Action Prediction of Opponents in MMORPG using Data Stream Mining Approach with Heuristic Motions

Chih-Ming Chiu, Shao-Shin Hung [*]

ABSTRACT
Millions of people now participate in massively multiplayer online role playing games (MMORPGs), placing tremendous and often unpredictable maintenance burdens on their operators. Thus, understanding the dynamic nature of MMORPGs players is critical for the designers and implementers of the systems and networks that host MMORPGs. To the best of our knowledge, little work builds character interaction model based on the data stream mining**.** This work improves our understanding the behaviors of opponents of player in MMORPGs by collecting the behavior data, extracting frequent behavior patterns, learning the hidden hints and making good prediction on responses to the unexpected impacts. Besides, we develop two efficient approaches for mining the behavior data for find the interesting behavior pattern for future prediction on responses of opponents. Our novel findings include the following: One, due to the constraints of limited resources of time, memory, and sample size, *MSS-MB* was proposed to meet these conditions; the other, due to the constraints of real-time and on-line, there may have some errors occurred in the processing period, *MSS-BE* was proposed to control the errors as needed. Finally, based on the experimental results, we can efficiently and effectively predict the responses of opponents in the MMORPGs.

Keywords
MMORPGs, animation, data stream mining, prediction, opponent

1. Introduction
With an exponentially growing population and the increasing diversity of network gamers, the virtual worlds constructed by MMORPGs (Massive Multiplayer Online Role Playing Games) are gradually becoming a field for the study of social behavior [2,3,4]. While psychologists analyze network game-playing behavior in terms of players' social interaction and experience, understanding user behavior is equally important to 3D on-line game researchers, because how users act determines how well 3D game systems, such as animation, or stimulation. For example, the dispersion of players across a virtual world affects how well an algorithm performs in distributing the workload to a number of servers in terms of bandwidth usage, load balancing, and users' perceived quality of games. Besides, among different types of digital games, massively multiplayer online role playing games (MMORPGs) continue to grow in popularity. For example, Dickey [4] also argues that playing MMORPGs can foster learning while requiring players to think, plan, and act critically and strategically. He further indicated that different types of quests in MMORPGs could facilitate different knowledge domains (e.g. declarative knowledge, procedural knowledge, strategic knowledge, and meta cognitive knowledge).

On the other side, character interaction model has become prevalent in the computer graphics industry, the general adaptation and re-use of the data is limited by the available techniques for editing and modifying data in controllable ways. Physical models are commonly used to add response to characters by generating modifications through collision forces applied to a dynamic simulation that takes over during contact for motion capture (or key frame motion.) In many games and movies, when impacts are detected, motion-capture driven characters are turned over to a dynamics engine to accept the impulse [12]. The problem with this method is

C.-M. Chiu[1] and S.-S. Hung[2,*]
[1]National Chung Cheng University, Chiayi, Taiwan
C.-M. Chiu
e-mail: zv6@mail.stut.edu.tw
S.-S. Hung
[2,*] WuFeng Institute of Technology, Chiayi, Taiwan
S.-S. Hung
e-mail: hss@cs.ccu.edu.tw

that their models and approaches are based on vector models [12, 13, 17] or final state machines [13, 17].

To gain a better understanding of the patterns of player interaction (or NPC: non-Player Character) and their implications for game design, we analyze "*how and what players interact*". Analyzing user behavior based on database-level traces is particularly useful for our purpose, since the inferred user behavioral patterns would naturally connect to interactions factors, such as styles of attacking (and defense) between participating parties. Based on an empirical analysis of player interaction inferred from network traces, this work aims to put forward architectural design recommendations for online games.

The followings highlight the differences. First, they only model the behaviors of players and their opponents in a complex way such that the property of "real-time" is neglected or missed; second, they cannot extract the meaningful and interesting patterns for learning the actions of opponents for future prediction of action styles (i.e., attacking or defense mode); third, based on the intelligent-based model, we can efficiently and effectively predict the future responses of opponents and this can attract more attentions of players or get players involved in the game with startling reality; finally, the adaptation for skills of players is less or missing. In this paper, under the constraints of real time and on line, we present a technique which automatically predicts future responses of the opponent in dynamic and adaptive reaction to an unanticipated impact.

Also the nature of adaptive behavior needs to be decided. As Mozer [5] states in his criticism of smart houses, it is extremely hard to infer the state of mind of the user from observing her behavior, so a more subtle approach needs to be considered, similar to that of recommender systems [6], where the information provided is filtered, according to patterns of similar behavior of other users.

Therefore, in MMORPGs, our User Model (UM) constitutes a basic component of every adaptive game system. The UM represents user characteristics that enable the system distinguish among different players. The UM is built using data that are requested either directly by the players or obtained by logging player interaction. There are different types of player data that can be used for building the UM, for example:

- Player's characteristics (such as age, gender, location, etc.);
- Player's preferences and interests;
- Player's knowledge and skills;
- Player's behavioral patterns.

While certain user characteristics may be derived directly from data, others are inferred after processing and interpretation of usage data. The approach taken by our system is to support continuous queries on data streams on line and real time**,** and then make predictions.

A data stream is an ordered sequence of items that arrives in timely order [7, 9, 10, 16, 17]. Different from data in traditional static databases, data streams are *continuous*, *unbounded*, usually come with *high speed* and have a data distribution that often *changes with time* [7]. Under careful observations, our high speed game play data (e.g. continuous sequential player's styles) coincide to have the same characteristics. Besides, there are some inherent challenges for data stream association rules mining [9]. First, owing to the unlimited amount of stream data and limited system resources, such as memory space, a mining mechanism that adapts itself to available resources is needed. Second, due to the characteristics of data streams, there is a huge amount of data in both offline and online data streams, and thus, there is not enough time to rescan the whole database or perform a multi-scan as in traditional data mining algorithms whenever an update occurs.

Because of these constraints, traditional data mining methods developed for static databases cannot be applied for mining data streams. Here, in our action MMORPG genre, the style data is generated as a stream, yet we would like to find sequential patterns from the player**s** for analysis. However, there has little work on finding effective methods for prediction of actions of opponents in MMORPGs**.** In this paper, we adopt the algorithms from L. Mendes et al. [16] and make necessary modifications for our work. We propose two effective approaches for mining sequential patterns from continuous player's style data: *MSS-BE* (Motion-based Stream Sequence miner using Bounded Error) and *MSS-MB* (Motion-based Stream Sequence miner using Memory Bounds).

The remainder of this paper is organized as follows. Related works are discussed in Section 2. Section 3, describes the proposed problem definition. Section 4 presents our system framework and 5 explain the recommended data streaming mining algorithms with illustrative examples. Section 6 then presents the experiment results. Conclusions are finally, drawn in Section 7, along with recommendations for future research.

2. Related Works

2.1 Character interaction models

Many virtual humans are used in computer games like RPGs, sports games, and action games. Fighting games like Virtua Fighter [18] are pioneer users of virtual humans. In these games, the reactions of virtual humans are created by connecting motions in a motion database such as Motion Graph [11, 12, 13, 17]. Therefore, variations among the reactions are limited, and there is a large cost to creating a motion database. Using a database approach, Lee and Lee [8] created realistic motions for boxers; but their method did not create dynamic reaction for contacts. Advances in human interface and virtual reality technologies make possible a wide variety of user inputs. Hence virtual humans are required to perform a wide variety of reactions. Dynamic control and simulation methods use controllers to compute joint torques based on current states and desired actions. These methods create dynamically correct motions from specified motions [12], and state machines [13]. These methods generate appropriate motions for variations of the input. State machines are used to create the motions and actions of virtual characters that correspond to environments Lattner et al. [17] present an approach which applies unsupervised symbolic learning off-line to a qualitative abstraction in order to predict future situation using frequent patterns. However, these researches did not support on-line and real-time direct character interactions.

2.2 Data streaming mining

Several novel algorithms have been proposed in the literature to tackle this problem. There are generally two approaches: *counter-based* methods, and *sketch-based* methods. Counter-based algorithms maintain counters for a fixed number of elements of the stream, and only this limited number of elements is monitored. If an item arrives in the stream that is monitored, the associated counter is incremented, else the algorithm decides whether to discard the item or reassign an existing counter to this item. The prominent counter-based algorithms include Sticky Sampling and Lossy Counting (LC) [16], Frequent (Freq) [7, 16], and Space-Saving (SS) [16].

The other approach is to maintain a sketch of the data stream, using techniques such as hashing, to map items to a reduced set of counters. Sketch-based techniques maintain approximate frequency counts of all elements in the stream, and they can also support deletions. As such, these algorithms are much more flexible than the counter-based methods. The prominent sketch-based algorithms include CountSketch1 (CCFC) [7, 9, 10], GroupTest (CGT) [7, 9, 10], Count-Min Sketch (CM) [16], and hCount (hC) [16].

3. Problem Definitions

Let $I = \{i_1, i_2, \cdots, i_j\}$ denote a set of literals, called items. A sequence *s* contains an ordered set of items *X* from *I* denoted by $< s_1, s_2, \cdots, s_j >$. A sequence $s =< a_1, a_2, \cdots, a_p >$ is a subsequence $s' =< b_1, b_2, \cdots, b_n >$ if there exist integer $i_1 < i_2 < \cdots < i_p$ such as $a_1 = b_{i_1}, a_2 = b_{i_2}, ..., a_p = b_{i_p}$.

A *data stream* of sequences is an arbitrarily large list of sequences. A sequence *s contains* another sequence *s′* if *s′* is a subsequence of *s*. The *count* of a sequence *s*, denoted by *count*(*s*), is defined as the number of sequences that contain*s*. The *support* of a sequence *s*, denoted by *supp*(*s*), is defined as *count*(*s*) divided by the total number of sequences contained. If $supp(s) \geq \sigma$, where σ is a user-supplied minimum support threshold, then we say that *s* is a frequent sequence, or a sequential pattern.

For example, suppose the length of our data stream is only 3 sequences: S_1 = <*a, b, c, d*>, S_2 = <*a, c, d*>, and S_3 = <*b, c, d*>. Let us assume we are given that σ = 0.5. The set of sequential patterns and their corresponding counts are as follows: <*a*>:2, <*b*>:2, <*c*>:3, <*d*>:3, <*a, c*>:2, <*a, d*>:2, <*b, c*>:2, <*b, d*>:2, and <*b, c, d*>2.

■ *Style Definition*: In this work, we define six different styles for distinguishing responses from opponents and NPC. The styles are shown in Table 1.

Table 1. Different styles in our character interaction model.

Id	Style	Name	Id	Style	Name
a		Up-Attack	d		Up-Defense
b		Middle-Attack	e		Middle-Defense
c		Down-Attack	f		Down-Defense

4. System Framework

Our system is divided into four phases including input data stream sequence, data batching mechanism, data stream mining and style prediction mechanism, and NPC animation control. First, player's on-line data is logged and broken into fixed-sized batches. Next, the data stream mining algorithms are utilized to build the lexicographic tree T_0. Once the tree is built, we use the on-line play's styles input to query the tree and make predictions. Finally, the returned next possible player's style and its probability of appearance are used to play the NPC's animation.

Notably, before the tree is built, we do not know about what player's next style is and thus just play the default corresponding attacked animations. The framework is illustrated in figure 1. In the following section, the detail algorithms are described.

5. Stream Sequential Pattern Mining Algorithms

In this section, we will explain our stream sequential pattern mining methods based on the algorithms proposed in Mendes. et.al. [16] and make necessary modifications and improvements. As stated in previous section, both *SS-BE* and *SS-MB* algorithms break up the stream into fixed-sized batches and perform sequential pattern mining on each batch. They use a lexicographic tree T_0 to store the subsequences seen in the data stream.

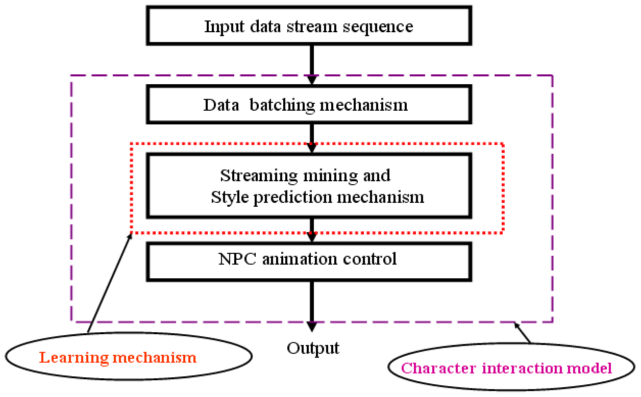
Input data stream sequence
Data batching mechanism
Streaming mining and Style prediction mechanism
NPC animation control
Learning mechanism
Output
Character interaction model

Fig. 1: Our system framework.

5.1. The *Motion-based SS-BE (MSS-BE)* Algorithm

Now, we will explain our modified mining algorithms. For the modified method, each node below the root in the tree T_0 will have four attributes:

- ■ *Style*: denotes a player's style represented by this node;
- ■ *Count*: denotes the count of the sequence corresponding to the path from the root to this node;
- ■ *PID:* denotes the last pruning batch number before this node was inserted in the tree;

- *batchCount*: indicates how many batches have contributed to the count of this node since the last time it was introduced to the tree

The *MSS-BE* algorithm will be given below and followed by the demonstration example.

Motion-based SS-BE (MSS-BE) Algorithm

// Tree T_0 is set initially to have the root node with its count that denotes the number of sequences occurred set to 0.

Input:

- D: an incoming stream of sequences $S_1, S_2, \ldots$;
- σ: the minimum support threshold;
- ε: the significance threshold where $0 \le \varepsilon < \sigma$;
- L: the length of batch;
- α: the batch support threshold where $0 \le \alpha \le \varepsilon$;
- δ: the pruning period.

Output: tree T_0

```
Begin
1. Break the data stream into fixed-sized batches,
   where each batch contains $L$ sequences;
2. While (each batch $B_k$ that arrives) do
3. Begin
4.   Apply PrefixSpan [1] to it, using support $\alpha$;
5.   For each frequent sequence $s_i$ that is found,
       having count $c_i$ do
6.       If a path corresponding to this sequence
           does not exist in the tree Then
7.           Create one new path, setting the
             batchCount and count values of the new
             nodes to 0 and the PID values to the last
             pruning batch number;
7.       Else
8.           Increment nodes' count by $c_i$ and their
             batchCount by 1;
9.       EndIf
10.  EndFor // For
11.  If the number of batches seen is a multiple of
         the pruning period, $\delta$ Then
12.      For each node in the tree do
13.          Let $B$ equal total number of batches
             elapsed minus the node's PID value;
14.          Let $B' = B - batchCount$;
15.          If count + $B'(\lceil \alpha L \rceil - 1) \le \varepsilon BL$ Then
16.              Delete the entire sub-tree rooted at
                 that node from the tree;
17.          EndIf
18.      EndFor
19.      Update the last pruning batch number;
20.  EndIf
21.  return $T_0$;
22. EndWhile
End // procedure MSS-BE ends
```

Our modifications are in underline and using the node's *PID* value to compute the *B*'s value instead of using the timestamp approach. This improves the efficiency of tree pruning.

Whenever the set of sequential patterns are needed, we can output all sequences corresponding to nodes in the tree having count $\ge (\sigma - \varepsilon)*N$, where N is the number of sequences occurred.

Example 1 (*MSS-BE*). Suppose the batch length L is 4, the minimum support threshold σ is 0.7, the significance threshold ε is 0.5, and the batch support threshold α is 0.4, and the pruning period δ is 2. We assume that player's styles arrive in the following data stream shown in Figure 2.

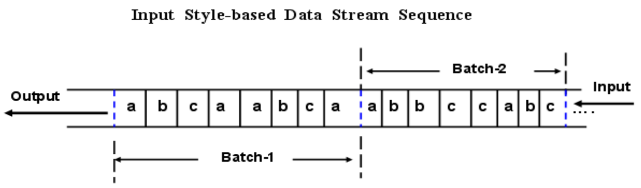
Input Style-based Data Stream Sequence
Batch-2
Output
Input
a b c a a b c a a b b c c a b c
Batch-1

Fig. 2: Overview of incoming stream sequence for *MSS-BE*.

S1.Break the data stream into fixed-sized batches, where each batch contains L sequences. Let batch B_1 be consisted of sequences <a,b>,<c,a>,<a,b> and <c,a>. Let batch B_2 be consisted of sequences <a,b>,<b,c>,<c,a> and <b,c>.

S2.The algorithm begins by applying PrefixSpan [1] to the first batch with minimum support 0.4. The frequent sequences found, followed by their support counts, are: <a>:4, <b>:2, <c>:2, <a,b>:2, and <c,a>:2. Nodes corresponding to each of these sequences are inserted into the tree T_0 with the respective counts, a *batchCount* of 1 and a *PID* of 0. The state of the tree at this point is shown in Figure 3(a). We then moves on to the next batch, applying PrefixSpan [1] to B_2, again with support 0.4. The frequent sequences found, followed by their support counts, are: <a>:2, <b>:3, <c>:3, and <b,c>:2. This causes the nodes corresponding to sequences <a>, <b>, and <c> to have their counts incremented by 2, 3, and 3, respectively. In addition, each of these nodes has its *batchCount* variable incremented to 2. A new node is created corresponding to the sequence <b,c>, having count of 2 and *batchCount* of 1. The state of the tree at this point is shown in Figure 3(b).

S3:Because the pruning period is 2, we must now prune the tree. For each node, B is the number of batches elapsed since the last pruning before that node was inserted in the tree. In this case, the last pruning can be thought of as occurring at time 0. Thus, $B = 2$ for all nodes. For each node, $B' = B - batchCount$. In this case, some nodes have $B' = 0$, whereas others have $B' = 1$. According to the algorithm, we prune from the tree all nodes satisfying:

$$count + B'(\lceil \alpha L \rceil - 1) \leq B*\varepsilon*L$$

In this case, that reduces to: $count + B' \leq 4$. The state of the tree after pruning is shown in Figure 4.

If the user requests the set of sequential patterns now, the algorithm outputs all sequences corresponding to nodes having *count* at least $(\sigma - \varepsilon)*N = (0.7-0.5) * 8 = 1.6$. The output sequences and counts are: <a>:6, <b>:5 and <c>:5.

5.2 The Motion-based *SS-MB* Algorithm

- *Style*: denotes a player's style represented by this node.
- Over_estimation: denotes an upper bound on the over-estimation of the count of this node in the tree compared to the true count of the sequence corresponding to this node.
- *Count*: denotes the count of the sequence corresponding to the path from the root to this node;

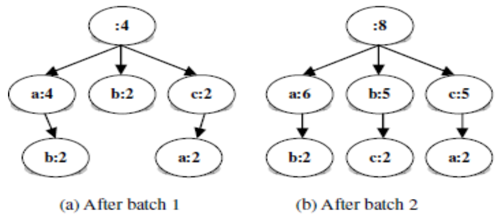

Fig. 3: States of T_0 in *M*SS-BE

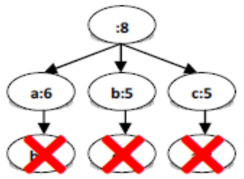

Fig. 4: States of T_0 in *MSS-BE* after pruning

Each node below the root in the tree T_0 will contain only three attributes. Another algorithm —*MSS-MB* algorithm and their pseudo codes are as follows.

Motion-based *SS-MB (MSS-MB) Algorithm*

// Tree T_0 is set initially to have the root node with its count that denotes the number of sequences occurred set to 0.

Input:

- *D*: a incoming stream of sequences *S1*, *S2*, . . .;
- σ: the minimum support threshold;
- ε: the significance threshold where $0 \leq \varepsilon < \sigma$;
- *L*: the length of batch;
- *m*: the maximum number of nodes in the tree.

Output: Tree T_0

```
Begin
1. Break the data stream into fixed-sized batches,
   where each batch contains L sequences;
2. min = 0;
3. While (each batch $B_k$ that arrives) do
4. Begin
5.   Apply PrefixSpan [1] to it, using support $\varepsilon$;
6.   For each frequent sequence $s_i$ that is found,
     having count $c_i$ do
7.       If a path corresponding to this sequence
         does not exist in the tree Then
8.           Create one new path, setting the
             over_estimation values of the new nodes to
             min and count values to count + min ;
9.     Else
```

```
10.         Increment nodes' count by $c_i$;
11.     **EndIf**
12.   **EndFor**
13.   **If** the number of nodes in the tree exceeds *m*
      **Then**
14.     Remove from the tree the node of minimum
        *count* ;
15.     Set *min* to equal the *count* of the last node
        removed.
16.   **EndIf**
17.   return $T_0$;
18. **EndWhile**
**End** // end of *MSS-MB* algorithm
```

Our modifications are in underline and we found that it is necessary for correctness of the algorithm according to the proof. Whenever the set of sequential patterns are needed, we can output all sequences corresponding to nodes in the tree having *count* > $(\sigma - \varepsilon)*N$, where N is the number of sequences occurred.

Example 2 (*MSS-MB*). Suppose the batch length L is 4, the minimum support threshold σ is 0.7, the significance threshold ε is 0.5, and the maximum number of nodes allowed in the tree after processing any given batch is 6. We assume that player's styles arrive as the Figure 5 shown.

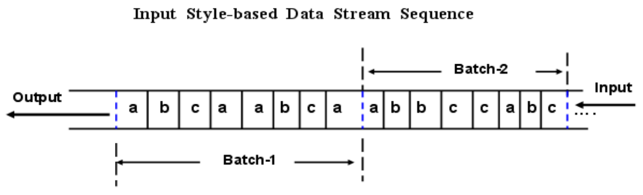
Input Style-based Data Stream Sequence
Batch-2
Output
Input
a b c a a b c a a b b c c a b c
Batch-1

Fig. 5: Overview of incoming stream sequence for *MSS-MB*.

S1 Break the data stream into fixed-sized batches, where each batch contains L sequences. Let batch B_1 be consisted of sequences <a,b>,<c,a>,<a,b> and <c,a>. Let batch B_2 be consisted of sequences <a,b>,<b,c>,<c,a> and <b,c>.

S2.The algorithm begins by applying PrefixSpan [1] to the first batch with minimum support ε = 0.5. The frequent sequences found, followed by their support counts, are: <a>:4, <b>:2, <c>:2, <a,b>:2, and <c,a>:2. Nodes corresponding to each of these sequences are inserted into the tree T_0 with the respective counts and an *over_estimation* of 0. The state of the tree after processing this batch is shown in Figure 6(a).We then moves on to the next batch, applying PrefixSpan [1] to B_2, again with support 0.5. The frequent sequences found, followed by their support counts, are: <a>:2, <b>:3, <c>:3, and <b,c>:2. So the nodes corresponding to sequences <a>, <b>, and <c> to have their counts incremented by 2, 3, and 3, respectively. A new node is created corresponding to the sequence <b,c>, having *count* as 2 + *min* = 2 and *over_estimation* as *min* = 0.

S3:Because there are now 7 nodes in the tree and the maximun is 6, we must remove the sequence having minimun *count* from the tree. Breaking ties arbitrarily, the node corresponding to the sequence <c, a> is removed. The variable *min* is set to 2 where this sequence's *count* before being deleted. The current state of the tree is shown in Figure 6(b).

If the user requests the set of sequential patterns now, the algorithm outputs all sequences corresponding to nodes having *count* > $(\sigma - \varepsilon)N$.

5.3. Action Prediction and Response Algorithm

Once the frequent pattern tree T_0 is created and dynamically maintained by the data stream algorithms. With a input of the player's styles in continuous game play, we can predict his next possible style by traversing the tree and compute the its probability of appearance. The following are our algorithms.

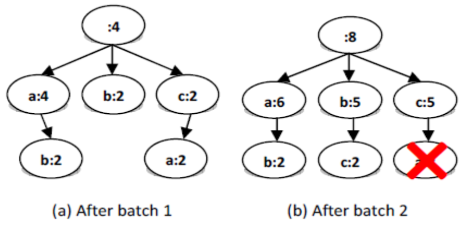

Fig. 6: States of T_0 in MSS-MB

5.3.1 Our Style Prediction Algorithm

Style Prediction (SP) Algorithm

```
// $T_0$ is the lexicographic tree to store the subsequences seen in the data stream. $S_i$ is a player's style seen in the data stream. $S_j$ denotes the player's possible style that follows $S_i$ and prob($S_j$) is its probability of appearance.
Input: Tree $T_0$ and some style $S_i$.
Output: Predicted style $S_j$ and its probability
        $Prob(S_j)$
Begin
1. If there exists first level node $T_k$ that its style
      match $S_i$ Then
2.     Search for the child node of $T_k$ with largest
       count and the sequence corresponding to
       this node that is frequent;
3.   IF found Then
4.   Return the style value $S_j$ and its probability
       of appearance $Prob(S_j) = count / N$ where $N$
       is the total sequences seen;
5.   EndIf
6. EndIf
7. Return "Unknown" and probability value 0;
8. End // end of SP algorithm
```

To speed up the prediction and lower the errors, the tree structure was implemented and shown below.

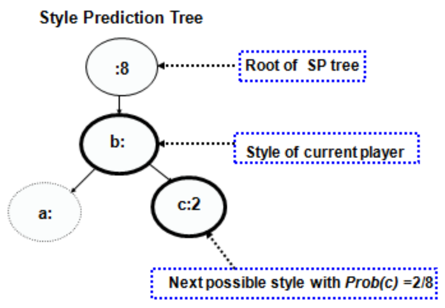

Fig. 7: Tree traversal of our *SP* algorithm

5.3.2 NPC Response Algorithm

```
// A streaming sequence of player's style such as
     $D = S_1, S_2,..$ (assumed that this is infinite)
Input: $D$.
Output: NPC animations results
Begin
1. While (each $S_k$ that arrives) do
2. Begin
3. Predict player's possible next style and its
      probability of appearance $P$ using SP
      algorithm;
4.   If $P \geq 0.5$ Then
5.     Play the corresponding defensive animation
         followed by an attacking animation;
6.   Else If $P > 0$ and $P < 0.5$ Then
7.     Play the corresponding defensive animation;
8.   Else
9.      Play the corresponding attacked animation;
10.  EndIf
11. EndWhile
End; // End of NPC response algorithm
```

Intuitively, this algorithm will mimic a human opponent's behavior in a hand-to-hand combat situation.

6. Experimental Results and Discussion

In this section, the effectiveness of the proposed stream mining algorithm is investigated. All algorithms were implemented in C++ in Torque Game Engine Advanced [14] and the award winning COA Online Game environment. The experiments were run on a PC with a AMD Athlon II X3 2.7GHz CPU and 4 GB main memory, running Microsoft Windows 7 64bit.

The player's and NPC's attacking and defensive style animations are recorded using the Xsens Moven motion-capture system [15]. All the styles are listed in Table 1 and can be applied to both player and non-player characters.

The simulation model we used and the experimental results are provided in Section 6.1 and Section 6.2, respectively.

6.1 Test data and Simulation Model

First, our MMORPG model is called “Caotic of Age Online.” [19] We implement these algorithms in this MMORPG and observe all possible results, and make necessary improvements. To ensure our algorithms feasible, we use the following data stream sequence as a sample player's style data shown in Figure 8.

They can be broken into total 3 batches and each batch is composed of 4 sequences. The first 16 styles and the order are the same as listed in Example 1 where each style is illustrated in Table

1. For the *MSS-BE* algorithm, we assume that the batch length L is 4, the minimum support threshold σ is 0.7, the significance threshold ε is 0.5, the batch support threshold α is 0.4, and the pruning period δ is 2. For the *MSS-MB* method, the maximum number of nodes allowed in the tree after processing any given batch is 7 and the other parameters' values are same as above.

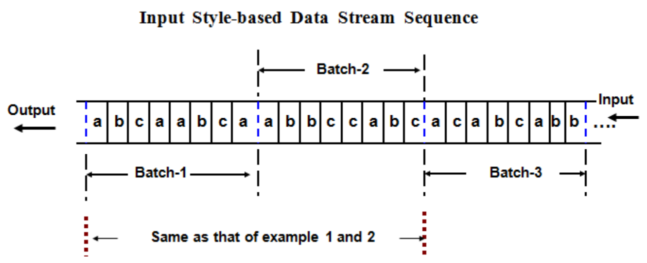
Input Style-based Data Stream Sequence
Batch-2
Output
Input
a b c a a b c a a b b c c a b c a c a b c a b b ...
Batch-1
Batch-3
Same as that of example 1 and 2

Fig. 8: Overview of incoming stream sequence for *NPC response algorithm*.

6.2 Experimental Results

In this subsection, we report our experimental results both on the *MSS-BE* and the *MSS-MB* algorithms. Both algorithms can achieve real-time response running in a separate working thread.

The legends used in the attached media are explained in Figure 9. By observing both Figure 10 and 11, we can easily realize that, in the first 16 styles that player performs; NPC responses in just the same behavior for both methods. The main difference shows afterward in the coming third batch. Because all the second level nodes of *MSS-BE* tree are pruned after batch 2, therefore we are unable to predict the next possible style during the third batch period.

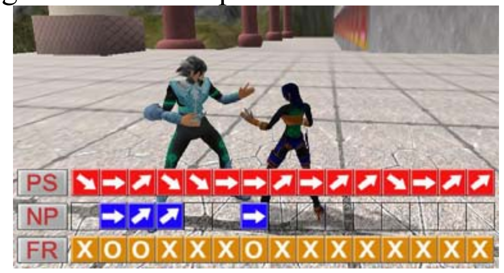

Fig.9: Legends shown in the attached media files, where PS denotes Player's Styles, NP is for NPC's Prediction of style, and FR represents Final Result.

Fig. 10: The simulation in action using *MSS-BE* algorithm.

Fig. 11: The simulation in action using *MSS-MB* algorithm.

Table 2. Results Comparisons

		Parameter Setup	
		σ=0.75	σ=0.7
		δ = 1 or 2	δ = 1 or 2
Methods	MSS -BE	Unpredictable after Batch 1 for δ = 1. And Unpredictable after Batch 2 for δ = 2.	Unpredictable after Batch 1 for δ = 1. And Unpredictable after Batch 2 for δ = 2.
	MSS -MB	Unpredictable after Batch 2.	OK

In summary, the final comparisons are listed in Table 2. Besides, these parameters emphasize the significance and can derive different results in our implementation. In addition, the *MSS-MB* algorithm is better overall for our style prediction algorithm by exploiting all of the available system memory. The interesting relationship between the minimum support threshold σ and the significance threshold ε is yet to be determined, and will be an interesting topic.

7. Conclusions and Future Work

In this paper, we propose two efficient tree-based algorithms, called MSS-BE, for mining the set of frequent stream patterns over data streams with bounded-error; the other is called MSS-MB, for mining the set of frequent stream patterns over data streams with memory-bounded. The differences between them lie in the purpose. MSS-BE focus on the number of errors under the user control; on the other side, MSS-MB pay attentions on the upper bound of used main memory. Based on these algorithms, we develop an on-line and real-time model for predicting response actions of opponents in MMORPGs. Experiments show that the proposed algorithms not only attain highly accurate mining results, but also run significant faster and consume less memory than do existing algorithms, such as vector-based MMORPGs.

In the future work, we still investigate correlations among the parameters suggested in our approaches. Besides, more complex situations between NPC and opponents will be our next goals and researches.

Acknowledgement
The authors would like to thank Wei-Shiang Chen, Feng-Yue Gau, and Rung-Tai Shie for their help to motion-capture the animations.

References

1. Pei, J. et al.: PrefixSpan: mining sequential patterns efficiently by prefix-projected pattern growth. In: Proceedings of the International Conference on Data Engineering (ICDE), pp. 3-14 (2001)
2. Ducheneaut.N. and Moore R. J: The social side of gaming: a study of interaction patterns in a massively multiplayer online game. In: CSCW '04: Proceedings of the 2004 ACM conference on Computer supported cooperative work, pp.360–369. ACM Press (2004)
3. Yee. N: The demographics of groups and group leadership The Daedalus Project, http://www.nickyee.com/daedalus/archives/000553.php.
4. Dickey, M.D.: Game Design and Learning: A Conjectural Analysis of How Massively Multiple Online Role-Playing Games (MMORPGs) Foster Intrinsic Motivation. Educational Technology Research and Development, pp.253–273 (2007)
5. Mozer M. C: Lessons from an adaptive home. In: Smart Environments: Technology, Protocols, and Applications, ed. D. Cook and R. Das, pp.273–298 (2005)
6. Adomavicius G. and Tuzhilin, A: Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. In: IEEE Transactions on Knowledge and Data Engineering 17(6), pp.734–749, (2005):
7 Charikar M., Chen, K, Marach-Colton, F: Finding frequent items in data streams. In: ICALP'02: Proceedings of the 29th International Colloquium on Automata, Languages and Programming, London, UK, pp. 693–703 (2002)
8. Lee, J. and Lee, K.H.: Precomputing avatar behavior from human motion data. In *Proc of SCA 2004*, pp.79-87, (2004)
9. G. Cormode, S. Muthukrishnan, What's hot and what's not: tracking most frequent items dynamically, ACM Trans. Database Syst. 30 (1) pp.249–278 (2005)
10. Shum, H. and Komura. T.: Generating realistic fighting scenes by game tree. In: Posters of the 2006 ACM SIGGRAPH/Eurographics Symposium on Computer Animation (SCA 2006) (2006)
11. Sužnjević, M., Dobrijević, O. Matijašević, M: MMORPG Player Actions: Network Performance, Session Patterns and Latency Requirements Analysis. In. Multimedia tools and applications*,* 45(1-3), pp.191-214 (2009)
12. Zordan,V.B., Majkowska, A., Chiu, B., Fast, M.: Dynamic response for motion capture animation. In: ACM Transaction on Graphics, 24(3), pp.697-701, (2005)
13. Griggs, K.:Assault on the senses[pc-run computer program for movies, In: *IEE Review 49*(3), pp.24-27 (2003)
14. Torque Game Engine Advanced, http://www.torquepowered.com/
15. Xsens Moven, http://www.xsens.com
16. Mendes, L.F., Ding B., and Han, J.: Stream sequential pattern mining with precise error bounds. In: Eighth IEEE International Conference on Data Mining (ICDM'08), pp.941-946, (2008)
17. Lattner, A.d., Miene. A., Visser. U., and Herzog, O.: Sequential pattern mining for situation and behavior prediction in simulated robotic soccer. In: RoboCup 2005: Robot Soccer World Cup IX, Vol 4020, pp. 118-129, LNCS (2005)
18. Suzuki, Y.: Virtua Fighter. Created by Sega studio AM2. (1993)
19. COAOnline, http://coaonline.mes.stut.edu.tw/coa_imagepage/imagepage_index.php.