Vidya: A God Game Based on Intelligent Agents Whose Actions are Devised Through Evolutionary Computation

Marcelo R. de Souza Pita, Salomão Sampaio Madeiro, and Fernando Buarque de Lima Neto
Department of Computing Systems - Polytechnic School of Engineering
Pernambuco State University, Recife – Brazil
{mrsp, ssm, fbln}@dsc.upe.br

Abstract—Vidya **is a strategy computer game, god-style, that can be seen as a rich environment where virtual beings compete among themselves for natural resources and strive within the artificial ecosystem. Although in this game the player cannot directly control the intelligent agents, he can give some intuitions to them. Together with these intuitions the agents, called** *Jivas* **– the most developed species of the ecosystem, devise actions through evolutionary computation. The game allows also the observation of all interactions among the various beings inhabiting** *Vidya***. Interactions happen in a quasi-autonomous manner which grants the game with an interesting dynamics. The evolved** *Jiva***'s intelligence, which build-up during the game, can be reused in other game scenarios. This work might help on further understanding of some emergent autonomous behaviors and parameterization of intelligent agents that live in closely coupled ecosystems.**

Keywords: Computational Intelligence, God Game, Vidya, Evolutionary Computation, Autonomous Behavior.

I. INTRODUCTION

VIDYA is a strategy computer game, posed as an environment where virtual beings compete for available natural resources of an ecosystem. Among many types of beings that inhabit the virtual world, there is a special one called *Jiva.* They are intelligent agents designed to be autonomous and survive against all the odds. In the *Vidya* game, the player is invited to be the *Jivas*' clan *Deva* (*i.e.* their god). That is, the one who provides guidance to selected clan members. These instructions are not provided not in fine details and appear in the *Jiva*'s mind as intuitions. They are referred in the game as *vidyas*, which are useful for helping the *Jivas* to take their own decisions. Due to their "autonomy", *Jivas* are capable of deciding which actions to perform for various situations in the world regardless of receiving any *vidya* provided by player.

Artificial Intelligence (AI) techniques [1] are heavily used in computer games [2]. They are today one of the main factors of success or failure of any non-casual game. Two main factors contribute to this in electronic games. First, following the appearance of graphical accelerator boards,

much of central processing potential became available for AI computations. Second, newer games are continuously required to present greater degrees of realism [3], [4]. The realism referred here applies not only in the graphical design and physical modeling of the game, but also in the evoked behavior of the agent

This means that a good non-casual game that explores computational intelligence not only has to be playful, but has to be aesthetically well cared-off as well; desirably an even balance should be obtained between these two ingredients.

In the *Vidya* game the algorithms are written in a way that intelligent behavior is at the center of the game's processing load, especially when controlling the behavior of the *Jivas*. There are others agents in the game, namely, animals like sheep, wolfs, cows, that express non-monotonic behavior. These agents' behaviors are solely reactive processes. In other words, for these lower beings of *Vidya*, behaviors are state-to-actions static mappings, without evolution. As opposed to that, *Jivas*' intelligence evolves thru time and increases their knowledge about the world. They autonomously learn by their own experience as well as godly received inspiration (*i.e.* the *vidyas* provided by the player). This happens because all *Jivas* learn by self-analyzing their own behavior, evaluating whether a given action at a specific time was appropriate. In the long run, this might help gathering new insights for Evolutionary Computation [5].

Apart from the ludic side of *Vidya*, the result of the intelligence modeling of *Jiva* is the creation of evolutionary intelligent agents that act and learn in a quasi-autonomous way. Because of *Vidya* modular construction of interacting characters, in the future it could be adapted to be a simulation environment (*i.e.* a test-bed) to help on the understanding of other social beings' behaviors.

II. INTELLIGENT TECHNIQUES UTILIZED

Genetic Algorithm (GA) is an intelligent technique widely used for performing complex search and optimization problems based on principles of natural evolution [6], [7]. Population, chromosomes, genes, fitness, reproduction, offspring, and mutation are important concepts brought from Genetics and Nature straight into this technique.

The Population is a set of individuals (*i.e.* chromosomes),

here possible solutions for the Jiva's decisions. The fitness function calculates a value used to classify how good an individual is according to some given criterion. Selection, reproduction and mutation are GA operators applied directly on the population to generate a new one. This results in an effective search system in the input space (i.e. *Jiva's* world).

In GA possible solutions for the problem at hand are individual chromosomes, which are composed of relevant characteristics that are coded as their genes. An initial population of individuals, *i.e.* set of chromosomes, is randomly generated. After being classified by some fitness criterion, they will evolve and guide de search process to an optimal solution for the problem. After fitness value calculation, pairs of individuals, selected due to methods such as the Roulette Wheel [8] or the Tournament [8], have their genes crossed-over. Following that their chromosomes are combined to generate a new individual. At this time, it's possible to introduce a random operator, *i.e.* mutation. This avoids premature convergence by modifications of one or more gene values of that new individual. Crossover and mutation will happen until a new population is generated. Figure 1 illustrates all mentioned GA steps.
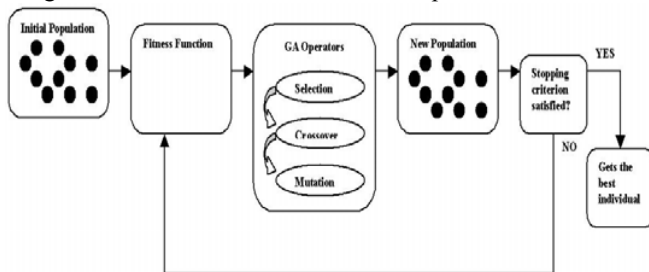

Fig. 1. Overview of all Genetic Algorithm steps.

Finding a good solution to a problem can be a very time consuming operation, especially whether one tries to explore the entire solutions surface. GA is capable of finding an answer as good as needed by executing sufficiently the above mentioned cycle. Note that for every cycle, there is one best individual in the current population. Thus, as soon as there is a need for the algorithm to stop (*e.g.* the user demands an action to be taken in the game), a non-random candidate solution – suitable for the problem – is readily offered by the GA. It is important to be noticed that in general GA does not take many cycles to find a reasonable solution.

III. *VIDYA*

A. The Vidya Game

Vidya is a single-player strategy-god game [9] game that includes AI and a new player-character interaction model. *Vidya* is also a test-bed for investigations that aims at advancing development of autonomous intelligent agent through Evolutionary Computing.

The intelligent decision module of *Vidya* agents was planned to be used in other games, mainly in characters that

need more autonomy, agents of ecosystem simulations, or even in studies of emergent social behaviors.

The *Vidya* game environment is shared by many types of beings (as shown Figure 2) that compete for available natural resources, sometimes not abundant. The main characters in this environment are the *Jivas*, they are autonomous intelligent agents that are endowed with evolutionary properties. Clan is the social unit of *Jivas*.

The player performs the role of a clan's *Deva* and may instruct the *Jivas* of one clan, by helping them to survive in the game world. This is not an easy task, especially because *Jivas* are autonomous beings and can choose to follow the players instruction – the *vidyas*, or simply choose to ignore the, so-perceived, "divine" intuition.
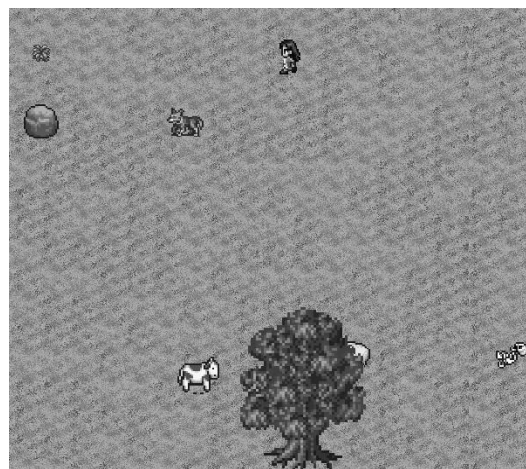

Fig. 2. Screen-shot of the game world, showing the *Jiva* (at upper-center) and other beings: grass, rock, fox, cow, tree and a dead-fox.

B. The Vidya Software

Vidya was completely developed using Java language [10]. Java is easy for programming, has plenty of *Application Programming Interfaces* (APIs) already implemented for games and is Object-Oriented (OO), which makes modeling much easier. The main concern regarding Java is its notorious not-so-good performance when compared to other programming languages, like C and C++. Figure 3 shows the high level architecture of the game.
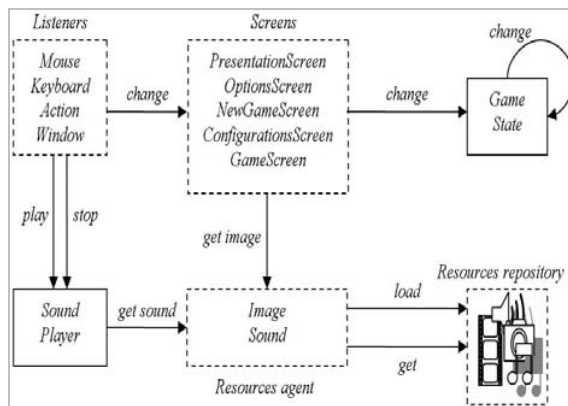

Fig. 3. High-level software architecture of the Vidya game.

The *Listeners* module tackles all player input, generating events that, sometimes exchange screens, change the game state (delegated by *GameScreen* screen). Events are still generated to control the *Sound Player*.

The *Resources repository* is a set of image and sound files that are accessed through *Resources agent*. The *Screens* and *Sound Player* modules access (via *Resources agent*) image and sound resources, respectively.

Inside the *Game State* module we programmed the game motor, the world and its objects, on which we find the *Jiva*, with its intelligent decision module.

C. The Jiva's Intelligent Decision Module

Jiva (and other beings inhabiting the game world) follows the agent model. Specifically, *Jiva* is an autonomous intelligent agent, because it is capable of taking intelligent decisions in an independent way.

Like all agents, *Jiva* perceives the environment, performs an inner processing in order to select a good action and act on the environment. The *Jiva*'s perception is partial, that is, it does not perceive the whole environment, but only a variable (smaller) part of it.

It is easy deduce that the basic problem of the *Jiva* is to perform well on the environment for a given limited perception. What they do is to minimize a cost function (or maximize a gain function). *Jiva*'s decisions for actions might be seen as control operations that intend to benefit from the environment in order to create ideal conditions for its own survival.

The task of the *Jiva*'s intelligent decision module is to be responsible for selecting (or deciding) the best action to be performed by the *Jiva*'s agent. This action selection is then realized using GA. Actions are single-mapped to individuals of the population, which is a subset of the available actions.

Jiva has a very large set of available actions that can be performed. The selection of the best one can be very time consuming if traditional techniques are used. With GA we narrow down the search space and the choices converge towards the best action promptly. This process produces a good candidate solution even at few initial (GA) generations.

Jiva perceives the world as a set of objects that are inside its perception square. This perception limit depends directly on *Jiva*'s vitality. In the beginning of the game, the *Jiva* has the maximum vitality, hence perception level. As time goes by it start losing its perception accordingly to the decay of vitality. *Jiva*'s vitality can be interpreted as the remaining life time of the *Jiva*. In all this could be interpreted as aging.

The perception square of the *Jiva* defines a set of cells (grid) that *Jivas* can perceive. These cells also define the complete possible future destinations for the *Jiva* (*i.e.* all possible positions that the *Jiva* can occupy at the next time step). Figure 4 shows the perception square of the *Jiva*, the perceived objects and the cells with in the perception square.

The *Jiva*'s perception is composed by a set of world objects. Each object in the perception is characterized by its type and positioning in relation to the *Jiva*'s *own* position.
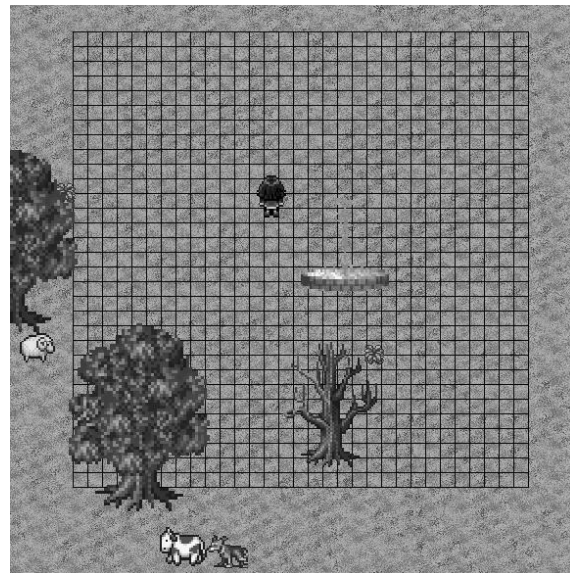

Fig. 4. Screen-shot of the game world, showing the perception square of a *Jiva*. The objects inside the square are being perceived (a water font, a small plant, a dead tree and a tree). The small squares that compose the perception are the perception cells. Outside of the perception square, in the bottom of image, a wolf (predator) is hunting a cow (prey).

In the present work, the *Jiva* is able to perceive occluded objects, that is, it can see all objects that are inside its perception square, even whether some objects are behind other objects. Uncertainties caused by occlusion of objects is an interesting topic [11], however, we left it for future work.

Jiva's agent, after it has perceived the environment and created internal representation for every object within its perception square, will select a good action for that perception. An action is defined as the destination cell to which the *Jiva* will move to. So, we characterize an action by a coordinate pair (x, y), meaning that the *Jiva* will move to that position in the next time step, as shown in Figure 5.

Fig. 5. Destination cell to where the *Jiva* possibly will move. Observe that there is a cell selected with a fruit on it; the *Jiva* may want to eat it.

The semantic value associated to an action depends on the object that is occupying the destination cell. For example, if a *Jiva* migrate to a cell where there is a fruit on it (Figure 5), the semantic value associated to this action is "eating the fruit". Alternatively, if in the destination cell there is a wolf, the semantic effect is "hunting the wolf". If the destination cell is empty (no object on it), the semantic value of the action is just "keep walking to another destination". For each object type in the destination cell there is a different semantic value associated. For some objects, the semantic value associated may be the same (*e.g.* sheep and cow).

The semantic implications of actions by the *Jiva* are not only dependent on the objects that are occupying the destination cells but are also dependent on its vital function to each being inhabiting *Vidya*. *E.g.*, a *Jiva* eating a fruit will increase a bit its vitality and energy, while eating a dead wolf will increase its energy and decreases its vitality.

D. Solving the Problem through Evolutionary Computing

Given the time constraints of the game, a unique GA instance might not solve the problem of selecting the best action for all possible situations. Notice that a good action for a given perception may not (probably will not) be good to another perception. Each different perception is a different problem, and then different GA instances have to be created for each one of these different perceptions. The *Jiva*'s intelligent decision module has a structure that maps perceptions on instances of GAs (on a one-to-one mapping).

Each GA instance has the role of selecting the best action for a given perception, following the natural selection method that characterizes a GA. This includes the evaluation, crossing and mutation of individuals.

When evaluating an action, the GA might care about not only its immediate cost, but infer on the long term cost for it. We will refer to this long term evaluation as an evaluation "F steps in the future", where F is the number of steps that the GA will see in the future to estimate a long term cost.

The long term cost can be inferred because the *Jiva*'s intelligent decision module can simulate the progress of an action in the future. Of course, the estimated long term cost have to be confronted with *Jiva*'s experience, that is, the real cost obtained by interacting with the environment.

The path of a *Jiva* in the game world is internally represented as a states machine (Figure 6), where each state is a different perception configuration of the *Jiva*'s agent.

Associated to each state (s_i) there is an intelligent processing element, that is an instance of a GA (GA$_i$). The task of this element is selecting the best action (a_{ij}) for the state. Due a performed action, the *Jiva* migrate to another position in the game world and has a new perception, for which there is another GA instance to solve it.

When the *Jiva* reaches a perception state, it has a set of positions to move to (the set of possible actions). Calculating the cost to go for each destination cell is an expensive task, because the number of possible destination cells is large. For example, if we have a perception square of 20×20 cells and we would like to know its cost 3 steps in the future ($F = 3$), 400 cells would be evaluated, having for each cell a triple analysis (the 3 steps in the future). After this task we would still have to perform a comparative analysis between these 400 cells to choose one.
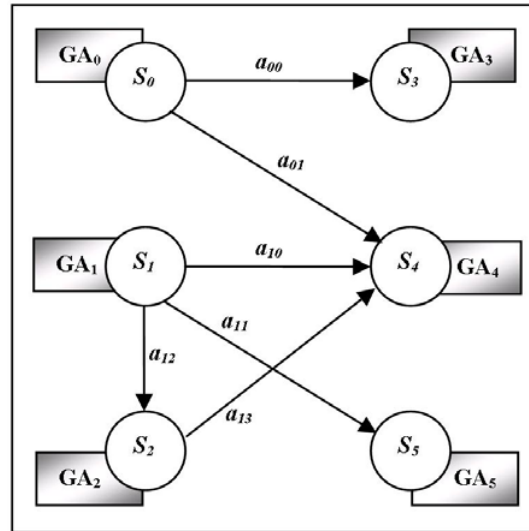

Fig. 6. State machine that represents the path (series of decisions) of a *Jiva* in the game world throughout time. Each state is a different perception configuration at a given time, for which there is one GA instance that will select the best action for that state at that particular time.

Using Genetic Algorithms, we have a convergent method (converging towards the best solution) with a reasonably smaller search space. Supposing a GA with a population size of 40 individuals, for the problem described in the last paragraph, we have a mean reduction of 90% in the search. Figure 7 shows the initial spatial distribution of the population for a given perception of the *Jiva*.

Located in a given state, the *Jiva* invokes the associated GA instance to suggest an action. Each time this state is reached, the GA instance advances one generation. In the first generation we already have a converging solution. This advance is governed by the well known GA sequence of tasks: evaluating, crossing-over and mutating individuals.

Before all, evaluation of the population is done for selecting the best action that will be performed by the *Jiva*. The evaluation phase only is completed when the action is performed and the *Jiva* has a real immediate cost obtained by experience. So, the evaluation phase is divided in two parts: before the selection and after the action.

Before the selection, each action is internally simulated F steps in the future. The action that has the best qualification in the simulation is then selected. Figure 8 shows the simulation algorithm for one action. After obtaining the qualification (*i.e.* inferred cost) for each action, the action that has the greater qualification will be selected and performed by the *Jiva*.
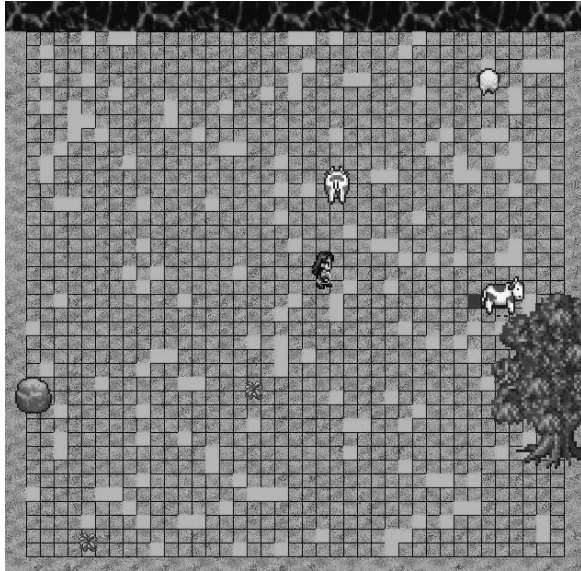
Fig. 7. Initial spatial distribution of the population for a given perception. We can see all cells that are inside the 41×41 perception area, the population distribution (168 individuals, that is, 10% of the total search space, in light gray cells) and the destination cell (dark gray cell), selected through an analytic simulation considering 5 steps in the future. The decision interpretation is: the *Jiva* is going to hunt the cow.

```
parameters: currentAction, currentState, F
return:     qualification


var qualification := 0


iterate 1 to F:
    /* Qualification update.                 */
    /* costToGo returns the cost to realize  */
    /* the current action in the current state. */
    qualification := qualification +
             costToGo(currentState, currentAction)

    /* New state reached */
    var newState := updateState(currentState,
                                 currentAction)
    /* GA instance for the new state reached */
    var gaInstance := getGAFor(newState)
    /* Best action for the GA instance */
    var newAction := getBestActionFor(gaInstance)

    /* Makes current action the new action */
    currentAction := newAction
    /* Makes current state the new state */
    currentState := newState


/* Returns accumulated qualification */
return qualification
```

Fig. 8. Algorithm for the simulation of an action *F* steps in the future.

For the time being, *Jiva*'s experience is out of scene.

Now, the winner action, after its realization, will be evaluated again. The immediate cost obtained by the *Jiva* in realizing the recent action will be summed to the qualification of the action, re-qualifying it. After this post-evaluation, the winner action may keep on being the best one, or may lose the first position to another action best qualified. So, this post-evaluation tries to correct inference errors in the simulation, re-ordering, in terms of qualification, the individuals of the population. These individuals are then ready to crossover and mutation phase.

An action for the *Jiva* is a move to a destination cell. So, actions are represented by a coordinate pair (x, y), indicating this destination.

The selection method used to choose pair of actions that will cross is the Roulette Wheel selection method [8]. Pairs of actions are selected and crossed, generating new actions that will compose the next generation of the population.

The mutation operator is also applied to the population. The mutation probability was arbitrarily set to 1/32, that is, in average, for each 32 actions 1 is mutated.

The use of GA in the *Jiva*'s intelligent decision module aims to reduce considerably the search space for selecting actions in a coherent manner using natural selection principles and generating good solution already in the initial generations.

Each time a state is reached, a new generation of better new actions replaces the old ones in the population. The old actions are evaluated considering their probable long term qualification *F* steps in the future, where *F* is a parameter of the game. This is achieved through progress simulations, where the selected action (that has the best qualification) is re-qualified by its immediate cost after realization. So, these old actions are submitted to crossover and mutation operations, resulting in a new generation of actions better qualified.

In the game industry there are strong indications that computer games will became more and more complex [12]; this can be seen in the last generation of computer games. This growing complexity has turned even hard the task of explicitly programming agent behaviors (in the cases of purely reactive agents). Even when AI is evolved, it is hard to obtaining a representative knowledge database for a specific game world, this because the number of different possible situations is very large due to action produced by players that are, at least in first analysis, non-deterministic. The use of adaptive agents capable to learn in new situations is a good way to overcome this difficulty.

The *Jiva*'s agent and its intelligent decision module is an example of efficient use of Evolutionary Computing in computer games, specifically in the behaviors of their autonomous agents. The produced intelligent decision module can be hypothetically reused in other computer games and in autonomous agents that need to be adaptive, that is, in various game worlds where learn abilities in new situations will be required.

E. A New Player-Character Interaction Model

As mentioned before, the main character in the *Vidya* game is the *Jiva* – the focal point between player and game.

Traditionally, interaction between player and characters in a computer game happens in a very direct way. Through controls, the player can operate almost completely the behavior of the characters (also referenced as *avatars*).

In the *Vidya* game, the player is a clan's *Deva*, whose role is to be the mentor of the clan and to promote survival in the game world. This is a hard task, because *Jivas* are intelligent agents, and can choose actions to perform, despite suggestions of the *Deva* (*i.e.* the player).

The not-detailed instructions provided by the player given to the *Jivas*' clan (individually for each character) are known in the *Vidya* game as *vidyas*. A *vidya* appears inside of each *Jiva* as one "intuition". The goal of the player should be to provide good *vidyas* to the *Jivas* that might help them to outlive others, Figures 9, 10 and 11 are examples of that.

Fig. 9. Player selecting a clan's *Jiva*. By selecting a *Jiva* of his clan, the player can perform the following tasks: (i) provide *vidya* for this *Jiva*; (ii) show/hide perception area of this *Jiva*; (iii) show/hide the destination cell.

Fig. 10. Player providing *vidya* for a *Jiva*. In this situation, the player has to select a cell inside the perception area that will be considered in the *Jiva*'s decision own decision.

Fig. 11. Possible targets considered by the *Jiva* in its decisions.

When the player intercepts the *Jiva* to provide *vidya*, the game pauses and the player can suggest a destination cell that the *Jiva* may follow. Among others possible destination cells, which are the individuals of the population of the GA instance for that particular perception, the provided *vidya* will selectively compete for the best qualification. The most qualified ones will pass their characteristics for the next generation. If the action suggested by the player is a good action, it will help the long term learning of the *Jiva* – this is the main goal of *Vidya*.

IV. EXPERIMENTS AND RESULTS

The experiments included in this section were used to evaluate the intelligent evolutionary algorithm proposed in the last section. Basically, this algorithm aims at implementing the *Jiva*'s intelligent behavior, which is based on Genetic Algorithms. In the proposed algorithm all *Jiva*'s actions are mapped on individuals of a given population. The feature of each individual is obtained through simulation of future action. Then, the so-called best actions will pass on its characteristics to the next generation.

The focus of all experiments is at computational performance evaluation of the algorithm as an indirect method for evaluating intelligence. The idea is to assess intelligence through analysis of the adaptability of the *Jiva*.

We can identify two main parameters within the algorithm: (i) the GA's population size – *PS* and (ii) the number of future steps that the simulations can see – *F*. We varied these two parameters to observe the algorithm behavior according to possible actions.

The set of individuals, for which the GA's population is a subset, has a size equals to the *Jiva*'s perceptive area. We assumed a constant perception area of 961 cells (31×31 square) in these experiments, so that they can be compared among themselves. The *PS* values were assumed to be 10%, 20% and 40% of all possible actions, that is, $PS = 96$, $PS = 192$ and $PS = 384$, respectively. The *F* values were assumed to be $F = 2$, $F = 4$ and $F = 6$.

Absolute values expressed in the results are not so relevant here. Actually, the experiments propose a comparative analysis of the same algorithm with different values assumed by parameters *PS* and *F*. They show the best parameter configuration, which maximizes the *Jiva*'s performance in both aspects: computation time and intelligent levels.

The hardware used was a entry-level personal computer AMD Duron™ 1.6 GHz, with mere 352MB of RAM.

A. Performance of the Algorithm

The graphic of Figure 12 shows the performance, in milliseconds, of the proposed algorithm for the selected parameters; that is, variations for *F* (steps into the future) and *PS* (population size of the GA).

The three lines shown in Figure 12 are trends evoked by the algorithm related to its computational performance. For every trend, processing time augment as function of *F*. The observed growth is practically linear. Each point in the graphic was obtained by an arithmetic average over 100 executions.
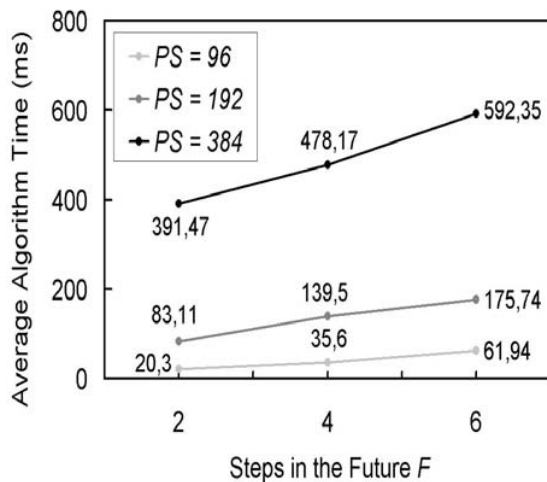

Fig. 12. Evoked performance of the proposed algorithm subject to selected parameters variations (*F* and *PS*).

In order to obtain this data we have to add extra routines to the main code. However, the observed results only refer to the examined code.

This algorithm behavior appears to be predictable with respect of processing load. The differences in the measured values across populations of different sizes, for the same values of *F*, seem to be predictable too and present an almost linear growth trend.

The absolute values obtained in the experiments also reveal that the overall computational performance of the algorithm is not large. For the tested cases, it did not exceed 600ms, which is perfectly acceptable in real game problems that require short response time.

B. Jiva's Intelligence Measurement

Measuring intelligence is difficult. To do it, we have used an indirect method that, in our case (the Jiva's intelligence

decision module) was to evaluate the gain (that can be positive or negative) obtained by the *Jiva*, during a period of time interacting with the environment. That is, we are using the *Jiva*'s adaptation capability in the *Vidya* game to measure the algorithm performance in providing intelligent behavior.

The most relevant *Jiva*'s features, the ones that interested most for evaluation of the *Jiva*'s adaptability, were: 'vitality', 'hydration' and 'energy'. A weighed average of these values, which can be obtained instantaneously from every *Jiva*, produces what we refer as 'general vital condition' (GVC); this value changes thru time. The weight for the general vital condition, assumed for vitality, hydration and energy were 50%, 30% and 20%, respectively.

If we measure the gain obtained in the general vital condition by the *Jiva* after a period of time, we can state that this value represents approximately the *Jiva*'s increased capability for surviving. Moreover, this also means how well it has performed to obtain additional resources for its own survival. Note that in our experiments we do not evaluate the social behavior of the *Jivas*, which probably is qualitatively superior, but only the *Jiva*'s individual behavior (*i.e.* relating to its individual survival). The analysis of emergent social behavior of the *Jivas* is left as a future work.

The graphic of Figure 13 shows the gain in general vital condition obtained by a *Jiva* for the selected parameters variations (*F* and *PS*), in a period of 10 min. Each value in the graphic is the arithmetic average over three executions. A death in any test represented a loss of -100 (negative gain), because 100 is the *Jiva*'s maximum initial general vital condition. *E.g.*, for *PS* = 96 and *F* = 6, the gain of -100 indicates that the *Jiva* has died in the three tests.
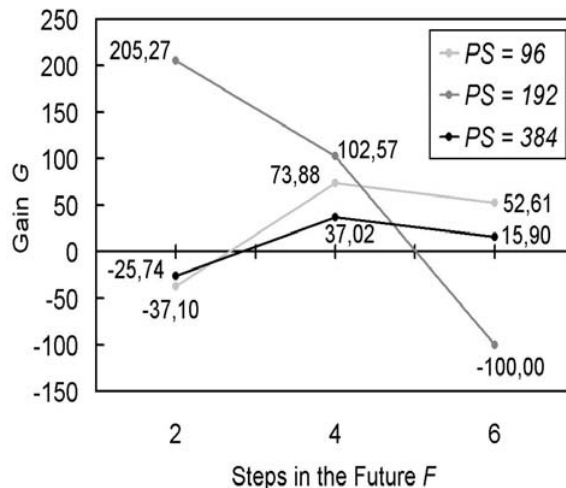

Fig. 13. Gain in the general vital condition of the *Jiva* for selected parameters variations (*F* and *PS*).

Theoretically, the larger is *PS* and *F*, the larger should be the gain in general vital condition of the *Jiva*. But, this expectation was disproved by our experiment.

When analyzing the average gains for variations in *PS,* say for *PS* = 96, we observed an average gain of 29.8 points in the general vital condition. For *PS* = 192, we have the greater standard deviation, with the best and the worst values, resulting in an average gain of 69.28 points for the GVC. For *PS* = 384, we have an average gain of 32.15 points for the *Jiva* GVC. Then, the best value for *PS* is 192.

When analyzing the average gain for variations in *F,* say for *F* = 2 we observed an average gain of 47.48 points; for *F* = 4, we have an average gain of 71.16 points; and, for *F* = 6, we have an average gain of -10.49. Based on these results, we conclude that the best value for *F* is 4.

Overall, the best parameter combination, considering the synergy among them was *PS* = 92 and *F* = 2.

These results are interesting because they are counter-intuitive. When *F* grows, the *Jiva* can see more steps in the future but, this fact does not take into account the immediate cost of realizing the specific action. This action, even if it produces the best gain in long term, may lead the *Jiva* to situations where the short-term cost is very large, causing its death. This elucidates why the best configuration uses small values for *F* (*i.e. F*= 2) and the best average value for *F* (*i.e. F*= 4) is not the greater one.

C. Compromise between Intelligence and Performance

Increasing values of *PS* in the experiments (*e.g. PS* = 384) also do not help, because the *Jiva* "thinks" much more before acting, consequently, becoming too slow in such a dynamic world.

The parameters configuration *PS = 92* and *F* = 2 was found to be the best compromise of parameters for *Jiva*'s adaptability. In addition to establishing this compromise, regarding computational performance, *PS* = 92 and for *F* = 2 is also the best option (as shown in Figure 8).

V. CONCLUSION

This paper introduced the *Vidya* game and put forward a particular autonomous intelligent agent, the *Jiva*. It is an artificial living being capable to take non-monotonic behavioral decisions and learning by self-analyzing its own behavior in the world. The game also offers a new player-character interaction model, where the *Jivas* are not fully controlled by the player.

Computer games are continuously requiring good artificial intelligent agents, and the *Jiva*'s AI module intends to contribute towards this demand, especially at modeling the *Jiva*'s intelligence through Evolutionary Computation.

The use of Genetic Algorithms here proved to be a good choice because not only it produces good results in improving Jiva's intelligence (which was assessed through simulations on the *Jiva*'s adaptability), but also the computational performance was acceptable in supporting agents decision processes.

Regarding computation performance, we have noticed that processing time increased almost linearly with the problem size, namely, increase of *PS* and *F* parameters. In the worst case, the algorithm did not exceeded 600ms of processing time for one decision step – which is acceptable given de relative complexity of the environment.

Regarding intelligence performance, the results were also very interesting. We have learned about tradeoffs in increasing or decreasing the population size of the AG according to future steps in the inner actions simulations. We noticed that not too many future steps in the inner actions simulations is favorable towards the game objectives. That is, the fastest the *Jiva* thinks the better.

Finally, the *Jiva*'s adaptability results due to parameter setting were also useful in determining the best configuration that corresponds to a compromise between computational intelligence and performance.

We left as future works the following tasks:

- To reuse *Jiva*'s intelligence in other computer games and game scenarios;
- To analyze emergent behaviors of *Jiva*s societies;
- To study more complex ecosystems and societies behaviors using the proposed environment;
- To tackle uncertainties due occluded perception;
- To program *Vidya* in faster programming languages, such as C++, for higher performances.

REFERENCES

[1] S. Russell and P. Norving, *Artificial Intelligence: a modern approach*. New Jersey: Prentice-Hall, 1995.
[2] A. Nareyek, "AI in computer games," in Queue, Game Development: Serious Business, Serious Coding, vol. 1, issue 10, ACM Press, New York, 2004.
[3] B. C. Bridger and C. S. Groskopf, "Fundamentals of artificial intelligence in computer games," *Proceedings of the 38th annual on Southeast regional conference*, ACM Press, Clemson, South Carolina, 2000.
[4] J.E. Laird and M. van Lent, "Interactive Computer Games: Human-Level AI's Killer Application," in *Proc. Nat'l Conf. A.I.,* AAAI Press, Menlo Park, Calif., 2000.
[5] Z. Michalewicz and M. Michalewicz, "Evolutionary computation techniques and their applications," in IEEE International Conference on Intelligent Processing Systems, 1997.
[6] D. E. Goldberg, *Genetic algorithms in search, optimization, and machine learning*. New York: Addison-Wesley, 1989.
[7] L. Nang and K. Matsuo, "A survey on the parallel genetic algorithms," in JSICE, 1994.
[8] Z. Jinghui, "Comparison of Performance between Different Selection Strategies on Simple Genetic Algorithms," in CIMCA, vol. 2, 2005, pp. 1115 – 1121.
[9] C. Crowford, *The art of computer game design*. Washington State University, 1982.
[10] Sun Microsystems. *Java 2 Platform, Standard Edition*. Available: http://java.sun.com/javase/index.jsp (30/10/2006).
[11] K. Fukushima, "Recognition of occluded patterns: a neural network model," in IJCNN, 2000.
[12] J. E. Laird, "Using computer game to develop advanced AI," in Computer, vol. 34, issue 7, 2001, pp. 70-75.