Orthogonally Evolved AI to Improve Difficulty Adjustment in Video Games

Arend Hintze[1(✉)], Randal S. Olson[2], and Joel Lehman[3]

[1] Michigan State University, Michigan, USA
hintze@msu.edu
[2] University of Pennsylvania, Pennsylvania, USA
[3] IT University of Copenhagen, Copenhagen, Denmark

Abstract. Computer games are most engaging when their difficulty is well matched to the player's ability, thereby providing an experience in which the player is neither overwhelmed nor bored. In games where the player interacts with computer-controlled opponents, the difficulty of the game can be adjusted not only by changing the distribution of opponents or game resources, but also through modifying the skill of the opponents. Applying evolutionary algorithms to evolve the artificial intelligence that controls opponent agents is one established method for adjusting opponent difficulty. Less-evolved agents (i.e., agents subject to fewer generations of evolution) make for easier opponents, while highly-evolved agents are more challenging to overcome. In this publication we test a new approach for difficulty adjustment in games: orthogonally evolved AI, where the player receives support from collaborating agents that are co-evolved with opponent agents (where collaborators and opponents have orthogonal incentives). The advantage is that game difficulty can be adjusted more granularly by manipulating two independent axes: by having more or less adept collaborators, and by having more or less adept opponents. Furthermore, human interaction can modulate (and be informed by) the performance and behavior of collaborating agents. In this way, orthogonally evolved AI both facilitates smoother difficulty adjustment and enables new game experiences.

Keywords: Difficulty adjustment · Coevolution · Evolutionary computation · Markov Networks

1 Introduction

A challenge in designing computer games is to match a game's difficulty appropriately to the skill level of a human player. Most commonly, game developers design explicit levels of difficulty from which a user can select. Evolving artificial intelligence (AI), i.e., applying evolutionary algorithms to adapt agents, has often been applied to improve video games [1,2], particularly for adjusting their difficulty [3,4]. Such difficulty adjustment approaches generally fall into two categories. In one, the player is immersed in a world where the computer-controlled

G. Squillero and P. Burelli (Eds.): EvoApplications 2016, Part I, LNCS 9597, pp. 525–540, 2016.
DOI: 10.1007/978-3-319-31204-0_34

game agents evolve in *real-time* as the game is played. In the other, opponent AI is evolved *offline*, and options for difficulty are extracted by exploiting the evolutionary history of the opponent AI (among many others [5–16]).

This work extends from the latter category, but instead of evolving offline only AI for opponent agents, AI is evolved for two kinds of game agents that have orthogonal motives. One class of AI agent opposes the player (called opponent agents), while the other class helps the player (called collaborator agents). Such AIs are evolved through competitive co-evolution, i.e., one opponent population and one collaborator population compete with each other. The idea is that pitting such populations against each other can result in an arms race [17,18] in which both agents become more competent as evolution progresses.

One naïve application of such orthogonally-evolved opponent and player AIs is to discard the evolutionary history of the player AI (because a human player will fill that role in the game), and to use only the evolutionary history of the evolved opponent AI to derive a range of opponent difficulties that can be deployed within the game. However, an interesting idea is to instead use *both* evolutionary histories. In particular, if co-evolution is conducted as a competition between a population of player-friendly collaborators (with similar capabilities as the player) and a population of player-antagonistic opponents, then the final game can include both evolved collaborative and opponent agents of different adaptedness. Here we investigate the advantages that such an orthogonal evolutionary approach has for difficulty adjustment.

The motivation is that a richer set of player experiences can result from players interacting both with evolved opponents and evolved collaborators. That is, the distinct evolutionary histories of the orthogonally-evolved populations yield two independent axes for difficulty adjustment. In this way, the player can interact with opponent and collaborator AIs taken from separate evolutionary time points. The hypothesis is that such orthogonal evolution gives game designers more options to adjust game difficulty and provide diverse player experiences.

Experiments in this paper are conducted through a browser-based game in which a player competes with evolved opponents and is assisted by evolved collaborators. In particular, a scientific predator-prey simulation [19] is adapted such that the player controls one prey agent among a group of AI-controlled prey collaborators with the objective of avoiding being consumed by an AI-controlled predator agent. Play tests conducted through Amazon's Mechanical Turk collected data relating player survival time to the adaptedness of both the AI-controlled predator and prey agents. Supporting the hypothesis, the results demonstrate that independently adjusting the level of adaptation of opponents and collaborators creates unique difficulty levels for players. The conclusion is that orthogonally-evolved AI may be a promising method for game designers to adjust game difficulty more granularly and to provide a wider variety of experiences for players.

2 Background

The next sections review previous mechanisms to create adjustable difficulty in video games, and the Markov Network encoding applied in the experiments to represent controllers for game agents.

2.1 Difficulty Adjustment in Video Games

Difficulty adjustment is important for video games, because how a user experiences (and potentially enjoys) a game is impacted directly by the fit between their capabilities and those necessary to progress in the game [4]. The traditional approaches are to either have a fixed level of difficulty, or to allow a player to choose from a set of hand-designed difficulties. However, a universal difficulty level may fail to satisfy many players; and hand-designed difficulties may require significant design effort, provide only a coarse means of adjustment, and require the player to self-rate their capabilities before interacting with the game. Thus, many methods for automatic difficulty adjustment have been explored [1,3,4,20].

One important facet of game difficulty is the adeptness of non-player character (NPC) agents in the game. For example, how far from optimality does the behavior of opponent agents stray? That is, the more optimal the opponent is, the more difficult the player challenge will be. This paper focuses on applying evolutionary algorithms (EA) to create agent controllers as a means to generate game scenarios that vary in difficulty. EAs are appropriate for automatic controller design because they provide a powerful and flexible mechanism to do so given only a measure of adeptness, and have therefor been used often in the past for such purposes [1,16,20].

Evolutionary approaches to difficulty adjustment can be categorized generally in two ways. In the first category, the computer-controlled agents evolve in real-time as the game is played. The idea is to dynamically alter the game AI based on player interaction, which can create unique and personalized player experiences [20–27]. However, such approaches require the game to be designed around AI adaptation, enabling new types of games but limiting their application to game AI in general. For example, in Creatures [21] the main game mechanic is to guide and teach a species of AI agents, while in NERO [23] a similarly-inspired mechanic is to train a battalion of agents to fight other ones. By their nature, such mechanics lead to unpredictable outcomes and can expose players to degenerate NPC behavior, which while compelling in their own right may also undermine a designer's ability to craft specific and predictable player experiences.

This study focuses on a second category in which AI for opponents is optimized offline, i.e., it remains unchanged during gameplay. The benefit of offline adaptation is that player experience can be more tightly controlled, enabling it potentially to be applied more broadly. One popular mechanism for such offline AI design is to use EAs to evolve agent controllers. In particular, if selection in an EA is oriented towards stronger AI behaviors, the difficulty of the game can then be adjusted by exploiting the evolutionary history of the opponent AI [5–16].

In this way, less evolved AI (e.g., from early generations of evolution) can serve as a player's opponent in early or easy levels, and more sophisticated AI (e.g., from later generations of evolution) can be featured in more difficult levels. However, most previous approaches focus singularly on the most optimal behavior evolved [8,9,16], and those that consider evolving interesting or diverse opponents [1,28] do not fully explore the possibilities enabled by competitive coevolution in this context. One such possibility (which is the central focus of this paper) is to leverage as a source of diverse difficulties the separate evolutionary trajectories of populations of agents with asymmetric abilities and conflicting motivations.

The next section reviews the encoding used to represent and evolve agent behaviors in the experiments.

2.2 Markov Networks

The experiment in this paper leverages a browser-game derived from the predator-prey simulation in [29]. Agents in the simulation (and thus the game) are controlled by Markov Networks (MNs), which are probabilistic controllers that makes decisions about how an agent interacts with its environment. Because a MN is responsible for the control decisions of its agent, it can be thought of as an *artificial brain* for the agent it controls. Although MNs are the particular artificial brain applied in the simulation, other methodologies for evolving controllers could also be used, such as neuroevolution or genetic programming. This section briefly describes MNs, but a more detailed description can be found in [30].

Agents in the game have sensors and actuators, as shown in Fig. 1. Every simulation time step, the MNs receive input via those sensors, perform a computation on inputs and any hidden states (i.e., their internal memory), then place the result of the computation into hidden or output states (e.g., actuators). When MNs are evolved with a GA, mutations affect (1) which states the MN pays attention to as input, (2) which states the MN outputs the result of its computation to, and (3) the internal logic that converts the input into the corresponding output.

When agents are embedded into a game simulation, sensory inputs from its retina are input into its MN every simulation step (labeled "retina" and "Markov Network", respectively in Fig. 1). The MN is then activated, which allows it to store the result of the computation into its hidden and output states for the next time step. MNs are networks of Markov Gates (MGs), which perform the computation for the MN. In Fig. 2, we see two example MGs, labeled "Gate 1" and "Gate 2." At time t, Gate 1 receives sensory input from states 0 and 2 and retrieves state information (i.e., memory) from state 4. At time $t+1$, Gate 1 then stores its output in hidden state 4 and output state 6. Similarly, at time t Gate 2 receives sensory input from state 2 and retrieves state information in state 6, then places its output into states 6 and 7 at time step $t+1$. When MGs place their output into the same state, the outputs are combined into a single

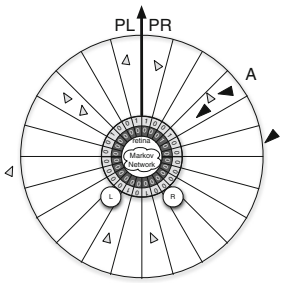

Fig. 1. An illustration of the agents in the model. Light grey triangles are prey agents and the dark grey triangles are predator agents. The agents have a 360° limited-distance retina (200 virtual meters) to observe their surroundings and detect the presence of other agents. The current heading of the agent is indicated by a bold arrow. Each agent has its own Markov Network, which decides where to move next based off of a combination of sensory input and memory. The left and right actuators (labeled "L" and "R") enable the agents to move forward, left, and right in discrete steps.

output using the OR logic function. Thus, the MN uses information from the environment and its memory to decide where to move in the next time step $t+1$.

In a MN, states are updated by MGs, which function similarly to digital logic gates, e.g., AND & OR. A digital logic gate, such as XOR, reads two binary states as input and outputs a single binary value according to the XOR logic. Similarly, MGs output binary values based on their input, but do so with a probabilistic logic table. Table 1 shows an example MG that could be used to control a prey agent that avoids nearby predator agents. For example, if a predator is to the right of the prey's heading (i.e., PL = 0 and PR = 1, corresponding to the second row of this table), then the outputs are move forward (MF) with a 20 % chance, turn right (TR) with a 5 % chance, turn left (TL) with a 65 % chance, and stay still (SS) with a 10 % chance. Thus, due to this probabilistic input-output mapping, the agent MNs are capable of producing stochastic agent behavior.

Table 1. An example MG that could be used to control a prey agent which avoids nearby predator agents. "PL" and "PR" correspond to the predator sensors just to the left and right of the agent's heading, respectively, as shown in Fig. 1. The columns labeled P(X) indicate the probability of the MG deciding on action X given the corresponding input pair. MF = Move Forward; TR = Turn Right; TL = Turn Left; SS = Stay Still.

PL	PR	P(MF)	P(TR)	P(TL)	P(SS)
0	0	0.7	0.05	0.05	0.2
0	1	0.2	0.05	0.65	0.1
1	0	0.2	0.65	0.05	0.1
1	1	0.05	0.8	0.1	0.05

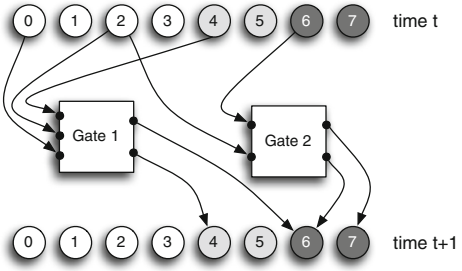

Fig. 2. An example Markov Network (MN) with four input states (white circles labeled 0-3), two hidden states (light grey circles labeled 4 and 5), two output states (dark grey circles labeled 6 and 7), and two Markov Gates (MGs, white squares labeled "Gate 1" and "Gate 2"). The MN receives input into the input states at time step t, then performs a computation with its MGs upon activation. Together, these MGs use information about the environment, information from memory, and information about the MN's previous action to decide where to move next.

A circular string of bytes is used to encode the genome, which contains all the information necessary to describe a MN. The genome is composed of *genes*, and each gene encodes a single MG. Therefore, a gene contains the information about which states the MG reads input from, which states the MG writes its output to, and the probability table defining the logic of the MG. The start of a gene is indicated by a *start codon*, which is represented by the sequence (42, 213) in the genome.

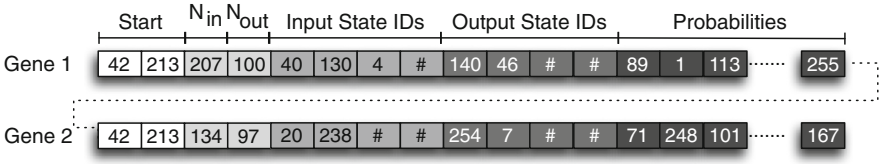

Fig. 3. Example circular byte strings encoding the two Markov Gates (MGs) in Fig. 2, denoted Gene 1 and Gene 2. The sequence (42, 213) represents the beginning of a new MG (white blocks). The next two bytes encode the number of input and output states used by the MG (light grey blocks), and the following eight bytes encode which states are used as input (medium grey blocks) and output (darker grey blocks). The remaining bytes in the string encode the probabilities of the MG's logic table (darkest grey blocks).

Figure 3 depicts an example genome. After the start codon, the next two bytes describe the number of inputs (N_{in}) and outputs (N_{out}) used in this MG, where each $N = 1 + (\text{byte} \mod N_{\text{max}})$. Here, $N_{\text{max}} = 4$. The following N_{max} bytes specify which states the MG reads from by mapping to a state ID number with the equation: $(\text{byte} \mod N_{\text{states}})$, where N_{states} is the total number of

input, output, and hidden states. Similarly, the next $N_{\max}$ bytes encode which states the MG writes to with the same equation as N_{in}. If too many inputs or outputs are specified, the remaining sites in that section of the gene are ignored, designated by the # signs. The remaining $2^{N_{\text{in}}+N_{\text{out}}}$ bytes of the gene define the probabilities in the logic table.

All evolutionary changes such as point mutations, duplications, deletions, or crossover are performed on the byte string genome. During a point mutation, a random byte in the genome is replaced with a new byte drawn from a uniform random distribution. If a duplication event occurs, two random positions are chosen in the genome and all bytes between those points are duplicated into another part of the genome. Similarly, when a deletion event occurs, two random positions are chosen in the genome and all bytes between those points are deleted. Crossover for MNs was not implemented in this experiment to allow for a succinct reconstruction of the line of descent of the population, which was important in the original study [29].

3 Approach

In typical applications of video game difficulty adjustment through evolved AI, only one class of AI (typically the opponent) is evolved, and the evolutionary history of the evolved AI yields a variety of differentially-adapted AIs. Near the beginning of evolutionary training we expect AIs to be incapable or maladapted, while after many generations of selection the AI becomes increasingly competent at performing the task it was selected for. This range of behaviors forms a continuum from which one can tailor the difficulty of player game experiences. Here instead of evolving only a single population of opponent AI agents, we co-evolve both the opponent agent and collaborative agents that help the player; these distinct agent types can have different capabilities and will have orthogonal fitness functions (because their motivations are in conflict).

The advantage of orthogonally-evolved AI is that it can enable players to interact not only with collaborative and opponent AIs taken from the same generation of evolution, but also with agents taken from separate, arbitrary generations. For example, the player can play not only with opponents and collaborators that are both capable (i.e., opponents and collaborators taken from the end of an evolutionary run), but can also face a more difficult situation if a well-adapted opponent is combined with a weakly-adapted and largely incapable team of player-collaborative agents. Or conversely, to engineer an easier game experience, a well-adapted collaborating team can be combined with an incapable opponent taken from an early generation of its evolution. The idea is that combining opponents and collaborators from different points of evolution will result in increased possibilities for player game experiences, as illustrated by Fig. 4.

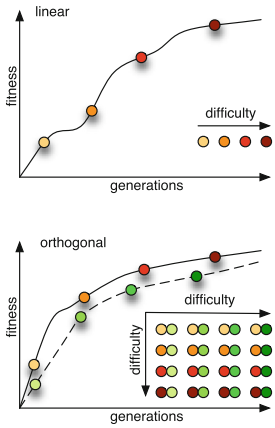

Fig. 4. Comparison of linear vs. orthogonal evolution of AI. The top figure (linear) shows a typical application of evolved AI. The difficulty (ability of the evolved AI) increases with generations of evolution. The bottom Figure (orthogonal) shows an example of orthogonal evolution where two populations are co-evolved with orthogonal incentives. Because AIs from both populations can be mixed arbitrarily, many more game situations can be constructed.

4 Experiment

The main hypothesis explored here is that the described method of using orthogonally evolved AIs helps to expand options for game difficulty. One means to test this hypothesis is for players to interact in a game setting with various mixtures of adapted and unadapted AIs for both opponents and collaborators. If the hypothesis is true, the expectation is that player performance will vary over all tested combinations. Conversely, if the hypothesis is false then there should be no additional significant differences in player performance from varying agent adapatedness across opponents and collaborators.

Testing this hypothesis requires a video game implementation with NPCs that can play alongside the player, which may not be possible or appropriate in every game. In this paper, the particular video game used for experimentation is derived from a simple predator-prey simulation from [29]. In the original simulation, predator and prey agents are controlled by evolved Markov networks inspired by computational abilities of biological brains. A single predator on a 2d surface is evolved to catch as many of the coexisting prey agents as possible. In contrast, the group of prey agents is collectively evolved to resist being caught (for detailed explanation see: [29]). In this way, the motivations of the predator and prey are orthogonal. Over generations, the predator evolves to more efficiently catch prey agents by learning to attack the outside of a swarm of prey agents more consistently. Prey agents in the simulation evolve to swarm together, because those that can not successfully swarm become isolated from the rest of the prey agents, and are more easily caught. The resulting evolved

swarming behavior is explained by the selfish herd hypothesis [19,31], which the simulation was designed to investigate.

A game was created which implemented the same simulation rules, but substituted a human player for one of the swarming prey agents. The human's objective is to evade the predator as long as possible (Fig. 5). All of the other agents in the game are controlled by MNs; importantly, the non-player prey agents are controlled by MNs taken from a separate population (and potentially a separate point in that population's evolutionary history) from that of the predator agent.

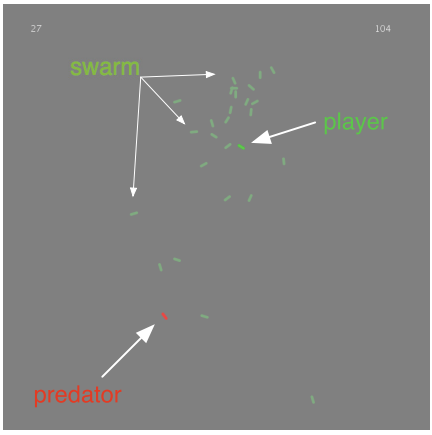

Fig. 5. Typical game situation. The player uses the keyboard's left and right keys to control the player agent (bright green) within a group of other collaborating agents (dim green). The predator agent (red) can kill the other agents if close enough to them. The player has 120 s (remaining time is shown at the top right) to evade the predator. Note that the number of remaining collaborating agents is shown at the top left. Figure is best viewed in color (Color figure online).

The simulation was adapted into a browser-based game, and human players were recruited from Amazon's Mechanical Turk to play. In the game, a group of swarming agents is antagonized by a predator agent. The player acts as one of the swarm agents and is tasked with avoiding the predator. All agents other than the player (i.e., the predator and the remaining prey agents) are controlled by previously evolved Markov networks (i.e., no evolution takes place while the game is played). The predators were evolved to catch prey while prey agents were evolved to flee predators and eventually swarm together to avoid capture. Predator and prey Markov networks can either come from a relatively *unevolved* stage (generation 900) or from a relatively *evolved* stage (generation 1,900). At the beginning of the game one of the four possible combinations of adaptation for the predator and prey Markov networks is randomly chosen. The game ends either if the predator catches the player or after 120 s pass.

4.1 Experimental Details

First, several evolutionary runs were performed using the EOS framework [29] with its default settings. Organisms on the line of descent [32] from the predator as well as from the prey populations were saved every 25 generations. A particular evolutionary run was then selected that showed a large gain in swarming and predation capability at end of the run compared to the beginning. The motivation was to ensure a significant recognizable difference between the capabilities of the AI over its evolutionary history. Agents were chosen from two time points (generations 900 and 1,900) to represent different levels of adaptation (referred to here as *evolved* and *unevolved*). Detailed description of the simulation and evolutionary setup can be found in [29].

The game was implemented in Processing [33] and can be run in a web browser using ProcessingJS. 200 Amazon Mechanical Turk users were recruited to play the game[1], which was embedded in a website. At the start of the game, one of the four possible experimental conditions was randomly chosen: unevolved prey & unevolved predator, unevolved prey & evolved predator, evolved prey & unevolved predator, or evolved prey & evolved predator. Each player was required to play for either 120 s or until caught by the predator. The game difficulty implicitly increases with time because as the predator decimates the prey agents, the player is increasingly likely to be hunted by the predator. At the end of the game how long the player survived (at best 120 s) and how many other prey agents were still alive at that point, was recorded.

5 Results

The results of the experiment show that in the four tested combinations, average player survival time for each individual combination significantly differs from each of the others (Fig. 6). Intuitively, one might expect that the game's difficulty depends mostly upon the predator's ability to catch prey, because only the predator poses direct danger to the player. This intuition suggests that the two environments with the unevolved predator should be the easiest and the two environments containing the evolved predator should be the most challenging. Interestingly, however, the results instead show that the difficulty of the game depends more on the ability of the prey to swarm than on the ability of the predator to catch. The more evolved the prey is, the easier the game becomes, while the predator's ability is only of secondary importance. In this way, the results highlight that evolving opponents and collaborators with orthogonal objectives, like in this predator prey example, indeed allows for more combinations of difficulty (Fig. 4). Thus, choosing AIs for distinct roles from different evolutionary time points can facilitate a smoother (and potentially more complex) progression of game difficulties.

[1] Our study was exempt by the Office of Research Support at the University of Texas at Austin. Number 2013-09-0084. Due to the exemption, by not taking any personal data, and due to the anonymity of the subjects, we did not need written consent.

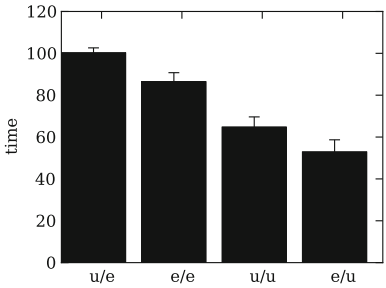

Fig. 6. Comparison of player performance. The average time in seconds players survived before being caught by the predator, for four different conditions, from left to right: (u/e) predator from an early point in evolution (generation 900) paired with evolved prey (generation 1,900), (e/e) both AIs from a late point in evolution (generation 1,900), (u/u) both AIs from an early point in evolution (generation 900), (e/u) predator from a late point in evolution (generation 1,900) paired with unevolved prey (generation 900)

Further, in this kind of interactive environment it is not only the swarming agents that influence player survivability. Conversely, the player actions can effect how well the swarm agents survive. When comparing the swarm agents' survival rate in the presence of the player, to a situation where the player's agent is controlled by the same AI as all the other prey agents, the result is that player interactions reduce the prey survivability only when the prey agents are controlled by the more evolved AI (Fig. 7). When prey agents are taken from an early generation the effect is more subtle. This result shows that not only can the difficulty the player experiences can be modulated by the degree of adaptation of prey agents, but that evolved prey agents are also more influenced by player actions if they are more evolved themselves.

Note that in order to assess the influence players have on the survivability of prey agents in the game, the following exponential decay function was used:

$$n = e^{a - \frac{a}{1-(\frac{x}{a})^2}} \tag{1}$$

As an approximation for the number of organisms alive over time in the presence of the player, the number of organisms alive after 120 s or at the time point the player died was used to fit Eq. 1. To assess prey survivability without the presence of the player, for each of the four possible conditions the game was run without a player, and the player organism was controlled by the same AI as the other swarm agents. For these runs, the data was aggregated to estimate the average number of prey, and also was fit to Eq. 1.

For both data sets the residuals against each of the fitted functions were computed and a Mann Whitney U test was performed to show that the residuals of each others fit were significantly different from one another.

6 Discussion

First, it is important to note that this approach of controlling swarming agents in video games with evolved MNs contrasts with more conventional approaches. For example, the Boids algorithm [34] is commonly applied to control swarming agents, and works by uniformly applying three elementary forces (separation, alignment, and cohesion) to each agent in the swarm. These simple forces govern the entire swarm and dictate where each individual moves. Swarm behavior can be varied by adjusting a limited set of parameters (e.g., the radius of influence, the force applied, and the turning rate). However, the potential for novelty is limited because such parameters do not change the fundamental underlying forces. Furthermore, in general adapting the Boids model to particular capabilities of antagonisitic agents (like the predator) requires specific human insight. To overcome the limitations of simpler models (like the Boids algorithm) the EOS model is applied here, where agents are individually controlled by an evolved Markov network (which could in theory approximate the Boids algorithm through learning). Such Markov networks have to our knowledge not been applied to video games before, and our work demonstrates their feasibility. An interesting benefit of such networks is that in contrast to more computationally demanding AI algorithms, once evolved Markov networks are computationally tractable to embed even within javascript browser games. Each agent in a swarm is controlled by its own Markov network, allowing for novelty and variability between swarm agents. In some video games it is likely more interesting and visually appealing for the player to have heterogeneous swarms, which the use of evolved Markov networks enables.

More broadly, the results present an elaboration on previous approaches to difficulty adjustment in video games, which primarily focus on the evolution of opponent AI to create a single axis for difficulty adjustment: The more generations over which the opponent evolves, the more challenging it is to overcome. In contrast, coevolving the opponent agent with collaborative agents enables a wider spectrum of possibilities. Instead of exploiting only the evolutionary history of the opponent to adjust difficulty, the player can engage with different combinations of opponents and collaborators, which in the case studied here allows for smoother difficulty adjustment. In this way, the results demonstrate that coevolving separate populations of agents with orthogonal objectives is a viable method to improve difficulty adjustment.

One surprising result is that the difficulty in the explored game depends more on the capability of the collaborator (prey) than on the opponent (predator). While it is difficult to pinpoint the exact reasons for such behavior, it appears that when the player interacts with collaborators that can effectively swarm and evade the opponent, the player has an effective example to mimic, which shortens player learning time and thereby improves the player's performance. Another reason for the importance of the collaborative agents is that in the swarming example applied here, the evolved swarm of collaborators actively aggregates, and thereby protects the player as long as the player stays within the bounds of the swarm. This type of altruistic group behavior can improve the

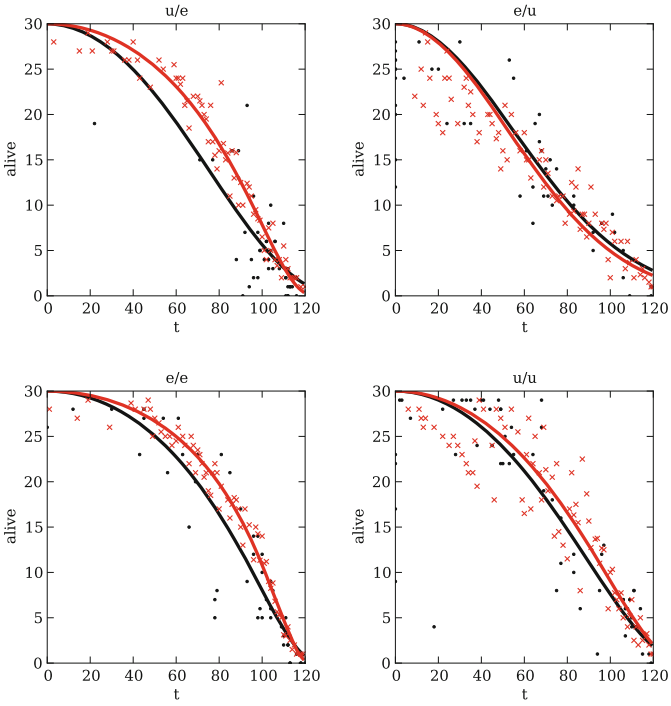

Fig. 7. Comparison of prey survivability. The figure compares the survival over time of prey agents, when interacting with a human-controlled prey, or when interacting only with computer-controlled agents. Each plot is titled by two letters that indicate the adaptedness of both the predator (first letter) and prey (second letter). The letter itself indicates whether agents come from an early point in evolution (u for "unevolved"; generation 900) or a late point in evolution (e for "evolved"; generation 1,900). Thus the top left plot titled "u/e" reflects pairing a predator from generation 900 with prey from generation 1,900). Black dots indicate the number of prey living when the player dies (when interacting with a human-controlled agent) and red Xs indicate the average number (over 5 sampled runs) of prey alive when there is no human interference. Curves are fit to the data points, and their color reflects the color of the data points from which they are derived. The distribution of the residuals between the red and black data points and their corresponding fit is significantly different between all four cases ($p < 0.001$). The conclusion is that player interaction influences the effectiveness of the other prey agents (Color figure online).

player's survivability, thereby providing an interesting example of an emergent game mechanism that is automatically discovered by orthogonal coevolution.

An additional idea explored in this paper is transplanting agents evolved originally in a scientific setting to create or enhance an entertaining video game. In particular, this paper creates a video game by exapting AI agents evolved in a simulation exploring biological hypotheses for the evolution swarming behavior [19]. A possible benefit is that through directly interacting with evolved AI

in a game-like environment, a reader of a paper can potentially more easily understand the paper, as well as better judge the quality and sophistication of the results. In this way, video games based on scientific simulations can potentially assist wider understanding of scientific results by non-experts. Future work will investigate the plausibility of such ideas.

A limitation of the approach is that it may not always be appropriate or easy to formulate a game situation in terms of orthogonally evolved populations. However, computer games are increasingly multiplayer and increasingly incorporate massive game worlds, providing natural opportunities for game designers to augment games with collaborative agents in addition to more typical confrontational opponents. In particular, MMORPGs not only commonly contain companions and enrich their environments using NPCs, but such games (or perhaps real time strategy games) may also benefit from integrating evolutionary mechanisms *into* their gameplay, and thereby allow opponents and NPCs to evolve as a game progresses. As shown in the results, player action indeed influences prey performance, which supports the idea that agents can adapt to players within the game.

7 Conclusion

This paper introduced the concept of orthogonal coevolution, and tested its effectiveness in a browser-based game adapted from a scientific simulation. The results demonstrate that evolving opponents in conjunction with evolved companions can lead to smoother difficulty adjustment and allow players to experience more varied situations. The conclusion is that such orthogonal coevolution may be a promising approach for adjusting video game difficulty.

Acknowledgments. We would like to thank Chris Adami for insightful comments and discussion of the project.

References

1. Yannakakis, G.N.: AI in Computer Games (2006)
2. Browne, C.: Evolutionary Game Design. Springer, Heidelberg (2011)
3. Spronck, P., Sprinkhuizen-Kuyper, I., Postma, E.: Difficulty scaling of game AI. In: Intelligent Games (2004)
4. Hunicke, R., Chapman, V.: AI for dynamic difficulty adjustment in games. In: Challenges in Game Artificial Intelligence AAAI (2004)
5. Overholtzer, C.A., Levy, S.D.: Evolving AI opponents in a first-person-shooter video game. In: AAAI Proceedings of the 20th National Conference on Artificial Intelligence (2005)
6. Cole, N., Louis, S.J., Miles, C.: Using a genetic algorithm to tune first-person shooter bots. Trans. IRE Prof. Group Audio **1**, 131–139 (2004)
7. Tan, T.G., Anthony, P., Teo, J., Ong, J.H.: Neural network ensembles for video game AI using evolutionary multi-objective optimization. In: Transactions of the IRE Professional Group on Audio, pp. 605–610, December 2011

8. Yau, Y.J., Teo, J., Anthony, P.: Pareto evolution and co-evolution in cognitive game AI synthesis. In: Obayashi, S., Deb, K., Poloni, C., Hiroyasu, T., Murata, T. (eds.) EMO 2007. LNCS, vol. 4403, pp. 227–241. Springer, Heidelberg (2007)
9. Yau, Y.J., Teo, J., Anthony, P.: Pareto evolution and co-evolution in cognitive neural agents synthesis for Tic-Tac-Toe. In: IEEE Symposium on Computational Intelligence and Games, pp. 304–311. IEEE (2007)
10. Mayer, H.A., Maier, P.: Coevolution of neural go players in a cultural environment. Trans. IRE Prof. Group Audio **2**, 1012–1017 (2005)
11. Lubberts, A., Miikkulainen, R.: Co-evolving a go-playing neural network. In: Algorithms Upon Themselves (2001)
12. Chellapilla, K., Fogel, D.B.: Evolving an expert checkers playing program without using human expertise. IEEE Trans. Evol. Comput. **5**(4), 422–428 (2001)
13. Chellapilla, K., Fogel, D.B.: Evolution, neural networks, games, and intelligence. In: Proceedings of the IEEE, pp. 1471–1496 (1999)
14. Lim, C.U., Baumgarten, R., Colton, S.: Evolving behaviour trees for the commercial game DEFCON. In: Di Chio, C., Cagnoni, S., Cotta, C., Ebner, M., Esparcia-Alcazar, A.I., Goh, C.-K., Merelo, J.J., et al. (eds.) EvoApplicatons 2010. LNCS, vol. 6024, pp. 100–110. Springer, Heidelberg (2010)
15. Hagelbäck, J., Johansson, S.J.: Using multi-agent potential fields in real-time strategy games. In: AAMAS 2008: Proceedings of the 7th International Joint Conference on Autonomous Agents and Multiagent Systems, International Foundation for Autonomous Agents and Multiagent Systems, May 2008
16. Priesterjahn, S., Kramer, O., Weimer, A., Goebels, A.: Evolution of human-competitive agents in modern computer games. In: IEEE International Conference on Evolutionary Computation, pp. 777–784. IEEE, November 2005–2006
17. van Valen, L.: A new evolutionary law. Evol. Theor. **1**, 1–30 (1973)
18. Bell, G.: The Masterpiece of Nature: The Evolution and Genetics of Sexuality. CUP Archive (1982)
19. Olson, R.S., Knoester, D.B., Adami, C.: Critical interplay between density-dependent predation and evolution of the selfish herd. In: GECCO 2013: Proceeding of the 15th Annual Conference on Genetic and Evolutionary Computation Conference, ACM Request Permissions, July 2013
20. Yannakakis, G.N., Hallam, J.: Evolving opponents for interesting interactive computer games. In: From Animals to Animats (2004)
21. Grand, S., Cliff, D., Malhotra, A.: Creatures: artificial life autonomous software agents for home entertainment. In: AGENTS 1997: Proceedings of the 1st International Conference on Autonomous Agents, ACM, February 1997
22. Pollack, J., Blair, A.: Co-evolution in the successful learning of backgammon strategy. Mach. Learn. **32**, 225–240 (1998)
23. Stanley, K.O., Bryant, B.D., Miikkulainen, R.: Evolving neural network agents in the NERO video game. In: Proceedings of the IEEE (2005)
24. Hastings, E.J., Guha, R.K., Stanley, K.O.: Evolving content in the galactic arms race video game. In: IEEE Symposium on Computational Intelligence and Games (CIG), pp. 241–248. IEEE (2009)
25. DeLooze, L.L., Viner, W.R.: Fuzzy Q-learning in a nondeterministic environment: developing an intelligent Ms. Pac-Man agent. In: CIG 2009: Proceedings of the 5th International Conference on Computational Intelligence and Games. IEEE Press, September 2009
26. Handa, H.: Constitution of Ms. PacMan player with critical-situation learning mechanism. Int. J. Knowl. Eng. Soft Data Paradig. **2**(3), 237–250 (2010)

27. Tong, C.K., Hui, O.J., Teo, J., On, C.K.: The evolution of gamebots for 3D first person shooter (FPS). Transactions of the IRE Professional Group on Audio, pp. 21–26, September 2011
28. Agapitos, A., Togelius, J., Lucas, S.M., Schmidhuber, J., Konstantinidis, A.: Generating diverse opponents with multiobjective evolution. In: IEEE Symposium on Computational Intelligence and Games, CIG 2008, pp. 135–142. IEEE (2008)
29. Olson, R.S., Hintze, A., Dyer, F.C., Knoester, D.B., Adami, C.: Predator confusion is sufficient to evolve swarming behaviour. J. Roy. Soc. Interface **10**(85), 20130305 (2013)
30. Marstaller, L., Hintze, A., Adami, C.: The evolution of representation in simple cognitive networks. Neural Comput. **25**(8), 2079–2107 (2013)
31. Hamilton, W.D.W.: Geometry for the selfish herd. J. Theor. Biol. **31**(2), 295–311 (1971)
32. Lenski, R.E., Ofria, C., Pennock, R.T., Adami, C.: The evolutionary origin of complex features. Nature **423**(6), 139–144 (2003)
33. Fry, B., Reas, C.: Processing Library for Visual Arts and Design
34. Toner, J., Tu, Y.: Flocks, herds, and schools: a quantitative theory of flocking. Trans. IRE Prof. Group Audio (April 1998)