Collaborative Diffusion: Programming Antiobjects

Alexander Repenning
AgentSheets Inc & University of Colorado
Boulder, Colorado, 80301
ralex@cs.colorado.edu

ABSTRACT

Object-oriented programming has worked quite well – so far. What are the objects, how do they relate to each other? Once we clarified these questions we typically feel confident to design and implement even the most complex systems. However, objects can deceive us. They can lure us into a false sense of understanding. The metaphor of objects can go too far by making us try to create objects that are too much inspired by the real world. This is a serious problem, as a resulting system may be significantly more complex than it would have to be, or worse, will not work at all. We postulate the notion of an *antiobject* as a kind of object that appears to essentially do the opposite of what we generally think the object should be doing. As a Gedankenexperiment antiobjects allow us to literally think outside the proverbial box or, in this case outside the object. This article discusses two examples, a Pacman game and a soccer simulation where antiobjects are employed as part of a game AI called Collaborative Diffusion. In Collaborative-Diffusion based soccer the player and grass tile agents are antiobjects. Counter to the intuition of most programmers the grass tile agents, on top of which all the players are moving, are doing the vast majority of the computation, while the soccer player agents are doing almost no computation. This article illustrates that this role reversal is not only a different way to look at objects but, for instance, in the case with Collaborative Diffusion, is simple to implement, incremental in nature and more robust than traditional approaches.

Categories and Subject Descriptors

D.1.3 [Programming Techniques]: Object-oriented Programming

General Terms

Algorithms, Performance, Design, Reliability, Human Factors, Languages, Theory.

Keywords

Object-Oriented Programming, Psychology of Programming, Collaborative Agents, End-User Programming, Diffusion, Game AI, Incremental AI, multi-agent architecture, Distributed Artificial Intelligence.

1. INTRODUCTION

Overall, object oriented programming has worked very well. After early concerns about performance the intuitive appeal of object-oriented programming concepts has literally changed the way we think about programming. Over time object orientation has migrated from an art mastered only by early hacker type programmers using object extensions to Lisp or using Smalltalk to a more mature engineering discipline. Intangible design intuition has been gradually replaced with more principled approaches culminating in the unified modeling language (UML). All this has led to a strong sense of what objects are and what they should be doing. Once we know what the objects are and how they related to each other we typically feel confident to design and implement even the most complex systems. However, objects can deceive us. They can lure us into a false sense of understanding. The metaphor of objects can go too far by making us try to create objects that are too much inspired by the real world. This is a serious problem as a resulting system may be significantly more complex that it would have to be, or worse, will not work at all.

In teaching we can observe the kind of thinking going on in novice programmers learning about object-oriented programming. I have been creating objected-oriented programming languages [22], using and teaching object-oriented design long enough to get a sense on how people conceptualize object-oriented programming. In my game design and development courses students are learning about game design and development by building weekly games using AgentSheets [23]. We start with classic applications including Frogger, Sokoban and Space Invaders. Students have access to reference material including detailed game descriptions found in the Wikipedia.

Sokoban is a simple game in which a person needs to push boxes onto targets. What makes Sokoban interesting is that the person cannot pull boxes. If a box is pushed into a corner the game is lost. At advanced levels Sokoban can be surprisingly intricate requiring players to carefully plan their moves. Figure 1 shows a graduate student level 1 implementation of the game.

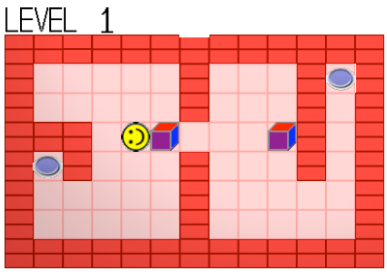

Figure 1. Sokoban Student Implementation. The Smiley person can only push boxes. Boxes need to be moved onto targets.

Part of the homework is to draw UML diagrams illustrating the basic game action. The sequence diagram below shows one situation in which the player is trying to push a box to the right onto an empty tile. When asked about the homework – after grading – some students indicate that drawing the UML diagrams helped them thinking about the game but the large majority of the students confesses that they made the UML diagrams after finishing the game. A large percentage of the students in these courses (computer science graduate and undergraduate students) has taken Object-Oriented Design course. Interestingly, if they have taken an OOD course played no significant role in their decision to use UML diagrams either as thinking tools – before programming – or as means of documentation, after programming.

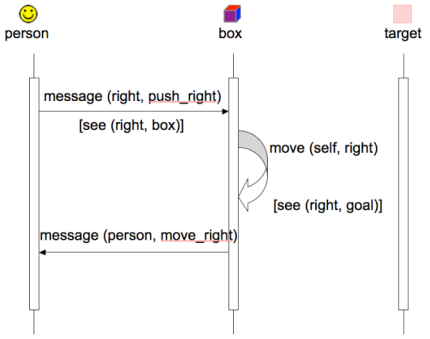

Figure 2. UML sequence diagram of person pushing box right onto empty tile. The person makes the box check if it can move. The box can move. It will move itself and the person.

Up to this point in the progression of increasingly complex games object-oriented design techniques have worked well. Students had a good intuition helping them to identify what the objects should be and how these objects should interact with each other.

Next students need to implement a game in which agents are pursuing each other through a complex world such as a maze. A sophisticated version of a tracking-based game would be a game like the Sims [32] where people agents are tracking other agents such as food or other people agents. A much simpler version of such a game would be Pacman where ghosts are trying to catch the user-controlled Pacman. To test students design intuitions we would show them a specific scenario in a Pacman game in which a ghost agent would try to tackle the Pacman.

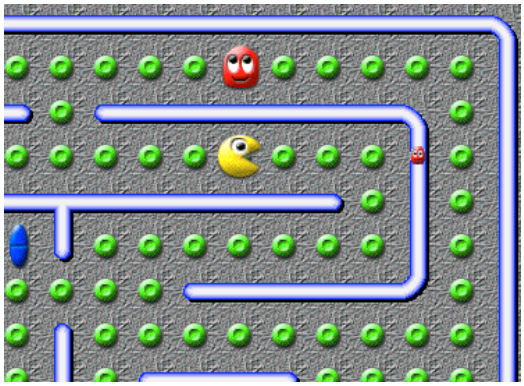

Figure 3. Ghost tries to pursue Pacman. Which way should it go and how should we program this?

The ghost finds itself on the other side of the wall (Figure 3). In a Euclidian sense the ghost is very close to the Pacman. In a topological sense the ghost is far away. The question at this point to the students is: how would you program this game? How can the ghost track the Pacman. Here are the categorized answers suggested by students:

1) ***Brownian Motion***: the ghosts should move around randomly. At each step in the game the ghost re-evaluates all the places it could go that are one step away. It will then pick one place randomly. Sooner or later they will find the Pacman. This is a very simple strategy to implement. Many games actually use this strategy. *Problems*: players will interpret the random walk as such. The ghosts do not appear to be intelligent at all.
2) ***Point and Shoot***: the ghost selects a random direction and keeps moving in that direction until it hits an obstacle. At this point it determines a new random direction. *Problems*: While the ghost appears to be less random than in approach #1 it still does not come across as intelligent because there is a good chance it will move in obvious ways away from the Pacman.
3) ***Incremental Approach***: the ghost should try to move closer. This also is done incrementally. At each step in the game the ghost will explore all the places it could go and select the one that is closest (in the Euclidian sense). *Problems*: This idea does not take obstacles into account and as such can be ineffective. In the situation in Figure 3 the ghost would be stuck at its current location because moving left of right would increase its distance to the Pacman.

A brief discussion of trade offs of each approach typically leads to a discussion how these ideas could be combined:

2a) ***Smart Point and Shoot***: The ghost selects a direction aiming at the Pacman. If the ghost hits an obstacle it will re-evaluate the direction to the Pacman again. *Problems*: If the direction is the same as the current direction the ghost gets stuck.

3a) ***Incremental Approach with Trap Detector***: If the ghost gets stuck by minimizing Euclidian distance it should change its strategy and pick a random direction. *Problems*: Without remembering where the ghost has been it will be difficult to make the decision when to switch back and forth between incremental approach and random move.

Object-oriented design just trapped us. The kind of discussion mentioned above can last for quite some time. As the discussion progresses it becomes clear to the students that while some approaches work better than others none of these ideas work well. After all, if there is a way from the ghost to the Pacman then the ghost should be able to find it. Instead, ghosts will wander around and get occasionally stuck. However, the direction of exploration appears to be clear. Better solutions require more sophisticated programming of the ghost. The ghost just needs to do a better search by having a more complex state, more methods, and more sophisticated ways to parse the scene. Perhaps all we need to do it to add a few transitions to our current UML diagram – or perhaps not.

The psychology of programming attempts to explain programmer's intuition. Syntonicity [31] is an introspective behavior that makes us want to be the ghost in order to think about what the ghost should do and how it should do it. More generally, body syntonicity makes programmers explore ideas that are compatible with their own feeling of being in a body. What would I do if I were this ghost?

From the viewpoint of a problem solving design space we have been lead to a local maximum. We found a solution that kind of works just not very well. The Pacman example of course only serves as illustration of the effect. After all, our goal is probably not to make the world best Pacman game. However, even in game design this more general pathway search problem is real and important. Of course there are well known solutions to solve the problem. A* algorithms [6] for instance do a good job at least to find non-moving targets.

This breakdown prompts us to think about reconceptualizing the problem. Perhaps it is not really the ghost agent that should be doing the heavy lifting at all. Perhaps the background tiles, which up to this point, we only considered to be passive background objects, i.e., decoration, with no behavior whatsoever should help somehow with the search. This is a case for *antiobjects*. As we shall see later, by removing computation from the agents that *appear to do the search* (the ghost) and redistributing it in a parallel fashion to the agents that define the space to be searched (the background tiles) we get a solution that is remarkably effective yet simple to implement. The challenge to find this solution is a psychological, not a technical one. An approach called Collaborative Diffusion will be described in detail in the following sections.

The ghost and floor tiles are examples of antiobjects. If there is such as notion of background and foreground we tend to think of the agents such as the floor tiles as background while the ghost agents are part of the foreground. Yet, as antiobjects their roles are reversed. The background does most of the work, the foreground just needs to react in simple ways to the computation of the background. This runs counter to our body syntonic experience. When we think of the Pacman game and think about objects such as floor tiles we do not envision any computation. We know, floor tiles do not compute in the real world, why should they in our game world? This is a hard question and perhaps the simplest answer is because they can. If we would build a physical embodiment of a Pacman game we would have a robot representing the ghost. Even if we wanted to, in the physical world we could not easily add computation to objects such as floor tiles. All we can do is to make our ghost robot really smart and equip it with good sensors.

In addition to reversing roles antiobject also distribute computation differently. The apparent computational intelligence of a single agent, for instance the ghost's ability to track down a moving target, is distributed to a potentially large set of agents. Instead of having one ghost doing the work, all floor tiles need to engage in highly parallel computation. As we shall see in the Collaborative Diffusion based Pacman implementation the floor tiles effectively diffuse the smell of the Pacman. All the floor tiles participate in a highly parallel diffusion computation. Meanwhile, the ghost agents can engage in simple hill climbing. Computationally speaking the floor tile antiobjects are doing practically all the work while the ghost antiobject is doing almost none.

We will claim in this paper that antiobjects work very well for Collaborative Diffusion, which is a general framework that can be applied to many game applications. Beyond the many simulations that we have built using Collaborative Diffusion ranging from AI pathfinding, bridge construction design and soccer simulations to the simulation of electric systems we have also successfully taught the use of antiobject to students. We started with teaching game design and development to graduate and undergraduate computer science students. Each time we faced that difficult moment to make the transition from what students intuition was to how we need to reconceptuallize the problem completely in order to solve it. Using UML diagrams did not help with this transition. However, once student made this step they have created wonderfully complex games with amazingly little effort. As we keep teaching we gradually introduced new visualization techniques to better illustrate the nature of antiobjects. Instead of using traditional debugging techniques such as method tracing we had to built tools that would be able to visualize in real time 3D plots superimposed over the game world. Using these techniques we tried to push the idea of antiobjects even further by trying to have kids in middle school to build AI-based games involving antiobjects. To our amazement that worked out well. Not only were we able to teach 11 year old middle school kids to create Collaborative Diffusion based games but the few kids that were introduced to the technique showed their friends how to do the same.

The following sections describe Collaborative Diffusion as an example of programming antiobjects. Antiobjects are a means to reconceptualizing computation. However, in addition to being different we will show that in the context of Game AI antiobjects exhibit a number of positive characteristics including effectiveness, incrementalism and robustness. To be able to make these points we will move from the more philosophical treatment of objects in this introduction to a more technical discussion comparing Collaborative Diffusion to related work in Artificial Intelligence.

2. ANTIOBJECTS & GAME AI

For some time now games have served as an ideal test bed to explore computational intelligence [24] and as motivational context for object-oriented design classes. As many have pointed out, the game industry is approaching the size of a multi-billion dollar movie industry. Games such as Halo 2 and Doom III have reached levels of colossal complexity. Hardware is advancing quickly as well. One of the next generation game consoles, the Sony PlayStation 3, is expected to deliver close to two Terra Flops of computational power. However, most of the computational power seems to be spent in either rendering complex scenes with extremely large polygon counts or computing physics. Before long, the rendering quality of games will produce photorealistic images that will be hard to differentiate from reality. But, what about Artificial Intelligence in games? The AI exhibited by many modern games is considered weak AI [25] in the sense that it is mostly concerned with practical engineering approaches to make machines appear to be intelligent without necessarily employing sophisticated computational schemes. Particularly weak are current implementation schemes involving multiple agents collaborating and competing with each other. This paper explores why it is hard to build collaborative agents and introduces the Collaborative Diffusion framework addressing these issues.

A quick survey of game developer resources indicates a rich presence of AI related topics. Numerous books, e.g., AI for Game Developers [3], and websites, e.g., gameai.com, are dedicated to what is colloquially called Game AI. Discussions found there are typically covering a spectrum ranging from simple state machine-based AI, over path finding to learning. There is surprisingly little coverage on general techniques that could be used to implement collaborative and competitive multi agent games. One possible explanation is that the apparent complexity of collaboration AI schemes found in academic research is simply exceeding the

general scope of game AI. In other words, it may simply be too hard to transfer academic research approaches dealing with collaborative agents to the more pragmatic world of game AI because of practicality issues. Additionally, it may also be possible that game contexts may impose additional constraints and cause scaling challenges difficult for existing AI approaches to deal with. These concerns are the motivation behind our approach. Instead of conceptualizing Game AI as the pragmatic cousin of AI new frameworks directly relevant to games must be explored. Despite this focus on games many of the findings will be relevant to general AI. Games are more than just test beds for AI approaches; they also become a conceptual development space yielding new notions of computational intelligence that can be transferred, for a change, from Game AI to regular AI.

Compared to rendering and physics simulation, Game AI has much room for improvement. Players are beginning to demand more refined game play based on sophisticated AI. Some of the most frequently played games today are based on the idea that the player is manipulating a game character, through first or third person view, exploring some intricate space populated with "enemy" agents. As part of a mission the player needs to reach a variety of goals also located in that space. The enemy agents will try to prevent the player from reaching said goals by trying to block ways, wound or even kill the player. The player, in turn, has a repertoire of weapons that can be used to eliminate enemies. What is missing in this type of game is collaborative and competitive behavior. The AI found in most is by no means trivial. They can locate the player and put up the right level of resistance making ultimately a game fun to play. However, their behavior, with few exceptions, is overly autonomous. They are typically not collaborating with other enemy agents in noticeable ways. Collaborating enemy agents would make the game play more interesting. The player will get the opportunity to deal with more complex levels of behaviors not only at the level of individual enemy agents but also at the level of group behaviors. Especially in game contexts this collaboration-based enemy agent advantage will have to be managed carefully. After all, a game featuring agents that are so smart that they immediately beat the player would be an aggravating game playing experience. To balance game play again in favor of the player additional agents could be introduced to a game collaborating with the player and competing with enemy agents.

Collaborate Diffusion is a framework based on antiobjects that can be used to build games with large numbers of agents collaborating and competing with each other. One particularly attractive characteristic of Collaborative Diffusion is how its simple implementation results in sophisticated emergent behavior. Figures 11-13 show collaborative behaviors in the context of a soccer game. Players from the same team are truly collaborating with each other and competing with all the members of the opposite team. For instance, if there is a player in a better position than a player covered by an opponent player, the covered player will pass the ball to that other player of his team. An intriguing aspect of this behavior is that it had not to be "programmed" into the agents but, instead, was an emerging property of the soccer Collaborative Diffusion process. The soccer game only serves as an example exhibiting collaboration and competition. The applicability of Collaborative Diffusion is not limited to soccer-like games or even games in general. Collaborative Diffusion has been applied to various complex optimization problems including applications such as bridge design [23], mudslide, electrical circuits, avalanche and traffic flow simulations. Common to all of these applications is the process of diffusion.

Architecturally speaking, antiobjects and Collaborative Diffusion grew out of some of our multi-agent scientific simulations running on a Connection Machine CM-2. Powered by sixty four thousand CPUs connected through a twelve dimensional hypercube the Connection Machine achieved unprecedented performance of about 2 Giga Flops. Much more important, from today's point of view, than computational performance were the conceptual insights gained from using the Connection Machines. The idea of massively parallel problem solving provides radically difference perspectives that can lead to new conceptual frameworks. Antiobjects, an example of such a framework, clash with traditional notions of object-oriented design in the sense that object-orient design does not help with the understanding or use of these new computational frameworks.

Today, thanks to the rapid advance of hardware, these frameworks can be implemented on more traditional sequential architectures. Ultimately, CPU cycles will always become cheaper than cognitive cycles. Just a couple of years ago Collaborative Diffusion would not have been computationally feasible to build consumer oriented games capable of running on desktop computers. Now that a $400 PlayStation 3 roughly has the floating-point power of 1000 Connection Machines we can allow ourselves to reconceptualize computation.

3. BASIC DIFFUSION

Collaborative Diffusion is a versatile collaboration and competition framework for building multi-agent games based on antiobjects. Agents such as the ones used in AgentSheets [23] live in a space (2D or 3D). Typically, the space is organized as a grid similar to a spreadsheet. The grid contains agents representing game characters, e.g., soccer players, but also the environment containing these characters, e.g., the soccer playfield. Agents may be stationary or mobile. To collaborate and compete with each other, character agents, as well as environment agents, jointly engage in one or more diffusion processes.

Game design patterns [2] help to conceptualize agent behaviors for typical game applications. Common to many games is the notion of pursuit and evasion. In Pacman ghost are pursuing Pacman, in the Sims, Sims characters are pursuing other sims, sources of food, entertainment, and so on. Agents participating in pursuit can be categorized into the following roles:

1) ***Goal Agents***: Goal agents are pursued by other agents. Goal agents may be static or mobile. For instance, a refrigerator is a static goal, typically located in a kitchen, pursued by an agent representing a hungry person in a "The Sims"-like game [32]. A soccer ball, in contrast, is a mobile goal attracting soccer players.

2) ***Pursuer Agents***: One or more pursuer agents may be interested in certain goal agents. Multiple pursuer agents sharing interest in the same goal may collaborate or compete with each other. If there are multiple types of goals, such as food, and entertainment, then the pursuer agent includes a goal selection mechanism determining the priorities of goals being pursued.

3) ***Path Environment Agents***: A path environment agent enables a pursuer agent to move towards a goal agent. In a household simulation a path environment agent may represent a kitchen tile, or a piece of carpet. A path environment is an active agent participating computationally in the diffusion process that is helping pursuer agents to locate goal agents.

4) ***Obstacle Environment Agents***: Like path environment agents, obstacle environment agents are part of an environment but they deter pursuer agents from reaching their goals. Walls, closed doors, fires, fences, and rivers can all interfere with the pursuer's attempt to reach a goal. Interference can be at different levels. A wall may

permanently prevent a pursuer from reaching its goal while a piece of furniture in the middle of a room may just pose a simple delay caused by the need to navigate around the furniture.

This categorization scheme can be applied to nearly all arcade style games ranging from early classics such as Space Invaders, and Pacman to modern titles such as Halo 2 and Doom III. Although the agent categorization scheme is mostly relevant to games the Collaborative Diffusion framework can be applied to a much wider domain of applications.

3.1 Single Diffusion

A game is built by defining the four kinds of agents described above and arranging instances of them in a grid-structured worksheet. The worksheet shown in Figure 4 contains a matrix consisting of 9 rows x 9 columns of floor tiles serving as environment path agents. All examples, including the diffusion visualizations shown here are built in AgentSheets [23].

A single Pacman will be the single goal agent located in the center of the worksheet. The Pacman is a user-controlled agent pursued by all the autonomous ghost agents.

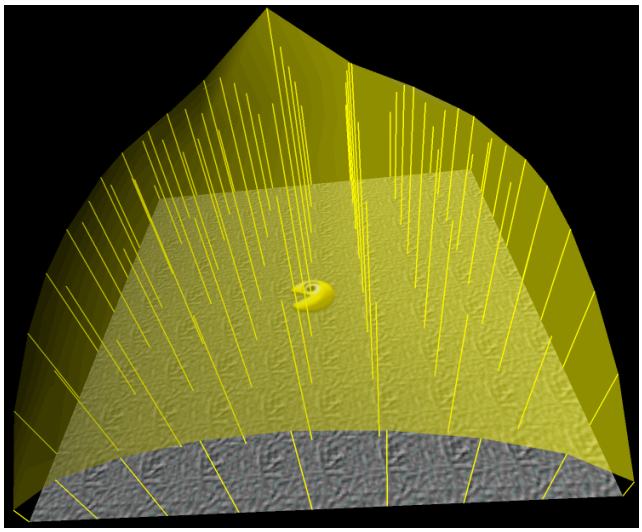

Figure 4. Worksheet with 9 x 9 floor tile agents and one Pacman agent in the center. The Pacman "scent" is diffused by the tile agents. The diffused value is shown as logarithmic plot over the worksheet area.

Diffusion is a gradual process in which physical and conceptual matter, such as molecules, heat, light, gas, sound, and ideas are spread over an N-dimensional physical or conceptual space over time. Alan Turing was one of the first researchers to note the broad impact of diffusion processes onto chemical and biological phenomena and became the first serious user of an electronic computer in order to model these diffusion processes [30]. He was interested in mathematically capturing reaction-diffusion systems and defined diffusion to occur when "each [chemical agent] moves from regions of greater to regions of less concentration."

Agent-based diffusion has been used in a number of applications including image feature extraction [14] and more recently distributed optimization [29]. Collaborative Diffusion does not diffuse discrete objects, i.e., agents, but uses agents to diffuse, track and modulate continuous diffusion values.

Diffusion values are used to spread the "scent" of the Pacman goal agent in the worksheet. The Pacman agent is given an attribute called *p* short for Pacman scent with an arbitrary but constant value. This value represents the desirability of a goal. High desirability is captured with a large value (e.g., 1000). Desirability may also assume negative values. Pursuer agents will actively avoid goals with negative desirability. This attribute will be diffused through the floor tile agents using a discrete version of the well-known general diffusion equation [8]:

Equation 1: Diffusion Equation

$$u_{0,t+1} = u_{0,t} + D\sum_{i=1}^{n}(u_{i,t} - u_{0,t})$$

where:
n = number of neighboring agents used as input for the diffusion equation
$u_{0,t}$ = diffusion value of center agent
$u_{i,t}$ = diffusion value of neighbor agent (i > 0)
D = diffusion coefficient [0..0.5]

The diffusion coefficient controls the speed of diffusion. A larger value will result in quick diffusion. In material science each material has a specific heat diffusion coefficient. Silver, for instance, has a much larger heat diffusion coefficient than a thermal insulator such as wood. Consequently Silver will spread heat much more rapidly than wood.

In a two-dimensional application the input is limited to the four immediate neighbors defined by the von Neumann neighborhood [28]. The Environment Path agents consisting of all the floor tiles will now each compute the diffused values of Pacman scent *p*. Figure 1 shows a logarithmic 3D plot of the *p* values over the area of the entire worksheet. The peek in the middle correlates with the position of the Pacman and has a value of 1000.

In the two-dimensional case, and with von Neumann neighborhood, a diffusion coefficient value of 0.25 represents a special case that further simplifies the diffusion equation. For D=0.25 a new diffusion value u will simply become the average value of its neighbors without even the need to refer to its own previous value. This is useful for quick spreadsheet-based experimentation.

Using a simple hill-climbing [7] approach, the pursuer agents track down the Pacman. Each pursuer agent compares the diffusion values of its four von Neumann neighbors and moves to the neighbor cell with the largest value.

So far this approach would not provide a significant advantage to more traditional tracking approaches. For instance, a pursuer agent could have determined the location of the most promising neighbor by computing the Euclidian distance from each of the neighbors to the goal and then selecting the neighbor with the smallest distance. However, in the presence of walls and other kinds of environment obstacle agents these simplistic Game AI approaches fail quickly. Figure 5, shows a complete playable version of a Pacman game.

Through the diffusion of the Pacman scent the ghost will find the correct solution. The important part is to understand how environment obstacles agents interact with the diffusion values. The walls, which are also represented by agents, will not diffuse the Pacman scent. Consequently, they will clamp down diffusion values. This clamping results in complex diffusion landscapes that, again, can be plotted in 3D (Figure 5). Conceptually these landscapes overlap with Axelrod's Landscape theory [1] but are better suited for dynamic schemes that include mobile agents. As the Pacman and the ghosts are moving around in this world, the landscape is continuously changing. But no matter where they are, the ghost will be able to track down the Pacman, as long as there is a path for a ghost to reach the Pacman.

The same approach works for tracking multiple goals. If multiple goal agents have the same value then the pursuers agent will track the closest goal agent. Goal agents may have different values representing desirability of the goal. For instance, if really good deli food is represented with a high "food" value, whereas some old sandwich is represented with a significantly lower "food" value then a pursuer agent is likely to pursue the more attractive source of food even if it is located further away. The desirability can also be a dynamic function. If, for instance, a pursuer agent reaches food, it may eat some or all of the food resulting in a decrease of the food's desirability.

Collaborative Diffusion shifts a great deal of the computational intelligence into the environment. Although, to the observer, the ghosts appear to be the intelligent entities because they can track the Pacman incredibly efficiently, in reality, most of the intelligence is in the environment. The environment becomes a computational reflection of the game topology, including the player location and their states.

Today low-end desktop computers are sufficiently powerful to compute diffusion of moderately sized worksheets. The Pacman implementation had to be dramatically slowed down to make it playable by a human. Without slowing it down, the ghosts will instantly track the Pacman leaving a human player no chance. All the examples shown in this paper run in real time. This includes the sub-symbolic, symbolic processing, agent animation, and the visualization of the 3D diffusion surface plots. All the Figures in this paper are unedited screen dumps showing running games with real-time 3D visualization enabled.

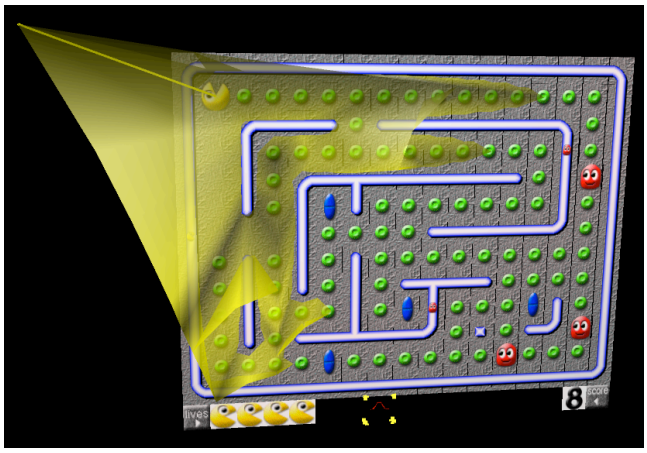

Figure 5. Collaborative Pacman Game. The user controlled Pacman is in the upper left corner. The Pacman "scent" is diffused over the entire worksheet. Diffusion values are plotted logarithmically. The plot is intersected by the worksheet. Walls are obstacle agents with a zero diffusion value.

3.2 Collaboration by Goal Obfuscation

Even the simple version of diffusion introduced so far, not including any explicit notion of collaboration, can lead to emergence [10]. The Pacman game serves as an example of an emerging collaboration called *Collaboration by Goal Obfuscation*.

Ghosts tracking down a Pacman can collaborate with each other by interacting with their environment. Consider the case where two ghosts are tracking a single Pacman that tries to hide in a corner of a maze (Figure 6 left). The two – or more – ghosts will always split up making it impossible for the Pacman to escape. The first ghost will select the shortest path, which is usually a good approximation of a human walking path [4]. No matter which path the first ghost takes the second one will select the other path (Figure 6 right). Through the environment the first ghost collaborates with the second one essentially communicating: "I will take care of this path, you take care of the other one."

Just how do the two ghost collaborate with each other? In contrast to spreadsheets where each cell only contains one value, an AgentSheets cell contains a stack of agents, e.g., a ghost agent on top of a pill agent on top of a floor tile agent. An agent evaluating a diffusion equation referring to a diffusion value of one of its neighboring cells will "see" the value of the agent on top of that stack. If that agent does not have a matching attribute it will return 0. Consequently, if a ghost agent does not compute its own Pacman diffusion value it will contribute a zero value to Pacman diffusions surrounding it. In narrow, single cell, corridors such as the ones found in Figure 3 the Pacman scent is blocked as it cannot flow around the ghost. This results in optimal collaboration cornering the Pacman as efficiently as possible.

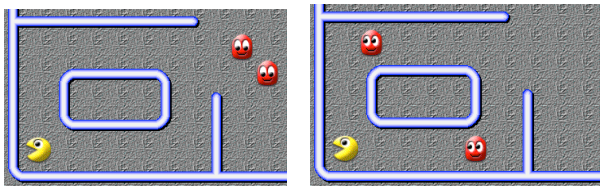

Figure 6. Two ghosts collaborate. They enter the maze (left), and split up to slay the Pacman (right).

This collaboration scheme is an emergent one in the sense that there is no code explicitly responsible to orchestrate a group of pursuing agents. There is no notion of collaboration neither at the sub-symbolic diffusion level nor at the symbolic goal selection level. Somewhat astoundingly, the same scheme generalizes to more complex cases such as the one described below (Figure 4). Five pursuers attack a single goal. Again without the use of centralized scheme the ghosts manage to spread out perfectly sending each pursuer into a unique path leaving the Pacman no chance (Figure 7 right). This behavior is optimal because it tracks the Pacman as efficiently as possible. If even just two ghost agents would follow each other in the same path they would leave the Pacman a path to escape.

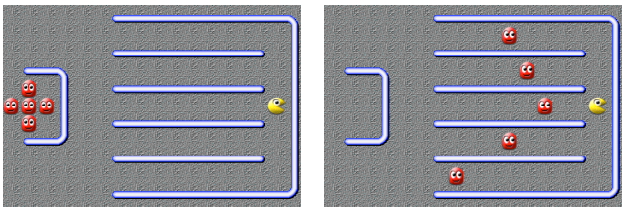

Figure 7. Five ghosts collaborate. They start grouped up (left), and distribute into all five paths (right) leaving the Pacman no route to escape.

Depending on the starting conditions, it is possible that initially two pursuing agents will end up in the same path because the only path left over may be far away, making a nearby but already handled path relatively attractive. But even in this case the redundant pursuer agent will recognize its mistake quickly, turn around and move to a path not yet featuring a pursuer agent.

Traditional AI search approaches such as A*, including incremental extensions such as LPA* [12], would have been efficient in a sequential scenario where only one pursuer gets sent at a time. In the parallel scenario each pursuer agent would have to restart its path finding search after each step. This parallelism is likely to impair the plan of each pursuer agent. Additionally the

goal agent may be moving as well. The net effect is that incremental heuristic searches such as LPA* are likely to loose most of their incrementalism when attempting to deal with collaboration. To make things worse, the computational cost becomes proportional to the number of pursuer agents. Collaborative Diffusion, in contrast, remains incremental and has constant computational cost.

4. COLLABORATIVE DIFFUSION

Up to this point I have shown how multiple pursuers can track down one or more goals by using sophisticated path finding and simple collaboration. How can one add more explicit collaboration and competition control mechanisms to diffusion and deal with more complex goal structures? Soccer is a game that includes additional challenges pushing the idea of Collaborative Diffusion to the next level.

In its simplest incarnation one can define a soccer game as a simulation of two teams with a certain number of players in each team. This particular implementation of this game was developed for the 2002 World Cup Soccer championship and demonstrated in Tokyo. Our soccer players have a two level goal structure. Initially, if they are away from the ball then they will try to track down the ball. Players from both teams track the same ball. Once they reach that goal they will have opposing objectives regarding where to kick the ball.

Figure 5 below shows a complete soccer game simulation with eleven players in each team. The red team is playing from left to right, and the blue team is playing from right to left. The ball, the left, and the right goals, are the Goal Agents. The red and blue team agents are the Pursuer Agents tracking the ball and the goals.

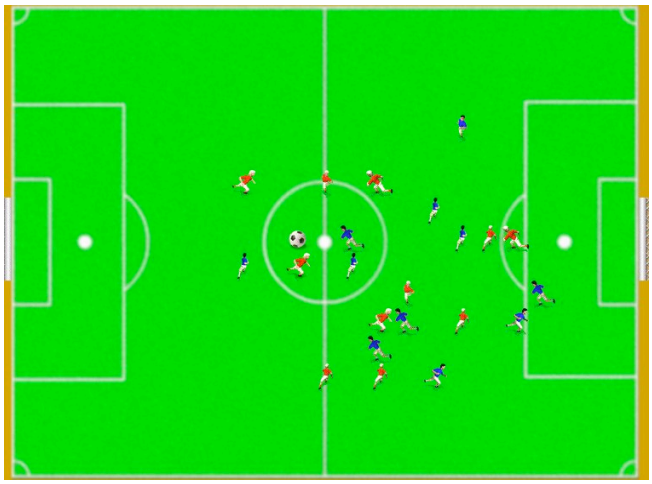

Figure 8. Soccer simulation of Germany versus Brazil.

The playground is covered with about 500 playground agents serving as Environment Path Agents. Playground agents diffuse three different diffusion values (ball, left goal, and right goal) using equation 1. In order to express collaboration and competition the player agents will also diffuse these values but will modulate the surfaces using the Collaborative Diffusion equation 2.

Equation 2: Collaborative Diffusion

$$u_{0,t+1} = \lambda\left[u_{0,t} + D\sum_{i=1}^{n}(u_{i,t} - u_{0,t})\right]$$

$\lambda > 1$: collaboration
$\lambda < 1$: competition
n = number of neighboring agents used as input for the diffusion equation
$u_{0,t}$ = diffusion value of center agent
$u_{i,t}$ = diffusion value of neighbor agent ($i > 0$)
D = diffusion coefficient [0..0.5]

When diffusing the Left Goal value, the blue agents use a $\lambda > 1$ to express collaboration, and the red agents use $\lambda < 1$ to express competition. Collaboration and competition manifest themselves as positive and negative dents on the Left Goal diffusion surface (Figure 6).

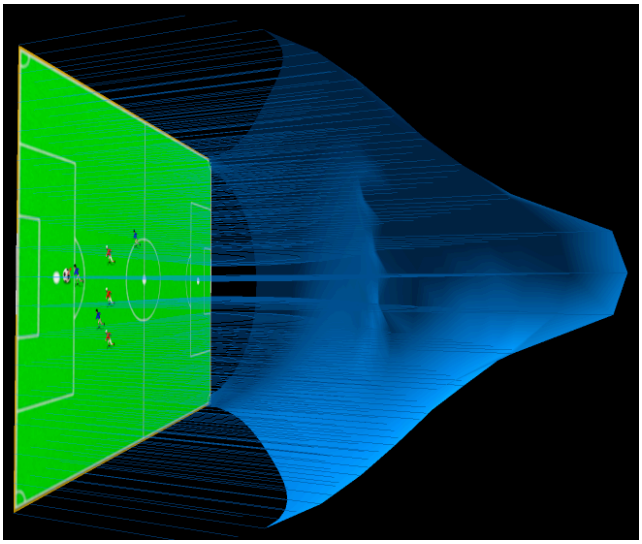

Figure 9. Left Goal diffusion. The peak of the diffusion correlates to the position of the left goal. The dents in the diffusion surface correspond to collaborating and competing players modulating the surface.

Symmetrically, when diffusing the Right Goal value the blue agents use a $\lambda < 1$ to express competition, and the red agents use $\lambda > 1$ to express collaboration. For an agent A to collaborate with an agent B with respect to a goal G means that B will have a positive effect on the diffusion surface representing G at the location of B. If A and B are soccer players from the same team, and A is in the possession of the ball, but B is in a better position to kick the ball towards the goal then *A is collaborating with B* by deciding to pass B the ball instead of trying to kick it toward the goal itself.

The degree of collaboration and competition is controlled via λ. λ values (table 1) close to 1 denote small collaboration and competition versus λ values that are significantly larger or smaller than 1 represent a more pronounced collaboration and competition.

A λ value of 1 denotes the lack of collaboration or, in other words, total autonomy. Large λ values yield exaggerated collaboration manifesting itself, for instance, in players sending each other passes when they should have attempted to just kick directly toward the goal. Analogously, very small λ values result in extreme competition.

The ability to control agent interaction through λ anywhere between extreme competition, competition, autonomy, collaboration, and extreme collaboration (Table 1) makes λ an important parameter of the sub-symbolic computational intelligence. At the symbolic level rules can be used to modify λ in real time. This is useful to implement dynamic collaboration schemes where the degree and quality of collaborations may change over time.

Table 1. Agent Interaction

λ	Agent Interaction
>> 1	Extreme Collaboration
> 1	Collaboration
= 1	Autonomy
< 1	Competition
<< 1	Extreme Competition

The values used for λ in the soccer simulation shown here are 1.08 for collaboration and 0.8 for competition. These values often need to be determined empirically. Through the end-user programming interface, developers can control λ values at the level of an entire simulation, agent classes, or individual agents in order to simulate different kinds of players.

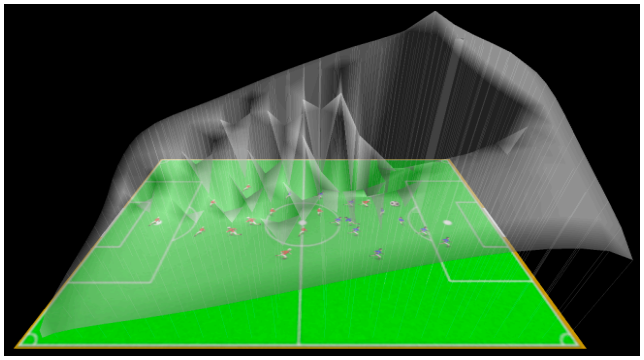

Figure 10. Ball diffusion. Peak indicates the location of the ball.

Player agents do not diffuse the ball value. This results in effects similar to the Collaboration by Goal Obfuscation presented previously in the Pacman example. The consequential ball diffusion surface becomes very complex (Figure 10). If the player agents would also diffuse the ball value then the ball would immediately attract all the players resulting in one large cluster including all the players.

Goal Selection

To adapt to different situations, sub-symbolic diffusion processes need to be controlled via a goal selection mechanism. This is an opportunity for end-user programming allowing users to create rules determining when and how agents select goals. A soccer player agent that is not next to the ball will initially pursue the ball as goal. Because of the Collaboration by Goal Obfuscation effect, the urge to track the ball will decrease if the ball is already surrounded by a large number of players. As a result of that, after a while some of the players will spread out on the play field. Some will stay behind in defending positions while others move beyond the ball in an attacking posture ready to receive passes from their fellow players (Figure 8).

Once a player reaches the ball, a new goal is selected. The new goal is to kick the ball, in accordance to the corresponding goal diffusion function, towards the opponent's goal. The Collaborative Diffusion again shifts the intelligence into the environmental computation consisting of the Left Goal, Ball, and Right Goal diffusions. Goal selection becomes simple. All the player has to do is to make the ball hill climb, one step, towards the opponent's goal.

More generally a goal selection can be implemented using different approaches ranging from simple computational models such as state machines, goal stacks and goal queues to more complex neurologically inspired approaches such as Subsumption architectures [15] or psychological approaches such as Maslow's hierarchy of needs [16]. Furthermore, goal selection can be done fully autonomously, user controlled or in a mixed user + system mode. An example of a mixed mode goal selection mechanism can be found in The Sims [32] where users select from actions (e.g., go and eat something) that are put into a goal queue but at the same time the system may add important actions based on the current state of a Sim.

Scenarios

The ensuing game simulation is highly complex because every player from each team influences the decision of every other player. In other worlds, through the Collaborative Diffusion all the players communicate with each other through the environment. The consequences of this collaboration/competition scheme are best illustrated through simplified scenarios. Scenarios are created by manually moving players and the ball into specific situations in "AI debugging" mode. In debugging mode players will not move around nor actually kick the ball. Instead, the scene will be visually annotated with information indicating the intentions of players. For instance, the visualization will show a little arrow pointing in the direction in which a player would have kicked a ball. Without this tool it can become extremely difficult to track agent interactions especially when running the game in real-time.

Passing: The red player possessing the ball (Figure 11) is getting close to the blue goal with only one defending blue player. The little red arrow indicates the intended ball kick direction. The player is not kicking the ball straight to the goal but instead is sending it as a pass to a second player closer to the goal from the red team.

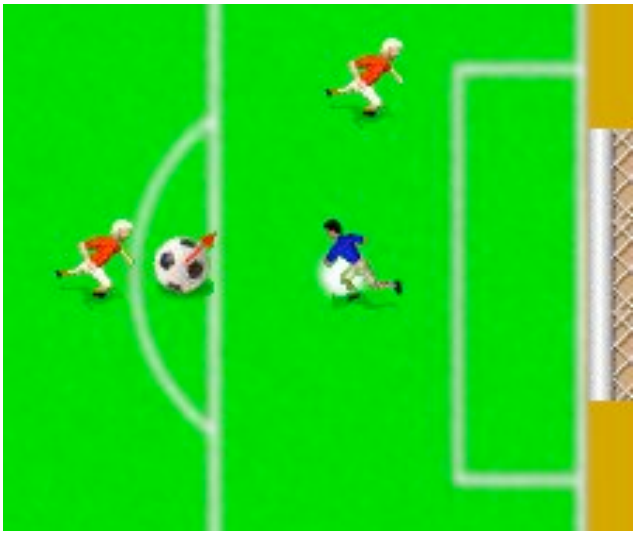

Figure 11. Collaborate by passing ball.

Sending such a pass is emerging behavior that was not programmed in any way. The system has no rules for sending a ball from one player to another. The passing behavior emerges from the collaboration/competition modulation. Sending the ball to a second player that is not covered by a player from the

opposite team is a good idea despite the fact that the total distance the ball has to travel is longer than a straight shot.

Path of less opponent coverage: Three red players attack blue goal (Figure 12), which is defended by three blue players. The kicking red player sends a pass to the lower red player, which is only covered by one blue player.

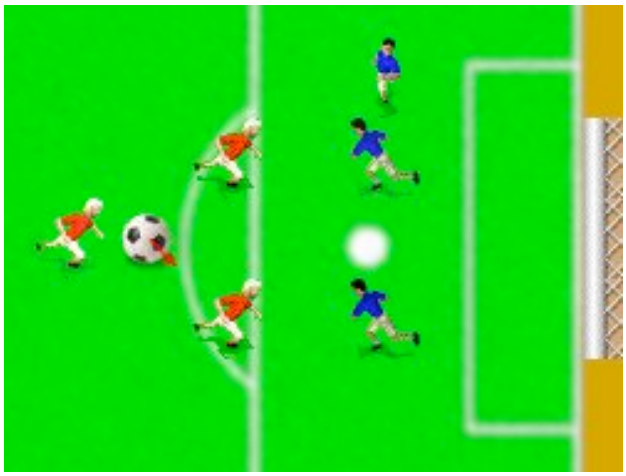

Figure 12. Player passes ball to less covered player.

Straight shot: With one more blue player making the situation symmetrical (Figure 13), both other red players are covered by two blue players each. The kicking red player decides to attempt a straight shoot towards to goal instead of sending a pass to either of his colleagues because both of them are covered well.

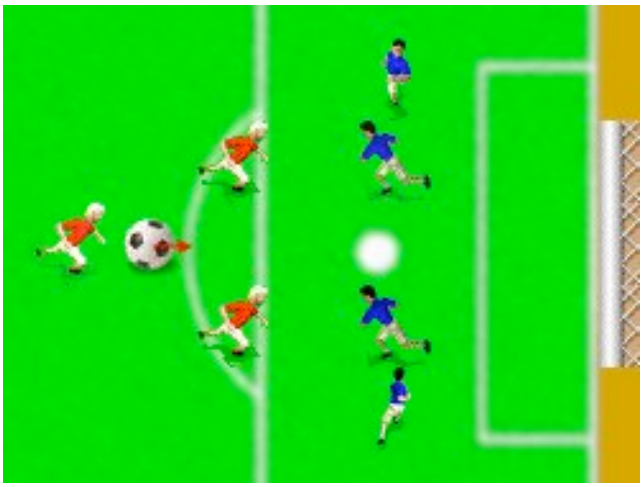

Figure 13. Players from same team are well covered. Player kicks ball straight towards goal.

5. DISCUSSION

This section describes the main characteristics of Collaborative Diffusion and discusses extensions and experiences.

Robust. One of the main goals of this research is to create a robust AI framework for developing real-time multi-agent game applications. The collaboration aspect of the framework should be robust in the sense that it should be independent of the context in which it is used and should work with a highly variable number of agents. Higher-level intelligence, including collaboration, should emerge from the interaction of simple agents. This degree of robustness is unlikely to result from symbolic-only AI approaches. This robustness can be achieved through a hybrid architecture fusing sub-symbolic with symbolic AI approaches. At the sub-symbolic level, diffusion processes are employed to compute complex collaboration goal functions. At the symbolic level, a rule-based language, called Visual AgenTalk [23], is used to control these diffusion processes. Game developers will be able to benefit from end-user programming [5, 11, 13, 17, 19, 23] and end-user development [20] to control the behavior of rule-based agents. End-user programming allows developers to easily define agent goal selection and to adjust sub-symbolic collaboration and competition parameters. Finally, a robust framework aimed at end-user developers requires the integration of specialized debugging tools capable of illustrating the complex interaction emerging from collaborative multi-agent applications. An initial step in this direction is the integration of real-time 3D visualizations in the diffusion process.

It has been somewhat surprising to see the robustness resulting from the combination of symbolic AI and sub-symbolic AI in Collaborative Diffusion. The soccer game got exposed to a number of elementary and middle school kids in the USA and Japan. Many of these kids created outrageous configurations of soccer games. Some tried extreme team size ratios (e.g., two against fifty); others explored the behavior of players when given multiple balls. In some cases there were more balls than players. Others eliminated goals temporarily, moved the goals to unconventional positions or even added multiple goals. In all of these situations the players kept "happily" playing by engaging in what at least appeared reasonable strategies to deal with unanticipated game modifications.

Environmental. Traditionally, Artificial Intelligence projects computational "intelligence" into the components of a simulation that most people would perceive to be intelligent. In the case of the soccer game the AI would be presumed to reside inside the agents representing soccer players. Collaborative Diffusion, in sharp contrast, projects most of the computational intelligence onto the environment. In the case of the soccer game that would be the soccer field. In the spirit of antiobjects, Collaborative Diffusion swaps the computational roles of active foregrounds and passive backgrounds. For instance to simulate flocking birds [9, 26, 27], traditional AI puts the birds into the foreground by projecting computational intelligence into agents modeling the behavior of birds. Collaborative Diffusion puts what is traditionally considered the background, e.g. the environment containing the birds, into the computational foreground. Simon's notion of embedding artifice in nature [26] is highly related. He employs the parable of an ant to explain the psychology of thinking: "An ant, viewed as a behavior system, is quite simple. The apparent complexity of its behavior over time is largely a reflection of the complexity of the environment in which it finds itself." Antiobjects in Collaborative Diffusion go one step further by not only shifting the perception of intelligence from the agent to its environment but also by shifting the computational intelligence itself from the agent to its environment. The artifice, to use Simon's term, now becomes a fully computational entity itself. An intrinsic limitation of this framework is that it applies only to the world of the artificial where computation is at the discretion of the developer and can be put anywhere. There is no equivalent in the real world. Soccer fields and the air surrounding birds do not participate in any kind of computation accessible to us. For instance, Collaborative Diffusion is not directly relevant to robotics because in robotics there is no other way than putting all the computational intelligence inside the robots.

Parallel. In contrast to the typical kinds of problem solving approaches discussed in symbolic Artificial Intelligence for games such as checkers and chess, Collaborative Diffusion is concerned with the parallel execution of autonomous agents. Approaches such as game trees and minimax procedures make a number of

fundamental assumptions including: two-person game, turn taking, and perfect information [7]. There is no turn taking in a game like soccer. Also, all the players are acting – to a degree – autonomously and in parallel. The information is not perfect because, for instance, balls kicked may not end up where they were supposed to go.

Increasingly, as CPUs can only live up to Moore's Law buy using multi-core chips [21], we need better ways to map computation onto parallel architectures. Traditional, for instance thread-based, parallelism has only a limited capability to optimally benefit from parallel architectures because distributed computation is often offset by the need to deal with additional communication and synchronization challenges. Collaborative Diffusion is not only suitable for parallel AI problems but its computational nature also lends itself to simple parallel implementations. There is no need for a 64 thousand CPU Connection Machine with a 12 dimensional hypercube bus. Most Collaborative Diffusion game applications will only need two or three dimensional diffusion arrays. Segmenting the diffusion space and distributing it onto multiple CPUs achieves efficient data parallelism. This will result in near linear acceleration on multi-CPU or multi-core desktop computers.

Acceleration through parallelism is most pronounced on next generation game consoles combining multi CPU and multi core options. For even larger hardware acceleration the diffusion process can be executed on programmable Graphical Processing Units (GPU), which are common on modern desktop computers. Harris et al. [8] report a speedup of nearly 25 (CPU Pentium 4, GPU GeForce 4) for running very large diffusion processes with one million data points on graphics hardware. These are strong indicators that Collaborative Diffusion is computationally scalable. Even on a single CPU and without hardware acceleration it can be used to implement sophisticated collaborations based on a few hundreds or thousands of agents. The kind of acceleration possible on next generation game consoles allows Collaborative Diffusion with millions of agents were each agent could participate, simultaneously, in a large number of diffusion layers.

Incremental. The computation used in searching strategies is not well suited for frame-based, real-time games [18]. Most search-based strategies will, at each turn in the game, re-process the entire state of the game, that is, the positions of all the chess pieces, and compute a set of separate data structures such as and/or graphs consisting of extrapolated variants of the current game state. These graphs are searched and pruned using evaluation functions and pruning heuristics to cope with the complexity of analysis. However, and this is what makes these approaches non-incremental, all the computation states get completely discharged after each turn. At each turn the game board is treated like a completely new game board despite the fact that in most games – including chess – game boards only evolve incrementally from one move to another. In the Collaborative Diffusion framework the agents representing the game players and the environment will retain the game information and will compute game evaluation functions incrementally. In a frame-based game environment this means that the current computation will benefit from all the computation preceding it. The net effect is that less computation will be necessary for each frame allowing complete collaboration and competition AI to be computed in real time. Incremental AI approaches are beginning to gain interest. Koenig et al. [12] advocate the use of incremental heuristic search approaches for route planning in traffic and game applications. In contrast to Collaborative Diffusion incremental heuristic searches such as Lifelong Planning A* do not scale well to multi agent scenarios and do not include notions of collaboration and competition.

Lessons learned from Teaching

Collaborative Diffusion has been created at AgentSheets Inc. and been used in teaching at the University of Colorado, Computer Science Department, in several courses including Artificial Intelligence, Object-Oriented Design, and Gamelet Design for Education.

When learning to make AI-based games students generally did not have problems at a conceptual level after overcoming the initial transition from objects to antiobjects. That is, most students quickly grasped the idea of diffusion and its application to AI. Showing the real-time 3D diffusion visualization in class turned out to be rather important. It was especially important to convey the ideas of value distribution resulting from different topologies including complex mazes. Additionally, seeing the dynamic aspect of diffusion also helped. For instance, the use of different diffusion coefficients becomes highly apparent when visualized in real-time while observing moving targets. With small coefficients the resulting shape change are reminiscent of high viscosity oil whereas with large coefficients the shape change will be nearly instant. Coefficients approaching the critical value of 0.5 will also become apparent as numerical instabilities will manifest themselves as highly dynamic noise. Not all versions of AgentSheets included 3D real-time plotting capability. It became clear that students using a software version of AgentSheets not including 3D real-time plotting were much more likely to either get stuck due to even small errors in their diffusion equations or more likely to produce simple games that were only trivial extensions of the examples given to them. All this can be interpreted as early evidence that the real-time visualization is an important aspect of the AI development process. Originally, we assumed that the use of real-time 3D visualization would only be important for us, the framework developers, but our experiences with students indicate that game developers gain perhaps even larger benefits from using real-time 3D visualization to "debug" their AI applications.

One of the largest teaching challenges for Collaborative Diffusion was the transition from objects to antiobjects itself. Few computer science students had a concrete sense of what diffusion is. Even fewer students had any kind of previous exposure to computational interpretations of diffusion. Nonetheless, by using relevant examples diffusion is quickly explained. Using an end-user programmable application such as a spreadsheet or AgentSheets helps to illustrate the computational aspects of diffusion. Equations 1 and 2 presented in this paper are non-intimidating to computer science students especially when viewed as spreadsheet formula expression. More challenging, by far, was the notion of environmental computation. The idea that the computational intelligence is put into the environment, e.g., the soccer field between the soccer players or the air between the flocking birds, is non-intuitive. Student's previous exposure to AI concepts neither helped nor hindered their initial comprehension of antiobjects or Collaborative Diffusion. However, students with AI background could quicker relate to the affordances of Collaborative Diffusion because they appreciated the unwieldy complexity that would have likely resulted from using more traditional AI approaches such as rule-based AI to implement collaborative behaviors.

We were surprised by the variety and depth of some of the student developed games. Although many students created relatively minimal games that, by in large, were permutations of Pacman-like games, others created sophisticated games. A number of students built functionally complete, but artistically simple, Sims games including complex house layouts and diffusion layers such as hunger, entertainment, hygiene and relaxation.

Early user experiments were conducted at a middle school to establish the boundary conditions of usability. Would it be possible to convey the notion of diffusion to 12 year old middle school students allowing them to make their own games based on diffusion? To teach diffusion it was necessary to have one-on-one tutoring sessions that took about 20 minutes each. Compared to the computer science undergraduate and graduate students the middle school students' comprehension of diffusion was much more shallow. In cookbook style fashion they had quickly learned how to setup agents with diffusion equations so that they could build AI-based games. Nonetheless, the students not only picked up diffusion but also built some of the most creative games.

6. CONCLUSIONS

Object-oriented design can be deceiving by luring us into computational ideas that are too much inspired by our own experiences in the physical world. Based on these experiences we make decisions regarding in which objects computation should reside. In applications such as Artificial Intelligence this can be particularly detrimental as it may result in solutions that are inefficient or, worse, may not work at all. Antiobjects seemingly defy current expectations of what objects are supposed to do. They challenge our intuition regarding where computation should be. When implemented as antiobjects, object that we assumed to be complex turn out to be simple and have little computation. Objects we assumed to be passive, such as non-functional background objects, turn out to host most of the computation.

In this paper we have shown an example application of antiobjects called Collaborative Diffusion. Collaborative Diffusion is a versatile collaboration and competition framework for building multi-agent games. Antiobjects are not only a way to think differently about computation but, as illustrated in the case of Collaborative Diffusion, they can make applications more effective, incremental and robust.

7. ACKNOWLEDGMENTS

This material is based in part upon work supported by the National Science Foundation under Grants Numbers 0349663 and 0205625. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

8. REFERENCES

1. Axelrod, R.M. The Complexity of Cooperation. Princeton University Press, 1997.
2. Bjork, S. and Holopainen, J. Patterns in Game Design. Charles River Media, Hingham, Massachusetts, 2005.
3. Bourg, D.M. and Seemann, G. AI for Game Developers. O'Reilly, 2004.
4. Brogan, D.C. and Johnson, N.L. Realistic Human Walking Paths. Proceedings of the 16th International Conference on Computer Animation and Social Agents (CASA 2003) (2003).
5. Cypher, A. Watch What I Do: Programming by Demonstration. The MIT Press, Cambridge, MA, 1993.
6. Gelperin, D. On the Optimality of A*. AI, 8 (1977). 69-76.
7. Genesereth, M.R. and Nilson, N.J. Logical Foundations of Artificial Intelligence. Morgan Kaufman Publishers, Inc., Los Altos, 1987.
8. Harris, M., Coombe, G., Scheuermann, T. and Lastra, A., Physically-Based Visual Simulation on Graphics Hardware. in Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware, (Saarbrucken, Germany, 2002), Eurographics Association.
9. Jadbabaie, A., Lin, J. and Morse, A.S. Coordination of Groups of Mobile Autonomous Agents Using Nearest Neighbor Rules. IEEEE Transactions On Automatic Control, 48 (2003). 988-1001.
10. Johnson, S. Emergence: The Connected Lives of Ants, Brains, Cities, and Software. Touchstone, New York, 2002.
11. Jones, C. End-user programming. IEEE Computer, 28 (1995). 68-70.
12. Koenig, S., Likhachev, M., Liu, Y. and Furcy, D. Incremental Heuristic Search in Artificial Intelligence. AI Magazine (2004).
13. Lieberman, H. Your Wish Is My Command: Programming by Example. Morgan Kaufmann Publishers, San Francisco, CA, 2001.
14. Liu, J., Tang, Y.Y. and Cao, Y.C. An Evolutionary Autonomous Agents Approach to Image Feature Extraction IEEE Transactions on Evolutionary Computation, 1 (1997). 141-158.
15. Maes, P. Designing Autonomous Agents. MIT Press, Cambridge, MA, 1990.
16. Maslow, A.H. A Theory of Human Motivation. Psychological Review (1943). 370-396.
17. Nardi, B. A Small Matter of Programming. MIT Press, Cambridge, MA, 1993.
18. Nareyek, A., Intelligent Agents for Computer Games. in Second International Conference on Computers and Games, (Hamamatsu, Japan, 2000), Springer-Verlag, London, UK.
19. Papert, S. The Children's Machine. Basic Books, New York, 1993.
20. Paternò, F. D1.2 Research Agenda: End-User Development: Empowering people to flexibly employ advanced information and communication technology, EUD-Net: End-User Development Network of Excellence, 2003, 17.
21. Patterson, D.A. Computer Science Education in the 21st Century. Communications of the ACM, 49 (2006). 27-30.
22. Repenning, A. Repräsentation von graphischen Objekten, Asea Brown Boveri Research Center, Artificial Intelligence group, Daetwill 5405, Switzerland, 1987.
23. Repenning, A. and Ioannidou, A. Agent-Based End-User Development. Communications of the ACM, 47 (2004). 43-46.

24. Schaeffer, J. A Gamut of Games. AI Magazine, 22 (2001). 29-46.

25. Searle, J.R. Minds, brains, and programs. Behavioral and Brain Sciences, 3 (1980). 417-457.

26. Simon, H.A. The Sciences of the Artificial. MIT Press, Cambridge, MA, 1981.

27. Spears, W.M., Spears, D.F., J.C. Hamann and Heil, R. Distributed, Physics-Based Control of Swarms of Vehicles. Autonomous Robots, 17 (2004). 137–162.

28. Toffoli, T. and Margolus, N. Cellular Automata Machines. The MIT Press, Cambridge, MA, 1987.

29. Tsui, K.C. and Liu, J., Multiagent Diffusion and Distributed Optimization. in Proceedings of the second international joint conference on Autonomous agents and multiagent systems, (Melbourne, Australia, 2003), ACM Press, New York, NY, USA, 169 - 176.

30. Turing, A.M. The Chemical Basis Of Morphogenesis. Philosophical Transactions of the Royal Society of London, 237 (1952). 37-72.

31. Watt, S., Syntonicity and the Psychology of Programming. in Proceedings of the Tenth Annual Meeting of the Psychology of Programming Interest Group, (Milton Keenes, UK, 1998), Knowledge Media Instritute, 75-86.

32. Wright, W. The Sims, http://thesims.ea.com/, 2001.