Preparation Behaviour Synthesis with Reinforcement Learning

Hubert P. H. Shum
Northumbria University
hubert.shum@northumbria.ac.uk

Ludovic Hoyet
Trinity College Dublin
hoyetl@tcd.ie

Edmond S. L. Ho
Hong Kong Baptist University
edmond@comp.hkbu.edu.hk

Taku Komura
University of Edinburgh
tkomura@info.ed.ac.uk

Franck Multon
University Rennes 2
fmulton@irisa.fr

Abstract

When humans perform a series of motions, they prepare for the next motion in advance so as to enhance the response time of their movements. This kind of preparation behaviour results in a natural and smooth transition of the overall movement. In this paper, we propose a new method to synthesize the behaviour using reinforcement learning. To create preparation movements, we propose a customized motion blending algorithm that is governed by a single numerical value, which we called the *level of preparation*. During the offline process, the system learns the optimal way to approach a target, as well as the realistic behaviour to prepare for interaction considering the level of preparation. At run-time, the trained controller indicates the character to move to a target with the appropriate level of preparation, resulting in human-like movements. We synthesized scenes in which the character has to move in a complex environment and interact with objects, such as a character crawling under and jumping over obstacles while walking. The method is useful not only for computer animation, but also for real-time applications such as computer games, in which the characters need to accomplish a series of tasks in a given environment.

Keywords: Preparation Behaviour, Motion Synthesis, Reinforcement Learning, SARSA, Motion Blending

1 Introduction

In an environment where humans have to quickly conduct one motion after another, they adapt their behaviours and prepare for the next motion while approaching the target. This allows smooth transition to the target action, as well as a quicker activation to interact with the target. For example, when a human anticipated that he/she has to crawl under an obstacle while running, it is unlikely that he/she runs straight to the obstacle, stops suddenly and starts crawling. Instead, one usually slows down and lowers the body before reaching the obstacle, in order to prepare for the crawling action.

Traditional kinematic motion synthesis approaches such as motion graph [1, 2] does not alter the movement behaviour based on the potential next motion. As a result, awkward behaviours, although kinematically correct, such as running to an obstacle and stopping absurdly to crawl may be created. A naive solution would be capturing different levels of preparation movements for all combinations of motions, and manually crafting the motion graph such that the character applies the appropriate movements to prepare for the target motion. However, this requires a huge amount of labour works, and the size of the motion database increases expo-

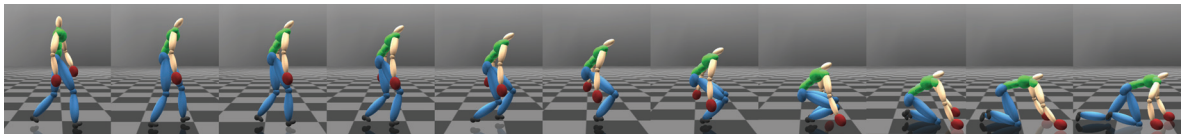

Figure 1: A motion sequence created using our method. The character prepares for the crawling motion by lowering its pelvis and crouches its torso while walking.

nentially with respect to the number of actions, making it cost inefficient.

We observe that when a human approaches a target, the preparation behaviours usually involve (1) a movement to prepare for launching the target motion, and (2) a change of movement speed. To enable efficient control over the synthesis process, we propose a single control parameter called the *level of preparation* for both behaviours. We design a customized motion blending algorithm to generate the overall movement. Given the current and target motions, as well as a corresponding level of preparation, we can create the movement with the same content as the current motion, with a new movement style to prepare for the target motion. As a result, our system can efficiently generate natural preparation behaviour without capturing extra motions.

We further propose a unified controller based on reinforcement learning that controls a character to approach a target with the proper preparation behaviour in real-time. During the training stage, our system learns the optimal motion to approach the target, and more importantly, the level of preparation that should be applied to create realistic preparation behaviour. At runtime, the trained controller anticipates how the preparation behaviour may affect the movement. It controls the character by indicating the optimal motion and the optimal level of preparation. The system can then synthesize the resultant movements with human-like preparation behaviour.

We conducted experiments for various target actions, such as punching, crawling, kicking and jumping. We show that our system can produce realistic preparation behaviour in real-time. The proposed algorithm is computationally efficient, making it suitable for applications such as interactive animation creation and computer games.

The rest of the paper is organized as follow. We first review previous work in Section 2. We then explain how we prepare the captured motion in Section 4. Our algorithm involves two major parts. Section 5 explains how to synthesize a preparation movement, and Section 6 explains how to create a controller to control a character with preparation behaviour. We present experimental results in Section 7, and conclude the paper in Section 8.

2 Related Works

Our idea to adapt motions towards the subsequent motion is related to motion blending, motion-style translation and spacetime motion editing. We first compare our method with such methodologies. Then, we evaluate algorithms involving data-driven approaches, which perform sub-optimally in our problem. Finally, we discuss long horizon techniques to intelligently control the characters by predicting the future.

Motion blending is a traditional technique to synthesize new motions by interpolating existing motion data. Motion warping [3] edits a motion by inserting an intermediate posture as an offset. Boulic et al. [4] blend several motions simultaneously by changing the weights for different degrees of freedom. Ménardais et al. [5] propose a method to compute the appropriate weights for blending different motions based on intuitive input by the user. Shum et al. [6] propose to blend motions considering the angular momentum profile. In these methods, the change of weights has to be hand-tuned during synthesis to create a realistic motion. We need an automatic method to compute the weights in order to control agents intelligently.

Motion style translation extracts movement style based on two motions, which are a plain motion without any style and a style-added motion. A style is extracted using Hidden Markov

Models [7] or linear time invariant models [8]. It is then injected to other plain motions in order to synthesize new style-added motions. A major problem is the need of capturing styled motions. Preparation movements of different degree of preparation are different, and hence using these approaches requires a large amount of captured motions. Moreover, it is difficult to ask human to precisely conduct preparation movement with small changes on the degree of preparation. On the contrary, our method computes the styled motion and minimizes the amount of labour works.

Spacetime optimization [9] is a method for optimizing a motion when the objectives of the motion conversion are known. It has been used to satisfy kinematic constraints [10], maintain kinematics similarity to the original motion while retargeting the character [11] and convert manually designed motions to physically plausible motions [12, 13]. The major problem is its computational cost: each motion computation requires an optimization of hundreds of parameters, which can hardly be done in real-time, especially when synthesizing motions in a dynamic environment.

With **data driven motion synthesis**, all motions including those preparing for the target motions are captured to compose data structures such as motion graphs [1, 2] or finite state machines [14, 15]. One can also interpolate/blend motions to synthesize intermediate motions [16, 17, 18]. However, such methods are data intensive, making it unsuitable for applications such as 3D computer games. In our approach, we represent the style of the motion by a single posture in the target motion. As a result, we can enhance the re-usability of the individual motions and reduce the total amount of data to be pre-captured.

Approaches involving **future prediction** predict upcoming situation and select the most appropriate motion. Support vector machine has been used to select the appropriate motion and avoid external disturbances [19]. Min-max search can control a character to intelligently fight with another character [20, 21]. Reinforcement learning produces control system for characters to fight [22], avoid obstacles [23], control pedestrians [24], and prepare for user controls [25]. In these works, the optimal sequence of captured motion clips are selected and concatenated. Thus, they can only reproduce the captured sequence of motions. In this research, we also apply reinforcement learning to learn the optimal motions for approaching a target while preparing for the up-coming event. The major difference with previous approaches is that we include an addition dimension in the action space, which is called the level of preparation. This allows us to adjust a basic motion for synthesizing a large variety of preparation behaviour.

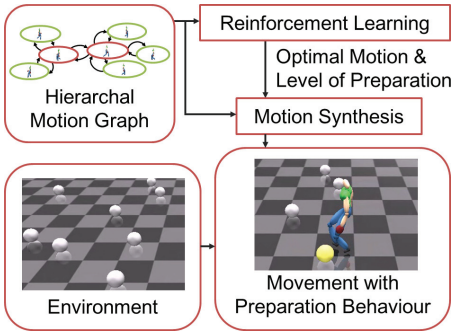

Figure 2: The overview of our method.

3 Methodology Overview

In this paper, we synthesize the behaviour to approach a target while preparing for the subsequent action to interact. The outline of our approach is shown in Figure 2. A hierarchical motion graph is used as the motion database. As an offline process, reinforcement learning is used to train a controller that decides the optimal motion and level of preparation. Based on a given environment and the information from the controller, we synthesize movement with preparation behaviour.

We have two contributions in this paper:

- We propose a customized motion blending algorithm to synthesize preparation movements with a given level of preparation, which is a single numerical parameter to describe the movement style and movement trajectory when preparing for a target motion. The algorithm allows synthesis of preparation movements without capturing extra motions.

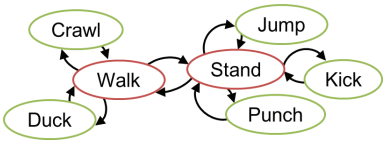

Figure 3: A hierarchical motion graph that includes cyclic locomotion as core nodes and target motions to interact with the environment.

- To control a character to approach a target and with the proper preparation behaviour, we propose a controller created by reinforcement learning. Unlike traditional reinforcement learning approaches, we alter a motion based on the level of preparation, and thus create a large variety of movements.

4 Motion Database

In this section, we explain how we organize the motion database, as well as extracting the reference postures in the motions, which contain the preparation characteristics.

4.1 Hierarchical Motion Graph

Here, we explain the motion graph constructed in this research.

We construct a hierarchical motion graph structure [26, 14, 15] that includes cyclic locomotion as core nodes, such as standing and walking, as well as target motions to interact with the environment, such as crawling, jumping and attacking. Edges in the motion graph represent valid transition of motion with no kinematics artifacts. Figure 3
the motion graph. Notice that while the character can conduct a specific target motion from the locomotion, the behaviour when switching to the target motion is unnatural. One example is walking in full speed and ducking suddenly. The overall movement is kinematically correct, but the behaviour is unrealistic.

4.2 Reference Posture Extraction

Here, we explain how we extract a reference posture from each target motion, which represents the corresponding preparation style.

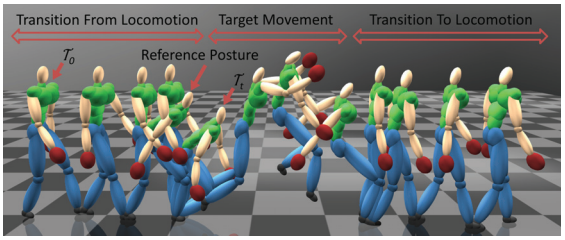

Figure 4: A typical target motion consisting of three parts.

Let us assume $\mathcal{L}$ to be a locomotion and $\mathcal{T}$ to be a target motion. Searching $\mathcal{T}$ for a posture that represents the style of the preparation behaviour is a semi-automatic process. Assuming $\mathcal{T}$ is connectible to $\mathcal{L}$ in the motion graph, it contains three major parts: (1) a kinematically smooth transition from ending posture of $\mathcal{L}$, (2) a duration that involves that target movement, and (3) a transition to the starting posture of $\mathcal{L}$ or another locomotion. Figure 4 shows an example with a jumping motion. In our system, the user identifies the first frame of the target movement, $\mathcal{T}_t$. Our system then automatically scans from $\mathcal{T}_t$ towards the first frame of $\mathcal{T}$, $\mathcal{T}_0$, and monitors the following conditions:

1. The center of mass of the body is within the supporting area.
2. The contact foot pattern is the same as that of $\mathcal{T}_0$.
3. The sum of squares of the posture difference of the legs with respect to $\mathcal{T}_0$ is within a predefined threshold.

Condition 1 ensures us to obtain a stable posture. Conditions 2 and 3 ensure the reference posture does not change the edge connectivity in the motion graph when synthesizing the preparation movement in Section 5. The first frame that satisfies all of conditions is used as the reference posture of the target motion, $\mathcal{T}_p$. If we cannot find such a frame, $\mathcal{T}_0$ is used as the reference posture.

5 Movement Synthesis

In this section, we explain our system to synthesize preparation movement.

We define the term preparation movement as the movement when a character is preparing to conduct a specific target motion. We observed that preparation movement usually involves two characteristics: (1) a change in movement style, and (2) a change in movement trajectory. To facilitate efficient behaviour control, we declare a parameter $\alpha \in \mathbb{R}$ called the *level of preparation*, which is a numerical value to control the two aspects of the preparation movement.

5.1 Movement Style Synthesis

Here, we explain our customized motion blending algorithm to create the preparation movement style.

When given two motions, previous techniques of parametric motion blending use linear blending weight to interpolate the movement of all the degrees of freedom. However, we observe that when humans shift their motions from one to another, the joint movements are not linearly warped altogether. For example, when we prepare for actions such as punching or crawling while walking, we usually adapt the arms and torso quickly in the beginning stage of the preparation movement, while the joints of the lower body are adapted gradually. This is because of the dominant role of the arms in the target motion, while the gait movement does not affect the target motion heavily. Such an observation leads us to control the joints separately during blending.

The blending behaviour for a joint is approximated with a curve equation. The blending weight of joint i is calculated as:

$$w_i = \sqrt[C_i]{1-(1-\alpha)^{C_i}} \tag{1}$$

where α is the level of preparation given by a high level controller that will be explained in Section 6, C_i is a constant individually set for each joint. The blended joint angles of joint i is calculated as:

$$\mathbf{q^i} = (1-w^i)\mathbf{q^i_{\mathcal{L}}} + w^i\mathbf{q^i_{\mathcal{T}_p}} \tag{2}$$

where $\mathbf{q^i_{\mathcal{L}}}$ and $\mathbf{q^i_{\mathcal{T}_p}}$ represent the joint angles in the locomotion $\mathcal{L}$ and reference posture $\mathcal{T}_p$ respectively. When $\alpha = 0$, the character is unprepared and the blending weights are zero. The posture is the same as those in $\mathcal{L}$. When $\alpha = 1$, the character is fully prepared and the weights become one. The resultant posture becomes the reference posture $\mathcal{T}_p$.

The constant C_i in equation 1 represents the blending behaviour of the joint, as it affects the curvature of the blending curve as shown in Figure 5. While it is possible to design the suitable C_i for each joint, we observe that for our motion database, the human joints can roughly be classified into the upper body and the lower body. In our system, we suggest the user with $C_i = 1.0$ for the lower body joints including the root and the legs, and $C_i = 0.5$ for the upper body joints. For other combinations of locomotion and target actions, the user can further edit C_i for specific joints to obtain satisfactory transitions.

An example of using traditional blending method [3] to prepare for a boxing motion is shown in (Figure 6 Top). Notice that the arms raise gradually in increasing level of preparation because a single linear blending parameter is used. The behaviour of the third and fourth postures appears unnatural, as the character raises the arm horizontally while walking. On the other hand, with our approach, the arms quickly switch to the boxing style when the level of preparation increase, while the lower body is adapted linearly (Figure 6 Bottom).

5.2 Movement Trajectory Adjustment

Here, we explain how we adjust the position of the root in the blended motion, which affects the movement trajectory of the character. Notice that the root orientation is scaled in the same way as other joints with Equation 2.

The displacement of the root on the horizontal plane (X-Z plane) depends on the lower body movement. In theory, the best way of calculation is based on the stride of the legs. However, because of the concern of computational cost, we

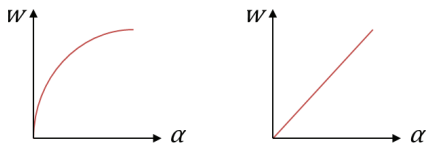

Figure 5: The blending curves for (Left) the upper body with C_i set to 0.5, and (Right) the lower body with C_i set to 1.0.

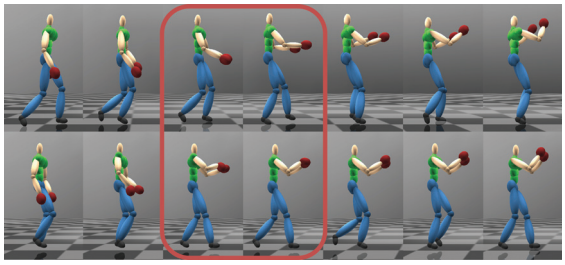

Figure 6: A walking motion with the preparation behaviour to boxing created by (top) traditional blending methods and (bottom) our method.

approximate the horizontal displacement by the blending weight of the root:

$$P_X = (1 - w_{root})P_{X\mathcal{L}} \quad (3)$$

$$P_Z = (1 - w_{root})P_{Z\mathcal{L}} \quad (4)$$

where w_{root} is the blending weight of the root calculated by Equation 1, $P_{X\mathcal{L}}$ and $P_{Z\mathcal{L}}$ represents the horizontal displacement of the locomotion $\mathcal{L}$.

The displacement in the vertical direction (y direction) is calculated based on the difference of the height of the feet before and after the blending. The average difference of the left and right feet height is computed and subtracted from the root height:

$$P_Y = P_{Y\mathcal{L}} - \frac{1}{2}(\Delta Y_l + \Delta Y_r) \quad (5)$$

where $P_{Y\mathcal{L}}$ is the root height in $\mathcal{L}$, and ΔY_l, ΔY_r are the height of the left and right feet relative to the root caused by blending the joint angles.

Finally, we use inverse kinematics to adjust the location of the feet. We detect the foot contact information from the original locomotion data, which is kept the same in the blended motion. When a foot is supposed to support the body, its location is kept the same throughout the supporting phase on the ground. In our implementation, we apply inverse kinematics based on a particle system to solve for the final posture [27].

6 Behaviour Controller

In this section, we explain how we apply reinforcement learning to train a unified controller,

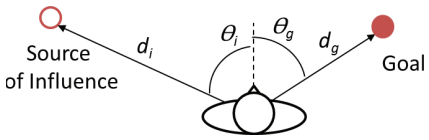

Figure 7: The separated goal and source of influence in our state representation.

which controls the motion that the character should perform to move towards a target, as well as the level of preparation that should be applied.

We define the term preparation behaviour as the high level behaviour to approach a target and prepare to interact. The behaviour involves multiple motions with a change of level of preparation throughout the series of movement.

6.1 State Space and Action Space

Here, we first explain the *states* and *actions* designed in our system.

We assume the character has a goal that it has to reach, and there is a source of influence that affects the preparation behaviour. The goal of movement and source of influence are usually identical, such as approaching a ball and preparing to kick it. The advantage of separating them into two variables is that it allows more general definition and thus a wider range of synthesis. For example, we can simulate a character walking in one direction, while preparing to avoid a nearby obstacle that blocks the way.

We create a state space $\mathbb{S}$ in which every state is defined as:

$$s = \{\alpha, Next(m), d_g, \theta_g, d_i, \theta_i\}, s \in \mathbb{S} \quad (6)$$

where α is the level of preparation in the current time instance, m is the last action performed, $Next(m)$ is the set of available actions to be performed after m according to the motion graph, d_g and θ_g are the distance and relative orientation between the character and the goal, d_i and θ_i are those with respect to the source of influence (Figure 7). Except from $Next(m)$, the parameters in the state are continuous numerical value.

We create an action space $\mathbb{A}$, in which each action is defined as:

$$a = \{m, \alpha\}, a \in \mathbb{A} \quad (7)$$

where m is the action to be performed, α is the corresponding level of preparation. Notice that different from traditional reinforcement learning [28], the action in our system is not simply a movement. Instead, it involves the level of preparation, α, that adjusts the movement. α is quantized during training. Because using discontinuous values of α in simulation could result in discrete behaviours, we apply a Gaussian filter on α to smooth the value of time.

6.2 Reward Function

Here, we explain the *reward function*, which evaluates the actions in a given state.

Let us assume the current state to be s_t, and the next state after performing the chosen action a_{t+1} to be s_{t+1}. α in the action is feedback to the state whenever an action is selected: $s_{t+1}.\alpha \leftarrow a_{t+1}.\alpha$.

The reward function that evaluates a_{t+1} consists of three terms. The first term evaluates how much the character faces the target:

$$f_\theta = -|s_{t+1}.\theta_g| \tag{8}$$

where $s_{t+1}.\theta_g$ denotes the angle with respect to the goal for the state s_{t+1}. The second term evaluates how much the character approaches the goal:

$$f_d = |s_{t+1}.d_g - s_t.d_g| \tag{9}$$

where $s_{t+1}.d_g$ and $s_t.d_g$ denote the distance to the goal for the two states. The last term evaluates how steadily the level of preparation changes:

$$f_\alpha = -|s_{t+1}.\alpha - s_t.\alpha|^2. \tag{10}$$

where $s_{t+1}.\alpha_{current}$ and $s_t.\alpha_{current}$ denote the level of preparation of the two states. This term is used to penalize sudden change of preparation level, and the square operator is used to magnify the difference between small and large changes. During training, when a character reaches the source of influence, α is forced to 1.0. Therefore, the trained system increases α gradually before reaching the source of influence to minimize this term.

Finally, the reward function is defined as:

$$r_t = w_\theta f_\theta + w_d f_d + w_\alpha f_\alpha \tag{11}$$

$$\tag{12}$$

where w_θ, w_d, w_α are constant weights, and are empirically set to 0.1, 0.5, and 0.8 respectively.

6.3 SARSA Learning

Here, we explain the concept of the *return*, and the SARSA approach that is used to solve it.

The return is the long term benefit of launching a series of actions based on a *control policy*, which tells the action to be performed in a given state:

$$R = \sum_{t=0}^{n} \gamma^t r_t \tag{13}$$

where γ is called a *discount value* that reduces the influence of future states due to their uncertainties, and n is the number of steps until the character reaches the terminal state.

To find an *optimal policy* that maximizes R in Equation 13 at every state, we apply SARSA learning because it is efficient and the exploration decreases in a proper manner. Here, we briefly explain the SARSA framework. The reader is referred to [29, 30] for further details.

Using SARSA, the system learns the *state action value* $Q(s,a)$, which is the estimated return of launching an action a at state s, for all possible state-action combinations. A control policy is defined as the optimal choice of action based on the set of the state action values for all states. The objective is to learn the optimal policy by converging $Q(s,a)$ to the maximum R in Equation 13.

The training is divided into *episodes*. At each episode, we randomly pick a state $s \in \mathbb{S}$, allow the character to perform an action $a \in \mathbb{A}$ based on the current policy, and observe the reward r. The character keeps launching actions until the terminal state, for which we define as reaching the goal position. SARSA represents Equation 13 as:

$$Q(s_t,a_t) \leftarrow Q(s_t,a_t) + \lambda(r_{t+1} + \gamma Q(s_{t+1},a_{t+1}) - Q(s_t,a_t)) \tag{14}$$

where λ is the *rate of learning*, which is set to 0.7 in our experiments, and γ refers to the discount value as shown in Equation 13, which is set to 0.3.

The control policy is updated whenever a state action value is changed. Training is conducted using a large number of episodes until the policy becomes optimal (i.e. the policy cannot be improved further). The algorithm converges provided that all state-action pairs are visited infinite number of times. In practice, to reduce the

convergence time, we keep track of the number of state action values updated per episode. If the number drops below 2% of total number of state, we terminate the training and assume the policy to be optimal.

6.4 Maintaining Exploration

It is important to maintain exploration during the training process in order to obtain a global optimal policy. We apply two strategies to do so.

Exploring Starts: We attach a flag for each action in all the states to indicate if the actions have been selected or not. During training, higher selection priority is given to the actions that have not been selected. This ensures uniform exploration in the state action space during the early stage of training.

ϵ-Greedy Policy: When selecting an action, there is an ϵ chance that the system randomly selects an action, instead of picking the one with the best state-action value. This ensures exploration throughout the training. We set ϵ to 0.1 in our system.

7 Experiments

In this section, we explain the experiments we conducted to evaluate our system.

7.1 System Setup

All experiments are performed using a computer with an Intel Dual-Core 3.0GHz CPU and 2GB of RAM.

We created a motion database with 78 motions, including walking, crawling, ducking, jumping, punching and kicking. The cyclic walking motions including stepping to different directions were used as the core nodes in the hierarchical motion graph, while the rest were used as target motions. The created graph contained 4 core nodes.

We quantized the state and action spaces for the SARSA learning. The sampling steps and number of sample for each variable are shown in Table 1. The training stage took around 1 hour, while the run-time system was conducted in real-time.

	d_i	θ_i	d_g	θ_g	α
Steps	$60cm$	$90°$	$60cm$	$45°$	0.2
Number	5	4	2	8	5

Table 1: The sampling steps and number of samples in the state space.

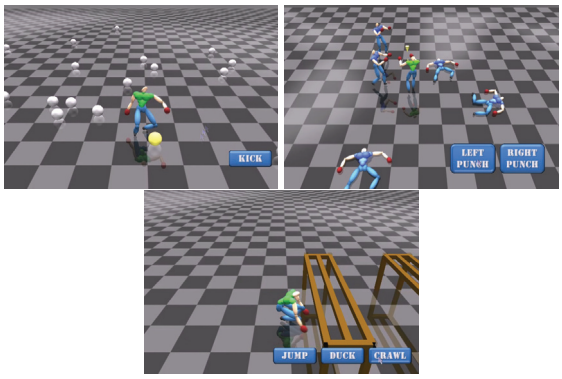

Figure 8: Preparation behaviours created by our system for different target motions.

7.2 Preparation Control

An interactive 3D application prototype was developed to show the effectiveness of our system. Notice that our preparation control system does not involve launching the target motions. While it is possible to define procedural rules that control the character launching such motions, we prefer an interactive system where the user can indicate the timing that the motions are launched for better flexibility.

In the first experiment, we created an interface for the user to select a ball in a given environment, which was used as both the goal location and the source of influence. The system controlled the character to approach the ball with the appropriate preparation behaviour. The user then indicated when a kicking motion is launched to interact with the ball. Notice that when the character approached the ball, it lowers its center of mass and slows down to prepare for the kicking motion (Figure 8 Left).

The second experiment involved interacting with NPCs that were controlled by a procedural controller to move towards the user controlled character. The user used the keyboard to indicate the goal, while using the mouse to launch target motions. The closest NPC was consid-

ered as the source of influence. As a result, the user controlled character raised the arm, slowed down and prepared the punch whenever a NPC was nearby. Notice that the left punch and the right punch motion had the same reference posture $\mathcal{T}_p$, and thus the preparation styles were the same (Figure 8 Middle).

In the third environment, the character had to navigate through a complex environment with different obstacles. Again, the user controlled the goal and the launch of the target motions. The source of influence was set as the closest obstacle in the facing direction, and the preparation style was based on the corresponding target motion to interact with the obstacle. The system created different levels of preparation behaviours to perform crawling, ducking and jumping. Notice that with our system, the character maintained a preparation style when getting through two nearby obstacles, generating a human-like behaviour (Figure 8 Right).

8 Conclusion and Discussions

In this paper, we proposed a method to synthesize preparation behaviour for the next motion to be launched. Using our method, we could generate realistic behaviours to prepare for the next action as humans do. Our system can produce convincing results in real-time, which makes it suitable for real-time applications such as games.

We observe that in console games nowadays, due to the lack of preparation movement, character movement is usually unrealistic. Because the character does not consider the next motion, it is common that a character performs a target motion, runs in full speed, and performs a target motion again. In our system, the character maintains the preparation style if there are remaining targets nearby.

A possible solution for preparation behaviour synthesis is the data-driven approach [17, 16]. For every pair of locomotion and subsequent action, one can capture different levels of preparatory motions. However, this approach is not preferred due to the large amount of motion that has to be captured. Also, it is difficult for a human to exhibit preparatory behaviour properly with small change of level of preparation, because the movements are usually performed subconsciously in daily life.

The major challenge of creating preparation behaviour is that it affects the movement of the locomotion. The character needs to approach the target quickly, while adjusting movement trajectory to prepare the target motion in advance. We set up cost functions and conducted reinforcement learning to solve the multi-modal problem as a whole.

A feature of our design is that the state space representation in Equation 6 contains the locomotion only without the target motion. This design allows us to reuse a trained system for different target motions, as long as the bending curve of the lower body remains unchanged. If the bending curves for two target motions are different, the lower body, and hence the movement trajectory, in the synthesized motion will be different. As a result, a controller has to be trained for each motion.

At the current stage, our system cannot synthesize the preparation behaviour when there are multiple potential target motions, unless they have the same reference posture and blending curve. As a future direction, we plan to control the character intelligently in a dynamic environment where arbitrary events can happen in a probabilistic manner. We could apply Bayesian models to let the characters predict what kind of behaviours could benefit them the most under different environments.

References

[1] Jehee Lee, Jinxiang Chai, Paul S. A. Reitsma, Jessica K. Hodgins, and Nancy S. Pollard. Interactive control of avatars animated with human motion data. *ACM Trans. Graph.*, 21(3):491–500, 2002.

[2] Lucas Kovar, Michael Gleicher, and Frédéric H. Pighin. Motion graphs. *ACM Trans. Graph.*, 21(3):473–482, 2002.

[3] Andrew Witkin and Zoran Popovic. Motion warping. In *Proceedings of the 22nd annual conference on Computer graphics and interactive techniques*, SIGGRAPH '95, pages 105–108, New York, NY, USA, 1995. ACM.

[4] Ronan Boulic, Pascal Bécheiraz, Luc Emering, and Daniel Thalmann. Integration of motion control techniques for virtual human and avatar real-time animation. In *Proceedings of the ACM symposium on Virtual reality software and technology*, VRST '97, pages 111–118, New York, NY, USA, 1997. ACM.

[5] S. Ménardais, F. Multon, R. Kulpa, and B. Arnaldi. Motion blending for real-time animation while accounting for the environment. In *Computer Graphics International*, pages 156–159, Crete, Greece, June 2004.

[6] Hubert P. H. Shum, Taku Komura, and Pranjul Yadav. Angular momentum guided motion concatenation. *Comput. Animat. Virtual Worlds*, 20(2-3):385–394, 2009.

[7] Matthew Brand and Aaron Hertzmann. Style machines. In *SIGGRAPH '00: Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, pages 183–192, New York, NY, USA, 2000. ACM Press/Addison-Wesley Publishing Co.

[8] Eugene Hsu, Kari Pulli, and Jovan Popović. Style translation for human motion. *ACM Trans. Graph.*, 24(3):1082–1089, 2005.

[9] Andrew Witkin and Michael Kass. Spacetime constraints. In *SIGGRAPH '88: Proceedings of the 15th annual conference on Computer graphics and interactive techniques*, pages 159–168, New York, NY, USA, 1988. ACM.

[10] Michael Gleicher and Peter Litwinowicz. Constraint-based motion adaptation. *The Journal of Visualization and Computer Animation*, 9(2):65–94, – 1998.

[11] Michael Gleicher. Retargetting motion to new characters. In *Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '98, pages 33–42, New York, NY, USA, 1998. ACM.

[12] C. Karen Liu and Zoran Popović. Synthesis of complex dynamic character motion from simple animations. In *SIGGRAPH '02: Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, pages 408–416, New York, NY, USA, 2002. ACM.

[13] Anthony C. Fang and Nancy S. Pollard. Efficient synthesis of physically valid human motion. *ACM Trans. Graph.*, 22(3):417–426, 2003.

[14] Manfred Lau and James J. Kuffner. Behavior planning for character animation. In *SCA '05: Proceedings of the 2005 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 271–280, New York, NY, USA, 2005. ACM.

[15] Taesoo Kwon and Sung Yong Shin. Motion modeling for on-line locomotion synthesis. In *SCA '05: Proceedings of the 2005 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 29–38, New York, NY, USA, 2005. ACM.

[16] Lucas Kovar and Michael Gleicher. Automated extraction and parameterization of motions in large data sets. *ACM Trans. Graph.*, 23(3):559–568, August 2004.

[17] Tomohiko Mukai and Shigeru Kuriyama. Geostatistical motion interpolation. *ACM Trans. Graph.*, 24(3):1062–1070, 2005.

[18] Alla Safonova and Jessica K. Hodgins. Construction and optimal search of interpolated motion graphs. In *SIGGRAPH '07: ACM SIGGRAPH 2007 papers*, page 106, New York, NY, USA, 2007. ACM.

[19] Victor Zordan, Adriano Macchietto, Jose Medin, Marc Soriano, Chun-Chih Wu, Ronald Metoyer, and Robert Rose. Anticipation from example. In *VRST '07: Proceedings of the 2007 ACM symposium on Virtual reality software and technology*, pages 81–84, New York, NY, USA, 2007. ACM.

[20] Hubert P. H. Shum, Taku Komura, and Shuntaro Yamazaki. Simulating interactions of avatars in high dimensional state space. In *I3D '08: Proceedings of the 2008 symposium on Interactive 3D graphics and games*, pages 131–138, New York, NY, USA, 2008. ACM.

[21] H.P.H. Shum, T. Komura, and S. Yamazaki. Simulating multiple character interactions with collaborative and adversarial goals. *IEEE Transactions on Visualization and Computer Graphics*, 18(5):741 –752, may 2012.

[22] Julian Gold Thore Graepel, Ralf Herbrich. Learning to fight. In *Proceedings of the International Conference on Computer Games: Artificial Intelligence, Design and Educatio*, 2004.

[23] Leslie Ikemoto, Okan Arikan, and David Forsyth. Learning to move autonomously in a hostile world. In *SIGGRAPH '05: ACM SIGGRAPH 2005 Sketches*, page 46, New York, NY, USA, 2005. ACM.

[24] Adrien Treuille, Yongjoon Lee, and Zoran Popović. Near-optimal character animation with continuous control. *ACM Trans. Graph.*, 26(3):7, 2007.

[25] James McCann and Nancy Pollard. Responsive characters from motion fragments. *ACM Trans. Graph.*, 26(3):6, 2007.

[26] Hyun Joon Shin and Hyun Seok Oh. Fat graphs: constructing an interactive character with continuous controls. In *SCA '06: Proceedings of the 2006 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 291–298, Aire-la-Ville, Switzerland, Switzerland, 2006. Eurographics Association.

[27] Thomas Jakobsen. Advanced character physics. *Game Developers Conference Proceedings*, 2001.

[28] Jehee Lee and Kang Hoon Lee. Precomputing avatar behavior from human motion data. In *SCA '04: Proceedings of the 2004 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 79–87, Aire-la-Ville, Switzerland, Switzerland, 2004. Eurographics Association.

[29] G. A. Rummery and M. Niranjan. On-line Q-learning using connectionist systems. Technical Report CUED/F-INFENG/TR 166, Cambridge University Engineering Department, 1994.

[30] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, MA, USA, 1998.