Efficient Maintenance of Effective NPC Teams in Massively Multiplayer Online Games by Character Binding

Yueguo Chen #1, Beng Chin Ooi #2, M. Tamer Özsu *3, Anthony K. H. Tung #4

#School of Computing, National University of Singapore, Singapore
{1chenyueg, 2ooibc, 4at}@comp.nus.edu.sg

*School of Computer Science, University of Waterloo, Cananda
3tozsu@cs.uwaterloo.ca

***Abstract*— Managing the behavior of a huge number of non-player characters (NPCs) is a major challenge in massively multiplayer online games (MMOGs) for a few reasons. First, the moving characters are highly dynamic and their positioning is highly skewed in swarm regions. Second, players are more intelligent and they often collaborate as teams. Third, the characters are complex by design as they can be armed with different weapons and armors, possess different capabilities, and switch from one role to another. Decision making on the behavior of NPCs in swarm regions is therefore complex because of the rich game semantics in these regions.**

NPCs in MMOGs are also required to act in teams so that they can fight against teamed player characters (PCs) effectively. This leads to the problems of dynamically managing team members and effectively analyzing information of the local game environment, which are real challenges in MMOGs with a large population due to the high computational cost of handling the dynamics of characters. To address these two problems, we propose a data centric character binding model which manages NPCs in teams. Effective NPC teams are dynamically maintained through self-tuning of binding memberships. Information of local game environment is summarized by bindings. The updates of binding information are efficiently handled by applying lazy updates and model-driven updates. We conduct experiments and the results show that our proposed character binding model achieves the much needed effectiveness and scalability desired by game AI in MMOGs.

I. Introduction

Recent years have witnessed the emergence and rapid growth of massively multiplayer online games (MMOGs), especially massively multiplayer online role-playing games (MMORPGs) such as World of Warcraft (WoW) [1], Sims Online [2], and EVE Online [3]. A massively multiplayer online game is a computer game that is supposedly capable of supporting thousands or millions of players simultaneously [1]. It is played on the Internet, and features at least one persistent world [4]. MMOGs are now indeed very popular. For example, the global memberships of WoW exceed 10 million in 2008 [5]. More than 800,000 users play WoW simultaneously during peak hours in China [6], which requires the support of more than 300 clusters of servers.

[1]http://en.wikipedia.org/wiki/MMOG

Currently, most MMOGs are deployed using a client-server system architecture. A player logs into the game world via a client software. He views a local region of the game world and controls the behavior of a player character (PC) via the client, which simply connects with one server (cluster) during a game session. Each server supports a number of online players (e.g., 2-4 thousands of players in WoW) simultaneously. For data consistency, the map of the game world is partitioned into different regions, with each server holding one independent region [7]. Players perform assigned tasks by collaborating and competing with each other in the game. There are also many non-player characters (NPCs) in the game world which have been purposely designed to fight against PCs to make the game more challenging and exciting. Human players accumulate their experience and enhance their power by performing some tasks and fighting with NPCs that are controlled by the AI engine (we will refer to this as the AI player in this paper).

One of the important features in such MMOGs is the behavior of individual NPCs in response to the actions of PCs. This is typically controlled by character scripts [8], [9] dynamically loaded and processed by game engines. What is missing from this, however, is the concept of a team. We take the popular game WoW as an example. One of the major fighting scenes in WoW is that a number of PCs form a troop fighting with one powerful giant. An AI-controlled giant usually retaliates at a PC that fires at it more than the others do. The gaming challenge in such a scenario is that an NPC giant is so powerful that it is impossible for one player to kill it by himself. Consequently, multiple players have to collaborate with each other to win the game. However, the use of a single powerful giant simplifies the game AI which in return reduces the interestingness and possible approaches to the game. As a result, the experience of fighting against giants can be easily diffused, and many players perform tasks by simply following the existing "work around" or trick. The game could be more interesting and less predictive if the PCs are fighting against teams of multiple NPCs that are coordinated by an AI player.

Team AI has been widely studied in computer games [10], [11], [12], [13]. However, there are some challenges to applying existing team AI to MMOGs. First, moving characters are

highly dynamic. They are often spatially correlated in swarm regions, and such correlations may change frequently due to the movements of characters. In such dynamic scenarios, it is costly to continuously guarantee the effectiveness of NPC team members and their fighting strategies. Second, characters in MMOGs are more complex. One character can be armed with various types of weapons and armors of different impactive radius. Different combinations of arms significantly affect the results of fighting, which increases the difficulty of effectively representing and reasoning about the game environment. Third, PCs in MMOGs are continuously controlled by players (different from PCs in real-time strategy (RTS) games). Therefore, each player in the game can feel the effectiveness of game AI at the local region around his role, which enhances the requirements of efficacy of game AI.

Characters (NPCs and PCs) in the game world live and move on a map. Their coordinates on the map are updated accordingly when they are moving. Information of characters (in RTS or MMOG) can be modelled as massive spatio-temporal data [9]. White et al. made a bold step forward in [9] by using database technologies to improve the performance of information aggregation in RTS games, so that the scalability of game population can be improved without sacrificing the expressiveness of game AI. Likewise, we strongly believe that database technologies could be used to help address the above challenges of team AI in MMOGs. In fact, computer game setting is increasingly being fused with social network applications and they present great opportunities for data management. However, it is a challenge to apply existing database technologies such as [14], [15], [16], [17] on spatio-temporal data to MMOGs because game data is not only dynamic, but also complex and coupled (spatially correlated).

In this paper, to effectively manage NPC teams in dynamical game environment, we propose a character binding model in which correlations among characters are modelled as couplings, colludings and bindings. Information of the local game environment is summarized by bindings and shared by characters. A binding layer is developed for high-level team AI to dynamically manage memberships of bindings and plan strategies for NPC teams. We observe that the dynamic maintenance of NPC teams is a kind of data management problem over complex spatial-temporal game data. Database concepts such as self-tuning, lazy updates and model-driven updates can be adapted and applied to effectively maintain bindings and efficiently update information of bindings. The major contributions of this paper include:

- We propose a character binding model to efficiently manage correlations among characters. Situation of characters in the local game environment is modelled and summarized by bindings. NPCs within a binding save the cost of collecting battle information individually as binding information is shared.
- We develop a binding layer providing interfaces for high-level team AI to effectively maintain bindings and plan strategies for NPC teams. Self-tuning techniques are applied to maintain effective bindings. Behaviors of NPCs in a team are highly instructed by the overall fighting strategy of the binding.
- We propose an update scheme in which frequent updates of characters, couplings and colludings are handled either by filtering or lazy updates. All updates are scheduled into different ticks so that the update cost within each tick can be reduced and balanced.
- We have conducted extensive experiments, and the results show that the character binding model achieves both good effectiveness and high efficiency in managing MMOG data.

The rest of the paper is organized as follows. Section II introduces basic components of MMOGs. Section III presents the character binding model. Section IV proposes self-tuning membership management to effectively maintain NPC teams. Section V introduces the update model of bindings. The experimental studies are described in Section VI. Related work and conclusions are given in Section VII and VIII respectively.

II. Preliminaries

A. Characters

MMOGs are highly dynamic due to the frequent movements of characters. In most cases, movements of characters are triggered by those PCs controlled by players. While the movements of NPCs are controlled by the AI player. NPCs move to appointed destinations on the map via path finding (note that many paths are precomputed in computer games). Besides those moving characters, the map also records other static objects such as terrain, trees, rivers and buildings, etc.

Characters in MMOGs can be described by some common features such as key, type, position, health, speed, weapon, armor, observable radius, etc. The most frequently changing feature is the positions of characters because of the movements of characters. The frequent positional updates of characters are a big burden for spatial indexes such as the B^x-tree [18]. Each character in MMOGs has an observable region restricting the map that it can see. Characters and units can be seen by a character p if they fall in its observable region, and there are no obstacles between them and p. Figure 1 shows an example where a PC observes three NPCs within its observable region.

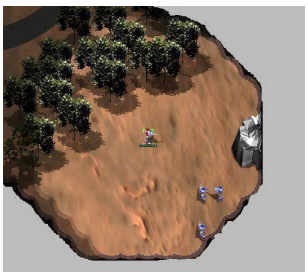


Fig. 1. The observable region of a character

The damage caused by the attack of a character p on a character q, notated as $f(p, q)$, is determined by factors such as the power of p's weapon, the armor of character q, and the distance between p and q. Each character may have a number

of weapons and armors which can be switched according to how the enemies are armed and their positions. The observable regions, the choice of weapons and armors increases the complexity of a game. However, they provide more tricks and capability for players to become more adept at the game.

B. Game Data Management

A character in an MMOG can be treated as a tuple with some attributes, maintained by a character table. Because of the highly dynamic nature of game data, one server is not capable of supporting a huge number of online players. Practically, the game map is partitioned into a number of subregions, and each server (cluster) simply maintains one subregion. Players in that subregion communicate with the server via clients to synchronize the data between clients and the server (illustrated in Figure 2). The major cost of managing game data is incurred at the server side. Each client dynamically subscribes to a region around the PC of that client. Only the updates of characters and units within the subscribed region are sent to the client. The communication cost is thus reduced since the number of characters within the subscribed region is very limited. The visualization is then performed at the client side based on the dynamic game data transmitted from the server and the static rendering data stored locally. The operations of players on the PC are transferred back to the server via the client.

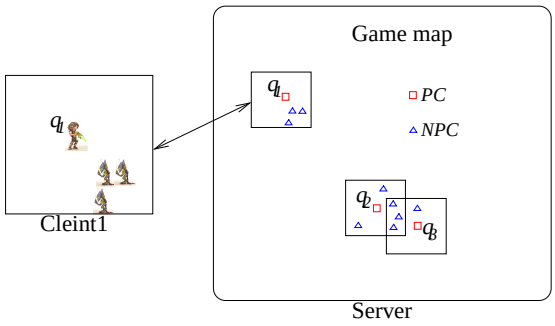


Fig. 2. The subscribed region of players

The evolution of a game's story is entirely developed and computed at the server side. The game's data is actually a form of spatio-temporal data, with the attribute values of characters changing over time. The evolution of characters' attributes is triggered by the actions of characters that are controlled either by human or AI players.

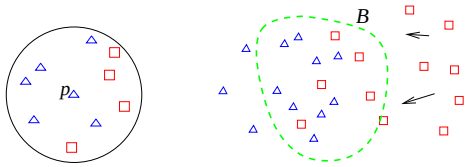
An MMOG server simulates the game one tick at a time. Within each tick, the game engine conducts three tasks [19]: querying the states of the game world, making decision on the behaviors of NPCs, and updating the attributes of characters. The updates within the subscribed region of a player should be notified to the client. Although the states of characters can be interpolated by using optimistic synchronization at the client side, the frequency of game update is important in MMOG simulation. On one hand, update frequency cannot be too low because low update frequency causes the game to lag and may generate data inconsistency between server and client; On the other hand, high update frequency increases the communication cost between server and client, which may result in large latency between server and client. Based on [20], existing game engines typically have an update rate of 3 to 10 ticks per second. As such, the small interval between two consecutive ticks is a challenge for game AI to achieve expressive behaviors on thousands of NPCs.

C. Game AI

Modern computer games with a large population prefer a game architecture called the data-driven game architecture [8], [9], [21], which separates the contents and codes of games. Contents of characters and objects, as well as behaviors of characters can be defined and stored in script files. The game engine simulates the game by loading and processing script files dynamically. Such design effectively isolates the workload of game programmers and that of game designers. It allows the contents of the games to be dynamically modified during the running time of games.

In the data-driven game architecture, the game AI of characters is presented as declarative rules and goals in character scripts [8]. Specific behaviors of characters are triggered when some conditions of rules are satisfied. The major computational cost of game AI is to check the rule conditions which are derived from the distribution of characters and objects in local game environment.

A simple case of game AI is the individual game AI in which each character is autonomic. An NPC makes its decision simply based on its own observation. However, the lack of cooperation decreases the efficacy of individual game AI. For example, an NPC p in Figure 3(a) may attack one of observed PCs when it sees that more NPCs exist around it than PCs. While, the other NPCs may choose to escape or stand by, leaving it to fight alone. Note that PCs are represented as squares, and NPCs are represented as triangles throughout the figures of this paper.



(a) Individual AI (b) Team AI

Fig. 3. Reasoning about the local game environment

Team game AI [10], [11], [12], [13] can obviously achieve competitive and cooperative behaviors for NPCs. Hierarchical planning [10], [11], [22] is preferred in team AI in which the behaviors of low-level characters are guided by the commands of high-level squads or groups. Compared to individual game AI, centralized team AI has a larger view of the battle situation. Therefore, it can plan the action of a group as a whole. It simplifies decision making by allowing complex team behaviors with less hassle [10].

D. The Challenge: AI for Dynamic Team Membership

Team AI creates and executes plans based on the states of team members, assuming that team membership seldom changes. However, the situation of battlefield in MMOGs is highly dynamic. The neighborhood of characters may change frequently due to the movements of characters. As a result, the membership of a team should be dynamically adjusted to keep the effectiveness of team AI strategies. Figure 3(b) shows an example where an optimized team can be ineffective after some movements of characters.

One simple way of team membership management is to organize teams through clustering, in which characters within one local cluster form a team. However, local fights cannot be well balanced in the clustering approach because the number of NPCs and PCs in a team cannot be freely controlled. To our knowledge, the existing studies on team AI [10], [11], [12], [13] ignore team membership management, which is actually very complex because the clear scope of teams is hard to be identified in dynamic game environment.

Complex characters of MMOGs also pose challenges for reasoning about information of the game environment observed by teams. The basic aggregate functions in SGL [9] cannot be applied to aggregate mutual impact between complex characters. All these challenges motivate us to propose the character binding model for enhancing AI in MMOGs.

III. A Character Binding Model

We propose a character binding model to address the two challenges of team AI in MMOGs: 1) maintaining effective granularity and coverage of teams in the battlefield; 2) collecting and analyzing team information within the local environment.

A. Couplings and Bindings

In MMOGs, individual game AI basically plans the behavior of an NPC p based on its own observation. The observable relationship affects the response of characters because many fighting strategies are triggered when characters can "see" their opponents. We use the definitions of couplings, colludings and bindings to formalize the observable relationship of characters among MMOGs.

Definition 1 (Coupling): When an NPC p sees a PC q, we say there is a coupling, p couples q, notated as $p \triangleright q$.

A coupling $p \triangleright q$ means that p and q are close. The NPC p should continuously monitor the status of PC q because q has threatened p. Special behaviors (e.g., fighting) of p may be triggered when it observes some events happening on q through a coupling $p \triangleright q$. The coupling $p \triangleright q$ is released when one of the coupled character is dead or p cannot see q any more.

Since an NPC p can see multiple PCs within its observable region, it can then be coupled with multiple PCs simultaneously. Among all the coupled PCs, an NPC p can choose at most one PC q as its target, which is notated as $p \rightarrow q$ and $q = p.target$. An NPC p treats its target as its direct opponent to fight with. Therefore, compared to the other coupled PCs, p pays more attention on its target, to which more complex analysis is conducted. Note that $p \rightarrow q$ also means that $p \triangleright q$. However, $p \rightarrow q$ does not requires that p has to see q. An NPC p can also target on a PC q outside its observable region by obtaining information of that PC from other team members as a kind of cooperation. We say that the NPC p is active (notated as $\hat{p}$) if it has a target; Otherwise, p is inactive (notated as $\check{p}$).

An NPC p observes not only PCs but also NPCs within its observable region. The collaboration among NPCs is also based on the observable relationships of NPCs. Therefore, we give the following definition to formalize the possible cooperative relationships among NPCs.

Definition 2 (Colluding): Two NPCs p_1 and p_2 are colluding, notated as $p_1 \diamond p_2$, if p_1 sees p_2 or p_2 sees p_1, or there is a third-party NPC p_3 such that $p_3 \diamond p_1$ and $p_3 \diamond p_2$.

A colluding $p_1 \diamond p_2$ means that the two NPCs p_1 and p_2 are contactable either directly or indirectly through some intermittent NPCs. NPCs p_1 and p_2 should not be too far away from each other (within a cooperative distance) if they are colluding. Therefore, they probably become allies and form a team to fight against their enemies together.

Definition 3 (Binding): A binding $B = P \triangleright Q$ consists of a set of NPCs P and PCs Q such that: 1) $\forall p \in P, \hat{p}$; 2) $\forall p_1, p_2 \in P$, $p_1 \diamond p_2$; 3) $Q = \cup_{p \in P, p \triangleright q} q$; 4) $P \neq \emptyset$ and $Q \neq \emptyset$.

Therefore, a binding is constructed by a team of active NPCs who are colluding with each other. All those PCs coupled with any NPC in the binding are also included by the binding, as the enemies of the NPCs in the binding. The binding clearly models the scope of an NPC team. Based on this, the NPCs in a binding can collaboratively fight against those PCs in the same binding.

Figure 4 gives an example of couplings, colludings and bindngs. Note that although the NPC p_3 is colluding with p_1 (directly) and p_2 (indirectly), it is not in B_1 because it is inactive. An inactive NPC will not fight with the coupled PCs until it transfers to active status by choosing a surrounding PC as its target. Those NPCs not colluding with each other cannot form a binding. Therefore, they cannot cooperate with each other as the NPCs within a binding.

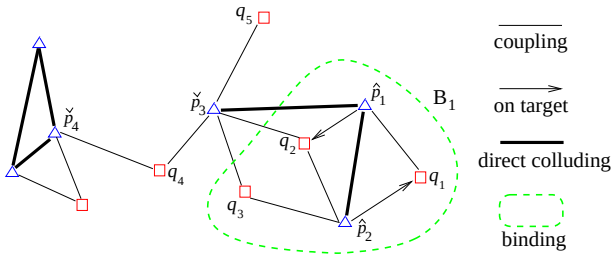


Fig. 4. Illustration of couplings, colludings and bindngs

B. Architecture of The Binding Model

The binding model brings some benefits for game AI of MMOGs. First, correlations (couplings and colludings) of neighboring characters are managed by bindings from which battle information of local regions can be efficiently summarized, analyzed and shared to individual AI of NPCs. Second, effective NPC teams can be maintained through dynamical binding membership management. Third, a binding manages the updates of characters, couplings and colludings within it. Those unimportant updates are either postponed or suppressed by bindings, without disturbing the high-level team AI.

To seamlessly apply the binding model to high-level team AI, we develop a binding layer to provide interfaces for high-level team AI so that NPC teams can be effectively controlled through binding management. The architecture of binding management is shown in Figure 5 Unlike the traditional game engines where the team AI directly cooperate with individual AI, the binding layer is abstracted as an intermediate layer between the high-level game AI and the low-level characters in our proposed architecture. Bindings can be viewed as the summarization or index of some local characters. They are abstracted and maintained within the binding layer.

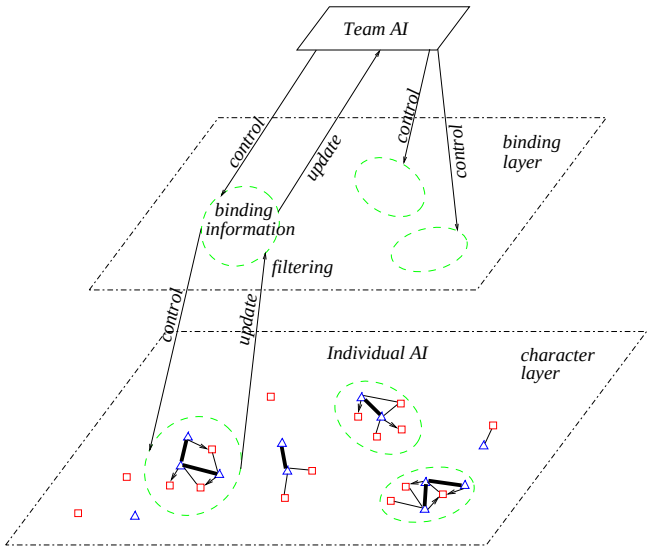


Fig. 5. The binding management architecture

The team AI collects information of the local game environment through bindings. The high-level decisions of team AI are conducted over bindings as the overall strategies of bindings. The behavior of NPCs within a binding is instructed by the overall strategy of the binding. There is a feedback-loop between the binding layer and the character layer. Each binding collects local battle information through managing the updates of characters, couplings and colludings while coordinating the behaviors of NPCs within it. The binding model can provide an effective way for enhancing the AI of NPCs in MMOGs.

C. Information Maintained by Bindings

The basic information maintained by a binding is the memberships of NPCs and PCs belonging to the binding. The coupling-ships and colluding-ships of characters can be updated frequently due to the movements of characters. This may further result in the creation or release of bindings, the join or leave of characters to a binding, the union or split of bindings, etc. The superiority or inferiority of NPCs to PCs within a binding are highly affected by the memberships of the binding. Therefore, dynamically maintaining effective binding memberships is the most important task of the binding model. We will introduce self-tuning techniques for maintaining effective bindings in Section IV.

The situation of NPC teams has to be modelled and analyzed based on the observation of the local game environment, to describe the superiority or inferiority between NPCs and PCs. The battle information is widely distributed throughout the map where characters exist, particularly more prominently in the local regions where large number of characters congregated. To effectively capture battle information of local game environment, a binding has to aggregate information such as fighting power, overall damage, number of enemies, the emergent NPCs, over those characters of the binding. All these information forms the basis of decision making of the AI player. The overall fighting strategy of a binding is just made by team AI based on those aggregated information. A lot of aggregating values generate challenges for efficient updates of bindings. We will introduce updating strategies of the binding model in Section V.

There may be hundreds to thousands of bindings handled by a server maintaining a partition of the map in MMOGs. An important task of the binding layer is to efficiently manage those bindings so that the additional cost incurred by the binding layer is not prohibitive.

D. How Does Individual AI Benefit from Bindings?

The AI player plans the overall fighting strategies of bindings based on the overall game logic (centralized AI scripts, as a means of high-level game design) and the aggregated information of bindings. The behavior of an NPC are partially controlled by the individual game AI through its own observation. However, it is strongly instructed by binding information when the NPC is within the binding or it observes some NPCs of some bindings. The individual AI of many NPCs can quickly evaluate the situation of local environment (much larger than their own observable regions) by accessing the shared binding information which summarizes battle information of regions covered by NPCs of the binding.

The complexity of collaboration among individual game AI is reduced by the guidance of high-level binding strategies. There are a number of overall fighting strategies applied by bindings such as, approach, attack, besiege, defense, retreat, allure and follow. Behaviors of NPCs in a binding are strongly guided by the overall fighting strategy of the binding, so that the individual strategies of NPCs in a binding can be consistent. For example, in an “attack” strategy, an NPC p

chooses a PC to which p has the largest coupling score as its target to fight against. In the meanwhile, p will try to avoid those PCs to which it has the negative couplings. The position of the target PC can be quickly obtained by p from its coupling, without accessing spatial index. The NPC p continuously monitors the states of its coupled PCs and the binding information to dynamically adjust its behavior, taking the advantages of the binding model.

IV. Self-Tunable Binding Management

The relative number of NPCs to PCs within a binding is quite important because it affects the results of fighting between NPCs and PCs significantly. The relative healths and fighting powers of characters can also affect the results of fighting in a binding. Effective team AI of NPCs should guarantee that there is enough fighting strength of NPCs in a team so that the team has enough chance to win in the local fighting with some PCs. However, the fighting strength of NPCs cannot be too large because too much NPC fighting strength means that PCs have little chance to win. Players may feel hopeless and lose interests with the game if they often fight with NPC teams much stronger than their own. However, maintaining a proper number of NPCs in bindings is not easy because of the highly dynamic aspects of MMOGs.

We propose to apply the self-tuning mechanism to maintain effective bindings in MMOGs. Self-tuning techniques have been widely applied in database systems [23] to always automatically keep the performance of databases in optimal states. To achieve self-tuning on binding management, the effectiveness of bindings must be well capture. We propose a statistic called binding score to evaluate the relative fighting strength of NPCs to PCs within bindings. Based on the binding score, the membership of a binding can be dynamically adjusted through the joining and leaving of characters, as well as the union and split of bindings.

A. The Criterion of Binding Tuning

In computer games, the effectiveness of NPCs is typically collected through influence map [13] and spatial aggregates [9], in which the distribution of common attributes such as damage, defence and health of characters are aggregated. However, characters in MMOGs are more complex. Each character may be weaker than some characters, but stronger than some other characters. The mutual impact between characters has to be considered as well.

In the following, we present a case study of formalization of battle information from mutual impact of characters, to effectively evaluate the effectiveness of couplings and bindings. Although different formalizations can be applied in different games, they follow the same philosophy of semantics summarization.

1) Coupling score: Given a coupling $p \triangleright q$, the fighting strengths of the NPC p and the PC q are computed as $s(p,q) = f(p,q) \cdot p.h$, $s(q,p) = f(q,p) \cdot q.h$ respectively, where $p.h$ and $q.h$ are the health of two characters. We define the coupling score $g(p \triangleright q)$ to evaluate the relative fighting strength of the two characters:

Definition 4 (Coupling score): The score of a coupling $p \triangleright q$ is $g(p \triangleright q) = s(p,q) - s(q,p)$.

The coupling score $g(p \triangleright q)$ measures the superiority of p to q if they are fighting with each other. The larger the attribute values (health, damage and defence) of p to those of q, the higher the $g(p \triangleright q)$. A coupling $p \triangleright q$ is a weak coupling, notated as $p \sim q$, if $|g(p \triangleright q)| \leq \theta$. Parameter θ is a positive threshold used to filter weak couplings over coupling score. A weak coupling $p \sim q$ means that the NPC p is comparative to the PC q, i.e., neither p nor q are overwhelming to the other.

In contrast, a coupling $p \triangleright q$ is a strong coupling if $|g(p \triangleright q)| > \theta$. Strong couplings can be further classified into positive couplings and negative couplings. A coupling $p \triangleright q$ is a negative coupling, notated as $p \succ q$, if $g(p \triangleright q) < -\theta$. The NPC p is inferior to the PC q when $p \succ q$. A coupling $p \triangleright q$ is a positive coupling, notated as $p \prec q$, if $g(p \triangleright q) > \theta$. The NPC p is superior to the PC q when $p \prec q$. The coupling score $g(p \triangleright q) = -\infty$ means that p has no threat to q while $g(p \triangleright q) = +\infty$ means that q has no threat to p. Strong couplings are more important than weak couplings because they give very good indication on the results of the fighting between p and q.

Coupling scores are very useful in guiding the behaviors of NPC characters. An NPC p should evade a PC q if $p \succ q$ as it is weaker than q. On the other hand, the NPC p can attack q if $p \prec q$ because it has enough confidence to win the fight.

2) Binding score: Given an NPC p in a binding $B = P \triangleright Q$, the fighting strength of p is defined as $s(p) = f(p, p.target) \cdot p.h$. Given a PC q in B, the fighting strength of q is defined as $s(q) = f(q,p') \cdot q.h$, where p' is the NPC which q is firing at or the nearest enemy of q. We then define the binding score to evaluate the relative fighting strength of NPCs and PCs within a binding:

Definition 5 (Binding score): The score of a binding $P \triangleright Q$ is $g(P \triangleright Q) = \frac{\sum_{p \in P} s(p) - \sum_{q \in Q} s(q)}{|Q|}$.

Similar to the couplings, a binding $P \triangleright Q$ is a weak binding, notated as $P \sim Q$ if $|g(P \triangleright Q)| \leq \theta$; It is a positive binding, notated as $P \prec Q$, if $g(P \triangleright Q) > \theta$; Otherwise, $g(P \triangleright Q) < -\theta$, $P \triangleright Q$ is a negative binding ($P \succ Q$). The binding score can be computed as an aggregation of fighting strength of characters in the binding. As an effective measure of battle situation between NPCs and PCs, the binding score helps the AI player to determine the overall fighting strategies of bindings and adjust the membership of NPCs in bindings. Therefore, we choose the binding score as the criterion of binding tuning.

B. Creation and Release of Bindings

An inactive NPC does not belongs to any bindings even though it is coupled with some PCs or colluded with some NPCs. A new binding is created when an inactive NPC p targets on an PC q, i.e., $p \rightarrow q$. Characters p and q, together with those PCs coupled with p form a new binding. Figure 6

uses the example of Figure 4 to show the creation of a binding B_2, which is triggered by the NPC p_3 when it decides to target on a coupled PC q_5. Note that those PCs that belong to some other bindings (e.g., q_2 and q_3) can also be included in the new binding as long as they are coupled with p_3. This means that a PC can span over multiple bindings. However, an NPC can only belong to one binding. In this example, the NPC p_3 is colluding with those NPCs in B_1. However, it does not join the binding B_1 at the time B_2 is created. The two bindings may be combined later when one of them needs to be enlarged due to the lack of NPC fighting strength.

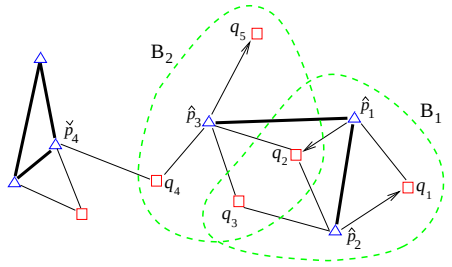


Fig. 6. An example of binding creation

A binding is released when there are no NPCs or PCs in the binding. It happens when all NPCs (or PCs) are killed or all NPCs release their targets in the binding. There will be no active NPCs when the binding is released. Behaviors of the NPCs (if exist) in the released binding will only response to their own observation before they join some other bindings.

C. Join and Leave of Characters in Bindings

The effectiveness of a binding is tuned according to the binding score, which is periodically updated by the binding. The goal of binding tuning is to tune the binding score by adjusting the membership of NPCs in the binding. A common way of binding membership adjustment is the join or leave of characters. When the binding score is too low, especially for a negative binding, the binding enlargement process will be triggered. Those inactive NPCs who are colluding with NPCs of the binding have a large probability to be recruited as new members of the binding.

A binding B to be enlarged will explicitly show its request as a kind of binding information. When some inactive NPC p directly colludes with an NPC of the binding B holding an enlargement request, the individual AI of p will process this request immediately, determining whether to apply to join the binding B based on its own status. The binding B maintains a list of all applicants who are willing to join it. It selects some (or all) applicants when it comes to the processing tick of the binding, using the binding enlargement algorithm shown in Algorithm 1.

The binding enlargement is processed as follows. All the NPCs in the applicant list are ranked based on their potential benefits to the binding B. Those NPCs with higher benefits are first recruited as new members of the binding B. The enlargement process terminates when $g(B)$ reaches α or the number

Algorithm 1 Binding enlargement

Input: B, a binding holding an enlargement request.
Input: $applist$, the list of NPCs applying to join B within the past processing period of B.
Input: α, the binding score of B to be achieved.
Input: γ, the maximal number of NPCs contianed by B.

1. **for all** $p \in applist$ **do**
2. $benefit(p) = \max_{p \triangleright q} s(p,q) - \sum_{p \triangleright q \& q \notin B.Q} s(q,p)$
3. **if** $benefit(p) > 0$ **then**
4. $heap.insert(benefit(p), p)$
5. **while** $g(B) < \alpha$ and $|B.P| < \gamma$ and $heap.size() > 0$ **do**
6. $p = heap.extractMax()$
7. insert p and those PCs ($p \triangleright q$ & $q \notin B.Q$) into B
8. $q' = findTarget(p, B.Q)$ //find an optimal target.
9. set $p \rightarrow q'$
10. **if** $g(B) \geq \alpha$ **then**
11. remove the enlargement request from B
12. break

of NPCs in B is large enough. Otherwise, the enlargement request still remains by the binding B. An example of binding enlargement is shown in Figure 7. The binding B is enlarged to the binding B' by recruiting p_3 and p_4 in. Note that p_5 is not recruited because it does not benefit enough. If the binding score of B' is still less than α, the other NPCs (e.g., p_6 and p_7) around B' still have the chance to join B' in the next processing round of the enlargement. In this way, a binding is enlarged ring by ring.

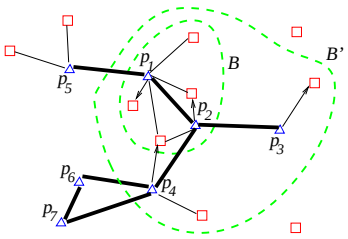


Fig. 7. An example of binding enlargement

On the other hand, when the binding score of a binding B is too large, there will be too many NPCs in B. They are attracted by a small number of PCs in the binding. This may not be good because too many redundant NPCs in one binding means some neighboring bindings are running short of NPCs with a high probability. To avoid this, the NPC-overloaded binding can initiate a binding reduction process to reduce the number of NPCs in the binding by releasing the targets of some NPCs. The NPCs to be removed from the binding are chosen mainly based on their positions and fighting strength. Those weak NPCs are usually first chosen to leave from the binding. The process of binding reduction is shown in Algorithm 2.

D. Inter-Binding Cooperation

The distribution of NPCs in the map of MMOGs are often very skewed. It is common that NPCs in some bindings are

Algorithm 2 Binding reduction

Input: B, a binding to be reduced.
Input: α, the binding score of B to be achieved.
Input: γ, the minimal number of NPCs contianed by B.

1. **while** $g(B) > \alpha$ and $|B.P| > \gamma$ **do**
2. $p = chooseNPCToRemove(B.P)$
3. $B.remove(p)$
4. **for all** q, such that $p \triangleright q$ **do**
5. **if** $\nexists p' \in B.P$, such that $p' \triangleright q$ **then**
6. $B.remove(q)$

redundant, while those neighboring bindings are running short of NPCs. A negative binding can pursue help from NPCs in the neighboring positive bindings. Cooperation among NPCs across multiple bindings can be achieved through the union and split of bindings. The centralized game AI controls the collaboration of bindings as it has an overview of all bindings.

1) Binding Union: A binding can request a union operation when it cannot get enough NPC fighting strength through binding enlargement. When an active NPC of one binding is colluding with an active NPC of another binding, the binding union process may be triggered if there is one binding holding a union request. The distribution of NPC fighting power can be tuned by combining a positive binding with the binding requests union operation. As a result of the binding union, all binding information, as well as the binding score need to be updated. A new fighting strategy will be generated to guide all the NPCs in the new binding.

2) Binding Split: With the evolution of game's story, some bindings may grow to be too large and the colludings between some NPCs within the bindings may be broken. The covering region of a binding can be enlarged significantly due to the growth of binding and the diffused movements of characters. As a result, characters in the binding may be separated into several clusters. The NPCs in different clusters may be not colluded with each other. The bindings will not be good enough to summarize information of local game environment.

When the number of NPCs are large enough, the binding validation process needs to be triggered to periodically check whether all NPCs in the binding are colluding with each other, and whether each PC is coupled with an NPC in the binding. Effective bindings can be tuned by splitting an invalid binding into multiple valid bindings. The binding validation can be done by traversing the spanning tree over the colluding graph of NPCs within the binding. We propose the Algorithm 3 for checking the binding validity and splitting invalid bindings simply using the colludings and couplings of NPCs. All colluded NPCs (expanded through the same spanning tree), together with those PCs coupled with them, form a binding. There will be multiple split bindings if multiple spanning trees exist in the colluding graph of the original binding B.

Note that a binding can also be forced to split by binding reduction, i.e., forcing some NPCs to release their target PCs in the binding. These NPCs together with some coupled PCs are then removed from the original binding. As a result, the

Algorithm 3 Binding validation and split

Input: B, a binding to be validated, where $B.P = \{p_1, \dots p_m\}$, $B.Q = \{q_1, \dots, q_n\}$
Output: $\{B_k\}$, a number of bindings generated from B.

1. **for** $i = 1 : m$ **do**
2. $p_i.cluster = 0$
3. $k = 0$ //identify the split binding
4. **while** $B.P \neq \emptyset$ **do**
5. $k = ++$
6. $toExpandNPCs.clear()$
7. $p_t = pop(B.P)$
8. $p_t.cluster = k$
9. $toExpandNPCs.push(p_t)$
10. **while** $toExpandNPCs.hasElements()$ **do**
11. $p_s = toExpandNPCs.pop()$
12. **for all** p_r, such that $p_s \diamond p_r$ **do**
13. **if** $p_r.cluster < p_s.cluster$ **then**
14. $toExpandNPCs.push(p_r)$
15. $B.P.remove(p_r)$
16. $p_r.cluster = k$
17. $B_k.P.push(p_s)$
18. **for all** q_j, such that $p_s \triangleright q_j$ **do**
19. $B_k.Q.push(q_j)$

colluding graph of the original binding may be split into a number of sub-graphs, from which new bindings are created. Choosing appropriate characters to leave from bindings is quite complex and application dependent. It is beyond the scope of this paper. However, it is also a kind of tuning operations of bindings.

V. Binding Updates

The effective decision of team AI is based on collecting and reasoning about information of game environment in the game space. Characters are often moving and fighting in the game space, which generate a huge number of updates of characters, couplings and colludings when the population of the game is large. Collecting binding information in the whole battle field will be very expensive if every update needs to be processed immediately. However, the overall fighting strategy cannot be frequently updated in practice. Not all the updates are important for decision making of the game AI. Many of them can be delayed or ignored, without affecting the accuracy of binding information too much. We have three update schemas to be applied by various kinds of information updates.

A. Eager Updates

Most of attributes of characters are maintained in a character table with each character as a tuple. The updates of these attributes are directly conducted on the plain table, which do not consume too much computational cost. In the meanwhile, the high-level bindings processing those binding aggregate information incrementally, as a byproduct of attribute updates. The overall fighting strategy of a binding can be adjusted

by team AI when some events on the aggregated values are triggered.

Unlike the character attributes, couplings and colludings of NPCs have to be maintained by dynamic data structure because the numbers of coupled PCs and colluded NPCs of an NPC are not fixed. An NPC continuously maintains and monitors information of all its couplings because the couplings provide important local information. The updates of PCs are directly propagated to the coupled NPC without delay so that individual AI of NPCs can immediately response to the behaviors of PCs.

B. Lazy Positional Updates

The frequent positional updates of moving characters result in the expensive update cost of the spatial index and the observable relationship of characters. The spatial index of moving characters has to be maintained because the observable and neighboring relationships of characters have to be done through range queries over the index. The update cost of thousands of moving characters is quite expensive, which may take the major computational cost of the game systems if it is done naïvely.

To avoid the frequent positional updates of moving characters, we apply the lazy update strategy. The position of a character is approximated by a covering bound surrounding its last updating position. The spatial index is built over the covering bounds of all characters (Figure 8 gives an example). To reduce the updates, the covering bound of a character is updated only when the character moves out of its covering bound. However, such a lazy update strategy generates errors in measuring the observable relationship of characters because the positions of moving characters in the index is uncertain within the bounds. This is handled by the traditional filter-and-refine process in spatial databases. On monitoring the observable relationships of characters, another spatial index is built recording the view bounds of NPCs. In the filtering phase, each NPC maintains a monitoring list of all characters whose covering bounds overlaps with its view bounding box. In the refine phase, the observable relationships of all the characters in the monitoring list of an NPC are checked based on their actual positions.

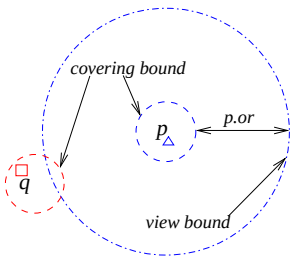


Fig. 8. The covering bounds and view bounds of moving characters

The monitoring list of a character p is incremented only when the lazy update of some characters (including p) results in the occurrence of overlap between the covering bounds of the other characters and the view bound of p. Note that the lazy update scheme is applicable to any spatial indices such as B^x-tree [18].

C. Sampled Updates

Bindings handle the updates of low-level characters, couplings and colludings in two ways. One is eager updates (push-based updates) in which low-level characters, couplings and colludings actively push important updates to the high-level binding without delay. The other is sampled updates (pull-based updates), in which bindings request updates from low-level characters, couplings and colludings periodically. Those unimportant low-level updates can be delayed or ignored.

Normally, the overall fighting strategy cannot be updated frequently as the cost of strategy jump (e.g., NPCs come and go) is very expensive. Consequently, information maintained by bindings usually does not need to be updated frequently. Similar to the model driven [24] approach applied in sensor network monitoring, a binding requests updates from the low-level when the high-level information is not accurate enough. A binding can adjust the update cost by controlling the number and the granularity (frequency) of binding aggregate information based on its status. Fine granularity aggregates are required only when a binding is in an emergent status.

In our update model, each sampled update has an update period which can be adjusted based on the changing rate of the monitored value. All updates are randomly scheduled into different ticks so that the overall update cost of ticks can be evenly distributed. A sampled update is processed only within its updating ticks.

VI. Experiments

There are quite a few open source MMORPG engines such as WorldForge [25]. However, they emphasize many aspects of MMOG engine such as client-server architecture, network protocols, cache of media, graphic simulation, etc. While our problem is restricted to enhancing game AI in the server side. To avoid the impacts of the other factors, we simply test our idea by developing a binding based preliminary MMOG engine of the server side. We test all the experiments on a PC of Pentium4 3.0G CPU with 1G RAM. Since there is no public dataset of user operations in MMOGs, we simulate some game scenarios by randomly generating characters and distributing characters following some patterns in the game space. There are hundreds of different kinds of weapons and armors. Each character holds 5 weapons and 5 armors chosen randomly. The average health and fighting strength of NPCs and PCs are relatively equal. The total number of NPCs and that of PCs are the same at the beginning.

The density of characters affects the couplings among characters, and therefore affects the efficiency of game simulation. In our experiments, we fix the average density of characters in each group of tests when testing the scalability. In the experiments, the average view radius of characters is set as 80m. While the average speed of characters is 10m/s. The simulation frequency is set as $n = 5$. Therefore, the time interval between two consecutive ticks is 0.2 seconds.

A. Effectiveness of the Binding Model

To show the effectiveness of the proposed game engine, we build a graphic interface so that the status of characters in any local region can be interactively monitored. A snapshot of a local map of our game engine is shown in Figure 9, where a binding is created from a number of PCs and NPCs. The membership and health of characters in bindings can be continuously monitored.

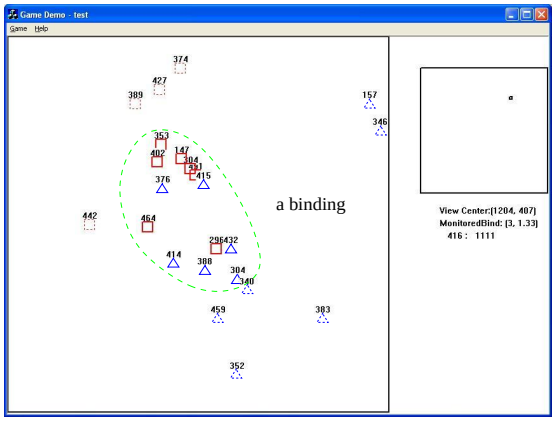


Fig. 9. The interface of a binding based MMOG

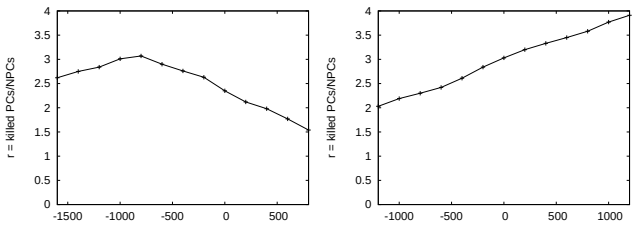
It is difficult to simulate thousands of PCs as those PCs controlled by human players will be more intelligent than the simulated PCs. However, the group behaviors of PCs can be simulated as people like to collaborate with some other players in real MMOGs. In our implementation, PCs are simulated by grouping them into clusters, with each cluster holding dozens of PCs at the beginning. PCs within one cluster follows the similar moving patterns so that they always keep close to each other. A PC chooses its target based on the coupling scores of the NPCs within its observable region. All the PCs within one cluster will stop to fight against NPCs when enough number of NPCs are detected.

We also randomly generate NPCs in clusters (which are relatively smaller than clusters of PCs in average), with 1 to 20 NPCs in each clusters. Those NPCs keep static if no PCs are observed by them. We implement the AI strategy of NPCs using two models: one is the proposed binding model; the other is a basic model as a straw man of the binding model. In the basic model, like those PCs, an NPC simply monitor its coupled PCs and made decision based on individual AI. It simply fights with the weakest opponent (in a non negative coupling) within its observable region, without collecting any other local battle information beyond its observable region. Obviously, there is no collaboration among NPCs in the basic model.

We compare the effectiveness and efficiency of the two models in Figure 10. In this experiment, the total number of NPCs and PCs are both 10,000 at the beginning. The efficacy of models is measured based on the relative number of killed PCs and NPCs ($r = \frac{killed\ PCs}{killed\ NPCs}$) with the evolution of game. The basic model achieves around $r = 0.47$ (Figure 10(b)), which means that one PC is relatively equal to two NPCs in fighting strength. This is because PCs have the advantage of larger population in local regions. While, the binding model achieves around $r = 3.20$, which means that one NPC is relatively equal to more than three PCs. Therefore, the results show that NPCs in the binding model are far more competitive than those in the basic model. This is achieved by local battle information analysis and NPC cooperation under the binding model.

There are two important parameters in the binding model, θ and α, which affect the efficacy of AI players significantly. We show the impaction of these two parameters in Figure 11. The parameter θ is important in identifying weak couplings and weak bindings so that NPCs can avoid from the attack of strong PCs. However, θ cannot be too large because large θ causes too many NPCs to be coward, and there will be not enough fighting NPCs involving in the war. On the other hand, θ cannot be too small as NPCs will not be aware of strong enemies when θ is small. We choose θ as -800 which achieves good efficacy in Figure 11.

Parameter α (used in Algorithm 1 and 2) determines the overall fighting strengths of NPCs triggered to fight with detected PCs. The larger the α, the higher fighting strength of NPCs within a binding, and the better efficacy can be achieved. In most of our experiments, we choose $\alpha = 200$, in which we got $r = 3.20$ in average.



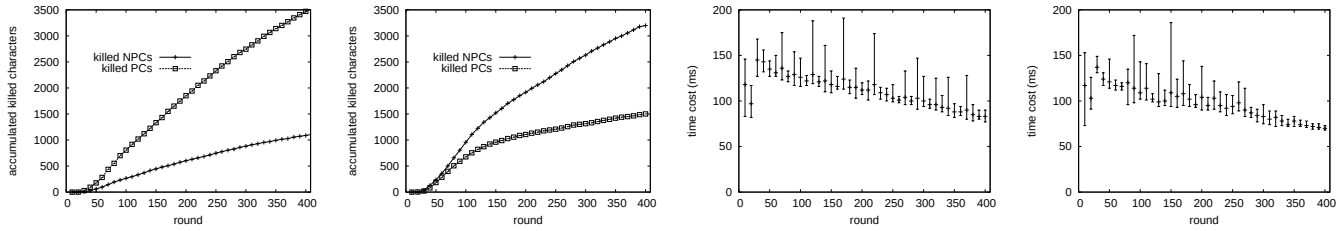
(a) Parameter θ (b) Parameter α

Fig. 11. The impaction of some parameters

B. Efficiency and Scalability

We compare the CPU cost of every ticks under two models in Figure 10(c) and 10(d) respectively. Each point in these two figures is a statistics (average, min, max) of CPU time cost over 10 consecutive ticks. Note that the proposed updating strategies have been applied in these two models. The results show that the CPU time cost of all ticks is bounded by $200ms$ under both of the two models. The CPU cost generally decreases when the time advances. This is because the population drops with the advance of fighting. The CPU cost increases at the beginning of the game simulation because more number of PCs are detected by NPCs, and start to fight with NPCs.

The time cost of the binding model is only slightly higher than the basic model. We measure the average CPU time cost from the start of the game to the tick when 10 percentages of characters have been killed. The average computational cost of the binding model is only 12.7% more than the basic model.



(a) Efficacy of the binding model (b) Efficacy of the basic model (c) Efficiency of the binding model (d) Efficiency of the basic model

Fig. 10. Comparison of the binding model and the basic model

This means that the binding model does not incur too much additional overload over the basic model.

We test the scalability of the binding model by varying the population in the game. The statistics of the first 100 ticks of each test are shown in Figure 12. We can observe that, the binding model can be scaled to around 20,000 characters in our experiment if time cost within one tick is strictly constricted as $200ms$ (as the game is simulated 5 ticks per seconds).

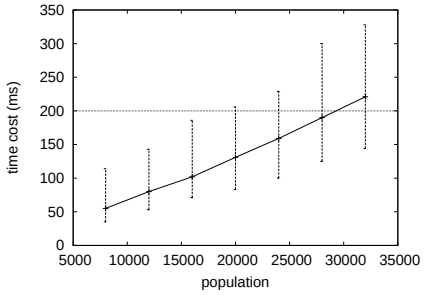


Fig. 12. The scalability of the binding model

C. Updates

We study the performance of lazy update scheme on indexing moving characters and checking observable relationship of characters. In this test, we set the map size based on the population such that each character observes 2 enemies in average within its observable region. All characters are randomly distributed in the map. They are moving around, but not fight with each other.

We first compare the computational cost of two update schemes: the straightforward update and the lazy update under different populations. In the straightforward update scheme, a character updates its position at each tick if it moves, the observable relationship is updated accordingly. In the lazy update scheme, a covering bound with a radius of 40m is applied in all characters. The results are shown in Figure 13. The lazy update scheme achieves 7-8 times faster than the straightforward update scheme. From this figure, we can see that the computational cost is linear to the population. Therefore, lazy update scheme achieves 7-8 times of scalability than the the straightforward update scheme. It supports a population of 20,000 simultaneously moving.

The efficiency of lazy update scheme can be further improved if some errors of distances between characters are tolerated. We achieve this by reducing the frequency of monitoring list refinement. Instead of refining the monitoring list every tick, we can do it periodically. As shown in Figure 14, around $\frac{1}{3}$ computational cost is reduced when the refinement is reduced to 5 ticks one time. While the maximal distance error between two characters is bounded by the sum of the speed of two characters because the frequency of game simulation is also 5 ticks one second.

We show the impact of the updating frequency of binding aggregates on the efficiency and effectiveness of game simulation in Figure 15. The average updating rates are adjusted from 1 to 10 ticks. The average time cost of the first 100 ticks is used as a measure of efficiency, and the relative killing rate r is also used as a measure of efficacy. According to the results, the time cost drops significantly when the sampling rate is enlarged from 1 to a larger value (e.g., 6), while the relative killing rate r drops slightly. Therefore, the updating model can help to improve the efficiency of game simulation, without losing the efficacy too much.

VII. Related Work

Spatial reasoning of game data is important for decision making because game AI needs to sense the dynamic aspects of a game state to achieve appropriate behaviors for NPCs. Tozour [26] introduced a spatial reasoning approach through a layered and grid-based spatial database to manage game data. However, one disadvantage of the grid-based model in spatial reasoning is the huge update cost due to the updates of influential regions within multiple layers.

Game data can be modeled as a complex type spatio-temporal data. In the database community, existing studies on continuous monitoring or queries over spatio-temporal data focus on nearest neighbor queries [14], [17], range queries [14], [15], and reverse nearest neighbor queries [16]. However, these queries are simple in semantics, and therefore cannot be applied to spatial reasoning of game data. There are some studies related to spatial aggregates in a spatio-temporal database. They focus on density queries [27], [28], which have been designed to identify dense regions in spatio-temporal data. However, density queries do not conduct aggregates over attributes of moving objects. There are usually no moving query regions in density queries.

White et al. [9] proposed scaling the population in RTS game by using a data-driven game engine. Formal query

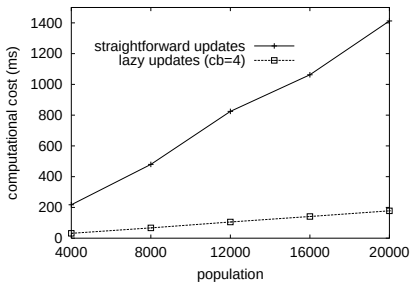


Fig. 13. Lazy updates v.s. straightforward updates of spatial index of moving characters

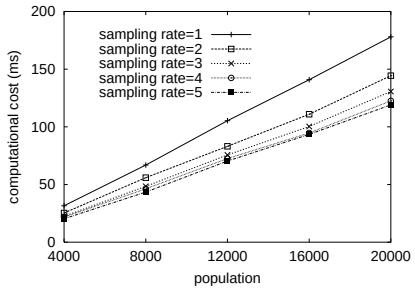


Fig. 14. Impact of sampling rates on the efficiency of positional updates

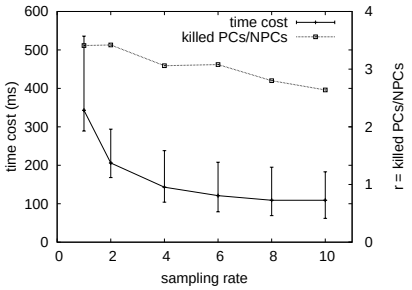


Fig. 15. The impact of average updating rate

languages on spatial aggregates, and some corresponding query optimization techniques are proposed. The proposed techniques cannot be applied to MMOGs as the characters are more complex (with various weapons and armors) in MMOGs, where simple spatial aggregate is not sufficient for spatial reasoning. The spatial reasoning of the large population of characters in MMOGs is indeed a big challenge for game AI.

Reducing the update cost of database system has been well studied in sensor networks [24], [29], [30]. The ideas of model-driven data acquisition [24] and data-driven data acquisition [30], where only those important updates of sensors are delivered to the base station, may be adapted to updates of characters and couplings in the game data update model.

VIII. Conclusion

In this paper, we propose the character binding model for team AI to effectively maintain NPC teams in dynamical environment and efficiently collect information from local game environment.

We adopt self-tuning techniques to manage memberships of bindings so that NPC teams can always be effective. Information of the local game environment is extracted and summarized by bindings. The team AI plan the overall fighting strategy of bindings. Individual NPCs in a binding adapt their own behaviors based on overall fighting strategy. Information and updates of coupled PCs of an NPC can be quickly obtained without accessing a spatial index.

The proposed lazy positional update and sampled update strategies manage game data efficiently. With the support of these techniques, the binding model achieves good scalability. Our experiments show that a commodity PC could efficiently support a population of 20,000 characters (NPCs+PCs) in our binding-based MMOG engine. In conclusion, using the character binding model, we achieve both the effectiveness and efficiency in managing NPC teams.

References

[1] http://www.worldofwarcraft.com/index.xml.
[2] http://www.ea.com/official/thesims/thesimsonline/us/nai/index.jsp.
[3] http://www.eve-online.com.
[4] http://en.wikipedia.org/wiki/Massively_multiplayer_online_game.
[5] http://www.mmogchart.com.
[6] http://www.breitbart.com.
[7] M. Assiotis and V. Tzanov, "A distributed architecture for MMORPG," in *NETGAMES*, 2006, p. 4.
[8] N. Combs, "Declarative AI design for games – considerations for MMOGs," in *AI Game Programming Wisdom 3*. Charles River Media, 2006, pp. 29–43.
[9] W. M. White, A. J. Demers, C. Koch, J. Gehrke, and R. Rajagopalan, "Scaling games to epic proportion," in *SIGMOD Conference*, 2007, pp. 31–42.
[10] C. Gibson and J. O'Brien, "The basics of team AI," in *Game Developers Conference*, 2001.
[11] J. Reynolds, "Tactical team AI using a command hierarchy," in *AI Game Programming Wisdom*. Charles River Media, 2001, pp. 260–271.
[12] W. Sterren, "Squad tactics: Planned maneuvers," in *AI Game Programming Wisdom*. Charles River Media, 2001, pp. 247–259.
[13] P. Tozour, "Strategic assessment techniques," in *Game Programming Gems 2*. Charles River Media, 2001.
[14] H. Hu, J. Xu, and D. L. Lee, "A generic framework for monitoring continuous spatial queries over moving objects." in *SIGMOD Conference*, 2005, pp. 479–490.
[15] M. F. Mokbel, X. Xiong, and W. G. Aref, "Sina: Scalable incremental processing of continuous queries in spatio-temporal databases." in *SIGMOD Conference*, 2004, pp. 623–634.
[16] T. Xia and D. Zhang, "Continuous reverse nearest neighbor monitoring." in *ICDE*, 2006, p. 77.
[17] X. Yu, K. Q. Pu, and N. Koudas, "Monitoring k-nearest neighbor queries over moving objects." in *ICDE*, 2005, pp. 631–642.
[18] C. S. Jensen, D. Lin, and B. C. Ooi, "Query and update efficient b+-tree based indexing of moving objects," in *VLDB*, 2004, pp. 768–779.
[19] W. M. White, C. Koch, N. G. 0003, J. Gehrke, and A. J. Demers, "Database research opportunities in computer games," *SIGMOD Record*, vol. 36, no. 3, pp. 7–13, 2007.
[20] B. Dalton, "Online gaming architecture: Dealing with the real-time data crunch in mmos," in *Proc. of Game Developers Conference*, 2007.
[21] M. DeLoura, *Game Programming Gems. 1*. Charles River Media, 2000.
[22] N. Wallace, "Hierarchical planning in dynamic worlds," in *AI Game Programming Wisdom 2*. Charles River Media, 2003, pp. 229–236.
[23] S. Chaudhuri and V. R. Narasayya, "Self-tuning database systems: A decade of progress," in *VLDB*, 2007, pp. 3–14.
[24] A. Deshpande, C. Guestrin, S. Madden, J. M. Hellerstein, and W. Hong, "Model-driven data acquisition in sensor networks," in *VLDB*, 2004, pp. 588–599.
[25] *The WorldForge project*, http://worldforge.org.
[26] P. Tozour, "Using a sptial database for runtime spatial analysis." in *Game AI Programming WISDOM 2*. Charles River Media, 2003, pp. 381–390.
[27] M. Hadjieleftheriou, G. Kollios, D. Gunopulos, and V. J. Tsotras, "Online discovery of dense areas in spatio-temporal databases." in *SSTD*, 2003, pp. 306–324.
[28] C. S. Jensen, D. Lin, B. C. Ooi, and R. Zhang, "Effective density queries on continuously moving objects." in *ICDE*, 2006, p. 71.
[29] M. Sharifzadeh and C. Shahabi, "Supporting spatial aggregation in sensor network databases," in *GIS*, 2004, pp. 166–175.
[30] A. Silberstein, G. Filpus, K. Munagala, and J. Yang, "Data-driven processing in sensor networks," in *CIDR*, 2007, pp. 10–21.