Building a Player Strategy Model by Analyzing Replays of Real-Time Strategy Games

Ji-Lung Hsieh, and Chuen-Tsai Sun

Abstract—**Developing computer-controlled groups to engage in combat, control the use of limited resources, and create units and buildings in Real-Time Strategy(RTS) Games is a novel application in game AI. However, tightly controlled online commercial game pose challenges to researchers interested in observing player activities, constructing player strategy models, and developing practical AI technology in them. Instead of setting up new programming environments or building a large amount of agent's decision rules by player's experience for conducting real-time AI research, the authors use replays of the commercial RTS game *StarCraft* to evaluate human player behaviors and to construct an intelligent system to learn human-like decisions and behaviors. A case-based reasoning approach was applied for the purpose of training our system to learn and predict player strategies. Our analysis indicates that the proposed system is capable of learning and predicting individual player strategies, and that players provide evidence of their personal characteristics through their building construction order.**

I. Introduction

COMPUTER game developers are constantly looking for ways to use artificial intelligence (AI) in such areas as non-player character (NPC) navigation, NPC coordination, NPC planning and learning, player modeling, resource management, and path finding and steering [1-3]. In computer games, AI would be used to both control individual characters and single computer opponents and to provide strategic direction for a "computer-controlled" player (e.g., simulated military forces or cities controlled by computer) [2]. Computer-controlled opponent is a background program which is capable of automatically engaging in combat, controlling the use of limited resources, creating army units and buildings, and mining gold and energy. But the programs that make combat with human players possible require an enormous design effort in terms of strategies and interaction options. The biggest challenge for researchers and game producers interested in game AI is designing NPC behaviors that give real players the feeling that they are interacting with other real players. At the same time, designers would like to create highly skilled computer-controlled players that provide challenging opportunities for game practice. Instead of building a large of amount of decision rules, designing adaptive computer-controlled opponent capable of learning by imitating human player is possible to shorten development cycle and increase variety and fun of game.

Manuscript received December 14, 2007. This work was supported in part by the National Science Council of Taiwan ROC (NSC96-2520-S-009-004).

J.L. Hsieh, PhD Candidate is with the Department of Computer Science in National Chiao-Tung University, 1001 Ta Hsueh Road, Hsinchu, Taiwan 300, ROC (phone: +88635712121#59284; e-mail: jirlong@gmail.com).

C.T. Sun, Prof. is with Department of Computer Science in National Chiao-Tung University, 1001 Ta Hsueh Road, Hsinchu, Taiwan 300, ROC (e-mail: ctsun@cs.nctu.edu.tw).

Research on game AI character strategy and decision making emerged from the design of AI opponents in two-player games such as *checkers* and *Othello* [4-6]. *Othello* in particular proved that computer-controlled opponents could be designed to not only compete with but also regularly defeat human players. However, in these games players take turns making moves, while today's designers are interested in creating teams of agents to engage in combat in continuous, non-turn-based environments [7, 8]. *RoboCup Soccer* (http://www.robocup.org/02.html), a real-time soccer game played by two teams of agents, is a popular environment for practicing an AI concept known as case-based reasoning (CBR)—solving problem by learning expertise embodied in past cases, rather than encoding classical AI rules [7].

Taking into consideration the complex environments of commercial games, Ponsen and Spronck [9] developed a lattice for representing and connecting abstract states for *Wargus*, a moderately complex RTS game that mimics the popular commercial game *WarCraft II*. Other researchers have used it to manage a real-time city and to prevent repetitive failures in *SimCity* [10], and to develop a computer-controlled opponent in a shooting game named *Quake II* [11]. Buro et al. suggest that real-time strategy (RTS) games provide a good environment for testing ideas for many of these aspects, especially real-time planning, decision making under uncertainty, opponent modeling and cooperation, resource management, and path finding [1]. Research on AI in RTS games is receiving broad support from defense agencies interested in strategy development as well as commercial game developers for creating vivid online scenarios [7].

In RTS games, players control their own units and engage in real time combat. Game play consists of rapid multiple-unit actions requiring resource collection and control, base building, technology development, and military unit control [12]. RTS user interfaces generally consist of a combination of computer keyboards and mouse pointing devices. Since familiarity with game controls and combat strategies is critical to winning, "pro-gamers (professional game players)" frequently practice 10 hours per day to develop and hone high-level tactics and tricks, both of which are attracting research attention [13]. Pro-gamers are known to collect data on and directly observe other players in competition for the purpose of learning and developing effective strategies to defeat their opponents [13].

Two of the biggest challenges to researching AI and human player strategies and behaviors in RTS games include unique game world features and game environments that are tightly controlled by game companies [1, 7, 14]. Nevertheless, several attempts have been made to motivate AI research in this area. The *Open-Real-Time-Strategy (ORTS)* project is proposing the establishment of a programming environment for conducting real-time experiments and the construction of an AI system that outperforms human players [1]. Key elements of RTS games that *ORTS* researchers are defining and controlling include terrain design, real-time interaction, unit action and movement, and team command. A second project, *Testbed for Integrating and Evaluating Learning Techniques (TIELT)*, is a free middleware tool that integrates AI decision systems and a simulator [14]. Aha et al. have used it to create an AI case-based plan selection system in *Wargus* [7]. *Wargus* allows users to play *WarCraft II* using a *Stratagus* engine that enables them to modify numerous in-game parameters depending on a researcher's needs.

We have decided to use a different research strategy: collecting thousands of game replays from the *GosuGamers* game community (http://sc.gosugamers.net/replays.php) and using the game logs to evaluate player behaviors, analyze player strategies, and train an AI system to learn player strategies. Most of our game replays have been collected from highly skilled pro-gamers participating in open competitions. Our long-term goal is to design a computer-controlled opponent that can learn player strategies and styles and employ them in game combat with human players.

II. Mining RTS Player Strategies

The method of collecting in-game demographic data has been recognized as an efficient scientific approach to investigate virtual world [15]. Some game companies allow their players being capable of collecting the in-game data. For example, the replays of *StarCraft* and *WarCraft* can be saved for further review and analysis as well as the designer of *World of Warcraft* allows its players gathering demographic data on player's avatar in whole game server [16]. Some researchers cooperate with game company to obtain logs of game server or networking data for exploring the player behavior and game ecology [17]. Other researchers locate their avatar at several entrances to obtain the player's chatting history or demographic data [18].

We used replays of *StarCraft* games to analyze player strategies in terms of building construction order and unit creating strategies. *StarCraft*, a science fiction RTS game produced by *Blizzard Entertainment*, was released for use with the *Windows* operating system in 1998. The game play of *StarCraft* is military combat which contains a lot of rapid unit creation, control, and upgrade, resource management, and attack tactics. Fans can combat with each other via *Blizzard*'s internet service *Battle.net*, while other players play in group against computer-control group. In its first year, *StarCraft* became the most popular game in South Korea, and is credited with starting a successful pro-gaming scene in that country. Pro-gamers in South Korea are media celebrities; a professional gamer can earn up to $100,000 US per year through contests. Gaming competitions, which are broadcast live and rebroadcast over three television channels dedicated to professional gaming, attract audiences of one million viewers every night [13]. *StarCraft II* is scheduled for release in late 2007

StarCraft replays contain sequential logs of player actions including mouse actions to select or move selected units, and player's keyboard command to create, build, and control units. A summary example of a replay log is presented as Table I. The main advantage of using action logs instead of video to review combat history is that action logs are much smaller and therefore easier to share for analysis by players. *StarCraft* players can use the *StarCraft* main program to simulate original games by reading log-based replays. A few players have created decoding tools such as *BWchart Replay Analyzer* (http://bwchart.teamliquid.net/) to decode non-text binary files for studying replays. *BWchart* gives users the ability to review statistical information, increases or decreases in actions per minute (APM), building order, hot key usage, and military distribution (broken down into airborne forces, mechanized troops, and infantry units). APM data can be interpreted as reflecting player familiarity with game control, building order as representative of a player's strategies for creating units and engaging in combat, and hot key usage as an indicator of NPC control efficiency. At least two real time strategy (RTS) games—*StarCraft* and *WarCraft III*—allow users to read their own activity logs for the purpose of analyzing their personal playing strategies. The same information offers researchers excellent opportunities to study player actions during all phases of a game. We used a software program called *LordMartin Replay Browser* (http://lmrb.net/) to convert binary replay files into text-based files for advanced analysis. The browser shows player IDs, maps, replay dates, and instruction content. Data are made available on at least two forms of player strategies: actions and building order.

TABLE I
Selected actions and corresponding descriptions. To preserve anonymity, players are named "Player1" and "Player2"

Actions	Description
[E:\Fly[jOin]_KT.MGW)Last.rep	Map path.
_KT.MGW)Last	Map name.
_2005-04-29 06:08	Recorded time.
_Player1,0,T,1,Human,2,200	Player1 ID and reported species.
_Player2,1,T,1,Human,2,233	Player2 ID and reported species.
6,Player1,Train,SCV	Player1 instruction for training a unit named space construction vehicle
6,Player1,Select,9,3631,3631	Selecting several units by mouse.
6,Player1,Move,20,-1,(4030,908), 3714,228	Move the selected units to somewhere.
36,Player1,Attack,21,-1,(3818,1012),0,228	Command selected units to attack somewhere.
168,Player2,Hotkey,19,0,Assign,0	Hotkey assignment
1416,player1,Build,(117,32),Supply Depot	Instruction for building a Supply Depot.
5592,player1,Research,48,3,Spider Mines	Instruction for researching a Spider Mine.

III. Learning Player Strategies by Case-based Reasoning

We adopted Aha et al.'s [7] case-based reasoning (CBR) approach to construct our system for learning and predicting individual player strategies by mining series of actions from replays. Instead of constructing new rule-base system, the main idea of CBR system is solving new problems based on the decision or solution of similar past experience or problems. CBR has been widely adopted in games such as *checkers* and *Othello*, team sports such as *RoboCup Soccer* and *Simcity*, and the real-time strategy game *Wargus* [7]. Instead of designing a finite state machine or rules to construct a computer-controlled player, we used CBR to train a system to learn player behaviors via series of actions found in player replay logs. In RTS games, game play actions or decisions that can be analyzed in terms of strategy include base building, unit selection, strategic movements, attacks, and technology development. Constructed buildings, military units, money, and other resources can be studied as game play *states* (referred to as *cases* in CBR). In RTS game, creating military unit and upgrading technology required constructing specified building in advance. Therefore, our proposed model treats buildings status as game state as well as the action of constructing building as strategy between two states. "Strategies", or "tactics" here, are player's choices on buildings construction. Our model also looks at six features for evaluating a game state: buildings, units, and technology research for two players. Both players' strategies can induce changes of state, as illustrated in the following formula:

$$Building\_set_a\{F_1^a, F_2^a, F_3^a, F_4^a, F_5^a, F_6^a\} \xrightarrow{Strategy_s} Building\_set_b\{F_1^b, F_2^b, F_3^b, F_4^b, F_5^b, F_6^b\}$$

F_i denotes the i feature. Each feature contains a suite of values. Using the *Terran* species in *StarCraft* as an example, the "player units" feature includes lists showing the count of space construction vehicles, seaborne vessels, etc. The "player's buildings" feature includes lists of the numbers of supply depots, command centers, and so on. We can calculate Euclidean distance between two states of one feature using the following equation:

$$Distance(F_i^a, F_i^b) = \sqrt{\sum_{j=1}^{n} (F_{i,j}^a - F_{i,j}^b)^2}$$

with parameter i indicating the i feature, parameter j the number of different kinds of feature values, and parameter a and b the two states. Because we adopted six features for our model, six distances are obtained between two states. Considering different variations in the number of units, technologies, and buildings, we used a ranking mechanism to normalize the six feature variables, with shorter distances indicating a higher rank. For example, if the distance of $feature_1$ is shorter than the distance of $feature_2$ and $feature_3$, and the distance of $feature_2$ is shorter than the distance of $feature_3$, then $Rank_{feuture1}$, $Rank_{feature2}$, and $Rank_{feature3}$ are 1, 2, and 3. Next, we can use the ranking feature to obtain a distance equation between two states. In the following equation, *Rank_Distance* represents the degree of similarity between two states. In RTS game, building construction state can be similar between two different replays, but the values of six features between them can be far different. Therefore, *Rank_Distance* is applied for normalizing the score of each strategy among different replays.

$$Rank\_Distance(Current, Past_k) = \sum_{i=1}^{6} Rank_i(Current, Past_k)$$

with $Rank_i(Current, Past_k)$ denoting the sorted rank of feature i's distance between the k past and current state, i denoting the i feature, and k the k past state.

We collected data on 100-300 replays (seeing the experiment in next section) of combat for the same player in order to train our decision system. The simple training process only needs to break down each replay (each round of combat) into a number of states before storing them. Once a player's action (e.g., constructing or demolishing buildings) changes the state of buildings, the changed state is recorded and the action analyzed in terms of strategy. The decision system consists of a large pool containing a lot of such recorded states and corresponding strategies. It is possible to define strategy performance by tracing the results of several replays, with the strategy performance value increased by one point following a victorious replay and decreased by one point following a losing replay. Each new inputted replay has the potential to change the performance results of identical strategies occurring in the original decision system. This can be expressed as:

$$Performance(Strategy_s, Past_k) = \begin{cases} Win: +1 \\ Unknown\ or\ Draw: Unchanged \\ Lose: -1 \end{cases}$$

Collected replays for a single player are divided into two groups: a training group that forms the strategy and state pool, and a group for verifying the predictive accuracy of the trained decision making AI system. The predicting and verifying process algorithm consists of six steps:

1) Inputting a query replay from the verifying group. Decomposing the inputted replay into states and strategies.
2) Using one inputted state to query the trained decision making system.
3) Finding equivalent states between inputted state and states in the decision system, then obtaining the corresponding strategy for each equivalent state to move to the next state. If inputted state can not be found in state pool, it and its corresponding strategy will be added into states pool.
4) Calculating each strategy's score by summing identical strategies' performance and dividing that value by *Rank_Distance* between query state and corresponding states, then choosing the best strategy.

$$Score(Strategy_s) = \sum \frac{Performance(Strategy_s, Past_k)}{Rank\_Distance(Current, Past_k)}$$

5) Determining if the "best choice" strategy is equal to the corresponding strategy in the query replay, then going back to step 2 and repeating the process until the end of the replay.
6) Calculating the percentage of strategies accurately predicted by the trained decision system.

IV. Result

A. Pre-analysis on Decision Space of Player Building Strategies

Collected data on building construction sequence can be used to analyze and categorize player strategies and playing styles. Those decisions can be visualized as decision trees, with each node representing building status and each link representing a decision to construct a new building. Fig. 1 shows a weighted decision tree formed by one *Terran* species player engaging in combat 100 times (100 replays) with a *Zerg* species player. The game begins at the top of the figure; link thickness represents how many times the same strategy was used during the 100 clashes. The data indicate that during the early stages of combat, the player had a strong tendency to use the same strategies again and again; during the middle stages, no single strategy was dominant; during the final stages, the player returned to the habit of repeatedly using the same strategies. This result can explain why only ten to thirty percentages of possible states are observed in real game play. The decision spaces of three *StarCraft* species are shown in Table II. Each state represents a set of buildings and each strategy represents a player's construction command. Although large differences exist between available states and strategies for each species, the statistical results show that players chose only average three to five strategies at each state. This result indicates that player had strong tendency to use similar strategies.

TABLE II
Number of building states and construction strategies for three species. Statistics were obtained before and after training 100 replays from each species

		Terran	*Zerg*	*Protoss*
Original decision space	Number of possible states (cases)	**2234**	**901**	**388**
	Number of strategies between states	**10887**	**3828**	**1354**
	Average number of available strategies at each state	**4.87**(10887/ 2234)	**4.25**(3828/ 901)	**3.49**(1354/ 388)
After training 100 replays	Number of observed states	**148** **6.6%** (148/2234)	**93** **10.3%** (93/901)	**109** **28.1%** (109/388)
	Number of strategies between observed states	**278**	**164**	**177**
	Average number of strategies at each observed state	**1.87** (278/148)	**1.76** (164/93)	**1.62** (177/109)

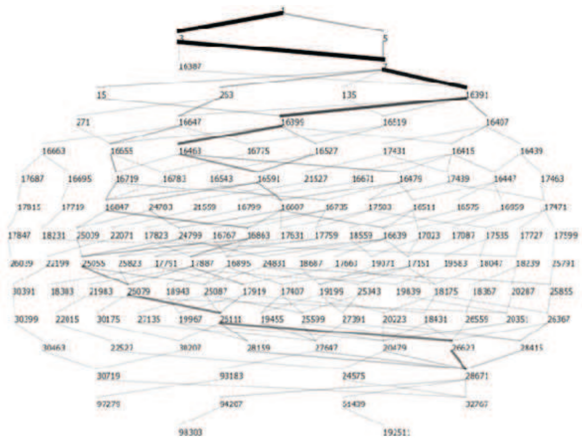

Fig. 1. Building sequence representing 100 combat incidents between a "*Terran*" species player and a "*Zerg*" species player. Link thickness indicates number of times the same strategy was used. Node ID number indicates building status.

B. Player Strategy Prediction

For each species, we chose one player's replays as an example for training our system and verifying prediction rate accuracy. Each player had 300 replays available for analysis. Ninety percent of these replays were used to train our CBR decision system, and the remaining ten percent were used to verify the predicting accuracy of the fully trained decision system. Predicting accuracy is expressed as the percentage of strategies accurately predicted by a trained decision system. Table III lists the system's predictive accuracy after being trained by 100, 200, and 300 replays. Changes in predicting accuracy for the three players are graphically illustrated in Fig. 2. In all three cases, predictive accuracy increased when more replays were inputted into the decision system; this was especially true for the *Protoss* species player due to its possible states are far fewer than other species'. Fig. 2 also shows the large variation in prediction accuracy for the *Zerg* species player.

TABLE III
System predictive accuracy. Statistics were obtained after training 100, 200, and 300 replays for each species (Replays of each species belong to only one player).

Spicies		After 100 replays	After 200 replays	After 300 replays
Terran	Observed (States, Strategies)	(148, 278)	(173, 349)	(195, 412)
	Predictive accuracy	56.5%	62.4	65.1
	Maximum Predictive accuracy	68.5%	80.6%	84.9%
Zerg	Observed (States, Strategies)	(93, 164)	(127, 248)	(136, 277)
	Predictive accuracy	62.0%	62.2%	63.4%
	Maximum Predictive accuracy	76.8%	83.4%	86.9%
Protoss	Observed (States, Strategies)	(109, 177)	(135, 235)	(155, 288)
	Predictive accuracy	67.9%	72.6%	72.8%
	Maximum Predictive accuracy	80.8.9%	86.9%	88.6%

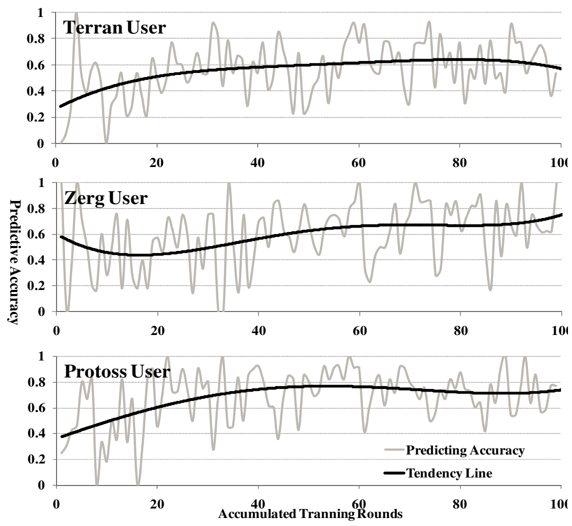

Fig. 2. Predictive accuracy of learning the first 100 replays for player using *Terran*, *Zerg*, and *Protoss* species. After 20 replays of training, predictive accuracy was generally higher than 50 percent. Graph indicates large variation in predictive accuracy for player using *Zerg* species due to large variation in player strategies.

V. Discussion and Conclusion

Although we have shown that our system can learn individual player strategies, our research is still limited by the closed game environments enforced by most game producers. Using our intelligent system to combat with human player needs further effort to design a good interface for automatically executing our system's instruction. However, RTS game replay analyses still be able to benefit game researchers in at least two ways currently:

1) Investigating player activities. The physical dispersion of the large majority of gamers poses significant challenges to conducting online game research. Most efforts to date have consisted of qualitative investigations of player motivations and activities—for instance, surveys [19], individual and group interviews, and secondary data analyses. However, the rapid interaction of RTS game play makes it impossible for players to recall all of their actions and events during combat. Due to the large number of tasks tied to complex unit control and building construction, it is equally difficult for players to clearly describe their game strategies. Some players are unwilling to discuss their strategies and tactics with other players or researchers. These factors make replay log sharing the most useful way to investigate player tactics and actions.
2) Evaluating user interface design. RTS game replays contain information on player use of keyboard and mouse controls, including mouse selection, normal keyboard commands, and hotkey assignments. These interface mechanisms and behaviors are useful for evaluating game control familiarity among players. Game companies can use this data to evaluate their user interface designs.

The current availability of in-game data supports researcher and game designer efforts to learn more about player behaviors in commercial games. The popular Massively Multiplayer Online Game *World of Warcraft (WoW)* also provides a flexible user interface that can be modified to collect other player's in-game actions. This and similar tools help to blur the boundary between virtual and physical worlds. There are at least three benefits for players, researchers, and game designers who utilize the data collection tools and analytical methods described in this paper:

1) They can support game community formation and maintenance. Shared replays give players opportunities for reviewing, discussing, and learning about their own or others' control skills, strategies, and behaviors. In South Korea, players are forming fan clubs for popular pro-gamers as well as for competition-related discussions. Replays encourage further analyses among fans and the publication of certain strategies in the same manner that chess players are known to discuss tactics and defenses. Such interactions help maintain a balance in abilities across species or classes by helping players upgrade their skills and identifying ways to improve playing mechanisms.
2) They can assist game company efforts to review player responses. Instead of passively monitoring commercial sales figures for their individual games, companies can analyze replays to make game design decisions and monitor player actions to determine how users react to changes. For example, level-upgrading mechanisms often draw a great deal of player interest in terms of speed and limitations. Companies can make upgrading more challenging or easier as they see fit.
3) Assist in game AI development. Most NPC interaction rules are created by game developers with little if any consideration of personal playing styles. By mining data on player actions, designers can construct a range of strategies for their NPCs to match specific human playing styles. This will help bring the day when users can practice with their own game AIs or with game AIs based on future opponents' playing styles. This goal requires a mechanism for automatically building interaction rule databases that can be embedded into NPCs. That can contribute to a player's efforts to adopt a new playing style.

References

[1] M. Buro, and T. M. Furtak, "RTS games and real-time AI research," In *Proc. the Behavior Representation in Modeling and Simulation Conference (BRIMS)*, Arlington VA, 2004, pp. 51–58.

[2] A. Nareyek, "AI in Computer Games," *Queue*, vol. 1, pp. 58-65, 2004.

[3] S. Rabin, "*AI Game Programming Wisdom, Vol. 2, 2003,*" Hingham, MA: Charles River Media, 2003.

[4] B. G. Buchanan, "A (Very) Brief History of Artificial Intelligence," *AI Magazine*, vol. 26, pp. 53-60, 2005.

[5] A. L. Samuel, "Some studies in machine learning using the game of checkers," in *Computers & thought*, MIT Press, pp. 71-105, 1995.

[6] J. Schaeffer, "A Gamut of Games," *AI Magazine*, vol. 22, pp. 29-46, 2001.

[7] D. W. Aha, M. Molineaux, and M. Ponsen, "Learning to win: Case-based plan selection in a real-time strategy game," *Proceedings of*

the Sixth International Conference on Case-Based Reasoning. Chicago, IL: Springer, 2005.

[8] M. Molineaux, D. W. Aha, and M. Ponsen, “Defeating novel opponents in a real-time strategy game,” *Reasoning Representation, and Learning in Computer Games(IJCAI Workshop),* Washington, DC, 2005.

[9] M. Ponsen, and P. Spronck, “Improving adaptive game AI with evolutionary learning,” *Computer Games: Artificial Intelligence, Design and Education,* pp. 389-396, 2004.

[10] M. J. Fasciano, “Everyday-world plan use (Technical Report TR-96-07),” Chicago, Illinois: The University of Chicago, Computer Science Department, 1996.

[11] J. E. Laird, "Using a computer game to develop advanced AI," *Computer*, vol. 34, pp. 70-75, 2001.

[12] B. Geryk, “A History of Real-Time Strategy Games,” Gamespot.com 2002. Available: http://www.gamespot.com/gamespot/features/all/real_time/.

[13] M. Molineaux and D. W. Aha, "TIELT: A Testbed for Gaming Environments," *In Proc. the Twentieth National Conference on Artificial Intelligence*, Menlo Park, CA, 2005.

[14] W. Bainbridge, "The Scientific Research Potential of Virtual Worlds," *Science*, vol.317, pp. 472-476, 2007.

[15] N. Ducheneaut, N. Yee, E. Nickell, and R. Moore, "Building an MMO With Mass Appeal: A Look at Gameplay in World of Warcraft," *Games and Culture,* vol. 1, pp.281-317, 2006.

[16] K. T. Chen, P. Huang, and C. L. Lei, "How sensitive are online gamers to network quality," *Communication of the ACM*, vol.49, pp.34-38, 2006.

[17] N. Ducheneaut, R. Moore, and E. Nickell, " Virtual Third Places: A Case Study of Sociability in Massively Multiplayer Games," *Computer Supported Cooperative Work (CSCW)*, vol. 16, pp.129-166, 2007

[18] F. Chee, “Understanding Korean experiences of online game hype, identity, and the menace of the ‘Wang-ta,’” In *Proc. Conference of the Digital Games Research Association (DiGRA 2005)*, Vancouver, CA., 2005.

[19] A. F. Seay, W. J. Jerome, K. S. Lee, and R. E. Kraut, "Project massive: a study of online gaming communities," In *CHI '04 extended abstracts on Human factors in computing systems,* Vienna, Austria, 2004, pp.1421-1424.