Pursuit-Evasion using Evolutionary Algorithms in an Immersive Three-Dimensional Environment

Malachy Eaton, *Member IEEE*, Martin McMillan and Michael Tuohy

Dept. of Computer Science and Information Systems
University of Limerick, Limerick, IRELAND

email: malachy.eaton@ul.ie

Abstract

In view of the biological prevalence of pursuit-evasion contests they provide a useful test-bed for research into novel bio-inspired computing and control systems. In this paper we investigate the evolution of pursuit-evasion strategies in a virtual-reality environment created using the Unreal World Editor. The Unreal World Editor (UnrealED), originally designed for use with thep opular 3D game Unreal, is an easily available editor which can be used for the creation and modification of a wide variety of immersive environments. In this paper we model buildings on the University of Limerick campus, and their associated exteriors. This paper makes use of an extension tot he original Unreal game engine called a mutator, more specifically the Gamebots mutator designed and released by the University of Southern California's Information Sciences Institute. This extension allows characters in the game to be controlled via network sockets connected to other programs. The game feeds sensory information to the character over the network connection. Based on this information, the client program can decide whata ctions the being should take and issues commands back over the network to the game, in order to control the actions of the entity. The client program incorporates a genetic algorithm to control the two individuals involved.

Keywords: **Evolutionary Computation, Multi-Agent systems, VirtualR eality, Machine Learning**

I. INTRODUCTION

Recently there has been an upsurgeo f interest in the use of immersive 3-dimensional environments, originally designed for games, for serious scientific research. This is perhaps not surprising given that some of the best virtual reality simulations around are available using these engines without resource to expensive specialist hardware.[5,6] Also a world editor may bep rovided allowing for the creation of custom virtual environments tailored to the particular research being conducted.

The UnrealEd 2.0 world editor allowst he creation of an Unreal world using a GUI interface from which rooms and landscapes can be modeled. The world that the server provides in the game is created using the editor. This is the editor supplied with the Unreal Tournament (a later version of the original Unreal) game and allowst he creation of the world by carving the shapes necessary from a larger building block, manipulating them to create the elements required. (See Figure 1) An extension of the Unreal game engine, called the Gamebots mutator, developed at the University of Southern California's Information Sciences Institute [1] is also used in this research. This extension allows the control of characters by other programs, as described in the section II.

We describe a set of preliminary experiments in the evolution of robot navigation based on a simulated three-dimensional virtual environment. Criticisms of some research into robot navigation in simulated environments stem from the perceived lack of complexity of the simulated environmenta s opposed to the real world, with the corresponding difficulty in transferring control algorithms derived in simulation into an embodied agent.[3,4] The approach presented in this paper has the advantage of being able to model a wide variety of virtual worlds using as much detail as is required, and also of incorporating features such as stairs and several floors of a building, which may be difficult to model using more conventional approaches.

The initial experiments described in this paper involve the evolution of basic obstacle avoidance and predator-prey behaviours. These experiments should be seen as preliminary and indicative of the potential for further interesting research using this general approach.

The building modeled in this paper is the Robert Schuman building on the University of Limerick campus, together with its immediate exterior. (See Figures 1 and 3) This is a three floorb uilding with a canteen, lecture theatres, and offices Another building on campus has also been modeled but is not used as part of the current work.

The floor plans were made available by the Buildings Department here at the University. A digital camera was also used tot ake multiple photographs of such details such as walls and floors throughout the building. This was necessary in order to be able to reproduce the textures realistically in them odeled world to provide for a fully immersive environment

II. The Client/Server Architecture

An extension of the Unreal game engine, called the Gamebots mutator [1] (not to be confused witht he mutation operator in evolutionary algorithms), developeda t the University of Southern California's Information Sciences Institute is used in this research. This extension allows the control of characters by other programs using a client/server architecture. The server is running a 'game' and using the mutator to send telemetry about the game to a port on the machine.

As part of this work we have written a client which connects to this port and interacts with the server by exchanging messages. This client is written in C++ and is uses Windows sockets to connect to the server. It comprises a number of functions necessary to communicate with the server. These include a parsing function that takes the server messages and extracts the relevant data to be passed to the genetic algorithm, and a corresponding function, which communicates the solution, generated by the genetic algorithm back to the server. The clienti s also responsible for the connection to the server, through the use of sockets. The client creates twos ockets to communicate with the server, one for each bot to be controlled and sends two initiation messages to the server to initiate the predator and thep rey in thew orld. The word 'bot' is derived from the word robot, and means any non-human controlled entity inhabiting the world.

The server initiates the game for the client to connect to and provides the interface to the world the bots inhabit, in the form of messages. The server sends messages about the game state in the world to the client both synchronously and asynchronously. Synchronous messages begin with a BEG command and end with an END command and represent the state of the world at that point in time. This includes a list ofi tems within the agents view at the moment, including items and players, together with the agents own state in terms of location and health. By default synchronous data is sent about ten times a second. The only synchronous information extracted for the current researchi s the bots' location. Asynchronous data include sensor updates, such as beginning to fall over a precipice, hearing a sound, or bumping into a wall. Asychronous data is not used in the current research. Bot action commands include those to rotate, to move to a location (RUNTO), to jump, or to shoot a weapon. The RUNTO command is the only one utilised in the current research. The game engine or server is contained in the Unreal Tournament game, along with the world editor for the game.

III. Pursuit-Evasion Contests

Given the prevalence of Pursuit-Evasion contests in animal behavior, their difficulty in analysis, and their applications in robotics and virtual environments, they form a test-bed of considerable scientific interest. Indeed many tasks performed by autonomous mobile robots, including obstacle avoidance and navigation may be seen as degenerate cases of pursuit-evasion contests [2].

In this paper we present initial work on the implementation of thesec ontests in combination with basic obstacle avoidance using the Unreal virtual environment (see Fig. 4). Thew ay in which the bots navigate the world is through a system of connected points known as navigation or "nav" points placed uniformly throughoutt he level. The predator and prey, which inhabit the world begin at one of several randomly defined nav points. We can view the generation and placemento f these navigation points as a method of reducing the dimensionality of the overall searchs pace by assuming a Voroni-type diagram of points, as opposed to an unconstrained search space.

The chromosome for the genetic algorithm consists of a vector comprisingt won avigationp oints int he virtual world. The total numbero fn avigation points in the simulated University is approximately 1000; these include points on the interior and exterior of the building, and include points on different floors of the building. When the initial population is created, each individual is randomly assigned two points, each point being one of the navigation points in the world. Each candidate thus has two points and represents a vector or path from the first point to the second in the world. This path may, or may not bisect an obstacle.

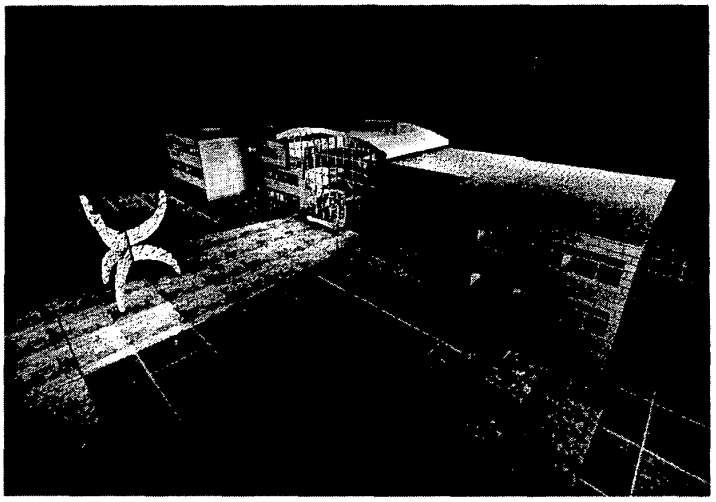

Fig. 1. Top down view of the University of Limerick Schuman building as modeled using the Unreal World Editor The objecti n the foreground is a stainless steel sculpture (also modeled).

IV. Evolutionary Algorithm Outline

Here we outline the evolutionary algorithm currently being used to control the movement of the bots through the simulated environment. Rather than evolve a specific control architecture we attempt to directly evolve paths dynamically through the environment, avoiding obstacles and either evading (prey) or seeking out(predator) the other bot. It should also be noted that this algorithm is stillc urrently at a developmental stage for reasons outlined in the conclusions section, however interesting behaviours are still seen to evolve.

The genetic algorithm (GA) used in evolving the behaviour of the predator and the prey in the scenario has a number of modules. The GA operates in the same manner for both, except for the differing implementation of the fitness functions for the two entities.

The individual functions of the GA are as follows:

Populate – creates the initial population of 100 individuals;

InitFitness – sets the fitness of the individuals initially;

Reset – resets the number of generations to zero for the next run;

Test – the main body of the GA, from which the other modules are invoked;

Prey_Fitness – assigns fitness to individual prey

Pred_Fitness – assigns fitness to individual predators

Roulette – selects two individuals from the population for crossover;

Mutate – mutates an individual, probability 0.1;

Crossover – generates two child vectors from their parents;

Output – sends the GA's generated solution to the server;

GoodPath – reinforces valid paths that have been found in the world

The Populate, InitFitness and Reset functions are self-explanatory, when called they are passed an identifier for either the predator or prey and take the relevant action. InitFitnesss etst he fitness for all candidates in thei nitial population to thes tarting value. TheT est part of the GA controls the execution of the GA. If the generation is 0, then a population is created comprising vectorsc hosen at random from the total search space, and fitness assigned to each of the individuals by calling the appropriate functions outlined above. The GA then proceeds to execute for a number of generations (typically 100). During a generation, the fitness functionf or the particular entity (predator or prey) is called for each individual and fitness is assigned. For each generation, the best fitness and the average fitness are recorded for statistical purposes. When all of the individualsi n a generation have been assigned a fitness, the generation is ranked with the individual with the highest fitness being placed at the top.

The next generation is then created. Firstly the top 50 individuals from the current generation are placed in

the next generation. The remainder of the new population is created using crossover and mutation, generating new individuals until the generation has been filled. The crossover operation is carried out using two individuals selected by theR oulette function and manipulating them to create two new valid individuals and has already been described. The best solution from the generation is then selected, and the generation is then iterated to begin the process again with the new population. The Roulette function is called twice within the test function to return twop arents to be manipulated. It works by generating a random number between zero and the total fitness count for a generation, then proceeding to iterate through the individuals adding their fitness. When the fitness so far being calculated reaches the random number already generated, the individual at thisp oint is returned. The two parents returned by the successive execution of this function are the used in the crossover operation. The output function takes the first point oft he solution and tells the relevant entity (predator or prey) to go to it. The function then waits for a certain number of time-steps in order to give enough time for the location to be reached. Theo utput function then calls the Goodpath function described below.

Crossover in the genetic algorithm is implemented using the function in Equation 1 for each of the X,Y and Z co-ordinates for each of the Parent points. For example, two vectors are taken in this case Parent1 and Parent2, both of which have a pair of points associated with them representedb y Point A and Point B (see *Figure 2). A child vector is* created by using the above function between Point A of Parent1 and Point A of Parent2, the same is done for Point B with both parents. The resulting points represent a vector randomly placed between the parents. (Rnd is a value randomly chosen between between 1 and 10).

$$Child = \left(Parent1 + \left(\frac{Parent2 - Parent1}{Rnd} \right) \right) \quad (1)$$

The resulting Child Point A and Child Point B, will not exactly correspond to points in the map, so the next stage is to select the closestp oints to them. The closest nav points in the world are selected and used to create the vector as illustrated in Figure 4.

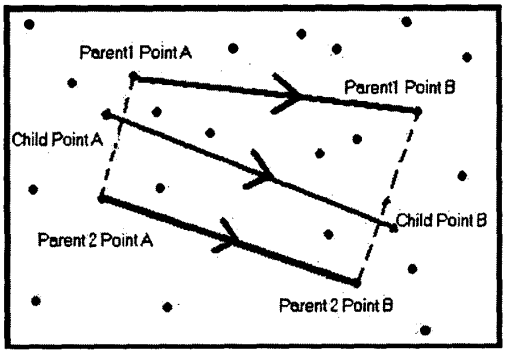

Fig. 2. The crossover mechanism employed

Fig. 3. The view looking inside the Schuman building. Note the stairs leading to the firsta nd second floors (also modeled) – bots can travel all around the floors of the building and itse xterior. The detail on the circular window on the left was supplied by digitalp hotography

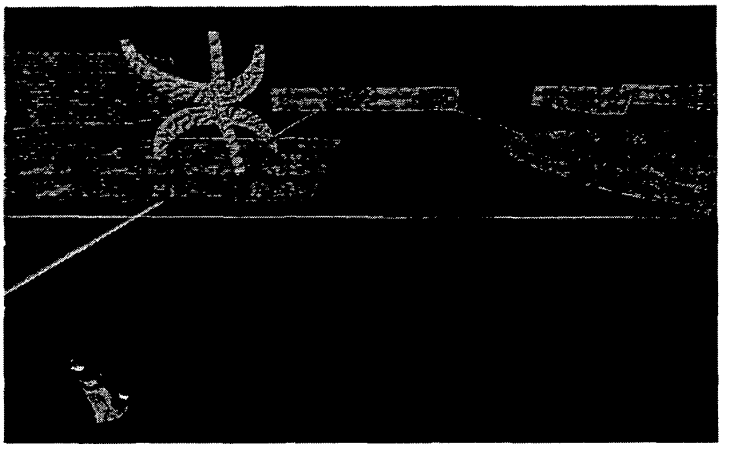

Fig. 4. Predator(right) bot chasing the prey(left) bot on the exterior of the Schuman building. The predator is in the pool which is left unfilled for testing purposes. In the background are artificial obstacles, also created for testing purposes. In general the detailo f the modeled world is not as high as in the previous screenshots in order to be able to view the behaviours in real time.

The Goodpath function worksi n conjunction with the output function and the second point of thes olution. Firstly the function checks to see if the both as reached the firstp ointi n the solution, which itw as told to go to in the output function. If the bot is at thatl ocation, then the Goodpath function tells the bot to go to the second point in the solution and then waits for an umber of timestepsi n order to give it time to be reached. The function then checks to see if the botm anaged to reach the second point in the solution. If so, then this chromosome first point tos econd point represents a good vector or path in the world. This vector is then copied five times into the next population thus reinforcing this path. This should encourage obstacle avoidance behaviour for this individual for future generations.

The fitness functions for the predator and prey are similar, butn ot the same due to the differing objectives of each. Thef itness of thep redator is calculated as follows. Firstly a check is made that the individual being evaluated is not the predator's current position, that the positions are not the same. If they are the same, then that individual is assigned a fitness of zero, effectively removing it from being considered as a solution. Next the GA calculates the Euclidian distance from the predator's currentl ocation to the firstp ointo f the individual under evaluation. For predator at location (x1,y1,z1) and individual under evaluation at position (x2,y2,z2),

$$\text{Distance}=\sqrt{(x2-x1)^2+(y2-y1)^2+(z2-z1)^2} \quad (2)$$

Similarly the distance between the first test point and the prey is calculated . Finally the distance between the first and second pointsi n the test vector ise valuated. In the simplest form oft he fitness function these three values are summed; lower values resulting in higher fitness. This reinforces short length vectors which have less chance of passing through obstacles, and those which initially bring the predatorc loser to the prey. It also reinforces vectors whosei nitial point is close to the predator reducing the chance of intervening obstacles. For the prey the fitness of an individual vector is calculated in the same fashion, with the distance between the test point and the predator being subtracted rather than added to the final fitness; again smaller total values resulting in higher fitness.

It shouldb e notedt hat these are preliminary fitness functionst hat are expected to be improved over time, however even with these simple functions interesting obstacle avoidance and predator-prey behaviours are seen to evolve. It is possible to view the evolution of these behaviours in real time from the predator or prey's perspective, or as an impartial observer from any point in the 3-D environment, at ground level or above.

V. PRELIMINARY CONCLUSIONS AND FUTURE WORK

We would view the contribution of this paper as threefold. Firstly in the use of a world editor for the creation of a realistic simulatede nvironment for the testing, and possible benchmarking of different robot control and navigation strategies. These editors have mainly been used to date for game applications. Associated with the use of the world editor for environment design is the use of a game engine for handling detailso f the physicsa nd graphicsi nvolved, leaving the researcher free to concentrate on the navigation and control aspects. It should also be possible in theory, although not touched on in this paper, to derive detailed visual data from the environment as the bots move around. This data can be very realistic if generated by digital imaging of real world environments and incorporated into texture maps.

While never replacing true embodied experimentation with real robots we suggest that our approach of modeling 3-dimensional worlds in some detail to serve as test-beds for the development of navigation and intelligentc ontrol strategies may serve as useful middle ground approach between real world experimentation and less complex simulations.

Secondly, the use of the gamebots mutator for the control of the simulated agents, or 'bots' in a predator-prey scenario. Of course, in degenerate cases this these tasks revert to simpler navigational problems. We also remember that the simulated environment is inherently three-dimensional as opposed to more conventional two-dimensional testbeds. This leads the way for a wide variety of possible tests cenarios, again leading to possible benchmarking applications, as mentioned above.

Thirdly, the introduction of a novel, and as yet unproven, approach to dynamic obstacle avoidance and goal seeking behaviour (although initial results are encouraging). Rather than evolve and test individual controla rchitectures, typically done by the evolution of the weights or some structural aspects of a neural network architecture, we instead attempt to directly evolve paths through the unknown environment based on joined-up vectors evolved using simple criteria. This approach shouldh ave the ability toa dapt toa changing environment (which can also be modeled using the world editor).

Of course there are some caveats to this work. There presently appear unfortunately to be some problems with the Gamebots mutator software; also it is not always clearw hat the response will be from the server in response to different combinations of input. This has slowed down detailed production of results for this paper, however those that have been obtained are encouraging, withb oth predator andp rey displaying interesting goal-directed behaviours.

Also, game engines, while extremely powerful tools, do not necessarily accurately replicate physics in terms of turbulence, friction and so on. This may become more important for future research and could perhaps be addressed by a new generation of engines, based on those existing, specifically tailored for research work.

Finally the approach we have taken in using evolutionary algorithms for dynamic path selection is, as yet, unproven and may prove inferior tod irectly evolving control architectures. However we do not see this as a majori ssue as ourm ain purpose in this paper is to suggest, if you like, a half-way-house between simulated experimentation in unrealistic environments, and full-scale embodied experimentation. We dub this field Immersive Robotics (or 'Immerbotics'). We considert hat, as the thrust increases for more complex and intelligent mobile robots, up to humanoid level, and even beyond, because of the potential prohibitive testing costs that may be involved, this field will become increasingly important and relevant.

VI. REFERENCES

1. Adobbati et al. Gamebots: A 3D virtual world test bed for multiagent research. In *Proceedings of the Second International Workshop on Infrastructure for Agents, MAS, and Scalable MAS*, Montral2 001.
2. Cliff D., and Miller G.F. Co-Evolution of Pursuit and Evasion I: Biological and Game-Theoretic Foundations", *Technical Report CSRP311*, University ofS ussex, School of Cognitive and Computing Sciences, August 1994.
3. Eaton M., Collins J.J.,Sheehan L., Towards a Common Set of Experimental Frameworks for the Evaluation an Benchmarking of Mobile Robot Control Architectures. In Sugisaka M. and Tanaka H. (eds) *Proceedings of the 5th International Symposium on Artificial Life and Robotics.*, 305-308, 2000.
4. Floreano D., Mondada F, Hardware Solutions for Evolutionary Robotics. In *Evolutionary Robotics, Proceedings of the first European Workshop*, Lecture Notes in Computer Sciencel 468, Springer-Verlag, 138-151, 1998.
5. Laird J.E,. Research in human-level AI using computer games. *Communications of the ACM* Jan 2002, Vol 45, No. 1 pp. 32-35
6. Lewis M and Jacobson J. Game engines in scientific research. *Communications of the ACM* Jan 2002, Vol 45, No. 1 pp. 27-31