6-4-2012

Development of embedded CAPTCHA elements for bot prevention in Fischer random chess

Ryan McDaniel
University of Louisville

Roman V. Yampolskiy
University of Louisville, roman.yampolskiy@louisville.edu

Original Publication Information

Ryan McDaniel, Roman V. Yampolskiy, "Development of Embedded CAPTCHA Elements for Bot Prevention in Fischer Random Chess", *International Journal of Computer Games Technology*, vol. 2012, Article ID 178578, 6 pages, 2012. https://doi.org/10.1155/2012/178578

Hindawi Publishing Corporation
International Journal of Computer Games Technology
Volume 2012, Article ID 178578, 6 pages
doi:10.1155/2012/178578

Research Article

Development of Embedded CAPTCHA Elements for Bot Prevention in Fischer Random Chess

Ryan McDaniel and Roman V. Yampolskiy

Computer Engineering and Computer Science, University of Louisville, Louisville, KY 40292, USA

Correspondence should be addressed to Ryan McDaniel, rcmcda01@louisville.edu

Received 31 December 2011; Revised 29 March 2012; Accepted 12 April 2012

Academic Editor: Narendra Chaudhari

Cheating in chess can take many forms and has existed almost as long as the game itself. The advent of computers has introduced a new form of cheating into the game. Thanks to the computational power of modern-day computers, a player can use a program to calculate thousands of moves for him or her, and determine the best possible scenario for each move and countermove. These programs are often referred to as "bots," and can even play the game without any user interaction. In this paper, we describe a methodology aimed at preventing bots from participating in online chess games. The proposed approach is based on the integration of a CAPTCHA protocol into a game scenario, and the subsequent inability of bots to accurately track the game states. This is achieved by rotating the images of the individual chess pieces and adjusting their resolution in an attempt to render them unreadable by a bot. Feedback from users during testing shows that there is minimal impact on their ability to play the game. Players rated the difficulty of reading the pieces on a scale of one to ten, with an average rank of 6.5. However, the average number of moves to adjust to the distorted pieces was only 3.75. This tells us that, although it is difficult to read the pieces at first, it is easy to adjust quickly to the new image.

1. Introduction

Chess programs have been designed and implemented on computers since the 1950s. In 1950, Shannon published "Programming a computer for playing chess," in which he presented a chess computer as possible proof of artificial intelligence [1]. At first, these chess programs were created only to test the waters of what computing could do to enhance the game. However, over the years, programs such as Rybka have become very powerful [2]. In 1997, a computer built by IBM, called Deep Blue, even beat then-world champion Garry Kasparov, marking the first time a computer was able to beat a reigning world champion [3]. Some of the chess programs available today include databases of past games and provide numerous ways for players to learn the game and improve their skills. These aspects are certainly positive; however, there are other forms of computer-assisted chess which are not. While cheating in chess can take many forms and has existed almost as long as the game itself, the advent of computers has introduced a new form of cheating into the game. Robots, or "bots," are computer programs that can read a chessboard and the pieces, determine the best possible move to make, and either recommend the move to a player or make the move for them [4]. These bots are easily accessible and can be very difficult to detect. Chess is not the only game plagued by bots, however. These technology cheats are very common in online games today, from traditional games such as poker and chess, all the way up to complex Massively Multiplayer Online Roleplaying Games (MMORPGs) like Blizzard Entertainment's World of Warcraft [5]. Keeping bots from ruining the game for honest players requires a constant effort, since whenever a game update to eliminate bots is implemented, the bot creators update their bot to circumvent the latest fix [6]. Cheating in online gaming can have far-reaching impact on honest players. For example, online poker is played for money. If someone is cheating with a bot, then they are having a direct impact on the other players by taking money from them [7]. Poker is gambling, however, and whenever money is involved, you can expect dishonesty as well. Chess, on the

other hand, has traditionally been about the spirit of the game. Quintessentially, chess is a war campaign, with two players battling it out, planning their short-term and long-term strategies, and utilizing either an offensive or defensive game plan. Inevitably, some plans end in defeat, some in victory, and yet still some end with a draw. In this regard, it is easy to see how one player planning his moves with a computer has a very unfair advantage over the other. The game is not intended to be played this way, so the chess-playing community gets frustrated with cheating players. Unknowingly playing a game of chess against a bot may have varied repercussions. Some players may just brush off the loss, attributing it to a stronger opponent, and trying to learn from it; they may or may not be suspicious that cheating was involved. Others may get so frustrated that they quit playing chess altogether after a few encounters [8]. It is the latter group that makes preventing cheating so very important. Quickly and accurately identifying a bots' presence in an online game is crucial; false positives cannot be tolerated.

2. Related Work

A popular method of ensuring the players involved in a game are, in fact, human players is "Completely Automated Public Turing test to tell Computers and Humans Apart" (CAPTCHA). A typical CAPTCHA test will present the player with distorted text and then require them to type that text into a box in order to continue. A computer program will be unable to read the text and respond correctly, preventing the bot from continuing beyond that point [4].

There are, however, ways to beat a CAPTCHA test. A bug in the CAPTCHA may be exploited to bypass the CAPTCHA test completely, for example, reusing the session ID of an image that is known to have passed the test.

Also, Optical Character Recognition (OCR) software is improving, allowing the bot to "read" the CAPTCHA text. CAPTCHA breaking algorithms have been designed with alarmingly high success rates as high as 70% on some websites [9].

Finally, the bot may present the CAPTCHA to the player as a part of the program. Due to the limitations of a CAPTCHA test, it is not an ideal solution for bot prevention; the test can be easily defeated if it is presented at the beginning of the game, and the test becomes an inconvenience to the player if it is presented during the game [10].

Research on bot detection and prevention in online games has expanded over the past few years, with methodologies ranging from direct impact on the player's game experience, to total transparency, with varying degrees of success. Input devices could be used as a type of hardware-based CAPTCHA; for example, a joystick could be used as a CAPTCHA device, or a specially designed keypad could be used to input a series of characters at certain points during a game. This method would require the output of the device to be authenticated by a game server or console, as well as making the device itself secure to avoid tampering [10]. The cost associated with tamper-proofing devices and the incompatibilities that would exist with legacy hardware limit the usefulness of this approach. Requiring special hardware and keying in characters on a keypad would have a direct negative impact on the game also, as it draws the players' attention away.

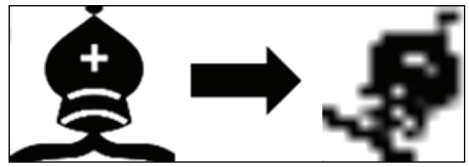

Figure 1: Embedding CAPTCHA by altering the appearance of a bishop.

Embedding a CAPTCHA into the game itself is a clever idea (see Figure 1). However, it can be difficult to implement properly. Randomizing certain aspects of the game can make it much more difficult for a bot to participate. Randomization creates a noninteractive CAPTCHA type of test, as the bot will have to analyze options; however, it is not a particularly powerful deterrent since it can be solved. A more formal test could be presented to a player as well, in the form of a simple text-based or image-based CAPTCHA, in order to allow access to various aspects of the game [11]. This type of test would at least force some human interaction, adding only minor disruption to the playing experience, ensuring that a bot cannot operate completely autonomously.

Server-side bot detection is a method that is concealed to users and typically focuses on the behavioral patterns of game clients. For example, the movement pattern of a character can be analyzed for overly repetitious actions to determine whether a bot or a human is in control [5].

Also, input data from devices such as a mouse or keyboard could be analyzed for button-press-length and interval to determine if a bot is controlling a character [6]. Server-side bot detection requires some resources to analyze the data that is collected, however, and could possibly be circumvented by a bot program [12]. Once a bot is removed from the game, the bot creator can easily change the program to avoid the behavior that resulted in detection.

3. Behavior of a Bot

A robot or "bot," for the purposes of this research, can be described as an artificially intelligent program, with either partial or full autonomy, which assists a player in an online game [4]. One must understand how a bot works before one can discuss how to combat it. A typical bot program will go through three basic steps, the first of which is collecting data for the input. The second step is the heart of the program, where the collected data will be used to create a course of action, predicated by the bots' purpose and design. For example, in this step, a poker bot would determine the action a player should take, while a chess bot would determine which piece should be moved. In step three, the bot will output the desired action to the player, or even perform that action for the player, in the case of a fully autonomous

bot. Bots commonly collect input data in one of two ways. A chess server may give the location of pieces in a log file, possibly even in real time, making it very easy to gather the information needed for a bot to process the locations and determine a move. If the data is not available via a log file, a second data collection option is called screen scraping [13]. In the case of chess, the bot will compare the images on the board with images in its database. The bot can then essentially know which piece is a king, which is a queen, and so forth. The position of each piece is also easily determined, since the board is an image as well. As a result, the bot can look at the board, identify each piece, and its respective location, and process that information to determine the best move to make.

4. Procedure

Written in C#, the software contains menu, grid, and options elements. The grid consists of sixty-four separate panels, each representing a single space on a chess board. Each 100 × 100 pixel panel is added to a two-dimensional array, arranged in an 8 panel × 8 panel square. This layout makes it simple to place a chess board image behind the panels, allowing the panels themselves to contain the chess piece images. While standard functions are used to determine the movability of pieces, it is the resolution, rotation, and randomization which provide additional ammunition to prevent bot play, and protect the players who simply wish to pursue a challenging game of chess against another human opponent.

The algorithm is based on two morphing functions:

Rotate Image Function

(i) Accepts image data type as argument.

(ii) Generate random number between −35 and +35.

(iii) Create new bitmap from image passed in.

(iv) Create new graphics object from bitmap and rotate.

(v) Draw image back to bitmap form and return it.

Reduce Resolution Function

(i) Accepts image and integer data types as arguments.

(ii) Returns the image to its original resolution if percentage = 0.

(iii) Pass image to Rotate Image Function.

(iv) Create a temporary image from the rotated image.

(v) Create new bitmap from temporary image with new size based on input.

As shown in Figure 2, functions to adjust the resolution and rotation of the chess pieces have been added to the user interface, using a text box on the right-hand side of the form to accept entry of an integer between zero and ninety-nine. Once a valid number is input into the text box and the adjacent "Ok" button is pressed, the resolution of all pieces is decreased by the value entered, as a percentage, and a rotation between −35 degrees and +35 degrees is applied to each piece individually. This will result in all pieces having the same resolution reduction, but a different rotation for each piece (see Figure 3).

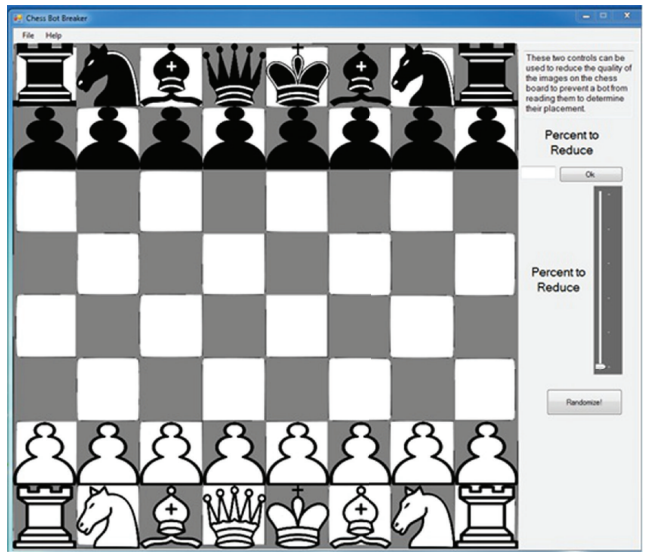

Figure 2: Standard chess board layout.

Track Bar Function

(i) Track bar minimum is 0, maximum is 5, increments by 1.

(ii) Generate random number between 5 and 10 and multiply it by track bar value.

(iii) Pass image and random number to Resolution Reduction Function.

(iv) Repeat steps 2 and 3 for each image to give every piece a different rotation and reduction.

Below the text box is a track bar that can be used to increment the distortion of the pieces in a slightly different way. The track bar consists of six values, starting with zero at the bottom and incrementing by one to five at the very top notch. The track bar starts at zero by default. When incremented, the track bar value is multiplied by a random number, labeled randNum, between five and ten, and passed on to the resolution reduction function. A new random number is generated for each chess piece, giving a certain amount of randomness to the resolution reduction of each individual piece. Incrementing the track bar increases the value to be multiplied by the random number, somewhat guaranteeing an increase in distortion as the track bar is incremented.

As illustrated in Figure 4, when the "Randomize!" button below the track bar is selected, the program rearranges the back row of both team's pieces according to the rules set forth for Fischer Random Chess. Another function, RandomResolution, is called to distort the images as well.

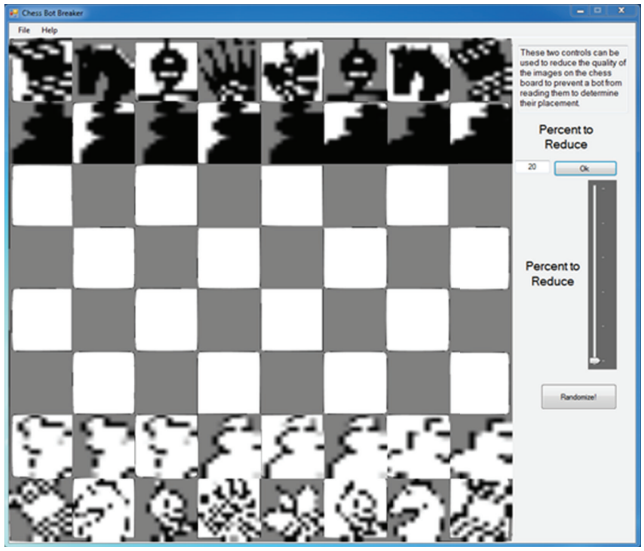

Figure 3: Distortion using the text box.

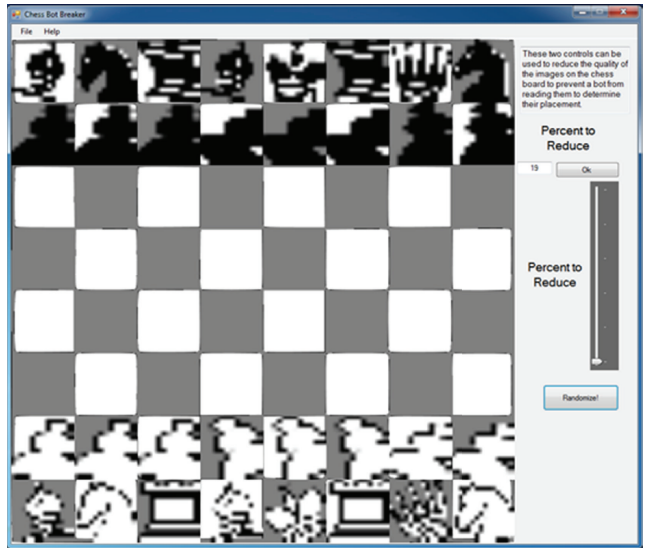

Figure 4: Fischer Random Chess.

A random value, between five and thirty percent, is chosen for resolution reduction and applied to every piece on the board. Rotation is again applied to each piece individually. The "Randomize!" button may be pressed as many times as desired; however, once a piece has been moved, the game is officially started and the button is disabled. Distortion of the pieces can still be controlled via the other two methods at any point during the game.

Fischer Random Chess Function

(i) Assign const integers to pieces (example: EMPTY = 0, KING = 1, QUEEN = 2, etc.).

(ii) Create two lists to keep track of empty spaces in the back row, one for odd and one for even spaces.

(iii) Create an array to hold piece positions in the back row.

(iv) Generate a random number between 1 and 6 to place the KING.

(v) Place KING into back row array at index just generated.

(vi) Generate 2 random numbers for placing ROOK. These must be between 0 and KING index, KING index and 7.

(vii) Place ROOK into back row array, 1 at each index just generated.

(viii) Update even and odd lists so no pieces are placed on occupied spaces.

(ix) Generate random number between 0 and even list size.

(x) Place BISHOP into back row array at index just generated.

(xi) Generate random number between 0 and odd list size.

(xii) Place BISHOP into back row array at index just generated.

(xiii) Update odd and even lists.

(xiv) Consolidate odd and even lists into 1 empty spaces list since no remaining pieces have an odd or even requirement.

(xv) Generate random number between 0 and empty spaces list size.

(xvi) Place QUEEN into back row array at index just generated.

(xvii) Update empty spaces list.

(xviii) Place KNIGHT into back row array at last 2 remaining indices.

5. Results

Each user filled out a feedback form during testing, as seen in Table 1, and the results were consistent. The form was used to collect some background information from 11 players to get an idea of their skill level at chess and determine how long it took them to adjust to the distorted pieces. Adjusting to the altered appearance of the pieces took most users a few seconds regardless of their skill level; after an average of four moves, none of the players had any trouble differentiating between pieces. Very few mistakes were made by the players. Some users did mention that additional changes could be implemented to make distinguishing pieces easier. For example, a letter representing the piece could be added to the image in a distorted way as well.

Humans are much better than computers at identifying patterns in an image [14]. Most modern text-based CAPTCHAs rely on letters which have been distorted. Therefore, distorting and rotating an image should prove very difficult for a bot to recognize. Including additional distortion effects will increase the difficulty of programming

Table 1: Feedback form embedded non-interactive CAPTCHA for Fischer Random Chess. Adjust the distortion to desired level. Try playing the game for at least 5–10 minutes. Feedback form used to gather information about the difficulty of the CAPTCHA tests.

What level of distortion was applied? (e.g., 17% or trackbar tick number 2)			
Your gender	☐ Male	☐ Female	
Your age			
Chess skill (beginner, intermediate, advanced)	☐ Beginner	☐ Intermediate	☐ Advanced
How many times have you played online chess?	☐ 0	☐ <10	☐ >10
How recently have you played online chess?	☐ Less than a month ago	☐ 1–6 months ago	☐ >6 months ago
On a scale of 1–10, with 10 being very difficult, how difficult was it to recognize pieces after distortion?			
Approximately how long did it take to get used to the look of the distorted pieces?	☐ Less than 3 moves	☐ 4–7 moves	☐ >8 moves
Have you ever played Fischer Random Chess?	☐ Yes	☐ No	
Comments			

a bot to read the pieces. For example, a skewing function would further help to prevent recognition. Adding background noise to each individual image would also provide more distortion for a bot to overcome, as well as changing the size of each piece slightly.

However, there are weaknesses to this approach to bot prevention. If there is only one available set of images for the chess pieces, then a bot simply has to compare the known images to distorted ones and make a guess based on similarity. This method could be fairly accurate, so it would be important to include multiple sets of images that can be used. Tracking piece movements could also give a bot clues as to what the pieces are; the bot may be fooled at the start, but as the game progresses, the movement trail left by the opponent may allow the bot to identify the pieces and resume control of the board.

The feedback, gathered from players with skill levels ranging from beginner to advanced, is promising. Players rated the difficulty of reading the pieces on a scale of one to ten, with an average rank of 6.5. However, the average number of moves to adjust to the distorted pieces was only 3.75, indicating that although this approach has a direct impact on the game experience, the user's ability to play the game is not hindered. The relatively high level of difficulty to read the pieces can be seen as a good indicator that a bot will have a hard time determining what the pieces are as well. The low number of moves to adjust to the distortion is a good sign that the player's experience will not be affected a great deal.

6. Conclusion

The program is designed to prevent a bot's ability to read a chess board, which renders the bot harmless and unable to suggest or make moves for the opposing player. This is accomplished by altering the visual aspects of the chess pieces on the board via user-controlled changes in resolution and/or rotation of the pieces; this skewing makes the pieces unrecognizable by a bot, while allowing human players to identify the pieces. With the added ability to play the game using the rules of Fischer Random Chess, a bot's inability to read the piece positions would prevent unfair advantages.

Additional research into bot prevention is clearly needed. One path may be further altering the area of CAPTCHA tests. For example, added altering of visual elements—such as skewing or stretching—could be tested. Other user-interface changes could include multiple image sets for swapping. These options continually evolve as standard CAPTCHA research moves forward. Additionally, although distorted audio or program-initiated questioning are alternatives to standard CAPTCHA tests, those methods could not be used with chess pieces. However, some of these methods are very creative; it is possible future research could find a way to incorporate an alternative method into the game of chess.

Breaking down a CAPTCHA is not always considered a total loss, as there are some positives that arise from it. For one, a weakness in the CAPTCHA has to be exposed, which can be fixed to strengthen the test in future revisions. Also, it is important to note that programming a bot to break a CAPTCHA test can be considered an advancement in Artificial Intelligence, as a bot has to try to emulate how a human would think in order to pass the test. This kind of competition is very important for promoting advancement in the fields of both artificial intelligence and security.

References

[1] C. E. Shannon, "Programming a computer for playing chess," *Philosophical Magazine*, 1950.

[2] Rybka, for the serious chess player, http://rybkachess.com/.

[3] IBM Research, Deep Blue. 1997, http://www.research.ibm.com/deepblue.

[4] R. V. Yampolskiy and V. Govindaraju, "Embedded noninteractive continuous bot detection," *Computers in Entertainment*, vol. 5, no. 4, article no. 7, 2008.

[5] S. Mitterhofer, C. Kruegel, E. Kirda, and C. Platzer, "Server-side bot detection in massively multiplayer online games," *IEEE Security and Privacy*, vol. 7, no. 3, pp. 29–36, 2009.

[6] S. Gianvecchio, Z. Wu, M. Xie, and H. Wang, "Battle of Botcraft: Fighting bots in online games with human observational proofs," in *Proceedings of the 16th ACM Conference on Computer and Communications Security (CCS '09)*, pp. 256–268, November 2009.

[7] D. Kushner, "On the Internet, nobody knows you're a Bot," *Wired Magazine*, p. 13, 2005.

[8] Ethical cheating in online chess. 2007, http://amirbagheri.virtuaboard.com/t34-ethical-cheating-in-online-chess.

[9] E. Bursztein, M. Matthieu, and J. Mitchell, "Text-based CAPTCHA strengths and weaknesses," in *ACM Computer and Communicatino Security*, Chicago, Ill, USA, 2011.

[10] P. Golle and N. Ducheneaut, "Preventing Bots from playing online games," *ACM Computers in Entertainment*, vol. 3, no. 3, 2005.

[11] D. Bushell, "In search of the perfect CAPTCHA," *Smashing Magazine*, 2011.

[12] D. Bethea, R. Cochran, and M. Reite, "Server-Side verification of client behavior in online games," in *Proceedings of the 17th Annual Network and Distributed System Security Symposium of the Internet Society*, San Diego, Calif, USA, 2010.

[13] J. Devlin, "How I built a working poker Bot," Part 1. 2008, http://www.codingthewheel.com/archives/how-i-built-a-working-poker-bot.

[14] J. Strickland, "How CAPTCHA works," 2011, http://computer.howstuffworks.com/captcha.htm/printable.