Creating AI Characters for Fighting Games Using Genetic Programming

Giovanna Martínez-Arellano, Richard Cant, and David Woods

***Abstract*—This paper proposes a character generation approach for the *M.U.G.E.N.* fighting game that can create engaging AI characters using a computationally cheap process without the intervention of the expert developer. The approach uses a genetic programming algorithm that refines randomly generated character strategies into better ones using tournament selection. The generated AI characters were tested by 27 human players and were rated according to results, perceived difficulty and how engaging the gameplay was. The main advantages of this procedure are that no prior knowledge of how to code the strategies of the AI character is needed and there is no need to interact with the internal code of the game. In addition, the procedure is capable of creating a wide diversity of players with different strategic skills, which could be potentially used as a starting point to a further adaptive process.**

***Index Terms*—AI, character, fighting games, genetic programming (GP).**

I. Introduction

THE development of AI characters has played an important part in the continual evolution of more realistic gaming environments. A main goal of these characters is to provide human-like behavior that can lead to engaging and challenging games [1]–[3], which in turn leads to commercial success. One of the main aspects that has been studied in the last decade to improve the gaming experience is the incorporation of adaptive behavior into the characters. It has been reported that characters that can adapt to the players level, personalizing the game, are more satisfying to play than characters that are unbeatable or too easy to beat [4].

The AI of most commercial games is based on a number of very well established techniques, which are simple and processor efficient, such as manually coded finite state machines (FSM) [5]. In fact, many game developers rely exclusively upon these. This is a major limitation due to their deterministic nature. Buro and Furtak [2] provide insight into the relative lack of sophistication of current generation AI methods used, stating that most companies create titles under severe time constraints and do not have the resources and the incentive to engage in AI research. However, the authors also acknowledge that multiplayer games often do not require high AI performance in order to become a commercial success as long as there are enough human players interested in playing the game online. The competitive environment online, nevertheless, can be seen as a deterrent to participation for new players. To overcome the limitations of FSM approaches, there has been some work on the use of machine learning techniques such as reinforcement learning [6], artificial neural networks (ANN) [7], and Markov models [8] to improve the AI characters strategy in real-time. Until now, little effort has been made to incorporate these into commercial games. The idea of using nondeterministic approaches is something the game industry is still approaching slowly.

Although the adaptive methods mentioned previously have shown progress on the development of more engaging game play, the time and resources taken to create such AI structures is not commensurate with the level of improvement they bring to the game as a whole. A general drawback of this kind of approach is that at the early stages of the game, characters may have a very erratic behavior, needing a certain amount of time to develop into challenging players. A more "intelligent" character at the early stages of the game could certainly overcome this shortfall. However, as is well known, creating these characters will depend on the time and expertise of the game developer.

In this paper, an approach for the creation of AI characters using genetic programming (GP) is presented. GP is used as a refinement process of the character fighting strategies which are initially created by a random process. The fitness (ELO rating) of the characters is estimated according to a set of matches played using the *M.U.G.E.N.* engine as framework. The main objective is to be able to build AI characters that are interesting to play without the need of the expert developer. These would be created in an offline process and, additionally, they could be used as a starting point of a possible online adaptation of the same evolutionary approach.

II. Related Work

The widely used state machine is itself a result of progression within the field of gaming AI. According to Kehoe, the initial gaming AI creations were rule-based systems, the most basic form an intelligent system can take [9]. Games such as *Pac-Man* are an example of rule-based AI systems, where the four pursuing "ghosts" make navigational decisions based upon simple rules and the position of the player [10], [11]. Kehoe presents FSM as a development of rule-based AI systems, as a FSM can evaluate many rules simultaneously and factor in the current state of the AI. Each "state" can have a completely different set

Manuscript received September 7, 2016; revised January 27, 2016, June 2, 2016, and October 5, 2016; accepted December 12, 2016. Date of publication December 20, 2016; date of current version December 13, 2017.

The authors are with the School of Science and Technology, Nottingham Trent University, Nottingham NG11 8NS, U.K. (e-mail: giovanna.martinezarellano@ntu.ac.uk; richard.cant@ntu.ac.uk; david.woods.home@hotmail.co.uk).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TCIAIG.2016.2642158

of rules to determine behavior, with the transitions managed by the state machine itself. The author suggests that the next step is the creation of an adaptive AI. An adaptive AI is defined as an AI which takes into account past interactions in determining future behavior, either via gathered historical data or real-time evaluation of the success/failure of randomly chosen actions.

A first step toward adaptive AI in commercial games was the release of the *Virtua Fighter 4* game for the Sony Playstation 2. The game allows for an "AI Construct" to be trained by the player via imitation. This approach in practice proved unpopular. It took excessive amounts of time for a player to teach the AI its basic capabilities, and further hours spent watching the game play itself. Even so, it did not yield compelling gameplay. The feature was then removed on the following release, and has not re-emerged in any other commercial fighting game.

Most of the research carried out on adaptive AI characters has been done using independently coded fighting games, which, with some exceptions [12], do not adequately replicate modern commercial fighting games. Danzi *et al.* developed an adaptive AI using reinforcement learning in an attempt to develop adaptable difficulty levels. They implemented their own fighting game, Knock'em, which is more representative of commercial games than other research implementations, as it allows for a wider range of moves. To allow the adaptation of the characters, positive reward is given if the AI character has more health that the human player. Prior to the evaluation, the AI character was trained against a random fighter in 500 fights. According to their experimental results, their adaptive characters were able to match other standard AI characters despite losing most matches.

Graepel *et al.* applied reinforcement learning to find near optimal policies for agents acting in a Markov decision process (MDP) [8]. They used the commercial game Tao Feng as a test bed. This game, based on a nondeterministic FSM, provides a wide diversity of fighting characters and styles and has over 100 different actions available to the player. They used two types of reward functions depending on the change in health of both combatants. Their results successfully demonstrated that from a starting random policy, agents could change their behavior to adapt to the built-in AI and expose/exploit gaps in the rule-based built-in AI.

Ricciardi and Thill also studied the possibility of adaptive techniques for generating functional fighting AI using an enhanced MDP [13]. Their idea was to have the AI adapt to varying strategies carried out by the human player in real time. This was challenging due to limited availability of training data. The AI characters were provided with only a few negative rules about the game to eliminate a large empty area of the search space, but given no further information to the character. The adaptive characters were tested using a simplified facsimile of a commercial fighting game and it was observed that the AI could gradually improve as it gathered more data from human opponents. The characters were also tested to recognize human player patterns according to past player models, in order to offer a better starting position for the AI. Despite achieving good results against static AI characters, no conclusions could be made when playing against human players, as the human players changed tactics too frequently to allow the AI to create a consistent model.

Ortiz *et al.* propose an adaptive character with a three subagent architecture that can adapt to the level of the user using reinforcement learning [6]. Positive reward is issued at the end of each round when the difference in health between the AI character and the human player is small and negative when it increases. This type of measure allows the algorithm to identify how closely match both players are. The closer they are, the more engaging the game is. This reward strategy is different from the one implemented by Danzi, who focused on rewarding those characters with higher health. Ortiz *et al.* carried out a set of experiments involving 28 human players who were asked to play between 15 and 30 rounds against a set of AI players (both static and adaptive). Although the authors argued that the adaptive players had the least negative results, the most positive reviews were given to a strong static player.

Another platform that has been created as a test bed for fighting games AI is the FightingICE platform, which was first introduced in [14] and further used in [15]. This platform was created as a tool for game competitions allowing students to test their codes. Yamamoto *et al.*, who created the platform, later presented an AI of their own that uses the k-nearest neighbor algorithm to predict its opponent's attack action and devise an effective countermeasure against the predicted attach. The authors tested their AI against winners of previous FightingICE competitions and demonstrated the strength of their AI, beating all other AI characters. Despite their encouraging results, there was no experimentation carried out with human players, and hence no conclusions can be made about how engaging their AI characters are.

An alternative approach to AI improvement in fighting games was constructed by Thunputtarakul and Kotrajaras [16], utilizing an emulated version of a commercial title in the Street Fighter series. This approach involved the construction and training of a "Ghost" AI, an AI structure that could rapidly observe actions taken by a player and assess their priorities, yielding as a result an AI which could reproduce its creator's playing styles even when the training time was short. Despite the successful results, there were some limitations to the technique such as "unlearning" styles previously encountered, rendering the AI vulnerable to a human player who simply altered their tactics. Further work in this area was developed by Saini [17], who, unlike Thunputtarakul, designed a k-nearest neighbor solution using in-game parameters, which are driving factors for the behavior and strategies exhibited during the match. In order to gather data from the player being mimicked, the suggested approach observes the human player and records its movements and the parameters of the game. This enables it to understand under what circumstances each strategy is used. The data is analysed in an offline mode and a strategy is selected using the k-nearest algorithm or a data-driven FSM. The data capture phase is critical in this approach, as one needs to make sure the human strategies can be captured, needing a certain number of matches to record the human player moves. Saini tested both of his proposed strategies against a human player and used human observers to evaluate the moves of the AI. According to those

observers, both strategies were capable of mimicking human strategies. A main limitation of these approaches is that they rely on the strategic play of human players. This was an evident limitation when the human player changed strategies back and forth frequently. Also these methods require the human player to spend time in potentially unrewarding ways. Miche *et al.* present the use of extreme learning machine (ELM) as an alternative to reduce this time and improve the player's experience. ELM needs less training data compared to a classical ANN in order for an agent to learn a "proper" behavior [18]. Hou *et al.* also addresses this issue, developing strategies which allow agents in a first-person shooter game to "imitate" another agent's behavior through the use of internal and external evolution of memes [19].

While adaptive and learning AI structures have been implemented within fighting games on an academic level, only a single commercial title has utilized these methods with limited success. Game developers have turned to alternative methods of improving AI interaction experiences such as strategic game design choices to maximise AI strengths and minimise failings, rather than invest large amounts of time and resources into complex AI with uncertain returns. This has led to the development of frameworks which allow the quick creation and behavior control of AI [18], [20]. Evolutionary techniques have been particularly used for the offline development of AI characters. The aim of the evolutionary process is to find those character moves that would be more effective in certain environment settings of the fighting game. Byrne *et al.* have applied a genetic algorithm (GA) approach to develop moves that respond to a certain game situation using the game Toribash as test bed [21]. This game, however, is different from commercial fighting games as the player needs to select a set of character joints to perform a move, rather than using pre-existing ones. The GA codifies a set of joint moves as a solution to a given scenario (opponent's last move, distance, etc.). These solutions are evolved by playing fixed opponents for several generations. A similar approach is presented by Cho *et al.*, where the individuals evolved represent the different actions to be performed by the character according to opponent's information, both codified in the chromosome [22]. The authors compare this approach to ANNs and evolutionary neural networks (ENN) using their own video game. The authors also test the inclusion of the opponent past moves codified in the chromosome. Those individuals that "remember" previous opponent moves tend to be more effective, however, the evolutionary process using larger chromosomes takes longer to run. The authors conclude that ENN are the best approach to develop AI characters both in terms of time convergence and player quality. Neither of these evolutionary approaches were tested for human player satisfaction.

GP, a population-based evolutionary algorithm, has been successfully and widely applied to the creation of competitive teams for cooperative video games, such as soccer teams [23], [24] and for individual games such as backgammon [25] and chess [26]. To the best of our knowledge, GP has not been applied to the creation of AI characters for fighting games. There are several reasons why it would be interesting to study this approach. As for any population based algorithm, it is able to produce a set of different solutions, in this case characters, allowing the programmer to select a subset of individuals based on the requirements of the particular situation. Another advantage of this approach is that there is no need to explicitly set up any rule or strategy to the AI character, avoiding the need of experienced game developers.

Fig. 1. Snapshot of the *M.U.G.E.N.* game.

In order to explore the potential of this approach, it was important to find a suitable fighting game to act as a test bed. Most of the test beds found in the literature are in-house implementations, therefore, comparison between different AI techniques is not straightforward. For this reason, we were interested in finding an engine that was currently widely used and supported, in order to facilitate the evaluation of our approach by matching our AI against publicly available AI. A suitable approximation to commercial developments was found in the *M.U.G.E.N.* engine. It was therefore a matter of coupling the GP algorithm with the engine in order to evolve AI characters. Due to the way in which this engine is implemented, it is not possible to adapt AI characters during the matches, so the engine could only be used to evaluate the characters ability. This required a certain number of design choices, adaptations and solutions that will be explained in detail in the following sections. To the best of our knowledge, no other AI techniques have used this engine as test bed.

III. M.U.G.E.N. Engine

Initially released by the company Elecbyte in 1999, the *M.U.G.E.N.* engine was designed as a fully customizable fighting game engine, intended to allow users to create their own game elements and playable characters, while retaining and functioning exactly in the same manner as a commercial release. The ability to script a custom AI to control these characters was added in the 2011 release, which uses a programmable FSM to dictate the AI's actions. The customization of the AI is allowed, however, only before or after a match, and not in between rounds. Fig. 1 shows a screen snapshot of the game.

The AI within the engine revolves around the concept of two types of state—"change" state and "move" state. Each move available to the character has a fixed move state number, which when entered will play out the move's animation and create

the requisite hit/hurtboxes that constitute the core mechanics of the game. Each move also possesses a change state, which defines the conditions when it will be executed, typically falling under the category of input triggers such as "The player has just pressed button 1, perform the move associated with move 1" or logical triggers such as "If the player has been knocked over, pressing button 1 will not execute a move, they must wait until their character stands up." These triggers form the core of the control mechanism of the game. A custom AI within the engine either replaces or adds to the triggers within each move's change state, allowing for automated behavior to occur. For example, a logical trigger "If the opponent is within 100 pixels, then perform move associated with button 1" could be added to the single input trigger. Each action a character can take could then possess a single input trigger and many logical triggers.

It should be noted that the flaws observed in a fighting game AI are easily attributable to imperfect implementations of a FSM; lack of adaptability, state transitions impossible for a human player, perfect reaction times. In order to adequately implement the GP approach, certain requirements were defined as an attempt to mitigate some of the FSM limitations. First, characters must adhere to all of the limitations and constraints applied to a human player of the game. It must not be possible for an AI to utilize a move or tactic which a human player could not physically recreate. Second, characters must not possess faster reaction times than would be possible for a human player. The implementation of the GP approach would need to incorporate these requirements in order to evolve valid solutions.

IV. Implementation of the GP Approach

GP [27] is a biologically inspired computation technique based on the evolution of individuals (i.e., programs) over time, through events such as crossover and mutation. Each of these programs represent a set of instructions to solve a specific problem. In a typical GP implementation, solutions are coded in a binary tree structure where interior nodes represent functions/logical operators and leaves represent terminals (data on which functions are applied). In the case of the *M.U.G.E.N.* AI, each program (character) is composed of a set of sequential blocks that represent the fighting strategy. In our approach, these blocks form the genetic code that underlies the evolutionary process. The binary representation was therefore replaced by a sequential one, where each node represents a block or move which can be randomly mutated over time. The components of the GA were implemented within the *M.U.G.E.N.* system as follows:

A. Creation of Initial Population

In order to create valid characters for the initial population of the evolutionary process, it is necessary to randomly generate valid groups of triggers to place within the change states of a character. To do this, a template version of a state definition must be created. This template retains all of the change states present in the original functional character, but removes any input triggers and replaces them with dependency upon an AI variable (a flag variable that renders the character unresponsive to player input and grants sole control to the AI).

In order to be able to apply the genetic operators upon the generated AI, it was necessary to define the "terminals" or potential actions that an AI could perform, which would be manipulated by later genetic operators. The structure of the *M.U.G.E.N.* engine custom AI made this a relatively simple process. Each move available to the character is already isolated into a unique block of code. The default character provided within the *M.U.G.E.N.* engine possesses 34 such terminals, each relating to either a move (punch, kick, throw, sweep, etc.) or an action (run forward, jump, run backwards). Thus, if suitable randomly selected triggers were to be placed within each of these terminals, a functional AI would be generated with full access to all the moves a player of that character would have. This fulfills the first requirement stated previously regarding the incorporation of only human feasible moves into the AI character.

Each of these blocks of code would then contain between one and four trigger points; triggers that when combined, represent one instance where the move can be performed. Within each trigger point, it was necessary to generate a trigger or combination of triggers which would cause the AI to perform the change state. A full list of the triggers recognized by the internal workings of the *M.U.G.E.N* engine is provided externally by the creators of the engine [28]. There are 113 in total. We removed the triggers that are purely intended as system functions or those irrelevant to the character used for testing (AILevel, ParentDist, IsHelper, and others). The triggers that were retained consist only of information visible to a human player examining the screen. They are as follows:

1) P2BodyDistX, P2BodyDistY: These return the relative position of the opponent on the screen in the *X* and *Y* directions. Inclusion of these triggers potentially allows an AI to see where its opponent is and act accordingly.

2) GuardDist: This trigger returns true if the opponent has entered an attacking state and the AI is within distance to guard it. By including this trigger, the AI can potentially become aware that it is currently being attacked.

3) P2MoveType, P2StateType: These evaluate the current state the opponent of the AI is in, in terms of both movement (crouching, standing, jumping) and state (attacking, idle, being hit). These triggers allow the AI to react based on the current actions of the opponent in a wider context than the earlier GuardDist trigger.

4) MoveContact, MoveGuarded: These return true when the AI has either connected with an attack move, or has successfully blocked an attack move by the opponent. The former allows an AI to detect when it has gained an advantage and potentially follow up successful aggression with extra actions. Conversely, the latter allows it to detect when it has successfully defended against an attack and enact retaliation.

These seven triggers form the basis of the observation ability of the generated AI within the game world. While much smaller than the full list, this is enough to allow an AI to observe and react, forming a randomly generated "strategy" that it would use to play the game.

The game engine reads the change states within the AI code sequentially and halts when one is activated, so terminals located at the end of the file would be less likely to be checked by the engine, reducing the probability that their moves will be performed. This means that the order in which the terminals are placed is important, not just their content. Therefore, both the contents and the order of the terminals should be randomized. Every character is coded in a *.cmd* file which contains a general header, a set of five fixed terminals and then the randomized terminals. Each terminal consists of the specification of the action to be carried out when it is activated followed by a list of triggers. The list of triggers is effectively a standard sum of products ended with some overall conditions.

The game engine checks change states approximately every 1/60th of a second. If conditions arise whereby a change state would be activated, it will occur and be enacted with a time far shorter than a human player could achieve. In order to increase these AI reaction times, each trigger group within a terminal needs to include a random chance of activation. Introducing a 50% chance of activation does not mean that the move is used only half of the time it is appropriate. Given the frequency that the engine checks every state, it means that on average the time taken for the engine to enact that change state under suitable conditions is increased by 50%, simulating a reaction time element. Furthermore, these random number checks were subject to further random variation; a high probability of success would mean the move would be used more quickly and frequently, while a lower chance would mean the move was available, but often neglected in favor of alternatives. Randomising these factors in an AI has numerous benefits with regards to AI quality. If the character can favor moves, it can be seen to have a "play style," and if it has variation in its timing, it can be seen as more "human" in its ability to react to the game world.

B. Definition of a Fitness Function

In order to evaluate whether a generated AI is performing well, a fitness function must be defined. The system chosen would need to be robust, designed to provide a comparative evaluation of ability and be capable of functioning in an environment where new entities are added or removed frequently. It should also allow for variance in individual performance, producing results over many matches rather than relying on a single Boolean win/loss performance condition as the basis of its rankings. This is due to the fact that testing a fighting game AI character for competence is a more difficult process than simply applying the results of a deterministic equation. The matches themselves are not solely dependent upon the actions of one player; the reactions of the second player also heavily contribute to success or failure. In addition to this, the actions of the characters are randomized and so each bout between any given pair of characters will be different.

For the reasons stated above, the ELO system was selected. This ranking system is currently used by the United States Chess Federation for worldwide ratings of player skill. It has also been adopted by many popular competitive multiplayer games such as Backgammon, Go and Scrabble, and to videogames across several genres, including but not limited to League of Legends, Counterstrike, and Pokemon, with some variations in their implementations. The ELO system works upon the principle that the performance of a given player in any given match is a random variable within a normal distribution, and that the overall rating of a player needs to take into account the skill of the opponent as well as whether a win or lose outcome is reached. An equation is applied to current ratings to predict the probability of one player securing a win against the other. The results of this equation are then used to calculate how many points will be added to the rating of the winner and how many will be subtracted from the rating of the loser. In order to calculate the expected score of a player the following formulas are used:

$$E_{P1} = \frac{1}{1 + 10^{(R_P 2 - R_P 1)/400}} \tag{1}$$

$$E_{P2} = \frac{1}{1 + 10^{(R_P 1 - R_P 2)/400}}. \tag{2}$$

P1 and P2 corresponding to player one and player two and R1 and R2 to their corresponding current ratings. After a match is played between the two players, the ratings are adjusted by an amount proportional to the difference between the expected score and the actual result. The formula used is

$$R'_{P1} = R_{P1} + K(S_{P1} - E_{P1}) \tag{3}$$

where K is a factor that controls the strength of the adjustment and S is a Boolean indicating a win or a loss. These are the standard ELO formulae. We have used a K factor of 50, which is accepted as a reasonable value when the ratings are not already well established.

The initial score given to all characters after creation is set to 1000. Next, a set of matches is carried out to find their actual ratings. A ranking of 1000 would remain the average rating regardless of how many matches were played. The stronger AI characters would be expected to achieve a rating above 1000 whilst the weaker ones would be below 1000. For each of the generations of the evolutionary process, this process is repeated but any characters that are retained unchanged will start from their existing rating.

In order to define how many matches a character should play in order to arrive to a realistic score in comparison to other characters within the same generation, a set of experiments was carried out. First an initial population of 30 individuals was created, setting their initial ranks to 1000. A round robin tournament was then performed, each individual playing a total of 29 matches. The change in the rank is shown in Fig. 2. It can be observed that after the first five matches, is difficult to identify good players from poorly performing ones. Somewhere between match 10 and match 15 a consistent pattern starts to emerge.

After the first round robin, two more round robins were performed to observe if there were any further changes in the scores. Results are shown in Fig. 3.

As it can be seen, most of the individuals show a reasonably stable score after 30 matches, with some exceptions. A second experiment was carried out by taking only two individuals to allow us to observe how the win ratio would evolve over a larger

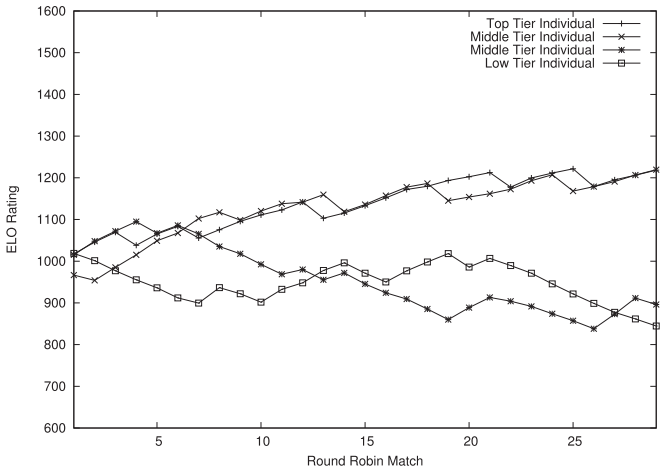


Fig. 2. ELO rating of four individuals during a first round robin. The top tier individual corresponds to the best individual so far in the league. Middle tier individuals are positioned in the middle of the league table. The low tier individual is the worst so far in the league.

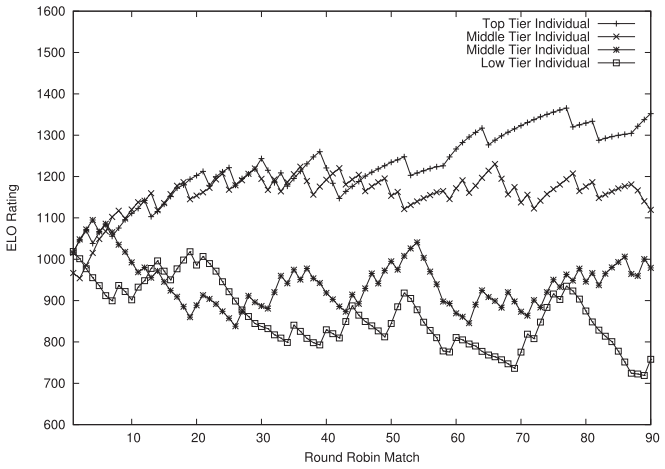


Fig. 3. ELO rating of four individuals during 3 round robin tournaments. The top tier individual corresponds to the best individual so far in the league. Middle tier individuals are positioned in the middle of the league table. The low tier individual is the worst so far in the league.

number of matches. Two randomly selected players from the previous 30 were chosen and set to play 90 matches. Fig. 4 shows how the percentage of matches won changes and eventually stabilises.

Based on these results, it was then decided to set the number of matches for each player per generation to 30.

Next, we needed to define the way the opponents for those matches were to be selected. In later generations, some of the players will already have a rating because they have been carried over unchanged from an earlier generation (see Section C below). Because we needed to keep the total number of matches per player down to 30, we could not hold a full round robin tournament. Instead, the population was grouped into three approximately equal divisions. The elite players from earlier generations were placed together to enable their relative rating to be refined further. Each individual was set to play a match against

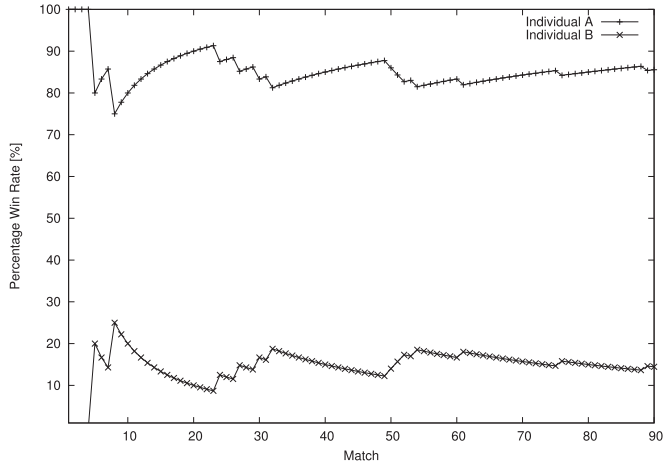


Fig. 4. Percentage of games won between two players during 90 matches.

a random opponent within the division. After each match the ratings were recalculated, new divisions were created and the process was repeated. After 30 matches the ratings were used to sort the individuals into fitness order to facilitate the selection process as described below.

C. Defining a Selection Method

The algorithm uses three parameters in order to create each new generation; one to select the number of individuals that will survive (elitism), a second one to determine the number that will be created by crossover, and a third one to determine the number of randomly created new individuals. The randomly created individuals will keep some diversity in the population. The elitism will allow the individuals with high ranking to survive. Also, highly ranked individuals will be used to create new individuals in future generations by crossover. These parameters can be set to different values and only an experimental process would allow to determine which are the most adequate. For the purposes of obtaining some preliminary results, these were set to 5% for elitism, 5% for new randomly generated individuals, and 90% for new individuals from crossover and mutation.

To determine which individuals will survive and which ones to select for crossover, the population is first sorted by rating following the matches performed for the fitness recalculation. After sorting, the percentage of individuals to survive will be taken from the "league table" starting from the top and copied to what will become the new population. The new individuals created by crossover and mutation are generated by randomly selecting two parents from the top third of the population. Each pair of randomly selected individuals are used to generate two new ones. Mutation is applied with a very low probability to the newly generated individuals and then they are integrated to the new population. Finally, in a third step, the set of new randomly generated individuals are added to the new population. Once the new population is complete, it replaces the previous one.

D. Genetic Operators

The use of genetic operators is essential for the evolutionary process as it will allow the creation of fitter individuals. This is

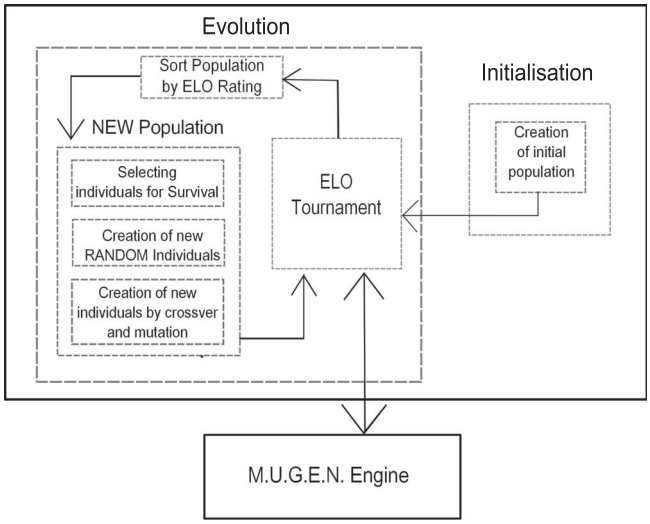


Fig. 5. Illustration of the GP implementation and its interaction with the *M.U.G.E.N.* engine.

achieved by extracting and carrying the good aspects of current individuals into future generations. The crossover is performed by selecting genetic material from two individuals and combining them to generate a new one. For this application in particular, the terminals, which correspond to the blocks or moves of the characters, are selected randomly from each of the parents. For example, a first child could be created from two parents with four terminals each by randomly selecting terminals 1, 2, and 4 from the first parent and terminal 3 from the second. Conversely, the second child would possess terminals 1, 2, and 4 from the second parent and terminal 3 from the first.

As mentioned previously, the order in which terminals are located in the file will impact the performance of the AI. For this reason, the implementation of the mutation was carried out by taking the newly generated individual and randomising the order of the terminals. As in any other application of GP, mutation is a parameter to be determined experimentally. although the general approach is to set it to a small value. The next section will present more detail on this aspect.

Because of the way these genetic operators work neither the number of terminals nor their internal complexity can increase. Consequently the common problem that exists in GP of over-complex solutions cannot arise here.

An illustration of the GP approach for AI character creation is given in Fig. 5. It can be observed that there is a continuous interaction between the GP and the game engine as information is gathered from every match in order to calculate the fitness of each individual at each generation. The process stops after a fixed number of generations.

V. Experiments and Discussions

The implementation described in the previous section had two main objectives. First, to be able to construct AI characters using an evolutionary approach, where the expertise of the game developer was not needed. The creation of AI that could beat the default AI provided by the *M.U.G.E.N.* engine would be considered a success. The second objective was to develop an AI character that could demonstrate some level of strategy that could be engaging and satisfying for the human player.

TABLE I
GP Parameters Which Were Used for the Initial Set of Experiments

	Parameter value
Number of Matches for ELO Rating	30
Number of Divisions and Size	3 divisions, 33% of the population each
Elitism	5%
New individuals by crossover	90%
New random individuals	5%
Mutation probability	3%

A. Initial Validation of the Technique

As in other machine learning techniques, the quality of the results of the GP approach and the speed with which they are produced may be affected by the values of the input parameters. Given the random nature of the process, it is also possible that a particular set of results could be good or bad by pure chance. The initial choice of parameters is based on previous experience of similar systems and the need to produce results in a reasonable time. A size of 100 characters was established as a starting number that could be further changed if needed depending on the results. To ensure that the results were repeatable, five runs using this size were executed for 30 generations, with a starting mutation rate of 3%. The rest of the parameters that were set for all runs are shown in Table I.

From these initial runs, it was important to make sure that the evolutionary process was actually creating characters with better performance than the ones created randomly at the start of the process. Our ELO system cannot be used to judge the performance of one generation versus another. This is because all the newly generated players are injected into the system with an initial default rating of 1000, thus, pulling the average rating of the whole generation back toward that value.

To ascertain how the performance was changing from one generation to another, another run was made, but this time all new characters of each generation (excluding the ones that survived from previous generations) were matched up against a constant set of 30 elite players obtained from the final generation of the original runs. These matches were completely separate from the matches played by the characters to calculate their ELO as part of the evolutionary process, and did not affect that process. Fig. 6 shows the average percentage win rate of new characters through time when matched against the same group of elite players.

The figure shows how there is an immediate increase of the winning percentage in the first few generations up to generation 8. Then the population average win rate increased more slowly, reaching a maximum average of 48% at generation 30. These results indicated that fitter characters were being created by the process. To allow multiple runs to be made in parallel we used standard university laboratory personal computers,

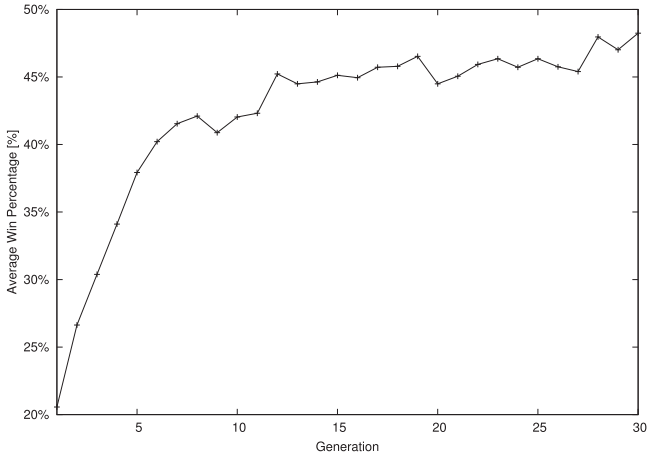


Fig. 6. Average win percentage of new players of each generation matching them against the same "elite" group.

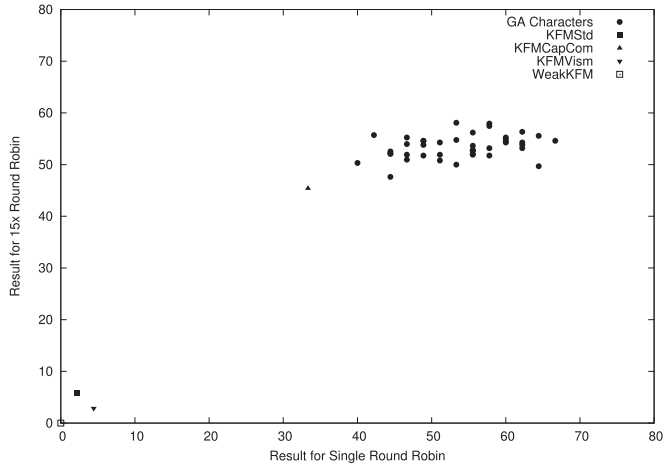


Fig. 7. Average win percentage of GA players and manually generated players in a 15x round robin tournament plotted against their win percentage in a single round robin.

which are available in large numbers. On these machines, runs with 100 individuals take on average 75 min per generation. This means that a 30 generation would take around 37 h to complete and a 100 generation run would take approximately 125 h.

Once the evolutionary process was proved to be working well with our initial set of parameters, further runs were made with different population sizes (100 and 300), number of generations (30 and 100) and mutation rates (0%, 1%, 3%, 6%, 15%, and 20%). Initial testing of the characters generated from populations of 300 did not show any significant improvement. Since these runs were very time consuming we decided not to pursue the larger populations further and hence they are not included in the results presented here. The best performing characters from the final generation of each run then formed the set that were taken forward for further testing against manually created AI characters and human players.

There are several types of result that could be established.

1) How good a player each of these characters was in absolute terms against other AI characters.
2) How effective these characters were in terms of their ability to win against a human opponent.
3) How these characters were perceived by a human opponent in terms of difficulty and satisfaction.
4) The extent to which these results depend on the parameters used for the GA process.

To this end we ran the following tests:

1) Round robin tournaments between our AI characters and the standard AI characters mentioned earlier.
2) Human tests, involving a group of students who expressed an interest in playing fighting games. The total number of students participating was 27. These players were of varying abilities, which may have affected their results and perceptions. However we did not have a large enough number to segregate them on that basis and still retain a statistically significant result.

B. Testing Against Standard AI Characters

We assessed the playing abilities of the various AIs using round robin tournaments. These were all assessed using win percentage rather than ELO. In a full round robin there is no advantage in using ELO and, indeed, it has the disadvantage that the results can be affected by the order of the matches. Some trial tournaments were held to determine the choice of characters to be used for the human testing and to obtain some initial information about the effect of different parameter choices. Subsequently, the characters that had been chosen for human testing were given a more exhaustive test by repeating the complete round robin 15 times and accumulating the results. The set of characters used for this test consisted of 41 GA characters plus the following four manually created characters: Kung Fu Man (KFMStd), which is the default AI provided by the engine. Capcom Kung Fu Man (KFMCap), which possesses all the abilities of KFMStd as well as new ones which are present in a variety of Capcom games such as power charge, fireballs, knee kick super combo. The V-ism variant (KFMVism), which is also based on KFMStd but has one extra ability, the custom combo attack which is performed much faster than in regular gameplay. Finally, the Weak Kung Fu Man (WeakKFM) is a character that performs a very defensive strategy against any player.

The choice of these players was determined by the following considerations:

1) To maintain objectivity, we should use only AIs that had been generated by others.
2) The AIs should not have abilities, such as super fast reactions that can make a player impossible to beat. In other words, they should obey similar constraints to those that we set ourselves when setting out the framework for generating the GA characters. This also has the advantage that such players would not stand out too obviously under human testing.

The GA characters used had a variety of different mutation rates (from 1% to 20%). Some characters were created over 30 generations whilst others used 100 generations. The results of this exercise are shown in Fig. 7.

The results of the multiround robin are plotted on the *y*-axis against the single round robin on the *x*-axis. This test shows

a convergence toward the mean for all of our GA generated characters as they can be seen to be much more tightly grouped in the *y*-axis than the *x*-axis. This indicates that the actual playing strengths of the different GA characters are very similar. To confirm this, we computed the standard deviation of the results for the GA characters only for each individual round robin and also for the average of all the tournaments. The average standard deviation for an individual round robin was 7.00. If there were no intrinsic differences between the characters, one would expect the standard deviation to fall away like $1/\sqrt{N}$. In that case, the standard deviation of the averaged results for all 15 tournaments would be 1.81. The actual result was only slightly higher at 2.15. Consequently, we conclude that the playing strengths of the GA characters cannot have been affected to any great extent by changing the mutation rate since different values of this parameter were represented in the characters tested. For the same reason, it is clear that runs longer than 30 generations are not needed. The manually created characters that we tested were all shown to be weaker in the initial test and this result was confirmed by the exhaustive test as can be seen from the graph. In the 15x round robin, KFMStd scored 6%, WeakKFM scored 0%, and KFMVism scored 3%. Only KFMCap achieved a result near to that of our GA characters (45%). The weakest GA character scored 48% and the strongest 58%. These results raised an issue as to whether the GA was in fact converging on a particular pattern every time, however examination of the code within the GA generated characters showed that they were in fact quite different.

Since the GA characters used different generation counts and mutation rates we concluded that the limiting factor is the choice of triggers and responses that we included, not the GA parameters. The overall structure of the *M.U.G.E.N.* engine may also restrict the possibilities. We checked this by doing runs using zero mutation without adding the new random individuals at each generation. At an earlier stage, we also performed tests on characters from runs with a larger population size (300). None of these changes had a big enough effect on the outcomes to suggest that these characters would not also converge toward the mean in a longer test. From this we concluded that the initial random population contains sufficient variation for the evolutionary process to reach its endpoint without further injection of randomness.

It is not surprising that the GA characters had all achieved roughly the same playing strength as that is the likely result of any evolutionary process. Provided that the mechanism works at all, the details of the process mostly affect the speed with which it happens. Thus, the end result is only dependent on the fitness function.

C. Testing Against Human Players

During human testing the student volunteers initially played some matches against each other to familiarise themselves with the game mechanics without exposure to any particular AI player. They, then played against each of 11 AI opponents (the *M.U.G.E.N.* set up allows 12 characters to be present of which one must be the human player). Of these opponents, one was always the standard AI issued with the game, one or two were chosen from the other manually created AIs, and the remainder were selected from our GA created characters. A total of 41 GA generated characters were used, selected from runs with a variety of parameter values. To produce statistically significant results for individual AIs, some characters were more heavily tested. Extra tests were done later to improve the statistical base further, and in these sessions two existing characters were included along with some new ones. The students were unaware of which characters were which and were free to play their opponents in any order they chose. Each round was a best of three bouts. The students were asked to play two rounds against each opponent, running through the complete set and then repeating. After each round they were asked to record the result and give a score for the perceived difficulty of playing against the opponent and also for the entertainment value or satisfaction of that opponent. These difficulty and satisfaction scores were on a scale of 1(worst)–5(best).

Not all students completed two rounds though others came back more than once and ended up playing up to four rounds against a particular opponent. All the win/loss scores were averaged and converted into a single number between 0 and 1, where a 1 would indicate that the AI won every individual bout. In every case, where the student entered multiple assessments of difficulty or satisfaction for a character, these scores were also averaged before further processing. Thus, at this stage of the process, we had three numbers for each student-AI combination.

All three types of scores were then normalised for each student by subtracting the score achieved by the standard AI. The scores could, therefore, be positive or negative, although in practice most were positive because of the poor relative performance of the standard AI. We did this because different students tested different AIs so otherwise the results could have been distorted by some students being more generous with their ratings than others.

We decided to approach the statistical analysis of the human based results by using randomized simulations.

The most obvious way of generating the simulated data would be to randomise the scores, perceived difficulty ratings, and satisfaction ratings by replacing them with random numbers in the specified range. However, this then leaves the question of what distribution should be used for these numbers. Ideally, the distribution should match that of our actual data. The simplest way to achieve this is to retain the scores and ratings that each student generated, but allocate them randomly to the AIs. This would mean that the average win/loss performance of each student and the "calibration" or emphasis that each gave to difficulty and satisfaction would be retained.

Each simulated experiment was generated by taking the results of the actual experiment and randomly reordering the labels of the AIs. The random reassignment was done separately for each student. The process is illustrated by Table II, which shows how the random simulation worked. (The data shown here is just to illustrate the method, it is not actual data.)

It can be seen from the table that the content of the columns labeled AI (shaded) have been randomized whilst the remaining

TABLE II
RANDOM SIMULATION METHOD

Experimental Data

	Student 1				Student 2				Student 3		
AI	Sc	Df	St	AI	Sc	Df	St	AI	Sc	Df	St
1	0.1	3	2	1	0	2	2	1	0.2	4	2
2	0.2	4	4	2	0.1	2	4	2	0.3	4	2
3	0.15	4	2	3	0.2	5	3	3	0.1	5	5
4	0.4	4	1	4	0.2	5	1	4	0.6	5	2

Randomised Simulation 1

	Student 1				Student 2				Student 3		
AI	Sc	Df	St	AI	Sc	Df	St	AI	Sc	Df	St
2	0.1	3	2	4	0	2	2	4	0.2	4	2
3	0.2	4	4	2	0.1	2	4	3	0.3	4	2
4	0.15	4	2	3	0.2	5	3	1	0.1	5	5
1	0.4	4	1	1	0.2	5	1	2	0.6	5	2

Randomised simulation 2

	Student 1				Student 2				Student 3		
AI	Sc	Df	St	AI	Sc	Df	St	AI	Sc	Df	St
3	0.1	3	2	3	0	2	2	3	0.2	4	2
4	0.2	4	4	1	0.1	2	4	4	0.3	4	2
2	0.15	4	2	2	0.2	5	3	1	0.1	5	5
1	0.4	4	1	4	0.2	5	1	2	0.6	5	2

AI: ID number of the AI that was tested
Sc: Score of the AI against the human opponent
Df: Perceived difficulty
St: Satisfaction

TABLE III
SCORE, DIFFICULTY RATING, AND SATISFACTION RATING FOR THE VARIOUS AIS RELATIVE TO KFMSTD

Result for	No. of results	Score (p value)	Difficulty (p value)	Satisfaction (p value)
GA Av.*	263	0.19 (0.0**)	1.31 (0.0**)	1.40 (0.0**)
KFMCap	16	0.019 (0.42)	0.55 (0.065)	0.75 (0.0047)
KFMVism	18	0.0037 (0.49)	−0.18 (0.32)	−0.11 (0.37)
Weak kfm	10	−0.033 (0.38)	−0.33 (0.25)	0.25 (0.31)
GA 01	14	0.29 (0.00077)	1.09 (0.0051)	1.41 (0.00022)
GA 02	32	0.19 (0.002)	1.39 (0.0**)	1.71 (0.0**)
GA 03	17	0.17 (0.017)	1.01 (0.0017)	1.47 (0.00002)
GA 04	17	0.32 (0.00005)	1.51 (0.00001)	1.53 (0.00001)
GA 07	14	0.21 (0.016)	0.96 (0.0063)	0.66 (0.016)
GA 08	14	0.19 (0.027)	1.41 (0.00007)	1.13 (0.00005)
GA 17	17	0.16 (0.037)	1.63 (0.0**)	1.26 (0.00001)
GA 45	13	0.26 (0.0028)	1.63 (0.00004)	0.96 (0.001)
GA 46	13	0.24 (0.0059)	1.60 (0.00003)	1.19 (0.0001)

* A grand total of 41 GA created characters were tested. The number of tests varied, most being tested 3 or 4 times. The results for characters tested ten times or more are shown explicitly. The average quoted here is taken over all of the characters but is weighted to give equal significance to each test. ** In these cases, the random process did not succeed in finding an event that matched or beat the measured data so p has an upper bound of approximately 0.00001.

columns are unaltered. Note also that there is a separate AI column for each student volunteer. For each result we counted the number of times, within the 100 000 simulations, for which the value was more extreme (i.e., further away from the expected average value) than the actual result. It is this count, expressed as a fraction of 1, that is reported as the p value in the results that follow.

The scores, perceived difficulties and satisfaction ratings for the various AIs are shown in Table III. Note that because KFMStd has been used as a fixed point to normalise all the other AIs, it is not shown in the table. The original raw values for KFMStd were: Win rate 0.028, Difficulty 1.84, and Satisfaction 2.19. As with the AI versus AI test, the GA generated characters comfortably beat the standard AI on all categories and the associated p values are all below the commonly used threshold of 0.05. Details of the parameter setting for the creation of these AI are presented in Table IV.

TABLE IV
MUTATION RATE AND NUMBER OF GENERATIONS USED TO CREATE THE VARIOUS AIS PRESENTED IN TABLE III

Character	Mutation	No. of generations
GA 01	0.03	30
GA 02	0.03	100
GA 03	0.03	100
GA 04	0.03	100
GA 07	0.01	100
GA 08	0.01	30
GA 17	0.06	100
GA 45	0.15	100
GA 46	0.20	100

TABLE V
SCORE, DIFFICULTY RATING, AND SATISFACTION RATING FOR THE VARIOUS AIS RELATIVE TO KFMCAP

Result for	Score (p value)	Difficulty (p value)	Satisfaction (p value)
GA Av - KFMCap	0.17 (0.012)	0.77 (0.0093)	0.65 (0.0072)
GA 02 - KFMCap	0.17 (0.034)	0.84 (0.016)	0.96 (0.003)
GA 04 - KFMCap	0.30 (0.0033)	0.97 (0.015)	0.78 (0.018)
GA 07 - KFMCap	0.19 (0.026)	0.42 (0.14)	−0.089 (0.38)
GA 17 - KFMCap	0.14 (0.082)	1.09 (0.0041)	0.51 (0.063)

Of the manually created AIs, only the Capcom character scored well enough to be a noticeable improvement on KFMStd. Given the p values, it is apparent that even this advantage is only really valid in the satisfaction category. However, since its scores are higher than those for KFMStd we have also calculated relative scores and p values of the best and worst GA characters versus KFMCap.

As can be seen from Table V most of the GA characters are better than KFMCap on most categories. The main exception was GA07, which was inferior on satisfaction, although that result is not statistically significant. A few other results have p values, which are too high for a conclusion to be drawn with confidence. These are GA17 on satisfaction, GA07 on difficulty, and GA17 on score.

Thus the results show that in general terms the GA created characters are superior to the manually coded AIs that were tested.

It is interesting to consider whether score, perceived difficulty, and satisfaction are all effectively measuring the same thing. To determine this, we calculated the correlations between score, perceived difficulty and satisfaction. This was done twice, once for all the characters and then again with only the GA characters included. The results are shown in Table VI.

TABLE VI
CORRELATION BETWEEN SCORE, DIFFICULTY, AND SATISFACTION

Measured correlation	All characters (p value)	GA characters only (p value)
Score versus difficulty	0.75 (0.0*)	0.56 (0.003)
Difficulty versus satisfaction	0.79 (0.0*)	0.52 (0.008)
Score versus satisfaction	0.6 (0.001)	0.28 (0.11)

* In these cases, the random process did not succeed in finding an event that matched or beat the measured data so p has an upper bound of approximately 0.00001.

TABLE VII
DIVERGENCE FROM THE MEAN OF DIFFERENT GA CHARACTERS

Result for	Score (p value)	Difficulty (p value)	Satisfaction (p value)
GA 01- GA Av.	0.1 (0.12)	−0.23 (0.27)	0.0081 (0.5)
GA 02- GA Av.	−0.005 (0.48)	0.09 (0.36)	0.35 (0.06)
GA 03- GA Av.	−0.03 (0.36)	−0.32 (0.14)	0.07 (0.4)
GA 04- GA Av.	0.14 (0.03)	0.22 (0.24)	0.14 (0.32)
GA 07- GA Av.	0.016 (0.43)	−0.37 (0.15)	−0.8 (0.004)
GA 08- GA Av.	−0.008 (0.48)	0.1 (0.4)	−0.29 (0.16)
GA 17- GA Av.	−0.035(0.34)	0.34 (0.15)	−0.15 (0.29)
GA 45- GA Av.	0.07 (0.22)	0.34 (0.19)	−0.46 (0.06)
GA 46- GA Av.	0.05 (0.28)	0.30 (0.22)	−0.22 (0.23)

When all the characters are included the results show a strong correlation between all three quantities. However, when only GA characters are included, the correlations are weaker and the correlation between score and satisfaction is especially weak and consequently not statistically significant. Our interpretation of this is that the students disliked very weak players but when evaluating players of roughly equivalent strength other factors came to the fore. When considering GA players only, it seems that some students equated score with difficulty (correlation 0.56), whilst others associated difficulty with satisfaction (correlation 0.52). However, relatively few can have made both associations because otherwise the correlation between score and satisfaction would have been greater than the recorded value of 0.28.

D. Impact of GA Parameters

The final issue that we considered was the question as to whether the parameters used in the GP algorithm would have an effect on the performance of the characters against human players. The convergence toward the mean observed in Fig. 7 suggests that we may not be able to establish such a connection. However, we have analysed our data to see whether the differences between different GA characters are big enough for these effects to be observable.

We, therefore, calculated the difference between the score of each individual GA character and the mean of all the remaining GA characters. The results are shown in Table VII.

The first thing to note is that in most cases the difference from the mean is clearly not statistically significant. However, there are a few cases that have low enough p values to require discussion. Only one score difference has a low p value, that is 0.03 for GA04. In the difficulty category, all the p values are too high for any statistical significance. In the satisfaction category, GA07 appears to be below average with a low p value. GA45 is also below average, with a p value that is marginal, whilst GA02 is above average with a similar p value. These values do seem to be significant, however, there are not enough of them to support a further analysis of the effect of the GA parameters. In addition, we must also be cautious because these instances have been located by searching through a group of ten candidates and hence one would expect at least one p value of around 0.1 to arise from pure chance every time. To make sure of this result, we did calculate the correlations between the GA parameters and the human test results. None of the correlations had a magnitude greater than 0.1 and all of the p values were clearly above the generally accepted threshold of 0.05 for a result to be considered statistically significant.

VI. CONCLUSION

We have shown that GP can be used to create AI characters for the *M.U.G.E.N* fighting game. Minimal human intervention is required in the process and unlike other techniques the method does not require any modification of the game engine itself.

Testing the resulting characters in terms of performance against publicly available hand coded AI characters showed that the characters produced by the evolutionary process were significantly better than hand coded ones. KFMCap was the only one that achieved a result near to that of our GA characters.

Further testing with human players showed that, although the GP characters were only able to win a few matches, they still outperformed the hand coded AI in terms of wins and were also rated more highly for perceived difficulty and user engagement. The correlation observed between satisfaction and perceived difficulty scores among all AI characters suggest that, in general, human players tend to dislike very weak AI characters. However, when they are presented with characters that are roughly equivalent in strength, their perception varies. This suggests that there are other factors that may determine the perception of the human player.

All of these results were seen to be insensitive to the population size and mutation parameters of the algorithm. The analysis of the results suggest that the performance of the characters is determined by the choice of triggers and responses used in the creation process and that any additional randomness in the evolutionary process (e.g., new random characters, mutation operator) does not affect the effectiveness of the algorithm.

The practical implications of applying a GP approach to generate AI are that substantial savings could be made in AI generation by utilizing GP to "grow" AI characters for games rather than directly implement them by hand. Another advantage is that the method automatically produces a large set of different characters.

Future work will be focused on carrying out more extensive experimentation with human players and investigating if new adaptations to the GP approach could improve the present results by, for example, changing the natural selection rules or the tournament design. Increasing the range of triggers will also be

studied to assess its impact on how quickly a more complicated logic can emerge.

ACKNOWLEDGMENT

The authors would like to thank A. Slack, the President of the Nottingham Trent University Developers Society, the computing students from Nottingham Trent University for their participation and feedback during the AI testing, and to Dr. C. Langensiepen and W. Xu, for reviewing the manuscript and providing some helpful suggestions.

REFERENCES

[1] J. E. Laird, "Using a computer game to develop advanced ai," *Computer*, vol. 34, no. 7, pp. 70–75, Jul. 2001. [Online]. Available: http://dx.doi.org/10.1109/2.933506

[2] M. Buro and T. M. Furtak, "Rts games and real-time ai research," in *Proc. Behavior Representation Model. Simul. Conf.*, 2004, pp. 51–58.

[3] G. N. Yannakakis and J. Hallam, "Towards optimizing entertainment in computer games," *Appl. Artif. Intell.*, vol. 21, no. 10, pp. 933–971, Nov. 2007. [Online]. Available: http://dx.doi.org/10.1080/08839510701527580

[4] S. Bakkes, C. T. Tan, and Y. Pisan, "Personalised gaming: A motivation and overview of literature," in *Proc. 8th Australasian Conf. Interactive Entertainment: Playing Syst.*, 2012, pp. 4:1–4:10. [Online]. Available: http://doi.acm.org/10.1145/2336727.2336731

[5] C. Fairclough, M. Fagan, B. M. Namee, and P. Cunningham, "Research directions for ai in computer games," in *Proc. 12th Irish Conf. Artif. Intell. Cogn. Sci.*, 2001, pp. 333–344.

[6] B. S. E. Ortiz, K. Moriyama, K.-i. Fukui, S. Kurihara, and M. Numao, "Three-subagent adapting architecture for fighting videogames," in *Proc. 11th Pacific Rim Int. Conf. Trends Artif. Intell.*, 2010, pp. 649–654. [Online]. Available: http://dl.acm.org/citation.cfm?id=1884293.1884361

[7] B. H. Cho, S. H. Jung, Y. R. Seong, and H. R. Oh, "Exploiting intelligence in fighting action games using neural networks," *IEICE - Trans. Inf. Syst.*, vol. E89-D, no. 3, pp. 1249–1256, Mar. 2006. [Online]. Available: http://dx.doi.org/10.1093/ietisy/e89-d.3.1249

[8] T. Graepel, R. Herbrich, and J. Gold, "Learning to fight," in *Proc. Int. Conf. Comput. Games: Artif. Intell. Design Edu.*, 2004, pp. 193–200.

[9] D. Kehoe, "Designing artificial intelligence for games," Jun. 2009. [Online]. Available: https://software.intel.com/en-us/articles/designing-artificial-intelligence-for-games-part-1.

[10] J. D. Bonet and C. Stauffer, "Learning to play pac-man using incremental reinforcement learning," in *Proc. Congr. Evol. Comput.*, 1999.

[11] S. Samothrakis, D. Robles, and S. Lucas, "Fast approximate max-n monte carlo tree search for ms pac-man," *IEEE Trans. Comput. Intell, AI Games*, vol. 3, no. 2, pp. 142–154, Jun. 2011.

[12] G. Danzi de Andrade, H. Pimentel Santana, A. W. Brotto Furtado, A. R. Gouveia Do Amaral Leitao, and G. Lisboa Ramalho, "Online adaptation of computer game agents: A reinforcement learning approach," in *Proc. II Workshop de Jogos e Entretenimento Digit.*, 2003, pp. 105–112.

[13] A. Ricciardi and P. Thill, "Adaptive ai for fighting games," 2008. [Online]. Available: http://cs229.stanford.edu/proj2008/RicciardiThill-AdaptiveAIForFighting Games.pdf.

[14] F. Lu, K. Yamamoto, L. H. Nomura, S. Mizuno, Y. Lee, and R. Thawonmas, "Fighting game artificial intelligence competition platform," in *Proc. 2013 IEEE 2nd Global Conf. Consum. Electron.*, 2013, pp. 320–323.

[15] K. Yamamoto, S. Mizuno, C. Y. Chu, and R. Thawonmas, "Deduction of fighting-game countermeasures using the k-nearest neighbor algorithm and a game simulator," in *Proc. 2014 IEEE Conf. Comput. Intell. Games*, 2014, pp. 1–5.

[16] W. Thunputtarakul and V. Kotrajaras, "Data analysis for ghost ai creation in commercial fighting games." in *Proc. GAMEON*, 2007, pp. 37–41.

[17] S. S. Saini, "Mimicking human player strategies in fighting games using game artificial intelligence techniques," Ph.D. dissertation, Loughborough University, Loughborough, U.K., 2014.

[18] Y. Miche, M.-H. Lim, A. Lendasse, and Y.-S. Ong, "Meme representations for game agents," *World Wide Web*, vol. 18, no. 2, pp. 215–234, 2015. [Online]. Available: http://dx.doi.org/10.1007/s11280-013-0219-3

[19] Y. Hou, L. Feng, and Y. S. Ong, "Creating human-like non-player game characters using a memetic multi-agent system," in *Proc. IEEE WCCI-Int. Joint Conf. Neural Netw.*, 2016, pp. 177–184.

[20] C. S. Ho, Y.-S. Ong, X. Chen, and A.-H. Tan, "Fame, soft flock formation control for collective behavior studies and rapid games development," in *Proc. 9th Int. Conf. Simulated Evol. Learn.*, 2012, pp. 258–269.

[21] J. Byrne, M. O'Neil, and A. Brabazon, "Optimising offensive moves in toribash using a genetic algorithm," in *Proc. 16th Int. Conf. Soft Comput. Mendel 2010*, pp. 78–85, 2010.

[22] B. Cho, C. Park, and K. Yang, "Comparison of ai techniques for fighting action games - Genetic algorithms/neural networks/evolutionary neural networks," in *Proc. 6th Int. Conf. Entertainment Comput.*, pp. 55–65, 2007.

[23] S. Luke *et al.*, "Genetic programming produced competitive soccer softbot teams for robocup97," *Genetic Program.*, vol. 1998, pp. 214–222, 1998.

[24] S. M. Gustafson and W. H. Hsu, *Layered Learning in Genetic Programming for a Cooperative Robot Soccer Problem*. New York, NY, USA: Springer, 2001.

[25] Y. Azaria and M. Sipper, "Gp-gammon: Genetically programming backgammon players," *Genetic Program. Evolvable Mach.*, vol. 6, no. 3, pp. 283–300, 2005.

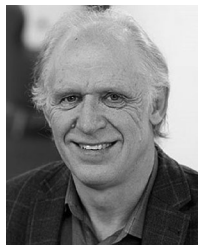
[26] A. Hauptman and M. Sipper, *GP-Endchess: Using Genetic Programming to Evolve Chess Endgame Players*. New York, NY, USA: Springer, 2005.

[27] J. R. Koza, *Genetic Programming: On the Programming of Computers by Means of Natural Selection*, vol. 1. Cambridge, MA, USA: MIT press, 1992.

[28] Elecbyte, "Mugen wiki: Category: Triggers," 2010, [Online]. Available: http://elecbyte.com/wiki/index.php/Category:Triggers

Giovanna Martínez-Arellano received the M.Sc. degree in computer science from the Centro de Investigación Científica y Educación Superior de Ensenada (CICESE) in Ensenada, B.C., México, in 2007 and the Ph.D. degree in computer science from Nottingham Trent University, Nottingham, U.K., in 2015.

She is currently a Lecturer in the School of Science and Technology, Nottingham Trent University. Her current research interests include machine learning, forecasting, modeling, and data mining.



Richard Cant received the First Class Honors degree in physics from the University of Manchester, Manchester, U.K., in 1975. He received the Ph.D. from the University of London (Imperial College) in 1980.

After completing Part III of the Mathematics Tripos at the University of Cambridge in 1976, he undertook research in Theoretical Physics at Imperial College, London, U.K. He continued research in theoretical physics as a Research Assistant at the University of Manchester until 1982. For the following 9 years he worked as a System Designer for Ferranti Computer Systems in Stockport before returning to academic life as a Senior Lecturer at the then Nottingham Polytechnic in 1991. His current research interests centre around computer generated imagery and artificial intelligence. A particular interest is the application of AI to games including the automatic synthesis of virtual environments and the creation of plausible AI characters.

David Woods received the Graduate degree with first class Honors in computer science from Nottingham Trent University, Nottingham, U.K., in 2014.

Mr. Woods received the prize for the best project.