Comparing apples with oranges: evaluating twelve paradigms of agency

Linus J. Luotsinen, Joakim N. Ekblad, T. Ryan Fitz-Gibbon, Charles Houchin, Justin Key, Majid Ali Khan, Jin Lyu, Johann Nguyen Rex Oleson, Gary Stein, Scott Vander Welde, Viet Trinh, and Ladislau Bölöni

School of Electrical Engineering and Computer Science
University of Central Florida,
Orlando, FL
{lluotsin}@mail.ucf.edu

Abstract. We report on a study in which twelve different paradigms were used to implement agents acting in an environment which borrows elements from artificial life and multi-player strategy games. In choosing the paradigms we strived to maintain a balance between high level, logic based approaches to low level, physics oriented models; between imperative programming, declarative approaches and "learning from basics" as well as between anthropomorphic or biologically inspired models on one hand and pragmatic, performance oriented approaches on the other.
Instead of strictly numerical comparisons (which can be applied to certain pairs of paradigms, but might be meaningless for others), we had chosen to view each paradigm as a methodology, and compare the design, development and debugging process of implementing the agents in the given paradigm.
We found that software engineering techniques could be easily applied to some approaches, while they appeared basically meaningless for other ones. The performance of some agents were easy to predict from the start of the development, for other ones, impossible. The effort required to achieve certain functionality varied widely between the different paradigms. Although far from providing a definitive verdict on the benefits of the different paradigms, our study provided a good insight into what type of conceptual, technical or organizational problems would a development team face depending on their choice of agent paradigm.

1 Introduction

Researchers have designed a bewildering variety of paradigms for the control of agents. Even if we restrict our inquiry to the case of embodied agents, that is, artifacts which operate either in the physical world or a simulation of it, virtually every paradigm of artificial intelligence, software engineering or control theory was deployed with more or less success. However, wide ranging comparisons of agent paradigms are rare. When new methods and paradigms are introduced, they are compared with only several, closely related approaches which are considered direct competitors of the proposed paradigm. Making or revisiting comparisons between paradigms is a controversial, difficult and hard-to-sell work. One might argue that a researcher might better spend his or her time in designing new paradigms or improving existing ones instead of comparing, say, swarm algorithms with affective computing in the design of embodied agents. There might be people offended by the results, with reasonable claims that the methodology was incorrect, the implementation of the paradigm substandard, or simply, the measured quantity is not relevant to the given paradigm.

The fundamental question, of course, is whether if any of these comparisons make sense. We argue that **if both paradigms A and B can be used in the implementation of the same requirements, then these two paradigms can (and indeed, should be) compared**. That is not to say that the comparison is easy or that it can be reduced to a single numerical "score". Different paradigms have different strengths and weaknesses, and the goal of a comparison study is to shed light on these differences. Although we do not expect definite answers on questions like "which paradigm would eventually lead to an agent passing the Turing test", we can provide insight into lesser but still important questions such as:

- Would the implementation provide adequate performance?
- Can a rigorous software engineering process be applied to the development?
- Can the performance be predicted?
- Can human expertise in the problem domain be transferred to the agent?
- What will the development effort be?
- Will the resulting agent be predictable in its actions?

The remainder of this paper is organized as follows. In Section 2 we present the Feed-Fight-Multiply game, our control problem. We succinctly describe the twelve agents we implemented in Section 3. We detail our findings in Section 4.

2 The Feed-Fight-Multiply world

To study the benefits and drawbacks of various agent paradigms, we decided to place them in a virtual environment in which many of the real world challenges are reflected. We did not choose one of the existing environments, because the existing implementations would have skewed the result of the comparison. One requirement towards the environment was the existence of *multiple paths to success.* We expected that agents implemented in various paradigms will have a different external behavior as well. By measuring success as the conformance to a predefined behavior we would have favored some paradigms and disadvantaged others. In addition, having multiple paths to success is a quality of most natural environments and many artificial ones.

Upon these considerations, we implemented the Feed-Fight-Multiply (or Mate) game, which borrows elements from turn-based multi-player strategy games and artificial life. Agents are sharing a two-dimensional environment having accessible zones and obstacles. The agents can sense their environment within the range of their *sensors.* Food resources appear at random points in the environment; consuming food increases the energy level of the agents. Finally, agents can multiply by (non-sexual) reproduction. The environment was implemented in the YAES simulation environment [3]. Figure 1 shows a typical FFM game in progress.

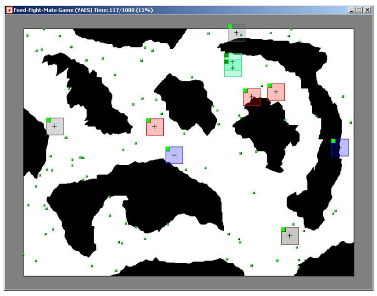

Fig. 1. Screenshot of the Feed-Fight-Multiply environment

An additional concern was to choose the level of the services provided by the environment. Evidently, natural environments do not provide any kind of service, but this would make the implementation of the agents unduly difficult. The guiding principle was that whenever the problem had a well known, standard implementation, we had chosen to implement it in the environment, and provide it as a service to the agents. These services included: the scanning of the sensor range for agents and food, tracking of moving agents and identifying agent types.

Finally, instead of keeping a single score, we decided to record multiple parameters of the agent behavior. This meant that not only there were multiple paths to success, but the final goals of the agents could be different as well. Of course, all agents were required to work towards their survival, but besides that, the criteria for success could be maximum amount of resources gathered, survival on minimum amount of resources, largest number of agents killed, number of individual agents of the same type at the end of the game, or others.

3 Twelve agents, twelve paradigms

We have developed twelve agents, implemented in twelve different paradigms of agency. In choosing the paradigms we strived to maintain a balance between high level, logic based approaches and low level, physics oriented models; between imperative programming, declarative approaches and "learning from basics" as well as between anthropomorphic or biologically inspired models on one hand and pragmatic, performance oriented approaches on the other. The implemented agents are concisely described in Table 1. The developers were instructed to develop paradigm-pure implementations and to design the agents such that the "spirit" of the paradigm is best expressed. When the paradigm could cover only some of the required functionality, the developers could use some limited heuristics.

Table 1. Concise description of the twelve implemented agents

Name	**Paradigm**	**Paradigm coverage**	**Team-work**	**Offline Learning**	**Realtime adapt.**
`AffectiveAgent`	Affective model, anthropomorphic lifecycle	Limited	No	No	Yes
`GenProgAgent`	Genetic programming	Full	Yes	Yes	No
`Reinforcer`	Reinforcement learning	Full	Yes	Yes	No
`CBRAgent`	Case based reasoning	Full	No	Yes	Yes
`RuleBasedAgent`	Forward reasoning	Full	Yes	No	No
`NaiveAgent`	Naïve programming (scripting)	Full	Yes	No	No
`GamerAgent`	Game theory	Limited	Yes	No	No
`CrowdAgent`	Crowd model	Limited	Yes	No	No
`NeuralLearner`	Neural networks	Full	No	Yes	No
`SPFAgent`	Social potential fields	Limited	Yes	No	No
`CxBRAgent`	Context based reasoning	Full	No	No	No
`KillerAgent`	Simple heuristics, with path-planning	Full	No	No	No

3.1 AffectiveAgent: anthropomorphic and affective model

The basic premise behind the affective agent paradigm is that the agent behaves with an emotional frame of reference with which to weigh its decisions. Besides providing the agent with emotional states such as anger, contentment or fear, we also made it to mimic the basic lifecycle of humans: agents have a childhood, maturity and old age, with their corresponding goals and priorities. In broad lines, our implementation is an adaptation of the agents from [14]. The affective model plays two roles in the behavior of the agent: *action selection* (e.g., what to do next based on the current emotional state) and *adaptation* (e.g., short or long-term changes in behavior due to the emotional states).

The short term variables which control the behavior of the agent are the *action tendency* and the *conflict tendency.* The dynamic action tendency is the probability whether an agent will fight or flee in a given situation. To adapt the action tendency to the outcome of the agent's interactions, the action tendency is updated by the *adaptation rule* depending whether the agent is experiencing loss or success.

This dynamic action tendency is used in calculating another dynamic parameter of the adaptive agents, namely, conflict tendency. This parameter determines whether an agent seeks conflicts or avoids them. Emotive states such as anger or fear are determined in terms of ranges of the action and conflict tendency.

Besides the mood of agent, its behavior is determined by its age. The agent remains content, without adaptation, until the agent comes of age, which is set, with apologies to Tolkien, to 33 cycles. After the age of maturity, the agents conflict tendency is adapted every 10 cycles. To avoid agents becoming bogged down in an emotional quagmire, a catalyst was installed in the way of a mood swing. At 50 cycles and every 25 subsequent cycles, the agents current action tendency is randomly reset to a new value and then action tendency and conflict tendency are recalculated. This provides a potentially dramatic change in mood. An agent could easily shift from an action tendency of 0.8 angry to 0.3 fearful.

The age of maturity was also employed to delay the agents mating. Moreover, the agents mating is also limited by mood and by energy level. An agent that is angry cannot mate. Only an emotional state of fearful or content will allow the agent to mate.

3.2 GenProgAgent: genetic programming

Genetic Programming (GP) [8] is an evolutionary algorithm in which the evolutionary units are computer programs or functions described by tree structures consisting of conditional branches, mathematical operators, variables and constants [2]. We based `GenProgAgent` on the generational genetic algorithm [6].

The evolution of the behavior of the agent is split in two stages. In the first stage we evolve *tactical behaviors*, which control primitive actions such as eat food, explore, attack and multiply. In the second stage we evolve *game strategies* by combining the behaviors from the first stage using Finite State Machine (FSM) structures.

Stage 1 - Evolving Tactical Behaviors. Four types of primitive behaviors were created: Eat-food, Explore, Attack and Flee.

The eat-food behavior was generated using a fitness function which defines an ideal individual as an agent that does not collide with obstacles and that eats all the available food resources. The fitness is derived based on the number of failed move sequences, the length of the failed move sequences and the amount of food eaten. Although the algorithm did not find an optimal solution, the best individual was able to, in most cases, effectively avoid obstacles and consume available food resources. The behaviors for Explore, Attack and Flee can be evolved similarly, or can be created from the Eat-Food behavior by replacing the heuristics. As we did not manage to evolve optimal primitive behaviors which reliably avoided being stuck on obstacles, we decided to augment the evolved behaviors with helper heuristics. The heuristics used provide directions to closest food, opponent agent and unexplored areas using A* search.

Stage 2 - Evolving Game Strategies. With the tactical behaviors already created, the next challenge is to decide which tactics to be applied at any given moment.

Tactical decision are considered the states of a finite state machine, and we apply genetic programming to evolve the transition rules for these structures. Two types of game strategies were created: Balanced and Aggressive. The balanced strategy seeks to create an agent that doesn't specialize on any type of behavior. The aggressive strategy seeks to create an agent that specialize on attacking and killing other agents. The game strategies were generated in a FFM game with 6 additional opponent agents. The purpose of the opponent agents is to generate hostile and conflicting situations from which strategies resolving these situations can be evolved.

3.3 Reinforcer: reinforcement learning

Reinforcement Learning (RL) has the ability to learn in an unknown domain without prior knowledge [10]. The specific technique chosen for the `ReinforcementAgent` was Temporal Difference (TD) learning. This approach uses a table of state action pairs and their corresponding reward. If an action leads to a good result, but this is not detected until several steps later, that good result will immediately propagate back to the initial action and therefore favor it in the future. The following formula was used in the `ReinforcementAgent` implementation.

$$\begin{aligned} Q(s_t, a_t) &= (1-\alpha) * Q(s_t, a_t) \\ &\quad + \alpha * (r_t + \gamma * max(Q(s_{s+t}, a_{t+1}))) \qquad (1) \\ \alpha &= 1/(1 + visits(s,a)) \qquad (2) \end{aligned}$$

Where α is related to the number of times that the state action pair has been visited, γ is a user defined value between 0 and 1, $Q(s,a)$ and $r(s,a)$ is the value and reward for current state action pair respectively.

The action set for `the ReinforcementAgent` is defined by 20 movement actions. Actions for eat, attack and flee are also included in the action set. The state set consist of various energy level thresholds, possible actions to take in each game round and on the objects and their directions as seen in the agent's sensors. In total there are 117760 state action pairs. Reinforcement is applied using direct stimuli from the environment. Negative reinforcement is imposed when the agent fail to perform some action or when the agent performs to many consecutive actions of equal type. In similar manner, positive reinforcements is given when an agent successfully performs some action.

3.4 CBRAgent: case-based reasoning

Case-Based Reasoning (CBR) [1, 13] is the process of intelligently solving new problems based on the previous similar problems. The basic steps of CBR are *retrieve*, *reuse*, *revise* and *retain*. First, CBR retrieves the most relevant case to the current problem at hand. The retrieved case is reused and revised to incorporate minor variations in the solutions. This adaptation step gives CBR a power to form more precise and accurate solutions to the future problems. Finally the revised solution is retained for future use. [9] has shown the capability of CBR as the intelligent search method for controlling the navigation behavior of the autonomous robot. Our implementation is an adaptation of this model.

The case was represented using ten parameters (the closest food, enemy and obstacle, the density of food and enemies, the ratio between the agents health and the opponents, and parameters determining the actions currently available to the agent). A weighting scheme was used to emphasize the more important features. We have identified 19 historic cases that are selected based on the performance of the agent in training simulations. Even with the small number of cases, the CBR-based agent shows drastic improvement over the random behavior and can be refined by adding more specific cases to the library. The selection of the case was done with a distance matrix based selection. The reliability of the case based on the previous failures is used as a weight in order to encourage the selection of variety of cases. The reliability of the case is dynamically adapted depending on whether the case can or cannot be successfully applied to the current situation.

After the action to be performed is selected by the CBR module, a set of heuristics are used to adapt the action to the current environment. First, the heuristic module checks whether the proposed action is feasible. In case of failure, the reliability measure of the case which proposed the action is reduced. The reliability of the case is used as a weight during the calculation of the distance matrix, so that case with the less reliability will have greater distance in the subsequent cycles. The second heuristic calculates the direction and the speed of the agent movement. For instance, eating a food item requires the direction toward the food and appropriate setting of the speed such that the agent stops at the food item.

3.5 RuleBasedAgent: forward reasoning

A rule based system consists of an inference engine and knowledge base. The engine's reasoning mechanism uses a forward-reasoning technique. The knowledge base contains a fact base and a rule base. The fact base acts as a repository of all the truths that is seen or understood by the agent. The fact base is periodically updated with new sensor data, which triggers the execution of rules. To reduce the number of rules necessary to determine the behavior of the system, we have decided to choose the atomic actions at a relatively high level of abstraction. For example, a typical rule would have the consequent of movement towards a particular spot. Many additional intermediate rules could have been implemented to determine exactly how to move towards the objective. Instead, once this rule has been fired, a helper function is called to determine where the objective is and binds various directional parameters of the consequent of the rule.

The rule base of the `RuleBasedAgent` agent consists of 15 rules. The consequent of each rule may result in another fact pushed onto the fact base or an action of the agent which would terminate the inference mechanism for that simulation cycle. The *salience* of the rule is used to aid in conflict resolution as well as the method of sorting in the rule base stack. This method prioritizes the various rules which in turn define the behavior of the agent.

The `RuleBasedAgent` implements all the five basic commands of the game (movement, feeding, fleeing, attacking and mating). Whenever a decision needs to be taken, it is determined by synthetic facts in the fact base of the agent. For instance, the choice to attack is determined by the fact AGGRESSIVE, while a choice to flee is triggered by the fact TIMID. Whenever there is a potential for an encounter with another agent, the decision of the aggressive or timid behavior is made by the relative energy of the agents.

The rule based agent implements flocking behavior with other rule based agents in its sensor range. If the fact base contains the synthetic fact FLOCK, the move commands will be restricted to movements which allow the agent to remain in the flock. The leader of the flock is the agent with the lowest id (the oldest agent). Two scenarios occur when food is within range while in a flock. If the GREEDY fact exists, the agent attempts to move and acquire the food closest to them. If the fact does not exist, then the agent only attempts to move and acquire a food resource if no other rule-based agent is closer to the resource. The second scenario promotes efficient feeding while in the pack to avoid agents ineffective attempts to acquire the same resource. The agents in the pack also synchronize their attack and fleeing behaviors. If the leader of the pack attacks an agent, all the members of the pack will attack, regardless of their energy levels or aggressiveness ratio.

3.6 NaiveAgent: naive programming, scripting

Naive programming is a style of coding that allows the developer to hand-optimize the code for a particular task. `NaiveAgent` relies on the hand scripting of encounters for its success - a technique frequently used in the development of multi-player games. For each possible encounter, a script was written specifying how the agent should react. In the following we discuss some of the heuristics used in the implementation:

Exploration: in the absence of other tasks, the `NaiveAgent` moves around the environment with its maximum speed. This way, by covering more area, the probability of finding food increases. It was found that the benefit of finding more food outweights the extra expenditure of energy. The higher coverage also increases the chance of encountering other agents, which is beneficial, given the aggressive nature of the `NaiveAgent`.

Obstacle avoidance: Instead of using a sophisticated decision-making process to guide the movement of the agent, the `NaiveAgent` simply moves right every time it encounters an obstacle. To avoid getting stuck, a `failcount` variable is incremented each time the agent makes a right turn. Only encountering another agent or food particle can reset the fail count. If the fail count is greater than five, the direction is chosen randomly.

Social behavior: If the `NaiveAgent` has a particle of food and another `NaiveAgent` is within the sensor range, only the agent with the lowest energy level is allowed to eat.

Aggression: Whenever a different agent is detected in the sensor range, and the agents' energy level is larger than 120% of the opponents, the `NaiveAgent` attacks the opponent. If more than two agents are in the range, the `NaiveAgent` attacks the weakest opponent.

3.7 GamerAgent: game theory

Game theory [7] is a mathematical formulation of cooperative or competitive interaction between multiple entities. The key concern in game theory is to extract rational (optimal) behavior from a given interaction between autonomous agents. We model the FFM world as a zero-sum game.

The game consists of two entities, and each one of them can choose from two strategies: attack or flee. The utility functions for each strategy are based on the ratio of energy levels $\mathit{\Delta}$ and the likelihood of attack or flee by the other agent based on previous interactions μ. We will denote $U_{a,b}$ the utility of taking action a when the opponent agent takes action b (where the actions can be A for attacking and F for fleeing. The utility functions are defined as follows:

$$U_{A,A} = (1 - \mu) \times 100 + \mathit{\Delta} \times 200 \tag{3}$$
$$U_{A,F} = \mu \times 100 + \mathit{\Delta} \times 200 \tag{4}$$
$$U_{F,A} = \mu \times 100 - \mathit{\Delta} \times 200 \tag{5}$$
$$U_{F,F} = (1 - \mu) \times 100 - \mathit{\Delta} \times 200 \tag{6}$$

Given the matrix, the optimal strategy is chosen by summing up the utility for each strategy. The strategy that provides the maximum utility is then chosen as the optimal strategy.

Several heuristics were used to guide the agent to explore the map and eat available food. The expert agent uses an internal data structure representing the perceived game map as a base for the heuristics.

3.8 CrowdAgent: crowd model

Crowd modeling techniques traditionally take inspiration either from fluid systems or particle systems. Both approaches deal with attractive, repulsive and frictional forces; in addition, particle systems place motion decision with the individual [4]. In the implementation of the `CrowdAgent`, we chose the aggression level of the agents as the grouping characteristic of the crowd. An agent will start out with an initial aggression rating, $A(0) = A_i$, and then migrate towards the aggression level of the agents surrounding them. This transition is governed by:

$$\begin{aligned} A(t + \mathit{\Delta} t) = A(t) + \frac{(A(0) - A(t))^3 * \mathit{\Delta} t}{am} + \\ \sum_{b=0}^{n} \left(\frac{(A_b(t) - A(t))}{cosh(A_b(t) - A(t))^2} \right) \cdot \frac{t}{max((D_b^2), 4)} \end{aligned} \tag{7}$$

Where D_b is the distance between the current agent and agent b. The first term of the equation guarantees that if the agent is not surrounded by other agents it will return to its initial aggression level. If there are other crowd agents in the neighborhood, the agent will have its aggression level pulled towards the aggression level of each of the surrounding agents. The motion of an agent is related to the position of all other agents in the sensor range, and what there aggression levels are. The equation of motion used is:

$$X(t+\Delta t) = X(t) + \Delta t \cdot V_x \cdot \textstyle\sum_{b=0}^{n} \frac{pF_b \cdot (X(t)-X_b(t))}{max(D_b)^3),1)} \tag{8}$$

A similar function is calculated for the Y direction. Once again we are summing over all agents in the sensor range, but this time we also generate a factor for the attraction between agents. The pF attribute is based on the aggression of the agent of interest and the aggression of the agent in the sensor range. This is an attractive/repulsive attribute which is defined by the piecewise function

$$f(n) = \begin{cases} -1 * \frac{pfA}{pfB^2} * |A - A_b|^2 + pfA \\ \quad \text{if } |A - A_b| <= pfB \\ pfC * 4 * \left(|A - A_b| - \frac{pfB+(10-pfB)}{2}\right)^2 - pfC \\ \quad \text{if } |A - A_b| > pfB \end{cases} \tag{9}$$

The pF factor will give an attractive influence between 0 and pfB, the remaining distance will give a repulsive influence. As long as the attractive forces are not made too large then the individuals will have the ability to separate from a group, and rejoin another group.

As the particle grouping paradigm deals only with the motion of agent in the presence of crowds, it was supplemented by a series of heuristics. In the absence of other agents, the agent will perform random wandering. If food is detected, the agent moves directly towards the food. The agent is reproducing with a random probability whenever the energy level is high enough. A simple heuristic was used for fighting: the agent tries to avoid coming in contact with other agents, but if it comes into contact, it will attack. Finally, a simple heuristics is used for obstacle avoidance. If an obstacle is in the direction you are trying to move then keep turning to the right until you find an open direction and go that way.

In practice we found that there was a need for at least 4 agents of this type to get any really dynamic interactions going, and this was also the needed level to guarantee a long survival time, the algorithm performing the best with 6 agents of this type, given the limitations of the environment size.

3.9 NeuralLearner: neural networks

Neural networks are a natural choice as the control paradigm for embodied agents. An agent is trained with a set of training data representing sensory inputs and desired actions as outputs, and a learning algorithm such as back-propagation [12] is used to teach the agent the optimal behavior.

The defining difficulty in our implementation was the acquisition of training data, a problem noticed by other artificial life reseachers as well [15]. The problem is that there is no input-output mapping inherent to artificial life simulations.

One must find a mapping that the neural network should estimate, and then acquire data based on that mapping. This requires the pre-existence of other agents, and the performance of the `NeuralLearner` will be determined by the performance of the model agent. Based on this balance, this project has two parts: search the entire input-output mapping space for a possible solution, then teach that solution to a neural network agent.

All the decisions in a `NeuralLearner` agent are made by a single multilayer neural network. The inputs to this network consisted of the agents current energy level, the presence and direction of another agents, food and obstacles. Also included in the input was whether or not the agent could currently eat, mate, attack, or flee. The output of this network was an action selection (move, eat, attack, flee, mate), a direction (north, south, east, or west), and speed value. To acquire data for the training of the `NeuralLearner` a random agent was first created to explore the artificial life world and record data to be used to train the network. The actions of the random agent where filtered, and the training set contained only the input-output pairs that either led to a direct increase in energy, or kept the agent alive over a long period of time. Unfortunately, the random agent usually (about 80% of the time) made a decision that did not lead to useful data. Hence, the random agent approach was a very inefficient method of acquiring data. To improve data acquisition, the random agent was pushed towards situations where it would have experiences, both good and bad.

The resulting data sets were used to perform offline training on the neural network of the agent. The network was then used statically with the agent, that is, no more learning took place.

3.10 SPFAgent: social potential fields

Social potential fields [11] are a way to control autonomous agents using inverse-power laws on attractive and repulsive forces between the agents and objects of the environment. We have implemented an agent whose movement is determined by a set of forces which attract or repulse the agent to agents and object of its sensor field. The resulting force is

$$\overline{F_i} = \sum_{j=1}^{N} \overline{v_{ij}} \cdot \left(-\frac{c_1}{r^{\sigma_1}} + \frac{c_2}{r^{\sigma_2}} \right) \tag{10}$$

where $\overline{v_{ij}}$ is the unit vector of the direction from agent i to agent j, and r is the distance from agent i to agent j. The parameters $c_1, c_2 \geq 0$ and $\sigma_1, \sigma_2 > 0$ are determining the nature of the forces between the agent and the object.

Once we decided on the general form of the forces, the next step is the choice of the parameters c_1, c_2, σ_1 and σ_2 such that the desired behavior of the agent is obtained. In practice, the determination of these parameters is a result of experience and experimentation. We have determined four sets of these parameters, which describe the relationship of a social potential field agent to (1) another social potential field agent, (2) an other agent, (3) food items and (4) obstacles. The experimentally obtained values are displayed in Table 2.

Table 2. The inverse power force law constants used in `SPFAgent`

Object of Interest	c_1	c_2	σ_1	σ_2
SPF Agent (a)	45.0	20.0	1.0	0.7
Other Agent (n)	45.0	0.0	1.0	0.7
Food (f)	0.0	20.0	1.0	0.8
Obstacle (o)	5.0	0.0	5.0	1.0

During testing, two major problems were found with the movements of the agents. Agents had a tendency to be stuck to into local minima, such as becoming immobile in the geometrical center of several food sources. Second, agents frequently overshot the food location and performed an oscillatory movement around it. A similar problem led to the agent bouncing indefinitely between two obstacles. These problems were solved by adding heuristics which (a) break the tie between the attraction forces and (b) prevent repetitive movements.

As the SPF paradigm describes only the movement of an agent, we applied a set of simple heuristics for the remaining actions. The attack heuristics dictates that the agent attacks any agent which gets closer than half of the critical distance 10. The mating heuristics encourages the mating of isolated SPF agents, but restricts the mating of SPF agents inside groups. As an emergent property, this heuristics leads to moderate size, relatively stable groups of SPF agents.

3.11 CxBRAgent: context based reasoning

Context-based Reasoning (CxBR) is a paradigm intended to model human tactical behaviors [5]. Contexts encapsulate knowledge about appropriate actions needed to address specific situations. The CxBR paradigm is composed of a tactical agent, mission context, major contexts, sub-contexts and sentinel rules which control the transitions between contexts.

`CxBRAgent` was implemented using eight different context constructs: 1) The ExploreContext is the default context of the mission. 2) The BackTrackContext is called from ExploreContext when there is nothing new to explore in the map at the current location of the agent. The agent will then retrace its step and search for new places to explore. 3) The AttackContext is deployed when there is a hostile entity within the sensor range. 4) The AvoidContext represents the case when there is a hostile entity within the sensor range and the agent cannot attack the other agent. The agent will move away from the hostile agent, trying to avoid being chased or attacked by the other agent. 5) The EatContext is called from either the ExploreContext or BackTrackContext when there is food within the sensor range. The agent will move towards the food and invoke the eat command on the food resource. 6) The FleeContext is invoked when the agent have been attacked. The agent attempts to flee away from an attacker. 7) The MateContext is invoked rules when the agent can mate. 8) The NearDeathContext is invoked when the energy level of the agent is below a certain threshold. It will attempt to extend its lifeline by spawning another agent.

The CxBR agent implements a simple path-planning algorithm which allows it to navigate an internal representation of the global FFM map. The same path-planner is used when approaching objects in the agent's local sensor range.

3.12 KillerAgent: simple heuristics

The `KillerAgent` is implemented using a set of simple heuristics. The agent keeps track of direction, failed moves and successful moves. If the number of consecutive failed moves exceed a predefined threshold then a direction switch is performed. The same simple idea is used if too many moves have been successful. However, the threshold is higher in this case. When the agent detects food in its sensor it will immediately navigate to it and eat the food. If the agent senses other agents within its sensor range it will prioritize an attack over any other action. If the agent itself is attacked it will flee. This agent does not utilize the multiply feature or any teamwork strategies.

4 Findings

Development process: We found that the explicit programming paradigms (scripting, rule based, CxBR and, to a certain level CBR) were easier to program, yielded a steady improvement in their performance during the development process and did not require costly rewriting (although, occasional Java refactorings were performed). The human knowledge integrated in these agents allowed them to outperform their implicit cousins in test runs.

Learning: All paradigms which relied on learning (Neural Networks, Genetic Programming and Reinforcement Learning) have been successful in creating agents which can survive in the environment in the absence of predators. For all paradigms the developers spent significant time designing learning scenarios in which the algorithms can be steered in the right direction. This was made difficult by the fact that these scenarios had to be populated with agents. The most problematic paradigm from this point of view turned out to be the neural network agent, whose supervised learning algorithm required an existing agent to perform the scenario to generate "correct" input and output pairs. Admittedly, this difficulty would be irrelevant in applications where such an imitation target exists and can be used at will.

What can be learned? All agents were successful in learning policies and inclinations (such as the ideal value of aggression), and they performed better than human intuition for these parameters. However, the learning agents were unsuccessful in learning (essentially, discovering) algorithms. The farthest they got was discovering approaches for collision avoidance, speed control for approaching the food, or avoiding to get stuck. However, we were not successful in developing path planning algorithms (such as an approach for visiting food locations). The genetic programming approach was the only learning model which would represent (and, theoretically evolve) such a model. However, an examination of the evolved genetic programs showed that they were very far from evolving anything like a path planning algorithm. This inevitably put them at a disadvantage against explicitly programmed agents which can deploy advanced algorithms such as A*, path planning, approximate Hamiltonian cycles and incrementally built internal maps.

A rose by another name. Many paradigms led to surprisingly similar implementations, while giving very different interpretations to the variables involved. For instance, the developers of affective models `AffectiveAgent` and `CrowdAgent` used the variables describing the emotional states as just another state variable and applying regular programming techniques on them, on hindsight labeling them with emotional significance (the write-up for affective agents contained terms such as "emotional quagmire" for being stuck in a local minima). On the other hand, developers of other agents tended to assign anthropomorphic significance to their state variables ("the agent gets angry"), even if their paradigm did not required it.

A similar phenomena was observed related to contexts. Context based reasoning, as implemented in the `CxBRAgent` requires the developer to actively identify the context of the agents operation and describes ways to handle it. However, the concept of context was actively used in at least four other agents.

The `SPFAgent` and the `CrowdAgent` ended up deploying very similar attraction and repulsion forces, starting from different physical models and very different high level interpretations.

The importance of the heuristics. Although the game was not easy (humans playing it at first time did not perform better than agents), human users could easily come up with rules of thumb which offered significant performance increase. **The ease of representing these heuristics in the agents was a determining factor in the performance**. Agents in which this could be done only in a very convoluted way (such as the learning agents, and, in lesser degree, the potential field, crowd and affective models), had scored the worst in direct comparisons, and led to significant frustration.

Paradigm-pure models considered harmful. We found that the requirement to use a single paradigm had significantly reduced the performance of the agents. This effect was less pronounced for the explicitly programmable models because the developers could "sneak in" some foreign concepts. These improved the performance on the short term, but led to messy design. We find it a good approach to plan for a hybrid model from the beginning, and explicitly plan the roles of various approaches in the functioning of the agent.

5 Conclusions

In this paper we report on the findings of a study in which twelve paradigms of agency were compared in an environment inspired from strategy games and artificial life. A more extensive report on the study, together with source code and playable simulation runs is available from the website `http://netmoc.cpe.ucf.edu/Yaes/index.html`.

References

1. A. Aamodt and E. Plaza. Case-based reasoning: foundational issues, methodological variations, and system approaches. *AI Commun.*, 7(1):39–59, 1994.

2. W. Banzhaf, P. Nordin, R. E. Keller, and F. D. Francone. Genetic programming - an introduction: On the automatic evolution of computer programs and its applications. In *Morgan Kauffman Publishers Inc.*, 1998.
3. L. Bölöni and D. Turgut. YAES - a modular simulator for mobile networks. In *8-th ACM/IEEE International Symposium on Modeling, Analysis and Simulation of Wireless and Mobile Systems MSWIM 2005*, 2005.
4. E. Bouvier, E. Cohen, and L. Najman. From crowd simulation to airbag deployment: Particle systems, a new paradigm of simulation. *J. Electronic Imaging*, 6(1):94–107, 1997.
5. A. J. Gonzalez and R. H. Ahlers. Context-based representation of intelligent behavior in simulated opponents. In *Proceedings of the Computer Generated Forces and Behavior Representation Conference*, 1996.
6. J. H. Holland. Adaptation in natural and artificial systems. In *University of Michigan Press, Ann Arbor*, 1975.
7. J.V.Neumann and O.Morgenstern. *Theory of Games and Economic Behavior.* Princeton University Press, 1944.
8. J. R. Koza. Genetically breeding populations of computer programs to solve problems in artificial intelligence. In *Proceedings of the Second International Conference on Tools for AI, Herndon, Virginia, USA*, pages 819–827. IEEE Computer Society Press, Los Alamitos, CA, USA, 6-9 Nov. 1990.
9. M. Likhachev, M. Kaess, Z. Kira, and R. C. Arkin. Spatio-temporal case-based reasoning for efficient reactive robot navigation. 2005.
10. T. M. Mitchell. *Machine Learning.* WCB/McGraw-Hill, 1997.
11. J. Reif and H. Wang. Social potential fields: A distributed behavioral control for autonomous robots. In *Proceedings of the International Workshop on Algorithmic Foundations of Robotics (WAFR)*, pages 431–459, 1995.
12. D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning internal representations by error propagation. In *Parallel distributed processing: explorations in the microstructure of cognition, vol. 1: foundations*, pages 318–362. MIT Press, Cambridge, MA, USA, 1986.
13. R. C. Schank. *Dynamic Memory: A Theory of Reminding and Learning in Computers and People.* Cambridge University Press, New York, NY, USA, 1983.
14. M. Scheutz. Useful roles of emotions in artificial agents: A case study from artificial life. In D. L. McGuinness and G. Ferguson, editors, *AAAI*, pages 42–48. AAAI Press / The MIT Press, 2004.
15. G. N. Yannakakis, J. Levine, J. Hallam, and M. Papageorgiou. Performance, robustness and effort cost comparison of machine learning mechanisms in FlatLand. *IEEE Proceedings of the 11th Mediterranean Conference on Control and Automation*, June 2003.