Evolving Predator and Prey Behaviours with Co-evolution using Genetic Programming and Decision Trees

Tiago Francisco
School of Technology and Management
Polytechnic Institute of Leiria, Portugal
tiago3f@gmail.com

Gustavo Miguel Jorge dos Reis
School of Technology and Management
Polytechnic Institute of Leiria, Portugal
gustavo.reis@estg.ipleiria.pt

ABSTRACT

Developing artificial behaviours to control artificial creatures or vehicles is a task that can be solved by means of Evolutionary Algorithms. The *Predator and Prey* is a problem where it is possible to evolve behaviours for both predator and prey, using artificial co-evolution: the predator must capture the prey and the prey must evade the predator. Both predator and prey have also different characteristics, the predator is faster and more agile and the prey is slower. This paper presents an alternative, using Genetic Programming with Decision Trees for evolving both Predator and Prey behaviours. The results obtained shows the feasibility of the approach.

Categories and Subject Descriptors

I.2.4 [**Artificial Intelligence**]: General; D.2.m [**Software Engineering**]: Miscellaneous

General Terms

Algorithms

1. INTRODUCTION

Developing artificial behaviours to control an artificial creature or a vehicle - in this case a spaceship - can be difficult, especially if the vehicle's or creature's characteristics differ from one another [1]. Developing those behaviours by hand is almost an impossible task [1] and one of the most common alternatives is the use of the so called Steering Behaviours [2] which reproduce results that mimic real life: seeking, fleeing, pursuit, evasion, flocking among others. It is also possible to *hard code* the behaviours but this leads to loss of abstraction: a behaviour must be abstract and must be prepared to all sorts of conditions. There are also evolutionary computation approaches to these kind of problems [3, 4, 5] but must of them result in a mathematical equation that produces a real-life result.

Our proposed approach uses Genetic Programming to generate a Decision Tree, which represents the behaviour that mimics the results from the above solutions, but only using "functions" to which a normal human player has access too. For instance, instead of trying to create a mathematical equation, the result will be a sort of decision-tree, which after analyzing the surrounding environment of the vehicle acts accordingly to achieve its goal.We also make use of sensors that will allow an agent to "see" its environment

This paper addresses the evolution of two distinct behaviours:

Predator Track down its prey and capture it. Capturing, or destroying in this case, the prey involves shooting a projectile at the prey. The predator can't waste ammunition, so the one with the better accuracy is considered ideal.

Prey Must flee its aggressor in any way possible.

In order to achieve a real-life behaviour and a behaviour that mimics the results from the previously mentioned solutions we used co-evolution - evolution of two different species (preys and predators) living on the same environment and competing with each-other. By using co-evolution we were expecting to evolve both predator and prey behaviours that were able to achieve their goal, no matter their position in the environment.

1.1 Game Bellerophon

The basis for this paper is a game we have developed. Bellerophon is space sim [1] game, in which the player can do almost anything: buy or trade products between space stations, hunt down outlaws, hunt and "steal" merchants or just wander around the universe.

1.2 Vehicles

The vehicles used in this paper are spaceships, from the game Bellerophon. As any other vehicle it can move on its environment using its engine as mean of locomotion. It can move forward and stop its motion by increasing or decreasing its speed. The controller can also stop the ship whenever he wants by reducing the speed until it reaches 0. The ship can also turn left and right, although it has to be moving forward to achieve a "turning" effect, unlike a car for instance.

[1]Space Simulator software simulates outer space and space flight with a wide variety of applications

This means is that in order to recreate a "space-physics-system" the controller of the ship must be applying thrust to his ship. If not the ship will only rotate around its own center point. The ship as also several characteristics: maximum speed, acceleration, turning speed, turning increment, which differ from ship to ship. Therefore they are different for predator and prey.

Ships also have a radar that gives the ship's controller information about his environment and his target/aggressor. The radar allows the controller of the ship to know about its surroundings, giving sensor-information among other things. The radar can give information about the controllers target, in a way that the controller can know if the target is in front of him, behind him, to left or to the right, in his ship's weapon range and also if it is behind his target or if it is to the left or to the right of the target (meaning the controller can know if the target "sees" him to the left of right).

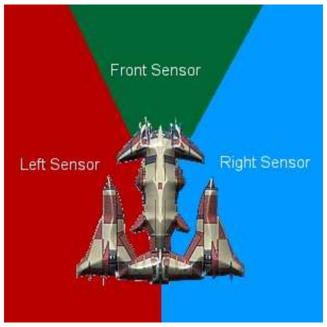

Figure 1: Example of sensor layout

We designed the radar in order to recreate a pilot's "field of vision", and where he can look. It also allows tracking of multiple targets - although not addressed in this paper - and quick adaptability to any changes in the environment. For instance, if one would add obstacles into the game field we could use this same sensors/radar to steer the ship around this obstacles. A radar also makes it easier to analyze results, since a human understands more easier logic (eg.: if-enemy-front) rather than numbers.

1.3 Objectives

As previously mentioned, the objective is to evolve two basic behaviours: predator and prey. These behaviours are intended to be used on the field of evolutionary combat algorithms and combat strategies, where the main focus will be evolving a decision tree to choose one of the possible different strategies [6]: attack, retreat, etc. The attack and retreat strategies will correspond to the behaviours predator and prey presented in this paper. The attack strategy will pursue a target and attempt to destroy it being faster and more agile that it's "prey" and other behaviour will try to evade its aggressor being slower.

2. PROPOSED APPROACH

Initially the test was intended to have more generations, but since the "evolution procedure" is in real-time 100 generations was the most time-efficient approach (around 33 minutes). In order for the prey to achieve its goal it must survive the entire run, or survive the longest time possible. On the other hand, the predator must capture the most number of preys possible while keeping its accuracy at the maximum level (shooting only when it is in weapon-range of it's prey). The accuracy included in the Predator's fitness function prevents it from being always shooting in hope of capturing a prey.

2.1 Function sets

The functions we used are intended to be easily understood by humans, and represent what a "human controller" would be able to do. For instance, we use "ifTargetLeft" if we want to know if our target is to our left, and "IfLeftTarget" to know if we are to the left of our target. Progn2 is a function that will execute both its terminals. For instance if Shoot and Turn Left are the terminals of a Progn2 function, both will be executed and the ship will fire and turn left

2.1.1 Predator

- Terminals:
  - Velocity Up
  - Velocity Down
  - Turn Right
  - Turn Left
  - Shoot
- Functions
  - ifTargetLeft
  - ifTargetRight
  - ifTargetFront
  - ifTargetWeaponRange
  - ifLeftTarget
  - ifRIghtTarget
  - ifBehindTarget
  - Progn2 (Executes both terminals)

2.1.2 Prey

- Terminals
  - Velocity Up
  - Velocity Down
  - Turn Right
  - Turn Left
- Functions:
  - ifRangeAggressor (danger zone)
  - ifWeaponRangeAggressor
  - ifAggressorRight
  - ifAggressorLeft
  - ifAggressorBehind
  - ifLeftAggressor
  - ifRightAgresssor
  - ifAggressorFieldOfView
  - Progn2 (Executes both terminals)

2.2 Fitness Functions

2.2.1 Predator

The fitness function of the predator is done by the Equation 1, where $NoPT$ stands for Number of Potential Targets, NoK for Number of Kills, SF for shots fired and $\frac{1}{D}$ for Distance.

$$Fitness_{Predator} = NoPT - NoK + SF + \frac{1}{D} + Penalty \quad (1)$$

The shots fired (SF) is the number of shots fired when the target is not on the ship's weapon range. If the controller is accurate this value will be 0. If the Predator doesn't kill (low number of kills) anyone or it doesn't move (low distance) it is heavy penalized.

2.2.2 Prey

The fitness function of the prey is done by the Equation 2, where $ToRSA$ stands for Time of Run that the Ship Was Alive D stands for Distance. RTD stands for Round Time Duration, this is normally a constant

$$Fitness_{Prey} = RTD - ToRSA + \frac{1}{D} + Penalty \quad (2)$$

If the Prey does not move or does not turn at least once, it is heavy penalized.

3. EXPERIMENTS AND RESULTS

The environment is a "system" of the game's universe. It is composed of 5000 by 5000 pixels and can have an unlimited number of spaceships. In order to promote competition between predators and insure only the best "reproduce", the configuration presented in Table 1 was used.

Table 1: Configurations used to insure that only the best will reproduce

Number of predators	200
Number of preys	50
Predator ship	Speed 200 px/s Acceleration 10 px/s Turning Increment 9^o
Prey ship	Speed 100 px/s Acceleration 5 px/s Turning Increment 9^o

All the ships begin each run in a random position in a 2500 by 2500 pixels area. Preys are captured with only one shot. If the Predator captures a prey it randomly selects another from the ones that are alive.

Both behaviours will be discussed in this section.

3.1 Predator

The Predator behaviour is the one that represents the best behaviour of the two because it is the one that mimics real-life solutions with more precision. Various results can be considered human-competitive because they can compete with human produced behaviours and in some cases beat

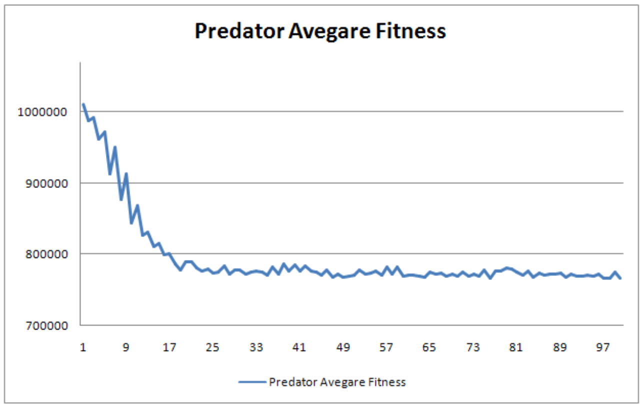

Figure 2: Predator Average fitness results

these behaviours. To test this claim, one of the most common results was pitted against a hard coded behaviour.

The result was very explicit. Both ships employed a very similar "capture" behaviour, the "circler" which will be described later on this paper. Also when compared to Steering Behaviours results[2][3], the best results obtained with the GP are very similar. Even the worst predators achieve the goal, which is pursue and capture its target. Also from time to time a totally different result appears, that some times defies common knowledge of pursuit, an example of this is a result that pursues his target while constantly rotating his ship. If the target is on the predator's field of view (in front of the ship), the behaviour will apply some forward speed to the ship moving it closer to the target, but continuing to rotate.

3.1.1 "Circler"

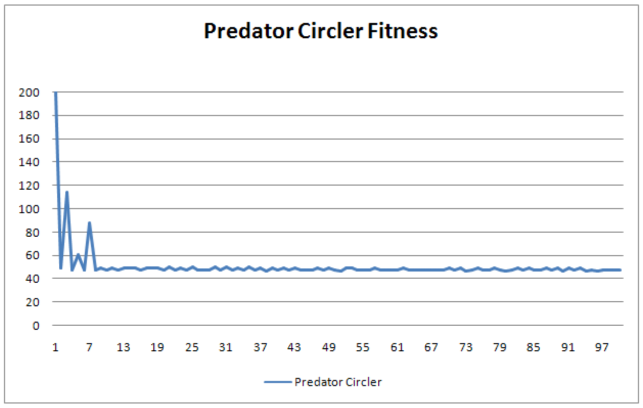

Figure 3: Predator "Circler" fitness results

This is the most common result, and the one that is similar to the hard coded behaviour. It also "mimics" the Steering Behaviour result in some way. The behaviour will "see" if it's target is in front of it's ship. If it is, it will move the ship towards the target, turning it accordingly to the targets relative position (to the left or to the right) . If the target

[2]http://www.steeringbehaviours.de
[3]http://www.red3d.com/cwr/steer/

is not on the ships field of view, the behaviour will move the ship forward and turn the ship left or right, according to the targets relative position. The name “Circler” is from this “action”. Whenever the predator “overtakes” its target or the target is not in front of the ship the behaviour will turn around trying to put the target on its field of view.

Predator “Circler” decision tree:

```
(progn2 (if-target-left (if-target-weapon-range
     (progn2 (if-target-left (if-target-weapon-range
         shoot (progn2 (if-left-target turnright turnright)
         (if-target-left turnright turnright))) (progn2
         (if-left-target turnright turnright) velup)) (if-target-weapon-range
         (if-target-weapon-range (if-target-front (if-right-target
             turnright velup) (if-target-front velup
             veldown)) (if-target-weapon-range (if-target-weapon-range
             (if-behind-target veldown shoot) turnright)
             turnleft)) turnright)) turnleft) turnright) (progn2
     (if-target-weapon-range (progn2 (if-target-left
         (if-target-weapon-range (if-target-left turnleft
             turnleft) turnleft) (if-target-weapon-range shoot
         (if-right-target turnright velup))) (progn2 (if-target-left
         (if-target-weapon-range shoot (progn2 (if-left-target
             turnright turnright) (if-target-weapon-range shoot
             (progn2 (if-left-target turnright turnright) (progn2
                 velup turnleft))))) (progn2 (if-left-target turnright
         turnright) velup)) (if-target-left (progn2
         (if-left-target (if-left-target turnright turnright)
             turnright) velup) (if-left-target turnright turnright))))
         (progn2 (progn2 (if-target-left (if-target-weapon-range
             shoot turnleft) (progn2 (if-target-weapon-range
             (if-behind-target velup turnright) turnright) (if-left-target
             turnright (if-target-left turnleft turnleft))))
             (if-target-weapon-range (if-target-weapon-range (if-right-target
                 turnright turnleft) (if-left-target turnright turnright))
                 (if-target-weapon-range (if-target-front velup
                     veldown) (if-target-left turnright turnright))))
             (if-left-target velup turnleft))) velup))
```

3.2 Prey

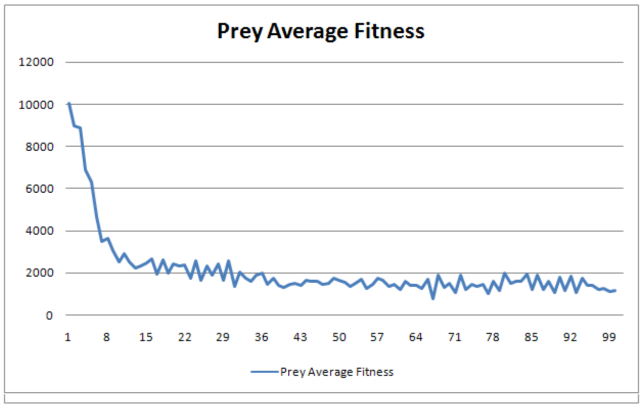

Figure 4: Prey average fitness results

The Prey Behaviour in comparison to the complexity and good real-life comparable results of the Predator, is less complex.

Most of the results involve either, turning the ship into one direction and causing the predator to follow, producing a circular movement for both or moving the opposite direction of the predator when the predator overtakes the prey. The only way to “evade” his aggressor is to let it “pass”, or as some results imply just enter into a circular motion.

3.2.1 “Circler”

The circler was the most common behaviour for the prey. Like the predator circler result, this one involves turning the ship while accelerating. This creates a “donut” like effect, thus the name “circler”, which makes the predator follow. If the predator doesn’t follow the prey will start moving for-

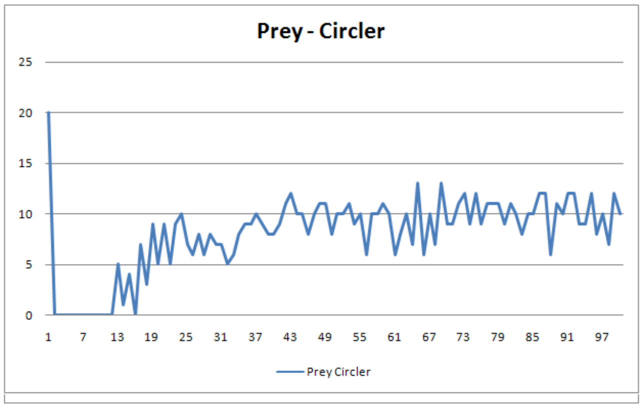

Figure 5: Prey “Circler” fitness results

ward, putting distance between it and its predator. If the predator continues to follow, the prey will continue the circular motion. One could say that the prey is always trying to put as much distance between itself and its predator, and since the predator is faster and always chasing it the best behaviour is to “circle”

Prey “Circler” decision tree:

```
(progn2 (progn2 (if-aggressor-behind turnleft
    (if-aggressor-left (progn2 (progn2 (progn2
        (if-aggressor-behind turnleft (if-aggressor-left
            (progn2 (if-aggressor-behind turnleft (progn2
                (if-aggressor-left velup turnleft) (if-aggressor-fov
                turnright turnright))) (if-aggressor-left velup
                turnleft)) turnleft)) (if-aggressor-behind velup
        turnright)) (if-aggressor-fov (progn2
        (if-left-aggressor turnleft (progn2 turnleft
            (if-aggressor-left velup turnleft))) veldown)
        velup)) (if-aggressor-left velup turnleft))
        turnleft)) turnleft) velup)
```

3.3 Original Experiment Variation

We decided to redo the experiment with a more classical approach, a predator that only pursued its target and a prey that only fled its aggressor. For this new “experiment” we developed new fitness functions for both species, but kept the rest of the experiment intact. This allowed us to compare the resulting behaviours of each experiment.

3.4 Fitness Functions

3.4.1 Predator

In this variation, the predator must keep the prey in its weapon range for the maximum time possible. RTD stands for Round Time Duration, $PTiWR$ stands for Prey Time in Weapon Range and D stands for distance

$$Fitness_{Predator} = RTD - PTiWR + \frac{1}{D} + Penalty \quad (3)$$

If the predator doesn’t move its heavyly penalized

3.4.2 Prey

In this variation, the prey must keep away from the predators weapon range for the maximum time possible. RTD stands for Round Time Duration, $PToWR$ stands for Prey Time out of Weapon Range and D stands for distance

$$Fitness_{Prey} = RTD - PToWR + \frac{1}{D} + Penalty \quad (4)$$

If the Prey does not move or does not turn at least once, it is heavy penalized.

3.5 Predator

Again we saw that the evolution process created capable predators. The "circler" result appeared once more, suggesting this result is perhaps the best for this problem-environment combination.

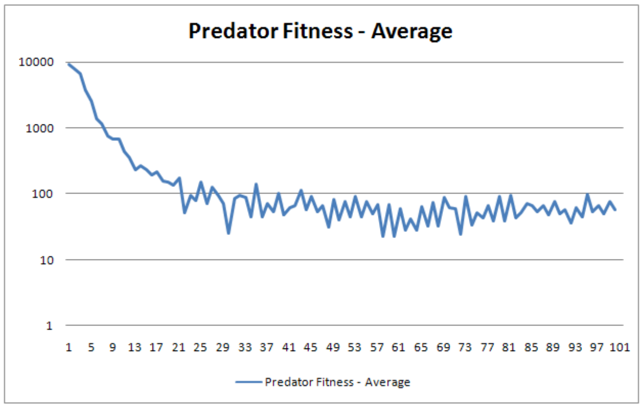

Figure 6: Classic Predator fitness results

Classic Predator "Circler" decision tree:

```
(progn2 (if-target-left (if-target-left (progn2
    (progn2 shoot turnleft) (if-target-front
    velup turnleft)) (progn2 velup turnright)) (progn2
    velup turnright)) (progn2 (if-target-left velup
    (if-target-left turnright turnleft)) (progn2
    (if-target-left (if-target-right (progn2 velup
        turnright) velup) (if-target-left (if-left-target
        turnleft (if-target-left (progn2 (if-target-front
        (if-right-target velup (if-target-front veldown
            (if-target-left shoot (if-right-target shoot
                velup)))) turnleft) (if-target-left turnleft
        velup)) (progn2 velup turnright))) (progn2
        velup turnright))) turnright)))
```

3.6 Prey

The prey was put under enormous pressure with this variation. Each prey was normally up against 2 or more predators since there are 200 predators for 50 preys. The results however were surprisingly good. Again, like the predator, we saw the circler result, but not as often. A new pattern appeared, the "Fly-By" behaviour.

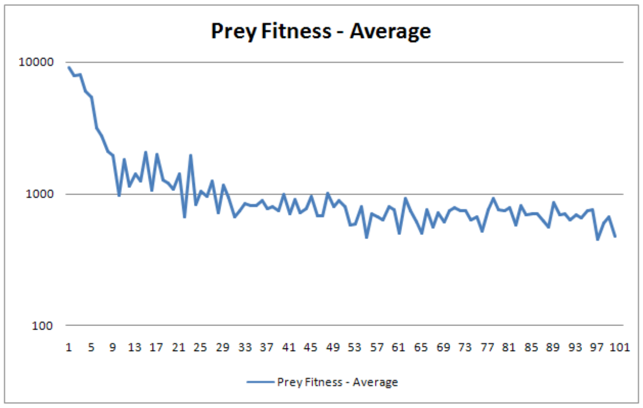

Figure 7: Classic Prey fitness results

3.6.1 "Fly-By"

This behaviour mimics the one seen with Steering Behaviours. It lets the predator get close, and when the predator overtakes the prey it changes direction, normally selecting the direction opposite to the one the predator attacked. Although it lets the predator get close, changing its direction after the "overtake" makes the predator restart its attack run, gaining some time to escape.

Classic Prey "Circler" decision tree:

```
(progn2 (if-aggressor-right turnleft (if-aggressor-behind
    (if-aggressor-fov (if-right-aggressor
        turnright (progn2 (if-aggressor-weapon-range
        turnright velup) (if-aggressor-fov (if-right-aggressor
        turnright (progn2 (if-aggressor-weapon-range
        turnright velup) (if-aggressor-right (if-aggressor-right
        (if-aggressor-range-danger velup velup) veldown)
        (if-aggressor-left (if-left-aggressor turnright
            turnleft) (if-left-aggressor turnright (if-aggressor-left
            velup veldown)))))) velup))) (progn2 (if-aggressor-right
        turnleft (if-aggressor-behind (if-aggressor-fov
        (if-left-aggressor turnright turnleft) velup)
        (if-left-aggressor turnright turnleft))) velup))
    (if-aggressor-right turnleft (if-aggressor-behind
        (progn2 (if-aggressor-right turnleft (if-aggressor-behind
            (if-aggressor-fov (if-right-aggressor
                turnright (if-right-aggressor turnright (if-left-aggressor
                (if-right-aggressor turnleft (if-right-aggressor
                    turnright (if-aggressor-left turnright turnleft)))
                velup))) velup) (if-aggressor-behind (if-aggressor-fov
            (if-left-aggressor (if-aggressor-left turnright
                turnleft) velup) velup) (if-left-aggressor
            turnright turnleft)))) velup) (if-right-aggressor
        turnright (progn2 (if-left-aggressor turnright
        turnleft) (if-aggressor-right (if-aggressor-range-danger
        velup velup) (if-right-aggressor turnright
        (if-aggressor-fov velup turnleft))))))))
    velup)
```

4. HUMAN COMPERABLE RESULTS

In order to test the human competitiveness of the results of both experiments, we pitted the behaviours against a hardcoded opposite, for instance a Evolved Predator against a human coded prey, and see it if could capture it.

For this test, we used the best behaviours from both experiments. We used hard coded predator and prey behaviours. These behaviours are very much like the "Circler" behaviours that were evolved.

4.1 Main Experiment Human Comparable Results

To consider a behaviour a human comparable result, the evolved predator had to beat the average damage the human coded predator inflicted on the human coded prey over 10 rounds. The opposite was set for the prey; the evolved prey had to receive less average damage than its human counterpart against the human coded predator over 10 rounds.

Both evolved behaviours outperformed their human counterparts. The most prominent was the predator, that inflicted 2,83 times the damage inflicted by the human coded predator. As for the prey, the evolved prey outperformed its human analogue, receiving 10% less damage.

Just by analyzing the results, one can say that our results are human competitive, since they beat the human coded algorithms by some margin.

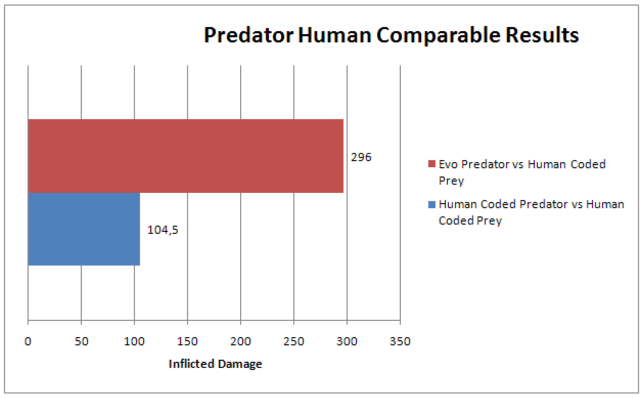

Figure 8: Predator human competitiveness

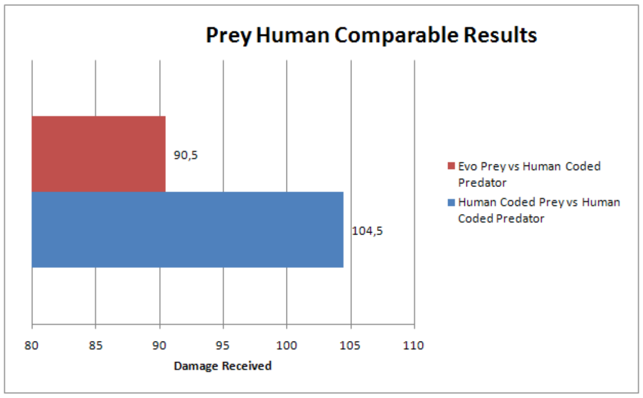

Figure 9: Prey human competitiveness

4.2 Variant Experiment Human Comparable Results

For this new human competitiveness test, we changed the human coded predator so that it could be compared to the evolved procedure. The change was simple; we simply removed its ability to shoot, since the "goal" was to keep it's the prey in its weapon range.

Like before, the evolved results beat the human coded algorithms. Both "classic" evolved prey and predator were able to best their human counterparts by a valid margin.

Again we can state that the "classic" evolved results are human competitive.

5. CONCLUSION

It can be concluded that it is possible to achieve human-competitive results using genetic programming to evolve predator and prey behaviours, and also that it is possible to do this using only actions and functions available to a player. Although the results for the prey behaviour are not as good as the predator results, one can say that in real life preys normally don't escape good predators. This paper also shows that genetic programming can help reduce the development of behaviours for various games or similar problems. Unexpected behaviours can be obtained, and while a human wouldn't normally develop a similar behaviour, it achieves it's goal.

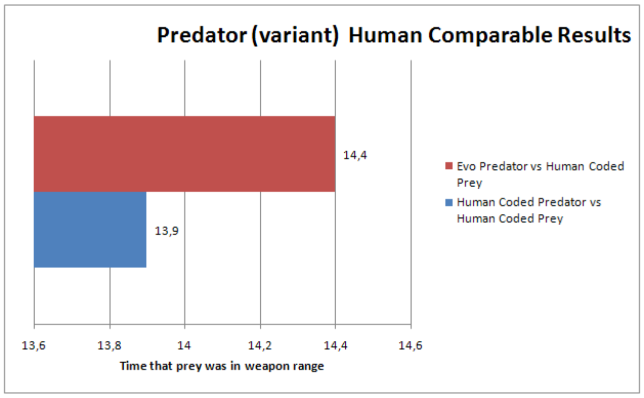

Figure 10: Predator human competitiveness – Variant Experiment

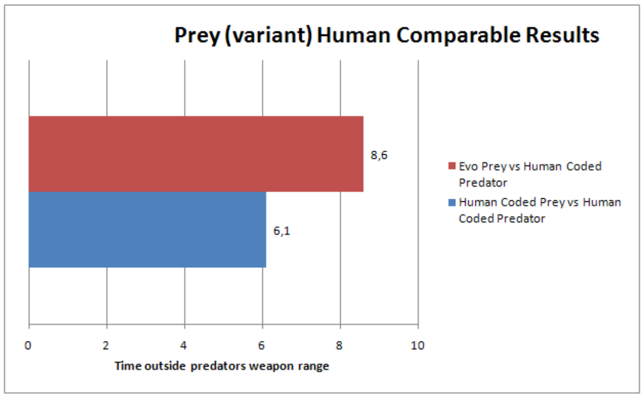

Figure 11: Prey human competitiveness – Variant Experiment

The results have shown that this approach can also be used for something completely different, for instance: a car racing game or a demonstration of flocking behaviours. To "solve" these new problems we only need to change the problem environment and the fitness function, while maintaining the terminals and functions. It is also important to note that the behaviours that result from this paper can be, and have been, used in another problem involving combat strategies[6].

We also showed that it is possible to use genetic programming to create human competitive results, that in some cases beat the human coded algorithms.

6. FUTURE WORK

We would like, in the near future, to evolve predator and prey behaviours with collisions. This would create behaviours that would still chase and flee its target/aggressor but at the same time avoid hitting other ships. This could be used, for instance, to evolve a police car and a robbers car where the police car chases the robbers in a busy highway We also want to port this study to a 3d environment, keeping the spirit of this paper on the new study. For that we would use the same ship design, and same function-sets.

7. REFERENCES

[1] C. W. Reynolds. Evolution of corridor following behavior in a noisy world. In *SAB94: Proceedings of the third international conference on Simulation of adaptive behavior : from animals to animats 3*, pages 402–410, Cambridge, MA, USA, 1994. MIT Press.

[2] C. W. Reynolds. Steering behaviors for autonomous characters. *1999 Game Developers Conference*, 1999.

[3] F. W. Moore and O. N. Garcia. A genetic programming approach to strategy optimization in the extended two-dimensional pursuer/evader problem. In J. R. Koza, K. Deb, M. Dorigo, D. B. Fogel, M. Garzon, H. Iba, and R. L. Riolo, editors, *Genetic Programming 1997: Proceedings of the Second Annual Conference*, pages 249–254, Stanford University, CA, USA, 13-16 1997. Morgan Kaufmann.

[4] C. W. Reynolds. Competition, coevolution and the game of tag.

[5] C. W. Reynolds. An evolved, vision-based behavioral model of coordinated group motion. In *Proceedings of the second international conference on From animals to animats 2 : simulation of adaptive behavior*, pages 384–392, Cambridge, MA, USA, 1993. MIT Press.

[6] T. Francisco and G. Reis. Evolving combat algorithms to control space ships in a 2d space simulation game with co-evolution using genetic programming and decision trees. In *GECCO '08: Proceedings of the 2008 GECCO conference companion on Genetic and evolutionary computation*, New York, NY, USA, 2008. ACM Press.