Teaching Genetic Algorithm-based Parameter Optimization Using Pacman

Carlos N. Silla Jr.
Intelligent Systems Laboratory (LASIN)
Graduate Program in Computer Science (PPGIa)
Pontifical Catholic University of Paraná (PUCPR)
Curitiba, PR, Brazil 80215–901
Email: carlos.sillajr@gmail.com

***Abstract*—Previous artificial intelligence education research (DeNero and Klein, 2010) has used the classic video game Pacman to teach introductory artificial intelligence concepts. One of the advantages of the work proposed in (DeNero and Klein, 2010) is that the same framework, in this case using the video game Pacman, can be used for different student assignments covering different artificial intelligence algorithms and methods. The issue of how to use the same practical framework is an important one, because if students have to learn a different framework for every assignment they will often feel discouraged. For this reason, the main contribution of this paper is to present a practical assignment to teach students about genetic algorithm-based parameter optimization using the Pacman framework.**

I. Introduction

Genetic Algorithms (GAs) are a search and optimization procedure that can be used to find solutions to several scientific and engineering problems [1]. The GAs were originally developed by John Holland in the 70's [2] and given its importance to real world problems they are often taught in the context of artificial intelligence undergraduate classes.

Previous approaches to teach GAs to undergraduate students include: Using a "hands on" strategy using poppet beads to represent *Biston betula* moths [3]; Developing visual and interactive tutorials, such as the GATutor [4] and the Evolutionary Algorithms Sandbox [5]; Using Japanese nanograms to teach about advanced GA features [6], [7]; and using GAs to design controllers for non-player characters in games [8].

In the context of artificial intelligence education research, DeNero and Klein [9] have used the classic video game Pacman to teach several introductory artificial intelligence concepts. One of the advantages of the work proposed in DeNero and Klein [9] is that the same framework, in this case using the video game Pacman, can be used for different student assignments covering different artificial intelligence algorithms and methods. The issue of how to use the same practical framework is an important one, because if students have to learn a different framework for every assignment they will often feel discouraged.

For this reason, the main contribution of this paper is to present a practical assignment to teach students about genetic algorithm-based parameter optimization using the Pacman framework. In order to use the approach present in this paper it is mandatory that the students have already developed an automated Pacman player using at least a greedy search with different heuristic functions (more details about this assignment are presented in Section 2). This is necessary as by using different heuristic functions the students will need to combine the outputs of the different heuristics. The way the students combine the outputs is to assign weights to each heuristic and they use a manual trial and error approach to adjust the weights, i.e. they try different parameters for each heuristic weight and evaluate what happens with their developed Pacman player.

The remainder of this paper is organized as follows: Section III presents a brief overview of Genetic Algorithms. Section IV presents the GA-based pacman assignment. Section V presents the discussion about the assignment in practice as well as student feedback and learning outcomes. Section VI presents the related work and finally in Section VII we present the conclusions of this work.

II. Pre-Requisites

In order to use the assignment proposed in this work (presented in Section IV) it is important that the students have already completed a previous assignment, which was originally suggested by DeNero and Klein. The main idea of the assignment is to use informed search methods to create an automatic player for Pacman.

A. The Java Pacman Framework

In our classes, we use a java-based version of the Pacman framework that was developed by Ted Grenager. The java Pacman framework runs with a textual interface by default (Figure 1), with a graphical user interface (Figure 2) with the parameter `-display gui` or with no interface with the parameter `-display none` (Figure 3). As it can be noted from Figure 2, this version of the game does not contain the powerups (fruits) that allows pacman to eat the ghosts.

In order to execute an automated player, it is necessary to pass the player as a parameter to the framework using `-pacman player` where `player` is the name of the class that has the implemented Pacman player. In order to create an automated player the students need to realize (implements in java) two interfaces, namely *PacmanPlayer* and *StateEvaluator*.

The *PacmanPlayer* interface enforces that the method *Move chooseMove(Game game)* is implemented. The `Move` class is an enumeration of the valid Pacman moves, namely UP, DOWN, RIGHT, LEFT and NONE. Therefore, the `chooseMove` method receives a game state and returns a move to be executed in the game. The *StateEvaluator* interface enforces that the method *double evaluateState(State s)* is implemented. The `State` class represents a state of the Pacman board.

The framework also allows for different scenarios to be evaluated, i.e. using different number and/or types of ghosts.

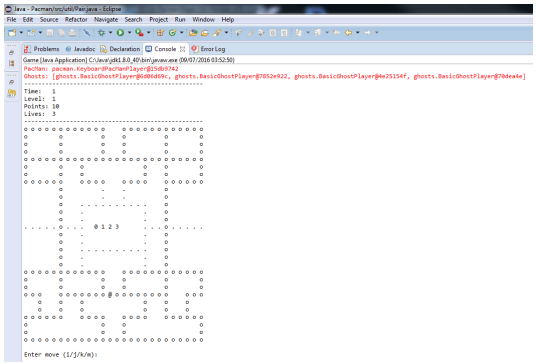

Fig. 1. The Pacman Framework Text Based Interface.

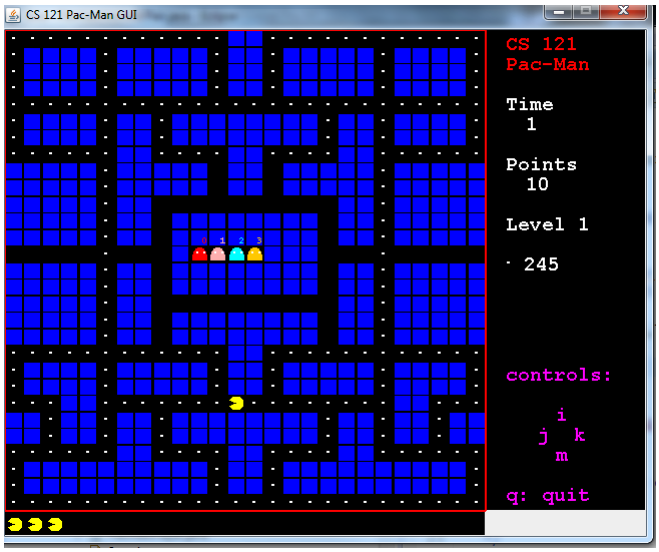

Fig. 2. The Pacman Framework Graphical User Interface.

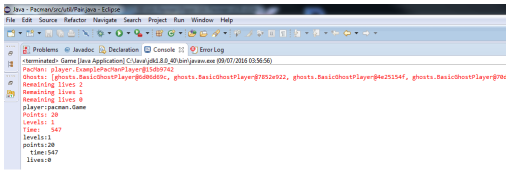

Fig. 3. The Pacman Framework Without an User Interface.

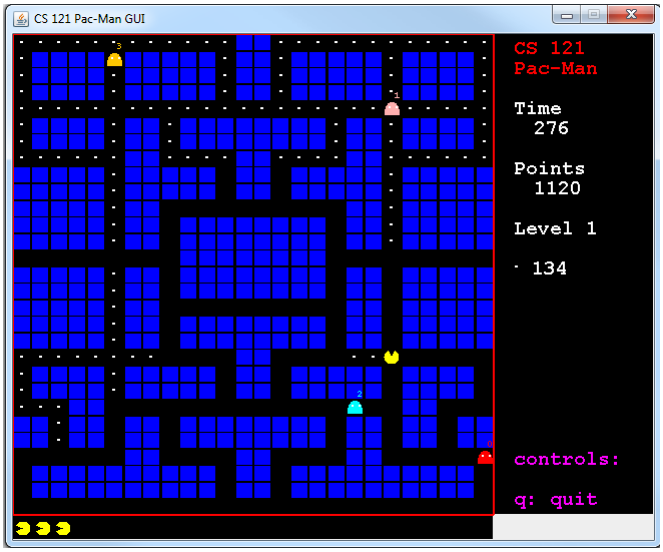

Fig. 4. An Example of a Particular Game State.

B. Pacman as an Informed Search Problem

Considering that the students do not need to worry about the Pacman game implementation, they can focus on the development of an artificial intelligence for the game. The main idea is to use informed-search methods for the problem. The easiest approach is to implement a greedy best first search algorithm [10]. In order to use the greedy best first search algorithm, we need to know how to differentiate between different states at a given time. For example, considering Figure 4 the automated Pacman player can perform all five valid moves, therefore all possible moves are evaluated and a particular move is chosen.

This raises the question about how to chose a particular move at a given game state. The answer to this question is to use an heuristic function. For example, the different states could be compared by using the number of items left on the board after Pacman performs a particular move. Using this heuristic, in the scenario illustrated in Figure 4, Pacman would move LEFT. However, another heuristic could be to stay at a safe distance from the closest ghost, in which case Pacman could move UP. Given the complexity of the game it is not recommended to use only one heuristic, such as eat the closest item, as Pacman might die too often. Therefore, it is necessary to implement different heuristics and combine them in order to have a good automated Pacman player. Normally the students combine their developed heuristics in a manual trial and error approach. Other informed search algorithms such as the Breath First Search and A* search algorithms can also be used to develop an automated Pacman player. However, they should be implemented using fixed-depth (i.e. limiting the maximum number of levels they will explore) given the complexity of the problem.

C. The First Pacman Assignment

In our artificial intelligence undergraduate classes, students are told to work on teams of at most two students and are asked to:

- Implement the greedy best-first search [10].
- Implement the Breath First Search Algorithm with fixed-depth [10].
- Implement the A* Search Algorithm with fixed-depth [10].
- Implement at least eight heuristics to evaluate the non-terminal states.
- Implement at least one strategy to avoid loops.
- Design and run experiments to evaluate the different components of their developed players, for example:
  - Which heuristics should be used?
  - What is the impact of different depth values?
  - What is the effect of different weights for the heuristics?
  - What was the best parameter setting you found for a given ghosts setting (e.g. using three random ghosts and one stalker ghost.)?
- Write a detailed report explaining everything that they did and reporting any issues they had.

We also run an in-class live tournament where the students choose whether or not they want to participate.

In order for the students to successfully employ the GA-based parameter optimization (presented in Section IV) it is mandatory that all teams implement at least the greedy best-first search and as many heuristics as they can. In the case that the students implement only the greedy best-first search and a number of n heuristics, they already have a parameter optimization problem, as shown in Equation 1, where they manually try to adjust the weights (w) of each heuristic (h).

$$w_1 \times h_1 + w_2 \times h_2 + \ldots + w_n \times h_n \quad (1)$$

In the case that the students also implement the fixed-depth search methods, they will also have to deal with another parameter, namely the depth of the search. Regardless of the search methods implemented by the students, it is important that they record the weights used for a particular ghost setting that achieved their best results. This information will be used as a baseline for their experiments in the GA-based assignment.

III. Brief Overview of GA Fundamentals

In this section we present a brief overview of Genetic Algorithms to elucidate how they can be used to optimize the parameters of the assignment presented in Section II. An overview of a basic genetic algorithm is shown in Figure 5.

One important aspect of the GAs is that the same structure presented in Figure 5 can be used to solve different types of search and optimization problems. The GAs have this flexibility because they have three main components that must be defined according to the problem:

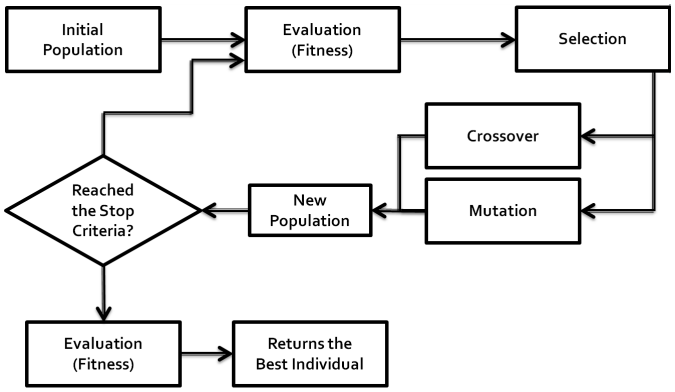

Fig. 5. Overview of the Basic GA (Adapted from [11]).

- The individual representation. In a GA an individual is one solution to the problem we are trying to solve. Each individual is encoded in a chromosome. Each indivisible piece of a chromosome is called gene and normally in a population all individuals have a fixed sized chromosome. Many researchers represent the chromosome of an individual as a string of bits, however chromosomes can be encoded using any type of data structure.
- A fitness evaluation function. In a GA the fitness functions measures how good (or bad) an individual is to the problem at hand. The fitness function will always be specific to a particular problem.
- The genetic operators. These operators are responsible for the evolutionary process of the population. The most common genetic operators are mutation and crossover. However before a genetic operator can be applied it is necessary to select individuals from whom new individuals will be created (off springs). The common selection processes are the roulette wheel and the tournament. In the roulette wheel, usually a pre-determined number of individuals will be selected based on a fitness-proportional draw, where individuals with better values of fitness have greater chances to be selected. In the tournament, a k number of individuals are selected at random and the one with the best fitness is selected. After selecting the individuals it is possible to apply the genetic operators of crossover and mutation. It should be noted that these operators have a user defined probability of being applied. The traditional crossover operator is known as one point crossover and exchanges the genetic material between two parents two create two off springs. The traditional mutation operator changes the value of a gene from 0 to 1 or vice-versa.

Normally in our classes, we demonstrate each step of the basic GA execution using the OneMax problem [12]. The onemax problem consists of given a bit string of size n, generating an individual where all values are 1's. This is a nice example to first show the students how the GA operates.

However, in order to allow the students to apply this techniques to real world problems it is important that they

TABLE I
ANALYSIS OF THE PACMAN PARAMETER OPTIMIZATION ASSIGNMENT

Team	Framework	# of Heuristics	Representation	Selection	Cross-over Type	Mutation Type	Improved Results with the GA?
1	ECJ	8	Float (Diff. Ranges)	Tournament	One/Two	Gauss	✓
2	ECJ	10	Float (Diff. Ranges)	Tournament	One/Two/Any	Gauss	✓
3	ECJ	3	Integers (0 to 100)	Tournament	One/Two	Reset	✓
4	ECJ	8	Float (Diff. Ranges)	Tournament	One/Two	Gauss	✓
5	ECJ	8	Integers (0 to 100)	Tournament	One/Two	Reset	✓
6	ECJ	4	Integers (0 to 100)	Tournament	One/Two	Reset	✓
7	ECJ	8	Integers (Diff. Ranges)	Tournament	One	Reset	✓
8	ECJ	8	Integers (0 to 100)	Tournament	Two	Reset	✓
9	ECJ	8	Integers (0 to 100)	Tournament	One	Reset	✓
10	ECJ	8	Integers (0 to 100)	Tournament	One	Reset	✓
11	ECJ	8	Integers (-100 to 100)	Tournament	One	Reset	✓
12	ECJ	9	Integers (-100 to 100)	Tournament	One/Two	Reset	✓
13	ECJ	8	Integers (-100 to 100)	Tournament	One/Two	Reset	✓

understand both the core concepts as well as why the standard GA's might need to be modified. For this reason, in our classes, we also demonstrate how the GA could be used to provide solutions to the Travelling Salesman Problem (TSP). The TSP in an interesting example because the individual representation can be based on the list of cities (one route in the TSP), there is a clear fitness function (find the smallest route) and the standard genetic operators of mutation and crossover might generate invalid routes. This two examples are enough, in our experience, to prepare the students to deal with the parameter-based optimization problem presented in the next section.

IV. THE GA-BASED PACMAN ASSIGNMENT

In order to motivate the students and help them develop practical skills for using GAs to solve an optimization problem, we use the same Pacman framework that the students have used in a previous assignment (presented in Section II) and ask the students to:

- Design and Implement a representation that takes into account the search depth (only if they implemented the Breath First and/or A* Search Algorithm with fixed-depth) and the weights assigned to each heuristic.
- Design and Implement at least two fitness functions to evaluate their developed individuals.
- Decide whether or not it is necessary to modify the genetic operators of mutation and crossover.
- Design and run experiments in at least two ghost settings to evaluate:
  - What is the impact of changing the population size?
  - What is the impact of changing the probability values of the genetic operators of mutation and crossover?
  - What was the best individual found in your experiments? Is this individual better than the solution you found in the previous assignment?
- Write a detailed report explaining everything that they did and reporting any issues they had.

V. EVALUATION OF THE ASSIGNMENT

Table I presents an overview of GA-based pacman assignment evaluation. The analysis of Table I and of the reports handed in by the students after they completed the assignment show some interesting insights.

First, all students choose to use the Java-based Evolutionary Computation Research System (ECJ[1]). This choice might have been influenced by the use of the ECJ framework in one of our practical classes where the students run some simulations using the max one tutorial. However, despite the students familiarity with the framework, this shows that the students decided to take a deep understanding of how the ECJ framework works rather than implementing their own AG. When we asked the students about their reason for using the ECJ framework, their common answer was that within the framework they only needed to implement/modify a few classes to have their assignment working. Furthermore, the students liked some of the built-in facilities that the ECJ framework offers, such as parallelism and the ability to create and restore checkpoints.

Second, most teams have used a vector of Integers to represent their individuals. While some teams used only positive values (normally from 0 to 100), others allowed their chromosomes to receive negative values as well. It should be noted that for the teams that implemented the breath first search and A* search algorithms, they normally encoded the depth of the search as being the first gene in the chromosome. Furthermore, all teams that implemented the depth-limited search methods took safe guard measures to not allow the use negative or relatively high (e.g. higher than 5) values in the gene responsible for the depth. Figure 6 presents information about which search method(s) were used in the GA-based optimization of each team.

Third, all teams used the selection by tournament. Fourth, while some teams only used the one point crossover that they learned about during the lectures, the majority of the teams

[1]Available at: https://cs.gmu.edu/∼eclab/projects/ecj/

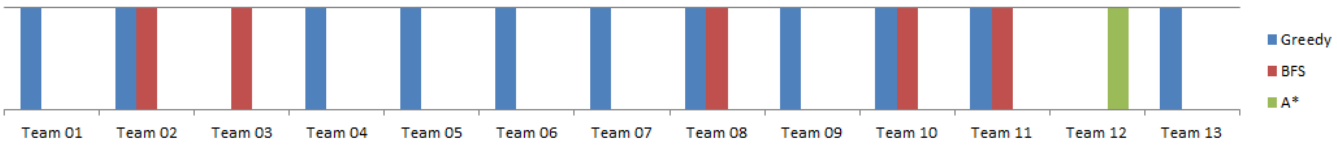

Fig. 6. Search method(s) used by each team in the GA-based Pacman assignment.

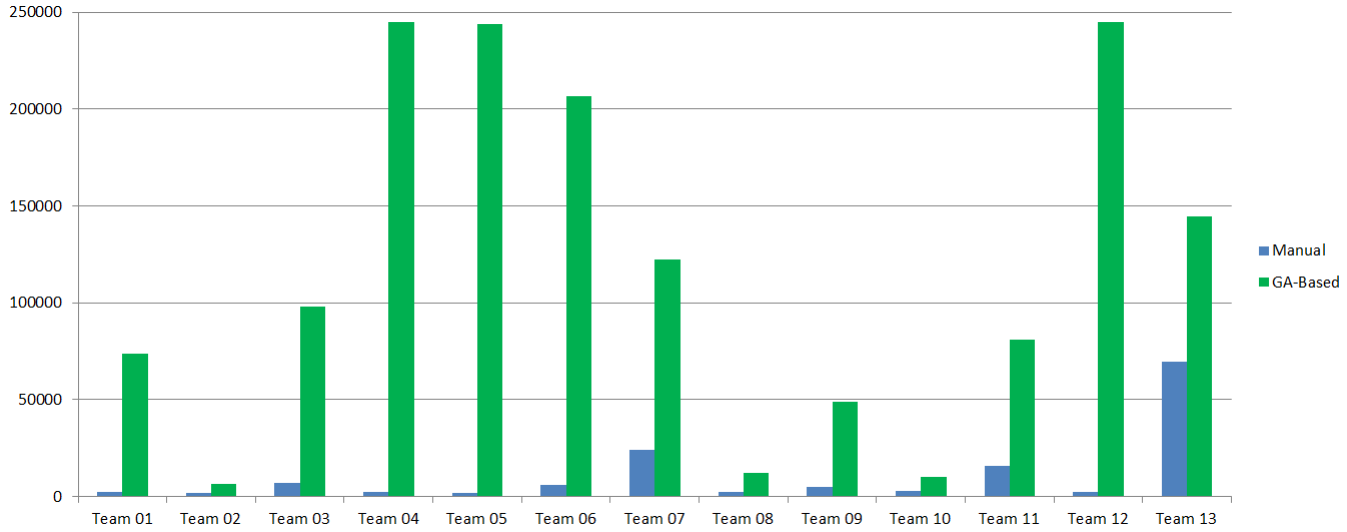

Fig. 7. Comparison of the scores obtained by manually adjusting the weights against the GA-based approach for each team.

also researched about the two point crossover or, in the case of one particular team, the any point crossover. Figure 8 presents the crossover types percentage used by the different teams.

Fifth, due to the representation being used (Integers in all but three cases, in which floats were used), the students also had to research how to adapt the mutation operator. For the integers representation the students employed the reset mutation type, which randomly changed the current value of a gene by a random value within the gene valid values range.

Sixth, as we have requested the students to design and implement at least two fitness functions, they all implemented the absolute score as one of the fitness function. Figure 9 presents the percentage of the different types of fitness functions used by the different teams. The analysis of Figure 9 shows that 31% of the teams did not propose a second fitness function, while the others teams used the score and the game time in different ways or the last achieved level before death.

Seventh, all teams reported that by using the GA-based approach, they were able to achieve better Pacman players than the one developed by manual trial and error. Figure 7 presents a comparison of the best scores obtained by manually adjusting the weights against the best GA-based approach weights for each team.

VI. Related Work

Genetic Algorithms are an important artificial intelligence technique that can aid in the development of solutions for different scientific and engineering applications [1]. Given their importance, different authors have addressed the issue of how to teach Genetic Algorithms in different ways.

In the work of Venables and Tan [3] the authors proposed a "hands on" strategy to teach genetic algorithms using poppet

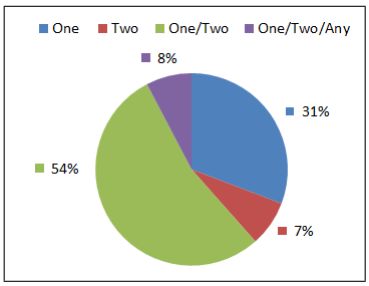

Fig. 8. Different types of crossover used by the students.

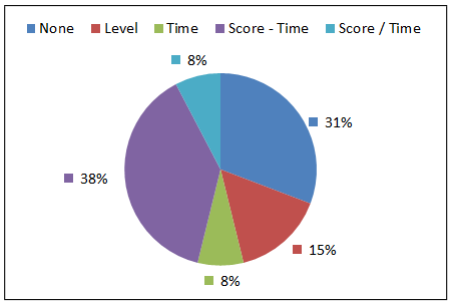

Fig. 9. Different types of fitness functions used by the students.

beads to represent *Biston betula* moths. The main idea is that individuals can be represented by using a string of black and white poppet beads, similar to a bit string vector. The students are then asked to: (1) work in pairs and randomly create a population of eight moths (with chromosome size = 8); (2) select the 4 fittest individuals; (3) clone these individuals and randomly divide them in pairs of two; (4) Use the cloned individuals to perform crossover; (5) Randomly change the color of one poppet in the current population (which is equivalent to the mutation operator).

In the work of Salcedo-Sanz et al. [6] and Tsai et al. [7] the authors use Japanese puzzle, also known as nanogram to teach advanced features of genetic algorithms. In order to solve the puzzles using the GA's the students need to learn about the problem and use binary matrices to represent the individuals. For this reasons they also need to adapt the inner working of the generation operators of mutation and crossover.

In the work of Fernandez-Manjon et al. [8], the authors propose two games that can be used to teach genetic algorithms. The first game is based on the rabbit and the dogs game, which consists of a board games with 3 dogs that try to catch a rabbit. The GA is used to automatically developed a controller for the game and they use different fitness functions according to the difficulty level (easy, intermediate or advanced). The students can also configure the number of: (1) individuals for each generations; (2) max number of generations; (3) mutation probability). The second game is the ten and a half card game which the objective is to draw cards to get as close as 10.5 as possible. As with the previous game the students can configure several parameters.

Other authors developed tools to aid the understanding of how genetic algorithms work in interactive framework. Examples of this approach are the GATutor developed by Prince et al. [4] and the Evolutionary Algorithms Sandbox developed by Gardner and Simon [5]. Cao and Wu [13] have used Matlab to teach genetic algorithms.

The review presented in this section about the state of the art on teaching about genetic algorithms shows that many interesting and inspiring initiatives have already been done. However, we are unaware (to the best of the authors knowledge) of any approaches that have the video game pacman to teach about optimization using genetic algorithms. In the context of optimization education, an interesting approach was developed by Syberfeldt and Syberfeldt [14] where they developed a serious game using a Lego factory to teach students about production optimization. The idea behind the game is that the student becomes the factory production manages and has to find the best possible configuration to maximize profit. The game is designed to make students realise the infeasibility of exhaustive testing and the benefits of AI-based optimization. The students then had to implement their own evolutionary algorithm.

VII. Conclusions

The main contribution of this work was to present and evaluate an assignment that allows the students to use the Pacman framework to learn about genetic algorithm-based parameter optimization. This is important as students often feel discouraged if they have to learn different frameworks for different assignments. The proposed assigned was inspired by the work of DeNero and Klein [9] who uses the classic video game Pacman as a framework for different assignments involving several artificial intelligence techniques.

The evaluation of the assignment in our undergraduate artificial intelligence classes have shown that the students properly learned about the core concepts of applying a genetic algorithm to an optimization problem. Furthermore, when comparing their GA-optimized players with their previously developed players all students reported that the performance of their players was enhanced.

Acknowledgments

The author would like to thank DeNero and Klein for their inspiring work on teaching artificial intelligence using the classic video game Pacman, Ted Grenager for making the java-based version of the Pacman framework and the anonymous reviewers for their valuable feedback.

References

[1] D. A. Coley, *An Introduction to Genetic Algorithms for Scientists and Engineers*. World Scientific, 1999.

[2] J. H. Holland, *Adaptation in Natural and Artificial Systems*. The University of Michigan Press, 1975.

[3] A. Venables and G. Tan, "A 'hands on' strategy for teaching genetic algorithms to undergraduates," *Journal of Information Technology Education*, vol. 6, 2007.

[4] C. Prince, R. L. Wainwright, D. A. Schoenefeld, and T. Tull, "Gatutor: a graphical tutorial system for genetic algorithms," in *Proceedings of the twenty-fifth SIGCSE symposium on Computer science education*, 1994, pp. 203–207.

[5] B. G. Gardner and D. Simon, "Evolutionary algorithm sandbox: A web-based graphical user interface for evolutionary algorithms," in *Proc. of the IEEE International Conference on Systems, Man and Cybernetics*, 2009, pp. 577–582.

[6] S. Salcedo-Sanz, J. A. Portilla-Figueras, E. G. Ortiz-Garca, A. M. Perez-Bellido, and X. Yao, "Teaching advanced features of evolutionary algorithms using japanese puzzles," *IEEE Transactions on Education*, vol. 50, no. 2, pp. 151–156, 2007.

[7] J.-T. Tsai, P.-Y. Chou, and J.-C. Fang, "Learning intelligent genetic algorithms using japanese nonograms," *IEEE Transactions on Education*, vol. 50, no. 2, pp. 164–168, 2012.

[8] J. M. Chaves-Gonzalez, N. Otero-Mateo, M. A. Vega-Rodriguez, J. M. Sanchez-Perez, and J. A. Gomez-Pulido, *Game Implementation: An Interesting Strategy to Teach Genetic Algorithms*. Springer, 2007, ch. Computers and Education, pp. 205–223.

[9] J. DeNero and D. Klein, "Teaching introductory artificial intelligence with pac-man," in *Proceedings of the Symposium on Educational Advances in Artificial Intelligence (EAAI)*, 2010.

[10] S. J. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*. Pearson Education, 2003.

[11] Y.-H. Liao and C.-T. Sun, "An educational genetic algorithms learning tool," *IEEE Transactions on Education*, vol. 44, no. 2, 2001.

[12] J. D. Schaffer and L. J. Eshelman, "On crossover as an evolutionary viable strategy," in *Proc. of the 4th Int. Conf. on Genetic Algorithms*, 1991, pp. 61–68.

[13] Y. J. Cao and Q. H. Wu, "Teaching genetic algorithm using matlab," *International Journal of Electrical Engineering Education*, vol. 36, no. 2, pp. 139–153, 1999.

[14] A. Syberfeldt and S. Syberfeldt, "A serious game for understanding artificial intelligence in production optimization," in *Proceedings of the 2010 IEEE Conference on Computational Intelligence and Games*, 2010, pp. 443–449.