Reactive Control of Ms. Pac Man using Information Retrieval based on Genetic Programming

Matthias F. Brandstetter
Centre for Computational Intelligence
De Montfort University
United Kingdom, Leicester LE1 9BH

Samad Ahmadi
Centre for Computational Intelligence
De Montfort University
United Kingdom, Leicester LE1 9BH

***Abstract*—During the last years the well-known Ms. Pac Man video game has been – and still is – an interesting test bed for the research on various concepts from the broad area of Artificial Intelligence (AI). Among these concepts is the use of Genetic Programming (GP) to control the game from a human player's perspective. Several GP-based approaches have been examined so far, where traditionally they define two types of GP terminals: one type for information retrieval, the second type for issuing actions (commands) to the game world. However, by using these action terminals the controller has to manage actions issued to the game during their runtime and to monitor their outcome. In order to avoid the need for active task management this paper presents a novel approach for the design of a GP-based Ms. Pac Man controller: the proposed approach solely relies on information retrieval terminals in order to rate all possible directions of movement at every time step during a running game. Based on these rating values the controller can move the agent through the mazes of the the game world of Ms. Pac Man. With this design, which forms the main contribution of our work, we decrease the overall GP solution complexity by removing all action control management tasks from the system. It is demonstrated that by following the proposed approach such a system can successfully control an autonomous agent in a computer game environment on the level of an amateur human player.**

I. Introduction

After *Namco* has published the original *Pac Man* arcade video game in 1980, it has now become one of the most famous video games of all time [1]. The game principle is simple: Let the player control the Pac Man character to guide him through a maze, avoiding any of the four enemy ghosts that are trying to catch Pac Man. The player gains points by letting Pac Man eat pills that are scattered around the maze, while four special power pills allow Pac Man to become the predator and chase the ghosts to gain even more points for a short period of time. After all pills have been consumed by Pac Man, the next maze filled with new pills is started.

The (unofficial[1]) sequel to the original Pac Man game, *Ms. Pac Man*, released in 1981 by *Midway*, introduced – beside new mazes, new sound effects, and other minor changes – new non-deterministic ghost movement patterns, which make it impossible for the human player to exploit pre-defined movement patterns, as in the original game. And, as noted in [2], even though the basic principle of the game is very simple, it is hard to gain many points due to this non-deterministic, randomised nature of the ghosts in the Ms. Pac Man game, which makes it an interesting test bed for research on different approaches of Artificial Intelligence based video game control.

This paper presents an evolved controller for the Ms. Pac Man video game based on Genetic Programming. It describes a – to the knowledge of the author – novel approach where, instead of directly controlling the movement of the Ms. Pac Man character, all valid movement directions are rated based on the current game state in order to determine the next (near) optimal direction to go. The main contribution of our work is the design of a solely reactive, GP-based controller for autonomous agents in the Ms. Pac Man game. As described in detail later in this paper, we encode a GP individual as an arithmetic expression that combines certain numerical features of the current state of the game, based on a specific movement direction. The output of the expression is interpreted as a number indicating the level of preference for this direction. All possible directions are evaluated, and the one with the strongest preference is actually taken.

The next section of this paper gives an overview of related work in the area of Pac Man controller research. Section III explains the details of the Ms. Pac Man game, while section IV summarises the theoretical concepts of GP in general. In section V the proposed system is explained, while section VI analyses the outcomes of the experiments that have been undertaken during this project. The last section then summarises the presented work and gives an outlook to possible subsequent projects.

II. Previous Work

Koza [3] was among the first to examine the use of GP in order to evolve a controller for a custom and feature-reduced version of Pac Man. He identified *task prioritisation* as one of the major characteristics of such a system. In his work Koza defined a set of 15 primitive GP operators, for both information retrieval (e.g. *Distance-To-Pill*) and Pac Man control (e.g. *Advance-To-Pill*). Operators of the latter category where able to automatically calculate the best (shortest) route to a given target (e.g. pill or ghost). Thus, the Pac Man character is controlled via abstract (high-level) movement control operations, based on the current state of the game.

In a more recent paper Alhejali and Lucas [4] extended Koza's work by additional GP terminals, such as *Is-In-Danger*

[1]http://www.arcade-museum.com/game_detail.php?game_id=8782

or *To-Safety*, which are even more abstract operators for information retrieval and Pac Man movement control. The research work by Alhejali and Lucas is based on a more feature-complete version of Ms. Pac Man, similar to the original arcarde video game. They reported of good results achieved by the evolved controller program, which in turn led to the initial idea for the work presented in this paper.

Other approaches based on alternative Evolutionary Computation (EC) techniques have been examined as well. In [5] a system based on evolving neural network controllers for Ms. Pac Man using simple genetic algorithms is described. [6] presents a Ms. Pac Man controller that implements an optimisation algorithm based on ant colonies. Parameters of these colonies are optimised via a genetic algorithm.

Beside EC also other areas of autonomous AI-based Pac Man control have been examined. A Monte Carlo approach is presented in [7], which tries to avoid the ghosts in "dangerous situations". Such a situation is defined as when Pac Man is surrounded by one or more ghosts. The look-ahead capability of the Monte Carlo simulation is used to obtain the accurate survival probability of different moving decisions in these situations.

A simple tree-search method for playing Ms. Pac Man is presented in [8]. This approach evaluates all possible paths that Ms. Pac Man can take from her current position by creating a tree that represents these paths. They are then evaluated based on information such as the amount of ghosts or pills on either of them, resulting in a decision where Ms. Pac Man should move next.

[9] describes a rule-based approach using hand-coded rules. The decision about the path to take next is made based on a vocabulary of higher level conditions and actions. The paper presents a reactive system, which means that it does not maintain a history of what have been seen so far, nor does it try to anticipate what to expect next. The desired Ms. Pac Man action is determined based on a set of different conditions expressed in an abstract description of the current game state.

Moreover, beside control tasks, Pac Man has also been subject to various other research areas. In [10] *Flow* is measured as concept for detecting game fun in the Pac Man game. The paper evaluates what "fun" actually means to different computer game players at different gaming expertise levels. Besides, it analyses aspects of game difficulty and diversity, as well as various Ms. Pac Man ghost control algorithms.

[11] analyses dynamic difficulty adjustment of game AI using Monte Carlo tree-search. The proposed system is able to dynamically adapt difficulty of the game by modulating the simulation time of the Monte Carlo tree-search algorithm. Longer simulation times are being said to result in more intelligent non-playing characters (i.e. ghosts in the Pac Man game), which is evaluated in that paper.

III. Ms. Pac Man Game Description

This section describes the Ms. Pac Man video game in general and the IEEE game simulation system[2] used for this project in particular. Please note that some aspects of that version differs from the original Ms. Pac Man implementation, as noted below.

The basic rules of the original game are easy to explain: the main protagonist, Ms. Pac Man, originally controlled by the human player, moves through a maze of horizontal and vertical hallways, trying to eat as many pills as possible (each consumed pill scores 10 points). She is chased by four ghosts, each of them with its own "personality" (i.e. movement strategy), trying to stop Ms. Pac Man as fast as possible. As soon as a ghost catches Ms. Pac Man, she loses one of her lives. At game start the player has three lives, it ends when all those lives have been exhausted. One additional life is gained when Ms. Pac Man has scored 10,000 points. As soon as all pills have been consumed by Ms. Pac Man, the next level, i.e. another maze, is loaded and filled with new pills. The IEEE simulation consists of four different mazes; after the fourth maze has been finished, the first maze is started again, and so forth.

Every maze contains four power pills. When Ms. Pac Man eats one of them, all ghosts turn blue and are edible by the Pac Man character for a short period of time (the higher the current level, the shorter this time span). By eating an edible ghost Ms. Pac Man scores 200 extra points, doubling this amount for every other ghost that is eaten after one power pill has been consumed, for a total of 200 + 400 + 800 + 1,600 = 3,000 additional points. When another power pill is consumed, this scoring scheme starts again from 200. Eating the power pill itself scores 50 points. In the original video game additional points could be earned by letting Ms. Pac Man eat some fruits that (pseudo-)randomly appeared somewhere within the maze. A sample game screen is shown in figure 1.

In comparison to the original Ms. Pac Man video game, in the IEEE simulation of the game the speed of all characters is always the same (except for the ghosts moving at half speed when they are edible). Additionally, each level (maze) is stopped after 3,000 time steps. When this happens before all pills have been consumed, all remaining pills are awarded to Ms. Pac Man, in order to prevent the ghosts from constantly blocking the power pills (in which case the game would result in a stalemate). This feature is "automatically" exploited by the implemented proof-of-concept system, as explained below in the Experiments section.

For all experiments in this project the *Legacy* ghost team controller has been used. This controller partly implements a randomised ghost behaviour. Additionally, all ghosts randomly reverse their movement direction on any game tick with a small probability. Moreover, the IEEE Ms. Pac Man simulator does not include any bonus fruits in the game.

IV. Genetic Programming Theory

GP, originally introduced by Koza [3] as a self-contained sub-domain of Evolutionary Computation, is, like other artificial EC techniques such as *Genetic Algorithms*, basically an approach to search for (near-)optimal solutions for a given

[2]http://www.pacman-vs-ghosts.net

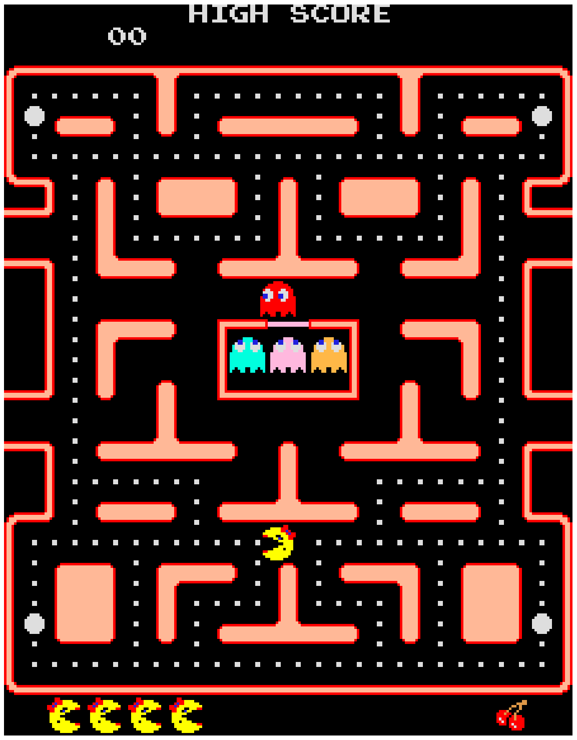

Fig. 1. Ms. Pac Man screenshot

problem using techniques similar to those found in the evolution process of real nature. In the case of GP such a solution is expressed as a computer program that can be interpreted or executed.

Typically, as first described by Koza, GP is implemented as a syntax tree. A good explanation of this representation can be found in [12]: The variables and constants in the program are the leaves of the tree, called *terminals*; (mathematical) operations are internal nodes called *functions*. The sets of allowed functions and terminals together form the *primitive set* of a GP system. Theoretically there is no limitation in the type of functions that can be used in a GP system. The definition of the terminal and function sets is up to the developer, it usually depends on the context of the application and is one of the most crucial steps in the design of a GP system.

An initial population of randomised individual programs in the mentioned tree representation is generated at first. Every of these programs are then executed and rated with the so called *fitness function*. Based on these fitness values suitable programs (usually those with a higher fitness value) are being chosen and used for creation of the next generation of programs. Such a new generation is created by recombination (*crossover*) and *mutation* of individual programs of the parent generation. Additionally, some of the individuals may also be propagated to the next generation without modification (*reproduction*). The evolved programs of the new generation are again being executed and assessed via the same fitness function as before, and a next generation is evolved. This process is repeated until a specific criterion is met, for example a pre-defined minimum fitness value or a maximum amount of generations (i.e. algorithm iterations).

V. Reactive Ms. Pac Man Control

A. Solution Design

As presented in section II above, various GP-based approaches for the autonomous control of the Ms. Pac Man game have been examined so far. Although these solutions of course follow their own ideas and design principles, they also share one common feature: two types of GP terminals are being used, one for game state information retrieval, and the other type for commands (actions) issued to the Ms. Pac Man character in order to guide her through the maze of the game world. The latter are based on an abstract view on the game environment. That is, instead of directly controlling Ms. Pac Man into movement directions UP, DOWN, RIGHT, LEFT, these action terminals guide Ms. Pac Man through the maze, for example, by letting her escape to a safe spot in the game world. In this example a "safe spot" is not a clearly defined area in the current maze, but a custom definition of the system designer of what "safe" actually is in the context of the controlled game. Such a GP-based controller therefore needs to include some kind of domain knowledge into these action terminals, which a GP system designer may want to avoid in general. Moreover, while this approach can lead to a good overall performance of the controller, it also creates a need for some kind of *action management*, as these abstract actions can last for a time span that is unknown a priori. In our example above the escape-action is not a one-step task, but needs to be updated with the current game state information continuously, as that is changing over time because of the moving ghosts. Moreover, in more complex situations, actions could theoretically also be stopped, paused, or subsumed by other actions, which would result in even more active monitoring and management of current, past, and future actions.

In order to address these issues the solution described in this paper follows another approach than the solutions presented above. First, the GP framework evolves a range of distinct individuals, solely based on game state retrieval terminals and basic mathematical operators, as listed below. For each of these individuals sample games are being played by the system. At every time step within such a game the valid movement directions (two or more of UP, DOWN, RIGHT, LEFT, if not blocked by a wall) are being rated, based on the final result of the individual's evolved program that is executed once for any non-blocked direction. The individual's program that has been executed results in a rating value for the according movement direction. Please refer to pseudo code listings 1 and 2 below for more details on this approach. The basic idea however is to let the GP framework find the optimal movement direction rating function on its own, solely using primitive mathematical operators, based on information about the current game state, without any high-level functions, such as *move-to-pill* or *escape-from-ghost*. We thus want to let the GP framework evolve the content for the function that combines current game state information to rate movement directions. This approach also takes into account that as a general design principle the complexity of a system should be kept to a minimum, as for example described in [13] and [14]. It is the main focus of the research presented in this paper to verify this assumption that even a simple reactive approach for a video game controller should be able to compete with the performance of an amateur human player.

Pseudo Code Listing 1: Main GP Framework.

```
1: initialise GP framework
2: while GP evolution is ongoing
3:   evolve new GP generation
4:   for each individual in new generation
5:     run new Ms. Pac Man game n times
6:     individual's fitness = avg. scores
7:   end for
8: end while
```

As described in the experiments section below, multiple sample games are being executed for each individual of a GP generation in order to determine their fitness value. Within one such sample game, executed n times on line 5 in listing 1 above, the algorithm in listing 2 is being executed.

Pseudo Code Listing 2: One Sample Game Algorithm.

```
1:  set controller = GP individual's parse tree
2:  start game's main loop
3:  at each time Ms. Pac Man has to move
4:    gather current game world state info
5:    transfer game state info to GP terminals
6:    for each valid movement direction
7:      run GP program for current direction
8:      save result as rating of direction
9:    end for
10:   determine best-rated direction
11:   move Ms. Pac Man to that direction
12: end of Ms. Pac Man movement
```

Note that this solution is a reactive (adaptive) controller, without any planning capabilities for future movement and anticipation of ghost behaviour, or memory of past actions. That is, the controller decides where to go next solely based on the current game state. This decision is calculated each time the main game loop calls the Ms. Pac Man control function.

B. GP Primitive Set

In general, a GP system is limited to its pre-defined function and terminal sets, thus it can only use these basic "building blocks" for creation of the solution programs (individuals). Other, possibly valid, solutions that would require different functions the developer has originally not thought of, can usually not be found by the GP algorithm. This paper therefore proposes that in some circumstances it may be advantageous for the overall solution to provide only low-level functions and terminals to the GP system and let it find a solution on its own, without any bias coming from the developer's definition of "good" functions the GP may need in order to find a proper solution.

The main drawback of this approach is that the lower-levelled the functions used by the GP algorithm are, the more complex the overall solution will become in order to express a high-level, abstract algorithm (as in any hand-crafted program as well). It is another aim of our work to verify this assumption in the context of Pac Man control. As described in the next section, experiments undertaken with the developed system lead to the conclusion that in this context a GP-based approach using only low-level mathematical and information retrieval operators can indeed find a proper solution program able to compete with other non-human Ms. Pac Man controllers (and even good novice human players).

As described above, only low-level functions are provided to the proposed GP solution. All mathematical calculations are based on floating point numbers and functions that represent the four basic arithmetic operations *add*, *subtract*, *multiply*, *divide*. The latter operator is implemented as *safe division*, as it returns 1.0 in the case of a division by zero. Additionally, operators for the calculation of *min*, *max*, and the *mean* of two numbers are provided to the GP search space.

The terminal set used for information retrieval consists of two major groups: terminals that return a single scoring value for a given direction based on a specific game state property, and constants that never change their values. The terminals that return a scoring value are all based on a specific direction from the view of the current Ms. Pac Man location. Their return values are normalised to a range of $[0.0, 1.0]$; the lower these values, the better these ratings[3]. The information retrieval terminals used in the proposed solution are:

- DistToNextPill – Distance to the next pill
- DistToNextPowerPill – Distance to the next power pill
- DistToEdibleGhost – Distance to the next edible ghost
- DistToNonEdibleGhost – Distance to the next non-edible ghost
- DistToNextCorner – Distance to the next corner
- DistToNextJunction – Distance to the next junction
- GhostBeforeJunction – Whether a hostile ghost blocks the way to the next junction
- PowerPillBeforeGhost – Whether a power pill is closer than a non-edible ghost
- GdPillCount – Amount of pills
- GdPowerPillCount – Amount of power pills
- GdEdibleGhostCount – Amount of edible ghosts
- GdNonEdibleGhostCount – Amount of non-edible ghosts
- GdJunctionCount – Amount of junctions

The values returned by these terminals are calculated by the controller at every time step it is called. The latter five, those with a "Gd" prefix, implement a look-ahead feature that requires special description. These terminals determine their return value by hypothetically moving into a direction, following each junction, for a total amount of k steps[4]. In other words: from Ms. Pac Man's current location check, by "looking" into the given direction for a distance of k, what the according Gd-terminal is intended for. So for example the

[3]This is of course a custom definition; we could instead also define that a higher value means a better rating and adapt the return value accordingly.

[4]Where k is a pre-defined value determined on a trial-and-error basis.

GdPillCount terminal counts the amount of normal pills that Ms. Pac Man would theoretically be able to collect if she would continue to move into the given direction for a distance of k steps.

The group of *constant terminals* on the other hand consists of the following values: *-1.0*, *0.0*, *0.1*, *0.5*, *2.0*, *5.0*, and *10.0*. These constant terminals may be used by the GP algorithm to modify the scoring values of the terminals described above.

C. GP Trees

By using the general GP methodologies described in section IV, based on the primitive set defined above, the solution presented in this paper evolves a GP program for every individual in the population. As described, such an evolved program is basically nothing else than a combination of various functions and terminals. In combination a selection of these primitives thus forms the GP tree of an individual. Each such tree returns a specific value, which is then used by the Ms. Pac Man controller as rating value for the given direction. Figure 2 shows a very simple example of such a GP tree that could be found the GP framework of the presented solution. The root node of this example tree represents the node that returns the final output value of this GP program. Note that the program itself does not has any side-effects, as none of the terminals defined above has any side-effects. Following the underlying idea of our solution design, we are solely interested in the outcome of the execution of a tree.

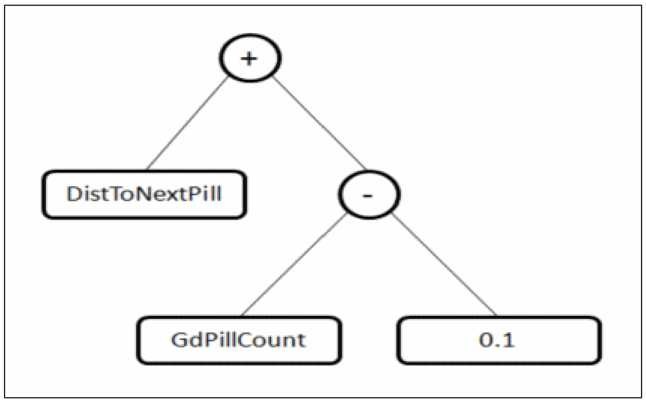

Fig. 2. Sample GP tree

VI. Experiments and Results

As in any other GP-based approach, the fitness of the candidate programs has to be evaluated via a domain-specific fitness function. The fitness value of a candidate program in the system presented in this paper is evaluated by running ten simulated games with the same controller and calculating the average of all ten final scores. As the evaluation of ten simulation runs per individual takes some time (in our experiments up to more than one second), the population size of one generation was capped to 100 individuals. Evolution was stopped as soon as the best all-time high score of all candidate programs remained unchanged for more than 50 generations. The other GP evolution parameters, such as the crossover and mutation probabilities, have been evaluated by following a trial-and-error approach. For the experiments presented in this section, these GP evolution parameters have been used:

- Min. initial tree depth: 2
- Max. initial tree depth: 5
- Crossover probability: 80%
- Mutation probability: 10%
- Reproduction probability: 10%
- Selection method: Tournament (10 individuals)

A range of experiments has been undertaken with different GP parameters and probability values set. A major issue that showed up during experiments is the very time-consuming fitness determination of all individuals of the population. For example, the evolution of one single generation of 100 individuals takes up to half an hour on the testing machine[5]. Total experimentation time is therefore very limited. Thus this section can only analyse experiments consisting of few simulation runs. More experimentation time and/or better (parallel) hardware is necessary to achieve more significant experimentation results.

A. Population Size

GP in general seems to benefit from a large population size. Koza [15] for example uses a population size of several thousand individuals in many of his experiments. This GP parameter is in particular essential for a system with a time-consuming fitness evaluation, as described above. Hence the first experiment evaluates the overall performance of the implemented GP system based on different population sizes during the evolution process. Table I shows the results of this performance evaluation, based on (the average of) five distinct evolutions per population size.

TABLE I
GP performance under different population sizes

Pop. Size	Avg. Generations	Avg. Fitness	Best Fitness
25 Individuals	168	15,404	24,120
50 Individuals	111	20,033	27,040
100 Individuals	118	22,026	27,740

B. Generalisation Ability

This experiment evaluates the generalisation ability of the implemented system. In general we want a GP algorithm to be able to work properly not only on the pre-defined fitness determination cases, but also on similar but untrained cases. A Pac Man controller thus should achieve good results in all mazes. Table II shows the results of the best individual for 100 simulation runs in each maze of the IEEE simulator (A, B, C, D). During the evolution (training) process these mazes are being loaded in the order A, B, C, D, A, B, and so forth. The fitness of a majority of all individuals in each generation

[5]Intel Core2 Duo CPU, 4 GB of RAM, Windows 7 Professional

is thus usually evaluated using mazes A and B, as of course fewer individuals are able to complete a maze and advance to higher levels.

TABLE II
PERFORMANCE OF BEST INDIVIDUAL IN ALL MAZES

Maze	Min. Score	Max. Score	Avg. Score
Maze A	2,420	40,640	18,992
Maze B	4,580	39,980	21,459
Maze C	2,350	32,320	16,871
Maze D	3,480	32,460	17,635
All Mazes	3,130	36,210	19,559

C. Controller Performance

Table III lists the performance of our best controller as well as the performance of Ms. Pac Man controllers referred to in section II. Note that these controllers have been evaluated on previous versions of the Ms. Pac Man game simulation system[6], hence comparability to our results is quite limited.

TABLE III
BEST GP INDIVIDUAL VS. OTHER CONTROLLERS

Controller	Min. Score	Max. Score	Avg. Score
Our GP-Controller	5,330	33,420	19,198
Controller in [4]	2,300	20,760	11,143
Controller in [6]	n/a	20,850	10,120
Controller in [7]	2,890	22,670	12,872
Controller in [8]	4,270	43,170	14,757

Overall, as listed in table III, our controller performs best, even though other approaches might produce better results in single games. It seems, as nearly all of our experiments have shown as well as demonstrated by these listed controllers, that for a game like Ms. Pac Man it's the amount of good results over a range of games that matters.

In another experiment the performance of the best evolved individual has been compared to four amateur human players. Each player played ten simulation runs. The result of this experiment is shown in table IV.

TABLE IV
BEST GP INDIVIDUAL VS. HUMAN PLAYERS

Controller	Min. Score	Max. Score	Avg. Score
Our GP-Controller	5,330	33,420	19,198
Human Player 1	7,860	26,030	16,455
Human Player 2	1,110	32,250	15,089
Human Player 3	3,210	14,520	7,331
Human Player 4	5,060	16,940	9,437

D. Sample Evolution Progress

A typical evolution progress chart is shown in figure 3. The fitness value of the best individual in this sample run is 27,040. Note that this value is not the best score of one single simulation run, but the average of ten runs using the same evolved controller. The population size of this evolution process was set to 100, it was stopped after 50 generations in which the fitness value of the all-time best individual did not change. The chart shows that after one third of the evolution progress the average fitness of all individuals did not significantly change any more, but still better individuals are being evolved from time to time.

[6]http://dces.essex.ac.uk/staff/sml/pacman/PacManContest.html

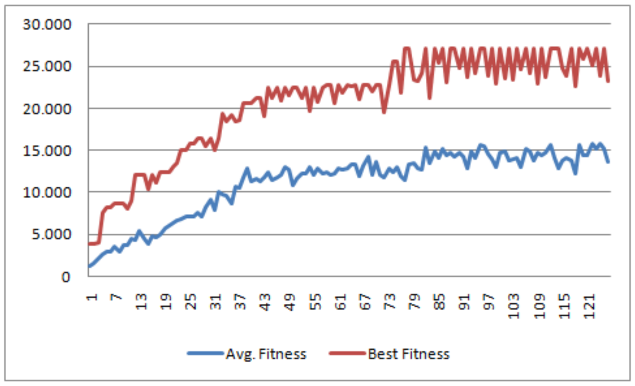

Fig. 3. Sample evolution progress

E. Analysis of Experiment Results

As the experimental data shown above demonstrates, the same controller may achieve results from about 5,000 to 35,000 points in two consecutive simulation runs. It seems that this huge difference is not only a characteristic of the solution presented in this paper, as those GP-based references mentioned in section II suffer from the same problem. Moreover, as various sample games played by human players during our experiments have shown, also good human players may finish a game with low scores from time to time. We thus propose that a game controller may also achieve bad results in some sample games, it's the amount of these bad samples in comparison to the total amount of sample games that matters. For the same reason the average of multiple sample games is used to determine one individual's fitness value.

As it turned out during experiments with the presented solution, adding boolean-based logical operators, such as *if-then-else* statements, did not help the GP algorithm to find better solutions. In fact, more functions and terminals increase the search space, which in general makes it harder for a GP system to find a good solution [16]. For the implemented solution thus no boolean-based logical operators have been used.

An interesting feature emerged in all evolved individuals with a high fitness value among all experiments: after all power pills have been consumed within a maze, the evolved Ms. Pac Man controller seems to ignore any of the remaining normal pills, it simply tries to avoid all ghosts. Within the Ms. Pac Man version of the IEEE simulator such a behaviour makes sense, as after a while the current maze is automatically finished, and all remaining pills are awarded to Ms. Pac Man.

Because of this fact it makes not much sense to risk a life by moving towards a normal pill and probably close to a ghost, hence the GP algorithm has correctly identified this unique characteristic of the game simulator.

Multiple different experiments with the proposed system also showed that the best evolved individual programs did not use the *DistToNextCorner* terminal. This is surprising, as in many different sample runs the Ms. Pac Man character died close to a corner, because in all mazes there is a long hallway without junction or escape for the Ms. Pac Man character at each corner. Thus it should be an important information for the controller which movement direction leads closer to a corner. More experiments must be undertaken to evaluate whether or not the assumption is true that the probability of Ms. Pac Man's death is increased closer to a maze corner.

Overall, the highest score achievable by the best evolved individuals seems to be capped at around 40,000 points (or about 30,000 points in average); none of any GP solution programs was able to get significantly more points than that. This fact leads to the question whether or not a higher score could theoretically be achieved by an evolved controller based on a design as proposed in this paper. It could, for example, be possible that better results could be achieved by using much larger populations (with many thousands of individuals). This would of course require faster hardware, possibly with support for large-scale parallel processing. Another possible improvement for the proposed solution design might be to add more or replace existing functions and/or terminals. However, it could also be possible that this approach is not able to achieve significantly more points. In that case it could be necessary to implement some mechanism of planning and high-level movement control. Further research is necessary to examine this issue.

VII. Conclusion

This paper has presented a novel approach of Ms. Pac Man control based on Genetic Programming. In comparison to directly control the Pac Man character based on abstract movement operations, the approach discussed in this document simply relies on information about the current game state in order to find the optimal movement direction at every iteration of the main loop of the game. The experiments undertaken with the implemented proof-of-concept system and their results analysed in this document demonstrate that the proposed approach is able to evolve controllers that can play Ms. Pac Man games on the level of a good beginner human player.

One of the main characteristics of nearly every GP system is the fact that its performance heavily relies on the parameters set for the evolution process. Since the approach presented in this paper aims at the design of a reactive controller for the Ms. Pac Man game, GP parameter tuning was not the main focus of our work. In a subsequent project we would therefore like to improve the overall system and game playing performance by systematically tune parameters of the GP evolution process, for example by using a Genetic Algorithm. In addition to that, a subsequent project could evaluate the performance of the presented approach for a system based on GP and multiple objectives, namely the evolution of a controller for the four ghosts in the Ms. Pac Man video game. Additionally, as the proposed system is a reactive controller solely based on information about the current game state, it would be interesting to evaluate whether or not higher scores can be achieved with the proposed solution (for example by evolving much larger populations on faster, parallel hardware), or if some kind of long-term planning and abstract movement control is necessary in order to achieve more points.

VIII. Acknowledgements

The authors would like to thank Mr. Simon M. Lucas, Mr. Philipp Rohlfshagen, and all others who have been involved in the creation of the Ms. Pac Man simulation system used for the work presented in this paper.

References

[1] M. J. Wolf, Ed., *The Video Game Explosion: A History from PONG to PlayStation and Beyond*. Greenwood, 11 2007.

[2] N. Bell, X. Fang, R. Hughes, G. Kendall, E. O'Reilly, and S. Qiu, "Ghost direction detection and other innovations for Ms. Pac-Man," in *Computational Intelligence and Games (CIG), 2010 IEEE Symposium on*, aug 2010, pp. 465–472.

[3] J. R. Koza, *Genetic Programming: On the Programming of Computers by Means of Natural Selection v. 1 (Complex Adaptive Systems)*. MIT Press, 2 1992.

[4] A. Alhejali and S. Lucas, "Evolving diverse Ms. Pac-Man playing agents using genetic programming," in *Computational Intelligence (UKCI), 2010 UK Workshop on*, sept 2010, pp. 1–6.

[5] M. Gallagher and M. Ledwich, "Evolving pac-man players: Can we learn from raw input?" in *Computational Intelligence and Games, 2007. CIG 2007, IEEE Symposium on*, april 2007, pp. 282–287.

[6] M. Emilio, M. Moises, R. Gustavo, and S. Yago, "Pac-mant: Optimization based on ant colonies applied to developing an agent for ms. pac-man," in *Computational Intelligence and Games (CIG), 2010 IEEE Symposium on*, aug 2010, pp. 458–464.

[7] B. Tong and C. W. Sung, "A monte-carlo approach for ghost avoidance in the ms. pac-man game," in *Games Innovations Conference (ICE-GIC), 2010 International IEEE Consumer Electronics Society's*, dec 2010, pp. 1–8.

[8] D. Robles and S. Lucas, "A simple tree search method for playing Ms. Pac-Man," in *Computational Intelligence and Games, 2009. CIG 2009, IEEE Symposium on*, sept 2009, pp. 249–255.

[9] A. Fitzgerald and C. Congdon, "RAMP: A rule-based agent for Ms. Pac-Man," in *Evolutionary Computation, 2009. CEC '09, IEEE Congress on*, may 2009, pp. 2646–2653.

[10] N. Beume, H. Danielsiek, C. Eichhorn, B. Naujoks, M. Preuss, K. Stiller, and S. Wessing, "Measuring flow as concept for detecting game fun in the pac-man game," in *Evolutionary Computation, 2008. CEC 2008, IEEE Congress on*, june 2008, pp. 3448–3455.

[11] Y. Hao, S. He, J. Wang, X. Liu, J. Yang, and W. Huang, "Dynamic Difficulty Adjustment of Game AI by MCTS for the game Pac-Man," in *Natural Computation (ICNC), 2010 Sixth International Conference on*, vol. 8, aug 2010, pp. 3918–3922.

[12] R. Poli, W. B. Langdon, and N. F. McPhee, *A Field Guide to Genetic Programming*. Lulu Enterprises, UK Ltd, mar 2008.

[13] S. V. Buldyrev, R. Parshani, G. Paul, H. E. Stanley, and S. Havlin, "Catastrophic cascade of failures in interdependent networks," *Nature*, vol. 464, no. 7291, pp. 1025–1028, Apr. 2010.

[14] J. A. Rijpma, "Complexity, tightcoupling and reliability: Connecting normal accidents theory and high reliability theory," *Journal of Contingencies and Crisis Management*, vol. 5, no. 1, pp. 15–23, 1997.

[15] J. R. Koza, *Genetic Programming II: Automatic Discovery of Reusable Programs*. MIT Press, 1994.

[16] M. Ebner, "On the search space of genetic programming and its relation to nature's search space," in *Evolutionary Computation, 1999. CEC 99. Proceedings of the 1999 Congress on*, vol. 2, 1999, pp. 1357–1361.