Locally-Adaptive Virtual Environments in Persistent-state Multi-player Games

Marc Lanctot Clark Verbrugge
School of Computer Science
McGill University
Montréal, Canada, H3A 2A7
phone: 1-514-398-2411
fax: 1-514-398-3883
email: `marc.lanctot@mail.mcgill.ca, clump@cs.mcgill.ca`

September 15, 2004

Abstract

We present a generic model for adaptation in large scale, persistent state computer games that allows the virtual world to change automatically, with reasonable efficiency. We demonstrate the utility of our technique through two different forms of dynamic common game content: 1) a basic weather cycle that adapts wind, rain and water accumulation to variations and changes in a large-scale terrain, and 2) a simple *reputation* system that allows agents in the virtual world to respond appropriately to a player's actual behaviour in a game.

1 Introduction

A basic problem encountered by vendors of large scale, persistent-state gaming environments is how to continuously improve and change the virtual environment so as to maintain player interest, and also reflect the activities of players in the virtual world. In a more generic sense this falls under *content creation* [Mellon, 2003], altering or adding new virtual content to the game. Manual approaches are typically used due to the creative requirements of general content creation and the complexity of determining realistic adaptation results, but impose extra game maintenance costs and administration requirements. Automatic approaches that sensibly alter and tune the game world with minimal human intervention are thus desireable.

We present a generic model for adaptation in computer games that allows the virtual world to change automatically, with reasonable efficiency. We demonstrate the utility of our technique through two different forms of dynamic common game content: 1) a basic weather cycle that adapts wind, rain and water accumulation to variations and changes in a large-scale terrain, and 2) a simple *reputation* system that allows agents in the virtual world to respond appropriately to a player's actual behaviour in a game.

Specific contributions of this work include:

- Design of a general adaptation framework suitable for modelling flow-based properties in game simulations. Our approach is based on cellular automata, ensuring only local information is required at each computation; this allows for reasonable scalability in distributed environments.
- Design and experimental verification of systems for two forms of popular, dynamic game content. We describe a simple, aesthetic and logically consistent adaptive weather model for game worlds, and a game reputation system that can dynamically respond to changing patterns of information dispersal and player behaviour.

1.1 Related Work

Virtual environments are of course well studied, and a variety of approaches already exist, though most efficiency and representation improvements concentrate on network and communication optimization [Macedonia et al., 1994] rather than adaptation of the environment itself. Even environments such as Bamboo that allow flexible modification to the environment through hot-pluggable modules [Watsen and Zyda, 1998] do not allow for incremental adaptation of game content.

Adaptation is a traditional target of A.I. research. Here a *virtual environment* is often separated into 2 major components: the static context, and the dynamic agents [Russell and Norvig, 2002]. In the context of

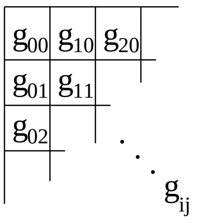


Figure 1: The virtual terrain.

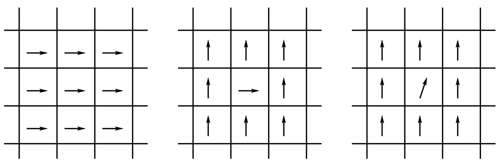


Figure 2: Affect of an update on one grid section, showing a) before the change b) before the update on the middle grid section c) after the change and update

computer games, adaptation has been investigated [Spronk et al., 2003], though like most other applications of A.I. [Carmel and Markovitch, 1998] it has been primarily directed at adapting agents (NPCs, game opponents) rather than the environment. Even non-constant, fluctuating environments are usually viewed as the process to react to, rather than the target of adaptation [Haynes and Wainwright, 1995]. Our motivations more closely resemble building an *artificial model* as in done in ALife [Steels, 1994]; we, however, focus on constructing an adaptive environment irrespective of adaptivity of the agents.

2 An Abstract Model for Adaptation

Our model is based on a finite 2-dimensional space, the virtual terrain. We assume the space is partitioned into a discrete mapping or *grid*. In the examples below we use the familiar situation of a subset of $\Re^2$ and a square grid G, as illustrated in Figure 1. In general the techniques we use apply equally well to any discrete *metric space* [Burago et al., 2001].

One can imagine that the elements of the grid, or *grid sections* have *properties* depending on the context of the system in which the model is used. The adaptation process aims to modify these properties based on the impact of events that occur in the surrounding sections. The procedure used is similar to the procedures associated with cellular automata: iterative update rules are applied based on properties of neighbouring cells [Gardner, 1970].

A simple example of an application of local property updates is *blurring* or *spatial low-pass/box filtering* in the field of image processing [Baxes, 1994]. Each pixel $p_{x,y}$ in an image has a scalar *intensity* property, $I(p_{x,y})$, and a neighbourhood of nearby pixels $N(p_{x,y})$. To create a blurred image, a new intensity for each point is defined:

$$p'_{x,y} = \frac{I(p_{x,y}) + \sum_{p \in N(p_{x,y})} I(p)}{|N(p_{x,y})| + 1}$$

and a simultaneous update rule is applied: $\forall x, y : p_{x,y} \leftarrow p'_{x,y}$. A good demonstration of the effects of this can be found at [Fisher et al., 2003].

2.1 Vector Averaging and Angular Propagation

Grid properties in cellular automata are typically scalar properties. SimCity, for example, is a classic game that relies on cellular automata techniques [Stanford, 1996], associating scalar quantities with grid cells. In SimCity each grid cell may have quantities such as pollution levels, crime rates, and land value, and so on. There is, however, no reason to restrict grid properties to scalar values. We define a *discrete vector field* as $V_G : G \rightarrow \Re^2$, so that for each grid section $g \in G$, there exists an associated vector, $V_G(g)$. The vector at cell $(3, 2)$ will be denoted $\vec{g}_{3,2}$. Note that a 2-dimensional vector can be thought of as a magnitude and angle; when we are interested in just one component of the vector we can reduce it to the scalar case; e.g., a simple angle value $\theta_{3,2}$.

Our technique is analogous to image blurring on vector components. We initially ignore the vector's magnitude and assume it does not change. Each grid section's vector is then modified to have a new angle computed as a weighted average of its own state and neighbouring angles. Suppose we have an average angle $\overline{\theta_g}$ for a grid section g and its neighbourhood. We define a *shift from* $\vec{g'}$ for each neighbour g' as the difference $\overline{\theta_g} - \theta_{g'}$. The total angular change is then some proportion of this shift, for example $\delta_g = \gamma \cdot \textit{shift from}(\vec{g'}, ...)$. The update rule then becomes: $\forall g \in G : \theta_g \leftarrow \theta_g + \delta_g$, applied simultaneously over all grid sections.

To demonstrate this, consider a single grid section surrounded by its *8-neighbourhood*, all of its vectors pointing eastward ($\theta = 0$) with arbitrary magnitude, as seen in Figure 2. Now, if we shift each surrounding vector by 90°, the average will shift by $\Delta\theta = (8/9) * 90°$, so the update will shift the middle vector's angle by $\delta = \gamma\Delta\theta$. Since the middle vector has shifted, upon the next application of the update (the next iteration) it will in turn cause a difference in average of all points for which it is a neighbour. This will cause those grid sections' vectors to update, and so on. As a result, a change in angle propagates through the grid via its neighbouring cells, but loses influence each iteration. By adjusting weights such as γ local turbulence can be damped according to the needs of the system being modelled.

2.2 Flow-based Fuzzy Property Update Rules

Non-constant scalar properties on grid sections are modified differently than the vector properties. The vectors on each grid section describe a strength and direction of *flow*. Here, the flow function is defined over a scalar property and computes how much of the property is transferred from a grid cell to the cells in its neighbourhood as a result of the vector property.

We use a *fuzzy* approach [McCuskey, 2000] to computing flow for a more natural flow dispersal. Formally, the flow function consists of n fuzzy components: $z_1, z_2, \cdots, z_n$. Here, z_j is an arbitrary fuzzy membership function $z_x(\vec{g}) \in [0,1]$ which represents the raw influence of that component over a given property. The influences of the components are then scaled so that they represent the local influence in comparison to other influences:

$$f_x(\vec{g}_{i,j}, p_{i,j}) = \frac{z_x(\vec{g}_{i,j}, p_{i,j})}{\sum_{y=1}^{n} z_y(\vec{g}_{i,j}, p_{i,j})}$$

To make this more clear, consider a scenario where the components are associated with the four major cardinal directions: z_N, z_E, z_S, z_W. The amount transfered in each direction is proportional to the corresponding flow influence value f_{dir}. At each iteration, $\Delta p_W = k_p * f_W(\vec{g}_{i,j}, p_{i,j}) * p_{i,j}$ is the amount of $p_{i,j}$ that's displaced westwards, where $0 < k_p <= 1$ is the *rate of transfer*. The simultaneous update rules for this component would then be: $R_1 : p_{i-1,j} \leftarrow p_{i-1,j} + \Delta p_W$ and $R_2 : p_{i,} \leftarrow p_{i,j} - \Delta p_W$. Components for other directions are treated similarly. Note that it is also possible to define hybrid components, formed by the conjunction or disjunction of the fuzzy properties; e.g., $z_{NW} = z_N$ AND z_W. Then the displacement of moisture would be listed as a rule set in a fuzzy controller system as is done in [McCuskey, 2000].

The actual behaviour of the flow depends on the membership functions used; if a system demands a smooth flow, then naturally the membership functions should reflect that. The role of the fuzzy membership functions are to shape the flow. If, for instance we use a "crisp" function, one with a sharply-defined peak such as:

$$z_W = \begin{cases} 1 & \text{if } \pi/2 - \epsilon <= \theta <= \pi/2 + \epsilon; \\ 0 & \text{otherwise.} \end{cases}$$

for small ϵ, then the westward flow will move somewhat discretely. A smoother function like:

$$z_W = \frac{4}{\pi}\sqrt{(\frac{\pi}{4})^2 - (x - \frac{\pi}{2})^2}$$

will lead to a smoother spreading. This will become more clear in the example systems that follow.

3 Example Systems

Here we describe two systems for adapting game content based on our general model. These provide experimental evidence of the generality of our approach, and are also novel forms of content generation in themselves. Experimental results for these systems are described in the following section.

3.1 An Adaptive Weather System

Weather simulation is typically considered a computationally intensive application, largely reserved for supercomputers. In the virtual worlds of computer games, however, physical accuracy is less critical, and much simpler approaches suffice to produce aesthetic, in-game climate effects.

In its simplest form, a weather cycle displaces moisture: water from lakes and seas is carried by wind to cooler locations, where the reduced water capacity of cooler air causes condensation; rain water eventually runs downhill to refill lakes and oceans [Enterprise, 2004]. There are several factors that can affect this process, including altitude and terrain structure, wind, temperature, and so on. We have based our weather system upon the following basic precepts:

1. Wind gathers moisture from bodies of water, and loses water at higher altitudes (cooler temperatures).

2. Water flows downstream.

3. Altitude affects wind patterns.

These basic rules will be transformed into update rules following the model outlined in the previous section. We must however first define another vector property used to describe how water flows downstream.

The *gradient vector* on a grid section is a vector sum composed of differences in scalar properties of surrounding cells. The magnitudes of the vectors are determined by subtracting the terrain *altitude* from the altitude of a neighbouring cell, with corners of the 8-neighbourhood having a weight factor or $\sqrt{2}/2$. The direction of each vector in the sum is given by

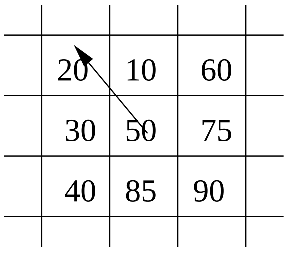


Figure 3: Example gradient vector calculation. Grid cells show local terrain altitudes.

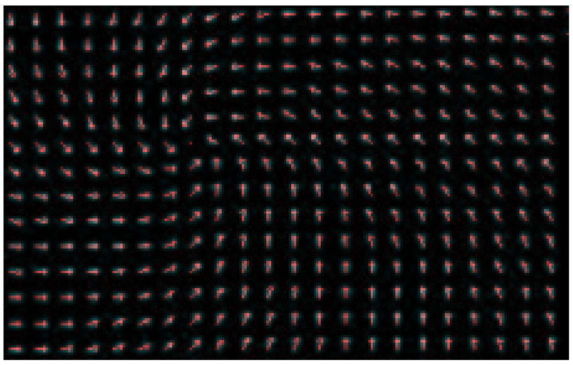


Figure 4: An example tornado.

the position of the neighbour relative to the center. Figure 3 shows an example gradient calculation: $g = 108.64\hat{W} + 120.35\hat{N}$ giving a vector with angle $tan^{-1}(120.35/108.64) = 48°$ north of west.

The gradient vector points to the direction of descent, and its magnitude represents the steepness of the grade. Moisture then flows downstream in the direction of the gradient as described in Section 2.2.

The inverse gradient points in the direction of ascent, and is used to determine how wind direction is altered by the current terrain. If a gust of wind is pointing into a wall, it will instead blow around it. For wind to move around high-altitude obstacles it must therefore be pushed away from the direction of the inverse gradient. We also incorporate an inertial factor, to give a smoother flow pattern; we designate the average of the gradient vector and current average wind vector as the vector that will be approached as described in Section 2.1.

Moisture is displaced by the four independent components that comprise the wind, represented by the cardinal directions: $\vec{g}_N, \vec{g}_E, \vec{g}_S, \vec{g}_W$. The value of each component is calculated by a fuzzy membership function. These values are then normalized, and represent a proportion of the amount of moisture displaced to surrounding grid sections, again as per the method in Section 2.2.

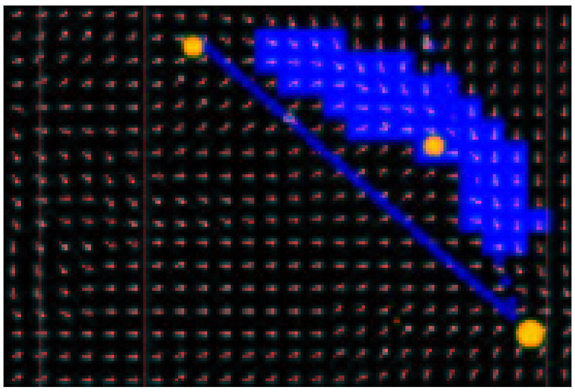
Interesting weather *events* are also possible. Events can be anything that affects the properties in the system such as earthquakes, tornadoes, tsunamis, etc. We have modeled "tornadoes" as local, non-linear dynamical systems with the stable fixed points at the center. We designed a 2-dimensional dynamical system within a specified sub-grid such that a given point is fixed point in a stable spiral [Strogatz, 2001]. The tornado moves by slightly displacing the fixed point at each iteration and reconstructing the dynamical system around it. The wind in the surrounding cells then adapts to this turbulence using the vector propogation rules found in Section 2.1. Figure 4 shows a static screenshot of a tornado on a flat terrain.

3.2 An Adaptive Reputation System

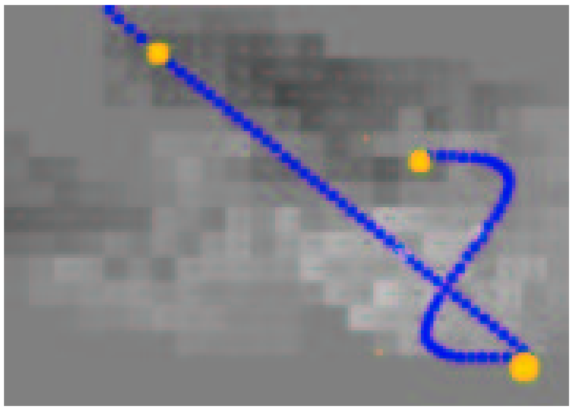
A player's in-game reputation is often an important component of the game environment, particularly for simulation and role playing games. Player actions that harm or help computer-controlled characters should result in a logically consistent reaction to the player, giving a sense of reality to the game environment. This is necessarily a dynamic property—player reputations need to be constantly updated, and should also ameliorate over time and distance.

There have been some commercial attempts at creating a general, flexible reputation system, but results have been disappointing [Brockington, 2003]. Our approach was inspired by the *Dungeons & Dragons* reputation system [Collins et al., 2004], which states that as a player progresses his or her reputation will rise by performing "heroic deeds." Symmetrically, of course there should also be the inverse property, to degrade reputation by performing negative actions.

In order to allow reputation to more realistically disperse, we employ a word-of-mouth model to flow reputation events. A game character's reputation is built by the spread of hearsay amongst the populace; flow vectors modelling the communication patterns of the general populace in each grid section are used to describe the direction in which word of a positive or negative action will spread. These vectors are bent to follow the "terrain" of potential communication, in the manner outlined in Section 2.1. For our example system we developed a virtual communication terrain using constraints provided by the virtual geographical terrain: paths (minimal routes) between cities and towns indicate trade, and hence communication routes with a communication "altitude" inversely proportional to value/popularity of the relative trade: where there is little trade altitude is high, representing impassability of information flow. The same wind model used in our weather system then traces out the flow of reputation information. Trade route popularity and connections



a)



b)

Figure 5: Screenshots of a Reputation test in action. The lines are the paths of the agents, the white triangles are the agents, and the circles are the cities which agents travel to and from. In a) the blue (lighter) smudges are the reputation points spreading. b) shows the communication terrain.

in a commercial game could be derived from the relative movements of other player characters. In our case we simulated route popularity by tracking movements of semi-randomized agents moving between cities following a heuristic, A* search algorithm to discover trade routes. Figure 5b shows an example of communication terrain constructed in this manner.

Positive and negative *reputation points* are created on a grid section when an event occurs that would affect someone's reputation: rescuing the princess, killing a commoner, stealing from tavern, *etc.,* and are proportional to the severity of the event. These points are displaced via the flow, and also dissipate at a slow rate. For each point that dissipates on a grid section, the reputation of the player is altered at that location. This process repeats until all the reputation points have dissipated, causing a local alteration in the player's reputation.

Test	Type (threads)	Map	GUI	System
1	Weather	N. Korea	yes	UP
2	Weather	Pakistan	yes	UP
3	Weather	Pakistan	no	UP
4	Weather	Pakistan	no	MP
5	Weather (2)	Pakistan	no	MP
6	Weather (4)	Pakistan	no	MP
7	Reputation	62 x 50	no	UP
8	Tests 3 and 7 combined			UP

Table 1: Testing scenarios.

4 Experiments and Analysis of Data

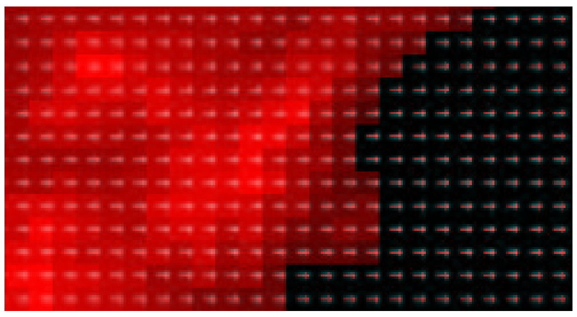
We conducted tests on both systems, to assess aesthetic quality, and also to quantitatively ensure the systems were stable and efficient; these are described in Table 1. The 3rd column describes the test data, either an altitude map of North Korea (48x40), or of Pakistan (62x50), obtained by downsizing more detailed maps from [Rojas et al., 2003]. The fourth column indicates whether the graphical user interface of our testing was present. Some tests were performed on a uniprocessor Pentium 4 1.70Ghz with 528M RAM (*UP* in column 5). Other tests designed to show scalability launch multiple computation threads, each assigned responsibility for a portion of the grid. Every iteration grid calculations are done independently and concurrently by these threads, and synchronized for the simultaneous update. These were performed on a quad-processor AMD Opteron 844 (1.80Ghz) with 3.6G RAM (*MP* in column 5). Every test lasted at most 10000 iterations.

The Weather tests are composed of tests 1–5. In each case all the wind vectors start pointing eastwards with a magnitude of 50, as shown in Figure 6a. This is an initial state far from any final state (Figure 6b), and so represents an extremal test for adaptation. We also included a tuning parameter, an *average bias value* $g = 0.9$. When calculating the vector to approach in the gradient-bending method of Section 2.1 this constant places bias on either the gradient ($0 <= g < 0.5$), or the current average ($0.5 < g <= 1$). The weighted average of these is the vector that is approached:

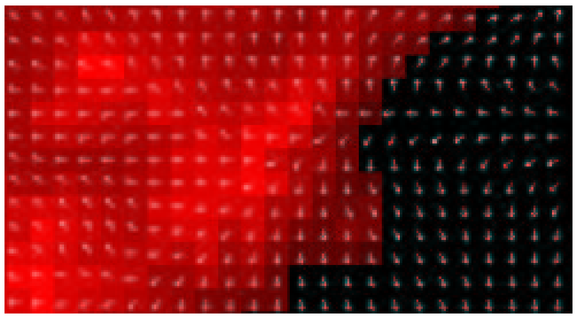
$$\vec{v}_{TARGET} = g \cdot \vec{v}_{AVG} + (1-g)\vec{v}_{GRAD}$$

The update rule then becomes a rule that approaches the target: $R_1 : \theta_{i,j} \leftarrow \theta_{i,j} + \gamma(\theta_{TARGET} - \theta_{i,j})$, as seen in Figure 7. This results in a smoother, if slower adaptation.

The Reputation tests were made up of simulations of agents moving from town to town, as described in Section 3.2. The probability that an agent would travel



a)



b)

Figure 6: Screenshots of a) before the Weather Test 1 b) end of Weather Test 1.

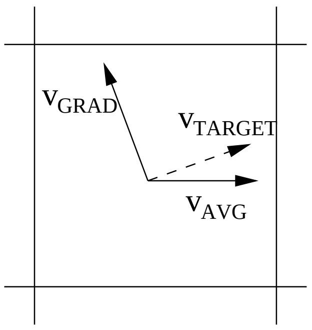


Figure 7: Example of obtaining $\vec{v}_{TARGET}$ when $g = 0.8$.

T_n	$\bar{t}_i(ms)$	$\bar{t}_{i_1}(ms)$	$\bar{t}_{i_2}(ms)$	$\bar{t}_{i_3}(ms)$
1	43.91	29.81	9.6	4.5
2	92.97	64.37	21.5	7.1
3	64.22	44.68	14.34	5.2
4	46.32	28.61	11.61	6.1
5	28.85	16.41	8.3	4.14
6	23.41	12.56	7.5	3.35
7	44.68	34.52	10.12	0.02
8	104.5	74.4	24.74	5.35

Table 2: Performance results for the tests.

from one town to another at any given iteration was set to 0.9. The source city was chosen randomly, weighted by the size of the city (larger cities produce more trade, and hence more communication sources), and the destination city was chosen based on a combined weight determined by city size divided by the square of relative distance (short trips are more common). Reputation events would normally come from player actions; in our tests they are manually generated during testing.

4.1 Performance Results

Experimental results are summarized in Table 2. In all cases, $\bar{t}_i$ represents the average simulation time per iteration of calculations. In the Weather tests $\bar{t}_{i_1}$, $\bar{t}_{i_2}$, $\bar{t}_{i_3}$ represent the average time spent on the gradient-bending and averaging, moisture spreading, and rain displacement, respectively. In the Reputation tests, these times represents the vector-averaging, reputation point spreading, and the vector-bending due to agents' influence, respectively. In the combined test, these times (except $\bar{t}_i$) are simply added together.

Results are encouraging. Tests 1 and 2 show a linear increase in average iteration cost with respect to grid size: the Pakistan map is 61% larger than the North Korea map, and average iteration time is 58% larger.

Scalability is shown in tests 4, 5, and 6. Average iteration cost drops by a factor of 1.98 using 4 threads for computation. Although this is far from ideal, these results still show that parallel computation is applicable here.

The value of casting these problems into a generic framework is evident in test 8 in comparison to 3 and 7. By combining the tests we allow not only a shared implementation, but also sharing of resources; this costs somewhat less than doing both tests separately, even though the tests do not use clever collaborative methods. We expect future additions to behave similarly.

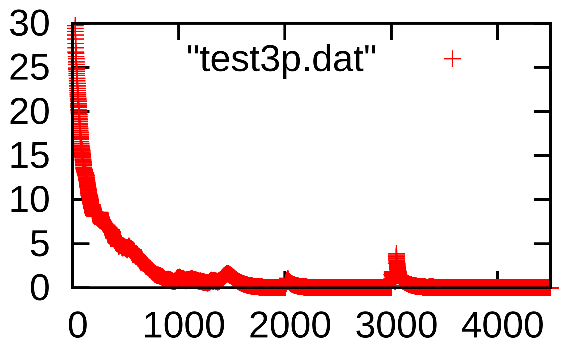


Figure 8: Δ_{mask}(radians) vs. iteration number in Weather test 3.

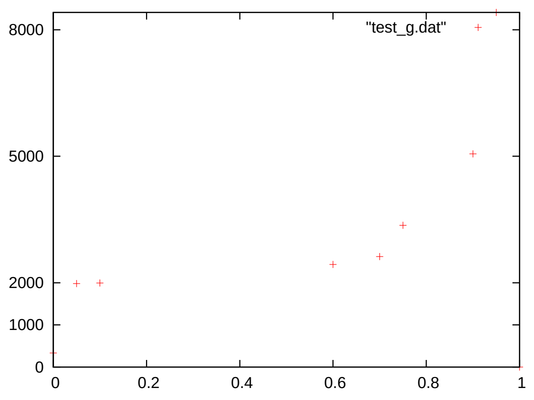


Figure 9: Iterations until convergence vs. g graph, with $\gamma = 0.1$.

4.2 Convergence Properties

Since our approach uses an iterative update strategy it is important to know how long it may take for a calculation to converge to appealing and stable results. Convergence can be guaranteed in the finite domain of computer-representable real numbers by ensuring our flow and gradient calculations are *monotonic* functions. This is unfortunately not easy to ensure given the use of complex, stateful (history-sensitive) functions in game simulation. We thus investigate convergence experimentally for our example systems.

Convergence in the Reputation test is not meaningful. Agent behaviour is deliberately non-deterministic in order to produce a complex communication terrain, and actual flow of reputation points is damped, and so trivially converges from any constant number of reputation events.

In the Weather case, we are trying to adapt a quantity (wind) within the system to its surroundings. To measure how this quantity is changing we add up all the proposed change in angles over all grid cells at every iteration and call this the Δ_{mask}; the results for test 3 are shown in Figure 8. In most cases it takes less than 5000 iterations to converge to the point where Δ_{mask} is consistently extremely small ($< 10^{-12}$). Less converged, but quite acceptable results are however achieved much sooner, typically by within 1000 iterations. In Figure 8 random perturbations to terrain altitude are performed on the grid at iterations 2000 and 3000; these small adaptations converge very quickly, in just a couple hundred iterations.

The bias parameter has a direct influence on the time it takes to converge. Figure 9 shows the number of iterations before convergence (variation less than 10^{-12}) for a selection of g values. Convergence in general behaves non-linearly. This is not surprising given the complexity of our system, but it does indicate the importance of experimental validation in the implementation. Note that our examination of slow and non-convergence shows that when it occurs it is typically expressed visually as continuous change localized to just one or two very small areas; the appearance is still overall quite good, and in a continuously adapting situation is difficult to discern.

5 Conclusion and Future Work

Environmental adaptation in persistent-state games is a relatively unresearched area. We have presented a general framework to describe such systems that is generic and efficient. We have shown that our approach is suitable for adaptive content management and creation. In order to do so we defined implementations within our framework for two very different forms of content, basic weather simulation, and an in-game player reputation system. Both systems respond to dynamic changes in input data, and we have experimentally demonstrated the feasibility of the technique, both in terms of performance and the ability to converge to good, aesthetic results.

Our approach is of course largely a poof of concept demonstration. The practical value of our technique would be best measured in the context of a commercial, large scale, multiplayer game, though research access to such an environment is difficult to acquire. Even in the context of a prototype, however, performance improvements can be realized; our implementation was designed for easy exploration of ideas more than performance, and a number of optimizations are possible. We've proved that the framework is scalable in two different ways: computationally by using concurrency, and content-wise by efficient re-usage of resources for combined systems. An expansion of useful, flow-based content is also of value, and we are investigating the expression of such concepts as adaptive economies, politics, and law in our system.

6 Acknowledgments

This research has been supported by Le Fonds Québécois de la Recherche sur la Nature et les Technologies and the McGill Faculty of Graduate Studies.

References

[Baxes, 1994] Baxes, G. A. (1994). *Digital Image Processing.* John Wiley & Sons.

[Brockington, 2003] Brockington, M. (2003). Building a reputation system: Hatred, forgiveness, and surrender in Neverwinter Nights. In Alexander, T., editor, *Massively Multiplayer Game Development*, pages 454–563. Charles River Media.

[Burago et al., 2001] Burago, D., Burago, Y., and Ivanov, S. (2001). *A Course in Metric Geometry.* American Mathematical Society.

[Carmel and Markovitch, 1998] Carmel, D. and Markovitch, S. (1998). Model-based learning of interaction strategies in multiagent systems. *Journal of Experimental and Theoretical Artificial Intelligence*, 10(3):309–332.

[Collins et al., 2004] Collins, A., Decker, J., Noonan, D., and Redman, R. (2004). *Unearthed Arcana.* Wizards of the Coast.

[Enterprise, 2004] Enterprise, N. E. S. (2004). GSFC earth science enterprise water and energy cycle. `http://gwec.gsfc.nasa.gov`.

[Fisher et al., 2003] Fisher, R., Perkins, S., Walker, A., and Wolfart, E. (2003). Mean filter. `http://homepages.inf.ed.ac.uk/rbf/HIPR2/mean.htm`.

[Gardner, 1970] Gardner, M. (1970). The fantastic combinations of John Conway's new solitaire game "life". *Scientific American*, pages 120–123.

[Haynes and Wainwright, 1995] Haynes, T. D. and Wainwright, R. L. (1995). A simulation of adaptive agents in hostile environment. In George, K. M., Carroll, J. H., Deaton, E., Oppenheim, D., and Hightower, J., editors, *Proceedings of the 1995 ACM Symposium on Applied Computing*, pages 318–323, Nashville, USA. ACM Press.

[Macedonia et al., 1994] Macedonia, M. R., Zyda, M. J., Pratt, D. R., Barham, P. T., and Zeswitz, S. (1994). NPSNET: A network software architecture for large-scale virtual environment. *Presence*, 3(4):265–287.

[McCuskey, 2000] McCuskey, M. (2000). Fuzzy logic for video games. In DeLoura, M., editor, *Game Programming Gems*, pages 319–329. Charles River Media.

[Mellon, 2003] Mellon, L. (2003). Research opportunities in game development. Tutorial at: PADS'03 Workshop on Parallel and Distributed Simulation.

[Rojas et al., 2003] Rojas, E., Hijmans, R. J., and Guarino, L. (2003). DIVA-GIS. `http://diva.riu.cip.cgiar.org`.

[Russell and Norvig, 2002] Russell, S. and Norvig, P. (2002). *Artificial Intelligence.* Prentice-Hall, 2nd edition.

[Spronk et al., 2003] Spronk, P., Sprinkhuizen-Kuyper, I., and Postma, E. (2003). Online adaptation of game opponent AI in simulation and in practice. In *Proceedings of the 4th International Conference on Intelligent Games and Simulation (GAME-ON 2003)*, pages 93–100.

[Stanford, 1996] Stanford, M. I. (1996). Uses and subversions of SimCity 2000.

[Steels, 1994] Steels, L. (1994). The artificial life roots of artificial intelligence.

[Strogatz, 2001] Strogatz, S. H. (2001). *Nonlinear Dynamics and Chaos: With Applications to Physics, Biology, Chemistry and Engineering.* Perseus Books Group.

[Watsen and Zyda, 1998] Watsen, K. and Zyda, M. (1998). Bamboo: A portable system for dynamically extensible, real-time, networked, virtual environments. In *IEEE Virtual Reality Annual International Symposium (VRAIS'98)*, pages 252–259, Atlanta, Georgia.