Simulating Multiple Character Interactions with Collaborative and Adversarial Goals

Hubert P.H. Shum, Taku Komura, and Shuntaro Yamazaki, *Member*, *IEEE*

Abstract—This paper proposes a new methodology for synthesizing animations of multiple characters, allowing them to intelligently compete with one another in dense environments, while still satisfying requirements set by an animator. To achieve these two conflicting objectives simultaneously, our method separately evaluates the competition and collaboration of the interactions, integrating the scores to select an action that maximizes both criteria. We extend the idea of min-max search, normally used for strategic games such as chess. Using our method, animators can efficiently produce scenes of dense character interactions such as those in collective sports or martial arts. The method is especially effective for producing animations along story lines, where the characters must follow multiple objectives, while still accommodating geometric and kinematic constraints from the environment.

Index Terms—Character animation, character interaction.

1 Introduction

Methods, which synthesize animations of multiple characters competitively interacting are in high demand in the computer animation and game industries. Due to the difficulties associated with capturing dense interactions through motion capture, methods based on optimization have been applied to combine singly captured motions into competitive interactions. In our preliminary research [1], we reported that the min-max-based optimization is effective for simulating competitive interactions of multiple characters.

One major difficulty is that from the animator's point of view, such scenes have counteracting requirements—they wish to show serious fighting between the characters at the local level, while maintaining precise control over the scene by specifying the location and movements of the crowd at the global level. This means that the characters need to compete for their own interests, while collaborating with one another to achieve the requirements of the animator. Our previous approach [1] and other optimization-based approaches for multicharacter control [2], [3] fall short at managing the cooperativeness and the competitiveness simultaneously. In these methods, an objective function is defined based on the interests of individual characters and they select actions that benefit them the most in the future. If we add a cooperative objective function that rewards the characters for following the instruction of the animators, the competing characters will try to penalize each other in achieving these goals. As a result, they do not co-operate well to follow the animator's plan.

- *H.P.H. Shum is with RIKEN, 2-1 Hirosawa, Wako, Saitama, Japan, 351-0198. E-mail: hubertshum@riken.jp.*
- *T. Komura is with the University of Edinburgh, 10 Crichton Street, Edinburgh, UK EH8 9YL. E-mail: tkomura@inf.ed.ac.uk.*
- *S. Yamazaki is with AIST, DHRC, 2-3-26, Aomi, Koto-ku, Tokyo, Japan, 135-0064. E-mail: shun-yamazaki@aist.go.jp.*

Manuscript received 9 Jan. 2010; revised 18 June 2010; accepted 23 Sept. 2010; published online 7 Dec. 2010.
Recommended for acceptance by M.C. Lin.
For information on obtaining reprints of this article, please send e-mail to: tvcg@computer.org, and reference IEEECS Log Number TVCG-2010-01-0003.
Digital Object Identifier no. 10.1109/TVCG.2010.257.

In this research, we propose a new method to achieve competitiveness and cooperativeness, as an extension to our previous min-max-based approach [1]. As opposed to a combined objective function in [1], the competitiveness and cooperativeness of the actions are evaluated separately by different objective functions (see Section 6). Our proposed method integrates the functions during the evaluation process (see Section 5).

One of the advantages of our system is that it is an interactive system that can reflect updates of the motion data set, action's scores, and the parameters of the constraints/objective function immediately. This feature is highly demanded by animators. Previous methods based on precomputation [4] or learning [2], [3], [5] require a huge amount of recomputation to reflect any updates of the parameters or data set. In order to achieve this, we choose to use a short horizon optimization using a rich set of actions. This increases the controllability of the characters and avoids wobbling or the repetition of similar series of actions due to quantization error, which tend to happen when using precomputation-based approaches [2], [4], [5] for persistent long interactions.

To demonstrate the effectiveness of our method, we simulate various competitive interactions between multiple characters, including boxing matches, sword fighting, chasing one another, and a mass-game scene, where the characters locally fight seriously with each other while moving based on a predefined rule to create a large-scale texture. We also show that the method is effective for creating animations along story lines in which high-level instructions are given.

2 Related Work

Motion editing and synthesis has become a huge-research area with many applications in computer graphics, robotics, and biomechanics. Recently, a lot of data-driven techniques to edit, retarget [6], [7], [8], or synthesize new sequences of character motion using precaptured motion data [9], [10],

[11], [12], [13], [14] are being proposed. The motion graph approach [9], [10], [11] is a method to interactively reproduce continuous motions of characters, based on a graph, that is, automatically generated from captured motion data. Since the motion graph produces a lot of edges and nodes without any context, it becomes difficult to control the character as the user wishes. Recently, therefore, works to resolve such problems by introducing hierarchical structures [15], [16], and interpolating the motions in the same category [14], [17], [18] are proposed. Most of these methods handle characters in the scene individually and extensions are needed to apply them to control multiple characters closely interacting with one another.

Recently, techniques for handling close interactions of multiple characters are attracting research. Scenes of this type are difficult to handle because of the large degrees of freedom, the close contacts that can cause penetrations of the bodies, and the complexity of selecting an optimal action that results in realistic animation. Here, we review different methods for creating scenes of multiple characters closely interacting with one another.

2.1 Capturing Multicharacter Animation

Some research has been done in capturing the motions of two characters closely interacting and synthesizing new motions using probabilistic methods [19], [20]. Since the motions of several people have to be captured simultaneously, there are limitations on the types of actions, which can be successfully recorded. Therefore, approaches to individually capture the motions and combine them to simulate close interactions have been developed.

2.2 Combining Singly Captured Motions

When simulating the interactions of multiple characters based on individually captured motions, the main problem is determining when to pick a particular action. One solution is to define a reward function that evaluates how much each action benefits the character in each state—many methods have been proposed in this direction. Lee and Lee [2] simulate a scene of two boxers fighting with each other using singly captured shadow boxing. They propose functions to guide the characters to approach and hit their opponent. Liu et al. [21] alternately computes the motions of individual characters in close contact by using space-time optimization. Treuille et al. [3] propose a method to control pedestrian characters to avoid one another using walking motion. Shum et al. [1] propose to use game tree expansion to intelligently control characters. This paper extends the work of Shum et al. [1] to generate characters with counteracting objectives.

2.3 Reinforcement Learning

Among the research synthesizing multicharacter animations by optimization [2] and [3] are based on reinforcement learning. In reinforcement learning [22], rewards are defined for each character and the characters select their actions so that their accumulated reward in the future is maximized. The rewards and transitions to different states are examined and the policy that determines the action at every state is precomputed. As a result, the characters can select the best action in real time. It has also been used to control pedestrians to avoid obstacles in the streets [23], [24], training a computer-controlled fighters [25], [26], and to simulate cooperative and competitive interactions between characters [5]. There are several problems with using reinforcement learning to simulate the close interactions of characters: First, it requires a huge amount of precomputation to find the optimal actions at every state—basically the state space increases exponentially proportional to the dimensionality. As a result, abstraction of the state based on sampling [2], [5], [24], or basis functions [3] must be used. If the abstraction is too rough, the actions are not optimal and the resulting animation can appear awkward. Second, if there are changes in system parameters such as the objective functions, body size, and available actions, the precomputed results are no longer valid—the time consuming process of evaluating all the state transitions and rewards must be repeated from the beginning. Therefore, in this study, instead of applying quantization and precomputation, we examine the precise status of each character by expanding the search tree based on the available set of actions during runtime. Although, the simulation has to be paused from time to time for the tree expansion process, the system can adapt to the updates in the system parameters and return the optimal series of actions.

2.4 Controlling Scenes of Multiple Characters

The advantage of our approach for controlling characters is that we can simulate realistic competition while enabling cooperation among characters to satisfy the directions given by the animator. Although, many methods enable animators to edit the trajectories of a crowd [27], arrange the formation of characters [28], [29], or design / edit the interactions of characters [4], [30], [31], there has been no approach, which can construct a scene, where the characters competing locally while co-operating to achieve the demands from animators.

A recent research by Kim et al. [32] also introduces a user-friendly interface for animators to plan and adjust interactions among multiple characters. However, since the characters have no intelligence, every interaction has to be specified manually by the animator. Our research focuses on controlling the characters intelligently so that they automatically interact and follow high-level commands.

3 System Overview

In our system, every character has its own action level motion graph [1] in which the edges represent semantic actions. The interactions of each character are simulated by expanding the game tree and evaluating the states in the future. For the evaluation of the future states, we prepare two functions—the competitive function and the cooperative function. The former simulates intelligence for competitive behaviors, while the latter allows the characters to achieve common goals such as instructions given by an animator. The functions are integrated in the min-max framework to decide the action of the character.

The outline of our system is shown in Fig. 1. It consists of five steps:

1. Capture the motion of a single actor.
2. Segment the motion into semantic actions and organize the actions in an action level motion graph.

Fig. 1. The overview of the proposed method to simulate competitive interactions.

3. Expand the game tree to predict future states of interaction.
4. Apply the competitive and cooperative objective functions to the min-max framework to evaluate the optimal action.
5. Let the characters perform the optimal action. Then, repeatedly expand the game tree to generate a continuous animation.

Steps 1 and 2 are precomputed, while steps 3 to 5 are performed at runtime.

The two major contributions in this paper are:

- We propose a new method to simulate dense interactions of multiple characters by applying techniques from game theory such as game tree expansion and min-max search.
- We propose a new approach that enables characters compete, while co-operating to satisfy requirements set by the animator.

The rest of the paper is composed as follows: Section 4 describes how the captured motions are preprocessed. Section 5 explains our framework for simulating dense interactions using game theory. Section 6 gives further details on the objective functions we designed. Section 7 describes the system, we use to model the contacts of the virtual characters. Section 8 presents the experimental results. Section 9 discusses the pros and cons of our method, and Section 10 concludes the paper.

4 Data Acquisition and Analysis

Here, we explain the steps used to preprocess the captured motion data and prepare the data structure for real-time character control.

First, we capture long sequences of motions from a single subject. We define the term "motion" as the raw-captured data and the term "action" as a semantic segment of the motion we captured. In the field of fighting, an action can be an attack (such as a left straight, jab, or a right kick), a defence (such as parrying, blocking, or ducking), a transition (such as stepping to the left, stepping forward, or a backward step), reactive motions when being hit or pushed away, or their combinations. Such tagging can be done for other activities as well.

We have developed an automatic motion analyzer that segments and classifies raw motions into actions. This is done by first partitioning a long motion sequence into segments divided by double support phases. If the sum of squares of the acceleration of the joints between two segments is above a predefined threshold at the moment of segmentation, the segments are merged, since the body can be conducting attacks and defences. Finally, we classify the actions according to the trajectories of the joints with large accelerations.

We build a motion graph [9], [10], [11] at the action level rather than at the frame level, as in [15], [16], [33]. This is done by extracting the starting poses and the ending poses of the actions and grouping similar poses together. Let us call this data structure the action level motion graph (see Fig. 2) in which the edges represent actions and the nodes represent postures. Planning based on such a graph is similar to the way a human does, as they also use basic action groups such as attacks and defences, as fundamental entities during fights.

5 Multiobjective Character Control

In our previous work, we simulated characters intelligently competing with their opponents [1]. In this method, whenever a character needs to select an action, we expand a game tree to compute the possible future outcomes and apply min-max search to select the optimal action. However, min-max search is surprisingly inefficient at encouraging collaboration between the competing characters such as following a predefined path. A system that considers this problem would need to define an objective function that awards the characters when they walk along the path. Unfortunately, under the min-max framework, the characters consider the benefits of their opponents to be their own penalties, and thus, prevent each other from following the path. As a result, the deeper the tree is expanded, the smarter they are at blocking one another, and the worse they are at following the path.

Instead of tweaking the objective functions, we propose a new scheme to embed them into the min-max framework. In this section, we explain our control method, which enables the characters to compete with one another locally, while

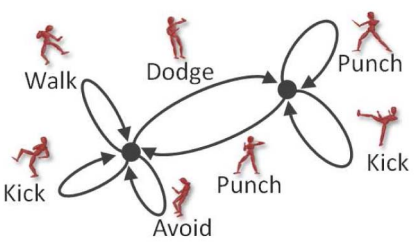


Fig. 2. An action level motion graph generated from the boxing motion.

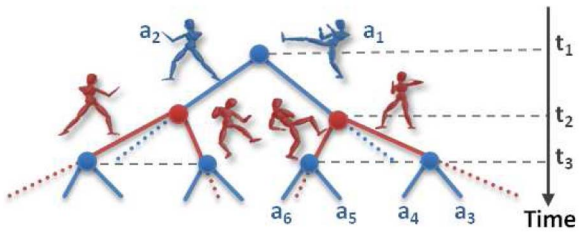


Fig. 3. An expanded game tree showing a fight between two characters. The distance along the vertical axis represents time. The nodes represent the states of the fight after either of the characters has selected a new action and the edges represent the choices of actions. The dotted lines indicate the continuation of the current actions, while the opponent selects its own.

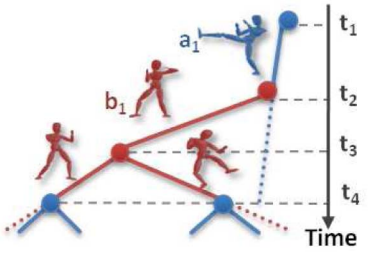


Fig. 4. The expansion of the game tree does not always alternate. Since the action selected by the blue character is long, the red character expands two levels of tree before the blue character further expands.

still cooperating with the others to satisfy the requirements of the animator. We first explain the game tree expansion that we have used for simulating competitive interactions in [1] (see Section 5.1). Next, we explain how we enhanced this approach to separately evaluate the competitiveness and cooperativeness of actions and integrate them (see Section 5.2). Finally, we explain how we can prune nonplausible choices of actions to increase the reality of the scene and reduce the computational cost (see Section 5.3).

5.1 Game Tree Expansion

We adopt methods used for artificial intelligence players in strategy games such as chess to control our virtual characters, since such methods can model the decision making process of a real human. Intelligent players consider the long-term benefit, rather than the immediate one, when making a choice. For example, in chess, a movement that shows the greatest effect in one ply, such as taking a valuable piece like a castle or a bishop, is not necessarily the best choice for achieving a win. By expanding the game tree and evaluating the static position after a few plies, a choice can be made that benefits the player in long run. Here, we apply a similar approach to evaluate the long term benefit of performing each action.

The major difference between character interaction and chess is that the choices made by the characters are performed in a continuous time domain. To apply the tree expansion method, we need to customize our game tree such that discrete planning can be performed with continuous actions of different durations. Every node in our game tree represents the state of interaction between two characters when either of them is about to select a new action. The edges from the node represent the possible choices of actions in such a state. Starting at the root node considering the character, that is, about to select an action, we add edges to the node based on the performable actions. Then, we evaluate future states of interaction for each of the edges and continue to expand the subtrees. Notice that the two characters perform their selected actions concurrently and whenever any of them finish their actions, a subtree is expanded. Fig. 3 shows an example of an expanded game tree, where the vertical axis represents time. The blue character starts the game tree expansion with two choices of actions, a_1 and a_2, at time t_1. Based on the choice of the blue character the red character has choices of actions to counteract at t_2. Notice that the action selected by the blue character is still going on when the red character starts a new action, as indicated by the blue dotted lines. When action a_1 ends at t_3, another level of nodes will be expanded by the blue character according to the current actions of the red character, as indicated by a_3 to a_6.

Since the actions in our game tree have different durations, the order of expansion does not necessarily alternate. If a character selects a long action, its opponent may perform several actions before another. In Fig. 4, the blue character selects a long action a_1 at t_1. When the red character expands a short motion b_1 at t_2, it can expand another node at t_3 as a_1 is still going on. Finally, when the a_1 ends at t_4, the blue character expands the tree again.

In some situations, a character may be forced to stop midway through the current action and perform another action. For instance, in fighting, when a character is hit, it will be either knocked down immediately, or just lose its balance and walk a few steps backward to recover. These response motions will be decided based on the current state of the body and the impulse added to the body. In such cases, we terminate the corresponding edge, insert a new node that indicates the current action being forced to stop, and insert another edge that corresponds to the response motion starting at that node. In Fig. 5, the blue character selects a_1 and the red character selects b_1. It turns out that the red character will be hit by the blue character at t_1. The red character is forced to discard the latter part of b_1 shown by the dotted line and perform a falling back action b_2.

5.2 Evaluating Competitiveness and Co-Operativeness

In this section, we explain how we evaluate the competitiveness and the cooperativeness of the series of actions performed by the characters and select an optimal action, which combines both perspectives. The major improvement

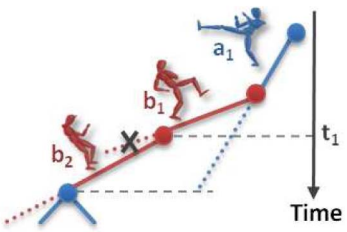


Fig. 5. The character is forced to stop the current action to perform a reactive action when being hit. Since the reactive motion is determined by the system rather than selected by the character, there is only one outgoing edge.

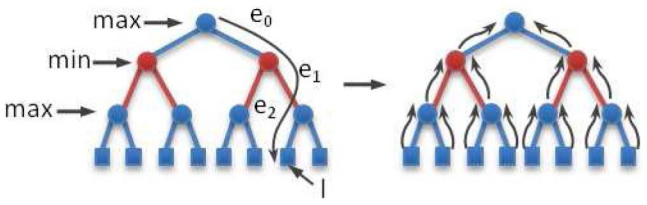


Fig. 6. (Left) The scores of the leaf nodes (squares in the figure) are evaluated from the root node to the leaf nodes with (1). (Right) Min-max is conducted by recursively applying (3) and (2) from leaf nodes to root node.

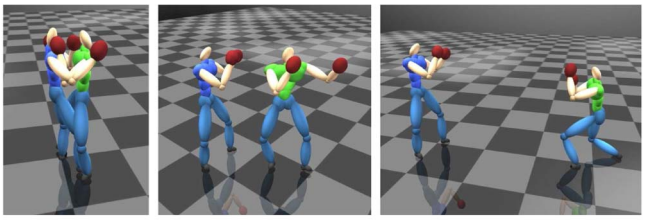


Fig. 7. Examples of actions launched by the green character that have to be pruned: (Left) Penetrating the opponent. (Middle) Turning its back to the opponent while fighting. (Right) Defending while opponent is far away.

of our method is that we define two separate functions to calculate the score of competitiveness and cooperativeness. When calculating the long-term benefits of launching an action, the competitive function evaluates the nodes in the same way as the ordinary min-max framework, such that the character selects a node that maximizes its own benefit while minimizing the opponent's benefits. On the contrary, the cooperative function evaluates the nodes as in ordinary dynamic programming, such that the benefits of both characters are simply accumulated over time.

First, we need to compute the scores of competitiveness and cooperativeness of the leaf nodes of the game tree. For every edge e, we define two functions to evaluate the competitiveness ($F^{comp}(e)$) and the cooperativeness ($F^{coop}(e)$). The details of the objective functions will be explained later in Section 6. Without loss of generality, suppose character A is competing with character B and is expanding the game tree to select an action. The node expanded by A is called a max node, as A tries to maximize its score, and that by B is called a min node, as it tries to minimize A's score. The scores of competitiveness and cooperativeness of a leaf node l in the tree are defined as:

$$\begin{aligned} S^{comp}(l) &= \sum_{e_i \in \mathbf{e}^{\mathbf{l}}_{\mathbf{max}}} F^{comp}(e_i) - \sum_{e_j \in \mathbf{e}^{\mathbf{l}}_{\mathbf{min}}} F^{comp}(e_j), \\ S^{coop}(l) &= \sum_{e_i \in \mathbf{e}^{\mathbf{l}}_{\mathbf{max}}} F^{coop}(e_i) + \sum_{e_j \in \mathbf{e}^{\mathbf{l}}_{\mathbf{min}}} F^{coop}(e_j), \end{aligned} \tag{1}$$

where $\mathbf{e}^{\mathbf{l}}_{\mathbf{min}}$ represents the set of edges from the max nodes and $\mathbf{e}^{\mathbf{l}}_{\mathbf{max}}$ represents those from the min nodes during the path from the root toward leaf node l. Fig. 6 (Left) gives an example of the leaf node evaluation. In the figure, for the leaf node l, its score of competitiveness and cooperativeness can be computed by $S^{comp}(l) = F^{comp}(e_0) + F^{comp}(e_2) - F^{comp}(e_1)$ and $S^{coop}(l) = F^{coop}(e_0) + F^{coop}(e_2) + F^{coop}(e_1)$, respectively.

The next step is to propagate the scores of competitiveness and cooperativeness from the leaf nodes toward the internal nodes and finally to the root node to select the action at the root. Both the competitive and cooperative scores of the internal nodes are computed based on the min-max rule recursively from the leaf nodes toward the root node, as shown in Fig. 6 (Right). We compute the scores of competitiveness/cooperativeness of an arbitrary internal node n, which has a set of children nodes represented by n_i. We select the best child n_{best} among n_i, and simply copy the scores:

$$\begin{aligned} S^{comp}(n) &= S^{comp}(n_{best}), \\ S^{coop}(n) &= S^{coop}(n_{best}), \end{aligned} \tag{2}$$

where n_{best} is selected among n_i by

$$n_{best} = \begin{cases} \text{argmax}_{n_i}(S^{comp}(n_i) + S^{coop}(n_i)) & n_i \text{ is max node} \\ \text{argmin}_{n_i}(S^{comp}(n_i) - S^{coop}(n_i)) & n_i \text{ is min node.} \end{cases} \tag{3}$$

Equations 3 and 2 are applied to evaluate all internal nodes recursively from the leaf nodes toward the root node. The optimal action to be performed is finally selected by applying (3) at the root node. It is to be noted that the competitive and cooperative functions are integrated differently at min and max nodes—by adding $S^{coop}(n_i)$ at the max nodes and subtracting $S^{coop}(n_i)$ at the min nodes, both characters will tend to select actions that results in large absolute values of $S^{coop}(n_i)$. As a result, the animator can effectively control the characters through the cooperative function while making the characters seriously compete with each other through the competitive function.

5.3 Pruning Nonplausible Choices

In order to reduce the computational cost and avoid nonplausible interactions, we prune the bad choices of actions when expanding each node in the game tree. Although, there are a huge number of choices for actions to launch, many of them result in illogical, meaningless behaviors (see Fig. 7). We evaluate the actions based on the following criteria and those unsatisfied are excluded from consideration.

- **Body penetration.** Penetration is one of the most significant artifacts in computer animation. Actions that cause one character to brutally overlap with any others are considered invalid. If none of the actions can avoid such penetrations, we try to select those that are penetration free at the last frame, so that the next interaction does not start with penetrations.
- **Facing angles.** Although, the virtual characters have full knowledge of their opponents and do not require vision, it looks strange for them to perform actions without looking at the opponents. Therefore, we require characters to face their opponents in the last frame of an action.
- **Out of range.** It is logically meaningless to launch actions such as punches and dodges if the opponent is very far away. However, the game tree approach does not consider logical meaning and the characters may perform such actions for other purposes such as waiting for time to elapse before more relevant motions can take place. Since such actions look unnatural, we prohibit characters from attacking or

defending when the opponent is farther away than 1 or 3 meters, respectively.

- **Illogical combinations of actions.** When being attacked, only defensive actions make logical sense, but such actions may not be selected if they are ineffective due to timing or position. However, real humans prefer to defend themselves even if it is in vain. In our system, we only allow the character to defense when the opponent is attacking.

The criteria listed above are in descending order of importance. If none of the actions can satisfy all criteria, the actions that satisfy those of higher importance will be selected.

By pruning the actions as explained, we can increase the reality of the interactions and greatly reduce the computational cost of expanding the game tree. Empirically, we can prune at least half of the available choices using these pruning policies. This would reduce the computational cost approximately by $O(\frac{1}{2}m^n)$, where m is the number of available actions and n is the depth of the game tree. The readers are referred to Section 7 for the actual performance during experiments.

6 Objective Functions

In this section, we explain the details of the objective functions used in (1) to evaluate the competitiveness and cooperativeness of the interactions. The competitive function evaluates the effectiveness of an action at competing with the opponent. The cooperative function evaluates how much the character is following the instructions given by the animator. By combining the two functions, controllable characters conducting realistic competitions can be realized.

6.1 Competitive Function

Here, we explain the details of the competitive function, which evaluates how well each character is competing with the other characters during close interactions. Through observations of various competitive interactions including fighting, chasing, and sports, we find that the objective function of the characters can be well represented by three terms. These are the **movement term** that encourages the characters to move toward the opponents and orient itself for interactions, the **scoring term** that evaluates the scoring criteria of the game and encourages the characters to compete, and the **action combination term** that evaluates the suitability of the action with respect to the actions done by the other characters.

The **movement term** guides the characters toward the target opponent by evaluating the distance and facing angle

$$f^{mov1} = w_\theta(\theta - \theta_d)^2 + w_r(r - r_d)^2, \tag{4}$$

where θ, r are the relative orientation and distance from the opponent, respectively, θ_d, r_d are their preferred values, and w_θ, w_r are the weight constants for each term. θ is computed by projecting the head-facing direction vector onto the ground. In our system, we always set $\theta_d = 0$ so that the character tries to face the opponent. r_d depends on the type of interaction and the movement style. For example, in boxing, an infighter prefers to keep short distance between themselves and their opponents. In that case, r_d is set small such that higher scores are given to actions that bring the character closer to its enemy. On the contrary, for an outboxer or a passive fighter, who prefers to escape from their opponent, high scores are given to actions that increase the distance between them.

The **scoring term** encourages the character to interact with the opponent following the rule of the game. It evaluates how effective the action is at competing with the opponent. In general, it is defined as the weighted sum of the damage the character gives to and receives from the opponent

$$f^{score} = w_D^+ D^+ - w_D^- D^-, \tag{5}$$

where D^+ is the damage that the character gives to the opponent, D^- is the damage received, and w_D^+, w_D^- are positive weight constants for each term. The weight constants depend on the competing style of the character. For boxing, if the fighter is an outboxer, that is, less aggressive, w_D^+ is set small and w_D^- is set large. If the fighter is running out of time and is losing the fight, it has to fight more aggressively regardless of the risk of being hit. In that case, w_D^+ is increased and w_D^- is decreased. In our fighting examples, the damage is set proportional to the velocity of the attacking part and the vulnerability of the part being hit

$$D = \frac{|\mathbf{v_{attack}}|}{v'} \times w_{being\_attack}, \tag{6}$$

where $|\mathbf{v_{attack}}|$ is the norm of the velocity of the attacking segment at the moment it lands onto the opponent, v' is a constant value set to 100 for normalizing the velocity, w_{being_attack} is a weight indicating the vulnerability of the body part, that is, being attacked. In our system, w_{being_attack} is set to 1.0 for the torso, 2.0 for the head, and 0.0 for the limbs. In the sword fighting experiments, the sword is considered to be the attacking segment.

The **action combination term** evaluates the suitability of a performed action when the opponent is conducting a specific action. The previous two terms explained above are useful in evaluating the numerical performances of actions when competing with opponents, but they fall short in representing the implicit factors that affect the action selection process. These implicit factors such as the appropriate style of defense for an attack are difficult to evaluate numerically and require expert knowledge. We manually set up an **action combination table** that helps to determine the optimal actions to perform based on the action taken by the opponent. Each record in the table contains three entities: the character's action, the opponent's action, and a suitability value that describes how effective the character's action is with respect to the opponent's action. A positive value encourages the character to perform such an action when the opponent's action corresponds to the one in the record, while a negative value discourages such an action

$$f^{comb} = \begin{cases} w_S S, & \text{if the action pair exists in the table,} \\ 0, & \text{otherwise,} \end{cases} \tag{7}$$

where w_S is a weight, S is the suitability value recorded in the table. In our system, this term is only used to evaluate the defensive power of each action with respect to each attack by the opponent, which is difficult to evaluate numerically from the motion data. It is known in boxing that sway back motions are effective for avoiding upper

cuts and hooks and head slips are good for avoiding straight punches. There are various factors such as the direction the punch is approaching from, and whether the defender can see the attacker all through the motion, that support these basic techniques. However, it is difficult to numerically evaluate all such factors only from the motion data. By using the action combination table, we can take into account the knowledge of experts such as how effective every defensive action is with respect to various attacks.

The competitive function is composed by summing the three terms:

$$F^{comp} = f^{mov1} + f^{score} + f^{comb}. \quad (8)$$

The competitive function is general enough to produce various competitive interactions such as fighting, chasing, and sports. For example, in chasing, we can increase the preferred distance for the character running away and shorten it for the chaser. For sports like basketball, we can design a scoring function that considers the probability of throwing the ball into the basket, so that the character will try to shoot when there is no opponent to their front.

6.2 Co-Operative Function

The cooperative function evaluates how much the characters are co-operating to achieve a common goal. In general, the animator specifies such a goal to design a scene.

Typical commands by an animator for controlling characters may be following a specific path when moving, and launching a specific style of actions at a specific time. Such observation leads us to implement a cooperative function composed of the **movement term** and the **action requirement term**.

The **movement term** guides the characters to the position specified by the animator. It evaluates, how close the characters' global position and orientation are to their desired values at the end of each action

$$f^{mov2} = w_\gamma(\gamma - \gamma_d)^2 + w_p(p - p_d)^2, \quad (9)$$

where γ, p are the global orientation and position of the character in the world co-ordinate system, γ_d, p_d are their desired values, w_γ and w_p are their weights. Empirically, we found that if we wish a character to simply follow a predefined trajectory, it is effective to remove the terms of orientation and define the trajectory as a series of check points, updating the values of p_d whenever a checkpoint is reached.

The **action requirement term** encourages the character to perform actions specified by the animator. It gives a high score if such actions are successfully performed

$$f^{req} = \begin{cases} w_a, & \text{when } A^+\text{performed}, \\ -w_a, & \text{when } A^-\text{performed}, \\ 0, & \text{otherwise}, \end{cases} \quad (10)$$

where w_a is the weight, A^+ is the set of actions to be performed and A^- is the set of actions not to be performed. By dynamically updating A^+ and A^-, the animator can control the flow of the animation. In our system, we use this term to control one character to knock down another. By requesting a character to perform falling down actions, we implicitly require that the character to be knocked down by the opponent, since falling down actions cannot be performed without being hit.

Finally, the two terms are summed to compose the cooperative function

$$F^{coop} = f^{mov2} + f^{req}. \quad (11)$$

7 Physical Interactions between Characters

We adopt Jakobsen's [34] technique that uses particles to simulate the rigid body's dynamics. Each joint is represented by a particle and is activated by external force calculated by a PD controller [35]. Since the location of segments can be constrained in this method, we fix the supporting foot onto the ground to prevent it from sliding. The body segments are modeled with spheres and cylinders to reduce the computational cost of collision detections. We add repulsive forces to the segments when a collision occurs to avoid penetrations.

The effect of stepping back or falling down are synthesized by concatenating the appropriate response motions in the database according to the condition of the impulse and the body postures [36], [37]. The initial motion, when the impulse is added is simulated by rigid body dynamics, then the posture is compared with the initial postures of the response motions in the database. Once an appropriate motion is discovered, the two motions are blended.

8 Experimental Results

We simulated scenes of multiple characters fighting and chasing each other using singly captured motions of shadow boxing, swinging a sword, and running around. In this section, we first explain the capturing sessions and preprocessing steps. Next, we give details of how the competitive interactions of fighting and chasing simulated between the characters. Finally, we explain how the scenes involving multiple characters are controlled by the animator.

8.1 Motion Capture and Preparation Processes

An optical motion capture system was used to capture the motions of a single actor. The frame rate was set to 60 postures per second. We have captured motions of shadow boxing for seven minutes, sword swinging for 15 minutes, and running around for 1.5 minutes. They were automatically segmented by our motion analyzer into 279, 612, and 215 actions, respectively. The analyzer can mistakenly split slow and less energetic actions that involve multiple steps, as we first segment the motions by the footstep pattern, and then, merge them according to the body acceleration. We manually validate and amend the segmentation, which usually takes 20 to 60 minutes depending on the number of the actions.

We implemented an action combination table explained in Section 6 to evaluate the effectiveness of defensive actions with respect to different attacks in the boxing database. To speed up the table creation process, instead of specifying all records manually, we first annotate the target position and direction for attacks. Next, the defences are annotated with the position and direction they are defending from (see Table 1). The system then automatically inserts records into the action combination table by pairing all the attacks with defences. The suitability value for each pair is defined as $S = n_{matched}/n_{defence}$, where $n_{matched}$ is the number of common elements in the attack and the defence

TABLE 1
Example Annotations of Attacks and Defences

Action	Position	Direction
Right Jab	Head, Upper Torso	Front
Upwards Kick	Upper Torso, Lower Torso	Upwards
Duck	Head, Upper Torso	Front, Horizontal
Jump Back	Upper Torso, Lower Torso	Horizontal, Upwards

TABLE 2
Example Records in the Action Combination Table

Character's Action	Opponent's Action	Suitability Value, S
Duck	Right Jab	$3/4 = 0.75$
Duck	Upwards Kick	$1/4 = 0.25$
Jump Back	Right Jab	$1/4 = 0.25$
Jump Back	Upwards Kick	$3/4 = 0.75$

TABLE 3
The Parameter Used in the Competitive Function to Simulate Various Effects

	w_θ	w_r	θ_d	r_d	w_D^+	w_D^-	w_S
General boxer	10^1	10^1	0°	$0.8m$	10^5	10^5	10^2
Infighter	10^1	10^1	0°	$0.5m$	$10^5/10^1$ †	10^5	10^2
Outboxer	10^1	10^1	0°	$2.0m$	$10^1/10^5$ †	10^5	10^2
Boxer (Path)	10^1	10^1	0°	$0.8m$	10^5	10^5	10^2
Sword fighter	10^1	10^1	0°	$1.5m$	10^5	10^5	10^2
Sword fighter (Path)	10^1	10^1	0°	$1.5m$	10^5	10^5	10^2
Chaser	10^1	10^1	0°	$0.1m$	10^5	0	0
Runaway	10^1	10^1	0°	$3.0m$	0	10^5	0
Chaser (Path)	10^1	10^1	0°	$0.1m$	10^5	0	0
Runaway (Path)	10^1	10^1	0°	$3.0m$	0	10^5	0

†*The parameters used for short range and long range attack, respectively.*

columns considering the position and the direction, and $n_{defence}$ is the total number of elements in the defence column (see Table 2). The equation favors defences that are effective to the attack while specific to the areas being attacked. Finally, the animator may amend or add records in the table. The whole process takes around one hour.

Each character model has 6 DOF for the translation and orientation of the root, and 72 DOF for the joint orientation. The parameters of the competitive and cooperative functions for each experiment are shown in Table 3 and Table 4, respectively.

8.2 Competitive Interactions between Two Characters

In order to examine the effectiveness of the min-max search at simulating competitive interactions, various fighting and chasing scenes were created by adjusting the parameters of the objective function and the depth of the game tree.

We first simulated a fighting scene between two characters by using the shadow boxing actions. We can create different styles of fighting by adjusting the objective function—infighting style was simulated by decreasing the preferred distance between the characters and giving higher scores to successful short range attacks such as upper-cuts and hooks (see Fig. 8 (Left)). An outboxing style was simulated by increasing the preferred distance between the characters and giving higher scores to successful long range attacks such as straight punches (see Fig. 8 (Middle)). The intelligence level of the characters can be controlled by altering the depth of the game tree expansion. We simulated

TABLE 4
The Parameter Used in the Co-Operative Function (Unlisted Simulations do not Require the Co-Operative Function)

	w_γ	w_p	w_a
Boxer (Path)	0	10^1	10^6
Sword fighter (Path)	0	10^1	10^6
Chaser (Path)	0	10^1	0
Runaway (Path)	0	10^1	0

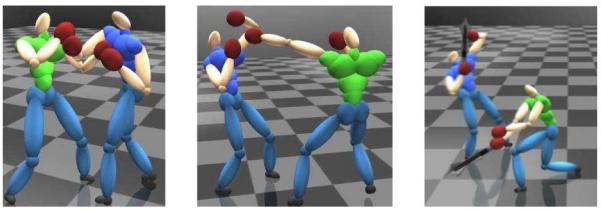


Fig. 8. (Left) Infighters simulated by our system that prefers a shorter distance from the opponent and short range attacks. (Middle) Outboxers simulated by our system that prefer a longer distance from the opponent and long range attacks such as kicks and straight punches. (Right) Two characters sword fighting by expanding the game tree.

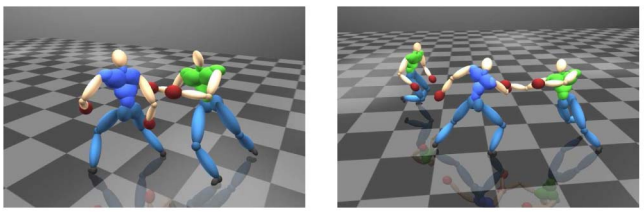


Fig. 9. (Left) The green character chasing and catching the blue character. (Right) Two characters chasing one character.

a less intelligent fighter by setting the intelligence level to two and a smart fighter by setting the level to four. In such a case, the intelligent fighter always wins the match as its decision is based on greater expansion of the game tree. Experiments were also done to examine the interactions of two characters sword fighting (see Fig. 8 (Right)). Again, the strength of the fighters could be controlled by changing the depth of game tree.

Next, we simulated one character chasing another by using the running motion (see Fig. 9 (Left)). The preferred distance between them is set short for the chaser and long for the escaper. Moreover, the scoring function was adjusted such that a high score is given to the chaser when it catches the escaper, and a high penalty is given to the escaper when it is caught. As a result, the chaser tries to approach the escaper, while the escaper tries to get away. When we increase the intelligence level of the chaser to four and lower that of the escaper to two, the chaser can catch the escaper quickly. Further, increasing the intelligence level generates similar results because a difference of two levels is already large enough for the chaser to easily catch the escaper. We can also simulate scenes of two characters collaboratively chasing one character by summing the objective functions of the two chasers (see Fig. 9 (Right)).

8.3 Animator Controlled Competitions

Here, we present examples of the animator controlling the scenes through the cooperative term. We show examples of the characters fighting or chasing one another while

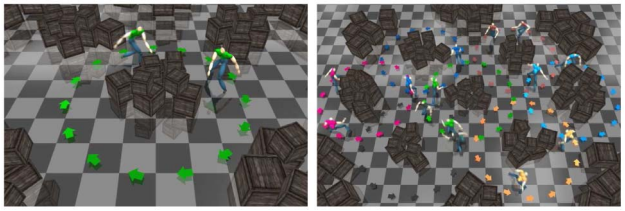


Fig. 10. (Left) One character chasing another character in a circular trajectory. (Right) A large-scale scene in which many characters conducting similar movements can be synthesized. The check points are adjusted so that the characters do not collide into one another.

following paths given by the animator. Each of the paths is modeled as a series of checkpoints to be reached by the characters. Each checkpoint is defined by a 2D position on the floor, an optional timing value, and optional requirements on the actions to be conducted when passing by. When the characters reach a checkpoint, they perform any required actions and wait until the indicated timing is passed. Then, the system specifies the next checkpoint. The movement term in the cooperative function in (12) awards an action that guides the characters to the next check point, while the action requirement term encourages the characters to conduct the specified actions. Since the movements of the characters are spatio-temporally constrained, we can design a scene containing a lot of characters with little risk of unexpected collisions. Using this method, we can simulate a large number of characters following instructions.

First, using the running around motion database, a scene in which one character chases another in a circular trajectory is created (see Fig. 10 (Left)). A number of checkpoints are prepared on the floor so that each of the characters passes them with the correct timing. We can create a scene, where many pairs perform such chasing while avoiding colliding with one another by defining multiple trajectories (see Fig. 10 (Right)).

Next, we show a scene with a large number of characters fighting in pairs along spiral trajectories, creating a spreading-out formation as indicated by the red area in Fig. 11a. The characters wearing red gloves are designed to knock down their opponents when the pairs reach the final check point, as shown in the right-most image of Fig. 11a. This is done by setting up the action requirement term in the cooperative function. Another example using the sword fighting motion was also created as shown in Fig. 11b. Although, the trajectories overlap spatially, they do not overlap in the temporal domain, and hence, the pairs of characters do not run into one another.

Finally, we show examples, where scenes are created along story lines given by the animator. In action films, often there are scenes, where the main character faces a situation that he/she is in danger but finally survives at the end due to some unexpected events. Using the boxing data, we created a scene in which the main character fights with a strong enemy and survives because the enemy is crashed into by a car (see Fig. 11c). By adjusting the depth of the tree expanded, we control the strength of the characters such that the main character keeps on getting hit. Both characters collaboratively walk along a given path, that is, composed of a sequence of checkpoints. We locate the last checkpoint at a street with a lot of passing cars, and use simple bounding boxes to detect collisions between the characters and the cars. Based on the storyline, the character colliding with the cars is forced to perform a falling down action. A similar example, where the main character is fighting with a strong enemy was created using the sword fighting data. The main character retreats into an alley, where a hero suddenly jumps in to knock down the enemy (see Fig. 11d). We design the trajectories for each character to follow and use the action requirement term to request the enemy be knocked down by the hero at the last checkpoint. The advantage of this approach is that the animator only needs to specify high-level instructions such as the path the characters pass through and the overall timing the events. The system will then plan the individual actions launched by the characters to complete such requirements.

8.4 Computational Costs

The computation time depends on the size of the action set, the connectivity of the motion graph, and the complexity of the objective function. In general, using a computer with a Pentium 4 Dual core (3 GHz) and 2 GB of RAM, it takes five minutes to create a video of length 30 seconds when the game tree is expanded for three levels for each character. However, the computational cost increases exponentially with the number of levels, which are expanded. For example, creating the same video with five level game trees will take hours to finish. Nevertheless, from our experience, long horizon planning usually does not create interesting animations. One reason is that they become too "smart" and careful, and therefore, avoid performing anything that leads to disadvantages. As a result, the characters become less active, which is not what the animator wants to see. We have examined from our experimental results that a depth level of up to four is effective to produce intelligent and interesting movements of the characters. In addition to that, the tree expansion process can easily be broken down into multiple parallel processes. More specifically, we can implement a multithread system in which each thread expands a subtree of the whole game tree. As multicore processors become cheap and popular, the performance of our method can be greatly enhanced.

The pruning rules mentioned in Section 5.3 were shown to be effective in our experiments. We simulated around 10 minutes of interactions with each of our motion databases using three level game trees. The numbers of actions before and after pruning are recorded and the average is calculated (see Table 5). Pruning is less effective on the sword swinging database, mostly because the preferred distance for sword fighting characters is long, and hence, the body penetration constraint becomes less effective.

9 Discussions

Using our method, animators can simulate the close competitive interactions of multiple characters based on individually captured motions and specify details such as the trajectories of the characters during the animation.

The action combination table mentioned in Section 6 provides animators with an interface to embed manually designed, plausible interactions into the scene. Stylized and artistic combinations of specific attacks and defences may rarely appear if the interactions are only evaluated by the objective function. Using the action combination table, the animator can make such interactions appear with minimal adjustment of the objective function. The action combination

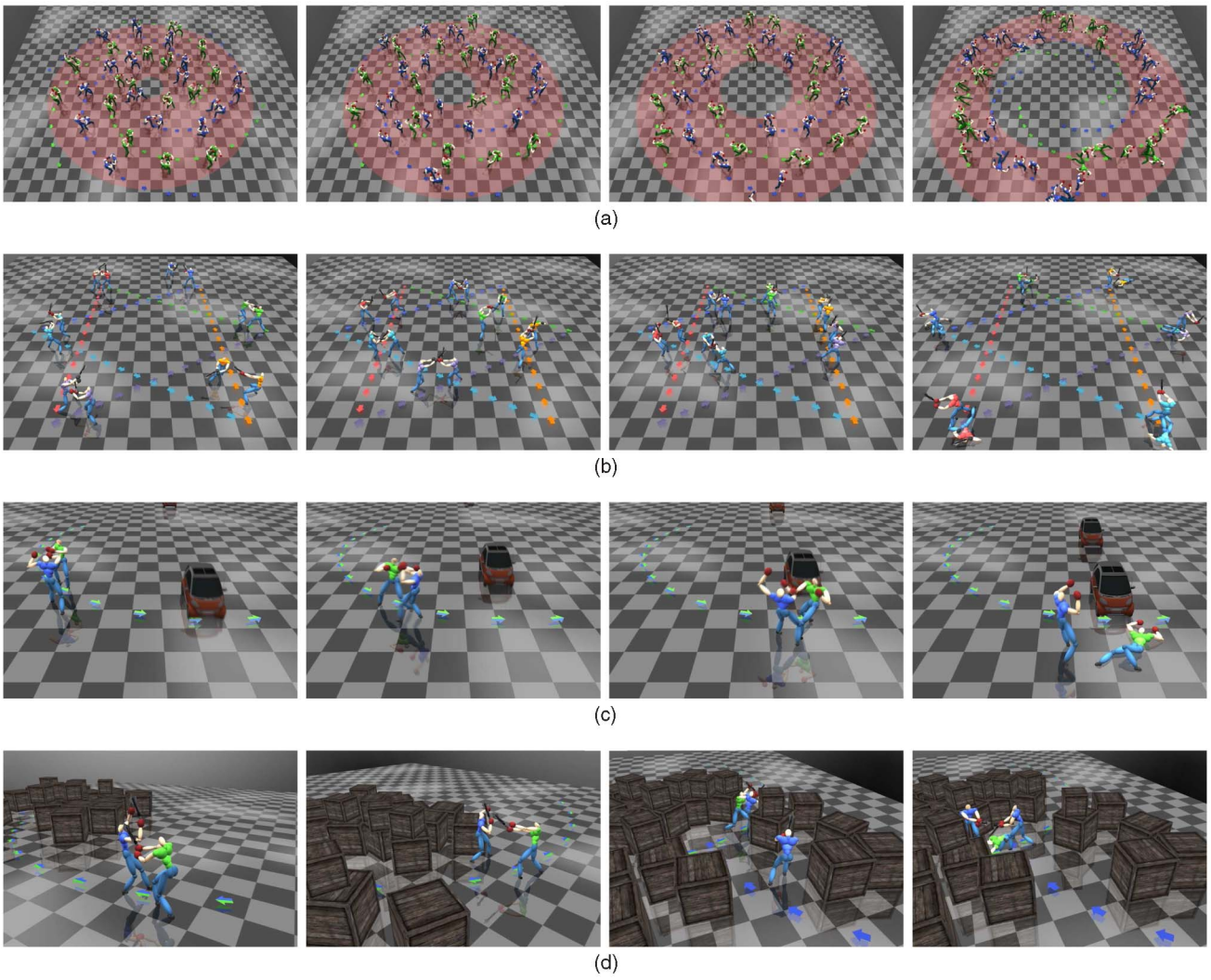


Fig. 11. (a) A crowd of characters fighting with opponents while following the formation specified by the animator, which is visualized by the red area. (b) Characters fighting with weapons in pairs and following the trajectories. (c), (d) examples following the story line. (c) The strong green enemy first attacks the blue character but gets hit by a car at the end. The two characters cooperatively follow the path while fighting. (d) The two characters sword fighting. The blue character retreats into an alley, where an ally joins in to knock down the green enemy.

term (f^{comb}) is currently defined as an element of the competitive function (F^{comp}). This is because the term is used to model the attack-defence relationship and we wish the smarter character to perform better defensive motions. However, if the animator wants the two characters to cooperatively perform stylized interactions, the term has to be moved to the cooperative function (F^{coop}); otherwise, the two characters will prevent their opponents from acting as indicated in the action combination table.

Our framework can be expanded to simulate interactions for multiple characters such as a basketball game. The simplest approach is to expand a game tree for all characters in the scene and evaluate the combined benefits from all allies and opponents. However, since expanding such a huge game tree is computational costly, it would be better to apply a multilevel control strategy—designing a representation based on the formation of the whole team, and expanding the game tree at that level. Min-max search can be used to select the best formation taking into account the counter formation by the opponent team. Individual actions by the characters can be simulated by simple path planning using the motion graph.

In our model, we assume each character has perfect knowledge about its opponent in terms of the opponent's strategy and the action evaluation functions. Due to such knowledge, the min-max search always gives the optimal result. However, if the knowledge of the opponent is incomplete and inaccurate, an opponent model search that takes into account the mistakes made by the opponents may perform better [38]. To implement opponent models in our system, one simple approach is to observe the actual moves made by the opponent. During the tree expansion process,

TABLE 5
Average Number of Actions before and after Pruning When Expanding One Node

	# of Actions Before Pruning	# of Actions After Pruning	Percentage Pruned
Boxing	120.0	35.5	70.4%
Sword Swinging	94.3	68.2	27.7%
Running-Around	95.8	40.3	57.9%

we determine the probability of the opponent launching each action, based on the observed history, and evaluate the opponent nodes using the expectation values. This would be an interesting direction for future research.

Our system is a simplification of nonzero-sum games in which the gain of a player may not necessary be the loss of the opponent [39]. We simplify the game by assuming that the nonzero-sum part of the system only occurs within the cooperative function and handle it differently such that the function benefits both characters. As a future work, it would be interesting to investigate the possibility of applying our method to more complex nonzero-sum games in which the gain depends not only on the loss of the opponent, but also on other objective factors such as the skills of the player.

When a system reaches an equilibrium, all players in the game are unable to obtain better rewards by changing their control strategies [39]. In our research, we try to avoid the equilibrium because when such a condition is reached, the system is very likely to stay in the same state for a long duration, which leads to monotonic animation. One example of the equilibrium in a fight scene is that both characters stuck defending [25], which is not pleasant from the animator's point of view. Fortunately, since each action has particular values for it's attacking/defending position and duration, the state space is highly irregular. This reduces the chance that the characters stay at an equilibrium point. Furthermore, we give a penalty to actions that have been recently used such that the equilibrium point shifts in the state space at every time step.

Our idea of multiobjective control is general enough to be used with other precomputation-based methods such as reinforcement learning [2] and state machine-based dynamic programming [5]. One general issue with such methods is that they usually suffer from the high-dimensional state space, and a naïve solution results in rough quantization [2] or reduced number of action samples [5], which can cause lower precision at future predictions and less controllability of the characters. Nonetheless, it has a great potential if it can be solved, for example, by representing the value function by a sum of basis functions [3], as we will be able to produce multiobjective character animation in real time.

The method can also be combined with interaction patches [4], which precomputes the close interactions between two characters based on the user specified pattern and their spatiotemporal concatenation. Interaction patches are effective for synthesizing stylized interactions with minimal number of sample motions. However, the disadvantages is that it is not very suitable for long persistent interactions between two characters, as similar combinations of actions are likely to be repeated over time, and the connectivity of the interaction patches is low if the two characters continuously interact. On the other hand, the proposed method is effective at making the characters follow control signals from an animator. Animators can effectively combine both for synthesizing an animation along a given story line. For example, the animator may precompute the final scene, where the enemies are finished off by interaction patches and guide the fighting characters there by the proposed multiobjective approach. Fig. 12 shows a comparison between the proposed method and interaction patches for producing persistent long interactions between two characters. The latter performs suboptimally because only a limited choice of patches are available for temporally concatenating, which results in poor quality following of the instructions given by the animator.

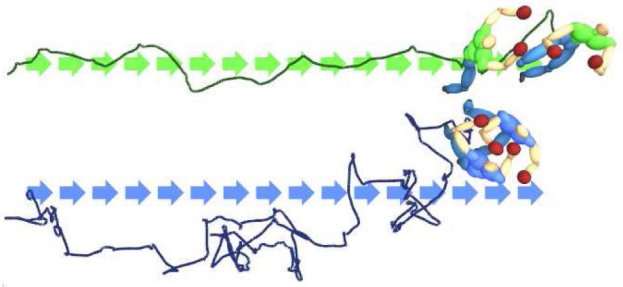


Fig. 12. The characters controlled by the proposed method can follow the path accurately (top) while those controlled by interaction patches cannot follow the path well due to the lack of samples (bottom). We used the same set of actions in these animations and the bottom animation is produced from 279 interaction patches.

We observed that in movies, the environment and the story line of a specific scene are usually designed and used only once throughout the whole movie. It would be ineffective to run a precomputation system for such a purpose. Instead, animators usually wish to interactively adjust parameters and see the results immediately. Our method does not require any training or precomputation stage, and any changes in the system are reflected immediately. The interaction graph [5] and other reinforcement learning systems require a long training stage that could take hours. Once the parameters and the objective functions are changed or if the story line is updated, this precomputation needs to be repeated from the beginning.

There are some drawbacks to our system. First, the criteria for pruning the subtree when expanding the game tree must be determined by an expert who knows the nature of the interactions well. Such criteria can change according to the interaction—however, the criteria listed in Section 5.3 can be applied well to various competitive interactions. Second, we cannot currently handle persistent, continuous contacts, which appear in martial arts when squeezing the joints or locking the torso or limbs. One way to handle such interactions is to combine the proposed approach with the topology co-ordinates [30]. Such a combination would be useful for 3D computer games that involve motions of continuous contact, such as wrestling games.

10 Conclusions

In this paper, we have presented a method to simulate competitive scenes in which multiple characters are closely interacting with one another. We first expanded the game tree and applied min-max search to determine the actions of each character. Then, we have shown that various styles of fighting and chasing can be simulated by changing parameters such as the depth of the game tree and the evaluation function. We also embedded the cooperative functions into the min-max framework so that the characters cooperatively follow instructions from the animator while competing with each other. As a result, we can create intelligent characters that compete well while being easily controlled.

ACKNOWLEDGMENTS

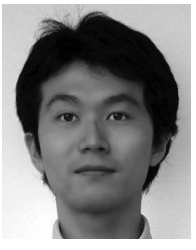
The authors would like to thank the anonymous reviewers for their constructive comments, Adam Barnett for his valuable suggestions, and Dr. Howard Leung for helping them to capture the motion data. This work was partially supported by grants from EPSRC (EP/H012338/1).

REFERENCES

[1] H.P.H. Shum, T. Komura, and S. Yamazaki, "Simulating Competitive Interactions Using Singly Captured Motions," *Proc. ACM Symp. Virtual Reality Software and Technology,* pp. 65-72, 2007.
[2] J. Lee and K.H. Lee, "Precomputing Avatar Behavior from Human Motion Data," *Proc. ACM SIGGRAPH '04,* pp. 79-87, 2004.
[3] A. Treuille, Y. Lee, and Z. Popovic', "Near-Optimal Character Animation with Continuous Control," *ACM Trans. Graphics,* vol. 26, no. 3, pp. 7:1-7:7, 2007.
[4] H.P.H. Shum, T. Komura, M. Shiraishi, and S. Yamazaki, "Interaction Patches for Multi-Character Animation," *ACM Trans. Graphics,* vol. 27, no. 5, 2008.
[5] H.P.H. Shum, T. Komura, and S. Yamazaki, "Simulating Interactions of Avatars in High Dimensional State Space," *Proc. ACM SIGGRAPH '08,* pp. 131-138, 2008.
[6] M. Gleicher, "Retargetting Motion to New Characters," *Proc. ACM SIGGRAPH '98,* pp. 33-42, 1998.
[7] J. Lee and S.Y. Shin, "A Hierarchical Approach to Interactive Motion Editing for Human-Like Figures," *Proc. ACM SIGGRAPH '99,* pp. 39-48, 1999.
[8] Y. Abe, C.K. Liu, and Z. Popović, "Momentum-Based Parameterization of Dynamic Character Motion," *Proc. ACM SIGGRAPH '04,* pp. 173-182, 2004.
[9] O. Arikan and D. Forsyth, "Motion Generation from Examples," *ACM Trans. Graphics,* vol. 21, no. 3, pp. 483-490, 2002.
[10] J. Lee, J. Chai, P.S.A. Reitsma, J.K. Hodgins, and N.S. Pollard, "Interactive Control of Avatars Animated with Human Motion Data," *ACM Trans. Graphics,* vol. 21, no. 3, pp. 491-500, 2002.
[11] L. Kovar, M. Gleicher, and F. Pighin, "Motion Graphs," *ACM Trans. Graphics,* vol. 21, no. 3, pp. 473-482, 2002.
[12] L. Kovar and M. Gleicher, "Automated Extraction and Parameterization of Motions in Large Data Sets," *ACM Trans. Graphics,* vol. 23, no. 3, pp. 559-568, 2004.
[13] T. Mukai and S. Kuriyama, "Geostatistical Motion Interpolation," *ACM Trans. Graphics,* vol. 24, no. 3, pp. 1062-1070, 2005.
[14] A. Safonova and J.K. Hodgins, "Construction and Optimal Search of Interpolated Motion Graphs," *ACM Trans. Graphics,* vol. 26, no. 3, p. 106, 2007.
[15] M. Lau and J.J. Kuffner, "Behavior Planning for Character Animation," *Proc. the ACM SIGGRAPH' 05,* pp. 271-280, 2005.
[16] T. Kwon and S.Y. Shin, "Motion Modeling for On-Line Locomotion Synthesis," *Proc. ACM SIGGRAPH '05,* pp. 29-38, 2005.
[17] R. Heck and M. Gleicher, "Parametric Motion Graphs," *Proc. Symp. Interactive 3D Graphics and Games,* 2007.
[18] H.J. Shin and H.S. Oh, "Fat Graphs: Constructing an Interactive Character with Continuous Controls," *Proc. ACM SIGGRAPH '06,* pp. 291-298, 2006.
[19] S.I. Park, T. Kwon, H.J. Shin, and S.Y. Shin, "Analysis and Synthesis of Interactive Two-Character Motions," Technical Note, CS/TR-2004-194, KAIST, 2004.
[20] T. Kwon, Y.-S. Cho, S.I. Park, and S.Y. Shin, "Two-Character Motion Analysis and Synthesis," *IEEE Trans. Visualization and Computer Graphics,* vol. 14, no. 3, pp. 707-720, May 2008.
[21] C.K. Liu, A. Hertzmann, and Z. Popović, "Composition of Complex Optimal Multi-Character Motions," *Proc. ACM SIGGRAPH '06,* pp. 215-222, 2006.
[22] R.S. Sutton and A.G. Barto, *Reinforcement Learning: An Introduction.* MIT Press, 1998.
[23] L. Ikemoto, O. Arikan, and D. Forsyth, "Learning to Move Autonomously in a Hostile World," Technical Report UCB/CSD-5-1395, Univ. California, 2005.
[24] W.-Y. Lo and M. Zwicker, "Real-Time Planning for Parameterized Human Motion," *Proc. ACM SIGGRAPH '08,* 2008.
[25] T. Graepel, R. Herbrich, and J. Gold, "Learning to Fight," *Proc. Int'l Conf. Computer Games: Artificial Intelligence Design and Education (CGAIDE '04),* pp. 193-200, 2004.
[26] K. Wampler, E. Andersen, E. Herbst, Y. Lee, and Z. Popovic, "Character Animation in Two-Player Adversarial Games," *ACM Trans. Graphics,* vol. 29, no. 3, pp. 1-13, 2010.
[27] T. Kwon, K.H. Lee, J. Lee, and S. Takahashi, "Group Motion Editing," *Proc. ACM SIGGRAPH '08,* pp. 1-8, 2008.
[28] Y.-C. Lai, S. Chenney, and S. Fan, "Group Motion Graphs," *Proc. ACM SIGGRAPH '05,* 2005.
[29] S. Takahashi, K. Yoshida, T. Kwon, K.H. Lee, J. Lee, and S.Y. Shin, "Spectral-Based Group Formation Control," *Computer Graphics Forum,* vol. 28, no. 2, pp. 639-648, 2009.
[30] E.S.L. Ho and T. Komura, "Character Motion Synthesis by Topology Coordinates," *Computer Graphics Forum,* vol. 28, no. 2, pp. 299-308, 2009.
[31] E.S.L. Ho, T. Komura, and C.-L. Tai, "Spatial Relationship Preserving Character Motion Adaptation," *ACM Trans. Graphics,* vol. 29, no. 4, pp. 1-8, 2010.
[32] M. Kim, K. Hyun, J. Kim, and J. Lee, "Synchronized Multi-Character Motion Editing," *ACM Trans. Graphics,* vol. 28, no. 3, pp. 1-9, 2009.
[33] M. Gleicher, H.J. Shin, L. Kovar, and A. Jepsen, "Snap-Together Motion: Assembling Run-Time Animations," *ACM Trans. Graphics,* vol. 22, no. 3, pp. 181-188, 2003.
[34] T. Jakobsen, "Advanced Character Physics," *Proc. Game Developers Conf.,* pp. 383-401, 2001.
[35] V.B. Zordan and J.K. Hodgins, "Motion Capture-Driven Simulations that Hit and React," *Proc. ACM SIGGRAPH '02,* pp. 89-96, 2002.
[36] O. Arikan, D.A. Forsyth, and J.F. O'Brien, "Pushing People Around," *Proc. ACM SIGGRAPH '05,* pp. 59-66, 2005.
[37] V.B. Zordan, A. Majkowska, B. Chiu, and M. Fast, "Dynamic Response for Motion Capture Animation," *ACM Trans. Graphics,* vol. 24, no. 3, pp. 697-701, 2005.
[38] D. Carmel and S. Markovitch, "Learning and Using Opponent Models in Adversary Search," Technical Report CIS9609, Technion, 1996.
[39] J.V. Neumann and O. Morgenstern, *Theory of Games and Economic Behavior.* Princeton Univ. Press, 1944.

Hubert P.H. Shum received the PhD degree (2010) from the School of Informatics in the University of Edinburgh and the MSc (2005) and BEng (2004) degrees from the City University of Hong Kong. He is currently a post-doctoral researcher in RIKEN Japan. His research interests include character animation, artificial intelligence, and robotics.

Taku Komura received the BSc, MSc, and DSc in information science from the University of Tokyo. He is currently a lecturer at School of Informatics, Edinburgh University, United Kingdom. His research interests include topology-based motion synthesis and application of machine learning to character animation.



Shuntaro Yamazaki received the PhD degree in computer science from the University of Tokyo, in 2004. He is a researcher of Digital Human Research Center at the National Institute of Advanced Industrial Science and Technology, Japan. His current research interests include computer vision and graphics in the area of 3D shape reconstruction and image-based rendering, and user interfaces for web-based communication. He is a member of the IEEE.