Self-Synthesized Controllers for Tower Defense Game using Genetic Programming

Leow Chin Leong, Gan Kim Soon, Tan Tse Guan, Chin Kim On, and Rayner Alfred
Center of Excellent in Semantic Agents,
Universiti Malaysia Sabah, 88400, Sabah, Malaysia
dragonjuly14@gmail.com, g_k_s967@yahoo.com,
tseguantan@gmail.com, kimonchin@ums.edu.my,
ralfred@ums.edu.my

Patricia Anthony
Department of Applied Computing, Faculty of Environment, Society and Design,
Lincoln University, Christchurch, New Zealand
patricia.anthony@lincoln.ac.nz

Abstract— **In this paper, we describe the results of implementing Genetic Programming (GP) using two different Artificial Neural Networks (ANN) topologies in a customized Tower Defense (TD) games. The ANNs used are (1) Feed-forward Neural Network (FFNN) and (2) Elman-Recurrent Neural Network (ERNN). TD game is one of the strategy game genres. Players are required to build towers in order to prevent the creeps from reaching their bases. Lives will be deducted if any creeps manage to reach the base. In this research, a map will be designed. The AI method used will self-synthesize and analyze the level of difficulty of the designed map. The GP acts as a tuner of the weights in ANNs. The ANNs will act as players to block the creeps from reaching the base. The map will then be evaluated by the ANNs in the testing phase. Our findings showed that GP works well with ERNN compared to GP with FFNN.**

Keywords- Artificial Neural Network (ANN); Genetic Programming (GP); Tower Defense (TD) Game; Feed-forward Neural Network (FFNN); Elman-Recurrent Neural Network (ERNN).

I. INTRODUCTION

Research on gaming is not a recent trend. It started since 1950s and still remains popular till today. There are several reasons that encourage this kind of research. Based on [1], human fascination with game playing is long-standing and pervasive. Besides, [1] also stated that some difficult games remain to be won, in particular games that people play very well but not the computers.

During the early stage, the researchers were focused on designing Non-Player Character (NPC) and path finding. The researches can be divided into two groups which are academic AI game research and industrial game AI [2]. There are lots of AI techniques that have been implemented in gaming area. The most popular algorithms are the A* Search algorithm [3], Finite State Machine [4, 5], Monte-Carlo Tree Search [6], Pareto-Archived Evolutionary Strategy [7], Reinforcement Learning [8], Pareto-based Differential Evolutionary algorithm [9], Genetic Algorithm [4, 10], Differential Evolution [11], Particle Swarm Optimization algorithm [12], and Evolutionary Programming [13].

However, different game genres require different techniques of AIs. For example, the A* algorithm is useful for Real Time Strategy games but it is not suitable for designing controllers in Role Playing games. Even though a lot of techniques had been studied, there are still rooms for improvement in gaming.

In this research, a hybrid Genetic Programming (GP) method and Artificial Neural Network (ANN) will be used in a Tower Defense (TD) game. There are many reasons that encourage TD game to be used as test bed. One important reason is that it is easy to code TD game. When dealing with complex programming, the optimization and testing have to be split into several stages and it would usually involve high cost when we consider the workloads and time taken to complete the task. The selection of TD game is also due to the complexity of its gaming environment. Even though TD maps can be easily designed, the game rules can be very complex, as it may involve various types of towers, different upgrade functions, various types of creeps, and paper and rock-paper-scissor technique (in this technique, no single type of tower can eliminate all types of creeps).

In a TD game, player needs to prevent the enemies from fulfilling their goals by building towers. The goal of the enemies is usually to enter the player's base or destroy the player's base. The gameplay is not very complicated. The game consists of certain levels. At the beginning of each level, creeps will be released to destroy the base. The strength of the creeps is proportionate to the number of levels. The creeps' movement speed and strength are increased after a certain level/wave. Player will be granted with certain amount of credits or resources that are required to build towers or to upgrade their towers in the later levels.

In this study, a modified GP will be implemented with (1) Feed-forward Neural Network (FFNN) and (2) Elman Recurrent Neural Network (ERNN) in a customized TD game in order to generate AI controllers that can be used in testing the designed map. Running such experiment helps to determine the suitable NN that best suit with GP. Furthermore, the GP will determine the level of difficulty of a generated map in the testing phases.

978-1-4799-1508-8/13/$31.00 ©2013 IEEE

In the next section, some related works will be discussed. Section III will briefly introduce FFNN and ERNN. The modified GP is discussed in Section IV. The experimental setup is described in Section V and the experimental results and discussions are described in Section VI. Finally, Section VII will summarize the findings and discusses future works.

II. Related Works In ANN and Gaming AI

The implementation of ANN in games shows highly promising results due to their ability to learn, adapt to new environments with different functions and their ability to predict based on learning data. [14] explained some important points when implementing ANN in games. ANN had been implemented in different gaming such as Reversi [15], board games [16, 17, 18], first person shooter game [19] and racing game [20]. The implementation of ANN helps to discover complex strategies that enable AI controller to compete with players. As a result, some AI controllers are unbeatable by experienced human players.

[21] describes a comparison of GP and Backpropagation (BP) ANNs in Othello game and in this work, it was found that GP produced better result compared to BP as BP predicted actions based on learning data which limits its prediction capability. In another setting, the GP was evolved based on learning data in different situations. Thus, the ability of the GP to learn was higher than the expectation. GP was also implemented to evolve programs to play the Mancala game [22] in which GP was able to successfully create the required Agents.

III. Artificial Neural Networks

ANN was first introduced by McCulloch and Pills Whor in 1943 [23]. It was a mathematical model that simulates the structure and functionalities of biological NNs [24]. ANN usually consists of three layers which are the input layer, the hidden layer and the output layer. It is possible to have multiple hidden layers in an ANN. Each layer consists of neurons that are linked to each other.

The first layer is the input layer where information is received. The value for the hidden layer is determined by the values of the input layer and the weights between the connections of input layer and hidden layer. There is no range on the weight. It can be either a positive value or a negative value. The weight is adjusted to train the learning data to produce a specific output and this adjustment is known as learning or training [19]. The output layer returns the result of the ANN model.

There are two major concerns that need to be considered when implementing ANN. The first concern is the type of problem to be solved as this affect the type of input and output required to be produced by ANN. Next is the type of topology used. There is no hard rule in determining which ANN topology is the best for different types of evolutionary algorithms. Hitherto, different NNs topologies are being implemented and being studied for various problems.

In this study, FFNN and ERNN will be implemented in order to study the effect of both topologies towards the implementation of GP. Both ANNs have three layers in their structure. However, FFNN are slightly different than ERNN in its learning process. FFNN involves straight propagation in its topology. The neurons at input layer will be directly passed to the hidden layer and from the hidden layer it is passed to the output layer.

For ERNN, the propagation process starts from the input layer to the hidden layer. A learning rule will be applied and there are connections from the hidden layer to the context units fixed in the input layer for the first propagation process. The flow of the hidden layer and the output layer for FFNN and ERNN are exactly the same. The learning for ERNN is a straight forward process from the hidden layer to the output layer. Fig. 1 shows the basic topology of FFNN and ERNN.

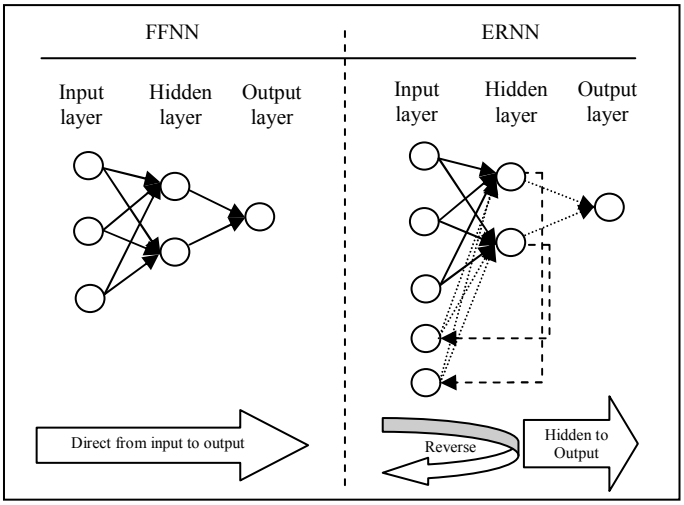

Figure 1. FFNN and ERNN topologies

IV. Genetic Programming

Genetic Programming (GP) is one of techniques in evolutionary algorithm which is based on Darwin's theory evolution [25]. GP has been implemented in different types of gaming such as chess [26], robocode [27] and backgammon [28]. GP and Genetic Algorithm are almost alike but they are not completely the same. The primary differences between GA and GP are [29]:

- GP is a tree structure and has variable length chromosomes whereas GA makes use of fixed length chromosomes and structure.
- GP typically incorporates a domain specific syntax that governs acceptable (or meaningful) arrangements of information on the chromosome.
- GP makes use of genetic operators that preserve the syntax of its tree-structured chromosomes during 'reproduction'.
- GP solutions are often coded in a manner that allows the chromosomes to be executed directly using an appropriate

interpreter. GA's are rarely coded in a directly executable form.

In this study, a fixed length of chromosomes ranging between -1.0 and 1.0 is randomly generated. The set of chromosomes is known as individuals in a population. The set of individuals will go through selection process in order to select parent(s) for further operation. A fitness function is used to determine the quality of each individual before it can be selected. There are a lot of selection methods that had been introduced such as tournament selection, ranking selection, fitness uniform selection [30]. In this paper, ranking selection is used for the parent selection.

Elitism selection and age-based selection are used for selecting the survivor. By using the elitism selection, two fittest individuals will be kept for next generation and the remaining eight individuals are selected using age-based selection. By using this selection, the individual child will replace the individual parent for each new generation.

The selected parents will either run the crossover process or the mutation process based on the probability of the reproduction process. The process will start from the first node of the parse tree. A random integer value is generated to represent the root node that will perform the crossover. The sub-tree at the crossover point will be replaced by the sub-tree of the second parent. In order to maintain the tree structure, the fragment of the second parent is inserted to the first parent in a symmetrical manner.

Mutation is a process that mutates one or more gene in the chromosome to produce a new individual. Mutation helps to produce a better salutation that can avoid the population from stagnating. It also starts in the first node of the parse tree. A random number will be generated to represent the node of the tree. The sub-tree for the mutation rate will be deleted and will be replaced by another randomly generated value. The range of the value is between -1.0 and 1.0. The crossover rate and the mutation rate used in the study are 0.9 and 0.1 based on [31]. This GP will be implemented with FFNN and ERNN to generate the required controllers to design the TD map and the algorithm is shown in Fig. 2.

V. Experimental Setup

We used Warcraft III World Editor to design the TD map for this study as it enables user to customize and create his own custom map. Not only that, this software also allows user to modify object attributes such as the appearance and the attribute of a unit, the building and the item. It uses its own scripting language to modify the map. The scripting language is known as Just Another Scripting Syntax (JASS programming). The simplicity and the ease of use of JASS allow the ANN and GP to be coded in a short time.

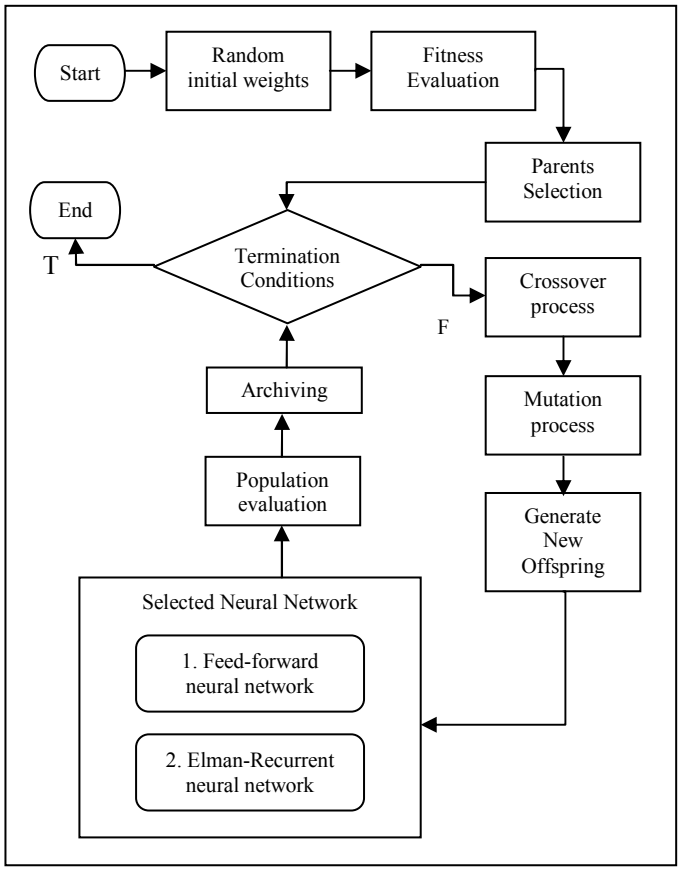

Figure 2. Genetic programming algorithm

Fig. 3 shows the designed TD map and the flow of the creeps' movement is indicated by the arrows. It can be seen that the creeps' path is straightforward. The game consists of five waves (20 creeps per wave). The health point and the movement speed of the creeps increase as the number of wave increases. User will be given 10 lives and a live will be deducted if one creeps managed to enter the base of the user. The game will end when all the creeps had been eliminated for the five waves or when 10 lives of the user are all deducted.

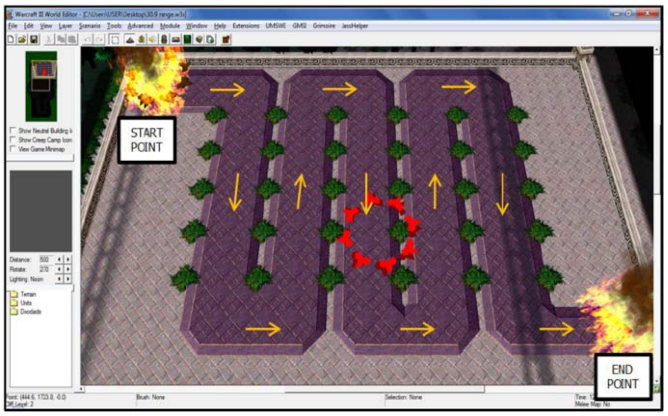

Figure 3. Overview of designed tower defense map

Fig. 4 shows the available placement for the tower to be built. There are 30 regions for the tower to be built. There is only one type of tower that can be built in this customized game. The list of regions is the input of ANN. Value 1 will be assigned to the region if a tower is to be built at that region and a value 0 is assigned if no tower is to be built.

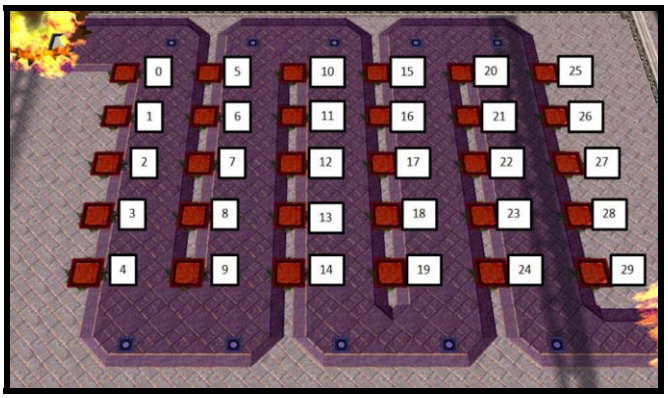

Figure 4. Tower building region

The FFNN consists of 31 inputs (30 actual inputs + 1 bias neuron), 16 hidden neurons (15 neurons + 1 bias neuron) and 30 output neurons. ERNN used 31 true input neurons (30 actual inputs + 1 bias neuron), 31 context input neurons (30 actual inputs + 1 bias neuron), 16 hidden neurons (15 neurons and 1 bias neuron) and 30 output neurons. The output of ANN shows the next tower to be built by returning 30 neurons which represents the region in the TD game. The activation function used in this study is Binary Sigmoid function.

There are ten chromosomes that are generated at the beginning of the experiment. Individual chromosome will be evaluated in each generation. The evaluation function used is based on the total number of kills minus 90. The crossover rate used in this research is 0.9 and the mutation rate is 0.1. The optimization process requires 150 generations.

In the testing phases, GP will determine the level of difficulty of the map by referring to the success rates given below.

- 90%-100% represents very easy.
- 65%-89% represents easy.
- 36%-64% represents intermediate.
- 11%-35% represents hard.
- 0%-10% represents very hard.

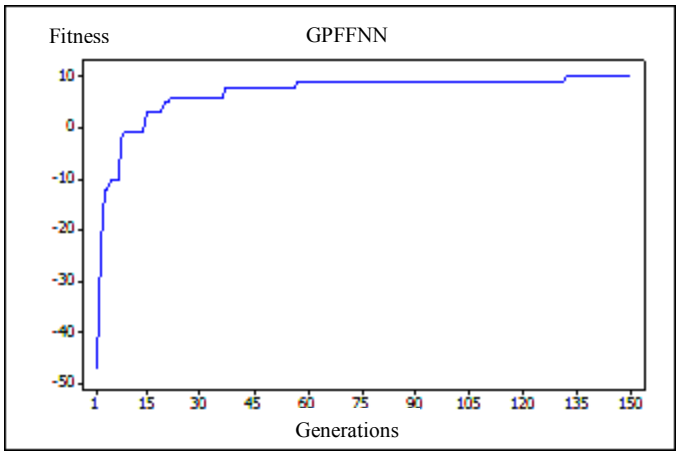

Figure 5. GPFFNN results

VI. Results and Discussions

A total of 10 rounds of the games were played for each GP with FFNN and GP with ERNN. Fig. 5 and Fig. 6 show the fitness score for 150 generations for GPFFNN and GPERNN for a given game. The highest fitness in the game is 10 which indicates that the user wins the game without losing any lives.

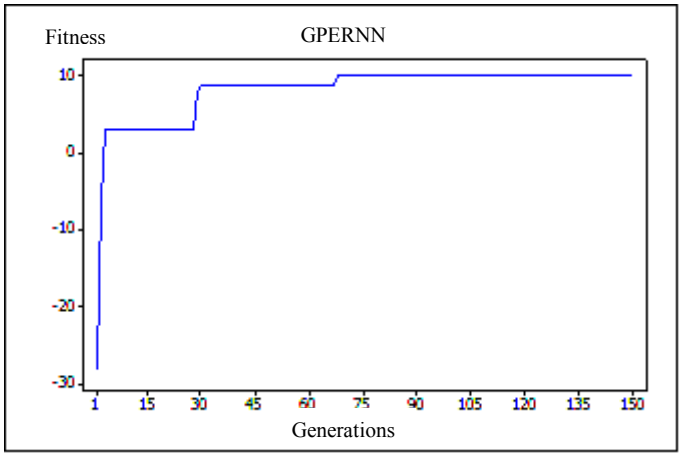

Figure 6. GPERNN results

Based on Fig. 5, the game did not produce a good result at the start of the game. The game scored -47 fitness values at the beginning of the game. Within 10 generations, the game started to improve by scoring -12 fitness value. However, at this point, the score is still a negative value. The game reached 0 fitness value in its 15th generation. It can be seen that the game improved over time and the best fitness value is recorded at 133th generation.

Fig. 6 shows the fitness value for GP with ERNN. Initially the game recorded -28 fitness score for the 1st generation. The game started to improve after the 3rd generation until the 6th generation where a positive value of 3 fitness score is reached. The game recorded a steady score until the 28th generation and increased to 8 in the 31st generation. The game reached the highest fitness score at the 69th generation. Based on the two

figures, we can conclude that GPERNN produce a better result compare to GPFFNN.

TABLE I. AVERAGE FITNESS OVER 150 GENERATIONS

Run	GPFFNN	GPERNN
1	4.55	2.37
2	-0.83	7.45
3	-1.71	4.83
4	2.86	4.59
5	-9.34	-0.39
6	4.25	9.18
7	0.92	7.01
8	0.99	7.28
9	3.03	-1.13
10	-9.39	5.72
Minimum	-9.39	-1.13
Maximum	4.55	9.18
Mean	-0.467	4.691
Standard Deviation	5.108	3.432

Table I shows that GPFFNN won 6 games and GPERNN won 8 games in a total of 10 games. The controller for GPFFNN lost the second game, third game, fifth game and the last game whereas the controller for GPERNN lost its game in the fifth and the ninth game. The average fitness score produced by GPFFNN for the ten games is less than 5. GPERNN produced a better average fitness score compared to GPFFNN as some of the game had average fitness score over 5 and one of the games recorded the highest fitness score of 9.18.

GPFFNN produced far lower fitness score compared to GPERNN. The lowest average fitness score for both methods are -9.39 and -1.13. GPERNN also produced a better maximum average fitness score compared to GPFFNN. The maximum average fitness score for GPFFNN are less than 5 (4.55) and the maximum fitness score for GPERNN is 9.18. The mean for GPFFNN is -0.467 and the standard deviation score is 5.108. The mean for GPERNN is 4.691 with a standard deviation score of 3.432.

TABLE II. AVERAGE WINNING RATE FOR GPFFNN AND GPERNN

RUN	*GAFFNN*	*GAERNN*
1	**44.47%**	28.20%
2	21.53%	**58.33%**
3	36.93%	**44.93%**
4	21.67%	**22.53%**
5	12.67%	**34.00%**
6	45.67%	**82.20%**
7	33.27%	**58.60%**
8	21.13%	**63.20%**
9	27.87%	**32.47%**
10	19.80%	**58.80%**
Average Winning Rate	28.50%	**48.33%**

Tests were also conducted for all the optimized controllers with winning rate. Table II shows the winning rates of GP with FFNN and GP with ERNN. Based on Table II, GP with ERNN controllers produced a higher average winning rate compared to GA with FFNN controllers. The data shows that nine out of ten of the controllers generated using GP with ERNN outperformed the controllers generated using GP with FFNN. Hence, it can be concluded that the controllers generated using GP with ERNN performed better compared to the controllers generated using GP with FFNN. Furthermore, the testing results also show the level of difficulty of the map used is intermediate as the success rate generated by GP with ERNN is between 36% and 64%.

VII. CONCLUSIONS AND FUTURE WORKS

Based on the results that we obtained, we can conclude that GP can be used to tune the weights of FFNN and ERNN in TD game. The FFNN and ERNN determine the position of the tower to be built without human intervention. The ANNs used are able to represent two different individuals in playing the TD games. The individual in ERNN produced a better average fitness score compared to the individual that was generated using FFNN.

However, both the average fitness score are not ideal as all the means of the fitness scores are below 8. This shows that different evolutionary algorithm techniques should be studied in order to determine the best evolutionary algorithm for tuning the weight of ANNs. However, this study implies that implementing AI techniques in TD games helps to make the game more exciting. It helps to increase the difficulties level when dealing with experience player(s) and making it difficult for them to win the game. In the other word, it is also impossible for a weak player to lose a game all the time.

In order to make the game closer to reality, there are some future works that can be done. Game score will be considered in the future TD games instead of the fitness score. The score will be increased proportionately to the level of each wave. Besides that, different towers should be allowed to be built. Hence this will enable the controller to determine when is the ideal time to build a new tower or to upgrade a tower and also the type of tower that should be used for each wave. Since the choices of tower will be increased, the types of creeps should also be diversified. The game should have different types of creeps such as air creeps and ground creeps. We can also have variations of tower that are unable to block certain types of creeps only. By doing this, the controller will be aware of the different characteristics of the environment rather than just making prediction of the towers.

ACKNOWLEDGMENT

This project is partially supported by Artificial Intelligence Research Unit (AiRU), Center of Excellence in Semantic Agents (COESA) and funded by the Ministry of Higher Education, Malaysia under ERGS0045-ICT-1/2013.

REFERENCES

[1] L.E. Susan, "Game Playing: The Next Moves" in AAAI-99 Proceedings, 1999.

[2] M. Sipper, Y. Azaria, A. Hauptman ad Y. Shichel, "Attaining Human-Competitive Game Playing with Genetic Programming" IEEE Trans. Syst. Man Cybem, 2005.

[3] K. Neil, Introduction to Game AI, Cengage Learning PTR; 1st edition. 2010.

[4] C.H. Ng, S.H. Niew, K.O. Chin, and J. Teo, "Infinite Mario Boss AI using genetic algorithm," IEEE Conference on Sustainable Utilization and Development in Engineering and Technology, Selangor, Malaysia, October 2011.

[5] K.T. Chang, J.H. Ong, J. Teo, and K.O. Chin, "The Evolutionary of Gamebots for 3D First Person Shooter (FPS)," IEEE The Sixth International Conference on Bio-Inspired Computing: Theories and Applications, Penang, Malaysia, 2011.

[6] P. Rohlfshagen and S. M. Lucas, "Ms Pac-Man Versus Ghost Team CEC 2011 competition," Proc. of the IEEE Congress on Evolutionary Computation, pp. 70-77, 2011.

[7] T.G. Tan, J. Teo, K.O. Chin, and P. Anthony, "Pareto Ensembles for Evolutionary Synthesis of Neurocontrollers in a 2D Maze-based Video Game," Applied Mechanics and Materials, Taiwan, pp. 3173-3177, 2012.

[8] D. Loiacono, A. Prete, P.L. Lanzi, and L. Cardamone, "Learning to overtake in TORCS using simple reinforcement learning," 2010 IEEE Congress on Evolutionary Computation (CEC), pp. 1-8, 2010.

[9] K.T. Chang, K.O. Chin, J. Teo, and J. Mountstephens, "Game AI Generation using Evolutionary Multi-Objective Optimization," IEEE World Congress on Computational Intelligence, pp. 2300-2307, Brisbane, Australia, 2012.

[10] K.T. Chang, K.O. Chin, J. Teo, and A.M. Jilui-Kiring, "Evolving Neural Controllers using GA for Warcraft 3-Real Time Strategy Game," IEEE The Sixth International Conference on Bio-Inspired Computing: Theories and Applications, Penang, Malaysia, 2011.

[11] K.T. Chang, J. Teo, K.O. Chin, and B.L. Chua, "Automatic Generation of Real Time Strategy Tournament Units using Differential Evolution," IEEE Conference on Sustainable Utilization and Development in Engineering and Technology, Selangor, Malaysia, 2011.

[12] K.B., Tan, J. Teo, K.O. Chin, P. Anthony, "An Evolutionary Multi-objective Optimization Approach to Computer Go Controller Synthesis," PRICAI 2012: Trends in Artificial Intelligence Lecture Notes in Computer Science, vol. 7458, pp. 801-806, 2012.

[13] K.T. Chang, J.H. Ong, J. Teo, and K.O. Chin, "The Evolution of Gamebots for 3D First Person Shooter (FPS)," IEEE The Sixth International Conference on Bio-Inspired Computing: Theories and Applications, Penang, Malaysia, 2011.

[14] G. Janusz, "Aritificial Intelligence in Games", Software Developer's Journal, 2005.

[15] D. M. Bourg and G. Seeman, AI for Game Developers, O'Reilly, 2004.

[16] D. Moriarty and R. Miikklainen, "Discovering Complex Othello Strategies Through Evolutionary Neural Networks", Connection Science, vol. 7, pp. 195-209, 1995.

[17] K. Chellapilla, and D.B. Fogel, "Evolving neural networks to play checkers without relying on expert knowledge," Neural Networks, IEEE Transactions on, vol. 10(6), pp. 1382-1391, 1999.

[18] D.B. Fogel, "Using evolutionary programing to create neural networks that are capable of playing tic-tac-toe," Neural Networks, IEEE International Conference on, vol. 2, pp. 875-880, 1993.

[19] M. Hitchens, "A Survey of First-person Shooters and their Avatars," Game Studies, 11(3). 2011.

[20] J. Togelius, and S. M. Lucas, "Evolving controllers for simulated car racing," In Evolutionary Computation, The 2005 IEEE Congress on, vol. 2, pp. 1906-1913, 2005.

[21] J. S. Kim, and M. A. Yahya, "The Implementation of Othello using Genetic Programming and Backpropagation Algorithm," American Association for Artificial Intelligence, 2002.

[22] Mancala game. http://www.corngolem.com/john/gp/project.html

[23] W. McCulloch, and W. Pitts, "A Logical Calculus of the Ideas Immanent in Nervous Activity," Bulletin of Mathematical Biophysics, vol. 5, pp. 115-133, 1943.

[24] A. Krenker, J. Bester, A. Kos, Introduction to the Artificial Neural Networks, Artificial Neural Networks – Methodology Advances and Biomedical Applications", InTech, pp. 3-18, 2011.

[25] M. Walker, Introduction to genetic programming, Tech. Np: University of Montana, 2001.

[26] A. Hauptman, and M. Sipper, "GP-Endchess: Using genetic programming to evolve chess endgame players," In Maarten Keijzer, Andrea Tettamanzi, Pierre Collet, Jano I. van Hemert, and Marco Tomassini, ed., Proceedings of the 8th European Conference on Genetic Programming, vol. 3447 of Lecturer Notes in Computer Science, pp. 120-131, Switzerland, 2005.

[27] Y. Schichel, E. Ziseman and M. Sipper, "GP-robocode: Using genetic programming to evolve robocode players," in Proceedings of the 8th European Conference on Genetic Programming, Lausanne, Switerland, March 30, M. Keijzer, A. Tettamanzi, P. Collet, J. van Hemert, and M. Tomassini, Eds. 2005, vol. 3447 of Lecture Notes in Computer Science, pp. 143 – 154, Springer-Verlag, Heidelberg, 2005.

[28] Y. Azaria and M. Sipper, "GP-Gammon; Using genetic programming to evolve backgammon players", in Proceedings of 8th European Conference on Gentic Programming, M. Keijzer, A. Tettamanzi, P. Collet, J. van Hemert, and M. Tomassini, eds. vol. 3447 of Lecture Notes in Computer Science, pp. 132 – 142, Springer-Verlag, Heidelberg, 2005.

[29] M.J. Willis, H. G. Hiden, P. Marenbach, B. McKay, G.A. Montague, "Genetic programming: an introduction and survey of applications," Genetic Algorithms in Engineering Systems: Innovations and Applications, Second International Conference On, pp. 314-319, 1997.

[30] H. Y. Xie, An Analysis of Selection in Genetic Programming, Thesis, Victoria University of Wellington, 2009.

[31] D. R. Munroe, "Genetic Programming: The ratio of Crossover to Mutation as a function of time," New Zealand, 2004.