Automatic Design of Balanced Board Games

Vincent Hom, Joe Marks

Harvard University Extension School
Cambridge, MA 02138
Contact: joseph.w.marks@gmail.com

Abstract

AI techniques are already widely used in game software to provide computer-controlled opponents for human players. However, game design is a more-challenging problem than game play. Designers typically expend great effort to ensure that their games are balanced and challenging. Dynamic game-balancing techniques have been developed to modify a game-engine's parameters in response to user play. In this paper we describe a first attempt at using AI techniques to design balanced board games like checkers and Go by modifying the rules of the game, not just the rule parameters. Our approach involves the use of a commercial general game-playing (GGP) engine that plays according to rules that are specified in a general game-definition language. We use a genetic algorithm (GA) to search the space of game rules, looking for turn-based board games that are well balanced, i.e., those that the GGP engine in self-play finds equally hard to win from either side and rarely draws. The GA finds better games than a random-search strategy that uses equivalent computational effort.

Introduction & Overview

The earliest examples of computer-controlled game play were AI programs that were specific to a particular game like chess. More recently, General Game Playing (CGP) has emerged as its own area of research: CGP software is capable of playing any game whose rules are specified in a given scripting language (Pell 1992, Reference B, Genesereth 2005). Much research has gone into making both specific and general game-playing software as capable as possible.

However, there is more to making a game interesting than just providing a capable opponent. The qualities of the game itself are important. In particular, most good games are balanced in the sense that they provide just enough challenge to make play enjoyable, but not so much that play is frustrating (Adams 2007). The notion of using AI techniques to dynamically change game-play parameters to achieve dynamic game balancing (DGB) has recently been investigated by several research teams (Demasi 2002, Hunicke 2004, Spronck 2004, Andrade 2005).

Another way to automatically achieve balance in computer games is to not just adjust the game parameters, but to change the actual rules of the game. We call this Automatic Game Design (AGD). AGD has not yet been studied systematically. In 2004 it was reported in the press that Jim Lewis, an inventor from New Jersey, had used a computer to develop an especially difficult form of the well-known sliding-block puzzle (Reference A). Lewis computed the trees of all possible moves from the initial configurations of multiple candidate puzzles. The puzzle that produced the broadest and tallest tree he dubbed *Quzzle*, which can now be bought at various sites on the Internet. However, Lewis's general approach of comparing games by considering the shape and size of the corresponding move trees is not a practical approach for AGD, nor is it backed by any theory equating game quality purely to the size of its search space. Puzzle aficionados consider *Quzzle* to be less interesting than many other sliding-block puzzles that were designed by hand (Pegg 2004).

In contrast, our approach to AGD is to generate well-balanced board games that are won evenly by both the first-moving and second-moving players and that result infrequently in a draw. Our work is based on a commercially available GGP called Zillions of Games (ZOG). ZOG is a universal gaming engine that allows the creation of many kinds of board games (Reference C). Games rules are specified in a custom scripting language. A collection of scripting commands is called a Zillions Rules File (ZRF). There are three major components of a ZRF: the board, the pieces, and the victory condition.

The board definition specifies the players, turn order, the board dimensions, the number of pieces that each player has at the beginning of the game, and the initial placement of pieces on the board. The piece definition characterizes the allowable behavior of game pieces. Here each piece's movement options are enumerated. Last, the victory condition defines the possible outcomes of the game. Here is an example of a ZRF for a simple game, Tic-Tac-Toe:

```
(players White Black)
(turn-order White Black)
(board
  (image "images\TicTacToe\TTTbrd.bmp")
  (grid
    (start-rectangle 16 16 112 112) ; top-left position
      (dimensions ;3x3
        ("a/b/c" (112 0)) ; rows
        ("3/2/1" (0 112))) ; columns
        (directions
           (n 0 -1) (e 1 0) (s 0 1) (w -1 0)
          (ne 1 -1) (nw -1 -1) (se 1 1) (sw -1 1))))
(board-setup
  (White (disk off 10))
  (Black (disk off 10)))

(piece
  (name disk)
    (image White "images\Reversi\WDisk.bmp"
           Black "images\Reversi\BDisk.bmp")
  (drops (add-to-empty)))

(define add-to-empty  ((verify empty?) add) )

(draw-condition (White Black) stalemated)

(win-condition (White Black)
  (or  (relative-config disk n disk n disk)
       (relative-config disk e disk e disk)
       (relative-config disk ne disk ne disk)
       (relative-config disk nw disk nw disk)))
```

Searching for Balanced Games

Our approach to AGD is to search for balanced games in the space of board games that can be described by combining independent elements for board, piece, and victory specifications that are derived from known games and variations thereof. A search algorithm combines different elements to form new games that are then tested in self-play using the ZOG game engine. As an approximation to what constitutes challenge and interest, we seek games in which each player has roughly equal chances and that rarely end in draws.

Even for a simple game space like the one described below, exhaustive search of all possible games is not feasible. We therefore use a genetic algorithm (GA) to perform a more-efficient search (Holland 1975, Goldberg 1989). Because our GA is searching in a space of game rules or programs, it can be thought of as a form of genetic programming (Koza 1992).

Search Space. The game elements we consider are those that are used in common games like Checkers, Reversi (also known under the brand name of Othello), Tic-Tac-Toe, and some of their simple variants. The different elements are:

8 board types : 3x3, 4x4, 5x5, 6x6,7x7, 8x8, 9x9, 10x10
3 piece types : Tic-Tac-Toe, Reversi, Checkers
6 victory conditions: 3 in row, 4 in row, 5 in row, 6 in row, most men win, no men remaining

A total of 144 different games can be generated from these traditional elements alone. Additional complexity is achieved by incorporating nontraditional game elements that we specified by hand. These elements can be separated into two categories: elements that can be applied once and elements that can be applied repeatedly.

The six game elements that can only be applied once are: *Recall Last Move*, *Recall Any Move*, *Move For You*, *Double Move*, *Scoring*, and *First Hybrid Disk*. The remaining two elements that can be applied multiple times are *Hybrid Disk* and *Non-Flipping Disk*. We imposed a limit of no more than two nontraditional elements per game. A total of 5,616 games can therefore be specified using combinations of the traditional and nontraditional game elements. The nontraditional elements are:

- *Recall Last Move* allows a player to take back his opponent's last move and force him to make a different move.
- *Recall Any Move* is similar to *Recall Last Move*. The difference is that a player can recall any of his opponent's moves up to that point in the game.
- *Move For You* allows a player to make a move for his opponent once during the course of game.
- *Double Move* allows a player to perform two moves in succession once during the game.
- The mutation of *Scoring* decides the winner of the game by counting the number of tokens that each player has in a row. For scoring, six in a row is awarded six points, five in a row is awarded five points, etc.
- *First Hybrid Disk* is a single special piece given to each player that counts as a piece for both players. The disk must be the first piece used by each player.
- *Hybrid Disk* is similar to *First Hybrid Disk.* The difference is that a player can use this disk any time during the course of the game, but the hybrid disk must be used in order for the player to win.

- *Non-Flipping Disk* is a disk that cannot change color or ownership through capture (for those games that have such a concept).

Search Algorithm. Our search algorithm is a simple genetic algorithm (GA) that iteratively refines a population of games. To generate a new child game, three parent games are randomly chosen. Each one donates the board, the pieces, and the victory condition, respectively, to the definition of the new child. In addition, mutations -- one of the eight nontraditional game elements from the previous section – can be inherited from the parent games or randomly added to the child game. If the new child game has not been generated previously, it is tested by self-play with the ZOG game engine. The new game is played 100 times with a time limit of one second per move and the results are used to generate a fitness score. The fitness score is the sum of *balance* and *diversity* terms. The formula for the balance term is:

$$100 - (|\text{Player A wins} - \text{Player B wins}| - \text{Draws})$$

Thus a game that conferred no advantage to the first or second mover and that generated no draws would get a maximum balance score of 100 points. The diversity value is the number of unique differences that a new game has relative to its closest neighbor in the game population. Its inclusion in the fitness function promotes a broader search of the game space. The maximum diversity score is 11 points: two games could differ in all three basic game elements and in all eight nontraditional game elements.

If the fitness score of a new game is higher than the score of the least-fit member of the game population, then the new game replaces the least-fit game, otherwise it is ignored.

In our experiments, the initial game pool was populated with 32 traditional games like Tic-Tac-Toe, Reversi, Checkers, and minor variants thereof. We chose these games by hand to take advantage of what is already known to be good: these games have evolved through human play over the ages. The GA starts by determining the fitness score for each game in the initial population. In our tests we then ran the GA for 500 iterations, which required several hundred hours of computation on a PC cluster.

To measure the effectiveness of the GA, we compared it to a random sampling strategy. The results are reported in the next section.

Results

Table 1 describes the five games from the initial population with the highest fitness scores. ("Bal" denotes balance; "Div" denotes diversity.) Since these games mostly follow rules that were designed and tested by people over time, it is not surprising that some of them score quite well.

Game	Bal	Div	A	B	Draws
Reversi_10x10	96	2	50	47	3
Reversi_8x8	81	1	53	40	7
Checkers_10x10	64	0	38	32	30
Reversi_6x6	59	1	29	70	1
Checkers_8x8	49	1	24	31	45
Average	69.8	1	38.8	44.0	17.2

Table 1. The five fittest games in the initial population for the GA.

The results from 500 iterations of the GA are shown in Table 2. They are a clear improvement over the starting population of games. They are also significantly better than the best of 500 randomly generated games, as shown in Table 3. Note that random sampling fails to match the hand-picked games used to seed the GA, let alone the final results achieved by the GA: most games in the space defined by our game elements are not very interesting.

Game	Fit	Div	A	B	Draws
GameA	95	1	47	50	3
GameB	91	1	45	47	8
GameC	91	1	45	53	2
GameD	87	1	50	43	7
GameE	63	2	34	31	35
Average	85.4	1	44.2	44.8	11

Table 2. The five fittest games in the population after 500 iterations of the GA.

Game	Fit	Div	A	B	Draws
SampleA	70	0	35	60	5
SampleB	68	0	34	64	2
SampleC	66	0	33	64	3
SampleD	62	0	31	68	1
SampleE	61	1	30	70	0
Average	65.4	0.2	32.6	65.2	2.2

Table 3. The five fittest games from 500 random samples of the game space.

The behaviors of the GA and random-sampling strategy over time are shown in Figures 1 and 2. Figure 1 plots the average fitness score over time of both approaches, and Figure 2 the average diversity. As can be seen from the plots, the GA appears to be reaching a plateau after 500 iterations.

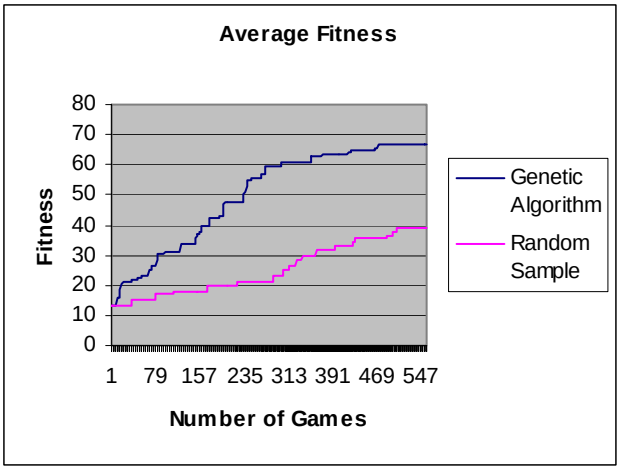

Figure 1. Average fitness scores over time of the GA and random-sampling algorithms.

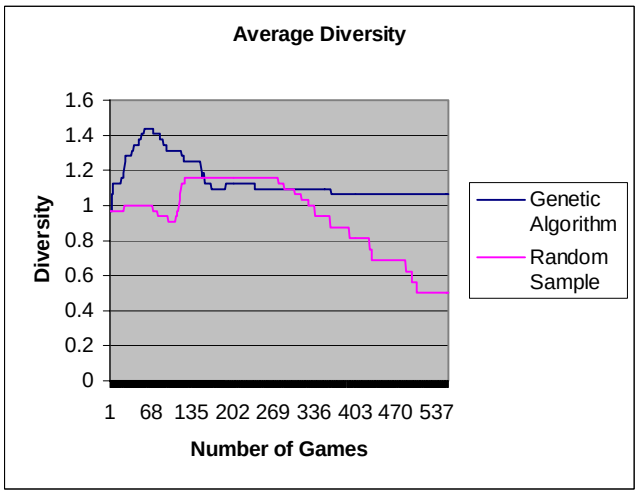

Figure 2. The average diversity of the games generated by the GA and random-sampling approaches over time.

The best games generated by the GA are described next:

Game A
Fitness: 95 Balance: 94 Diversity: 1
A 5x5 board using Go-Moku pieces. The objective is to get 3 men in a row. The mutation of recalling any one of the opponent's moves can be performed once only by each player. Although this game has a high fitness value, it is undeserved. Victory is trivial because the first player to move can win the game in less than five moves. Using ZOG to test this game, each computer-controlled player was given one second to complete a move. This length of time is insufficient for the computer to find the decisive course of action. So given the time limit of one second, the game is a balanced game. However, giving ZOG a time limit of three seconds, the game becomes a lopsided win for the opening player. This is an example of the oft-noticed ability of GAs to find anomalous or buggy solutions!

Game B
Fitness: 91 Balance:90 Diversity:1
A 6x6 board using Reversi pieces. The game is like Reversi only the objective is to have no pieces on the board, or to stalemate your opponent. To help clarify the rules of the game, it is helpful to consider the victory condition:

(or (loss-condition (White Black) (pieces-remaining 0)
(loss-condition (White Black) Stalemate))

Thus a player loses if he has no moves or he is the last player to place a disk on the board. In self-play, ZOG's search strategy tends to favor the first condition by trying to minimize the number of pieces on the board. Nevertheless, it's still sometimes possible to win via the stalemate criterion, so having fewer pieces doesn't guarantee a win.

Game C
Fitness: 91 Balance: 90 Diversity: 1
This game uses an 8x8 board and Reversi pieces. The objective is to get 5 pieces in a row. Each user is given one non-flipping disk to use during the game. This piece remains the same color throughout the game.

Game D
Fitness: 87 Balance: 86 Diversity: 1
This game uses a 10x10 board and Reversi pieces. The goal is to have the most pieces on the board at the end of the game. As in Game C, each player has one non-flipping disk.

Game E
Fitness: 63 Balance: 62 Diversity:1
A 10x10 board using Tic-Tac-Toe pieces. The goal is to attain 5 pieces in a row. Recalling the opponent's last move can be performed once only. This nontraditional game rule allows a player to undo his opponent's last move and force another course of action. Introduction of this rule reduces the number of draws in ZOG self-play by half for this game.

Conclusion

The use of AI techniques to design challenging and interesting games is a relatively new line of inquiry. Initial investigations have focused on dynamic modification of game parameters to match the computer's playing ability to that of its human opponent. In this paper we propose the idea of automatically designing the rules of a game to

make it intrinsically more interesting. We have shown how novel games can be tested through self-play with a general game-playing engine using a very simple notion of challenge and interest: how much advantage moving first of second confers, and how likely the game is to be drawn. And we have shown how a GA can be used to find games that are optimal with respect to these criteria.

Our work could be extended in many ways. First, the fitness function could incorporate a broader notion of challenge and interest. For example, we could give ZOG more time to compute the moves for one side and then measure how well this extra time translates to improved results: the more interesting the game, perhaps the greater the advantage conferred by having more time to think. (Thanks to an anonymous reviewer for this suggestion.) Alternatively, the number of times in which the initiative shifts from one player to another over the course of a game could be an important characteristic: this could be captured by the number of zero crossings in the derivative of the game engine's scoring function over time. Or like Lewis's work on *Quzzle* (Reference A), characteristics of the game search tree could factor into the fitness function. In particular, we would like to have used a dynamic search strategy in the game engine that continues searching until a quiescent position is reached: this might allow us to detect anomalous games that appear to be balanced only because the game engine could not find a winning sequence of moves (e.g., Game A from our experiment). However, exploring such ideas will require access to the source code for a general game-playing engine, which we did not have for the ZOG game engine.

A second direction to explore is the automatic design of new game elements. In our work all the game elements were inherited from known games or designed by hand; the GA was used to find novel combinations of these game elements. A more-fundamental approach would involve the automatic design of new boards, pieces, or victory conditions, which is a much harder problem.

A third direction is to investigate the automatic design of game rules for other than simple board games. As one anonymous reviewer noted, automating the design of balanced card games could be a logical next step. The commercial payoff of this research is likely to lie in the automatic design of game rules for strategy games like *Civilization* or *SimCity*. Ralph Koster's ideas for game grammars (Reference D) could be a useful starting point for this line of inquiry.

Finally, the true test of this approach will be to follow Andrade et al. in performing human experiments to determine if people at different skill levels find the automatically designed games challenging and interesting (Andrade 2006).

References

A. A Hard, Simple Problem. 2004. http://www.economist.com/science/displayStory.cfm?story_id=3445734. *The Economist*.

B. http://games.stanford.edu/

C. http://www.zillions-of-games.com/

D. http://www.theoryoffun.com/grammar/gdc2005.htm

Adams, E., and Rollings, A. 2007. *Fundamentals of Game Design*. Prentice Hall.

Andrade, G., Ramalho, G., Santana, H., and Corruble, V. 2005. Challenge-Sensitive Action Selection: an Application to Game Balancing. *Proc. of the IEEE/WIC/ACM Intl. Conf. on Intelligent Agent Technology (IAT-05)*, pp. 194-200, Compiègne, France.

Andrade, G., Ramalho, G., Gomes, A., and Corruble, V. 2006. Dynamic Game Balancing: an Evaluation of User Satisfaction. *Proc. of the 2nd Artificial Intelligence and Interactive Digital Entertainment Conf. (AIIDE'06)*, Marina del Rey, CA.

Demasi, P., and Cruz, A. 2002. Online Coevolution for Action Games. *Proc. of The 3rd Intl. Conf. on Intelligent Games And Simulation*, pp. 113-120, London.

Genesereth, M. R., Love, N., and Pell, B. 2005. General Game Playing – overview of the AAAI Competition. *AAAI Magazine* (26)2: 62-72.

Goldberg, D. 1989. *Genetic Algorithms in Search, Optimization, and Machine Learning.* Addison-Wesley.

Holland, J. 1975. *Adaptation in Natural and Artificial Systems.* University of Michigan Press.

Hunicke, R., and Chapman, V. AI for Dynamic Difficulty Adjustment in Games. 2004. *AAAI Workshop on Challenges in Game Artificial Intelligence*, pp. 91-96, San Jose.

Koza, J. 1992. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press.

Pegg, E. 2004. Sliding-Block Puzzles. At http://www.maa.org/editorial/mathgames/mathgames_12_13_04.html, Mathematical Association of America.

Pell, B. 1992. Metagame: A New Challenge for Games and Learning. In van den Erik, H.J. and Allis, L.V., eds., *Heuristic Programming in Artificial Intelligence 3 – The Third Computer Olympiad*. Ellis Horwood. Related papers can be found at ftp://ftp.cl.cam.ac.uk/users/bdp.

Spronck, P., Sprinkhuizen-Kuyper, I., and Postma, E. Difficulty Scaling of Game AI. 2004. *Proc. of the 5th Intl. Conf. on Intelligent Games and Simulation*, pp. 33-37, Belgium.