JOURNAL OF INFORMATION SCIENCE AND ENGINEERING 37, 535-551 (2021)
DOI: 10.6688/JISE.202105_37(3).0003

A Study on Agent-Based Box-Manipulation Animation Using Deep Reinforcement Learning

HSIANG-YU YANG, CHIEN-CHOU WONG AND SAI-KEUNG WONG+
College of Computer Science
National Yang Ming Chiao Tung University
Hsinchu, 300 Taiwan

This paper focuses on push-manipulation in an agent-based animation. A policy is learned in a learning session in which an agent perceives its own internal state and the surrounding environment and determines its actions. In each time step, the agent performs an action. Then it receives a reward that is a combination of different types of reward terms, including *forward progress*, *orientation progress*, *collision avoidance*, and *finish time*. Based on the received reward, the policy is improved gradually. We develop a system that controls an agent to transport a box. We investigate the effects of each reward term and study the impacts of various inputs on the performance of the agent in environments with obstacles. The inputs include the number of rays for perceiving the environment, obstacle settings, and box sizes. We performed some experiments and analyzed our findings in details. The experiment results show that the behaviors of agents are affected by the reward terms and various inputs in certain aspects, such as the movement smoothness of the agents, wandering about the box, loss of orientation, sensitivity about collision avoidance, and pushing styles.

Keywords: reinforcement learning, agent-based, animation, box manipulation, reward terms

1. INTRODUCTION

Agent-based box manipulation has been researched in robotics and computer animation. The basic idea is to control a single agent or multiple agents to transport a box to a predefined destination. Techniques for agent-based box manipulation have wide applications in robotics, computer games, movies, and computer animations. Various classes of techniques have been developed, such as rule-based, sensor-based, and reinforcement learning. Reinforcement learning techniques have been applied to robot control [1], robotic soccer [2], and motion of biped characters [3]. In reinforcement learning techniques, there is a learning session in which policies are learned based on rewards received by an agent in a learning environment. The agent collects the information of the surrounding environment and then performs an action. The reward received by the agent depends on the degree of forward progress and the system stability. After the learning session, the learned policies are adopted to control the agent to carry out certain tasks in environments similar to the learning environment. Reinforcement learning [4] and deep neural networks [5] can be integrated to achieve deep reinforcement learning [6].

Received August 11, 2020; revised October 6, 2020; accepted November 1, 2020.
Communicated by Chih-Hung Wu.
+ Corresponding author.

Box-manipulation techniques have been developed in crowd animation. In [7, 8], agents are simulated to manipulate boxes in an environment with pedestrians. In [7], the behaviors of agents in *pushing*, *pulling with ropes*, and *carrying* are simulated. A user edits a path for the agents to push a box along the path. The process of editing the path is time consuming and tedious. In [8], only push-manipulation is considered and collision probability fields are employed for achieving collision avoidance [9]. In both methods [7, 8], the contact points between agents and the box are assumed to be fixed. The methods rely on some fixed rules to control the agents to perform actions. The disadvantage of using rules is that they may not be applicable in environments with different settings. Such rules have parameters which are required to tune carefully. Also, the agents and the boxes look like a rigid body as a whole when they move.

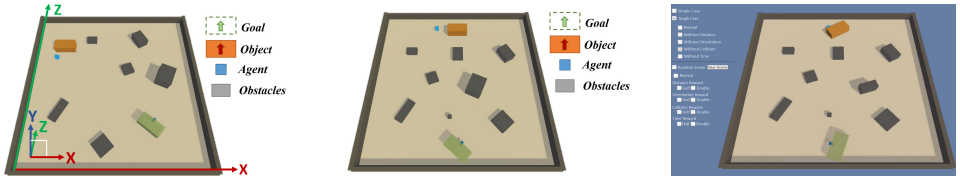
Yang and Wong [10] proposed a method which incorporates deep reinforcement learning and some heuristic rules. The method generates animations of agents pushing and pulling boxes in an environment with obstacles and movable objects. Tools are proposed to assist the completion of animations, including 1) moving objects away from the path of a main object and 2) assigning agents to move unassigned objects. Thus, the method reduces unnecessary labor effort. But the method has several reward terms whose effects are unclear. We adopted the same deep reinforcement leaning approach as [10] and developed a system for generating box-manipulation animation. Our preliminary study investigated the effects of reward terms [11]. In this version, we studied the impacts of various inputs and performed an in-depth analysis about the performance of the agents. Our contributions are as follows.

1. Investigate the effects of the reward terms in-depth. It is importance to understand well the reward terms so that further improvement can be conducted.
2. Study the impacts of various inputs on the performance of the system, *e.g.*, the number of rays, obstacle settings, and box sizes.
3. Analyze the behaviors of the agents under different inputs.

2. RELATED WORK

Neural networks and genetic algorithm have been employed in building controllers in games [12]. Human-level controllers can play games based on raw images of game content [13]. Agents learn to navigate in a Minecraft maze based on a memory-based architecture integrated with deep reinforcement learning [14]. Techniques are developed to automatically generate animation of biped characters [3,15,16]. Deep learning techniques are employed in building controllers in physics based simulation, such as rigid body control [17], cloth dressing [18], and winged creatures [19]. Deep learning techniques are applied in crowd simulation [20-22].

The common transportation techniques include pushing, pulling, grasping and caging. Push-manipulation skills of robots can be learned from experience in real world [23]. Agents can form different formation patterns to transport a large set of passive objects [24]. While agents are moving, they should avoid collision with other objects. To achieve collision avoidance, factors such as obstacle fields and pedestrian flows are considered [7, 8].



(a) (b) (c)

Fig. 1. Two scene examples and the user-interface (UI); Objects include an agent, a box, a goal, and some obstacles; (a) A sparse environment; The objects are organized in a sparse manner; (b) A dense environment; There are three layers of obstacles between the box and the goal; (c) The UI of the testing system.

(a) (b) (c) (d)

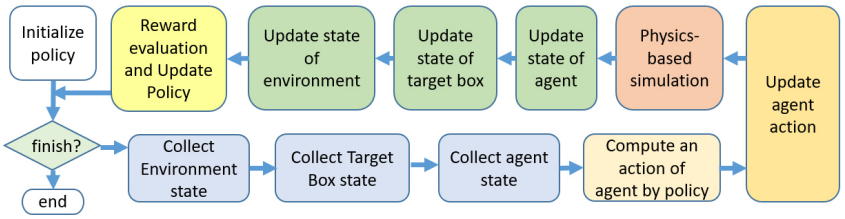
Fig. 2. A living room (a)-(b) and a library (c)-(d); An agent pushes a box to a destination; The yellow curves indicate the movement paths of the agent; Our method produces natural motion of the agent while the agent pushes the box.

3. SYSTEM OVERVIEW

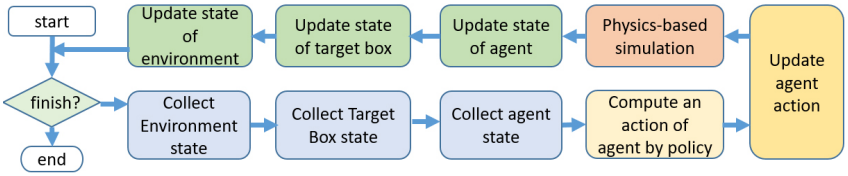
We develop our simulation based on the work by Yang and Wong [10]. Objects (*e.g.*, agents and boxes) move on the x-z plane in a three-dimensional space (*i.e.*, world frame) and the y-axis represents height. Each object is associated with a local reference frame (shortly local frame). When the object moves (*e.g.*, rotate or translate), its local frame changes accordingly. There are three kinds of objects: agents, passive objects, and static objects. An agent is controlled by a learned policy which is obtained in a learning session. An object i has some attributes including position $\mathbf{p}_i$, velocity $\mathbf{v}_i$, and direction $\mathbf{h}_i$. $\mathbf{h}_i$ is a unit vector which is the x-axis of object i in the world frame. Passive objects are boxes which are movable. Static objects cannot be moved. Agents and movable objects can rotate about the y-axis. In an animation, there is a target box which is movable; and an agent can push it. Figs. 1 shows two scene examples for tests. We consider that objects are organized so that they do not overlap with each other and there are no dead-ends. Our method can be applied in producing an agent-based animation in different environments such as a living room and a library (Fig. 2).

The orientation ϕ_i of an object i is the signed angle between $\mathbf{h}_i$ and the x-axis of the world frame. An object can be determined uniquely by its geometric center, $\mathbf{p}_i$, and its orientation ϕ_i in the 2D spatial domain. We consider a target box j. A goal of the target box has two conditions: 1) position condition and 2) orientation condition. Denote ε_p and ε_ϕ as the control parameters for the two conditions. The two conditions of goal $g(j, C_p, C_\phi)$ are as follows:

1. Position condition. $C_p(j,g) : \|\mathbf{p}_j - \mathbf{q}_g\| \leq \varepsilon_p$, where $\mathbf{q}_g$ is the goal position.



(a) The learning process.



(b) The animation process.

Fig. 3. The flowcharts of the learning process (a) and the animation process (b).

2. Orientation condition. $C_\phi(j,g) : |\phi(j,g)| \leq \varepsilon_\phi$, where $\phi(j,g)$ is the signed angle between $\mathbf{h}_j$ and $\mathbf{h}_g$; and $\mathbf{h}_g$ is the goal direction (or goal orientation).

When all the conditions are satisfied, the goal $g(j,C_p,C_\phi)$ is finished. The orientation condition is ignored if $\varepsilon_\phi \to +\infty$.

There are two main processes: 1) learning and 2) animation. In the learning process, two policies are learned, one for controlling an agent to navigate and another for controlling an agent to push a box to a destination. The learning process has the following steps:

1. **Policy specification**. Set a behavior to be learned by a policy and initialize the policy parameters.
2. **Scene construction**. In an environment, randomly generate an agent, a target box, a goal, and static objects. The objects do not overlap and there are no dead-ends.
3. **Observation gathering**. Collect environment observations of the agent.
4. **Action computation**. Employ the policy to compute an action of the agent.
5. **State update**. Update the states of the agent and the box based on the agent action.
6. **Reward computation**. Compute the reward for the agent based on the simulation result.
7. **Policy update**. Update the policy of the agent.
8. Repeat steps 3 to 7 until the episodic session is finished.

After the learning process, the learned policies are employed in the animation process to control agents. Fig. 3 shows the flowcharts of the learning process and the animation process.

4. REINFORCEMENT LEARNING

An agent exhibits *navigation* and *pushing* behaviors. An agent performs navigation to move to a position or a box. When the agent is near to the box, it pushes the box

to the goal position. In the following, we consider that a policy b (either pushing or navigation) is learned to control the agent. The policy $\pi^b(\mathbf{a}|\mathbf{s})$ represents a conditional distribution of actions $\mathbf{a}$ over a given state $\mathbf{s}$. The agent's state is $\mathbf{s}_t$ at time t. The next action $\mathbf{a}_{t+1}$ which maximizes the long term reward is determined by the policy. After the agent performs the action, it receives a reward r_t. Then $\mathbf{s}_{t+1}$ is obtained as the new state of the agent. Consequently, a sequence of states, actions, and rewards, is obtained, *i.e.*, $\{(\mathbf{s}_0,\mathbf{a}_0,r_0),(\mathbf{s}_1,\mathbf{a}_1,r_1),\cdots\}$. Denote $r(\mathbf{s}_{t+i},\mathbf{a}_{t+i})$ as the received reward and $\gamma\,(\in[0,1])$ a discount factor. The cumulative reward is computed as

$$R_t = \Sigma_{i=0}^{\infty}\gamma^i r(\mathbf{s}_{t+i},\mathbf{a}_{t+i}). \tag{1}$$

The parameters θ of the policy are adjusted to maximize the expected reward $\mathbf{J}(\theta) = \mathbb{E}_\pi[R_t]$. Proximal policy optimization [25] is employed to update the parameters. Please refer to [25] for details.

4.1 States of an Agent

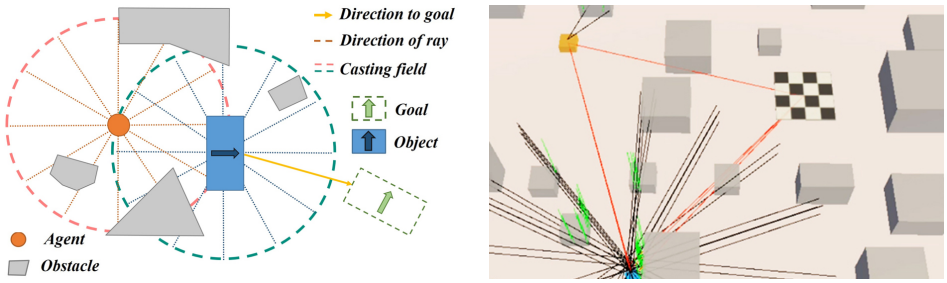
At each simulation step, the state of an agent i is determined based on the goal conditions and the observation information of the agent. We consider a goal $g(j,C_p,C_\phi)$ for a target j. The two goal conditions are $C_p(j,g)$ and $C_\phi(j,g)$. Here, the target j can be either an agent or a box. If j is an agent, the agent performs navigation.

The state $\mathbf{s}_t$ of agent i at time t has two components: an internal state $\mathbf{s}_{int}$ and an external state $\mathbf{s}_{ext}$ (Figs. 4 (a) and (b)). We have $\mathbf{s}_{int} = (\mathbf{s}_p,\mathbf{s}_h,\mathbf{s}_i)$, where $\mathbf{s}_p = (\mathbf{q}_g - \mathbf{p}_j)$ (measured in the local frame of j), $\mathbf{s}_h = \phi(j,g)$, and $\mathbf{s}_i = (\mathbf{p}_j - \mathbf{p}_i)$ (measured in the local frame of i); and $\mathbf{s}_{ext}$ contains the depths of a set of rays and the labels of the hit objects (*i.e.*, obstacles and target box). The agent has two layers of rays [10], *i.e.*, a set of horizontal rays and another set of rays with a tilt angle, *e.g.*, 15 degrees. If the agent pushes a box, rays are also attached at the box and such rays represent the observation of the agent about the surrounding environment of the box. A pair of values are stored for each ray. The first value indicates whether the ray hits an object and the second value is the distance between the ray's origin and the hit object. Intuitively, the rays encode how far an agent can see or how far a box can be pushed forward.

4.2 The Learning Process

A policy is learned in an episodic task in which an agent i performs a behavior. The state of the agent can be obtained in a physics based engine. In our case, we use Unity and set the simulation time step as Δt (*e.g.*, 0.02s). Two output format can be constructed: continuum format and discrete format. In continuum format, the policy π^b returns an action $\mathbf{a} = (k_\phi,\mathbf{k}_v)$. Then we update the orientation, velocity, and position of the agent as follows [10]:

$$\begin{aligned}
\phi_i &\leftarrow \phi_i + k_\phi \Delta t \\
\mathbf{v}_i &\leftarrow \mathbf{v}_i + \mathbf{k}_v \Delta t \\
\mathbf{p}_i &\leftarrow \mathbf{p}_i + \mathbf{v}_i \Delta t + \frac{1}{2}\mathbf{k}_v \Delta^2 t
\end{aligned} \tag{2}$$



(a) (b)

Fig. 4. Rays for gathering environment observations. In (a), an agent and a target box have their own set of rays. In (b), the (black) rays associated with an agent and a target box. The chessboard is the goal.

where $k_\phi \in [-1,1]$, $\mathbf{v}_i$ is the velocity, and $\mathbf{k}_v$ is the acceleration. In the discrete format, the policy returns an action that encodes six movements which are forward, backward, moving left, moving right, clockwise rotation, and anti-clockwise rotation. The physics engine performs collision detection and collision response. After that all the objects are updated at the end of each simulation step. The process is repeated until a termination condition is reached, *e.g.*, the maximum number of steps or finishing the goal conditions.

4.3 Rewards

In general, a reward r can be computed as a sum of several independent reward terms r_i. These reward terms should capture the forward progress of a task and maintain the system stability. Formally, we have $r = \sum_i r_i$. In the simulation space, there are an agent, a box, and a set of static obstacles. While the agent moves or pushes the box, collision may occur. In such case, the adopted action should be discouraged. However, if the agent moves closer to a goal position or avoids collision, such action should be encouraged. Of course, if the agent finishes the goal faster, a higher reward should be received. Four reward terms are considered. They are *forward progress* r_d, *orientation progress* r_o, *collision avoidance* r_c, and *finish time* r_f. The Appendix details the four reward terms. The term r_d monitors the degree of forward progress. When the target is getting closer to the goal position, a higher reward is received. The term r_o monitors the degree of orientation progress. The term r_f monitors the *finish time* and a higher reward is received if the goal is finished earlier. The term r_c discourages collision with obstacles while the agent moves or pushes a box. However, there is no penalty for the agent colliding with the box.

5. EXPERIMENT DESIGN

We investigated the effects of the reward terms in two different environments as can be seen in Figs. 1 (a) and (b). In Fig. 1 (a), objects are far away from each other. In Fig. 1 (b), there are three layers of obstacles between a box and a goal position. During the learning session, the position, orientation and dimension of each obstacle were fixed

in every episode. Furthermore, the position of the box and the goal position were the same in episodes. But the orientation of the box and the goal orientation were randomly generated. The dimensions of the agent and the box were 1^3 and $6 \times 3 \times 2$, respectively. A standard set of weights for the four reward terms was tuned to produce desired animation results. Then two schemes were performed: ablation analysis and variations of weights. In ablation analysis, a reward term was ignored. In *variations of weights*, we doubled or took half of each reward weight in the learning processes. Furthermore, we conducted experiments with different conditions to analyze the system performance. The conditions included different number of rays, obstacles with different scale factors, and boxes (or blocks) with different sizes. Furthermore, we reported how the agents performed under these conditions.

6. EXPERIMENTS AND RESULTS

Our system was built in Unity which had a deep reinforcement learning toolkit [26]. Fig. 4 (b) shows an example in a learning session.

6.1 Ablation Analysis

Our experiments were performed on an Intel(R) Core(TM) i7-7700 CPU @ 3.60GHz quad core with 16.0 GB RAM and an NVIDIA GeForce GTX 1080. Our system adopted the default setting in the Unity Machine Learning toolkit. We used a fully connected neural network consisting of two hidden layers and each layer had 256 nodes. The inputs contained the relative position between the agent and the box, distance between the box and the goal, relative orientation between the box and the goal (if adopted), and the hit distances of the rays associated with the box and the agent. The activation function was *swish*. The continuum format was adopted for the output. Interactive performance was achieved in all the examples in the animation process. Figs. 5 (a1) and (a2) show the cumulative rewards in the sparse environment. The agent can push the box to the goal when all the four reward terms are adopted. The followings summarize our findings:

1. Without the distance reward r_d in the learning session, an agent does not know in which direction that it should push the box towards the goal. In the learning process, the agent may get penalty if it rotates the box in a wrong direction or the target box collides with the obstacles. Thus, the agent learns that it should not touch the box. It wanders near the target box.
2. Without the orientation reward r_o, the agent does not adjust the orientation of the box to satisfy the goal orientation. While the box is satisfied with the position condition $C_p(\mathbf{p}_i, \mathbf{q})$, the agent may receive the large penalty if it moves the box far away from the goal. Thus, the agent is not willing to interact with the box if the position condition is satisfied.
3. Without the collision reward r_c, the agent may not avoid the obstacles while transporting the box. Even the obstacles are right in front of the box, the agent attempts to push the box to hit the obstacles.
4. Without the finished time reward r_t, the agent gets higher cumulative reward in the learning session. But sometimes the agent does not adjust the orientation of the box to satisfy the goal orientation when the box is getting closer to the goal.

We also trained the agents in the dense environment. Figs. 5 (b1) and (b2) show the reward curves. In this case, the agents failed to push the target box to the goal most of time. Thus, the received rewards were pretty low. The target box was stuck at the static obstacles and the agents could not move the target box away from them. However, if we set some intermediate points at open places and instructed the agents to move the target box to them one by one, the agents could successfully push the target box to desired positions. We found that if the target box was smaller, its chance of getting stuck might be lower.

6.2 Variations of Weights for Reward Terms

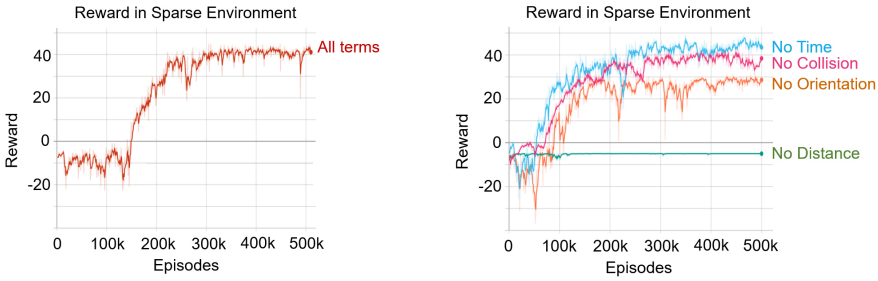
Fig. 6 shows the cumulative reward in different cases in the sparse environment. The results are summarized as follows:

1. Distance reward is $\frac{w_1}{2} r_d$. The agent may push the target box in an opposite direction to the goal at the beginning.
2. Distance reward is $2w_1 r_d$. The agent performs in a way similar to the case in which there is no orientation reward r_o. Furthermore, the agent may not be willing to adjust the orientation if the position condition is satisfied.
3. Orientation reward is $\frac{w_2}{2} r_o$. The agent performs similar to the normal condition. But it takes longer time and less reward in the learning process.
4. Orientation reward is $2w_2 r_o$. The agent adjusts the orientation earlier than the normal condition.
5. Collision reward is $\frac{w_3}{2} r_c$. The agent collides with the obstacles for around two or more times with obstacles in the same environment.
6. Collision reward is $2w_3 r_c$. The agent makes sure that the box does not collide with the obstacles before it pushes the box to the goal position.
7. Finish-time reward is $\frac{w_4}{2} r_t$. The agent takes longer time to adjust the orientation of the box. And it may not rotate the box in the fastest way.
8. Finish-time reward is $2w_4 r_t$. The agent transports the box in a way with a shorter distance. But it does not avoid collision with the obstacles.

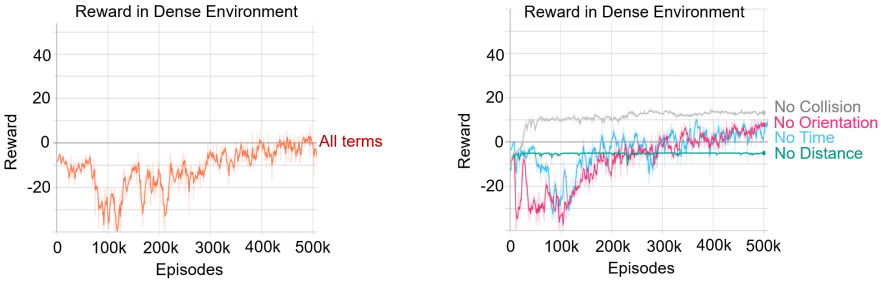
An animator can set the weights to achieve the desired effects in an animation. Sometimes, the animator may want to make the agent push the box carefully or carelessly. Different animation styles of an agent can be achieved by setting the weights properly. Of course, it is desirable to be able to set the weights automatically so that labor work can be reduced. In the future, we shall collect motion data of all the objects and then analyze motion patterns of the main objects (*e.g.*, the agent or the box) in an automatical way. In this way, an animator can intuitively select a desired animation type.

6.3 Results With Various Inputs

We investigated how the agent performed for various inputs. The training environment was computed as follows. In each epoch, the positions of the obstacles were generated randomly. Furthermore, a gap between obstacles must be 3 units or higher so that the agent did not push the target box to a dead end. Based on our observations, the agent could push the target box to the destination when the reward was higher than 4. For the reward lower than 4, the agent often failed to push the target box to the destination. In



(a1) Reward with all terms in the sparse environment. (a2) Reward with one term taken out.



(b1) Reward with all terms in the dense environment. (b2) Reward with one term taken out.

Fig. 5. (a1)-(a2) Cumulative rewards of agents in the sparse environment; (b1)-(b2) Cumulative rewards of agents in the dense environment.

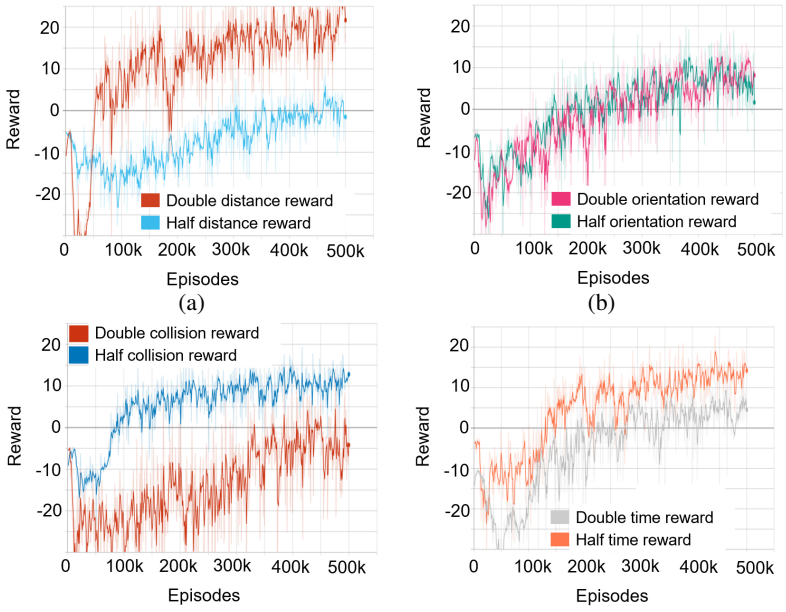
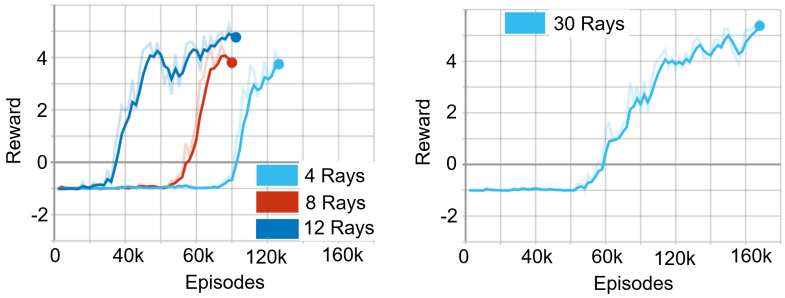


Fig. 6. Cumulative rewards in cases with different reward weights in the learning session; (a) Double and half distance reward weights; (b) Double and half orientation reward weights; (c) Double and half collision reward weights; (d) Double and half time reward weights.



(a) (b)

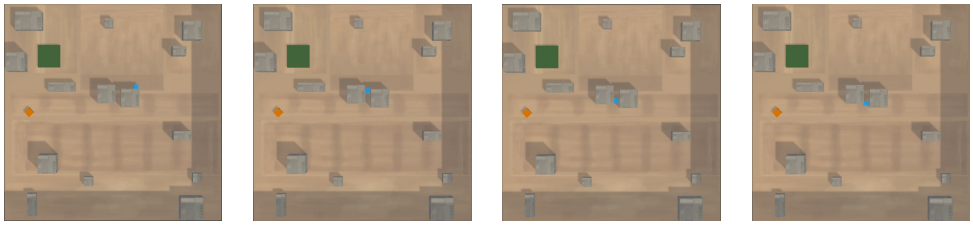
Fig. 7. Rewards v.s. number of rays; (a) Numbers of rays are 4, 8, and 12; (b) Number of rays is 30.

general, an acceptable approach for training the agent was that the convergence occurs at around 40k episodes. In the following experiments, the discrete format was adopted for the output and the orientation constraint was ignored. The setting was: 1) number of rays was 12; 2) the box size was one; and 3) the obstacle size was randomly generated.

6.3.1 Number of rays

The rays attached at the agent represent the perception of the agent about the environment. Each ray is an input. Thus, for a higher number of rays, more inputs are fed to the neural network and the number of the neurons becomes higher. More samples of the environment should be used for training the agent. Therefore, the network takes more number of episodes to converge for higher number of rays. However, if the number of rays is too few, the agent cannot recognize the environment fully. Consequently, the agent cannot make a reliable action when it is at the same position of the environment. This is because the orientation of the agent may be different and the rays cannot capture the surrounding environment consistently. Thus, the agent may not perform well in navigation and the box transportation task. We studied how the agent behaved under the conditions that had different numbers of rays. The numbers of rays were in the set $R_{ray} = \{4, 8, 12, 30\}$.

Denote n_r as the number of rays. Fig. 7 shows the rewards for $n_r \in R_{ray}$. Fig. 7 (a) shows that it takes fewer number of episodes to converge when the number of rays is 12, comparing to the alternatives $\{4, 8\}$. As can be seen, when the number of rays is few, more samples of the environment are required for convergence. Furthermore, the received reward is less for fewer number of rays. For $n_r \in \{4, 8, 12\}$, the rewards and episodes are $(4, 125k)$, $(4, 100k)$, and $(4.5, 102k)$. The agent can push smoothly the box along a curve when $n_r = 12$. In the case $n_r = 4$, Fig. 8 shows that the agent may move along a narrow passage and Fig. 9 shows that the agent does not understand well the relative orientation between the target box and the destination. Fig. 10 (a) shows a case in which the agent is stuck when the target box is at a corner. If n_r is higher than 12, *e.g.*, 30, the agent tends to perform actions in a *push-and-adjust* manner (Fig. 10 (b)). That is that the agent pushes the box and then adjusts its position immediately. After that it adjusts its orientation and then goes on to push the box. The agent fails to push the box continuously along a smooth curve but it tries to push the box along a straight line.



(a) (b) (c) (d)

Fig. 8. The agent may make a poor decision when $n_r = 4$; The agent (the blue block) may pick a narrow passage to move; It often collides with the obstacles.

(a) (b) (c) (d)

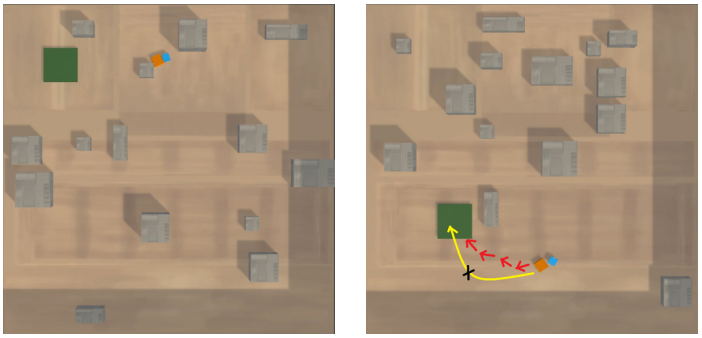
Fig. 9. The agent may not have a "good" understanding about the relative orientation of the target box (orange box) with respect to the destination when $n_r = 4$; In this case, the agent fails to move the box towards the destination directly; (a)-(b) It moves too far away in front of the box; (c)-(d) Then it returns to push the box.

6.3.2 Obstacle size

In an environment, if the obstacles are fixed (*i.e.*, same positions and sizes), the agent tends to move in a similar path from a similar start position to the destination. Thus, the agent has constructed a "mental map" for the environment after it is trained. The agent often moves along similar paths. We set the obstacles at fixed positions but their sizes were changed with the same scale factors. We investigated how the agent performs.

To avoid obstacles that are too small or too large, we set the obstacles with four different sizes. Furthermore, we set the positions of the obstacles so that when the obstacles had the largest scale factor, the gap between adjacent obstacles were large enough for the agent to push the target box. Denote f_{scale} as the scale factor. We have two different conditions. CA) The first condition is that we scale the obstacles with the same scale factor in a random manner. CB) The second condition is that we scale the obstacles individually and each of them has a random scale factor.

Fig. 11 shows that it takes much fewer episodes (60k) to converge in CA, comparing to CB (75k). One reason is that the number of possible combinations of obstacle settings in condition CA is much fewer than those in condition CB. Thus, it is easier to train the agent in condition CA than in condition CB. Fig. 12 shows the results for $f_{scale} \in \{1,4\}$. In both conditions CA and CB, the agent tends to move along similar paths to the destination. The agent performs quite well when the passage is narrow. However, when the gaps between adjacent obstacles become larger, the agent tends to wander around for a while (*i.e.*, a few seconds) before it tries to push the target box.



(a) $n_r = 4$ (b) $n_r = 30$

Fig. 10. Problems when $n_r = 4$ and $n_r = 30$; (a) When $n_r = 4$, the agent cannot push the box around an obstacle corner well. It takes several seconds to push the box away from the corner before reaching to the destination; (b) When $n_r = 30$, the agent tends to perform actions in a *push-and-adjust* manner. After the agent pushes the target box each time, the orientation of the target box may change. Thus, the agent adjusts its position and attempts to push the box straightly towards the destination, *i.e.*, the red dashed curve. When $n_r = 12$, the agent pushes the box along the yellow curve in a smooth manner. The agent does not push the box along the yellow curve when $n_r = 30$.

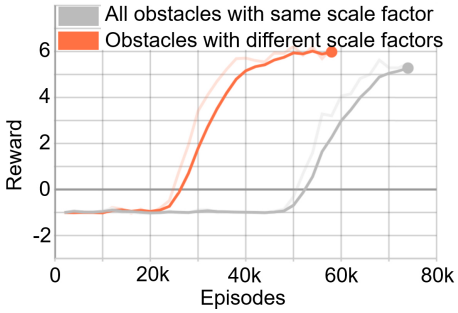
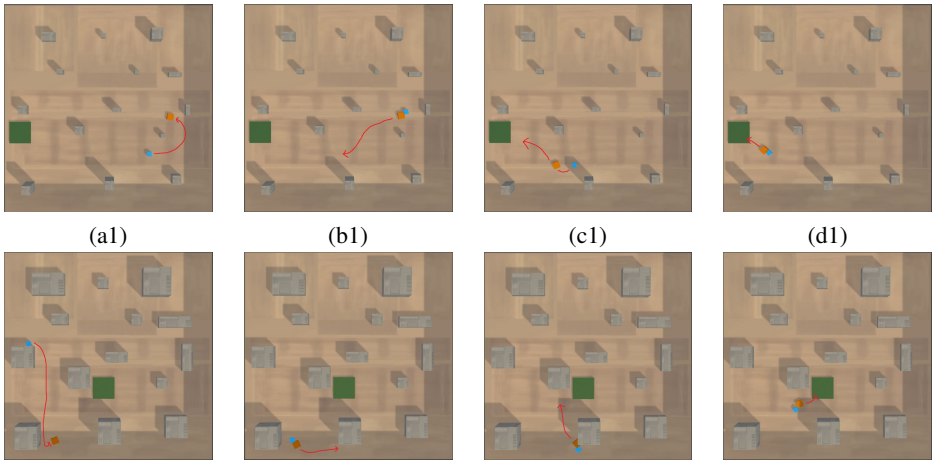


Fig. 11. The reward curves for obstacles with the same scale factors and different scale factors.

6.4 Block Size

The block size affects how the agent pushes the target box around corners of obstacles and navigates along narrow passages. In general, we can train the agent to push the target box with a specific size. However, this requires us to learn one policy for controlling the agent to push the box with a specific size. Thus, for each specific block size, the learnt policy can only be applied to the dedicated block size. Thus, we extended the architecture so that it accepted the block size as an input. Not only that we could reduce the training time but made the agent handle blocks with different sizes. Two conditions are considered as follows. DA: the agent is trained for different specific block sizes. DB: the agent is trained for taking the block size as input. Fig. 13 shows the rewards under condition DA and DB.

In condition DA, we used three different block sizes which were small (LV1), medium (LV6) and large (LV11). Fig. 13 (a) shows that the agent performed the worst in

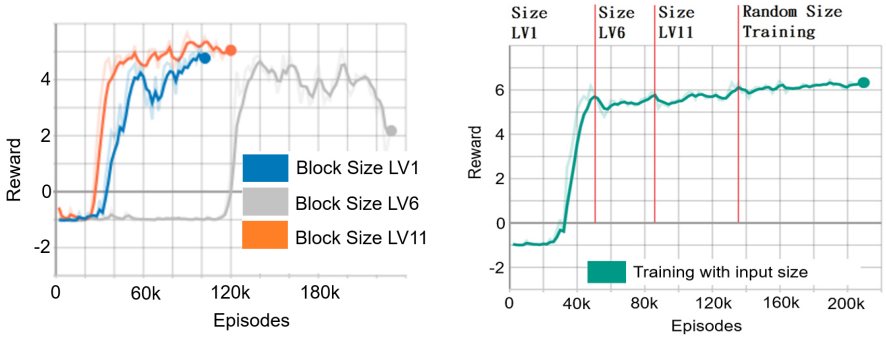


(a1) (b1) (c1) (d1)

(a4) (b4) (c4) (d4)

Fig. 12. Obstacles with different scale factors; (a1)-(d1): Scale factor $f_{scale} = 1$; The agent can push the box smoothly to the destination. (a4)-(d4): $f_{scale} = 4$.

LV6, compared with LV1 and LV11. Not only that it took a large number of episodes (over 220k) to train, but the reward was the poorest. Furthermore, there was a trend that the performance of the agent became worse for training longer. We retrained the policy for several times, we found that there was a probability that the learning session failed. This was because during the learning session, the agent touched the block and pushed it away from the the destination. Subsequently, the penalty was obtained. And thus the agent "believed" in that pushing the box that would result in a penalty. Therefore, the agent avoided moving the box completely. This case could be avoided if the block and the agent could be generated properly. Thus, if the block and the agent were generated properly in the training session for the case LV6, the reward curve was similar to the cases for LV1 and LV11.



(a) (b)

Fig. 13. Rewards for block sizes; (a) The rewards for training the agent with different block sizes; (b) The rewards for training the agent with block size as input.

In condition DB, the block size was treated as an input of the neural network. We trained the agent in the order of LV1, LV6, and L11. Fig. 13 (a) shows the reward curve. The reward curve for LV1 is similar the curve for LV1 in condition DA. After the agent was trained in LV1, we proceeded to train it in LV6 and LV11. As can be seen, the agent could navigate and push the block properly in LV6 and LV11. Unfortunately, after the learning session, the agent could not achieve what we expected when the block size was set to LV1 and LV6. When the block size was changed, the agent confused to perform actions properly. Therefore, we trained the agent in another condition DC. In DC, after the agent was trained in the order of LV1, LV6, and LV11, it was trained in environments with random box sizes (LV1, LV6, and LV11). Eventually the agent learned successfully to manipulate the box with different sizes.

7. CONCLUSION

We implemented a system to learn policies to control agents to navigate and push a box. We studied the effects of reward terms and highlighted the results. The results show that the four reward terms are important. Furthermore, we investigated the performance of the system and the agents for different settings, including the number of rays, obstacle scale factors, and box sizes. The number of rays greatly affects how the agents *perceive* the environment and makes correct decisions. If the box size is treated as an input to the neural network, the learning session should be carefully designed so that the agents can learn the pushing skill properly for boxes at obstacle corners. Our current result suggests that we can train the agents to push boxes with a small scale factor to a large scale factor. In the training session, the agents should be trained in environments in which the box size is randomly generated. There are future avenues. Techniques can be developed to intelligently overcome the collision avoidance problem when dealing with close obstacles and floors with different friction coefficients. We would like to develop techniques for effective object removal and helper modeling in crowd evacuation [27-29] and emotion-based crowd behaviors [30]. We would like to extend our techniques in optimizing paths for crowd simulation [31]. Also, human-like agents can perform stylized actions, *e.g.*, manipulating objects in a good mood or in a bad mood, and moving in a sad, angry, or happy manner. Finally, techniques with deep learning can be investigated to control proactive and reactive agents in collaboration with users [32] in virtual reality. We thanked the anonymous reviewers for their insightful comments. This project was supported by the Ministry of Science and Technology, Taiwan under grant No. MOST 108-2221-E-009-080 and MOST 109-2221-E-009-121-MY3.

APPENDIX

The reward weights are w_1, w_2, w_3 and w_4 which balance the reward terms. The four terms are defined as follows. $r_f = -w_3\Delta t$. $r_c = 0$ if there is no collision; otherwise $r_c = -w_4\Delta t$. $r_d = 0$ if collision occurs; otherwise $r_d = w_1(d^{t-1} - d^t)$. Here, d^{t-1} and d^t are the distances between the target and the goal position at the previous and current

time-steps, respectively. Let ℓ_j be the radius of the object field of target j. We have

$$r_o = \begin{cases} 0 & \text{if collision occurs} \\ w_2\tau & \text{else if } |\phi^t(j,g)| \leq \theta_\alpha \\ w_2\tau|\phi^{t-1}(j,g) - \phi^t(j,g)| & \text{else if } |\phi(\mathbf{h}_j^{t-1}, \mathbf{h}_g)| \geq |\phi(\mathbf{h}_j^t, \mathbf{h}_g)| \\ -w_2\tau|\phi^{t-1}(j,g) - \phi^t(j,g)| & \text{otherwise} \end{cases}$$

where t is the current time; θ_α is a threshold; $\phi^{t-1}(j,g)$ and $\phi^t(j,g)$ are the signed angles between the x-axis of the target and the goal orientation, respectively; $\mathbf{h}_j^{t-1}$ is the x-axis of the target; and $\tau = e^{-\lambda_1\left(\frac{\|d^t\|}{\lambda_2\ell_j}\right)^2}$. λ_1 and λ_2 are parameters.

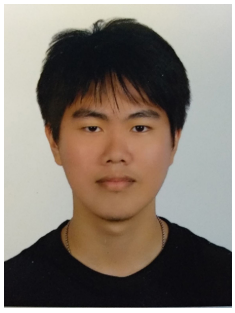
REFERENCES

1. F. Zhang, J. Leitner, M. Milford, B. Upcroft, and P. Corke, "Towards vision-based deep reinforcement learning for robotic motion control," *arXiv preprint*, 2015, arXiv: 1511.03791.
2. P. Stone, R. S. Sutton, and G. Kuhlmann, "Reinforcement learning for robocup soccer keepaway," *Adaptive Behavior*, Vol. 13, 2005, pp. 165-188.
3. N. Heess, S. Sriram, J. Lemmon, *et al.*, "Emergence of locomotion behaviours in rich environments," *arXiv preprint*, 2017, arXiv: 1707.02286.
4. M. Riedmiller, "Neural fitted q iteration-first experiences with a data efficient neural reinforcement learning method," in *Proceedings of European Conference on Machine Learning*, 2005, pp. 317-328.
5. Y. Bengio, A. Courville, and P. Vincent, "Representation learning: A review and new perspectives," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 35, 2013, pp. 1798-1828.
6. K. Arulkumaran, M. P. Deisenroth, M. Brundage, and A. A. Bharath, "A brief survey of deep reinforcement learning," *arXiv preprint*, 2017, arXiv: 1708.05866.
7. S.-K. Wong, Y.-H. Chou, and H.-Y. Yang, "A framework for simulating agent-based cooperative tasks in crowd simulation," in *Proceedings of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*, 2018, pp. 11:1-11:10.
8. H. Chen and S.-K. Wong, "Transporting objects by multiagent cooperation in crowd simulation," *Computer Animation and Virtual Worlds*, Vol. 29, 2018, p. e1826.
9. D. Wolinski, M. C. Lin, and J. Pettré, "Warpdriver: context-aware probabilistic motion prediction for crowd simulation," *ACM Transactions on Graphics*, Vol. 35, 2016, pp. 164:1-164:11.
10. H.-Y. Yang and S.-K. Wong, "Agent-based cooperative animation for box-manipu- lation using reinforcement learning," in *Proceedings of the ACM on Computer Graphics and Interactive Techniques*, Vol. 2, 2019, pp. 2:1-2:18.
11. H. Yang, C. Wong, and S. Wong, "Effects of reward terms in agent-based box-manipulation animation using deep reinforcement learning," in *Proceedings of International Conference on Technologies and Applications of Artificial Intelligence*, 2019, pp. 1-6.

12. P. Charoenkwan, S. Fang, and S. Wong, "A study on genetic algorithm and neural network for implementing mini-games," in *Proceedings of International Conference on Technology and Applications of Artificial Intelligence*, 2010, pp. 158-165.
13. V. Mnih, K. Kavukcuoglu, D. Silver, *et al.*, "Human-level control through deep reinforcement learning," *Nature*, Vol. 518, 2015, p. 529.
14. J. Oh, V. Chockalingam, S. Singh, and H. Lee, "Control of memory, active perception, and action in minecraft," *arXiv preprint*, 2016, arXiv: 1605.09128.
15. X. B. Peng, G. Berseth, K. Yin, and M. Van De Panne, "Deeploco: Dynamic locomotion skills using hierarchical deep reinforcement learning," *ACM Transactions on Graphics*, Vol. 36, 2017, p. 41.
16. X. B. Peng, P. Abbeel, S. Levine, and M. van de Panne, "Deepmimic: Example-guided deep reinforcement learning of physics-based character skills," *arXiv preprint*, 2018, arXiv: 1804.02717.
17. P. Ma, Y. Tian, Z. Pan, B. Ren, and D. Manocha, "Fluid directed rigid body control using deep reinforcement learning," *ACM Transactions on Graphics*, Vol. 37, 2018, pp. 96:1-96:11.
18. A. Clegg, W. Yu, J. Tan, C. K. Liu, and G. Turk, "Learning to dress: Synthesizing human dressing motion via deep reinforcement learning," *ACM Transactions on Graph*, Vol. 37, 2018, pp. 179:1-179:10.
19. J. Won, J. Park, K. Kim, and J. Lee, "How to train your dragon: example-guided control of flapping flight," *ACM Transactions on Graphics*, Vol. 36, 2017, pp. 1-13.
20. J. Lee, J. Won, and J. Lee, "Crowd simulation by deep reinforcement learning," in *Proceedings of the 11th Annual International Conference on Motion, Interaction, and Games*, 2018, pp. 1-7.
21. L. Sun, J. Zhai, and W. Qin, "Crowd navigation in an unknown and dynamic environment based on deep reinforcement learning," *IEEE Access*, Vol. 7, 2019, pp. 109 544-109 554.
22. M. Oshita, "Agent navigation using deep learning with agent space heat map for crowd simulation," *Computer Animation and Virtual Worlds*, Vol. 30, 2019, p. e1878.
23. T. Meriçli, M. Veloso, and H. L. Akın, "Push-manipulation of complex passive mobile objects using experimentally acquired motion models," *Autonomous Robots*, Vol. 38, 2015, pp. 317-329.
24. S. Rodriguez, M. Morales, and N. M. Amato, "Multi-agent push behaviors for large sets of passive objects," in *Proceedings of IEEE International Conference on Intelligent Robots and Systems*, 2016, pp. 4437-4442.
25. J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint*, 2017, arXiv: 1707.06347.
26. A. Juliani, V.-P. Berges, E. Vckay, Y. Gao, H. Henry, M. Mattar, and D. Lange, "Unity: A general platform for intelligent agents," *arXiv preprint*, 2018, arXiv: 1809.02627.
27. S.-K. Wong, Y.-S. Wang, P.-K. Tang, and T.-Y. Tsai, "Optimized evacuation route based on crowd simulation," *Computational Visual Media*, Vol. 3, 2017, pp. 243-261.
28. G.-W. Lin and S.-K. Wong, "Evacuation simulation with consideration of obstacle removal and using game theory," *Physical Review E*, Vol. 97, 2018, p. 062303.

29. J. Kwak, M. H. Lees, W. Cai, and M. E. Ong, "Modeling helping behavior in emergency evacuations using volunteer's dilemma game," in *Proceedings of International Conference on Computational Science*, 2020, pp. 513-523.
30. Y. Mao, Z. Li, Y. Li, and W. He, "Emotion-based diversity crowd behavior simulation in public emergency," *The Visual Computer*, Vol. 35, 2019, pp. 1725-1739.
31. S.-K. Wong, P.-K. Tang, F.-S. Li, Z.-M. Wang, and S.-T. Yu, "Guidance path scheduling using particle swarm optimization in crowd simulation," *Computer Animation and Virtual Worlds*, Vol. 26, pp. 387-395.
32. K.-Y. Liu, M. Volonte, Y.-C. Hsu, S. V. Babu, and S.-K. Wong, "Interaction with proactive and reactive agents in box manipulation tasks in virtual environments," *Computer Animation and Virtual Worlds*, Vol. 30, 2019, p. e1881.

Hsiang-Yu Yang was a master student in the Institute of Computer Science and Engineering at National Yang Ming Chiao Tung University, Hsinchu, Taiwan. His research interests include reinforcement learning, machine learning, and computer animation.



Chien-Chou Wong is a master student in the Institude of Computer Science and Engineering at National Yang Ming Chiao Tung University, Hsinchu, Taiwan. His research interests include computer animation, machine learning, and crowd simulation.

Sai-Keung Wong received the M.Phil. and Ph.D. degrees in Computer Science from the Hong Kong University of Science and Technology in 1999 and 2005, respectively. Since August 2020, he has been a Professor with the Department of Computer Science and the Institute of Multimedia Engineering, National Yang Ming Chiao Tung University, Hsinchu, Taiwan. His research interests include computer graphics, virtual reality, and human-computer interaction. Dr. Wong received the Distinguished Mentor Award in 2011 and the Excellent Teaching Award in 2014.