Artificial Intelligence in Gaming

Conference Paper · April 2010

CITATIONS
0

READS
223

5 authors, including:

Jeetendra Nihalani
1 PUBLICATION **0** CITATIONS
SEE PROFILE

Rohan Narang
1 PUBLICATION **0** CITATIONS
SEE PROFILE

Shalini Bhatia
University of Canberra
41 PUBLICATIONS **278** CITATIONS
SEE PROFILE

Artificial Intelligence in Gaming

Jeetendra Nihalani, Gokul Chandrasekaran, Rohan Narang, Mahsheed Eshraghi, and Shalini Bhatia
Computer Engineering Department
Thadomal Shahani Engineering College
Mumbai, India
jeetnihalani@gmail.com, chandrasekaran.gokul@gmail.com, rohan_narang15@hotmail.com,
mahsheed.eshraghi@gmail.com, shalini.tsec@gmail.com

***Abstract*—The game Theseus and Minotaur requires a player to help maneuver Theseus through a maze to an exit without being captured by Minotaur. For every step Theseus takes, Minotaur takes two using a fixed algorithm. It is time consuming and requires a lot of previous experience for a human to generate solvable mazes of an adequate difficulty level. This paper explains the modifications made to an already existing technique to generate these mazes using genetic algorithms. The technique is extended to help create mazes for a modified version of the game that includes an extra element called moving blocks.**

Keywords-artificial intelligence; genetic algorithms; automation; Theseus and Minotaur

I. INTRODUCTION

Existing for centuries games have evolved with the people that play them. In this age of computers, remarkably intelligent game-play exists. The computer can now predict what move you will make next and counter that move. Techniques of artificial intelligence (AI) have influenced many fields of game play from the intelligence of the opponent to the creation of different levels of difficulty for users to enjoy.

There are a variety of techniques that can be used to incorporate AI in games. One of the most elementary forms of AI uses uninformed search to come to a solution. These can broadly be categorized into breadth first search and depth first search. These algorithms use knowledge which is limited to the definition of the problem [1]. As an example, the minimax algorithm uses a depth first search to find the next best move for the computer. For the breadth first search technique memory requirements are a bigger problem than execution time. The time required to perform uniformed search increases exponentially with the size of the problem.

Informed search goes one step further to use problem specific knowledge beyond the definition of the problem. It uses a heuristic function to find solutions more efficiently than an uninformed strategy.

Informed search requires the computer to keep many nodes on the frontier in memory, and requires the programmer to choose a good heuristic function, which is not easy to formulate for many problems [2].

Rule-based systems are highly inefficient. In order to use just one rule, the system must examine all the rules. Despite the best intentions, hidden dependencies between rules may exist. This makes it hard to predict the effects of adding or deleting a rule [3].

Neural networks are also being used in gaming. The drawbacks of neural networks are that it is difficult to understand and, at times, it is difficult to predict what the output of a neural network will be. The individual relations between the input and the output variables are not developed by engineering judgment. Hence, the model tends to be a black box or input/output table without analytical basis [4].

Yet another technique to simulate intelligence in gaming is the use of genetic algorithms. This paper illustrates how it can be used in developing mazes for the puzzle game Theseus and Minotaur.

II. THE PROBLEM

A. The Game

Theseus and Minotaur is a puzzle that has been tremendously popular across the internet [5]. It is based on a Greek mythology in which there exists a beast, half man and half bull, called Minotaur who lives in a cave. The story goes that in the entire land there existed only one youth brave enough to tackle Minotaur. This youth, named Theseus, valiantly went into the cave and killed Minotaur.

In the computer version of the game, Theseus is represented as a black dot in a maze and Minotaur as a red one. In this game, invented by Robert Abbott, the human player is given control of Theseus. The player can move up, down, left or right unless the direction he wishes to move is obstructed by a wall. For each move that is made by the player, Minotaur, the computer, makes two. In order to compute each of the two moves Minotaur must make, the computer follows the same procedure. It first checks if Minotaur will get closer to Theseus if it moves horizontally. If this is true and there is no wall obstructing Minotaur, the computer plays this move. Otherwise, the computer checks if Minotaur will be closer to Theseus if it moves vertically. If this is true and there is no wall obstructing Minotaur, the computer plays the move. Otherwise, the computer is forced to remain in its position.

Armed with the knowledge of how Minotaur will move, a player is expected to maneuver himself to a designated exit without being caught by Minotaur. The algorithm that Minotaur follows allows the player to trap it, in strategic locations on the map, behind walls.

B. Maze Generation

The creation of mazes which require a player to think more and hence are difficult to play requires a lot of time and clever thinking on the part of a human designer. It is desirable to have this task automated so as to produce a variety of difficult mazes in a relatively short period of time. A brute force technique to generate mazes is

unsuitable since the search space, all possible mazes of a given size, is too large.

Various techniques have been used before to automate maze generation. Toby Nelson has designed a program which generates mazes randomly, then refines the best ones by testing various mutations. The mazes are ranked according to measures like solution length and the number of false paths. In his Java implementation of the game, Nelson, added fourteen of his own mazes. This new set of mazes was generally easier than Abbott's original, though each still required several small leaps of logic. Nelson's first thirteen mazes provided a steady ramp of difficulty for solvers before they reached Abbott's original. And Nelson added a final maze. This maze, called The Dread Maze Fifteen, was far more complex than Abbott's.

A similar program, designed by James W. Stephens, provides the basis for the many wonderful puzzles at PuzzleBeast [5].

Search techniques that move towards a maxima help reduce the number of possibities that are required be checked. Suchao Chaisilprungrueng and Hadi Moradi in the paper titled 'Designing a Theseus-Minotaur game using Genetic Algorithms' [6] have explained a technique used to automate maze generation. The technique of automated maze generation explained in this paper is different from that in the way that the fitness function and the process of selection of mazes are implemented.

III. GENETIC ALGORITHMS

Part of a larger group of algorithms called evolutionary algorithms the genetic algorithm technique is a search technique which mimics evolution. Given a problem, each possible solution to it is represented in the form of an individual in a population. Each individual in a population is given a fitness value based on its proximity to the required result. Individuals of this population are then used to produce the next generation of solutions. The next generation is created by selecting individuals with a probability directly proportional to their fitness, crossing them over and mutating the resultant individuals. With each passing generation the new set of solutions formed tends towards a good solution to the problem. The genetic algorithm thus finds a solution to a problem without checking every possible solution and hence reduces the number of states that are required to be searched.

A. Individual

In an implementation of a genetic algorithm, an individual is a possible solution to the problem statement. There are a variety of possible encodings of a solution. Traditionally, solutions are represented as strings of 0's and 1's, but other encodings are also possible [7].

In this project, an individual is one possible configuration of the maze. The starting position of Theseus, the starting position of Minotaur, the exit position, the configuration of horizontal walls and the configuration of vertical walls are encoded in a string of 0's and 1's. This string represents an individual.

The length of an individual of a maze of a given size is fixed. The total cells in a maze are given by:

*Total Cells = number of rows * number of columns*

The cells which are on the boundary of the maze consist of a wall or walls based on which boundary they are located (for example: the north bounding cell would consist of a north wall by default). These walls are not accounted for in the binary string since they must exist by default in every maze. Suppose there are 'r' rows and 'c' columns in a maze. The number vertical walls in a single row excluding the walls at the boundary can be given by (c-1). Therefore the total number of vertical walls is given by r*(c-1). Similarly the total number of horizontal walls can be calculated. Hence, the number of bits required to specify the horizontal and vertical wall locations can be calculated as follows:

*No. of bits for horizontal walls = number of columns * (number of rows-1)*

No. of bits for vertical walls = (number of columns-1) number of rows*

The number of bits that are needed to specify the locations of Theseus, Minotaur, and the exit are calculated based on the total number of cells in the maze. For example: if the total number of cells in a maze is 8 then each of these positions can be specified by 3 bits. In case there are totally 6 cells, a minimum of 3 bits are needed to reference each of them distinctly. The strings '110' and '111' are left unused as the required number of cells has already been referenced. A value of the log_2 of the total number of cells gives us the number of bits needed to represent any valid maze location.

Position length = ceil(log_2 (total number of cells))

Therefore, the total number of bits in the binary string is given by:

*Total No. of bits = 3 * (Position length) + No. of bits for horizontal walls + No. of bits for vertical walls.*

B. Population

Genetic algorithms are implemented by generating a random population of solutions to the problem and then evolving these solutions into better ones. Evolution is carried out till the desired solution is obtained or till a desired level of accuracy is achieved. In each generation, the fitness of every individual in the population is computed. Based on their fitness values multiple individuals from the current generation are selected, modified and placed in the next population.

In this project, the initial population is a set of randomly created mazes. Parameters called *probability of horizontal wall generation* and *probability of vertical wall generation* are used to create the horizontal and vertical walls, represented by 0's and 1's, of individual mazes in the initial population. The nature of the game dictates that horizontal traps – horizontal walls that allow Theseus to trap Minotaur – are easier to perceive while vertical traps – vertical walls that allow Theseus to trap Minotaur – are more difficult to identify. Hence, by starting with a population of solutions having a large number of vertical walls it is intended to generate final solutions with the more difficult vertical traps.

C. Fitness Function

The most important aspect to consider while implementing genetic algorithms is the fitness function. Fitness represents the quality of a solution. The fitness function is problem dependent and should be designed keeping in mind various factors that influence the solution of a problem.

To calculate the fitness of a maze, the following parameters are considered:

--The number of moves Theseus makes (a1): A difficult maze requires Theseus to make more moves to reach the exit while a lesser number of moves signify a simpler maze.

-- The number of moves Minotaur makes (a2): If Minotaur is stuck at a position for an extended period of time then it is not very difficult for Theseus to reach the exit. Hence, the fitness of a maze is directly proportional to this value. When a maze is generated such that Minotaur is trapped by walls on all four sides this parameter is 0 and hence, the fitness of the maze is low. This prevents such mazes from propagating to the next generation.

--The number of moves Minotaur is static (a3): This parameter gives a simple count of the number of moves that Minotaur is trapped for.

--The number of times Minotaur is trapped (a4): This parameter introduces an element of strategic planning on the part of Theseus as to where Minotaur must be trapped.

--The number of false paths (a5): A false path is a series of steps which if made, does not result in a solution. With a large number of false paths, a larger look ahead is required to determine if a solution using that particular approach is possible.

Based on the parameters stated above, the fitness of a maze is given by:

*Fitness function = k1 * a1 + k2 * a2 + k3 * a3 + k4 * a4 + a5 * k5*

where kx = k1, k2, k3, k4, k5 are proportionality constants which are a product of a corresponding scaling factor and importance of the given parameter. Thus, kx = importance (impx) * scaling factor (sx) where (x = 1, 2, 3, 4, 5)

A scaling factor is used in order to normalize the values of all the parameters. Consider a fitness function comprising of parameters *a1* and *a2*. Consider a maze, maze1 with a1 = 40 and a2 = 6 and consider another maze, maze2 with a1 = 30 and a2 = 9. If we consider only parameter *a1*, maze1 is better by a factor of 1.3333 (i.e. 40/30). Similarly, if we consider only *a2*, maze2 is better by a factor of 1.5 (i.e. 9/6). Thus if both these factors are equally important, maze2 is the better maze. It would be incorrect to say that maze1 has a higher fitness because its fitness is 40 + 6 = 46, which is greater than 30 + 9 = 39.

This ambiguity is addressed by normalizing each parameter by dividing it by the maximum value of the parameter found amongst all the individuals in the population. The normalized parameters can then be multiplied by another constant called importance to determine the fitness value of the individual. It is important to note that this fitness formula helps calculate the fitness of an individual in a given population. This value is useful only to select a given individual from the current population. If the fitness of the individual must be calculated when it is put in another population, the normalization of parameters must be done with respect to the maximum value of each parameter present in the new population.

D. Crossover

After calculating the fitness of each individual in a population, the next step is to generate the next generation of individuals. Crossover is performed between binary string representations of two mazes. A random position is selected from which the right substrings of both mazes are interchanged. If any of the resultant mazes is not a valid maze, the previous step is performed again. Crossover between individuals is carried out by selecting two individuals for each crossover. Those individuals with a higher fitness have a greater chance of being selected to be crossed over. This is based on the principle of the survival of the fitness seen in nature.

E. Mutation

Each new individual created by crossing two parents may or may not be mutated based on the *probability of mutation*. Five random bits in the binary representation are changed to produce a mutated individual which is then checked to see if it represents a legal maze. If the maze is legal, the mutation stops.

An example will demonstrate the meaning of a legal maze. Let the maze have sides of length six. Therefore, in order to represent each position on the maze we need 6 bits. Out of the 2^6 possible combinations, only 36 are valid. If, after mutation, there is a maze that results such that its string contains a binary representation of any position other than 0 to 35, it must be discarded. Such a maze is illegal.

F. Elitism

The two fittest mazes in a generation are transferred unchanged to the next generation. This ensures that the fitness of the best maze in the next generation is at least as high as its previous generation.

IV. MAZE SOLVER

In order to check if a maze is solvable and compute the values of the parameters used in the fitness function a maze solver is necessary. This module uses a version of breadth first search to find a solution to a maze. The first solution found using breadth first search is the shortest solution to the maze.

One of the parameters that the fitness function comprises of is the number of false paths – paths which lead to Minotaur capturing Theseus. In order to find the number of false paths it is required to consider all possible paths that the user can take. Since in order to find the number false paths all possible combinations must be considered, the A* algorithm is not applicable.

Breadth first search is used to find a solution. All possible moves that can be made at each cell position are considered. However, to avoid visiting previously considered states the program keeps track of the cell locations that Theseus and Minotaur have visited using the visited matrix.

Initially a queue contains the starting maze state. This is popped and the five possible moves (left, right, down, up, and no move) that Theseus can make are considered. The valid moves are put into the queue. The elements of the queue are then popped and evaluated in a similar manner. The algorithm terminates when the queue that stores the states that must be visited becomes empty. At termination either a solution is found and all false paths are exhausted or a solution is not found. Mazes for which solutions are not found are declared as unsolvable.

Since visited states are considered only once after which they are never put into the queue, the time complexity of the maze solver is proportional to the maximum number of possible states of the maze. A state is represented by the location of Theseus and Minotaur. Let 'n' be the number of cells in the maze which is equal to rows * columns. Theseus can take 'n' distinct positions and Minotaur can take 'n' distinct positions. Therefore, there are n * n number of distinct states that are evaluated in the worst case assuming that all cell positions are reachable. Thus, the time complexity of the maze solver is $O(n^2)$. Since all states must be evaluated to find the number of false paths, the time complexity is always $O(n^2)$.

As an example, consider a maze of 10 rows and 10 columns. It has a total of 100 cell positions or n = 100. Each time the maze solver is invoked for a maze with n = 100, there are 10,000 distinct states of the maze that will be checked.

Using the parameters found after an exhaustive breadth first search, the fitness of the maze is calculated.

V. RESULTS

Mazes generated using the following parameters are represented below.

Horizontal Length = 6
Vertical Length = 6
Constant Population Size = 100
Probability of Horizontal Wall Generation = 0.25
Probability of Vertical Wall Generation = 0.15
Elitism Implemented
Fitness = a1*k1 + a2*k2 + a3*k3 + a4*k4 + a5*k5
a1= number of player moves
k1 = 2 * 100 / max. value of number of player moves in population
a2 = number of enemy moves
k2 = 3 * 100 / max. value of number of enemy moves in population
a3 = number of static enemy moves
k3 = 1 * 100 / max. value of number of static enemy moves in population
a4 = number of blocks faced by enemy
k4 = 1.5 * 100 / max. value of number of blocks faced by enemy in population
a5 = number of false paths
k5 = 1 * 100 / max. value of number of false paths in population

Table I represents the values of the parameters that determine the fitness of the fittest maze of a given generation along with its fitness value. The fitness values presented in Table I were generated by placing the fittest maze of each generation in a separate population. After reaching the last generation, the fitness of the mazes in this population was calculated using the maximum values of each parameter in this population as the basis for normalization.

Figs. 1-6 show the mazes with the highest fitness values in their corresponding generations. It can be seen that the walls gradually change from generation 0 to generation 1760. The fitness maze of generation 1760 is the most evolved.

The improvement of the fitness of the fittest maze can be seen in Fig. 7 which graphically represents the fitness of the maze v/s the generation number.

VI. FURTHER WORK

This project has been modified to generate mazes for a modified version of Theseus and Minotaur. This version includes a concept of moving blocks which are blocks placed in different positions on the maze. With each move of Theseus and Minotaur they move along a predefined path. A block in a particular position prevents Theseus or Minotaur from moving to that position. Additionally, if a Theseus or Minotaur moves to a particular position and in the next move a block moves to that position, the corresponding player is killed. The movement of the blocks is periodical along fixed paths and therefore a player can look ahead and determine the moves needed to get to the exit. This implementation still requires testing.

Figure 1. TABLE I

Generation	Fitness of Fittest Maze	Player Moves	Normalized Number of Enemy Moves	Normalized Number of Static Enemy Moves	Normalized Number of Times Enemy must be Blocked	Normalized Number of False Paths
0	177.5	19	27	14	11	28
320	753.5	89	92	86	77	98
640	785	89	97	84	88	100
960	784.5	92	97	89	83	96
1280	793	96	97	97	94	72
1760	816	98	100	98	100	72

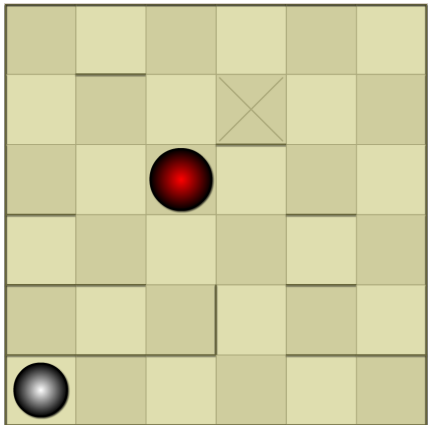

Figure 1. Fittest Individual in Generation 0.

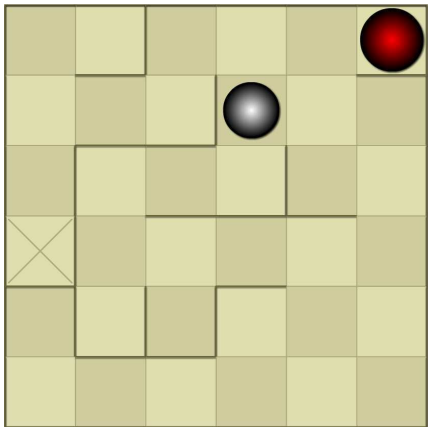

Figure 2. Fittest Individual in Generation 320.

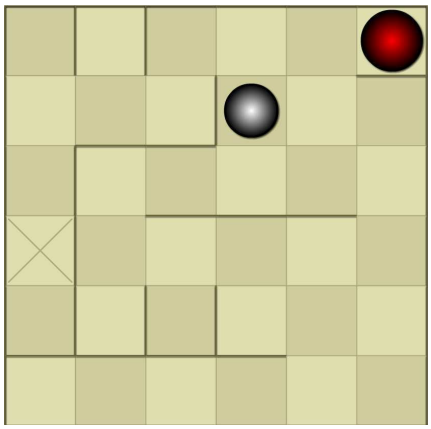

Figure 3. Fittest Individual in Generation 640.

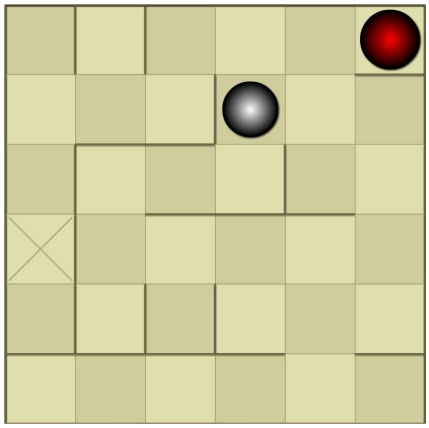

Figure 4. Fittest Individual in Generation 960.

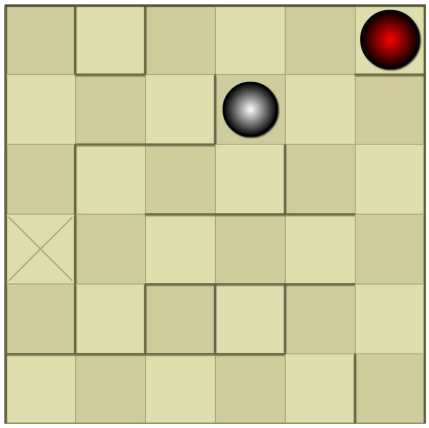

Figure 5. Fittest Individual in Generation 1280.

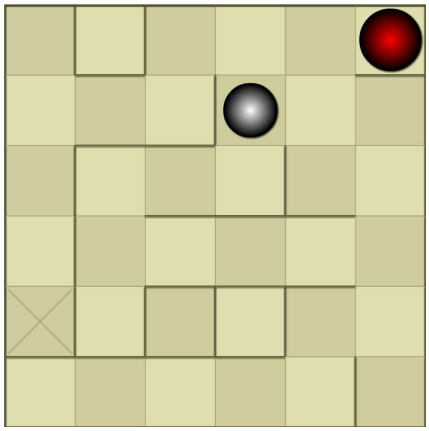

Figure 6. Fittest Individual in Generation 1760.

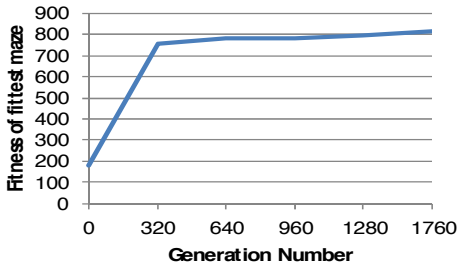

Figure 7. Graph showing the fitness of the fittest maze v/s the generation number.

ACKNOWLEDGMENT

Jeetendra C. Nihalani, Gokul Chandrasekaran, Rohan Narang and Mahsheed Z. Eshraghi thank Ms. Shalini Bhatia for having sincerely guided us during the implementation of this project. They also thank Mr. Robert Abbott, inventor of the game Theseus and Minotaur, for allowing the use of his game in the project and for having provided invaluable inputs related to the difficulty of mazes generated.

REFERENCES

[1] Russell, Stuart and Norwig, Peter, "Artificial Intelligence A Modern Approach," s.l.: Dorling Kindersley, 2009. p. 122.

[2] Danyluk, Andrea Pohoreckyj, "CS 108 - Lecture 32", *Computer Science at Williams College Web site,* [Online]. [Cited: October 21, 2009] http://www.cs.williams.edu/~andrea/cs108/Lectures/InfSearch/infSearch.html.

[3] "CS 2360," *Computer Science at Georgia Tech Web site,* [Online]. (May 30, 1996) [Cited: October 22, 2009.] http://www.cc.gatech.edu/computing/classes/cs2360/spring96/lec20.html.

[4] Vogt, Andrew and Bared, Joe G., "Accident Models for Two-Lane Rural Roads: Segments and Intersections." *US Department of Transportation - Federal Highway Administration Web site*. [Online]. (1998) [Cited: October 22, 2009] http://www.tfhrc.gov/safety/98133/ch02/body_ch02_05.html.

[5] Delgado, Tony "Tablesaw", "COLUMN: 'Beyond Tetris' - Theseus and the Minotaur / Mummy Maze," *Game Set Watch,* [Online]. (Febraury 12, 2007) [Cited: March 6, 2010] http://www.gamesetwatch.com/2007/02/column_beyond_tetris_theseus_a_1.php.

[6] Chaisilprungrueng, Suchao and Moradi, Hadi., "Computer Science: University of Southern California." *University of Southern California Web Site,* [Online]. [Cited: July 1, 2009] http://www.cs.usc.edu/Research/techreports/papers/03-800.pdf.

[7] "Crossiover (Genetic Algorithm)," *Wikipedia,* [Online]. [Cited: October 8, 2009] http://en.wikipedia.org/wiki/Crossover_(genetic_algorithm).