Mitigating Information Exposure to Cheaters in Real-Time Strategy Games

Chris Chambers Wu-chang Feng Wu-chi Feng Debanjan Saha
Portland State University IBM Research
{chambers,wuchang,wuchi}@cs.pdx.edu dsaha@us.ibm.com

ABSTRACT

Cheating in on-line games is a prevalent problem for both game makers and players. The popular real-time strategy game genre is especially vulnerable to cheats, as it is frequently hosted as a peer-to-peer game. As the genre has moved towards a distributed simulation approach to gameplay, the number of cheats has been reduced to bug exploits and "maphacks": a form of information exposure that reveals the opponent's units and positions when they should be hidden. This paper proposes a technique for detecting maphacking based on bit commitment and explores the tradeoffs in network traffic and information exposure inherent in reducing information exposure in peer-to-peer games.

Categories and Subject Descriptors

C.2.4 [**Distributed Systems**]: Distributed Applications

General Terms

Algorithms, Performance, Security

Keywords

cheating,games,peer-to-peer

1. INTRODUCTION

Video games grow more popular every year, with the total market approaching $10US billion dollars in 2004 sales [1]. The Interactive Digital Software Association (IDSA) 2004 report cites half of all Americans aging six and older as video game players, and shows a marked rise in players of on-line games over the last two years [2]. Popular on-line games such as Half-life, Lineage, and Warcraft record hundreds of thousands of players per day.

Cheating in on-line games is an area of growing concern for both the security and networking research community as well as the video game industry [3, 4, 5, 6, 7]. This is because of the difficultly in preventing cheating as well as the substantial money at stake in terms of sales and continued player satisfaction. One genre in which cheating is prevalent is the popular real-time strategy (RTS) genre.

In RTS games, a few (2-6) players command virtual armies on a game field and attempt to gain tactical or strategic superiority over their enemies. Players act as the generals, issuing orders to each unit individually or in groups. The size of these armies typically starts small (one to ten units) and grows larger as the game progresses, not usually growing larger than two hundred units.

Central to RTS games and our work is the concept of the *fog of war*, summarized here: player A cannot see player B's unit x unless a unit controlled by player A observes x. Each unit has a scouting radius and any enemy unit within this radius is revealed to the player. The player's vision is comprised of the union of the vision of each of his units, and everything outside of that area is in the fog of war.

This work focuses on *maphacks*, a form of information exposure cheating in RTS games where one player runs a modified version of the game that eliminates the fog of war and displays the entire game state, including the other player's units and move choices, thereby gaining an extremely large advantage in the game.

Players typically have a selection of many units to command, and the games are generally balanced with a "rock, paper, scissors" scheme: one kind of unit is strong against another kind, but weak against a third. Typically the games are finished when a player concedes or loses all units. In this context a maphack, by removing the fog of war for one player, confers an unfair advantage on the user.

Because of the large number of units involved per player, and the financial impact of hosting client/server games, RTS games are typically played via a peer-to-peer architecture. Maphacks are common in RTS games because the players exchange only user input information over the network. Each player's computer simulates the complete game individually. This technique of distributed simulation prevents many other forms of cheating by placing no trust in the other players. For example, players cannot fabricate units that they did not legally build. However distributed simulation leaves the complete game state on each computer, leaving the game open to maphacks.

NOSSDAV'05, June 13–14, 2005, Stevenson, Washington, USA.

Figure 1: Example RTS game (Warcraft 3) interface. The map in the lower left corner shows the player's units and viewable area.

At a high level, our scheme for securing RTS games from maphacks alters the network protocol from exchanging perfect information about what the other player is doing to exchanging information based on what region each player can see. We call the region a player can see his *viewable area*. We propose to utilize distributed simulation for actions within each player's viewable area, but to hide all other actions. We then secure these other actions from cheats by using bit-commitment and post-game verification.

We evaluate our scheme on both network impact and effectiveness in preventing information exposure. We create a model of the network protocol and perform some initial measurements on the network size of viewable areas. We hypothesize that the viewable area bandwidth will dominate the network impact of our scheme, but show the likely increase in message sizes will only be a few hundred bytes.

We also quantify the amount of player information concealed by viewable areas. By exchanging viewable areas, players no longer know the exact game state, but they know something about the other player's units. We create a model of an RTS game and simulate the increased uncertainty and information loss as we vary the unit density on the map and size of the viewable area. We demonstrate a substantial reduction in the total information available and increased uncertainty of unit position.

In the rest of the paper we first discuss work closely related to ours. In Section 3 we show how to use bit commitment to solve the easier problem of securing a simple game such as Battleship from a maphack, and then detail how we apply the same technique to RTS games. In Section 4 we show our initial experiments quantifying the increased bandwidth, uncertainty and information loss introduced by our scheme. In Section 6 we present our conclusions.

2. RELATED WORK

In recent years there have been several attempts to classify and categorize cheating in on-line games [6, 5, 7]. These attempts discuss maphacks specifically, or under the more general category of information exposure.

Baughman et. al. apply bit-commitment to secrets in online games in the context of dead-reckoned games and peer-to-peer games [8]. They introduce a scheme called AS that prevents look-ahead cheats by requiring players to commit to their moves in advance of revealing them. They also use a zero-knowledge proof to determine if two players occupy the same general region (cell) of space without revealing location information. Given the small cell size required for RTS games and the large number of units, this technique would scale exponentially and is infeasible in this context. Our work builds upon theirs by using bit commitment to hide secrets, but focuses on the challenges of RTS games.

Buro addresses the issue of maphacks in RTS games by presenting a client-server architecture (ORTS) to perform visibility culling for each player [9]. ORTS does not meet our goal of a peer-to-peer architecture and instead requires server resources for each game played. With hundreds of thousands of RTS games played on-line per day this solution has scalability issues we wish to avoid.

3. METHODOLOGY

A basic building-block of modern cryptography is bit commitment: a party's ability to make a choice without revealing it and then, at a later date, reveal the choice. Central to the concept is that the committer cannot change his choice after making it, and that others cannot determine the choice before it has been revealed.

We first demonstrate how bit-commitment can secure the simple game of *Battleship* and then we apply it to the more complicated case of RTS games.

3.1 Preventing Cheating in Battleship

In Battleship, each player has five ships placed on a grid. Players take turns calling out a single grid position and telling each other whether the shot was a hit or miss. A player wins when all positions on the other's ships are hit.

Without bit-commitment, Battleship is easy to cheat at, especially in an environment such as a networked game. The kind of cheating depends on whether or not you know the other player's ship positions. It is assumed that this information would not be intentionally displayed to the user, but the reality of today's cheating-heavy environment is that if the information is available on a person's computer, someone will write a program to reveal it.

If player p_1 knows where player p_2's ships are, p_1 can easily cheat by calling out a sequence of shots that hit. If, on the other hand, p_1 does not know where p_2's ships are, p_2 can cheat by telling p_1 that all shots are misses. Player p_1 would never be able to verify that player p_2 was cheating.

One simple technique to secure Battleship is to use bit commitment. Each player p_i picks a secret s_i and a set of initial ship positions sp_i. Each player then sends $h(s_i, sp_i)$ to the other player where h is a cryptographic hash function. Each player must take the other's word when they declare if each shot missed or hit, but at the end of the game, players

exchange (s_i, sp_i). They can verify these against the initial hash, then verify each of the given answers as correct.

Note that the game is not secured in the sense that it is impossible to cheat, but rather each can verify that the other did not cheat. This is the approach we would like to take with RTS games as well.

3.2 Preventing Cheating in an RTS game

Our goal is to secure RTS games such that cheats will be detected. Detection of cheaters is an adequate goal for on-line games because of the high level of control held over players. Players are typically authenticated via a code on their purchased copy of the game to a central server before beginning a peer-to-peer on-line game. This gives the hosting company the ability to globally ban known cheaters from playing.

Cheating in RTS games presents more challenges than cheating in Battleship. Battleship has a few static secrets: the ship locations. RTS games have dynamic sets of units, each of which has a dynamic location. Some of the enemies secrets are supposed to be known, and some are not, based on a player's viewable region.

Our scheme is designed to minimize network traffic while concealing as much information as possible about the enemy without permitting cheating. While the protocol could generalize to a multiplayer peer-to-peer game, we confine our discussion to the simpler two player game for this paper. Our scheme is as follows:

Initial exchange: Each player i generates an initial game state gs_i according to the game rules. Each player generates a secret s_i and sends $h(s_i, gs_i)$ along with the player's viewable area v_i to the other player.

In-game exchanges: For each time slice player P_i performs the following:

1. Send viewable area v
2. Receive opponent's viewable area v'
3. If current move m is in v', send it clear-text
4. Otherwise, send $h(m, s_i)$
5. If P_i's units u just entered v', send them clear-text

Post-game exchange and verification: After the game is completed, each player P_i sends s_i as well as a log of all the moves m_i for which they sent hashes $h(m_i, s_i)$. Then each player simulates the game with complete knowledge of all moves and checks the validity of each sent hash, viewable area and unit.

Using this protocol, players can lie about their viewable area, their hashed move, and what units they control. In the post-game exchange and verification, these lies will be detected. For this process we believe that the Baughman et. al. [8] definition of a *logger service* for each client to record secret moves is adequate. Verifying proper gameplay is beyond the scope of this paper, but we assume it is possible given the moves, hashes, and gameplay engine.

3.2.1 Viewable areas

The network impact of sending the viewable area could be very large, depending on its accuracy and representation. The two extremes of representation for a viewable area are a vectorized representation of units and radii, or a rasterized representation. As the representation more accurately depicts the location of the individual units (as in a vectorized representation), the amount of uncertainty about where the opponents units are decreases. We want to increase uncertainty and minimize bandwidth overhead, so we believe a rasterized viewable area is appropriate for RTS games

We create our rasterized viewable areas from the actual viewable area by mapping onto a raster. We further increase uncertainty and decrease network impact by downsampling to a smaller raster. For our experiments this ratio of larger to smaller raster is fixed at 64:1, and we vary the unit density by varying the number of units.

3.2.2 Proving cheating

The protocol as presented is sufficient for a player to know if a game was played fairly at the verification step. To meet the larger goal of proving to another party that cheating took place, each player must have a public and private key pair. The natural place to store the public keys would be at the authentication server for the game. To alter the protocol to allow for provable cheating each player must send an additional message during the in-game exchange: a signed, dated cryptographic hash of the player's message $(v_1, m_1|h(m_1, s_1), u_1)$ for that timeslice.

By cryptographically signing each message sent with a player's private key, players can achieve non-repudiation; a player can prove that another player cheated if and only if cheating actually took place. This technique enables the central authentication server to ban cheaters, forcing them to buy another copy of the game to play again.

4. EVALUATION

We evaluate the impact of the bit-commitment scheme on three characteristics: the uncertainty it adds, the quantity of information it loses, and the added cost in bandwidth. We model our experiments after the map sizes, unit numbers and proportions used in Warcraft 3.

4.1 Uncertainty

We wish to quantify the amount of information concealed by sending viewable regions instead of unit locations. One general measure of information is Shannon's uncertainty, which measures the disorder and unpredictability contained in a random variable. Shannon uncertainty is defined on random variable x with n possible values over probability distribution $p(x)$ as

$$H(x) = -\sum_{i=1}^{n} p_i log(p_i) \quad (1)$$

We use this equation to calculate the gained uncertainty between a raster containing (random) unit locations and

Experiment	Map area	View Radius	Avg. Num Units
Warcraft 3	11200^2	860	100
vary-num	640^2	49	*vary(1-100)*
vary-rad	640^2	*vary(1-100)*	6
quant	640^2	49	*vary(1-100)*
overlap	640^2	49	*vary(1-100)*

Table 1: Data on experiments performed to quantify uncertainty and information loss

one containing only viewable regions. We assume black and white pixels are equally likely. We evaluate the uncertainty impact of varying the number of units and the view radius of each unit as outlined in Table 4.1.

Figure 2 shows the amount of uncertainty gained as compared to the uncertainty in the maphacked version of the game. Experiment *vary-num* varies the number of units from one to 100 and leaves the radius fixed at the map proportions of Warcraft 3. Each point represents the average of 50 uncertainty calculations with x randomly distributed units. Even at one unit we see a 0.2 uncertainty gain, and this rises rapidly as we add units. At 20 units we gain the most uncertainty, and past that we see some noise in the signal as a result of the increased probability of unit regions overlapping.

For experiment *vary-rad* we vary the radius of the units by an order of magnitude around the Warcraft 3 radius, while keeping the number of units proportional with the map size. Figure 3 shows a substantial initial uncertainty gain initially even at a radius of one, with uncertainty leveling off slowly as the radius exceeds 100. We conclude that the specific radius per unit is less important than the number of units in the game in increasing uncertainty.

The uncertainty gain from unit radius and quantization is considerable. Shannon uncertainty does not directly correlate to the amount of gameplay information hidden (for example, it does not capture the hidden unit types) but it is a useful comparison as it is completely separate from the meaning of the information transmitted. Our results indicate that the peak uncertainty of our scheme falls within the bounds of normal gameplay in terms of unit numbers and viewing radius.

4.2 Information loss

We also present a second metric for evaluating the scheme: information loss. Whereas uncertainty quantifies the likelihood of guessing the color of a pixel, information loss quantifies the number of data points that are deleted. For example, when quantizing a large map into a two by two black and white grid, it is not possible to represent more than four points, no matter how many points existed initially. The lost information in our scheme comes from two sources: the dispersal of a unit's location over an area via its view radius, and the quantization of a large image into a small one.

We model each of these two sources. For quantization, we scatter points in a large map, downsample to the small map,

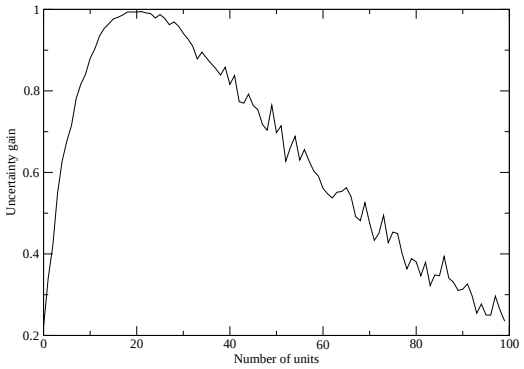

Figure 2: Uncertainty gain from varying the number of units (experiment *vary-num*)

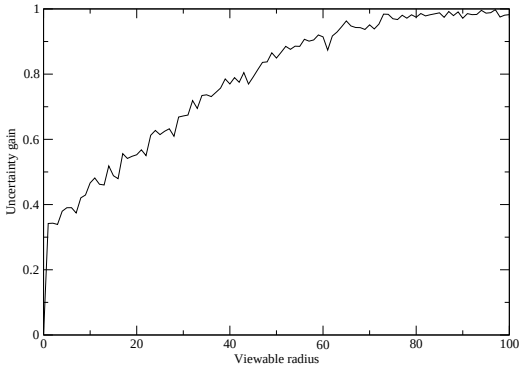

Figure 3: Uncertainty gain from varying the viewable radius of each unit (experiment *vary-rad*)

and count the number of points. The ratio of downsampled points to original points is the measured information loss.

For the view overlap, we scatter points in a large map. When we calculate the viewable area for each point, we disperse its information value (say the constant 1) throughout its viewable area, but do not add anything to an area that is nonzero. By summing over the map and comparing to the original amount of information we have measured the information lost to overlap.

We calculate this loss with experiments *quant* and *overlap* from Table 4.1. Figure 4.2 shows that the information loss from overlap rises more rapidly than quantization for this map size, but both level off very slowly, and the combined positional information loss for our scheme is 11% for proportional numbers of units and map size.

We expect our modeling results show less information loss than trace-driven data would. This is because it is more common for units in RTS games to position in clusters in-

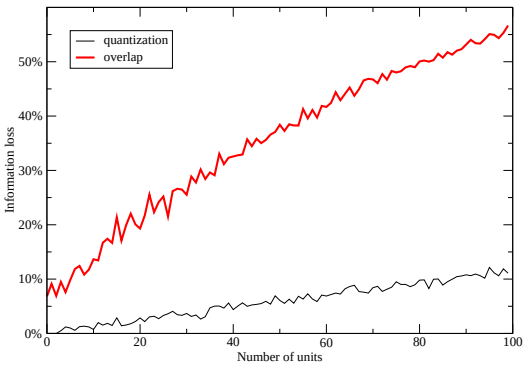

Figure 4: Information loss from quantization and overlap

stead of randomly, which increases information loss in both quantization and overlap.

4.3 Bandwidth

To calculate bandwidth requirements over time, we build towards an equation that determines how much data is sent by one player in a game played up to a particular instant.

RTS games supposedly happen in "real time", but in fact they do have turns, albeit of the high granularity of a millisecond. In theory players could act every millisecond, but a typical move rate is an action every second, or four to five per second for especially intensive bursts. Our formal definition for bandwidth consumed should therefore scale down to milliseconds, but take into account the case of no user input for a given time slice.

Let vr_i be the enemy's viewable region at time i. We define m_i as the player's move at the given moment. This move can be considered a string containing the keyboard and mouse input.

Let

$$m_i = \left\{ \begin{array}{c} \text{player's move at time } i \\ \epsilon \text{ if no move} \end{array} \right\}$$

We define sm_i, the secured version of the move as

$$sm_i = \left\{ \begin{array}{c} m_i \text{ if } m_i \in vr_i \\ h(m_i, s) \text{ if } m_i \notin vr_i \\ \epsilon \text{ if } m_i = \epsilon \end{array} \right\}$$

We define n_i, the new units at moment i as

$$n_i = \left\{ \begin{array}{c} \text{the string of units entering } vr_i \text{ at time } i \\ \epsilon \text{ if no units enter } vr_i \text{ at time } i \end{array} \right\}$$

Let $sign(x)$ be a function that cryptographically signs string x with a player's secret key. We define s_i, the signature for the message at moment i as

$$s_i = \left\{ \begin{array}{c} sign(vr_i, sm_i, n_i) \\ \epsilon \text{ if } (vr_i, sm_i, n_i) = \epsilon \end{array} \right\}$$

Using these definitions, we can construct the size of the data sent up to time t as

$$dataSent(t) = \sum_{i=1}^{t} |vr_i| + \sum_{i=1}^{t} |s_i| + \sum_{i=1}^{t} |sm_i| + \sum_{i=1}^{t} |n_i|$$

The last two summations of this equation are, for infrequent user input, considerably smaller in number of nonzero terms than the first two summations. Additionally, if $|vr_i|$ is stored as an image, it will likely exceed the data requirements for a string of user-input, or a signed hash. Users of today's Internet cannot expect to send and receive vr_i every millisecond. Therefore we relax the restriction that the viewable region be sent every time slice, and instead, send the region every r ms. This changes the definition to

$$dataSent(t) = \sum_{i=1}^{\lfloor t/r \rfloor} |vr_i| + \sum_{i=1}^{t} (|s_i| + |sm_i| + |n_i|) \qquad (2)$$

Viewable areas can dominate equation 2 if they are large, as they may be sent frequently regardless of player interaction. On the other hand, if the cryptographic hashing or signing process is space-intensive, signatures will dominate the equation.

5. DISCUSSION

We propose to more fully evaluate the network impact of our protocol by driving it from user traces of real-world Warcraft 3 games. These are freely available for download, and contain the information in them of which actions each user takes. They do not contain the viewable area information or unit positions, but we have built a prototype system which extracts the data from a running replay. Given enough of these replays, viewable area information, and unit positions, we expect to accurately evaluate the success of our scheme.

One technique to extract the needed data from a replay is to create a video capture of the replay, decode the video and focus on the "mini-map", which displays a small graphic indicating a player's units and their viewable area. The data is approximate; the mini-map is a downsampled two-dimensional representation of a three-dimensional collection of units and necessarily inaccurate. However, we can draw order-of-magnitude conclusions from this data. We analyze a single replay and calculate the bandwidth consumed by the viewable areas as well as the number of units controlled by the player as the game progresses.

In Figure 5 we show an approximation of the number of units controlled by a single player over the course of a typical game of Warcraft 3. It is difficult to deduce if a player-controlled block of pixels in the mini-map is a single large unit or several small ones. We choose to over-estimate the number of units by assuming each block is composed of many units, and as Figure 5 shows, provide an estimated upper bound of 150 units for a game. This demonstrates that our experiments cover the appropriate range of units relative to the map size.

To estimate viewable region bandwidth, we extract the viewable region from a replay of a tournament Warcraft 3

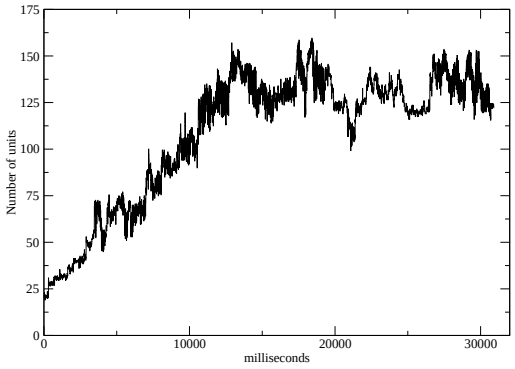

Figure 5: Number of units over time as extracted from a Warcraft 3 replay

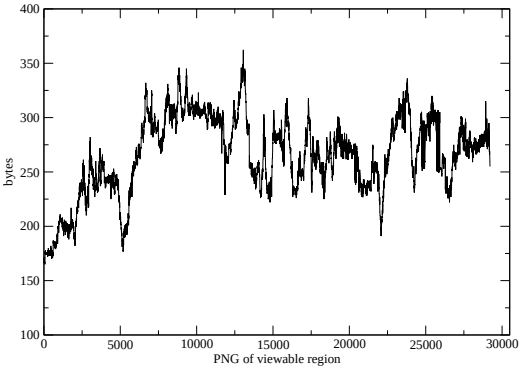

Figure 6: Viewable regions encoded as PNG

game and convert 30000 frames to black and white PNG format images. Figure 6 shows the sizes of these regions, which peak at 360 bytes. To compare this with the space required for cryptographic signatures, the size of a signature using SHA-1 as the hash and RSA-1024 for the encryption is 128 bytes.

Aside from viewable regions and signatures, the other two components of Equation 2 require small lists of units (a few bytes) or user-interface commands (also a few bytes). Given the comparatively large size of the viewable regions, this suggests that the even with a large $r = 5000ms$, the viewable regions and to a lesser extent the signatures would dominate the bandwidth. We expect to verify this in future work.

6. CONCLUSION

We have presented a technique for securing on-line peer-to-peer RTS games from maphack cheats using bit commitment. This scheme does not completely eliminate information exposure of player locations but instead exposes only a player's viewable area. We recognize at least two limitations to this technique: it does not prevent cheating, but merely detects it, and the detection step requires a potentially complex verification procedure.

The goal of the scheme is to hide information. We have evaluated the total information hidden using the metrics of uncertainty and information loss. The information concealed by our scheme is substantial, with the total uncertainty approaching one and information loss approaching 11% for typical gameplay parameters. Varying the radius of view changes the amount of uncertainty very slowly compared to varying the number of units.

We have presented an initial model for the bandwidth consumed by this scheme. While we believe it to be dominated by the viewable regions (and, to a lesser extent, the cryptographic signatures), we also believe the increased bandwidth required to be generally acceptable for real play. We recognize the need for real gameplay data in accurately evaluating the scheme's impact on both bandwidth and uncertainty.

As future work, we plan to capture a volume of gameplay data from Warcraft 3 traces and perform a more extensive evaluation of our scheme. We believe that providing a dataset of real-world gameplay traces, including player unit locations and viewable regions would be a valuable asset for the community to evaluate this and other schemes for improving RTS games.

7. REFERENCES

[1] The NPD Group, "NPD News," `http://www.npd.com/dynamic/releases/press_050119.html`, 2005.

[2] IDSA: Interactive Digital Software Association, "IDSA Digital Press Room," `http://www.idsa.com/pressroom.html`, 2004.

[3] S. Davis, "Why cheating matters," in *Proceedings of the Game Developer's Conference*, 2001.

[4] David Becker, "Cheaters take profits out of online gaming," `http://news.zdnet.com/2100-3513_22-933853.html`.

[5] M. Pritchard, "How to Hurt the Hackers: The Scoop on Internet Cheating and How You Can Combat It," `http://www.gamasutra.com/features/200000724/pritchard_01.htm`.

[6] K. Mørch, "Cheating in online games- threats and solutions," in *Publication No: DART/01/03*. January 2003, Norwegian Computing Center/Applied Research and Development.

[7] J. Yan and H-J Choi, "Security Issues in Online Games," *The Electronic Library: International Journal for the Application of Technology in Information Environments*, vol. 20, no. 2, 2002.

[8] Nathaniel E. Baughman and Brian Neil Levine, "Cheat-proof playout for centralized and distributed online games," in *INFOCOM*, 2001, pp. 104–113.

[9] M. Buro, "ORTS: A Hack-free RTS Game Environment," in *Proceedings of the International Computers and Games Conference*, 2002.