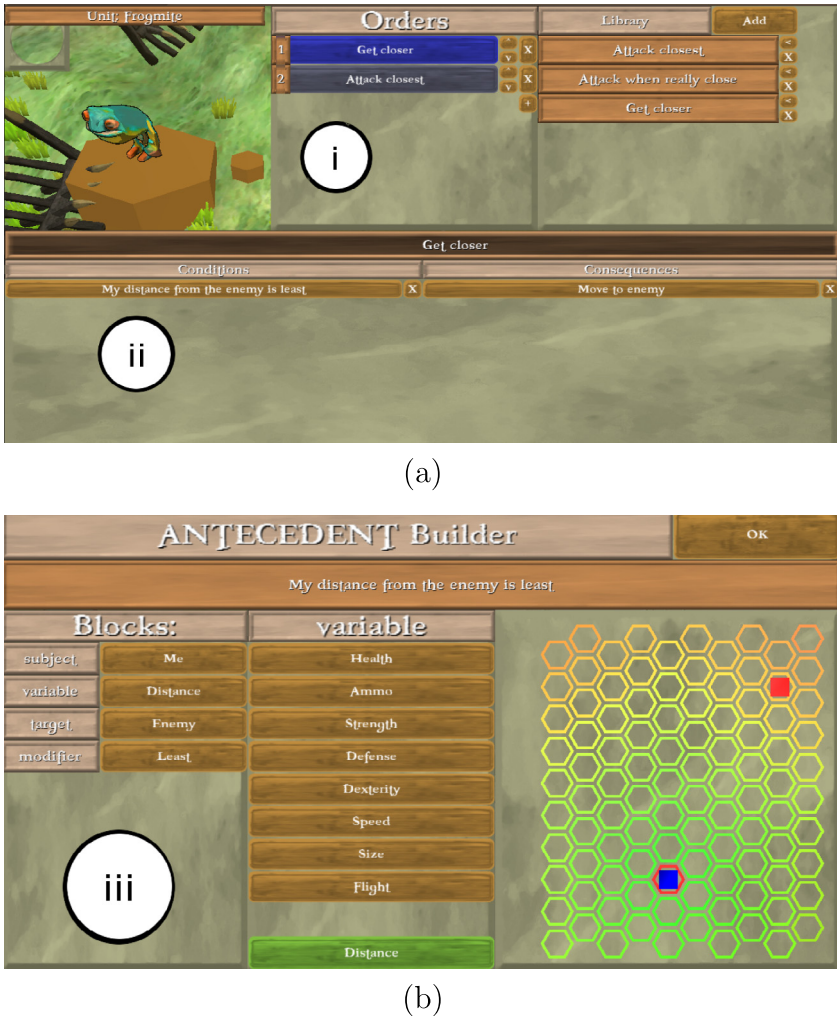

Fig. 4. The interface of the Order Creation System that allows the player to create sets of orders for each unit. We can see the selected unit, the order-list layer (i) and its library, the order layer (ii) and the preposition layer (iii).

layer (ii) shows its contents: it has one condition and one consequence. The condition reads "*my distance from the enemy is least*" and it is thus concerned with the distance of the unit from any one of its enemies. The consequence reads "*move to enemy*." Combined, the two prepositions allow the unit to move towards the closest enemy. The second order was created similarly, although its construction is not shown in the figures, and reads "*if my distance from the enemy is less than small, then attack the enemy*," allowing the unit to attack when in range. Notice that the *Get closer* order appears first in the sequence and will thus get higher weight; the unit prioritizes moving over attacking. Fig. 4(b) shows the preposition layer (iii). In this case, the condition of the *Get closer* order is selected and we can see how it has been built using the word blocks on the left. We note that the condition has been created using the subject *Me* (i.e., the acting unit), the variable *Distance*, the target *Enemy* and the label *Least*. We can also note that the variable block has been clicked for modification and the options for its value are thus shown (distance, the chosen one, is highlighted in green).

6. Fuzzy Tactics: game scenarios

We now present examples of scenarios that can be played in the current version of the game. The scenarios are set up as plain empty battlefields with predefined allied units (identified by an *A*) and enemy units (identified by an *E*), with fixed equipment, placement and skills. Unit statistics are not considered and set to the same values for each unit. We test different rules and simulate the resulting battles. A video showing the scenario discussed here is available at http://www.polimigamecollective.org/fuzzy-tactics/.

6.1. Scenario 1

In the first scenario, we show a simple example that highlights the great potential of our approach. Only two units are placed on the battlefield: an allied warrior unit A_w equipped with a melee weapon and an enemy archer unit E_a equipped with a ranged weapon. Fig. 5 shows the placement of the units on the battlefield.

For illustration purposes, we are only interested in the distance between the units. We thus generate one fuzzy input variable: E_a's distance to A_w from 1 (minimum) to 16 cells (maximum). For short, we call this variable just *distance*. The output fuzzy variables are:

- E_a attacks A_w
- E_a holds
- E_a moves to A_w
- A_w attacks E_a
- A_w holds
- A_w moves to E_a

E_a is given the order "*if the enemy's distance is equal or larger than small, then attack.*" A_w is given no order and will thus use

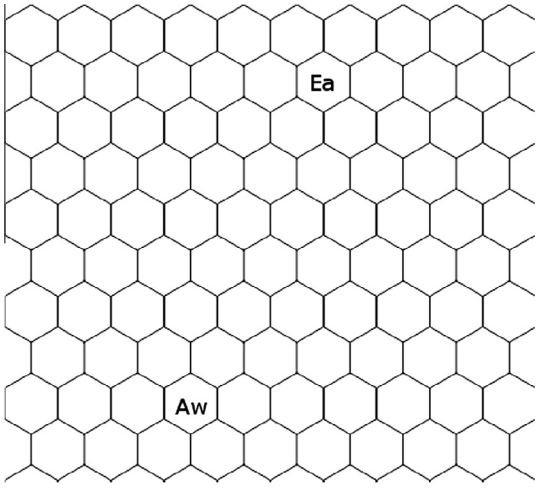

Fig. 5. Fuzzy Tactics, setup of scenario 1. The allied warrior A_w against the enemy archer E_a.

the standard action *hold* for the whole battle. Internally, the following fuzzy rules are generated:

- IF distance IS very-small THEN E_a holds
- IF distance IS small THEN E_a attacks
- IF distance IS medium-small THEN E_a attacks
- IF distance IS small THEN E_a attacks
- IF distance IS medium THEN E_a attacks
- IF distance IS medium-large THEN E_a attacks
- IF distance IS large THEN E_a attacks
- IF distance IS very-large THEN E_a attacks

With this setup, when we start the simulation by pressing the play button, we see that E_a can easily dispose of our unit, resulting in a loss. This is because the distance variable is around 6 units (medium = 0.7, medium-large = 0.3), hence the output is always (E_a attacks = 1).

We retry, instead, by giving A_w the order "*if the enemy's distance is equal or smaller than small, then attack*" and the order "*if the enemy's distance is larger than small, then move to the enemy*." Internally, the following fuzzy rules are generated:

- IF distance IS very-small THEN E_a holds
- IF distance IS small THEN E_a attacks
- IF distance IS medium-small THEN E_a attacks
- IF distance IS small THEN E_a attacks
- IF distance IS medium THEN E_a attacks
- IF distance IS medium-large THEN E_a attacks
- IF distance IS large THEN E_a attacks
- IF distance IS very-large THEN E_a attacks
- IF distance is very-small THEN A_w attacks
- IF distance is small THEN A_w attacks
- IF distance IS medium-small THEN A_w moves
- IF distance IS small THEN A_w moves
- IF distance IS medium THEN A_w moves
- IF distance IS medium-large THEN A_w moves
- IF distance IS large THEN A_w moves
- IF distance IS very-large THEN A_w attacks

We thus obtain the following behavior: at the beginning of the battle, E_a attacks from afar, while A_w tries to close the distance between the two units. As A_w gets close enough, it will start to attack, while E_a stands still, due to its only order not triggering and thus reverting to the standard *hold* action. As a result, the player wins. As a last test, requiring a new *flee* action and output variable, we give E_a an additional order "*if the enemy's distance is below small, then flee*". With this last order added, the enemy archer will flee when the warrior approaches and the warrior will follow it until it can successfully strike it down.

In this simple scenario, with just a couple of rules, interesting behavior can already be seen. Note also that the addition of a single rule can change the result of the battle dramatically.

6.2. Scenario 2

In the second scenario, we show the emergent dynamics that can arise from the interaction of different units. We place two enemy warrior units, E_{w1} and E_{w2}, with the same orders we used for our winning allied warrior in scenario 1. On the player side, we place an allied warrior A_w (again, with the same orders) and an allied healer A_h that can use the *cure* skill to heal a damaged allied unit. The units' placement can be seen in Fig. 6. We generate six fuzzy input variables,

- E_{w1}'s distance to E_{w2}
- E_{w1}'s distance to A_w
- E_{w1}'s distance to A_h
- E_{w2}'s distance to A_w
- E_{w2}'s distance to A_h
- A_w's distance to A_h

while the output fuzzy variables are

- E_{w1} attacks A_w
- E_{w1} attacks A_h
- E_{w1} holds
- E_{w1} moves to A_w
- E_{w1} moves to A_h
- E_{w2} attacks A_w
- E_{w2} attacks A_h
- E_{w2} holds
- E_{w2} moves to A_w
- E_{w2} moves to A_h
- A_w attacks E_{w1}
- A_w attacks E_{w2}
- A_w holds
- A_w moves to E_{w1}
- A_w moves to E_{w2}

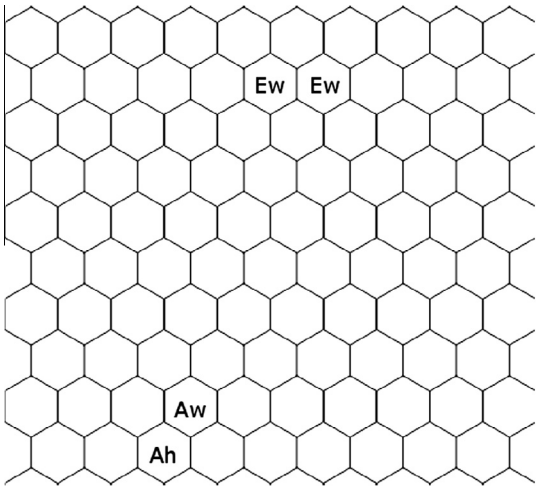

Fig. 6. Fuzzy Tactics, scenario 2. On the allied side, a healer A_h and a warrior A_w get ready for the battle. On the enemy side, we have two warriors E_w.

- A_h attacks E_{w1}
- A_h attacks E_{w2}
- A_h holds
- A_h moves to E_{w1}
- A_h moves to E_{w2}
- A_h casts cure on A_w
- A_h casts cure on A_h

Note the additional outputs for casting the magic of A_h; the healer A_h is given the order "*if an ally's health is below medium, then cast cure on the ally*" and the order "*if my health is below medium, then cast cure on me*." Due to the fuzzy rule weighting favoring orders that come first in the order sequence, A_h will prioritize healing its allies, then itself.

With this setup, the offensive power of the enemy's army is greater (two warriors instead of one), but running the simulation leads to a possibly unanticipated result. The two enemy warriors and the allied warrior approach each other and start fighting in melee range, while the healer stands behind. As A_w's health gets lower, A_h casts its cure skill repeatedly on it, replenishing its health. Eventually, A_w kills both enemy warriors and the player wins. If the player's strategy (or the enemy's) was different, or the balance of statistics and skills of the different units were different, we could get a quite different result. As an example, if the *cure* skill were weaker, A_w would die even under the effects of the healing. On the other hand, if the enemy warriors were ordered to kill the healer first, for example by making sure to target lower-health units first, the enemy could win the battle.

This scenario shows how much small differences in the starting conditions in either the battle setup or the issued rules can drastically change the results of a battle, even with a modest number of units, highlighting the presence of emergent gameplay. In addition, it shows that the flexibility given by differentiating units can be a useful asset for a player's strategy.

6.3. Scenario 3

In our last scenario, we show how a symmetric condition may result in a battle loop, and stress the solutions that can be taken to avoid the situation. The battle configuration consists of two enemies (E_w an enemy warrior and E_h an enemy healer) with orders set up as in the previous scenarios. Symmetrically, we have two allied units A_w and A_h on the other side of the battlefield (see the setup in Fig. 7).

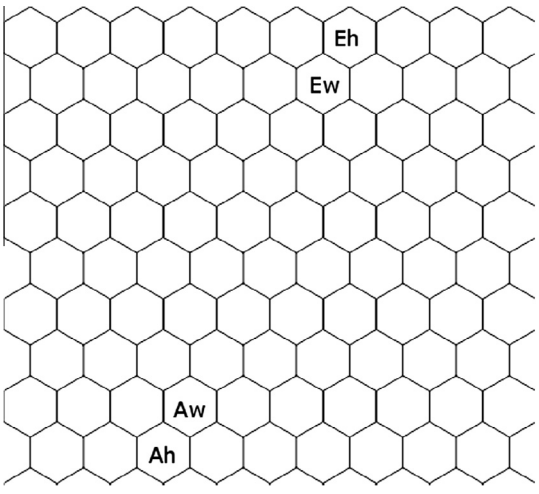

Fig. 7. Fuzzy Tactics, scenario 3. An allied warrior A_w and a healer A_h can be seen in the lower side of the battlefield, while the symmetric enemy warrior E_w and healer E_h can be seen on the other side.

Since the conditions are symmetric and since there are no random factors, the result is a battle loop: the opposing warriors will get close and start to fight, while the healers will stand back and cast their cure skill when necessary. The battle will thus go on forever, with neither side being able to defeat the other. We avoid such situations in the game by setting a time limit for the battle so as to declare a draw if there is no clear winner when the time elapses. However, such loop behavior can be readily avoided if the player changes his tactics, for example by focusing on the healer first. From a game design point of view, introducing limited resources in the battle would also avoid this. For example, by introducing a limited pool of points to spend on skills, the healers would not be able to cast their cure skill indefinitely.

7. Conclusions

In this paper, we (i) defined the video game genre of scripting games, and (ii) introduced *Fuzzy Tactics*, a scripting game that makes good use of fuzzy logic to create innovative gameplay, working as an example of how expert systems can be integrated into game design. The definition of scripting games can be useful for research, as such games focus on indirect control and favor strategic scenarios, potentially functioning as good test beds for the expert system community. In particular, we presented an innovative approach to the development of scripting games by leveraging the power of fuzzy logic to create a game, *Fuzzy Tactics*, that allows deeper gameplay than other similar games. At the same time, the game is also made more intuitive due to the use of natural language, allowing its audience to be broadened. Fuzzy logic, or other expert systems, can thus be useful from a game design point of view; by incorporating computational intelligence methods into games in novel ways, we can explore new types of gameplays and expand into novel areas of game design. *Fuzzy Tactics* also proposes new game mechanics at the intersection of the strategic games genre and the scripting games genre, giving hardcore players a way to work on actual game intelligence while also testing their strategic skills. At the same time, the same innovative game mechanics pose new challenges for designing user interfaces that can make the underlying fuzzy engine accessible to the broader strategy gamers' audience.

The first prototype we developed implements the main game features, although it is restricted only to three types of units. A video of the gameplay, showing the scenarios described in the previous sections, is available at http://www.polimigamecollective.org/fuzzy-tactics/; we plan to release an alpha version to validate the usability of the Order Creation System and of the game interface. Extensive playtesting will also assess whether meaningful play is generated through the peculiar gameplay that Fuzzy Tactics offers. We aim to refine and expand the game with new content, such as new units, rules and skills, and we will test whether there are enough tactical choices to satisfy the tastes of this genre's veterans.

This work has given us many ideas for future work concerning the benefits of fuzzy logic in *Fuzzy Tactics*, extending the research impact of the game. We plan to introduce and playtest additional innovative gameplay elements made possible only through the use of fuzzy logic. For instance, we plan to model units emotions to be able to create units that attack when angered or flee when scared. We also plan to add special skills to the units that can interfere with the fuzzy system reasoning, for example by making a unit appear weaker than it really is, thus deceiving enemy units into performing wrong actions. Additionally, we plan to add a degree of free will to the units, so that their behavior is conditioned by their own intelligence; as an example, think of a less intelligent unit that does not understand complex orders, or of a honorable

combatant that refuses to commit to all orders. Finally, we want to investigate the use of learning techniques to create units that can adapt to the situation by modifying the fuzzy rules based on their performance. We remark that such innovations are made possible by the use of expert systems in our game.

Additional future directions regard the broader topic of leveraging expert systems for gameplay. Nowadays, players expect most games to be realistic in everything, not only in their graphics, and artificially intelligent characters are one of the main basis for comparisons that player use to measure the realism of a game. By leveraging expert systems to model the behavior of game characters agents, higher realism can be pursued. *Fuzzy Tactics* can be a first step towards this direction.

References

Acampora, G. (2010). Synthesizing bots emotional behaviors through fuzzy cognitive processes. In *Proceedings of the 6th international conference on computational intelligence and games, CIG'10*.

Activision (1999). Civilization: Call to power. <http://en.wikipedia.org/wiki/Civilization:_Call_to_Power>.

Adams, E. (2009). *Fundamentals of game design* (2nd ed.). New Riders.

Alexander, T. (2002). An optimized fuzzy logic architecture for decision-making. *AI Game Programming Wisdom*.

Artdink (1997). Carnage heart. <http://www.artdink.co.jp/>.

Atomic Games (1996). Close combat. <http://en.wikipedia.org/wiki/Close_Combat_(series)>.

Atomic Games (1998). Close combat: A bridge too far. <http://en.wikipedia.org/wiki/Close_Combat:_A_Bridge_Too_Far>.

Avery, P., & Michalewicz, Z. (2008) Adapting to human game play. In *Proceedings of the 4th international conference on computational intelligence and games, CIG'08*.

Bourg, D. M., & Seemann, G. (2004). *AI for game developers*. O'Reilly Media.

Buckland, M. (2004). *Programming game AI by example* (1st ed.). Jones & Bartlett Publishers<http://www.amazon.com/exec/obidos/redirect?tag=citeulike07-20&path=ASIN/1556220782>.

Cardamone, L., Lanzi, P. L., Loiacono, D., & Onieva, E. (2013). Advanced overtaking behaviors for blocking opponents in racing games using a fuzzy architecture. *Expert Systems and Application, 40*, 6447–6458.

CoolioNiato (2008). Light bot. <http://www.kongregate.com/games/coolio_niato/light-bot>.

DeLooze, L. L., & Viner, W. R. (2009). Fuzzy q-learning in a nondeterministic environment: Developing an intelligent ms. pac-man agent. In *Proceedings of the 5th international conference on computational intelligence and games, CIG'09* (pp. 162–169). Piscataway, NJ, USA: IEEE Press<http://dl.acm.org/citation.cfm?id=1719293.1719327>.

Illwinter Design, G. (2007). Dominions III. <http://www.illwinter.com/dom3/>.

El-Nasr, M. S., Yen, J., & Ioerger, T. R. (2000). Flame fuzzy logic adaptive model of emotions. *Autonomous Agents and Multi-Agent Systems, 3*, 219–257.

Epic (1999). Unreal tournament. <http://en.wikipedia.org/wiki/Unreal_Tournament>.

Epic (2003). Unreal tournament 2004. <http://en.wikipedia.org/wiki/Unreal_Tournament_2004>.

Epic Games (1998). Unreal. <http://en.wikipedia.org/wiki/Unreal>.

Fujii, S., Nakashima, T., Ishibuchi, H. (2008). A study on constructing fuzzy systems for high-level decision making in a car racing game (pp. 3626–3633). <http://dx.doi.org/10.1109/CEC.2008.4631289>.

Gabriyel Wong, J. W. (2006). A fuzzy-control approach to managing scene complexity. *Game Programming Gems, 6*.

Gameware Development, Creatures (1996). <http://www.gamewaredevelopment.com/creatures_index.php>.

Hastings, E., Guha, R., & Stanley, K. (2009). Automatic content generation in the galactic arms race video game. In *Proceedings of the 5th international conference on computational intelligence and games, CIG'09*.

Ho, D. T., & Garibaldi, J. (2008). A fuzzy approach for the 2007 cig simulated car racing competition. In *Proceedings of the 4th international conference on computational intelligence and games, CIG'08*.

Ho, D. T., & Garibaldi, J. (2008). A fuzzy approach for the 2007 cig simulated car racing competition. In *IEEE symposium on computational intelligence and games, CIG '08* (pp. 127–134). <http://dx.doi.org/10.1109/CIG.2008.5035631>.

Hsu, S. H., Kao, C.-H., & Wu, M.-C. (2009). Design facial appearance for roles in video games. *Expert Systems with Applications, 36*, 4929–4934.

Id Software (1999). Quake III arena. <http://en.wikipedia.org/wiki/Quake_III_Arena>.

Ishibuchi, H., Sakamoto, R., & Nakashima, T. (2003). Learning fuzzy rules from iterative execution of games. *Fuzzy Sets and Systems*.

Johnson, D., & Wiles, J. (2001). Computer games with intelligence. In *FUZZ-IEEE* (pp. 1355–1358). IEEE.

Johnson, D., & Wiles, J. (2001). Computer games with intelligence. In *In Proceedings of the 10th IEEE international conference on fuzzy systems* (pp. 61–68). IEEE.

Jones, D. G., & Dewdney, A. K. (1980). Core wars. <http://en.wikipedia.org/wiki/Core_War>.

Juul, J. (2002). The open and the closed: Games of emergence and games of progression. *Computer Games and Digital Cultures*.

Levillain, F., Orero, J., & Rifqi, M. (2010). Characterizing player's experience from physiological signals using fuzzy decision trees. In *Proceedings of the 6th international conference on computational intelligence and games, CIG'10*.

Li, Y., Musilek, P., & Wyard-Scott, L. (2004). Fuzzy logic in agent-based game design. In IEEE annual meeting of the fuzzy information, 2004. Proceedings NAFIPS '04.

Lionhead Studios (2001). Black&white. <http://lionhead.com/black-white/>.

Loiacono, D., Togelius, J., Lanzi, P., Kinnaird-Heether, L., Lucas, S., Simmerson, M., Perez, D., Reynolds, R., & Saez, Y. (2008). The wcci 2008 simulated car racing competition. In *IEEE symposium on computational intelligence and Games, CIG '08* (pp. 119–126). http://dx.doi.org/10.1109/CIG.2008.5035630.

Loiacono, D., Lanzi, P., Togelius, J., Onieva, E., Pelta, D., Butz, M., et al. (2010). The 2009 simulated car racing championship. *IEEE Transactions on Computational Intelligence and AI in Games, 2*, 131–147.

Lo, Y.-F., & Wen, M.-H. (2010). A fuzzy-ahp-based technique for the decision of design feature selection in massively multiplayer online role-playing game development. *Expert Systems with Applications, 37*, 8685–8693.

Maxis (2000). The sims. <http://en.wikipedia.org/wiki/The_Sims>.

McCuskey, M. (2000). Fuzzy logic for video games. *Game Programming Gems, 1*.

Merrick, K. E., & Maher, M. L. (2009). *Motivated reinforcement learning: Curious characters for multiuser games*. Springer.

Midway (1982). Ms. pac-man. <http://en.wikipedia.org/wiki/Ms._Pac-Man>.

Millington, I. (2006). *Artificial intelligence for games*. *The Morgan Kaufmann series in interactive 3D technology*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.

Mode 7 (2011). Frozen synapse. <http://www.frozensynapse.com/>.

Namco (1980). Pac-man. <http://en.wikipedia.org/wiki/Pac-Man>.

Namco (1995). Battlecity. <http://en.wikipedia.org/wiki/Battle_City_(video_game)>.

Nelson, M. (1980). Robocode. <http://robocode.sourceforge.net/>.

O'Brien, L. (1996). Fuzzy logic in games. *Game Developer Magazine*.

Ohsone, K., & Onisawa, T. (2008). Friendly partner system of poker game with facial expressions. In *Proceedings of the 4th international conference on computational intelligence and games, CIG'08*.

Onieva, E., Cardamone, L., Loiacono, D., & Lanzi, P. L. (2010). Overtaking opponents with blocking strategies using fuzzy logic. In G. N. Yannakakis & J. Togelius (Eds.), *CIG* (pp. 123–130). IEEE.

Onieva, E., Pelta, D. A., Alonso, J., Milanes, V., & Perez, J. 2009. A modular parametric architecture for the torcs racing engine. In *IEEE symposium on computational intelligence and games, CIG 2009* (pp. 256–262). <http://dx.doi.org/10.1109/CIG.2009.5286466>.

Perez, D., Recio, G., & Saez, Y. 2009. Evolving a fuzzy controller for a car racing competition. In *IEEE Symposium on Computational Intelligence and Games, CIG 2009* (pp. 263–270). <http://dx.doi.org/10.1109/CIG.2009.5286467>.

Pinto, H., & Alvares, L. O. (2006). Constructing a goal-oriented robot for unreal tournament using fuzzy sensors, finite-state machines, and extended behavior networks. *Game Programming Gems, 6*.

Positech Games (2012). Gratitious space battles. <http://positech.co.uk/gratuitousspacebattles/>.

Prieditis, A., & Dalal, M. (2006). Applying model-based decision-making methods to games: Applying the locus ai engine to quake III. *Game Programming Gems, 6*.

Real Time Battle (2006). <http://realtimebattle.sourceforge.net/>.

Shaout, A., King, B. W., & Reisner, L. A. (2006). Real-time game design of pac-man using fuzzy logic. *The International Arab Journal of Information Technology, 3*, 315–325.

Sweetser, P., & Wiles, J. (2002). Current ai in games: A review. *Australian Journal of Intelligent Information Processing Systems, 8*.

van Waveren. J. M. P. (2001). *The quake III arena bot* [Master's thesis]. University of Technology Delft.

Viana, R. (2012). Cargo bot. <http://twolivesleft.com/CargoBot/>.

Vyssotsky, V. A., Morris-Sr, R., & Mcllroy, M. D. (1961). Darwin. <http://en.wikipedia.org/wiki/Darwin_(programming_game)>.

Westra, J., & Dignum, F. (2009). Evolutionary neural networks for non-player characters in quake III. In *Proceedings of the 5th international conference on computational intelligence and games, CIG'09* (pp. 302–309). Piscataway, NJ, USA: IEEE Press<http://dl.acm.org/citation.cfm?id=1719293.1719345>.

Windward Studios (1997). Enemy nations.

Woodcock, S. (1999). *Game AI: The state of the industry*.

Woodcock, S. (1995–2000). Games making interesting use of artificial intelligence techniques, Game AI.

Yosemite Entertainment (1998). Police quest: Swat 2. <http://en.wikipedia.org/wiki/Police_Quest:_SWAT_2>.

Zachtronics Industries (2011). Spacechem. <http://www.spacechemthegame.com/>.

Zarozinski, M. (2001). Imploding combinatorial explosion in a fuzzy system. *Game Programming Gems, 2*.

Zarozinski, M. (2002). An open source fuzzy library. *AI Game Programming Wisdom*.