Evolved Communication Strategies and Emergent Behaviour of Multi-Agents in Pursuit Domains

Gina Grossi
Department of Computer Science
Brock University
St. Catharines, Ontario
Email: gina.grossi4@gmail.com

Brian Ross
Department of Computer Science
Brock University
St. Catharines, Ontario
Email: bross@brocku.ca

***Abstract*—This study investigates how genetic programs can be effectively used in a multi-agent system to allow agents to learn to communicate. Using the pursuit domain and a co-operative learning strategy, communication protocols are compared as multiple predator agents learn the meaning of commands in order to achieve their common goal of first finding and then tracking prey. The outcome of this study reveals a general synchronization behaviour emerging from simple message passing among agents. An additional outcome shows a learned behaviour in the best result which resembles the behaviour of guards and reinforcements that can be found in popular stealth video games.**

***Keywords*—multi-agent system, pursuit domain, communication, co-operative learning, emergent behaviour, video games.**

I. Introduction

Evolving co-operating multi-agent teams is a difficult problem researched extensively over the years [1]. As seen in the work by Reverte et al [2], an appropriate test bed for multi-agent systems is the predator-prey pursuit problem (the pursuit domain). The pursuit domain contains multiple agents, known as "predators", who have the job of working together to chase and capture an agent, known as the "prey". The job of the prey is to evade the predators (not be captured). A typical scenario consists of an infinite, discrete world in the form of a toroidal grid containing 4 predators and 1 prey. Agents move sequentially and are not allowed to occupy the same cell [2]. Using several variants of the pursuit domain, early work by Haynes et al [3] and Denzinger and Fuchs [4] show that agents can successfully learn to co-ordinate movements.

Research has shown that communication is effective in multi-agent systems where little information about the environment is known. Using the pursuit domain, Iba [5] and Kam-Chuen and Giles [6] demonstrate that multiple predator agents can successfully learn to use a new simple command language in order to capture a prey agent. Iba [5] also shows that communication is effective in co-ordinating robot navigation. In earlier work, Yanco and Stein [7] develop an adaptive communication protocol for co-operating mobile robots.

Previous work using genetic programming (GP) shows that complex behaviour can emerge from simple interactions among agents. In their research, Tanev et al [8] demonstrate the emerging surrounding behaviour of agents developed from proximity defined interactions of predator agents in the pursuit domain. In other work, Zhang et al [9] explore the idea that realistic complex tasks require more than one type of emergent behaviour to solve a problem and implement a robust fitness measure ("fitness switching") to encourage the emergence of multiple behaviours.

In addition to the pursuit domain learning strategies in game environments have also been studied. Luke et al [10] present a competitive learning strategy using genetic programming to co-evolve agents that are members of a soccer team. Also in a simulated soccer game, Kou et al [11] use a predefined language to allow a "coach" agent to co-ordinate movement of "player" agents. In the game of Ms. PacMan, Ms. PacMan and ghost team controllers are co-evolved by Cardona et al [12] and GP is used by Alhejali and Lucas [13] to evolve Ms. PacMan behaviours using training camps.

This study continues the investigation of using GP to evolve emerged behaviours in a multi-agent system. Using the pursuit domain and a co-operative learning strategy, multiple predator agents are tasked to learn the meaning of a simple set of commands with the goal of first finding and then following a prey. Similar to Reverte et al [2] predator agents have little knowledge of the environment. They do not know the location of the prey (unless they are in field of view (FOV) of the prey) and they do not know the location of other predators but (similar to robot agents in Iba [5]) they do know the relative (nearest, second nearest and farthest) direction of other predator agents. Predator agents can send messages (that *may* contain prey information if they are in FOV of prey) to other predator agents at any time.

Different communication protocols are compared in two experiments, with each experiment defining a different type of movement for the prey. In the first experiment, *Prey Linear Movement*, all communication protocols are trained and tested in the pursuit domain environment. The prey, starting from a random position within its own start area, moves linearly (Up $\uparrow$) on the grid. In the second experiment, *Prey Random Movement*, the environment is the same however, the prey moves in a random pattern (Up, Down, Left or Right).

Results of this study show emergent behaviour in the top performing communication protocols in both experiments. Specifically, a synchronized alternating message sending pattern emerges from simple message passing among predator agents. In addition, the learned behaviour and collaboration of agents in the best result resembles the behaviour of guard and reinforcements that can be found in popular stealth video games (e.g. *Metal Gear Solid (MGS)* [14]).

The remainder of this paper is organized as follows. Section 2 introduces the problem, communication protocols, and environment. Section 3 describes the experiment details, and Section 4 follows with the results of the experiments.

Conclusions are drawn in Section 5.

II. Problem, Communication Protocols, and Environment

Multi-agent learning and communication has been studied for years. At first glance, it is sometimes difficult to sort out the differences between multi-agent learning, (co-operative vs competitive) and communication. This section provides a brief background of some of the research in multi-agent evolutionary learning and communication.

Pannait and Luke [1] provide a survey of multi-agent based research giving a good explanation of the differences between multi-agent learning strategies and communication strategies. They also show how co-operative and competitive strategies contribute to multi-agent learning. Multi-agent evolutionary learning strategies can involve one or more learning agents having individual (local) and/or common (global) fitness goals. The agents' learning can take place as part of a team or happen concurrently. Team based learning can be homogeneous or heterogeneous. Team members of a homogeneous team use the same learning algorithm to evolve, while team members of a heterogeneous team use their own learning algorithm to evolve. Concurrent learning can happen in a fully co-operative scenario, where agents work together to complete a goal, in a competitive scenario, where agents compete against each other (co-evolution) to achieve a goal, or in a hybrid version using a combination of co-operative and competitive learning.

Communication strategies [1] are divided into two parts, the actual language (commands) used to communicate and the communication channel. The language can be direct or indirect. Direct language uses hard-coded commands or a learned language. In the case of hard-coded commands, agents learn commands that have pre-defined meanings. In the case of a learned language, agents learn to associate meaning to commands through various trials. Indirect communication is inspired by insects use of pheromones and involves the implicit transfer of information from agent to agent through the modification of the environment.

The communication channel [1] involves the mechanism used to communicate commands. For direct communication, this can involve a central message board, such as a blackboard, in which all agents can read/write commands to a global post, or it can involve message passing, in which individual agents directly send and receive messages to/from each other. For indirect communication, the communication channel can involve leaving hints, footsteps or bread crumb trails.

A. Problem

The purpose of this research is to investigate how well genetic programs can influence learning using different types of communication via message passing. Predator agents are tasked to learn the meaning of a simple set of commands with the goal of first finding and then tracking prey. The solution in this study is achieved by implementing the following communication and learning strategies.

B. Learning Strategy

The learning strategy uses a fully co-operative implementation with a global fitness measure. The predator agents work together to complete their common goal of first finding and then following the prey (as closely as possible). The global fitness measure is a minimization function which calculates the total distance between all predator agents and the prey over a limited period. The motivation for this fitness function is to compare how different communication protocols perform in allowing the predators to track the prey's movement. Agents collaborate to minimize the global fitness value using a heterogeneous team based learning strategy such that each agent uses its own learning algorithm to evolve.

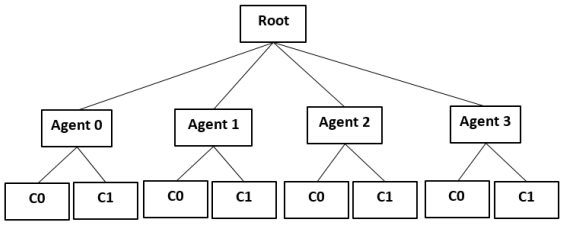

Fig. 1. Top-level GP Structure

C. Communication Strategy and Communication Channel

The communication strategy in this study uses a learned language. The language consists of two generic commands, *C0* and *C1*. The *C1* command, along with simple environment data, is sent from one agent to another through a message passing communication channel. An agent learns to associate a meaning to each command through evolving branches of its GP tree. Each agent has 2 child branches (command trees) where each branch is associated with one command (see Figure 1). The first command, *C0*, is evaluated when the agent has no messages in its message buffer. The second command, *C1*, is evaluated when the agent has at least one message.

The communication channel uses a message passing system in which each agent has its own message buffer queue. Upon evaluation of its command tree, an agent may send no messages, one message (with C1 command) or many messages (each with C1 command) to another agent. Once a message is sent it is placed in the receiving agent's message buffer. Before evaluation, if there is a message in the agent's buffer, the message is removed and a data variable for the agent is updated with the message data from the removed message. This data contains directional information about the prey if the sending agent was in FOV of they prey at the time the sending agent sent the message. A maximum number of 4 messages can be held in each message buffer. At the time of receiving a message, if an agent's message buffer is full then the first message (oldest message) in the buffer is removed and the newly received message is added to the end of the buffer.

D. Communication Protocols

A communication protocol defines the method by which a sending agent sends a message to a receiving agent's message buffer. Seven different types of message passing are examined including agents sending messages individually, as a leader or as a member of a team. Table I defines the communication protocols. In this table, Agents 1 to 4 are shown as A0, A1, A2 and A3. The description of how message passing is accomplished for each communication protocol is as follows. **Send22**: Two teams of two agents. A0 and A1 form one team and A2 and A3 form the other team. Each agent sends to its partner only. **Send21**: Two teams of two agents. A0 and A1 form one team and A2 and A3 form the other team. A0 sends

TABLE I.

COMMUNICATION PROTOCOLS

Communication Protocols	Method of Message Passing
Send22	A0 ↔ A1 A2 ↔ A3
Send21	A0 → A1 A2 → A3
SendLine	A0 → A1 → A2 → A3
SendLine2D	A0↔A1↔A2↔ A3
Send13	A0 → A1, A2, A3
SendAll	A0 → A1, A2, A3 A1 → A0, A2, A3 A2 → A0, A1, A3 A3 → A0, A1, A2
SendK (similar to Iba [5]) S0 S1 S2	 A → nearest agent A → 2nd nearest agent A → farthest agent

to A1, A2 sends to A3. **SendLine**: Each agent sends to one other agent only (except for the last agent) in the form of a line such that A0 sends to A1, A1 sends to A2 and A2 sends to A3. **SendLine2D**: Each agent sends to two other agents (except for the first and last agents) in the form of a line such that A0 sends to A1, A1 sends to A0 and A2, A2 sends to A1 and A3 and A3 sends to A2. **Send13**: The first agent A0 (acting as a leader), sends to all other agents A1, A2 and A3. **SendAll**: Each agent sends to every other agent. **SendK**: A set of "send" commands (similar to the ones used by robot agents in Iba [5]) that allow the agent, *A*, to send to other agents based on proximity. *S0* allows *A* to send to its nearest agent, *S1* allows *A* to send to its second nearest agent, and *S2* allows *A* to send to its farthest agent.

E. Environment

Experiments in this study use the Pursuit Domain Package (PDP) by Kok and Vlassis [15]. According to Reverte et al [2], PDP is a toolkit which simulates the predator-prey problem. It includes one prey agent and four predator agents and allows the modification of parameters to instantiate different experimental scenarios. Its environment consists of a grid in which agents are allowed to move to every adjacent cell. In this study a 20 x 20 grid is used, where 1 cell = 1 unit of distance. Predator agents have a field of view (FOV) = 2. The initial starting position of the prey is chosen at random and is confined to the prey's starting area on the grid. The prey's starting area, as well as each predator's starting area, are seen in Figure 2. The starting areas are arranged so that they don't overlap (similar to Iba [5] and Haynes et al [16]). Agent 0 is shown as light blue, Agent 1 is shown as purple, Agent 2 is shown as dark blue, Agent 3 is shown as green and the Prey (triangle) is shown as orange. The green lines on the grid outline the confined starting areas of the prey and each agent on the grid. The yellow cells on the grid define each agent's FOV. All four predator agents and the prey can move to only one cell (Up, Down, Left, Right or Stay) in 1 time cycle. The grid is toroidal such that if the prey's (or any predator's) next move is outside of the grid boundary, then it will move to the next cell on the opposite edge (i.e. the movements are wrapped at the edges).

The rules for information sharing from the work by Reverte et al [2] are changed as follows: 1) Predator agents are not aware of each others location but they are aware of all other predators' relative (nearest, second nearest and farthest) direction (similar to robot agents in Iba [5]). Predators are able to send messages to each other regardless of proximity. 2) Only predators within the FOV of the prey can share information about the direction of the prey via message sending.

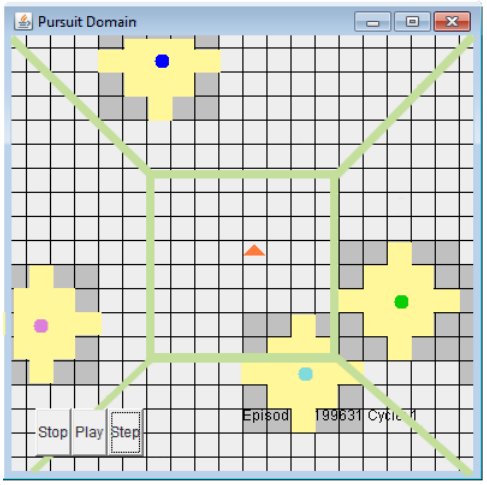

Fig. 2. Pursuit Domain Environment

TABLE II.

GP PARAMETERS

GP Parameter	Value
Max tree depth	4-6
Population size	1000
Generations	125
Selection	Tournament, size = 4
Crossover	90%
Mutation	10%
Runs per experiment	20

III. EXPERIMENT DETAILS

This section describes the experiment details for two experiments, *Prey Linear Movement* and *Prey Random Movement*. Each experiment uses the same settings for the GP parameters found in Table II. The fitness function, GP language, testing and training methods are also the same for the two experiments and are described below. The only difference between the two experiments lies in the movement of the prey. In the first experiment, *Prey Linear Movement*, the prey starts from a random position within its own start area and moves linearly (Up ↑) on the grid. In *Prey Random Movement*, the prey moves in a random pattern (Up, Down, Left or Right) on the grid. Both agents and prey move only one step per time cycle. Both experiments tested each communication protocol individually.

A. GP Language

This section describes the top-level GP structure, terminal set and function set. The GP language is limited in order to allow high-level behaviours to emerge.

1) Top-level GP Structure: Figure 1 shows the overall GP structure used in both experiments for this study. The root node contains 4 child nodes each of which represents the GP tree for one agent. Each agent node contains its own message buffer and two child branches (command trees), *C0* and *C1*. The evaluation of agents occurs in the same order from left to right, starting at A0 and ending with A3. At the time of an agent's evaluation, the message buffer is checked. If the

TABLE III.

TERMINAL SET

Name	Description
Up,Down,Left,Right,Stay	move commands: $\uparrow, \downarrow, \leftarrow, \rightarrow$
North,South,West,East	$(0,1), (0,-1), (-1,0), (1,0)$
C1	communication command
Goal	direction from prey to agent
	if agent is within FOV
AgentDir0-2 (similar to Iba [5])	direction from nearest(0))
	2nd nearest (1) and farthest agent(2)
LRM	Last Received Message

TABLE IV.

FUNCTION SET

Function	Description
Add	vector addition
Sub	vector subtraction
S2	scales vector by 2
S1/2	scales vector by 1/2
Rotate90	rotates vector by 90 degrees
Reverse	multiplies vector by -1
IfGrtEql	compares the length of two vectors
IfDot	calculates the dot product of two vectors
IfDist	checks if agent is withing FOV of prey
Send	sends a message to another agent
	(see Communication Protocols, Table I)

message buffer contains one or more messages then the agent evaluates its *C1* child, otherwise it evaluates its *C0* child. The evaluation of all four agents occurs in 1 time cycle and each evaluation results in one movement (Up, Down, Left, Right or Stay) on the grid. The GP structure consists of a heterogeneous team of agents such that each agent uses its own tree.

2) Terminal Set: The terminal set used is defined in Table III. Movement commands (Up, Down, Left, Right and Stay) are sent directly to the agent as a result of the evaluation of its command tree. The language is typed such that only 1 movement is sent per evaluation. The direction vectors, North, South, West and East contain the unit vector of each direction. There are two communication commands, *C0 and C1*. *C0* is used as the default command if no messages have been sent to an agent and *C1* is the command used when messages are sent by agents. The Goal terminal gives the direction to the agent only if the agent is within field of view (FOV) of the prey, otherwise it gives a default direction of (40,40). Directions to the nearest, 2nd nearest and farthest agent are given in AgentDir0, AgentDir1 and AgentDir2 respectively (similar to the ones used for robot navigation in Iba [5]). The terminal node to hold data for the last message removed from an agent's message buffer is named Last Received Message (LRM). Before evaluation of its tree, an agent checks its message buffer. If it contains messages, the first message is removed and its data is set to the LRM variable to be used for that evaluation cycle. If there are no messages in its buffer, the LRM node is set to the default vector (40,40).

3) Function Set: The function set is seen in Table IV. Functions include mathematical operations on two dimensional vectors, logical operations and message sending commands. Each logical operation consists of at least two child nodes [5]. The result of the logical operation will evaluate only one of the child nodes. For *IfGrtEql*, if the length of the first input vector is $\geq$ to the length of the second input vector, then the third child node is evaluated, otherwise, the fourth child is evaluated. In *IfDot*, the result of the dot product of the first two input vectors is calculated. If $0 < result \leq 1$ then the third child node is evaluated, otherwise, the fourth child is evaluated. For *IfDist*, if the agent is within FOV of the prey, then the first child node is evaluated, otherwise, the second child is evaluated. Finally, the message sending command is listed as *Send*. In both experiments, all seven communication protocols, as listed in Table I, are tested individually. For each test, the *Send* command is replaced with the specific communication protocol. For all communication protocols, when an agent issues a send command it first checks to see if it is within FOV of the prey. If it is within FOV, the message data sent to the receiving agent contains the direction from the receiving agent to the prey, otherwise, it contains a default value of (40,40).

B. Training and Testing Methods

The training and testing of a GP individual consists of cycles and episodes. Training and testing begin with the agents and prey starting in a random position within their own area on the grid (See Figure 2). In one cycle, all four agents evaluate their command tree once, one at a time. Each evaluation results in one movement of the agent, where one movement equates to one (cell) on the grid (and one unit in distance). After 30 cycles, one episode is complete (i.e. 1 episode = 30 cycles). In training, each GP individual is given 10 episodes and the positions of the agents/prey are reset to the original starting position after each episode is complete. The test run uses the GP individual with the best fitness in training. This GP individual is tested with 30 episodes instead of 10 and the test run sets a new random start position for each agent and the prey before a new episode begins.

C. Fitness Function

Fitness is measured by finding the sum of episode fitness scores. The episode fitness is the sum of each of the agent's distance to the prey in 30 cycles, where each cell on the grid represents 1 unit of distance. GP individuals with better fitness scores will minimize the distance sum as agents track (keep as close as possible to) the prey. Equation (1) shows the total distance fitness calculation used in training:

$$TotDist = \sum_{k=1}^{q} \sum_{j=1}^{m} \sum_{i=0}^{3} \sqrt{(A_i.x - P.x)^2 + (A_i.y - P.y)^2} \tag{1}$$

Here, A_i represents the location of $Agent_i$, where $i = 0...3$, *P* is the location of the prey, *m* represents the number of cycles and *q* is the number of episodes. We set q to 10 in training and to 30 in testing in our experiments. Similar to Equation (1), the test run fitness measures the average distance of all the episodes as seen in Equation (2):

$$AveDist = \frac{TotDist}{q} \tag{2}$$

IV. RESULTS

Table V displays the performance of test runs for the best individual GP found in each training run by listing the minimum fitness, the maximum fitness, and the average fitness of 20 test runs.

TABLE V.

TEST FITNESS SUMMARY (20 RUNS). FITNESS IS A MINIMIZATION FUNCTION. SEE EQUATION (2).

Prey Movement	Communication Type	Min Fitness of 20 runs	Ave Fitness of 20 runs	Max Fitness of 20 runs
Linear	SendAll	707	788	852
	Send22	727	789	878
	Send21	713	801	876
	Send13	712	803	849
	SLine2D	747	806	875
	SLine	728	808	920
	SendK	783	827	868
Random	SendAll	638	720	776
	Send21	683	729	805
	SLine2D	682	730	763
	SLine	674	735	774
	SendK	672	735	772
	Send13	673	735	767
	Send22	698	739	803

TABLE VI.

POSSIBLE MOVES PER AGENT

SendAll (Linear Run 14)			*SendAll* (Random Run 12)		
	C0	C1		C0	C1
Agent 0	D R S	U D S	Agent 0	L S	U L S
Agent 1	U D R	R	Agent 1	U L S	L R
Agent 2	D R	U S	Agent 2	L	U D
Agent 3	U D L	U S	Agent3	L R	L R
Send22 (Linear Run 15)			*Send22* (Random Run 3)		
	C0	C1		C0	C1
Agent 0	U D	D	Agent 0	U	D L
Agent 1	U D L S	U D L	Agent 1	D L R	D L R
Agent 2	U D S	U D S	Agent 2	L R S	D S
Agent 3	U L	U D	Agent 3	U D R S	U L

A. Summary of Results

The average test fitness values in Table V reveal that for linear movement of prey, SendAll and Send22 are the top performers with scores of 788 and 789 respectively, and SendK is the worst performer with a score of 827. The results for the random movement of the prey show that there are small differences in the average fitness values across all communication protocols with an average fitness range from 720 to 739. Again SendAll is seen as the top performer with a score of 720 and interestingly, Send22 is the worst performer with a score of 739. To verify the significance of the results, the One-Way ANOVA test with a 95% confidence interval is used. The ANOVA test uses a two-tailed T-test with seven factors, where each factor represents one communication protocol including the final test fitness for each of the 20 test runs. The ANOVA test shows that there is a significant difference in the fitness results between top performers (SendAll and Send22) and the worst performer (SendK) for linear movement of prey and shows that there is not a significant difference in the results for random movement of the prey.

Further analysis identifies the characteristics of the GP environment that allow SendAll to achieve the top fitness scores in both experiments and that influence Send22 to finish as the top performer in the *Prey Linear Movement* experiment. The analysis reveals an alternating message synchronization pattern emerging among predator agents. In addition, the learned behaviour and collaboration of agents in the best individual for SendAll resembles the behaviour of guards and reinforcements that can be found in popular stealth video games (e.g. *Metal Gear Solid (MGS)* [14]).

Table VI illustrates the division of movements between the C0 and C1 branches of an agent's command tree. The results show that this division benefits the coordination of agents by helping them to find and follow the prey. This table was created by examining the trees for top performers of the SendAll and Send22 communication protocols. The C0 and C1 columns represent the child branches of an agent and list the possible moves (Up (U), Down(D), Left(L), Right(R) or Stay(S)) the agent could make as a result of evaluating the branch. To find and follow the prey in the *Prey Linear Movement* experiment, agents must learn to move in at most 2 directions. The first direction they need to learn should get them to the center of the grid. Learning this direction will help the agents find the prey. In order to follow the prey, the second direction they need to learn is the "Up" direction (since the prey is continually moving in that direction). In the *Prey Random Movement* experiment, the agents have a more difficult task, learning to move in all 4 directions to find and follow the prey. In this case, the prey can choose a random movement (Up, Down, Left or Right) each cycle.

Table VI shows that in the *Prey Linear Movement* experiments, all agents have the possibility of moving in the "Up" direction in at least one of their branches (C0 or C1) whereas, they have the possibility of moving in 1-4 directions required in the *Prey Random Movement* experiments. These trees give most agents the ability to complete their task. However, it is the content of the message buffer (whether it has a message or not) that limit an agent's movement on a given cycle. Looking at the message buffers reveals that agents create an alternating synchronized pattern in sending and receiving messages, which in turn coordinates an agent's movement per cycle via the C0 and C1 branches of their command trees.

B. Guard Behaviour through Collaboration

A behaviour found in the video game series *Metal Gear Solid (MGS)* [14] is a guard protecting an area. Generally, the guard protects an area by remaining in a defined area. Once the guard spots an intruder, it begins to follow the intruder and notify other guards for reinforcement. Upon notification, reinforcement guards track (and attack) the intruder. The agents from the best test run in the *Prey Linear Movement* experiment for the SendAll communication protocol, evolved a simple form of this guard and reinforcement behaviour.

In the best test run (Run15), the agents collaborate through message sending and learn to rely on data sent by other agents to find and follow the prey. Table VII shows the message sending pattern for all four agents in this run. This table was created by examining the contents of the message buffers before each agent evaluated its command tree. Examining the message buffers show that Agent 0 (A0) and Agent 3 (A3) send a message to all other agents almost every cycle, Agent 2 (A2) sends multiple messages at once when it is in view of the prey, and Agent 1 (A1) does not send any messages at all.

Additionally, it is seen in Figure 3 that Agent 2's C1 branch does not send a message and does not move (Stays) if it is not in view of the prey. However, if it is in view of the prey, Agent 2 sends out four messages at once to every other agent. Because Agent 2 is in view of the prey, the data in the messages it sends contains the direction to the prey relative to the receiving agent. Sending out four messages at once to each agent almost guarantees that the receiving agents (Agent 0, 1 and 3) have a message from Agent 2. These messages would

TABLE VII.

GUARD BEHAVIOUR MESSAGE SENDING PATTERN

SendAll(Linear) Run 15

Sender		*Receivers*	*Description*
$A0$	$\rightarrow$	$A1, A2, A3$	*(always sends)*
$A1$			*(never sends)*
$A2$	$\rightarrow$	$A0, A1, A3$	*(multiple sends when in view of prey)*
$A3$	$\rightarrow$	$A0, A1, A2$	*(always sends)*

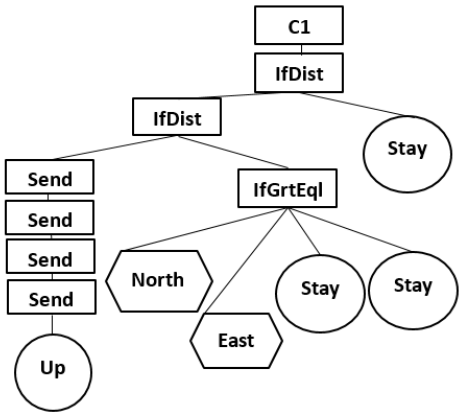

Fig. 3. Guard Behaviour GP: Example of Agent 2's C1 branch

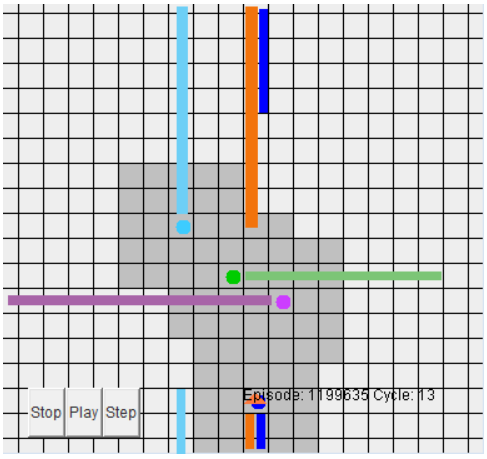

Fig. 4. Guard Behaviour Cycle 13. Coloured lines show the path of the agent from Cycle 0-13. Agent 2 (dark blue) is in FOV of prey and sending messages to all agents notifying them of the prey's location. At Cycle 13, Agent 3 (green), using LRM data sent from Agent 2, waits for prey and Agent 1 (purple) changes direction and begins to move left.

set the receiving agent's LRM data with relative directional data to the prey. The command trees for Agent 1 and Agent 3 contain the LRM node and, from viewing agents' message buffers and the test run video playback, seem to use this data to locate and follow the prey. This interaction shows that Agents 1, 2 and 3 collaborate via message sending/receiving to find and follow the prey.

Reviewing the video play back of the guard behaviour test run (single frames shown in Figures 4, 5 and 6), shows that for almost all episodes, Agent 2 (blue) acts a "guard" waiting on spot until it views the prey. When it sees the prey it sends multiple messages to all other agents, who at the time are not in FOV of the prey. These messages seem to help Agent

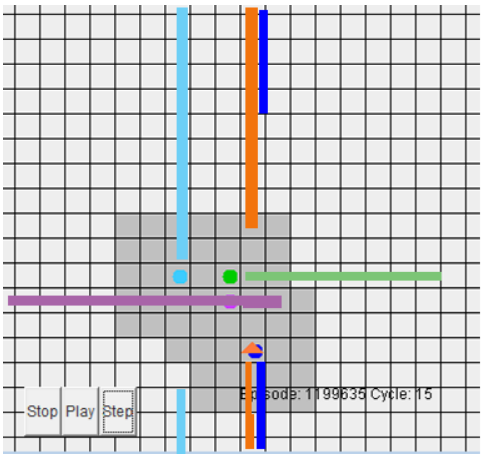

Fig. 5. Guard Behaviour Cycle 15. Coloured lines show the path of the agent from Cycle 0-15. Using LRM data sent from Agent 2 (dark blue), Agent 1 (purple) stops moving left and waits for prey.

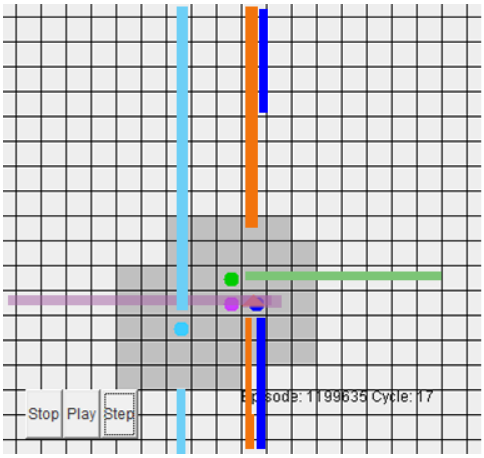

Fig. 6. Guard Behaviour Cycle 17. Coloured lines show the path of the agent from Cycle 0-17. Agent 1 (purple), Agent 2 (dark blue) and Agent 3 (green) are all in FOV of prey and all follow prey in the next cycles (not shown).

1(purple) and Agent 3 (green) find and follow the prey. The guard and reinforcement behaviour is shown in the cycle time line found in Table VIII and Figures 4, 5 and 6.

C. Synchronized Alternating Sending Pattern

The results show that most tests evolved competent agents that were able to find and follow the prey. Many of the communication protocols did not produce significant differences in fitness scores or perceived behaviours. However, some experiments did regularly evolve interesting behaviours that show high-levels of coordination among agents. These behaviours are highlighted in the remainder of this section.

1) Prey Linear Movement: Table IX describes the coordination of message passing among agents for one of the top individuals of the SendAll and Send22 communication protocols. It was created by examining the contents of the message buffers before each agent evaluated their command tree. SendAll (Run 14) has two agents (Agents 1 and 3) that almost always send out messages to all other agents and has two agents (Agents 0 and 2) that rarely/never send messages. The message buffer data shows that Agents 1 and 3 coordinate their message passing by synchronizing their "sends" so that

TABLE VIII.

GUARD BEHAVIOUR TIME LINE

Time Line Event	Cycle(s)	Description
0 (not shown)	0-4	*Agent 0 moves Down, Agent 1 moves to the Right, Agent 2 waits in position, Agent 3 moves to the Left*
1 (not shown)	5	*Agent 2 enters FOV of prey, floods all other agents' message buffers with prey location data*
2 (not shown)	5-12	*Agent 0 continues to move Down Agent 1 continues to moves to the Right, Agent 2 follows prey and sends messages, Agent 3 continues to move to the Left*
3 (see Fig. 4)	13	*Using LRM data from Agent 2 Agent 1 begins to move Left and Agent 3 Stays, Agent 0 continues to move Down, Agent 2 follows prey and sends messages*
4 (see Fig. 5)	15	*Using LRM data from Agent 2, Agent 1 and 3 wait to be in FOV of prey Agent 0 continues to move Down, Agent 2 follows prey and sends messages*
5 (see Fig. 6)	17	*Agent 1, 2 and 3 are all in FOV of prey and follow prey in the Up direction*

TABLE IX.

MESSAGE SENDING PATTERNS (PREY LINEAR MOVEMENT)

SendAll Run 14

Sender		*Receivers*	*Description*
$A0$	$\rightarrow$	$A1, A2, A3$	*rarely sends*
$A1$	$\rightarrow$	$A0, A2, A3$	*sends with A3 every other cycle*
$A2$			*(never sends)*
$A3$	$\rightarrow$	$A0, A1, A2$	*sends with A1 every other cycle*

Send22 Run 15

Sender		*Receivers*	*Description*
$A0$			*never sends*
$A1$			*never sends*
$A2$	$\rightarrow$	$A3$	*sends every other cycle*
$A3$	$\rightarrow$	$A2$	*sends every other cycle*

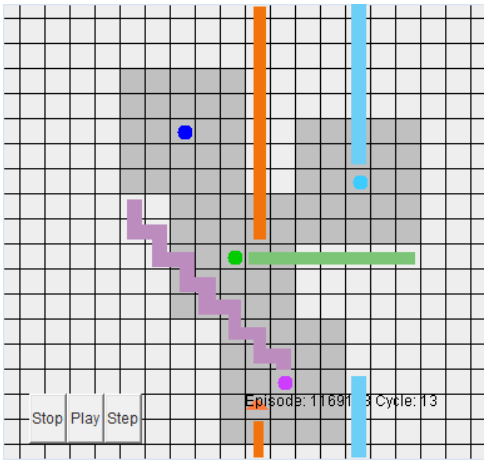

Fig. 7. Staircase Pattern SendAll (Agent 1 (purple))

they send a message to all other agents on the same cycle, every other cycle.

This synchronization results in Agents 1 and 3 having messages in their buffers every other cycle causing them to

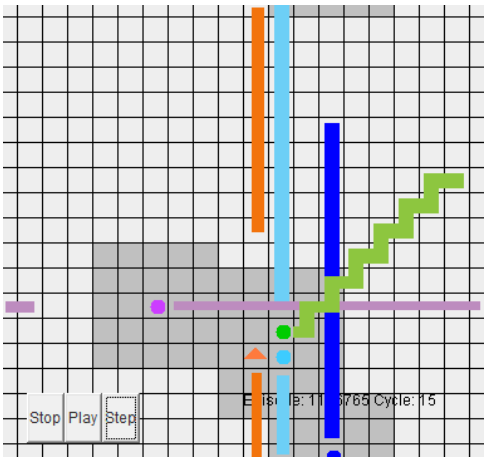

Fig. 8. Staircase Pattern Send22 (Agent 3 (green))

alternate the evaluation of their C0 and C1 branch each cycle. This coordination is an advantage for these agents because the required movement nodes (to achieve their goal of finding the prey) are divided between both branches. For example, looking at Table VI, for Run 14, it is seen that in Agent 1's C0 branch, it has nodes to move Up, Down, and Right, and in its C1 branch has a node to move Right. Based on Agent's 1 starting position area in Figure 2, Agent 1 must learn to move Right (towards center of grid) in order to find the prey, and it must learn to move Up in order to track the prey. The alternating pattern of message receiving allows Agent 1 to use two directions to find the prey. Switching the evaluation of its C0 and C1 branches each cycle, causes Agent 1 to move Right (C1 branch) then Down (C0 branch). This synchronization causes Agent 1 to move in a distinct staircase pattern until it finds the prey as seen in Figure 7. Once it finds the prey it is able to move in the Up direction to follow the prey (not shown).

Table IX also shows the message sending pattern of all agents in Send22 (Run 15). This data shows that Agents 0 and 1 never send messages to each other. However, the message buffers for Agent 2 and Agent 3, once again show a synchronized alternating sending pattern. Agent 2 and Agent 3 use alternate cycles to send messages to each other. In turn, this causes Agents 2 and 3 to evaluate their C0 or C1 branch every other cycle. Similar to previous tests, this causes Agent 3 to move in a distinct staircase pattern until it finds the prey (see Figure 8). Once it finds the prey it is able to move in the Up direction to follow the prey (not shown).

2) Prey Random Movement: Table X shows the message sending pattern for all four agents for SendAll (Run 12) in the *Prey Random Movement* experiment. Examining the message buffers shows that Agent 0 and Agent 3 send a message to all other agents every other cycle, while Agent 1 and Agent 2 rarely send messages at all. Similar to previous results, Run 12 has two agents (Agents 0 and 3) that use a synchronized alternating sending pattern. In this case, Agent 0 and Agent 3 send one message to all other agents every other cycle so that Agents 1 and 2 always have a message in their buffer and Agents 0 and 3 have a message in their buffer every other cycle. This synchronization does not help Agent 3 because its C0 and C1 branches combined contain only 2 (Left and Right) of the 4 required movements to follow the prey. Thus, Agent 3

TABLE X.

MESSAGE SENDING PATTERNS (PREY RANDOM MOVEMENT)

SendAll Run 12

Sender		*Receivers*	*Description*
$A0$	$\rightarrow$	$A1, A2, A3$	*(sends every other cycle)*
$A1$	$\rightarrow$	$A0, A2, A3$	*(rarely sends)*
$A2$	$\rightarrow$	$A0, A1, A3$	*(rarely sends)*
$A3$	$\rightarrow$	$A0, A1, A2$	*(sends every other cycle)*

Send22 Run 3

Sender		*Receivers*	*Description*
$A0$			*(rarely sends)*
$A1$			*(rarely sends)*
$A2$	$\rightarrow$	$A3$	*(sends every other cycle)*
$A3$	$\rightarrow$	$A2$	*(sends every other cycle)*

can find and follow the prey only as the prey moves in the left and right directions. When viewing all episodes in the video playback for test Run 12, it is seen that some of the agents are able to find and follow the prey using 2 or 3 directions. However in most cases, agents fail to continue to follow the prey when the prey moves in a direction that is not included in either its C0 or C1 branch.

Table X also shows the message sending pattern of all agents in Send 22 (Run 3). Examining the message buffers for this run shows that Agent 0 and Agent 1 rarely send messages to each other. Agents 2 and 3 often send messages to each other every other frame, similar to the alternating synchronized sending pattern that is seen in previous experiments. In this case, alternating the message "sends" does not seem to help Agents 2 and 3. The test run playback, along with the message buffers, shows that when Agent 3 is in view of the prey, it does not send out a message to Agent 2 because it is not Agent 3's "turn" in the alternate send pattern to send a message. The fact that Agent 3 sees the prey is lost to its partner. The unpredictable random movement of the prey causes the agents to easily move out of view of the prey. This test run shows that the alternating sending pattern can cause agents to miss opportunities to notify other agents of the prey's location.

V. CONCLUSION

This study continued the research of using GP in a multi-agent system. Using the pursuit domain and a co-operative learning strategy, multiple predator agents learned the meaning of communication commands in order to achieve their goal of finding and following a prey. A number of different communication protocols were examined in two different experiments, *Prey Linear Movement* and *Prey Random Movement*. The outcome of this study revealed an emergent behaviour of a synchronized alternating sending pattern among predator agents. This synchronized behaviour was shown to help the coordination of agents in the top performing communication protocols in the linear movement experiment. The random movement experiment provided a more difficult problem and results showed that synchronized message sending was not as effective. In addition, the results showed the learned behaviour and collaboration of agents resembled the behaviour of guards and reinforcements that can be found in popular stealth video games (e.g. *Metal Gear Solid (MGS)* [14]).

The finding of guard and reinforcement behaviour could be beneficial to future work specifically related to game production. For example, an *agent behaviour* design tool could train predators to respond to different game scenarios. Future work will involve an augmented solution to the *Prey Random Movement* problem with the goal of allowing the agents to not only find, but also follow the prey's movement in all four directions. In addition it would be beneficial in future work to test how this solution performs in different variants of the pursuit domain (e.g. more/fewer predator agents, more prey, larger grids, diagonal movement etc...).

ACKNOWLEDGMENT

Research supported through NSERC RGPIN-2016-03653.

REFERENCES

[1] L. Pannait and S. Luke, "Cooperative Multi-Agent Learning: The State of the Art," *Autonomous Agents and Multi-Agent Systems*, vol. 11, pp. 387–434, 2005.

[2] J. Reverte, F. Gallego, R. Satorre, and F. Llorens, "Cooperation strategies for pursuit games: from a greedy to an evolutive approach," *MICAI 2008: Advances in Artificial Intelligence*, vol. 5137, pp. 806–815, 2008.

[3] T. Haynes, S. Sen, D. Schoenefeld, and R. Wainwright, "Evolving multiagent coordination strategies with genetic programming," Proc. Artificial Intelligence, Tech. Rep., 1995.

[4] J. Denzinger and M. Fuchs, "Experiments in learning prototypical situations for variants of the pursuit game." in *Proceedings on the International Conference of Multi-Agent Systems(ICMAS-1996)*, 1996, pp. 48–55.

[5] H. Iba, "Evolutionary learning of communicating agents," *Journal of Information Sciences*, vol. 108, pp. 181–205, 1998.

[6] J. Kam-Chuen and C. Giles, "Talking Helps: Evolving communicating agents for the predator-prey pursuit problem," *Artificial Life*, vol. 6, pp. 237–254, 2000.

[7] H. Yanco and L. Stein, *An adaptive communication protocol for cooperating mobile robots*, H. R. Meyer, JA and S. Wilson, Eds. Cambridge Ma: The MIT Press, 1993.

[8] I. Tanev, M. Brzozowski, and K. Shimohara, "Evolution, generality and robustness of emerged surrounding behaviour in continuous predators-prey pursuit problem," *Genetic Programming and Evolvable Machines*, pp. 301–318, 2005.

[9] B. Zhang and D. Cho, "Evolving complex group behaviors using genetic programming with fitness switching," *Artificial Life and Robotics*, vol. 4, pp. 103–108, 2000.

[10] S. Luke, C. Hohm, J. Farris, G. Jackson, and J. Hendler, "Co-evolving soccer softbot team coordination with genetic programming," in *Proceedings of the First International Workshop on RoboCup, IJCAI-97*, Nagoya, Japan, 1997.

[11] J.Y.Kuo, F. Huang, S. Ma, and Y. Fangjiang, "Applying hybrid learning approach to robocup's strategy," *The Journal of Systems and Software*, vol. 86, pp. 1993–1944, 2013.

[12] A. Cardona, J. Togelius, and M. Nelson, "Competitive coevolution in Ms. Pac-man," in *2013 IEEE Congress on Evolutionary Computation (CEC)*, Cancun, Mexico, 2013, pp. 1403–1410.

[13] A. Alhejali and S. Lucas, "Using a training camp with genetic programming to evolve Ms. Pac-man agents," in *2011 IEEE Conference on Computational Intelligence in Games (CIG)*, Seoul, Korea, 2011, pp. 118–125.

[14] (2004) Metal Gear Solid The Twin Snakes. [Online]. Available: http://www.konami.jp/gs/game/mgs_tts/

[15] J. R. Kok and N. Vlassis, "The pursuit domain package," Informatics Institute, University of Amsterdam, The Netherlands, Tech. Rep. IAS-UVA-03-03, Aug. 2003.

[16] T. Haynes, K. Lau, and S. Sen, "Learning cases to compliment rules for conflict resolution in multiagent systems." in *Working Notes for the AAAI Symposium on Adaptation, Co-evolution and Learning in Multiagent Systems*, S.Sen, Ed., Standford University, CA, 1996, pp. 51–56.