Defence Research and Development Canada
Recherche et développement pour la défense Canada

DEFENCE DÉFENSE

Designing controllers for computer generated forces with evolutionary computing: Experiments in a simple synthetic environment

A. Taylor

Defence Research and Development Canada – Ottawa

Technical Memorandum
DRDC Ottawa TM 2012-162
October 2013

Canada

Designing controllers for computer generated forces with evolutionary computing: Experiments in a simple synthetic environment

A. Taylor

Defence Research and Development Canada – Ottawa
Technical Memorandum
DRDC Ottawa TM 2012-162
October 2013

Abstract

Military exercises and experiments are increasingly relying on synthetic environments to reduce costs and enable realistic and dynamic scenarios. However current simulation technology is limited by the simplicity of the controllers for the entities in the environment. Realistic simulations thus require human operators to play the roles of red and white forces, increasing their cost and complexity. Applied research project 13oc aims to reduce that human workload by improving artificial intelligence in synthetic environments. One approach identified early in the project was to use learning in artificial intelligence (AI). Further work identified evolutionary computing as a method of simplifying the design of AI. Described herein are experiments using evolutionary computing to design controllers in a simple synthetic environment. Controllers with a simple framework are evolved with genetic algorithms and compared to a hand-crafted finite-state-machine controller. Given careful parameter choices, the evolved controller is able to meet the performance of the hand-crafted one. Additional work is proposed to investigate ways of further simplifying the controller design process, and migrating the technology to military-specific synthetic environments.

Résumé

Les exercices et les essais militaires se fient de plus en plus aux environnements synthétiques pour réduire les coûts et permettre l’utilisation de scénarios réalistes et dynamiques. Toutefois, la technologie actuelle de simulation est limitée par la manque de complexité des contrôleurs des entités de l’environnement. Des opérateurs humains doivent donc jouer les rôles des forces rouges et blanches afin de créer des simulations réalistes, ce qui accroît les coûts et la complexité. Le projet de recherche appliquée 13oc vise à réduire la charge de travail humaine grâce à l’utilisation d’une intelligence artificielle (IA) améliorée dans les environnements synthétiques. Une approche définie en début de projet était d’utiliser une IA fondée sur l’apprentissage automatique. Les travaux effectués ont démontré que l’informatique évolutionnaire permet de simplifier la conception de l’IA. Nous décrivons dans la présente des expériences qui utilisent l’informatique évolutionnaire pour concevoir des contrôleurs pour un environnement synthétique simple. Des contrôleurs fondés sur un cadre simple évoluent au moyen d’algorithmes génétiques et sont ensuite comparés à un contrôleur automate fini créé à la main. Un contrôleur évolué fondé sur un choix judicieux de paramètres obtient un rendement comparable à celui du contrôleur créé à la main. Nous proposons d’effectuer d’autres travaux afin d’étudier des méthodes pour simplifier encore plus le processus de conception des contrôleurs et d’appliquer les techniques dans des environnements synthétiques militaires.

This page intentionally left blank.

Executive summary

Designing controllers for computer generated forces with evolutionary computing: Experiments in a simple synthetic environment

A. Taylor; DRDC Ottawa TM 2012-162; Defence Research and Development Canada – Ottawa; October 2013.

Background: The Canadian Forces are increasingly relying on simulation to drive training, exercises, and experimentation at all levels of command. But simulation technology is limited by the quality of artificial intelligence (AI) controllers driving the behaviour of entities in the synthetic environments. Applied Research Project 13oc aimed to improve this AI in Computer Generated Forces (CGFs). One aspect of AI identified for improvement was the ability to learn. Further research suggested work in evolutionary computing, such as genetic algorithms (GAs), as a means to achieve this capability. This paper presents experimental results in using GAs to automatically generate AI for a simple synthetic environment. The GA AI is compared with a human designed finite state machine (FSM) AI, which serves as a benchmark for performance.

Principal results: The concept behind the GA AI was to populate a simple framework for control; the synthetic environment world state as seen by the AI is simplified to be one of a finite number of states. Each state has an associated control response. The GA is used to optimize the control responses for each state. Parameters governing the number of discrete states were found by trial and error. Three GA controllers were evolved for comparison with the FSM controller. It was found that too few GA discrete states were unable to produce complex behaviour, and too many were unable to be fully optimized by the GA in a reasonable time. A controller with a medium number of states was able to approximately meet the FSM controller's performance. Its behaviour was qualitatively different than that of the other controllers.

Significance of results: This work is a first step in exploring the use of evolutionary computing for simplifying controller design in synthetic environments. It successfully demonstrated the ability of these methods to produce a controller on par with a hand-designed one. Thus these methods show promise in simplifying the design of AI for military CGFs.

Future work: There are many areas for further study. The basic controller optimized by the GA in this work is simple. But even so, both the controller and the GA required a great deal of attention by the designer – too much to claim success on the goal of automatic design. More sophisticated evolutionary methods such as genetic programming, or other methods of machine learning, may provide more sophisticated AI with less design work upfront. Meta-optimization methods may simplify the discovery of useful GA parameters. Furthermore, this technology needs to be transitioned into military-specific simulations, such as VBS2 or OneSAF.

Sommaire

Designing controllers for computer generated forces with evolutionary computing: Experiments in a simple synthetic environment

A. Taylor ; DRDC Ottawa TM 2012-162 ; Recherche et développement pour la défense Canada – Ottawa ; octobre 2013.

Contexte : Les exercices et les essais militaires se fient de plus en plus aux environnements synthétiques dans le cadre de l'instruction, d'exercices et d'essais à tous les niveaux de commandement. Par contre, les techniques de simulation sont limitées par la qualité des contrôleurs d'intelligence artificielle (IA) qui gèrent le comportement des entités dans les environnements synthétiques. Le projet de recherche appliquée 13oc vise à améliorer l'IA des forces générées par ordinateur (FGO). Un aspect d'IA ciblé aux fins d'amélioration était la capacité d'apprendre. Nos recherches ont suggéré que l'informatique évolutionnaire, par exemple les algorithmes génétiques (AG), permettrait de réaliser cette capacité. Nous présentons ici les résultats d'essais de génération automatique au moyen d'AG d'IA dans un environnement synthétique simple. L'IA de l'AG est comparée à une IA d'automate fini (AF) conçue à la main qui sert de point de référence en matière de performance.

Résultats principaux : Le concept qui sous-tend l'IA générée au moyen d'un AG est de créer un cadre simple de contrôle : l'état global de l'environnement synthétique vu par l'IA est simplifié pour ne représenter qu'un seul parmi le nombre fini d'états. Chaque état est associé à une réaction du contrôle. Les AG permettent d'optimiser les réactions du contrôle pour chaque état. Des expériences ont permis de déterminer le nombre paramètres à utiliser pour régir les états discrets. Trois contrôleurs ont été évolués au moyen d'AG et comparés au contrôleur AF. Nous avons ainsi déterminé qu'un nombre inadéquat d'états finis de l'AG ne permet pas de générer un comportement complexe, alors qu'un nombre excessif ne permet pas l'optimisation par l'AG dans un délai raisonnable. Par contre, un contrôleur prenant en charge un nombre intermédiaire d'états peut égaler la performance du contrôleur AF. Son comportement était qualitativement différent de celui des autres contrôleurs.

Portée des résultats : Nos travaux représentent la première étape d'exploration de l'utilisation de l'informatique évolutionnaire pour simplifier la conception de contrôleurs dans des environnements synthétiques. Ils ont permis de démontrer que ces méthodes peuvent créer un contrôleur comparable à ceux conçus à la main. Par conséquent, ces méthodes promettent de simplifier la conception d'IA pour les FGO militaires.

Recherches futures : Plusieurs domaines d'étude s'ouvrent à nous. Le contrôleur de base optimisé au moyen d'un algorithme génétique dans le cadre de la présente étude est fort simple. Néanmoins, le contrôleur généré par l'AG a exigé une grande attention de la part du concepteur : ce n'est donc pas une réussite sur le plan de la conception automatisée. Des méthodes évolutionnaires plus sophistiquées, par exemple la programmation génétique

ou d’autres méthodes d’apprentissage automatique, peuvent permettre de créer des IA plus sophistiquées au moyen d’un moindre effort de conception initial. Des méthodes de métaoptimisation peuvent simplifier la découverte des paramètres utiles d’AG. De plus, il faut appliquer la technologie dans le cadre de simulations militaires effectuées, par exemple, au moyen de VBS2 ou de OneSAF.

This page intentionally left blank.

Table of contents

List of figures

This page intentionally left blank.

1 Introduction

The military is increasingly relying on simulation-based exercises for training, experimentation, and concept development. In order to fully leverage the benefits of synthetic environments, computer controlled entities must behave realistically and appropriately for a scenario. But this is a difficult goal to achieve. Scenario designers do not have the resources to design behaviour for every possible situation that each entity may encounter. As a result, humans are routinely employed to control adversaries in distributed simulations.

Applied research project 13oc aimed to reduce the human workload in simulation by improving artificial intelligence (AI) controllers in Computer Generated Forces (CGFs). Better AI would reduce the number of supporting personnel required to run scenarios. The project identified specific capability gaps in widely available artificial intelligence for CGF software [1]. Chief among these was the ability to learn. Consequently machine learning and evolutionary computation were investigated as methods to simplify AI design. As a first step, a survey of evolutionary and learning methods for designing AI was performed [2]. The survey identified a range of methods of varying sophistication. As a follow-up to the study, experiments were created to us a simple evolutionary algorithm to automatically generating AI. Here the goal is to optimize AI parameters choices using a genetic algorithm (GA). This experiment is a first case study; the desired outcome is to help direct future work to solve more challenging AI design problems using learning and evolutionary computing.

The hypothesis for the experiment is this: in some synthetic environment, can a controller designed with evolutionary methods perform as well as a benchmark human-designed controller? A basis for the experiment was found in an AI programming textbook [3]. The book provides code for an Asteroids video game clone, using it to illustrate various methods of AI popular in the video game industry. The benchmark AI is a Finite State Machine from the text. The text also describes a GA-generated controller designed solely to avoid collisions. This GA controller was extended to be capable of playing the full game (described in more detail below). In this document: the game and the two AI structures are described in detail; the experiments, in which a GA used to generate the GA-based AI, are described; and a performance analysis is presented of the two controllers.

1.1 The synthetic environment test bed

The synthetic environment used for these experiments is based on the classic arcade game Asteroids [1] [3]. In Asteroids, the player controls a ship in a two-dimensional playfield – see Figure 1. The playfield wraps over both edges (i.e. the game is played on the surface of a toroid). In each "wave", a number of asteroids appear on the screen and must be destroyed by the player to earn points. When an asteroid is shot it splits into two smaller asteroids; the smallest possible asteroids are destroyed when shot. The player loses a life if the ship collides with an asteroid. The player's ship has momentum, and is capable of turning, thrusting, and shooting.

[1] http://en.wikipedia.org/wiki/Asteroids_(video_game)

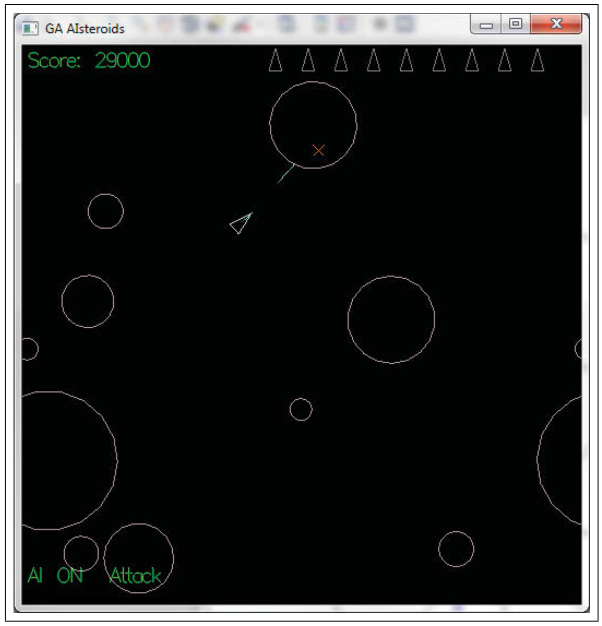


Figure 1: A screen capture of the version of Asteroids used in our experiments.

The game was modified from the textbook source in several ways for these experiments. In contrast to the classic Asteroids game, the ship in the experimental version has no hyperspace feature (a teleport ability designed to escape collisions), and players do not earn extra lives as the game progresses. There are no power-ups (which upgrade the ship's weapon). The game was also modified with options to run as fast as possible (not limited to wall-clock time), and to not render the screen, which also decreases each game's running time. Further modifications support multiple batch runs, and the production of log files containing information about each game. These changes reduced the time and complexity of running the many thousands of trials necessary for the GA and the evaluation of AI controllers.

1.2 Overview of the artificial intelligence controllers

Here two types of AIs are compared. The first is a finite state machine (FSM), referred to as FSM-AI. It is essentially the same as the one described in the text [3]. The second is a GA-based controller, denoted the GA-AI. It is an extension of the GA-based collision avoidance controller in the text. Both take in information on the game state, derive a simplified world view, and produce control decisions, limited to turning, thrusting, and firing the ship's guns.

The AI controllers described below (as well as humans players) make decisions based on the game state, so it is helpful to understand its full scope. The complete state of the environment is available to the player at any time. The game world consists of the objects in the game: the ship, bullets, and asteroids. Each of these has properties: a position, size, and velocity vector. The ship also has a heading angle and control state (thrusting, turning, firing). The ship also has a number of remaining lives and score. Game speeds are measured internally in units per second. Time in the game is measured in frames, with each frame lasting 1/60 seconds. The game can be run at "wall-clock time", at 60 frames per second, or without a frame limit at the maximum speed allowed by its host's processor speed.

1.2.1 Finite State Machine AI

The FSM-AI controller is a slightly modified version of the one in the text [3] (it was modified to remove any power-up logic, and also to properly deal with the wrapping-nature of the screen edges, and its ability to reverse thrust was removed for consistency with the GA-AI's abilities). The FSM-AI finite state machine is shown in Figure 2. Each FSM state activates a different program that makes control decisions based on the immediate world state.

The FSM takes in a subset of information from the game world, derived from its internal state and the state of the nearest asteroid (all other asteroids are ignored). From this, it calculates:

- Distance and velocity vectors to the nearest asteroid.
- An estimate of the asteroid's future position. Depending on the state, this will be an estimate of where to shoot, or where a collision will occur based on the ship and asteroid's current speed and heading.

The FSM has four states, called Idle, Approach, Attack, and Evade. At the start of each frame, the FSM-AI code determines if a transition is required. As can be seen in Figure 2, transitions are governed by two factors: the distance to the asteroid and a flag indicating that a collision is imminent. If the collision flag is set, all states will transition to Evade. When the asteroid is further than a predefined approach distance, d_{APPROACH}, the FSM switches to Approach. When the asteroid is closer than d_{APPROACH}, the FSM switches to Attack.

The FSM's behaviour in each individual state is as follows:

- Idle: Transition state; only exists to simplify adding new states and deal with frames where there are no asteroids.
- Approach: turn towards asteroid's predicted future position and thrust.
- Attack: turn towards asteroid's current position; when within one degree, fire.
- Evade: if the asteroid is in front, turn towards it and fire. If it is behind the ship or to the side, turn away from the asteroid and thrust.

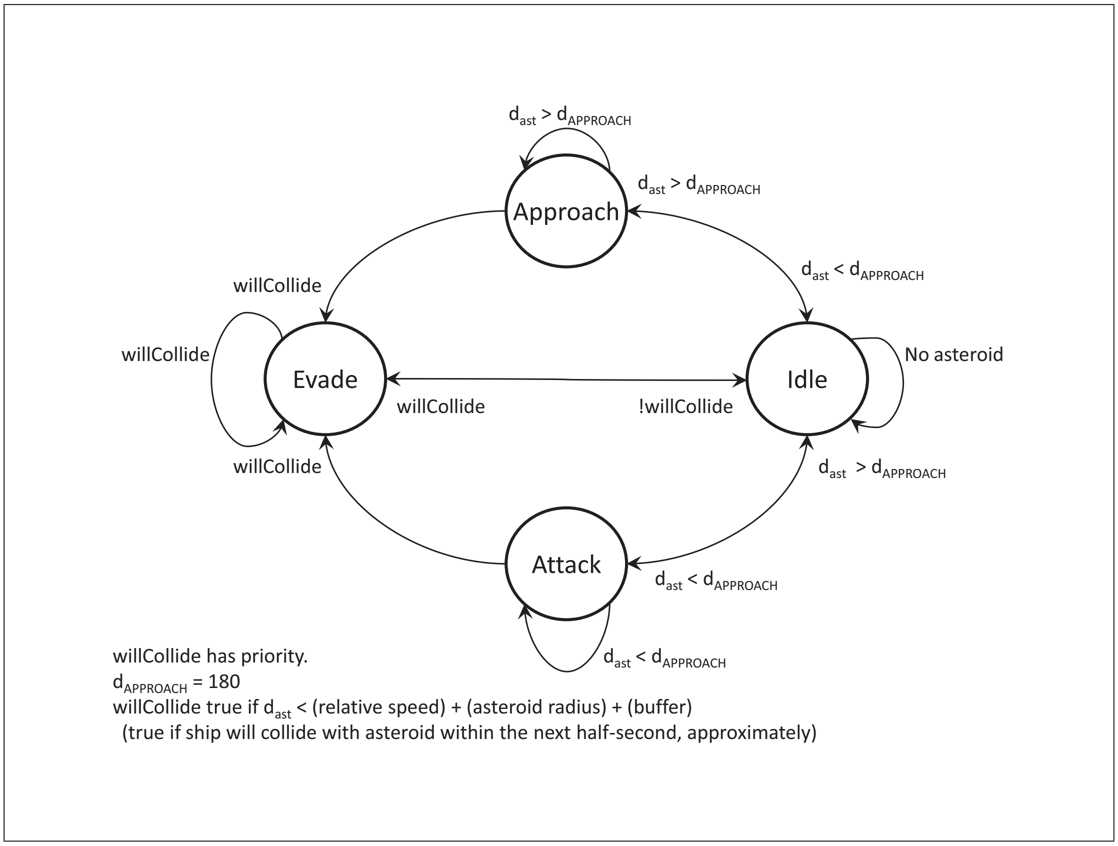


Figure 2: The FSM-AI state machine. Conditions for each transition are shown on the arrows. Transition variables are: d_{ast}, the distance to the nearest asteroid; d_{APPROACH}, a hard-coded threshold distance; and willCollide, a flag set as described in the legend. So for example, if the FSM-AI is in the Attack state, it will remain there as long as the asteroid remains within the approach distance of 180 units. If the willCollide flag becomes set, the AI transitions to Evade. If the asteroid is destroyed, the transition will depend on the distance to the nearest one.

The FSM is a purely reactive controller. Every decision is based on the immediate game state – it has no memory of previous states[2] As will be seen in Section 2.4, the FSM-AI is a highly effective controller for the game.

1.2.2 Genetic Algorithm AI

The Genetic Algorithm Artificial Intelligence (GA-AI) is based on a framework for a simple reactive controller. The idea is to simplify the game world state by categorizing it into a small finite set of discrete states, and responding to each one with a predefined control output. The discrete world state is based on three numbers: the angle, distance, and

[2]The FSM *does* make use of time-history-dependent information such as velocity, but this is obtained directly from the game engine and is not calculated using memory.

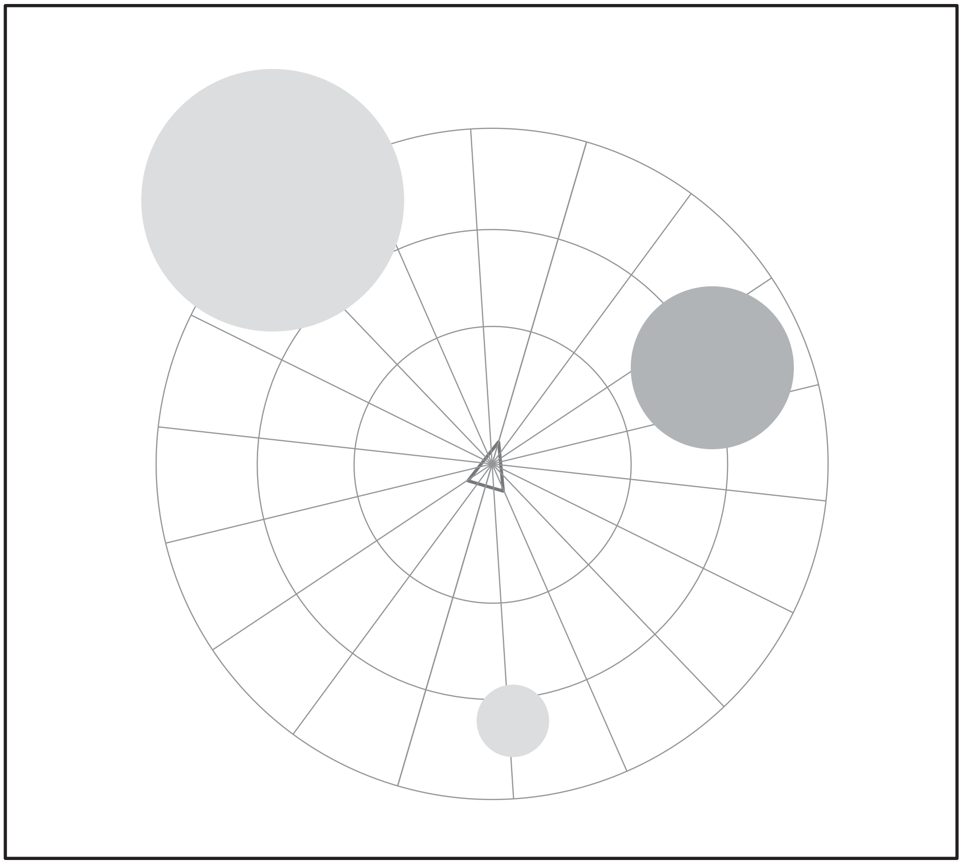


Figure 3: An illustration of the GA-AI discrete view of the world. The active sector is the one containing the closest point on the asteroid's perimeter to the ship (asteroids are perfect circles). In the situation illustrated here, the AI will react to the asteroid to its forward right, indicated by its darker face colour.

"collision" (defined as the relative speed) to the nearest asteroid. Each of these is mapped to a fixed interval and quantized. This arrangement is illustrated in Figure 3 (note that only distance and angle are shown in the figure – there is a separate grid for each possible collision state). Each possible combination of the three values maps to a fixed control response: a combination of shoot, turn, and thrust decisions. The active response is determined by the nearest asteroid's position in the grid. Specifically, the active grid cell is the one containing the closest point on the asteroid's perimeter.

The genetic algorithm provides the control responses. In order to find the best possible response, populations of randomly generated controllers compete and are combined and mutated to form new generations. This goal of this search procedure is to produce behaviours capable of playing the game effectively. Experiments using this GA are described in the next section.

2 Experiments

In this section the results of evolving the GA-AI are compared with the FSM-AI.

2.1 Genetic algorithm Implementation

As described in Section 1.2.2, The GA-AI maps the game world state to a single discrete state. The GA goal is to automatically populate the GA-AI agent's discrete world view with control actions. The GA does this by searching the space of possible controllers to find effective ones.

Genetic Algorithms are a mature optimization method [4, 5]. They are used to optimize a string of parameters, called a genome, according to a fitness criteria. For this work the genome is the list of control decisions for each discrete input state, and the fitness is the game score. The algorithm starts with a randomly generated population of genomes. Each is tested for fitness by playing the game. A new generation is constructed from the old using crossover and mutation operators. First two parents are selected from the previous generation with probability proportional to their fitness. With crossover probability ($p_{\text{crossover}} = 0.7$), their genes are swapped at a random point, or they are both copied directly, producing two children. The control decisions in each child are then randomly mutated to new values with a low mutation probability ($p_{\text{mutation}} = 0.001$). The procedure is repeated until some criteria is met, e.g. new populations are no better than previous ones, or a fixed number of iterations have occurred. In the trials below, the GA was run with 100 individuals over 500 generations.

The fitness is based on the game score. Each individual is allowed to play the game with ten lives, for a maximum of 1000 seconds[3]. It was found that game scores for a given individual are highly variable, even with the hand-crafted FSM-AI; taking the median of multiple games reduces variability, and thus strengthens the fitness signal. However playing more games also increases the execution time for each generation. The median of ten games was found to be a reasonable tradeoff between these constraints. As will be seen, game scores for a given controller can vary by orders of magnitude.

2.2 Evolving three populations of controllers

Experiments were performed to generate controllers using the GA-AI framework. An important decision in the framework is choosing the number of discrete distance, collision, and direction states. It was found that having a high number of states led to poor performance – the GA, running with our parameters, was unable to find good solutions with too many variables. Controllers with very few states were also unable to perform well. It was decided to compare the evolution of three controllers, each with an order of magnitude more parameters than the last:

[3]The time limit mitigates stalemates, in which a ship's behaviour creates a situation where it will not die or destroy the remaining asteroids. Stalemates occur for both the GA-AI and FSM-AI.

- Tiny: 8 direction states, 1 distance state, 1 collision state (8 total states, 24 free parameters)
- Little: 18 direction states, 2 distance states, 2 collision states (72 total states, 206 free parameters).
- Big: 18 direction states, 10 distance states, 10 collision states (1800 total states, 3600 free parameters).

Note the number of free parameters (control decisions) to be optimized in each controller variant. For each state, there are three possible control choices (turn, thrust, and shoot), giving three times the number of states in free parameters.

Figures 4, 5, and 6 show the Tiny, Little, and Big population evolutions respectively. Each graph shows the population's mean fitness, best individual fitness, and best individual fitness to date. The population diversity, as measured by average number of differences between individual genomes, is also shown. Several observations about the three populations follow.

Rate of improvement. The Tiny population finds good genomes relatively quickly, in the first 100 generations. The Little population increases in fitness more gradually, punctuated with periods of rapid improvement. Progress slows after 250 generations, but still increases steadily. The Big population improves most slowly, with no rapid improvements. The best Big fitness after 500 generations is just under 90000 – much lower than either of the other two controllers.

Population diversity. The Tiny population diversity rapidly falls to about 2, at which point the entire population has almost exactly the same genome. A gradual increase towards the end of the run may be related to the final jump in performance. The Little population diversity drops more slowly, stabilizing around 15 or 7%. The Big population diversity drops rapidly with the initial generations, stabilizing at 500 or about 28%.

Overall performance The Tiny controller peaks at a fitness of approximately 120000. The Little controller peaks at around 140000, and the Big tops out at just under 90000. Overall, the Little controller is the best performing one. The Tiny controller may be insufficiently complex to achieve better performance. The Big controller has too many parameters to optimize given the GA parameters and algorithm used here. The Little controller is a special case of the Big one, so the Big controller could at least match its performance. However the GA used here was unable to produce that level of performance, perhaps due to the size of the search space defined by the Big controller's parameters.

2.3 Comparing the FSM-AI and GA-AI performance

The FSM-AI and best Tiny, Little, and Big GA-AI controllers were compared by running many games with each one and generating histograms of their scores. Recall that fitness in the GA is defined as the median score in 10 games. The histograms here give a more

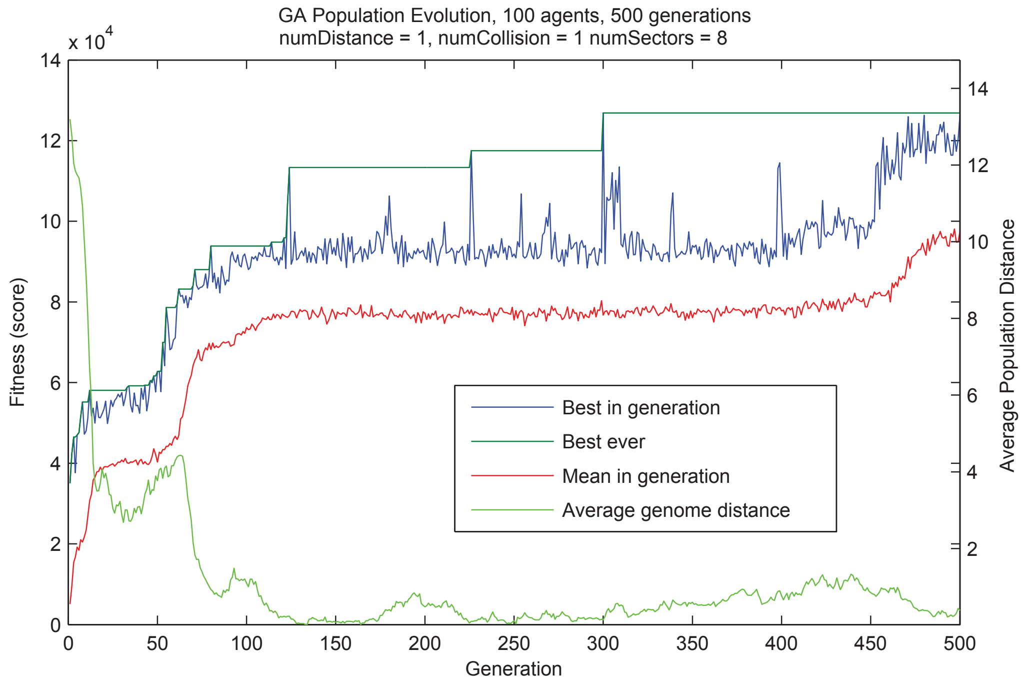


Figure 4: Evolution of a population for a simple controller (“Tiny”) with just eight direction states.

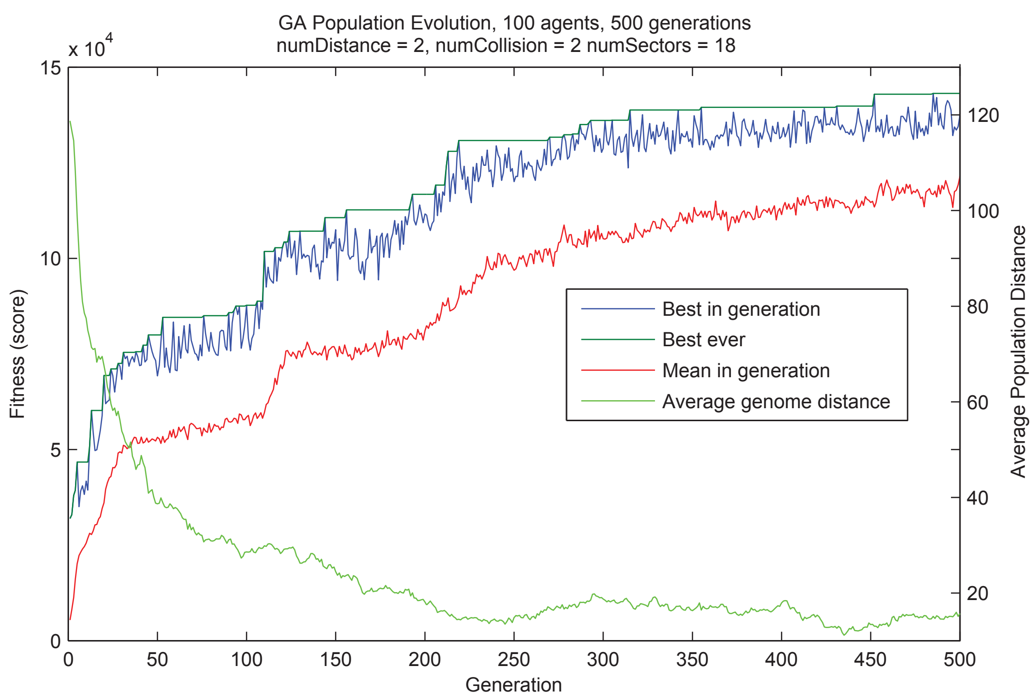


Figure 5: Evolution of a somewhat complex controller ('Little') with 2 distance and collision states (each), and 18 direction states.

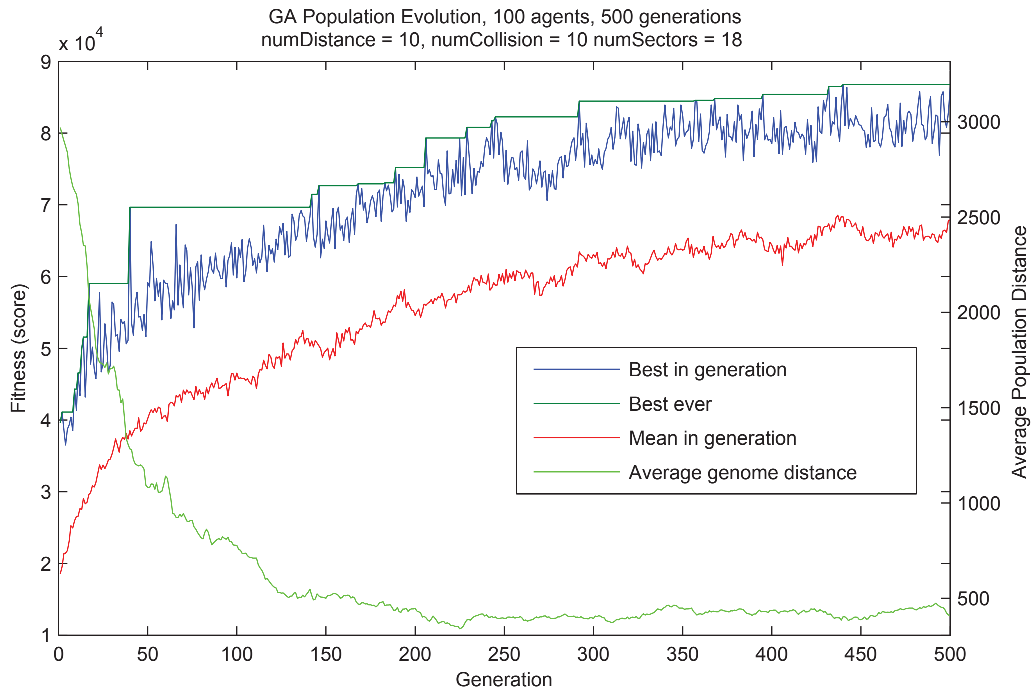


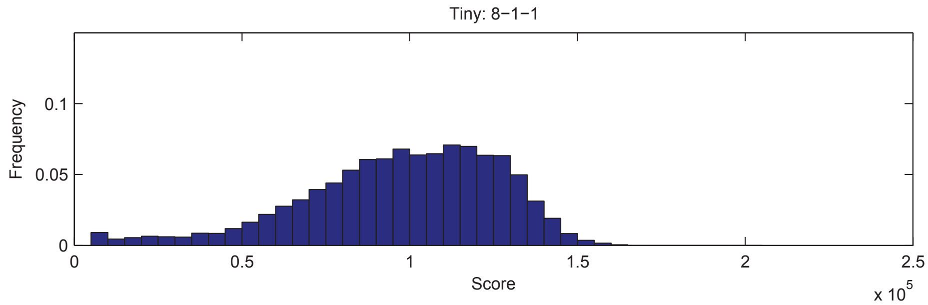
Figure 6: Evolution of a complex controller ('Big') with 10 distance and collision states (each), and 18 direction states.

complete measure of any controller's performance. Each AI type played the game 10000 times, with ten lives and a time limit of 1000 seconds.

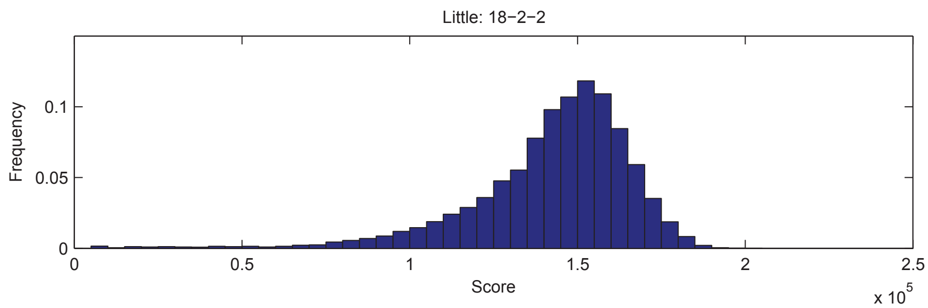
The histograms of each AI's performance are shown in Figure 7. Score frequency is given as a percentage of total games. There are two patterns in performance. The Tiny GA-AI and FSM-AI have broader score spreads, with long tails into low scoring regions. This could indicate that randomness plays a greater role in their performance than for the other controllers – the low scores may mean they are vulnerable to fast losses. Alternatively, because of the time limit on games, it could indicate that they are more prone to being caught in stalemates and fail to accumulate points quickly. The Little and Big GA-AI controllers are more consistent. It is interesting that the Big GA-AI is consistently bad – it is the worst performer overall. The Little GA-AI performance is in par with the FSM-AI.

2.4 Analysis and qualitative description of each controller's behaviour

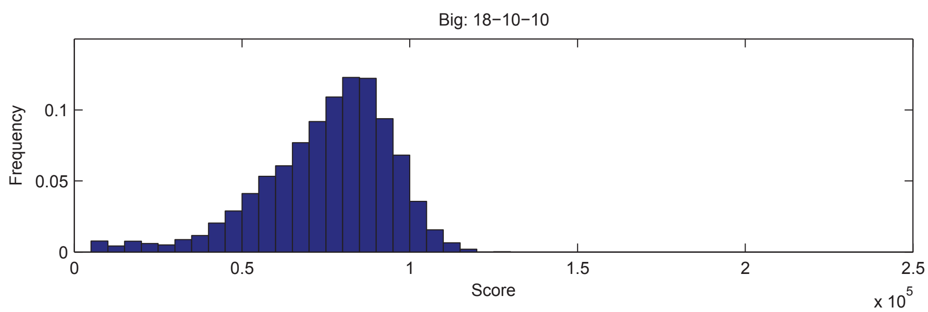
Figure 8 shows the best Tiny GA-AI controller. As seen in the previous section this controller is able to reach reasonably high scores, although it has a long tail of low scores. Since it lacks distance or collision states, it can only respond to the direction of the nearest asteroid. The controller almost always shoots. In play, it tends to circle clockwise around targets in its front 180 degree arc. This often puts them in its line of fire, and also avoids



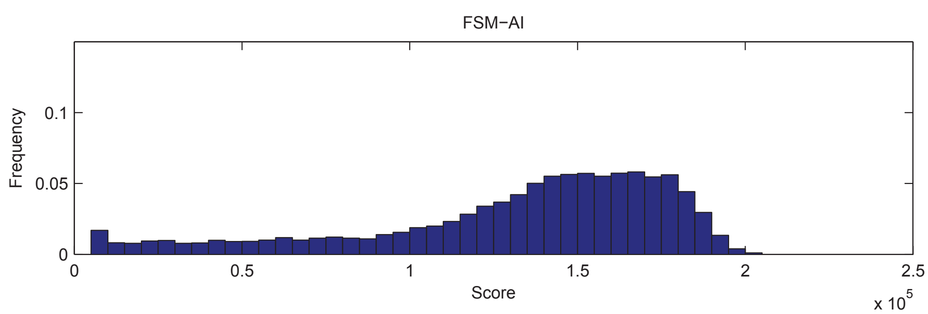
(a) Best Tiny GA-AI controller performance distribution.



(b) Best Little GA-AI controller performance distribution.



(c) Best Big GA-AI controller performance distribution.



(d) FSM controller performance distribution.

Figure 7: Performance distribution of the best GA-AI and FSM controllers over ten-thousand games.

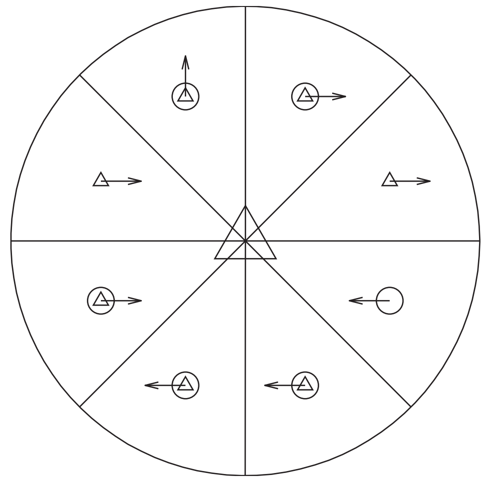


Figure 8: The best performing Tiny GA-AI controller (8 direction states, 1 distance state, 1 collision state). In each sector: arrows indicate turn commands, triangles indicate thrust, and circles indicate shooting.

collisions. Targets in its rear arc cause the ship to turn away and thrust, also avoiding collisions. The wrapping nature of the game world means the target will be in view in the front again in short order. The overall behaviour appears very simple: the ship mainly turns in one direction and thrusts, constantly firing. However in doing this it blasts through asteroid fields, circling around them and rarely colliding with them. Though simple, this is an effective strategy. Although the Tiny GA-AI is not the best player, it is remarkable that such a simple controller is able to reach scores of the same order as the much more sophisticated FSM-AI.

Figure 9 illustrates the genome of the best-performing Little GA-AI. It is difficult to interpret its behaviour by examining the control decisions in the figure. But to an observer, this controller's behaviour appears much more nuanced than Tiny to the observer. It turns towards asteroids, thrusting if they are distant, and correcting its aim to hit even small targets. When overshooting or chasing an asteroid far away, it breaks off and thrusts, leading to the acquisition of a different target or a more favourable approach to the first thanks to the screen wrapping. This behaviour is qualitatively different than the FSM-AI, but the GA-AI matches it in performance. This is especially surprising given that the GA-AI lacks the ability to predict future asteroid positions. However it has discovered other strategies that compensate for this deficiency, and produced a highly effective play style.

Figure 10 shows the best overall Big GA-AI controller. Only the response for collision state 1 is shown – the full controller has ten such responses. It is difficult to discern an overall pattern to the Big controller's behaviour by looking at its decisions, or by watching it play. It appears that the GA has retained much of the randomness from its initial population; for example, it makes many small and often counter-productive course corrections while approaching a target. In some situations it appears to be acting entirely randomly,

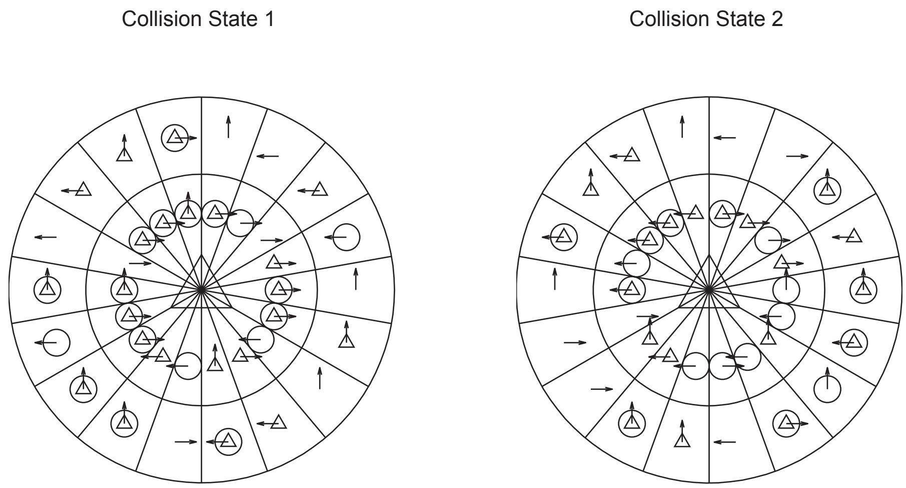


Figure 9: The best performing Little GA-AI controller (18 direction states, 2 distance states, and 2 collision states). In each sector: arrows indicate turn commands, triangles indicate thrust, and circles indicate shooting.

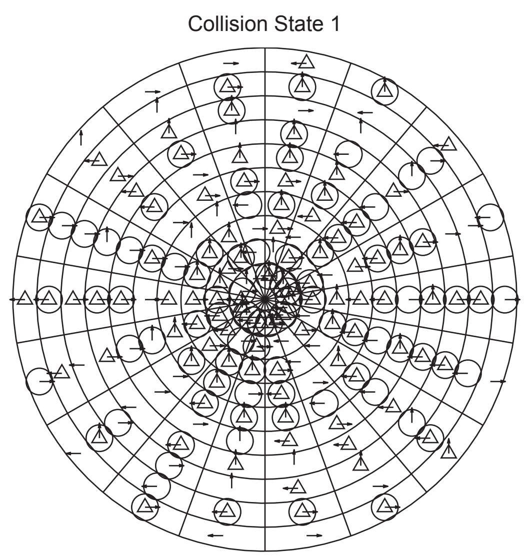


Figure 10: The best performing Big GA-AI controller (18 direction states, 10 distance states, and 10 collision states). In each sector: arrows indicate turn commands, triangles indicate thrust, and circles indicate shooting.

frequently making decisions that lead to asteroid collisions. It is likely that there is a lot of noise in the controller, meaning its control responses have not been optimized. Without further experimentation on improving the search for better parameter choices, it is difficult to say whether this higher-dimensional controller can provide improved or more effective performance.

The FSM-AI behaviour is markedly different than that of any of the GA-AI controllers. Refer to Figure 2 to review the FSM-AI's state machine. The difference in behaviour in different states is very obvious:

- Attack: The ship never thrusts. It turns toward the asteroid's future predicted position, then stops turning and fires until the asteroid is destroyed or the ship must adjust its aim. In a crowded asteroid field, the AI spends most of its time in this state, destroying its targets with robot-like efficiency.
- Approach: The ship turns until it is facing the predicted asteroid intercept position, then thrusts, making minor corrections as it travels. The AI switches to Attack when it gets close enough.
- Evade: When in Evade, the behaviour depends on the position of the asteroid. If the ship is facing its target, it turns towards it a shoot. If the asteroid is behind it, the ship turns away and thrusts.
- Idle: A transition state. The AI remains in this state for just one frame, transitioning to one of the other three immediately.

The FSM-AI spends most of its time in the Attack state. As a result, its behaviour appears very robot-like and efficient: it mainly drifts through the asteroid field, lining up shots, and thrusting only when threatened with a collision. In contrast, the GA-AI combines shooting with other controls. Human players tend to combine actions as well, making the FSM-AI actually seem more artificial than the GA-AI ones. And the matching performance of the Little GA-AI, bereft of the ability to predict future asteroid positions, is a success for the GA design method.

3 Conclusions

Using the GA-AI framework,a genetic algorithm was employed to evolve a controller for a clone of the game Asteroids. The resulting controller was capable of performance close to that of a hand-crafted FSM controller. From this point of view the experiment is a success, but it raises many questions about the use of this or related techniques for automatically generating controllers for more sophisticated environments.

The constraints imposed by the controller structure create inherent limitations in its potential. For example, the GA-AI is only aware of the nearest asteroid; however there will always be situations where a further asteroid is actually more threatening than the closest one. It would be preferable to present the controller with all the game state information and let it discover useful game meta-data (e.g. estimating future target positions) on its own. This would require a learning framework more sophisticated than simple parameter optimization with GAs. In the evolutionary computing literature most researchers use genetic programming; this allows for the creation of more sophisticated controllers (e.g. see Szita [6]). Alternatively the evolution algorithm could be improved; techniques like complexification [2] or meta-optimization [7] could be employed to achieve good results with more complex or evolving controller structures. In a more sophisticated synthetic environment, algorithms will not have the luxury of being able to run the simulation tens of thousands of times to evolve effective controllers. If this is the case, algorithms based on learning may be more efficient at developing strategies from the environment than those based on evolution. Furthermore, the work needs to move out of the Asteroids test bed used here, for reasons outlined below.

3.1 Future work

A number of topics are suggested for follow-on work:

- **Increasing fitness robustness**. Devising an effective fitness signal was difficult; using the median of 10 games in this work was key to producing useful results, but there is room for more efficient methods. Statistics has many methods comparing noisy measurements (e.g. Welch's t-test [4]); these could be employed to reduce the number of trials required, or improve the strength of the fitness measurements. Either would improve the efficiency of the genetic algorithm.
- **Optimizing the genetic algorithm parameters**. The GA parameters used in this work were found through trial and error; again, other parameter combinations resulted in dramatically worse results. Extending the GA to automatically tune itself through some meta-optimization would vastly simplify its use in other contexts. Simplifications such as this will be required if the automatic design approach is to become useful to the general scenario designer.

[4] `http://en.wikipedia.org/wiki/Welch's_t_test`

- **Separate the evolutionary algorithm from the synthetic environment**. The test bed described here included the code for applying the genetic algorithm. A more flexible approach would be to run the algorithm externally. This would also allow for the use of existing evolutionary computation suites, e.g. ECJ [4].

- **More flexible controller structures**. The AI shown here uses the GA for nothing more than tuning its parameters. The GA-AI controllers here are very structured, defined by just three parameters. The ideal number of quantization levels for each parameter is unknown. To enable automatic design, the controller structure must be able to evolve or learn on its own. Future work should pursue the use of genetic programming, or other methods capable of exploring more complex structures and deriving more information from the synthetic environment. Controllers capable of gradually increasing their own complexity may be a way of managing this process without being overwhelmed by a large parameter space.

- **Using a mature synthetic environment**. A great deal of time was spent debugging the Asteroids environment used here. The academic community has identified a small number of game platforms used in its research and in competitions, such as Ms. Pac-Man [8], Super Mario [9], or a first-person game like Unreal Tournament [10]. Using these would save time, as they are already robust platforms for AI experimentation. These also benefit from already having benchmarks for performance comparison. Alternatively, pursuing this approach in the context of military-specific synthetic environments such as VBS2 or OneSAF will be necessary in eventually demonstrating its effectiveness to the client community.

- **Pursuing other learning methods**. While the GA used here was effective, other learning and evolutionary computing methods may be more efficient. Many other algorithms and methods have been published in the academic community [2]. The best learning AI solutions from these environments could be applied to military-specific synthetic environments.

References

[1] Taylor, A., Abdellaoui, N., and Parkinson, G. (2009), Artificial Intelligence in Computer Generated Forces: Comparative Analysis, Huntsville, Alabama: The Society for Modeling and Simulation International.

[2] Taylor, A. (2011), Survey of evolutionary learning for generating agent controllers in synthetic environments, (Technical Memorandum DRDC Ottawa TM 2011-159) Defence R&D Canada – Ottawa, Ottawa, Canada.

[3] Schwab, B. (2004), AI game engine programming, Charles River Media.

[4] Luke, S. (2009), Essentials of Metaheuristics, Lulu. Available for free at http://cs.gmu.edu/~sean/book/metaheuristics/.

[5] Mitchell, M. (1998), An introduction to genetic algorithms, The MIT press.

[6] Szita, I. and Lõrincz, A. (2007), Learning to play using low-complexity rule-based policies: Illustrations through Ms. Pac-Man, *Journal of Artificial Intelligence Research*, 30(1), 659–684.

[7] Smit, S. K. and Eiben, A. E. (2009), Comparing parameter tuning methods for evolutionary algorithms, In *IEEE Congress on Evolutionary Computation, 2009. CEC '09*, pp. 399–406, IEEE.

[8] Lucas, S. M. (2007), Ms Pac-Man competition, *ACM SIGEVOlution*, 2(4), 37–38.

[9] Karakovskiy, S. and Togelius, J. (2012), The Mario AI Benchmark and Competitions, *IEEE Transactions on Computational Intelligence and AI in Games*, 4(1), 55–67.

[10] van Hoorn, N., Togelius, J., and Schmidhuber, J. (2009), Hierarchical controller learning in a First-Person Shooter, In *IEEE Symposium on Computational Intelligence and Games, 2009. CIG 2009*, pp. 294–301, IEEE.

DOCUMENT CONTROL DATA

(Security markings for the title, abstract and indexing annotation must be entered when the document is Classified or Designated.)

1. ORIGINATOR (The name and address of the organization preparing the document. Organizations for whom the document was prepared, e.g. Centre sponsoring a contractor's report, or tasking agency, are entered in section 8.)

Defence Research and Development Canada – Ottawa
3701 Carling Avenue, Ottawa ON K1A 0Z4, Canada

2a. SECURITY MARKING (Overall security marking of the document, including supplemental markings if applicable.)

UNCLASSIFIED

2b. CONTROLLED GOODS

(NON-CONTROLLED GOODS)
DMC A
REVIEW: GCEC December 2013

3. TITLE (The complete document title as indicated on the title page. Its classification should be indicated by the appropriate abbreviation (S, C or U) in parentheses after the title.)

Designing controllers for computer generated forces with evolutionary computing: Experiments in a simple synthetic environment

4. AUTHORS (Last name, followed by initials – ranks, titles, etc. not to be used.)

Taylor, A.

5. DATE OF PUBLICATION (Month and year of publication of document.)	6a. NO. OF PAGES (Total containing information. Include Annexes, Appendices, etc.)	6b. NO. OF REFS (Total cited in document.)
October 2013	34	10

7. DESCRIPTIVE NOTES (The category of the document, e.g. technical report, technical note or memorandum. If appropriate, enter the type of report, e.g. interim, progress, summary, annual or final. Give the inclusive dates when a specific reporting period is covered.)

Technical Memorandum

8. SPONSORING ACTIVITY (The name of the department project office or laboratory sponsoring the research and development – include address.)

Defence Research and Development Canada – Ottawa
3701 Carling Avenue, Ottawa ON K1A 0Z4, Canada

9a. PROJECT OR GRANT NO. (If appropriate, the applicable research and development project or grant number under which the document was written. Please specify whether project or grant.)	9b. CONTRACT NO. (If appropriate, the applicable number under which the document was written.)
13oc	

10a. ORIGINATOR'S DOCUMENT NUMBER (The official document number by which the document is identified by the originating activity. This number must be unique to this document.)	10b. OTHER DOCUMENT NO(s). (Any other numbers which may be assigned this document either by the originator or by the sponsor.)
DRDC Ottawa TM 2012-162	

11. DOCUMENT AVAILABILITY (Any limitations on further dissemination of the document, other than those imposed by security classification.)

(X) Unlimited distribution
( ) Defence departments and defence contractors; further distribution only as approved
( ) Defence departments and Canadian defence contractors; further distribution only as approved
( ) Government departments and agencies; further distribution only as approved
( ) Defence departments; further distribution only as approved
( ) Other (please specify):

12. DOCUMENT ANNOUNCEMENT (Any limitation to the bibliographic announcement of this document. This will normally correspond to the Document Availability (11). However, where further distribution (beyond the audience specified in (11)) is possible, a wider announcement audience may be selected.)

UNLIMITED

13. ABSTRACT (A brief and factual summary of the document. It may also appear elsewhere in the body of the document itself. It is highly desirable that the abstract of classified documents be unclassified. Each paragraph of the abstract shall begin with an indication of the security classification of the information in the paragraph (unless the document itself is unclassified) represented as (S), (C), or (U). It is not necessary to include here abstracts in both official languages unless the text is bilingual.)

Military exercises and experiments are increasingly relying on synthetic environments to reduce costs and enable realistic and dynamic scenarios. However current simulation technology is limited by the simplicity of the controllers for the entities in the environment. Realistic simulations thus require human operators to play the roles of red and white forces, increasing their cost and complexity. Applied research project 13oc aims to reduce that human workload by improving artificial intelligence in synthetic environments. One approach identified early in the project was to use learning in artificial intelligence (AI). Further work identified evolutionary computing as a method of simplifying the design of AI. Described herein are experiments using evolutionary computing to design controllers in a simple synthetic environment. Controllers with a simple framework are evolved with genetic algorithms and compared to a handcrafted finite-state-machine controller. Given careful parameter choices, the evolved controller is able to meet the performance of the hand-crafted one. Additional work is proposed to investigate ways of further simplifying the controller design process, and migrating the technology to military-specific synthetic environments.

Les exercices et les essais militaires se fient de plus en plus aux environnements synthé-tiques pour réduire les coûts et permettre l'utilisation de scénarios réalistes et dynamiques. Toutefois, la technologie actuelle de simulation est limitée par la manque de complexité des contrôleurs des entités de l'environnement. Des opérateurs humains doivent donc jouer les rôles des forces rouges et blanches afin de créer des simulations réalistes, ce qui accroît les coûts et la complexité. Le projet de recherche appliquée 13oc vise à réduire la charge de travail humaine grâce à l'utilisation d'une intelligence artificielle (IA) améliorée dans les environnements synthétiques. Une approche définie en début de projet était d'utiliser une IA fondée sur l'apprentissage automatique. Les travaux effectués ont démontré que l'informatique évolutionnaire permet de simplifier la conception de l'IA. Nous décrivons dans la présente des expériences qui utilisent l'informatique évolutionnaire pour concevoir des contrôleurs pour un environnement synthétique simple. Des contrôleurs fondés sur un cadre simple évoluent au moyen d'algorithmes génétiques et sont ensuite comparés à un contrôleur automate fini créé à la main. Un contrôleur évolué fondé sur un choix judicieux de paramètres obtient un rendement comparable à celui du contrôleur créé à la main. Nous proposons d'effectuer d'autres travaux afin d'étudier des méthodes pour simplifier encore plus le processus de conception des contrôleurs et d'appliquer les techniques dans des envi-ronnements synthétiques militaires.

14. KEYWORDS, DESCRIPTORS or IDENTIFIERS (Technically meaningful terms or short phrases that characterize a document and could be helpful in cataloguing the document. They should be selected so that no security classification is required. Identifiers, such as equipment model designation, trade name, military project code name, geographic location may also be included. If possible keywords should be selected from a published thesaurus. e.g. Thesaurus of Engineering and Scientific Terms (TEST) and that thesaurus identified. If it is not possible to select indexing terms which are Unclassified, the classification of each should be indicated as with the title.)

artificial intelligence; genetic algorithm; evolutionary computing

Defence R&D Canada

Canada's leader in Defence
and National Security
Science and Technology

R & D pour la défense Canada

Chef de file au Canada en matière
de science et de technologie pour
la défense et la sécurité nationale

DEFENCE R&D DÉFENSE

www.drdc-rddc.gc.ca